Learning With Humans: A Case Study with Pokmon Showdown!

Jonathan Araujo and Fabrício Olivetti de Franca, *Member, IEEE*
Universidade Federal do ABC (UFABC)
Center of Mathematics, Computing and Cognition (CMCC)
R. Santa Adélia 166, CEP 09210-170, Santo André, Brazil
Email: folivetti@ufabc.edu.br

***Abstract*—In many Computational Intelligence in Games research, the goal of evolving intelligence requires an heuristic-based approach.**

Computational Intelligence in Games comprises many challenges such as the procedural level generation, evolving adversary difficulty and the learning of autonomous playing agents. This last challenge has the objective of creating an autonomous playing agent capable of winning against an opponent on an specific game. Though a human being can learn a general winning strategy (i.e., avoid the obstacles and defeat the enemies), learning algorithms have a tendency to overspecialize for a given training scenario (i.e., perform an exact sequence of actions to win) that does not work when facing variations of the original scenario. To further study this problem, we have applied three variations of Neuroevolution algorithms to the EvoMan game playing learning framework with the main objective of developing an autonomous agent capable of playing with different scenarios than those observed during the training stages. This framework is based on the bosses fights of the well known game called Mega Man. The experiments show that the evolved agents are not capable of winning every challenge imposed to them but they are still capable of learning a generalized behavior.

Keywords*—*video game playing, autonomous agent, neuroevolution.

I. Introduction

Computational Intelligence in Games [1] is a research field that studies the application of Machine Learning techniques in different applications in Games and Video Games.

One of these applications is the creation of an autonomous agent capable of playing and winning a given game. In classic board games, the objective is for the agent to be capable of beating a human champion. Many board games, like chess and Go, have perfect information but with an intractable game tree, making it difficult to find the optimum strategy.

Regarding Video Games, the autonomous agent can be used to test the Machine Learning algorithms [1], the dificulty of a certain game [2] or to create an agent that imitates the human behavior [3].

In all of these applications, when the agent must compete against others players, the Machine Learning algorithm employs a supervised approach by playing against pre-programmed agents, using artificial scenarios or coevolving against others learning algorithms [4].

This may limit the learning process of the agent since it will be biased towards the scenarios presented to the algorithm. Also, when the objective is to mimic a human player, additional rule-based heuristics may be required to complement the learned behavior [5].

On the other hand, training the agent against human players is unfeasible to most games, since the learning process often requires a large number of iterations until the agent starts showing a competing behavior. This often means that the agent must play millions of matches before being competitive.

One way to train against human players is to create a communication API enabling an autonomous agent to play an online game that contains thousands of players actively playing on a daily basis.

In this paper we introduce the Pokmon Showdown! Game Playing API, that enables an autonomous agent to play the Pokmon Showdown [1] game, an online strategy game that reunites about $10,000$ of players online at any time of the day.

With this API it will be possible to verify whether an autonomous agent learning from a real-world scenario converges to a more competitive behavior. This API is tested with Monte Carlo Tree Search algorithm trained against human players and compared against the same algorithm trained through a coevolutionary procedure with the same amount of matches.

The paper is organized as follows: Section II reviews some of the researches related to this project, Section III introduces the Pokmon Showdown! and the proposed API. In Section IV the methodology used for this research is described together with the obtained experimental results. Finally, in Section V we conclude the paper with some final remarks of the experiment, as well as some perspective experiments for future investigation.

II. Related Research

In this Section we will make a brief review of the current research in Computational Intelligence in Games (CIG), related to the topic of this paper, as a motivation for the current research.

[1]http://pokemonshowdown.com/

A. Autonomous Game Playing Agents

One of the challenges sought to be surpassed in CIG is the creation of autonomous agents capable of winning a certain game with super-human abilities. In other words, if the game is meant to be played by only one player, the agent must be capable of surpassing the high-score of a human player; on the other hand, if the game is a two player competitive game, the agent must learn how to win against a human player.

In early years, the CIG community focused on classic board games like chess, checkers, GO, Othello [1]. Nowadays, this research has expanded to electronic Video Games, in which the agents are trained within an emulated version of the game [6], an adaptation of the real game [7], [8] or through an API that communicates with the original game [9].

Though a few games can be solved by a standard minimax algorithm, most interesting games have one or more features that restrict the use of this algorithm as it is. For example, a game may have a large search space such as that building a game tree becomes computationaly prohibitive. Another issue is the presence of uncertainties, limiting the estimation of the scores for each branch.

With the goal of treating such problems, the use of Evolutionary Computation has been widespread in the community within different contexts. For example, a seminal paper by Fogel et al. [10] applied a Multi-layer perceptron (MLP) that was evolved through co-evolution in order to generate an heuristic function capable of estimating the scores of each branch of a minimax tree for the Checkers game.

The combination of Evolutionary Computation with MLP, to evolve the weights or the topology itself, is called neuroevolution. The main advantage of neuroevolution is the flexibility and, in many cases, simplicity of implementation while maintaining a competitive learning capability [4].

Another commonly used technique is Monte Carlo Tree Search [11], [12] (MCTS), that expands only the most promising branches of a minimax tree estimated through random simulations. Each simulation is guided by the knowledge obtained through previous simulations with the expectation of reaching the optimal branching of the game tree, without the need of expanding every possible branch.

The simplicity of both techniques lies on the fact that the only informations you need about the game is a fitness function that measures how well an agent performs and, if desirable, an heuristic function to estimate the most promising branches.

In competitive games, the fitness function can be a measure of how well an agent performed against other player. Ideally, in such cases, the adversary is a highly skilled player from whom the agent will learn.

Both techniques have been succesfuly applied in video game playing problems like Ms. Pac-Man [13], [14], racing games [15], [16] (TORCS), Real-time Strategy Games [17], [18] (StarCraft) and many others.

One donwside of both approaches is the high number of function evaluations (i.e., games played) required until the agent learns an acceptable competing behavior. This limits the use of highly skilled human adversaries during the training stage, as it would be impractical to play thousands of matches against them.

So, one question that remains unanswered is whether training an autonomous agent against a human player would benefit the learning process.

B. Human-like Behavior

Another interesting challenge in CIG is the creation of agents capable of mimicking human behavior, like a turing-test for video game playing. In this challenge, the agents must be capable of playing a match against human players (the results of the match does not matter) without the human player suspecting they are actually autonomous agents.

To overcome this challenge may be particularly difficult since there is no clear fitness function that can quantify how similar an agent is to a human behavior. As such, the most successful approaches have hardcoded the human behavior into the agent through rule-based heuristics [5].

This challenge has been turned into a competition in 2008 [3] in which the agents should evolve a behavior capable of fooling human players into believing they are also humans.

The competition consisted of matches of the multiplayer game called Unreal Tournament [2]. Every match contained an equal number of human players, native bots (pre-programmed with the game) and autonomous agents competing in teams. The teams, player names and skins were randomly select so there was no previous information about who is who. The primary weapon was called *Judging Gun* and was used to tag each player as Bot or Human.

Whenever the human player guess correctly whether the enemy is a human or a bot, they receive $+10$ points and the enemy is eliminated from the match. If the guess is wrong, they receive -10 points and is eliminated from the match.

The first two agents that was considered winners of this challenge was $UT\hat{2}$ proposed in [5] and MirrorBot, proposed in [19].

The $UT\hat{2}$ Bot segmented the agent behavior into modules that modelled human traces like getting *unstuck*, getting a dropped weapon, firing the *juding gun*, battling, etc. Most modules are executed as a rule-based playback of real human actions recorded from different matches. The *battle* module was trained through the use of a Multiobjective neuroevolution combining NSGA-II [20] and NEAT [21] algorithms by maximizing the damage dealt and minimizing damage received and object collision (humans hardly collide with scenario objects).

The MirrorBot agent implements two modules named *default* and *mirror*. The default module is a rule-base module that allows the agent to navigate the map, judge a given player as human or bot and attack enemy players. Whenever the agent finds a friendly player, it swaps into mirror mode where it will start recording the movements from the other player and it will play it back after a short delay.

Regarding the practical implicatons of such challenge, the evolution of agents that act similar to human players may be

[2]https://www.unrealtournament.com/

interesting to the game industry in order to create an AI capable of creating a realistic challenge.

Another question comes from this challenge, the autonomous agent would have a human-like behavior if trained against human players?

On a related research, in [22] the authors have evaluated the influence of a human-guided learning environment. In the proposed environment, each human could train their agents by specifying the challenges the agenst would have to accomplish. The authors showed that different types of strategies emerged from such environment.

In the next section we will propose a framework that will allow us to perform experiments with agents training against humans so that we can work toward answering those questions.

III. Pokmon Showdown!

Pokmon [23] is a long stand RPG game series owned by The Pokmon Company [3] that takes place in a fictional world inhabitaded by creatures known as Pokmon (from *Pocket Monster*). The player starts the game with one of such Pokmons and he must catch and train new Pokmons while exploring the world.

The trained Pokmons are then used to battle each other on a tournament during the gameplay story. Each Pokmon has unique abilities and properties that evolves while aquiring experience points.

The battles are designed as a turn-based system where each player chooses the next action from: performing an attack, change Pokmon (the player may carry up to six Pokmons), use an item and flee from battle.

After each player decides the action to take, the game system determines the first action to be performed by some fixed rules: if one player chooses to use an item, change Pokmon or flee, this action is performed first; if both players choose the same action, the one player with a Pokmon of higher speed status has its action performed first.

Every attack deals some damage to the opponent Pokmon and, when a Pokmon *health points* (HP) reaches zero, this Pokmon faints and is replaced by the next in line. The player who loses all of its Pokmons first will lose the battle.

The amount of damage dealt with each attack is a function of the properties from both Pokmons (the one attacking and the one being attacked) and their types.

The Pokmon Showdown! is a free web-based Pokmon battle simulator with many different servers containing thousands of players competing simultaneously. The game source code is open-sourced under the MIT license [4].

This game contains more than 700 different types of Pokmons, some of them based on the original games and some created by the maintainers of this game. Each Pokmon is described by different properties, some of them are used in battle to determine the amount of damage being dealt by an attack:

[3]http://www.pokemon.com/

[4]https://github.com/Zarel/Pokemon-Showdown

- **Type:** each Pokmon may belong to one or two different types (see Table **??**).
- **HP:** health points the Pokmon currently possess.
- **Attack:** total points of attack for dealing damage.
- **Defense:** total points for defending against an attack.
- **Special Attack:** total number of special attacks this Pokmon can use.
- **Special Defense:** total number of special defenses this Pokmon can use.
- **Speed:** the speed points of the Pokmon.
- **Abilities:** each Pokmon has a total of 3 possible abilities from a list of 200 different abilities. The player may choose one of those three for his own Pokmon.
- **Level:** a number between $[0, 100]$ that describes the experience of the Pokmon in battle.
- **Effort Values (EV):** points that can be used to increase the points of *HP*, *attack*, *defense*, *special attack*, *special defense* and *speed*.
- **Individual Values (IV):** 31 points that can be used to increase one of *HP*, *attack*, *defense*, *special attack*, *special defense* and *speed*.
- **Nature:** each Pokmon can belong to one of 25 different natures. The nature may affect the points of *attack*, *defense*, *special attack*, *special defense* and *speed*, increasing or decreasing them by 10%.

The total points for *attack*, *defense*, *special attack*, *special defense* and *speed* is calculated by the following equation:

$$V_i = \left(\frac{(2 \times B + IV + \frac{EV}{4}) \times Level}{100} + 5 \right) \times N(i), \quad (1)$$

where V_i is the value for status i, B_i is the base value of status i for this Pokmon and $N(i)$ is the modifier for status i given this Pokmon nature (see Table I).

The total HP of the Pokmon is similarly calculated by:

$$V_i = \frac{(2 \times B + IV + \frac{EV}{4}) \times Level}{100} + Level + 10. \quad (2)$$

It should be noticed that the ability chosen for the Pokmon may increase the Base value (B) for any of the status points.

The damaged dealt by each attack is calculated based on the properties of the Pokmon and of the chosen attack:

$$Damage = \left(\frac{2 \times Level + 10}{250} \times \frac{Attack}{Defense} \times B_a + 2 \right) \times Mod_{p,a}, \quad (3)$$

where B_a is the base status value of attack a and $Mod_{p,a}$ is the modifier for the Pokmon p with attack a, given by:

TABLE I. Different types of Nature a Pokmon can have and their modifiers.

Nature	Modifier 1	Modifier 2
Lonely	$1.1\times$ Attack	$0.9\times$ Defense
Brave	$1.1\times$ Attack	$0.9\times$ Speed
Adamant	$1.1\times$ Attack	$0.9\times$ Special Attack
Naughty	$1.1\times$ Attack	$0.9\times$ Special Defense
Bold	$1.1\times$ Defense	$0.9\times$ Defense
Relaxed	$1.1\times$ Defense	$0.9\times$ Speed
Impish	$1.1\times$ Defense	$0.9\times$ Special Attack
Lax	$1.1\times$ Defense	$0.9\times$ Special Defense
Modest	$1.1\times$ Special Attack	$0.9\times$ Attack
Mild	$1.1\times$ Special Attack	$0.9\times$ Defense
Quiet	$1.1\times$ Special Attack	$0.9\times$ Speed
Rash	$1.1\times$ Special Attack	$0.9\times$ Special Defense
Calm	$1.1\times$ Special Defense	$0.9\times$ Attack
Gentle	$1.1\times$ Special Defense	$0.9\times$ Defense
Sassy	$1.1\times$ Special Defense	$0.9\times$ Speed
Careful	$1.1\times$ Special Defense	$0.9\times$ Special Attack
Timid	$1.1\times$ Speed	$0.9\times$ Attack
Hasty	$1.1\times$ Speed	$0.9\times$ Defense
Jolly	$1.1\times$ Speed	$0.9\times$ Special Attack
Naive	$1.1\times$ Speed	$0.9\times$ Special Defense
Hardy	None	None
Docile	None	None
Serious	None	None
Bashful	None	None
Quirky	None	None

$$Mod_{p,a} = St_{a,p} \times T_{p,p'} \times Critic \times Effect \times U(1.85, 2), \quad (4)$$

where $St_{a,p}$ is a modifier based whether the attack type is the same of the Pokmon, $T_{p,p'}$ is a modifier based whether the type of Pokmon p has any (dis)advantage to the attacked Pokmon p', $Critic$ will evaluate to 1.5 with a chance of 6.25% or else it will be equal to 1, $Effect$ is a modifier depending on the current effect of the Pokmon due to the use of an item or ability, $U(1.85, 2)$ is an uniformly generated random number between 1.85 and 2.

The Type system of this game was initially concieved as a rock-paper-scissor scheme, where a given type x had advantage points towards type y and disadvantage points against type z. For Pokmon Showdown! there is an specific hard-coded table that calculates the type advantages.

A. Pokemon API

In order to allow an autonomous agent to play against human adversaries we have created an API to communicate with the game by using websockets written in JavaScript.

IV. Experiments

In this Section we will describe the experimental setup used to test the API and to investigate the questions raised earlier on this paper. First we will explain the algorithm Monte Carlo Tree Search with more details, next we will describe the methodology to investigate those two questions and finally we will report the obtained results.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an algorithm devised to estimate the most promising branches of a game search tree through sampling and simulation process. The algorithm is based on a Game Tree where each level corresponds to the possible choices one player can make given the current status of the game. On a two player game the odd levels correspond to the first player and the even levels to the second player.

Each node of this tree contain information regarding the number of matches that used a path containing this particular node and how many of these matches rendered a victory for the player corresponding to this node.

By using this tree the MCTS iteratively perform four steps:

- **Selection:** heuristically chooses a path of the tree until it reaches a node in which at least one of its child was never chosen during a match.
- **Expansion:** chooses one of the univisted childs uniformly at random.
- **Simulation:** end the match expanding this branch until it reaches a leaf node by choosing the next actions at random or with the help of an heuristic function.
- **Backpropagation:** the nodes information regarding the number of matches and the number of victories for each player is updated.

In this work we will use the UCB1 [24] strategy for the Selection step in which the child of the current node that will be selected is the one that maximizes:

$$nv_i + \sqrt{\frac{2 \ln n}{n_i}}, \quad (5)$$

where nv_i is the total number of victories obtained by following node i, n is the total number of simulations and n_i is the total number of matches that used node i. This is the upper bound for the confidence interval.

For the Simulation step we will choose the next actions of the agent at random so that we do not introduce any bias for the learning process. In future works we will also experiment with a crafted heuristic function and an heuristic funcion learned from a Neural Network [12].

B. Methodology

In order to answer the imposed questions we will perform two different experiments, each one composed of a training stage, consisting of $10,000$ matches and a testing stage, consisting of $1,000$ matches.

The two experiments will differ during the training stage in which one will train against human players and the other will co-evolve against another instance of the MCTS.

After the training stage, both agents will be tested against human players and we will compare both by measuring the number of victories, average of the sum of the HP value of each Pokmon after the end of the match and the number of human players that typed the word *bot* (or equivalent in other languages) on the chat box.

The purpose of the first measure is to verify whether an autonomous agent trained against human can perform better than another one trained on a controlled environment.

The second measure is an extension of the first and it will measure how well each agent performed during the matches. With this measure, we can assess which agent has generated the best strategy.

Finally, the last measure is to verify how often each agent is identified as a bot. If the difference between both bots are statistically significant, it can be an indication of whether training with humans have any influence on how the agent behaves.

C. Results

V. Conclusion

References

[1] S. M. Lucas, "Computational intelligence and games: Challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008.

[2] G. N. Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," *From animals to animats*, vol. 8, pp. 499–508, 2004.

[3] P. Hingston, "A turing test for computer game bots," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 3, pp. 169–186, 2009.

[4] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," 2014.

[5] J. Schrum, I. V. Karpov, and R. Miikkulainen, "Ut 2: Human-like behavior via neuroevolution of combat behavior and replay of human traces," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 2011, pp. 329–336.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, 2012.

[7] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[8] K. S. M. de Araujo and F. O. de Franca, "Um ambiente de jogo eletrnico para avaliar algoritmos coevolutivos," in *Proceedings of the 12th Congresso Brasileiro de Inteligłncia Computacional*, 2015.

[9] J. Renz, "Aibirds: The angry birds artificial intelligence competition." in *AAAI*, 2015, pp. 4326–4327.

[10] K. Chellapilla and D. B. Fogel, "Evolving an expert checkers playing program without using human expertise," *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 4, pp. 422–428, 2001.

[11] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[13] M. Gallagher and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. IEEE, 2007, pp. 282–287.

[14] S. Samothrakis, D. Robles, and S. Lucas, "Fast approximate max-n monte carlo tree search for ms pac-man," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 2, pp. 142–154, 2011.

[15] J. Fischer, N. Falsted, M. Vielwerth, J. Togelius, and S. Risi, "Monte carlo tree search for simulated car racing."

[16] D. Galanopoulos, C. Athanasiadis, and A. Tefas, "Evolutionary optimization of a neural network controller for car racing simulation," in *Artificial Intelligence: Theories and Applications*. Springer, 2012, pp. 149–156.

[17] J. S. Zhen and I. Watson, "Neuroevolution for micromanagement in the real-time strategy game starcraft: Brood war," in *AI 2013: Advances in Artificial Intelligence*. Springer, 2013, pp. 259–270.

[18] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios." in *AIIDE*, 2012.

[19] M. Polceanu, "Mirrorbot: Using human-inspired mirroring behavior to pass a turing test," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[21] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[22] I. V. Karpov, L. M. Johnson, and R. Miikkulainen, "Evaluating team behaviors constructed with human-guided machine learning," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 292–298.

[23] S. Kent, *The Ultimate History of Video Games: from Pong to Pokemon and beyond... the story behind the craze that touched our li ves and changed the world*. Three Rivers Press, 2010.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.