Reinforcement Learning for a Turn-Based Small Scale Attrition Game

Rodrigo Rill-García

Instituto Nacional de Astrofísica, Óptica y Electrónica, Santa María Tonanzintla Pue 72840, Mexico,
rodrigo.rill@inaoep.mx

Abstract. Turn-based RPGs involve sequential steps towards an specific goal: winning with limited resources. However, decision making in this kind of games implies a non-trivial problem even for human players when it comes to out-of-tutorial battles. As players are often required to play through many battles to develop a high-level intuition/reasoning on this kind of games, the present work uses a Reinforcement Learning approach to make an AI available to play one of those games in an analogue way to how novice human players would.

Keywords: Reinforcement Learning · Markov Decision Process · Q-Learning · Turn-based RPGs.

1 Introduction

1.1 Pokémon: a Brief Introduction to the Selected Game

Pokémon is a popular Japanese RPG (Role Playing Game) which stands a world championship every year, thus being an attractive option for IA techniques testing.

For the purpose of this work, a brief description of the game is given below:

- Each player has a 3-pokémon team, each with a certain remaining health from 0 (fainted or "dead") to 100% and 4 possible moves (each with a limited number of uses)
- The winner is decided by a last man standing criteria
- Only one pokémon per player can be at the battle ground at the same time
- Every turn, both players select simultaneously an action, either one of the 4 moves of their pokémon or changing for one of the other not fainted pokémon in their team
- From this point, the game takes full control to return the results of the actions taken by both players
- This turns-scheme is followed until one team is left with no non-fainted pokémon

As the game consists in a battle with limited resources until a winner is declared by draining out all the opponent's agents, this game is included in the category of attrition games according to the model originally defined by Maynard [1]. For further information, the community driven encyclopedia Bulbapedia [2] is a good option.

1.2 Why a Learning Approach

By 2010, Furtak and Buro [3] presented proofs on the complexity of two-player attrition games, concluding that for most games this is a hard computational task. Even for basic games of this kind, getting a deterministic winning strategy is a problem PSPACE-hard in EXPTIME. Therefore, this work approaches the solution to this problem by a Reinforcement Learning (RL) strategy instead of a searching strategy.

This achieves two advantages:

- As the main idea is learning *a priori* a policy for playing, the system is able to play in real time -just by doing a query every turn- avoiding an exhaustive search
- The learning is not limited to prior training, as the system is able to learn on-line by playing

This last point is a key-concept for this work, as that is exactly how human players learn to play and improve in the game (and, more generally, in strategy games). Also, this dynamism extends the lifetime of the system, as it allows it to change along the metagame (unlike other classical games, RPGs tend to add or modify game mechanics and agents over time, thus motivating players to create new strategies and to abandon some old ones).

To sum up, Pokémon can be summarized as an analyze state(turn) $\rightarrow$ take action sequence game, and the proposed solution is to learn the best action for every state by emulating the human learning process, which brings out the idea of RL.

However, a model for a Pokémon battle is not explicitly defined. Therefore, in order to learn over experience without needing an explicit model (as there is no real interest in modeling the game), the Q-Learning algorithm was selected, because its implementation is simple and it has proofs on convergence to an optimal policy [4].

2 Related Work

Proposed solutions for this kind of games (and more precisely Pokémon) resort to classical IA strategies for games, mainly search strategies. However, the branching factor of possible actions per turn invariably turns any decision tree in a huge search space, so refined search techniques are needed to show an intelligent behavior along a battle.

Under this scope, some members of the Pokémon community have made efforts into effective bots for competitive playing. One example is the Minimax-based system developed by the users vasumv and sharadmv.[1] Their proposal was based on minimizing the bot's losses with a Minimax strategy; to model the opposite player choices, they downloaded 11,000 battle logs of the top 500 users from the platform[2]. Other similar solution was developed by user rameshvarun. [3]. Unfortunately, those works are no longer supported by the platform in which they were tested.

Towards academic community proposals, Stanford students took a Reinforcement Learning approach by comparing different algorithms, listed here from lower to greatest performance: ϵ-greedy, Minimax, Expectimax and Expectimax w/ TD-lambda[4]. Their results ranked as an average-to-good player, being overcome by their human expert. Just as the previously mentioned works, this team used replays from top players for learning (20,000 battles from top 500 players).

In the IEEE Conference on Computational Intelligence and Games (CIG), Xu and Verbrugge [5] presented an alternative based in cost-benefit heuristics to take full advantage of abilities beyond attacking, improving then basic greedy strategies without exhaustive search. That work focus in those high strategy level moves (sleeping and healing) instead of the battle as a whole problem.

In the scope of this conference, the New York University Tandon School of Engineering organized a competition for Pokemon Showdown AI to be hosted on CIG in 2017. The paper presented by the organizers of the "Showdown AI Competition" [6] shows some preliminary results on IA strategies for the game such as Breadth-First Search, minimax, Q-Learning, One Turn Lookahead, Type Selector and Pruned BFS, playing one against another to guide participants on the ways to go. From this paper, a good list of the problems involved in a pokémon battle can be pointed out: branching factor, turn count and infinite looping, turn atomicity, categorical dimensions, stochasticity, hidden information, deception and simulation cost, among others.

Finally, the author of this work along Madrid, J.G. developed a Knowledge-Based Learning bot for this game, applying the author's expert knowledge to develop an IF-THEN rules system. As that work is the only one with working code available, it'll be used as base-line for evaluation[5].

[1] This project can be found at `https://github.com/vasumv/pokemon_ai`

[2] A further explanation of their system can be consulted here: `https://www.reddit.com/r/stunfisk/comments/3i4hww/pokemon_showdown_ai/`

[3] Their project is also allocated in Github: `https://github.com/rameshvarun/showdownbot`

[4] The report for their project can be found here: `https://web.stanford.edu/class/cs221/2017/restricted/p-final/kkhosla/final.pdf`

[5] The paper derived from that project can be found in `https://drive.google.com/file/d/1XqOUAkDY_5QD-6kZWd9dKVj8OFQZHkmw/view?usp=sharing`

3 Methodology and Development

As pointed out in the previous section, one of the problems to be solved out in a pokémon battle deals with stochasticity in a sequential decision-making process. If we also define rewards over the decision taken over each turn (state), a pokémon battle can be modeled with a Markov Decision Process (MDP). Furthermore, it can be described as a Markov Game [7], which is an extension of game theory to MDP-like environments (a pokémon battle is not exactly a MDP because two opposing agents are looking for their own goal while disrupting the other one at the same time).

Lets consider a MDP defined as a tuple $M =< S, A, \phi, R >$ [8], where S is the whole set of possible states, A the whole set of possible actions, ϕ a stochastic transition function from state s to state s' by taking action a, and R a reward function for taking an action a in an state s. The sets S and A can be defined according to the developer's criteria and knowledge about the game, but ϕ and R are more complex to define without an explicit model of the game.

To work around this problem, the reward r will be computed *a posteriori* by an heuristic evaluating the results of the turn. However, the stochastic state transition function ϕ is yet to be defined. But as the objective of this project is to learn a policy for every pair $< s, a >$, a Q-learning algorithm will be implemented to learn an optimal policy π. As a model free technique, there is no need to explicitly calculate ϕ; however, the learned policy is implicitly expressing this transition function, then taking decisions over uncertainty. At the end, the optimal policy will be described as "taking the able action with the maximum expected reward", according to the Q function that will be described later in this section.

3.1 Defining the Game Representation in States

This system was developed under the context of Pokémon's 6th generation[6], meaning a set of 721 different pokémon plus variant forms of some of those 721 characters. To reduce the dimensionality, and according to the idea that every competitive game of this kind has a metagame centralized over a subset of "competitive characters", 500 high-ranked battles[7] were analyzed to get a list of the pokémon used by top players.

After this analysis, a list of 301 different pokémon was acquired. Considering this list as the set P, the set S of the MDP was defined as:

$$S = \{ \ (x, y) \mid x, y \ \epsilon \ P \ \}$$

Basically, the states-space is defined as the cartesian product of set P with itself, thus having $|S| = 301^2$. For this representation, x represents the player's

[6] By the time this work was released, the franchise was already in its 7th generation, but 6th generation was chosen because of related work and the system that was chosen as base-line.

[7] Those battles were replays stored in `https://replay.pokemonshowdown.com/search/?format=battlespotsingles&rating`

pokémon, while y stands for the opponent's one; therefore, an state consists on reading which ones are the pokémon in the battle ground prior to deciding an action.

3.2 Defining the Set of Possible Actions

Similar to set P, a set M was defined by observing the attacks (moves) used by the players in the 500 analyzed battles. From 622 different options, the resulting cardinality of M was 300. With this sets, the set of possible actions A was defined as:

$$A = \{\ z \mid z \ \epsilon\ P \cup A \cup \{no\_action\}\}$$

Taking no action is not an option available for a player, but under certain game mechanics there are times when a pokémon is unable to move, thus resulting in "no_action" from its part. Therefore, $|A| = (301 + 300 + 1) = 602$.

3.3 Defining a Reward

As told before, by the lack of an explicit model of the game, the reward function was defined by an heuristic over the results of each turn. Assuming the system was in state s and took the decision a, $R(s, a)$ is calculated by summing up the values of every result reached in the next table:

Table 1. Possible outcomes and their rewards.

Criteria	Reward Over Result On Foe	Reward Over Result On Player
One Hit KO	+1100	-1100
KO	+1000	-1000
X% Damage	+(X * 10)	-(X * 10)
Unable to act	+1100	-1100

Multiple instances of the table can be reached in the same turn, and the system is rewarded whenever a negative effect is inflicted in the foe, and punished every time a negative effect is inflicted in the player.

3.4 Defining the Learning Algorithm

Recapitulating, a function $Q : S \times A \to \mathbb{R}$ is desired, and this should be learned by experience. For this purpose, lets call *episode* to a whole pokémon battle. Also, let's define Q as a $|S| \times |A|$ matrix, where Q_{ij} corresponds to the expected reward of taking action j while being in state i.

At the end of each episode, the battle is analyzed turn by turn (that is, state by state). At each state, the Q matrix is updated using the next formula:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha * (\ R(s_t, a_t) + \gamma * max_a Q(s_{t+1}, a)\ )$$

For this particular implementation, the learning rate α was instanced to 0.1. According to literature, $0 \leq \alpha \leq 1$ defines how much new information is taken into account to update the Q function; under this scope, a low value for α is meant to believe more in previous information than in incoming data. $\alpha = 0.1$ was chosen to avoid over fitting; that is, the algorithm gives low importance to cases that are not often reached, and accumulates belief in cases often seen.

γ, known as discount factor, was chosen to be 0.8. According to literature, this variable is also in the range [0,1], and defines how much importance is given to the expected reward in the future (talking about the maximum expected reward by taking the best action in the next state). To avoid getting stuck in a local maximum (that is, being totally greedy according to the information known in the present time), γ was chosen to have a great value. That is, $\gamma = 0.8$ to give great importance to the future development of the game.

3.5 Training

Once the learning algorithm was defined, the system was ready to be trained. This process was divided into 3 phases, as described below.

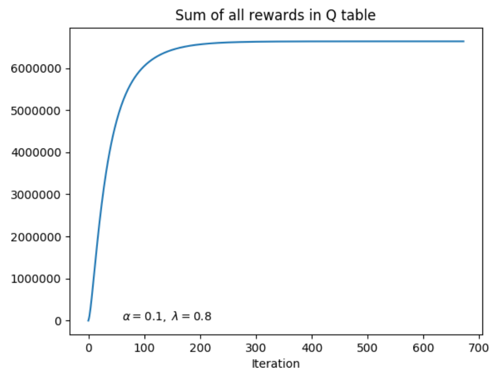

Fig. 1. Accumulated rewards learned from top players battles. The horizontal axis represents the number of episodes analyzed.

Learning from top players To give a first guide for training, the system was trained by mere observation instead of exploring. For this, the 500 battles mentioned before were used[8]. These battles were analyzed until the algorithm reached a convergence with respect to the total accumulated rewards in the Q matrix (as seen in Fig. 1).

Learning from itself In this phase, the system was made to play against itself in order to try to improve its Q function values by reinforcement. For this training, two random teams were chosen (one for each player). With these teams, the system played 10 battles versus itself; once those battles were over, new teams were chosen to play again.

Given the huge states-space (301^2), it was necessary to guide the learning process in order to reach convergence in an acceptable time, without neglecting the exploring capability of Q-Learning by choosing random actions. To achieve these, a threshold-greedy strategy was used, as described below:

[8] From these, 483 battles were useful. The rest were either corrupted or one player forfeited in the first turn

- Given the current state, if $max_{a_{able}}Q(s, a_{able}) > threshold$ (where a_{able} stands for all the available actions for the player in that moment), $argmax_{a_{able}}Q(s, a_{able})$ is chosen
- If no action was selected in the previous step, given s_t is the current state defined as $s_t(x_t, y_t)$, the set of similar states S_s is defined as $S_s = \{s|s \neq s_t, y = y_t\}$. For every a_{able}, lets define an imported reward function $R_i(a) = E[Q(s_s, a)], \forall Q(s_s, a) \neq 0$ (basically, the average reward expected for an action based on the expected rewards calculated before when such action was used against the same foe). If $max_{a_{able}}R_i(a_{able}) > threshold$, $argmax_{a_{able}}R_i(a_{able})$ is chosen.
- If none of the previous 2 options was taken, a random action a_{able} is chosen

In this way, the system is allowed to exploit promising actions while not getting stuck in a low rewards policy for training. One advantage of this type of training is that every episode counts as, actually, two different episodes (one from each side point of view).

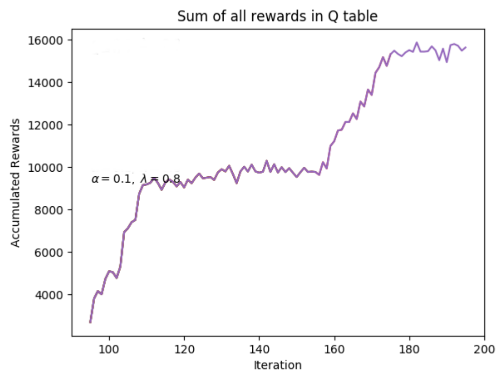

Fig. 2. Accumulated rewards learned from battling itself. While the winner increases its rewards, the loser decreases them; however, the trend is increasing the accumulated rewards.

Learning From Real-Time Battles Finally, the system was trained again by an observation method. The idea was to learn about possible actions not explored by the system while battling itself, and reinforcing the knowledge of actions already explored. Basically a way of tuning the knowledge.

4 Experiments and Results

The system was tested against an adaptive Knowledge-Based System. The implementation was using selenium webdriver for Python, in order to play on the web browser based Pokémon Showdown! Battle Simulator Beta[9]. For evaluation, experiments were done using 10 teams, performing 10 battles per team. One experiment was using the very same random team for both systems, while another one was using two different random teams.

Two special considerations must be pointed out: for fair comparison, both systems played under the restricted rules defined by the KBS. Also, both systems were able to learn along the 10 battles (while the KBS acquires data about the opponent's team to improve its decisions, the Q-Learning system learns rewards on the actions taken by both players).

[9] Pokémon Showdown home page: `https://pokemonshowdown.com/`

Besides, to test performance against human players, a brief experiment was done in a *N-attempts to win* scheme. For this experiment, 10 random teams were chosen for both the system and the player (teams were different for both participants). For each team, the system had 10 attempts to beat the player; if the system was able to win a battle before the 11th battle, it was considered a success and the experiment was over for that team. If the 10th battle was over and the system was not able to win a single battle, it was considered a failure and the experiment was over for that team.

4.1 Same team

For this experiment, the performance was just as expected, with the Q-learning system winning most of the battles for most of the teams. Over the 100 performed battles, the global victory ratio was 0.58 (58%). A more detailed perspective can be obtained from Fig. 3.

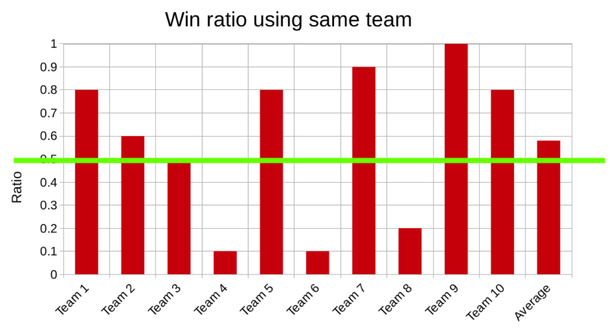

Fig. 3. Win ratio per team from the Q-learning system's perspective. The green line is located at 50%. Therefore, with 6/10 teams the system won more then 50% of battles.

In Fig. 4 the set of battles for team 3 is analyzed, showing the accumulated victories of each system at each battle. This team was chosen as representative of a hard match-up for the Q-learning system, adapting itself for winning after a defeat.

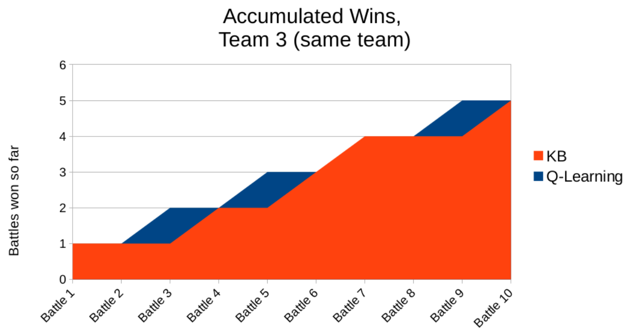

Fig. 4. Accumulated wins of each system for every battle with Team 3.

4.2 Different teams

For this experiment the performance was lower, with the KBS dominating 71% of the battles. Therefore, the Q-learning system's performance was below the base-line. As before, a more detailed perspective can be obtained by inspecting Fig. 5.

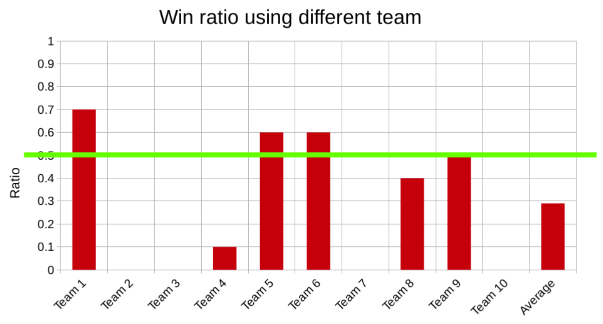

Fig. 5. Win ratio per team from the Q-learning system's perspective. The green line is located at 50%. Therefore, with just with 3/10 teams the system won more then 50% of battles, losing 100% of battles with 4 teams.

4.3 10-Attempts to Win

From this experiment, the system showed the ability to overcome human players by learning of them over time. From the 10 teams set, the experiment was a success for 6 teams. From those, with 4 teams the system achieved victory before the 5th attempt. These results are shown in Fig. 6

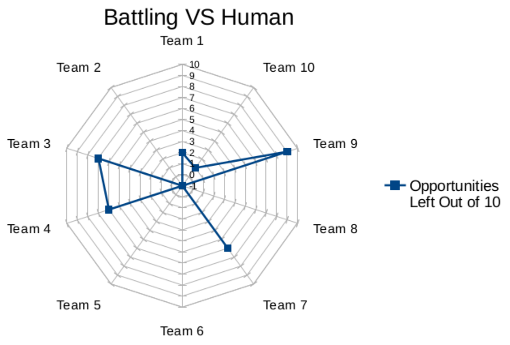

Fig. 6. Results of the 10-attempts experiment. For each team, the graph shows how many attempts the system had left when it achieved victory for first time. -1 means the system lost the 10 attempts.

5 Conclusions and Future Work

Even a simple representation of the game led to a huge states-space, therefore a great amount of battles is needed for a proper training (under the assumption of taking 7 different actions per state, each one once, and an average of 15 states per battle, more than 20,000 battles would be needed). Under this scope, the system is vulnerable in states not visited, or not enough visited (as taking one action once in a state does not allow a proper training to learn its expected reward).

This most likely explains the results gotten in both experiments. By using the same teams, even if the Q-Learning loses the first battles (look again at

Fig. 4), its ability to learn on-line allows it to learn how to use its team better by looking the rewards achieved by both players using the same pokémon. When using different teams, it could be said that the system learns at most half as quick as before; when facing difficult combats, it would take a long time to learn the best policies for the given states.

However, as the system is able to learn on-line, by practicing the same team many times the Q-function learning can be guided to convergence in desirable states. Even more, the learning process is highly parallelizable as many battles can be run simultaneously on many machines updating a single Q-function, either for the same or different teams.

The same team experiment results suggest that the Q-learning approach is a good option as it showed promising data. As future work, and to deal with the huge states-space (and further refining those states for a more detailed game description), a Deep Q-Learning technique can be used (taking into account the computational costs inherent to Deep Learning). Another option comes in the form of Factored Markov Decision Processes, using human knowledge to create compact representations and thus reducing the states-space.

Also as future work, but focusing on gameplay rather than the learning process, two approaches are proposed. The first one is using the learned Q function for a Minimax technique; as this function works for every possible state rather than players, the mini and max actions can be estimated by the expected reward from each player's perspective. In this way, a prediction of the opponents action can be made and used by the system just like a human-player. Also, for extending the action evaluation over the battle, the policy learned by the system can be tested prior to results with a Monte Carlo approach.

References

1. Maynard Smith, J. (1974). "Theory of games and the evolution of animal conflicts." Journal of Theoretical Biology 47, 209-221.
2. "Bulbapedia, the community driven Pokémon encyclopedia." https://bulbapedia.bulbagarden.net/wiki/Main_Page
3. Furtak, T. and Buro, M. (2010). "On the complexity of two-player attrition games played on graphs." Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment Conference.
4. Dayan, P. and Watkins, C. (1992). Q-Learning. Machine Learning 8, 279292.
5. Xu, S. and Verbrugge, C. (2016). "Heuristics for sleep and heal in combat." 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini, 1-8.
6. Lee, S. and Togelius, J. (2017). "Showdown AI Competition." 2017 IEEE Conference on Computational Intelligence and Games (CIG), New York, NY, 191-198.
7. Van Der Wal, J. (1981). "Stochastic dynamic programming". Mathematical Centre Tracts 139, Morgan Kaufmann, Amsterdam.
8. Sucar, L. E. (2015). "Probabilistic Graphical Models: Principles and Applications", Springer.