report_notebook

December 30, 2024

0.0.1 ARTIFICIAL INTELLIGENCE FUNDAMENTALS PROJECT 2024/2025 - Charisardo Team

Github repository

1. Introduction Our project focused on the design and implementation of several agent algorithms capable of engaging in Pokémon battles with randomly selected teams, determining the best move from those available. The task was part of the battle track of the VGC AI Competition 2024. We simulated the battles using the VGC AI Framework, which we modified to introduce non-determinism and to add additional features to the Pokémon. We developed four strategies using two approaches: * Greedy search: *Heuristical*, *WeightedGreedy* and *WeightedGreedy2* * Minimax search with alpha-beta pruning: *PrunedTreeSearch*

2. Related works To build the Pokémon teams and simulate the battles we used VGC AI framework, an environment designed to emulate the scenario of human video game championships of Pokémon with AI agents, including the game balance aspect. We used the policies already implemented in the framework as a benchmark to evaluate our algorithms. More information about the VCG AI framework are avaiable in the related paper. We also compared our strategies to the two best competitors in the 2024 competition, *Punisher*, implemented by Daniel Ladwig and Marlon Hörner, and *hayo5_BattlePolicy*, created by Jo Ha Young. To design and implement some of our algorithms we referred to the textbook *Artificial Intelligence: a Modern Approach*, by Russell and Norvig, in particular chapters 3.5, 5.2 and 5.3.

3. Environment and task Pokémon battles are conducted between two players, each controlling a team of three Pokémon, of which one is active and the others are party. Each Pokémon can perform four specific moves, characterized by type, maximum power, accuracy, and maximum PP.

At each turn, players select one of the four moves of their active Pokémon or switch it with one of the party. Switching, the player loses the opportunity to perform a move during that turn. At each turn it will be randomly selected who plays the first move. Players are aware of the possible moves available to the opponent's Pokémon but do not know which move the opponent will choose in the current turn, as decisions are made simultaneously. When a Pokémon's HP reaches zero, it faints, forcing the player to switch to another Pokémon. The match continues until all Pokémon on one team have fainted, at which point the opposing player is declared the winner.

Our goal is to develop a gameplay strategy that allows us to compete with a randomly generated team against any opponent, spending a reasonable amount of time.

3.1 Team generation To generate teams, we implemented the function *OwnTeamGenerator()*, which randomly selects the characteristics of the moves for the Pokémon in the team. This function also enabled us to vary the accuracy, i.e., the success rate of moves, which in the original framework was always set to 1. By introducing this non-deterministic element, we increased the complexity of the game.

4. Methodologies spiegate algoritmi

4.1 Heuristical

4.2 PrunedTreeSearch

4.3 WeightedGreedy 1/2 We tested a greedy algorithm that selects the move with the highest expected value for the damage. Calculated by simulating the inflicted damage for each move and multiplying by the accuracy. If by switching pokemon the expected value is higher than a factor (i. e. 2), the algorithm switches the pokemon.

The second greedy algorithm is similar to the first one, but it also considers the opponent's best greedy damage. And takes the best move that maximizes the difference between the two.

Surprisingly these two simple algorithms performed better overall as shown in the results section.

5. Assessment We evaluated the performance of our algorithms comparing them to each other, to the two best policies of 2024 competition (*'Punisher'*, the winner, and *'hayo5_BattlePolicy'*, second place) and to six already implemented policies: *'RandomPlayer'*, *'TypeSelector'*, *'BreadthFirstSearch'*, *'OneTurnLookahead'*, *'PrunedBFS'* and *'TunedTreeTraversal'*. **specificare cosa fanno?**

We simulated 600 battles for each combination of policies changing each time the composition of the two teams. We used win rate, number of turns and required time as evaluation metrics. To balance the assessment, for every battle between two player we do 2 battles with the same team composition and switch the player that starts first.

5.1 Win rate For each combination of policies, we calculated the rate of won matches over the total. Results are shown in **Figure 1**, where our algorithms are highlighted with two asterisks and the competition winners are indicated by one asterisk. The strategies are ordered by the total win rate, which is reported in the last column.

We can see that our strategies have a good performance against the random player, in particular the greedy approaches have a win rate higher than 0.85, and they are the 3 best policies between the 12 proposed. *PrunedTreeSearch* has a similar win rate, while *Heuristical* perform slightly worse, but it still have a good result, much higher than 0.50.

With respect to the total win rate, we can see that our weighted greedy approaches have again the highest results, followed by *OneTurnLookAhead* and *Punisher*, two others greedy strategies. The success of the greedy approaches is probably due to the randomness of the game: it's unlikely that randomly selected Pokémons can combine their moves to obtain a particularly effective strategy, so maximizing the utility of next move is often the best choice, and the key is to define a good heuristic function to represent utility. Between the tree search approaches our agent perform the best together with *BreadthFirstSearch*.

Anyway, considering the total win rate, no policy have clearly better results (all results are below 0.60), but our best agents are quite good against the weakest benchmark agents (*RandomPlayer*, *TunedTreeTraversal* and *PrunedBFS*).

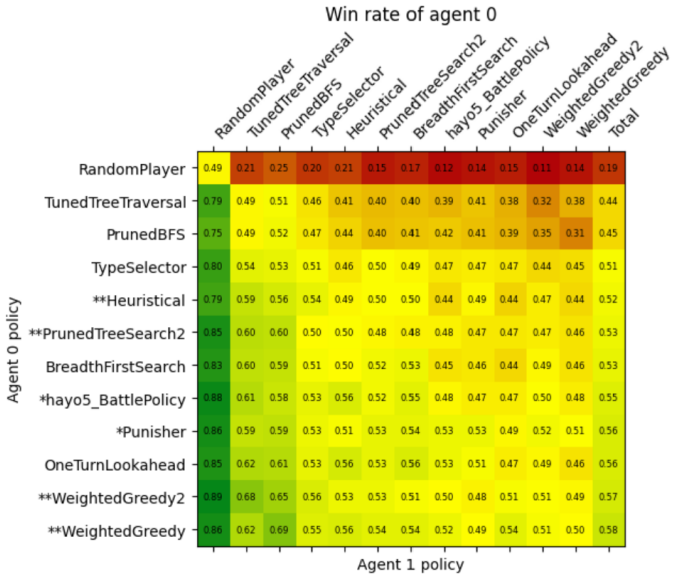

Figure 1: Win rate for all combination of policies and total value for each policy

5.2 Number of turns To evaluate the efficacy of each strategy we also observed the mean number of turns required to end a battle, shown in **Figure 2**. We can see that for most strategies the value is between 6 and 7, with very small differences for policies from *BredthFirstSearch* to *WeightedGreedy*. *Heuristical* has the highest value excluding *RandomPlayer*, but it's still under 7.

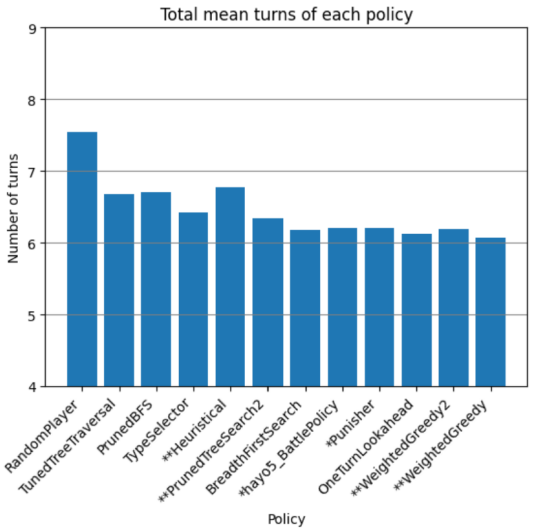

Figure 2: Mean number of turn required to end a battle for each policy

vorrei metterci std ma forse inutile

5.3 Time Finally, since the performances of the policies were similar in terms of win rate and number of turns, we evaluated the average time of each strategy to choose the move. The results

are presented in **Figure 3**. As clearly evident in **Figure 3a)**, our policy *PrunedTreeSearch2* is much slower than the others, making it unusable, expecially considering its non outstanding results in terms of win rate and number of turns. In **Figure 3b)** there is a focus on the range [0 - 0.7] s, and we can see that our best agents are still much slower than comparably approaches which perform slightly worse: *hayo5_BattlePolicy* has a mean time of 4.1e-05 s, *Punisher* of 1.1e-05 and *OneTurnLookahead* of 1.6e-05, while *WeightedGreedy* and *WeightedGreedy2* are 3.0e-02 and 6.1e-02 respectevely. Heuristical is faster, with a time of 3.9e-03 s, but still slower than other strategies. Regarding our policies, we can observe that *WeightedGreedy2*, which is a modification of *WeightedGreedy*, requires a double time, without an improvement in battle outcome.

Probably the biggest weakness of our policies is the computational time, expecially compared to *OneTurnLookAhead*, *Punisher* and *hayo5_BattlePolicy*. This difference may be due to the complexity added to the game setting a random accuracy, which forced our agents to more considerations. Anyway, we think that the time required by our strategies (with exception of *PrunedTreeSearch*) is not too high for this task.

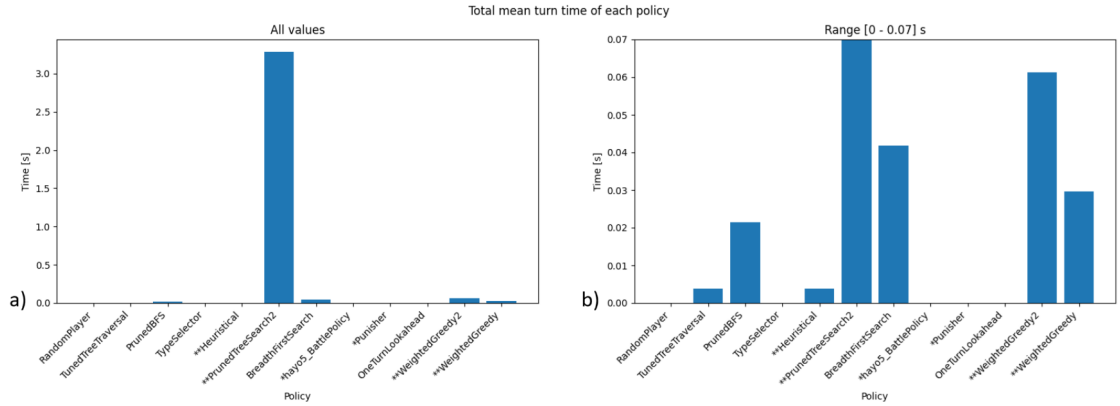

Figure 3: Mean time to perform a turn for each policy. **a)** All values range, **b)** Range [0 - 0.07] s

6. Conclusions We designed and implemented four algorithms, of which one performs a three search and the others use a greedy approach. We evaluated them combining measures about win rate, number of turns and time to perform a turn, comparing them to other eight strategies. From **Figure 1** is evident a distinction of win rate between tree search and greedy approach (with the exception of the weakest greedy approaches, *TypeSelector* and *Heuristical*), with the latter's advantantage. As anticipated in section 5., this is probably due to the task structure: with a randomly generated team it's unlikely to have a combination of moves and Pokémons particularly efficent, and the best move is often the one which maximize our utility in the current turn. Our *WeightedGreedy* and *WeightedGreedy2* have similar (and good) values regarding win rate and number of turns, but the first is much faster, making it the best policy. Anyway, we think that *WeightedGreedy2* is a good policy, and that the choice between the two could depend on the opponent type. For exemple, in **Figure 1** we can see that *WeightedGreedy2* is better against *TunedTreeTraversal*.

7 Appendix:

7.1 Team contribution:

- **Vito** modified the environment, added accuracy randomness
- **Su Qi** and **Salvatore** worked on implementation of *PrunedTreeSeach*
- **Su Qi** implemented *WeightedGreedy1-2*
- **Antonio** implemented *Heuristical*
- **Matilde** carried out initial analyses and final assessment

Everyone contibuted on idea development, bug fixing and report writing.

7.2 Relationship with the course