An Adversarial Approach for Automated *Pokémon* Team Building and Metagame Balance

Simão Reis, Rita Novais, Luís Paulo Reis, and Nuno Lau

***Abstract*—Metagame balance is a crucial task in game development, and automation of this process could assist game developers by vastly reducing time costs. We explore and evaluate a metagame balance model over the recently proposed VGC AI Competition Framework. We propose an adversarial model where team builder agents try to maximize their win rate by narrowing to the most optimal team configurations, resulting in a reduction of the diversity of *Pokémon* employed, while a balancing agent readapts the *Pokémon* inner attributes to incentivize the team builder agents to incorporate a greater variety of *Pokémon* into their teams increasing the metagame's overall diversity and balance. Furthermore, we develop multiple team builder agents divided into two groups: the first group assumes that individual *Pokémon* advantages are the primary factor to determine the outcome of game matches; the second group also exploits the implicit synergy between teammates. These agents make use of metagaming, linear optimization, and evolutionary search to find strong combinations against the current metagame. The strongest team builder is faced against the team metagame balance agent for its evaluation. Deep learning is also employed to predict the outcome of matches and recommend constructive elements of teams.**

***Index Terms*—Automatic game design, competitive games, deep reinforcement learning, evolutionary algorithms, game balance, game theory, optimization.**

I. Introduction

ESPORTS and commercially successful competitive games have changed the panorama of how video games are presented to players and consumed by them. This universe of games is often characterized not only by its genre but also by the fact players can control one or several units, having the choice of what unit(s) they intend to use beforehand. It is important that the choices of the players are meaningful and provide opportunities for diverse gameplay. The process to enforce such requirements in a game is known as balance.

In reality, there is a lack of consensus on what a "balanced game" really means. Becker and Görlich [1] examine reports from 14 lead game designers and critics, trying to construct a conceptual basis for the term. The authors conclude that the four elements more present in the reports are: 1) difficulty; 2) balancing (as a purposive act); 3) symmetry; and 4) expected characteristics of a well-balanced game. One of those expected characteristics is that multiple winning strategies should be viable. The authors also report that balancing as a process is not unique, with probable cause for the lack of consensus in the term. Most importantly, the balancing process is difficult because all the elements of a game work in a fragile equilibrium where small changes may drastically change how the game works, thus changing how the players see the game.

Carter et al. [2] discuss the definition of the term metagame. The term is employed by players, game designers, and academics as both activities performed outside the game, and as a high-level strategic analysis of the behavior of opponents and their strategic choices before a game starts. It is important to distinguish between metagame balance and in-game balance: Metagame balance is an inverse process where strategies are stimulated to be selected by players before a game starts by tuning game elements; and in-game balance is an act that affects the relevance or individual in-game actions toward victory, without being strictly dominating in a significant quantity of game states.

The VGC AI Competition [3] provides a common framework to build and test *Pokémon* battle agents, team builder agents, and meta-balance agents in *Pokémon* (Game Freak, Creatures, Nintendo, 1997). The framework runs a self-contained metagame ecosystem, recreating a controlled Esports environment that game designers tune. In the official series, *Pokémon* can be configured in a team with a number of members depending on the format. Currently, for the VGC AI Competition, three *Pokémon* must be selected from a roster generated at the beginning of the competition with an arbitrary size. Therefore, for team building, the roster is unknown before the balance competition starts; this adds enough complexity that humans are required to perform metagaming, observing opponents' team-building trends from public statistics to anticipate their opponent's choices. Inside the VGC AI Competition, these data are provided from the framework itself to team builder agents, containing usage rates of individual *Pokémon* and teams. For metagame balance, the roster has to be fine-tuned periodically, while a team builder population competes, requiring adapting to the builder agents.

Manuscript received 29 October 2022; revised 28 February 2023 and 26 April 2023; accepted 2 May 2023. Date of publication 4 May 2023; date of current version 18 June 2024. This work was supported in part by the Foundation for Science and Technology and the European Social Fund under Grant SFRH/BD/129445/2017, Grant COVID/BD/151875/2021, and Grant 2021.05237.BD, in part by the Artificial Intelligence and Computer Science Laboratory under Grant UIDB/00027/2020, and in part by the Institute of Electronics and Informatics Engineering of Aveiro under Grant UIDB/00127/2020. *(Corresponding author: Simão Reis.)*

Simão Reis, Rita Novais, and Luís Paulo Reis are with the Artificial Intelligence and Computer Science Laboratory, Faculty of Engineering, University of Porto, 4099-002 Porto, Portugal (e-mail: simao.reis@outlook.pt; up202103589@up.pt; lpreis@fe.up.pt).

Nuno Lau is with the Institute of Electronics and Informatics Engineering of Aveiro, University of Aveiro, 3810-193 Aveiro, Portugal (e-mail: nunolau@ua.pt).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TG.2023.3273157.

Digital Object Identifier 10.1109/TG.2023.3273157

Previous work assumes the strategies to be balanced already found or that they do not change over time [4], which is an unrealistic setting during game design or maintenance stages. We explore a new metagame balance adversarial paradigm to reach a scenario where the number of competitively viable *Pokémon* over the VGC AI Competition is increased: while rational team builders try to narrow to the strongest teams, the balance agents try to maximize the number of deployed *Pokémon* by team builders through *Pokémon* adjustments.

We also propose a set of new team-building agents for our adversarial model. The team builder and game-balancing agents mix evolutionary techniques, linear programming, a generator neural network (NN), and an NN that predicts the win rate of matches. The training schema for the predictor network consists of generating a dataset of random pairs of legal teams and simulating battles between the same to extract target win rates. If team A beats team B, the win rate w is used as a target for the predictor network. This allows for sample augmentation because if team A beats team B with a win rate of w, then team B wins against team A by $1 - w$. For our generator network, we curated the training dataset by searching strong counter teams A against B and using their constructive elements as target Q-values to rank the most important constructive elements, allowing us also to rank *Pokémon* against a target *Pokémon*.

The outcome of *Pokémon* matches depends on a metagaming phase and on the individual skill of the players battling. Meta-knowledge takes an important role not only during team building but during battle, because players perform prediction, bluffing, take risks, and provide false information while becoming unpredictable to exploit suboptimal plays of the opponent. We set all the battle agents as the same greedy tree search agent (that does not use meta-information), reducing the outcome of a game match to the chosen teams and making it feasible to train a win rate predictor. A similar approach was done by Chen et al. [5], where a small set of pretrained agents with specific game styles was used to play a small set of decks. We believe this to be reasonable at this stage, as developers often use only high-level competitive play data, where player behavior is close to optimal, to gain insight into design flaws. For a more robust model targeting a wider audience, it would either need to parameterize a player population's expected policies through human modeling or reach a superhuman agent able to perform all the mentioned skills, which is outside the scope of this article.

II. Related Work

A. Deck/Team Builder Agents

Ward et al. [6] collected a dataset of human Magic: the gathering draft format decisions and proposed a heuristic agent, an expert-tuned complex heuristic agent, a Naive Bayes agent (that promotes co-occurrence of pairs of cards), and a deep NN agent trained by supervision. Bertram et al. [7] discuss how the draft format is context dependent and is not sufficient to just evaluate cards in a vacuum, as their value is not dependent on the sum of their individual values but on combinations, and suggest a contextual preference ranking framework, using a Siamese network trained with a triplet loss. Kowalski and Miernik [8] used a variant of the evolutionary algorithm to draft cards in the arena mode of Legends of Code and Magic. Many other works focused on drafting can be found in the literature. Bhatt et al. [9] explored evolutionary strategies on Hearthstone (Blizzard, 2014), where cards deemed important for the success of a deck are preserved and mutation explores which new cards can be added to increase the win rate of that deck. Q-DeckRec [5] learns a deck search policy and solves deck building with less computation against baseline agents.

Crane et al. [10] defined Team Counter-Selection Games as games where candidate selection is impacted by rotating meta where new candidates create counter meta to the previous. They use epistemic reasoning to anticipate the possible outcomes of current teams and find the best counter candidates and evaluate their solution on online *Pokémon Go* (Niantic, 2016) leagues. da Silva Oliveira et al. [11] explored three optimization techniques, i.e., iterative local search (ILS), genetic algorithm (GA), and memetic algorithm (MA) to provide team recommendations for *Pokémon Go*. ILS obtained the best execution time, while MA obtained the best fitness.

Our scenario differs from the previous ones, as agents have to compete and adjust to a constantly changing metagame and roster, requiring new approaches for *Pokémon* team building. The conceived approaches are inspired by the latter two. The GA is used to find counterstrategies and perform some form of epistemic reasoning over a team selection normal form metagame, whose number of entries may be highly dimensional, and past choices from a population can be queried.

B. Automated Game Balance

Andrade et al. [12] presented an evaluation of different game-balancing approaches by human players. Four agents were evaluated: a state machine with static behavior, a genetic agent, a reinforcement learning (RL) agent, and a trained challenge-sensitive RL agent. The results demonstrated that the agents with game-balancing approaches performed close to the human player and received better feedback in terms of user satisfaction—these results showed that game balance is highly correlated with user satisfaction.

Leigh et al. [13] studied the application of coevolutionary algorithms, GA variants, for adversarial evolution in continuous games. The balance was evaluated by the distribution of strategies employed by the agents. Jaffe et al. [14] proposed a prototype balancing system for two-player perfect information discrete games. This prototype of restricted play includes several categories of balance features and restricted behaviors to estimate their impact on the game. Delaurentis et al. [15] explore how game design methodologies can be brought over to engineering design applications.

An unbalanced game is identified through perturbations on important features identified using an explainable AI technique called SHapley Additive exPlanations [16]. Moroşan and Poli [17] focused on tailoring the prowess of dominant strategies using evolutionary techniques, testing in both Pac-Man and StarCraft. Beau and Bakkes [18] proposed the evaluation of impactful actions that cause power to unbalance in two-player asymmetric games using Monte Carlo simulations and adjusting the attributes by removing available game actions. Volz

et al. [19] evaluate multiobjective optimization for automated game balancing, namely maximize fairness in win rate and the number of possible tricks of Top Trumps card game, using Open Trumps [20], a simulator and dataset of procedurally generated decks. Beyer et al. [21] discussed automated game balancing in the Business Process Management domain. Morosan and Poli [22] discussed how previous methods of academic research in game balance had minimal impact on the industry, as many methods are computationally expensive. Jointly with a game studio, the authors defined a specification language and test GA search over a vector of parameters.

Tomasev et al. [23] used AlphaZero to explore alternative atomic rules of chess, through self-play learning, and to perform quantitative analysis relating the winning rate of the white and black players. Diversity is also analyzed in terms of visited states' entropy, where the variety of rich games is maximized. Pfau et al. [24] proposed deep player behavior modeling, where a dataset generated over six months, from 213 players from the online game *Aion* (NCSoft, 2009), is used to train a deep model that maps game states and previous player skill information to actions. This information is used to capture imbalances in the game properties.

Kavanagh and Miller [25] introduced chained strategy generation (CSG), a static analysis procedure where dominant strategies are identified, and adversarial probabilities are calculated, simulating the evolution of a population of players. The method is tested in RPGLite [26] and provides an available gameplay dataset. Kavanagh et al. [27] extended this work to a more complex version of RPGLite. Kavanagh and Miller [28] used state and action lookup tables and action costs, quantifying how much is lost by performing a given action, allowing to evaluate a player's ability and account that information for the game balance process with CSG.

Some of the works indicate that using population modeling could be achievable to represent a real Esports player base. An in-game balance track could also be added to the VGC AI Competition, to guarantee that the use of certain moves can lead to winning strategies, or even full move sets and game mechanics, but are outside the scope of this article.

C. Automated Metagame Balance

To address the problem of achieving a balance between the available strategies in a game, Liu and Marschner [29] modeled this problem considering zero-sum games and proposed a game-theoretic approach to automatically balancing strategies based on the mathematical model of the games. The authors defined handicap functions that affected the payoff of the game based on the strategies chosen by the two players and a handicap variable impacting the total strength of each strategy. The main goal was to achieve a balanced game where every strategy has the incentive to be employed. The authors applied multiplicative handicaps in the *Pokémon* battle game, but they only considered a single attribute. To find the handicaps, they used the Sinkhorn–Knopp algorithm and showed that the algorithm was able to balance the game.

de Mesentier Silva et al. [4] proposed a method for representing the meta-balance and aimed to minimize the number of changes through evolutionary search while maximizing the balance. They concluded that win rate when played and win rate when drawn are potential heuristics that can be used to choose which cards to nerf (reduce their advantage). The authors defined a balanced meta where all decks have a 50% win rate; however, they suggest other metrics like average deck win rate, which permits local intransitivity and deck diversity. Hernandez et al. [30] present a tool for balancing multiplayer games during game design, using a graph representation of their target metagame (they define metagame as a set of high-level strategies that are abstracted from the atomic game actions) and use simulation-based optimization to minimize the distance of the current meta against the target meta. Mahlmann et al. [31] explored three different fitness functions to generate balanced card sets in the card game of Dominion and found that some cards make the game balanced independently of player skill or behavior. The authors applied an integer-valued GA with two fitness functions to promote interestingness and balance. Fontaine et al. [32] developed a technique called MAP-Elites with Sliding Boundaries, a quality diversity algorithm, and applied it to the design and rebalancing of Hearthstone, as it can uncover complex relationships in gameplay.

Our metagame scenario differs in that changes are done periodically by adapting to a team-building agent population. The major novelty of this article resides in the balance adversarial model: while rational team builders naturally narrow their choices to the strongest teams inside a metagame context, the balance agents must search for desirable *Pokémon* attributes to increase motivation for more diverse play.

III. Terminology and Formalization

Consider a set of game pieces ψ_i called a Roster $R = \{\psi_0, \ldots, \psi_n\}$, battle (or gameplay) policies π, a base multiplayer game $G(\psi_i, \psi_j, \pi_i, \pi_j)$ parameterized by a duple of pieces (ψ_i, ψ_j), and a duple of battle policies (π_i, π_j), where π is assumed fixed and the same for each agent. The outcome of $G(\psi_i, \psi_j)$ is a win rate pair $w_i = w(\psi_i, \psi_j), w_j = w(\psi_j, \psi_i)$, where $w_i + w_j = 1$. Finally, consider metadata usage $\mathbf{u}$ (or Meta), which is a vector containing the usage rate u of each piece $\mathbf{u}[i]{=}u(\psi_i)$. Each entry of a normal-form metagame $\mathbf{M}_{i,j} = \mathbf{M}[i,j] = w(\psi_i, \psi_j)$ contains the individual pairwise win rates. The aim of a meta policy π_M is to maximize its expected win rate E while decreasing the opponents $\pi_M = \max_i \min_j E(\mathbf{M}_{i,j})$. In the context of this work, pieces can either be *Pokémon* teams $\psi_i \equiv t_i$ or individual *Pokémon* $\psi_i \equiv p_i$. Team usage will be designed by $\mathbf{u}_t$, while the usage of an individual *Pokémon* is designed as $\mathbf{u}_p$. In the same fashion, we design $\mathbf{M}_p$ as the metagame of individual *Pokémon* and $\mathbf{M}_t$ as the metagame whose row and column entries are teams.

Assuming that the metagame is being played by rational agents trying to maximize their win rate and, therefore, also minimize the win rate of opponents (constant-sum game), we define perfect meta-balance as a scenario where any pair of pieces have an equal chance of being selected (or equal expected usage rate) and, therefore, rendering the decisions shallow, as an optimal player becomes no better than a random player. The opposite is extreme meta unbalance where the average usage

TABLE I
INDIVIDUAL *POKÉMON* COUNTER

Roster R	Metagame $\mathbf{M}_p$			Meta $\mathbf{u}_p$	Weight ω	Policy π_M
	p_0	p_1	p_2		$\mathbf{M}_p \cdot \mathbf{u}_p$	Softmax (ω)
p_0	0.5	1.0	0.3	0.8	1.52	0.5
p_1	0.0	0.5	0.7	0.7	0.53	0.17
p_2	0.7	0.3	0.5	0.6	1.01	0.33

rate of a small subset of pieces has a strictly dominant advantage against every opposing strategy, resulting in an optimal agent that is no different than a constant action agent. Extreme unbalance implies null diversity (no variety in pieces selected). We are looking for a middle ground, as discussed by Becker and Görlich [1], a scenario where small power gaps exist, allowing for a variety of meaningful strategic choices. We define this type of scenario as a soft unbalance. In a more formal view, a setting where there exists some intransitivity between strategies, where particular pieces are favorable against others, and in mixed Nash equilibrium (NE) (a joint strategy where players have no advantage to deviate alone from), a significant number of pieces have a reasonable chance of being picked [4]. To promote soft unbalance and disfavor random meta builders and constant-team builders alike while maintaining variety and meaningful choices, we create a subgroup of strategies that are closely balanced between themselves but that show some transitivity against the remaining group.

IV. TEAM BUILDER AGENTS

In this article, the battle policy π is set as a search algorithm based on Minimax from Lee and Togelius [33]. The advantage of such an agent is that it allows us to change the core game without needing to retrain the battle agent. The disadvantage is the computation time, slowing down battle simulations.

A. Team-Building Algorithms

Our first team builder agent is an extension of weighted probability team generator [10] (which we denominate Individual *Pokémon* Counter). It employs epistemic reasoning, maximizing its win rate while trying to remain unpredictable (see Table I, where p_0, p_1, and p_2 are individual *Pokémon*). The usage rates are weighted with the win rate table between pairs of *Pokémon*s. The final weights are used for a Softmax policy.

Using Individual *Pokémon* Counter as a conceptual base, we designed multiple improved variants. The first two team-building algorithms also assume that a *Pokémon* role toward victory only depends on its individual advantages against other *Pokémon* ignoring teammate synergies.

Maximize *Pokémon* Coverage (see Algorithm 1) tries at each *Pokémon* selection to cover *Pokémon* not yet covered by its currently selected ones. First, a coverage vector $\mathbf{c}$ is initialized. After selecting a *Pokémon*, $\mathbf{c}$ is decreased in the positions where the selected *Pokémon* has a good matchup against. When it selects the next *Pokémon*, it weights each *Pokémon* matchup vector (or metagame matrix column) by its respective position in the coverage vector. It is assumed that any *Pokémon* has an equal chance of being selected by opponents. Ratio ρ is a hyperparameter that controls the degree to which $\mathbf{c}$ is decreased,

Algorithm 1: Maximize *Pokémon* Coverage.

Result: Battle Team t
Data: Pokémon Roster R, Ratio $\rho \in [0, 1]$, Threshold $\tau \in [0, 1]$

Coverage $\mathbf{c} = \langle 1.0, \ldots, 1.0 \rangle$ with $\dim(\mathbf{c}) = |R|$
$t = \emptyset$
Meta-Game $\mathbf{M}_p \leftarrow$ pairwise win rates from R
for $|t| < 3$ **do**
  Meta Policy $\pi_M \leftarrow \frac{\mathbf{M}_p \cdot \mathbf{c}}{||\mathbf{M}_p \cdot \mathbf{c}||}$
  Sample Pokémon $p_i \leftarrow \pi_M$
  if $p_i \notin t$ **then**
    $t \leftarrow t \cup \{p_i\}$
    for $j < |R|$ **do**
      if $\mathbf{M}_p[i, j] \geq \tau$ **then**
        $\mathbf{c}[j] \leftarrow \rho \mathbf{c}[j]$
    end
end

Algorithm 2: Minimax Builder.

Result: Battle Team t
Data: Pokémon Roster R

$t \leftarrow \emptyset$
Meta-Game $\mathbf{M}_p \leftarrow$ Pairwise win rates from R
for $|t| < 3$ **do**
  Meta Policy $\pi_M \leftarrow$ Left side of NE joint strategy of $\mathbf{M}_p$
  Sample Pokémon $p \leftarrow \pi_M$
  if $p \notin t$ **then**
    $t \leftarrow t \cup \{p\}$
end

and the threshold τ determines which is the minimum win rate for strategies to be considered for a reduction in the coverage vector.

Minimax Builder (see Algorithm 2) assumes that opponents are playing rationally, and therefore, the NE of the normal-form metagame $\mathbf{M}_p$ is computed. Linear programming is used to find the NE solution efficiently. Minimax Builder (as Maximize *Pokémon* Coverage) has the advantage of being meta-independent, as it does not depend on past team selections.

Individual *Pokémon* Counter, Maximize *Pokémon* Coverage, and Minimax Builder do not exploit more intrinsic game properties like teammate synergies besides individual *Pokémon* pairwise matchup advantages. Our second set of algorithms considers full teams as pieces of the metagame. A metaheuristic search method, such as GA, is used to search for good solutions. An exhaustive search would result in $51 \times 50 \times 49 \sim 125$k for $|R| = 51$, while with GA in the worst scenario, we are able to find an optimal solution in eight solutions and 2k generations resulting in 16k lookups.

Meta Counter (see Algorithm 3) predicts possible strong meta teams T_M, and giving the usage rate over the column entries T_M maximizes the inner product between the solution team t_i metagame row entries with each team usage rate under entries

Algorithm 3: Meta Counter.

Result: Battle Team t
Data: Pokémon Roster R, Team History H, Fitness Function F_{t_1}, Size Threshold $h > 0$
if $|H| > h$ **then**
  Team Meta $\mathbf{u}_t \leftarrow \mathbf{u}_t[j]$ for each non-repeated permutation t_j of H
  Sample Meta Teams $T_M \leftarrow \text{Softmax}(\mathbf{u}_t)$
else
  Sample Meta Teams $T_M \leftarrow \text{MinimaxBuilder}(R)$
end
$t \leftarrow \text{GA.Optimize}(F_{t_1}(T_M))$

T_M

$$F_{t_1}(t_i)\{T_M\} = \sum_{t_j \in T_M} \mathbf{M}_t[i,j] \times \mathbf{u}_t[j]. \quad (1)$$

The usage rate of a team here does not really represent the number of occurrences of a certain permutation of three *Pokémon*, but instead pseudo-likelihood of it being selected (it does not correspond to a probability in the rigorous sense). The team t_i usage rate is estimated by the sum of the individual members' usage rates

$$\mathbf{u}_t[i] = \frac{1}{|t_i|} \sum_{p_j \in t_i} \mathbf{u}_p[j]. \quad (2)$$

The history H is a list of the past composed teams. The rationale is the following: If a team is strong in the current meta, then it has a greater likelihood of being picked, and so do its individual members. Their usage rate works like a signature, because if individuals have a higher rate, so do permutations containing them. Finally, we define how the meta teams we want to optimize against are elected. There are two cases. The teams of H are selected with a probability distribution given by the Softmax of $\mathbf{u}_t$. If there is not sufficient elements in H, then a meta-independent algorithm such as Minimax Builder is used during the first iterations of the team-building cycle to select the opponent meta teams.

Maximize Team Coverage (see Algorithm 4), instead of finding an overall counter team against all teams, searches separated optimal teams against each meta team. Then, it fills $\mathbf{M}_t$ with the pairwise win rates of all meta teams and counter teams and computes the NE solution as selection policy.

B. Deep Matchup Predictor

The win rates of the meta can be determined empirically by running an arbitrary number of gameplay between two teams, but such effort is computationally expensive, and that is one of the main motivations why the VGC AI Competition regulation determines a limited time for team building. Our proposal is to train a parameterized predictor function f_{θ_1} that, given two teams, should output the predicted win rate of each. Previous tries showed success in predicting the outcome using supervised learning in competitive games [34]. To train this model, we generate a dataset of random team pairs piloted by the battle

Algorithm 4: Maximize Team Coverage.

Result: Battle Team t
Data: Pokémon Roster R, Team History H, Fitness Function F_{t_1}, Size Threshold $h > 0$
if $|H| > h$ **then**
  Team Meta $\mathbf{u}_t \leftarrow \mathbf{u}_t[j]$ for each non-repeated permutation t_j of H
  Sample Meta Teams $T_M \leftarrow \text{Softmax}(\mathbf{u}_t)$
else
  Sample Meta Teams $T_M \leftarrow \text{MinimaxBuilder}(R)$
end
Counter Teams $T_C \leftarrow \emptyset$
for $t_m \in T_M$ **do**
  $t_c \leftarrow \text{GA.Optimize}(F_{t_1}(\{t_m\}))$
  $T_C \leftarrow T_C \cup \{t_c\}$
end
$T_M \leftarrow T_M \cup T_C$
Team Meta-Game $\mathbf{M}_t \leftarrow$ pairwise win rates from T_M
Team Meta Policy $\pi_{M_t} \leftarrow$ Left side of NE joint strategy of $\mathbf{M}_t$
Sample $t \leftarrow \pi_{M_t}$

policy π and determine their average win rate. For function f_{θ_1}, we input team pairs t_0 and t_1 and put as target the win rate $w \in [0, 1]$. The samples for f_{θ_1} can be augmented by knowing that if our estimation of the win rate of t_0 and t_1 is w, then the win rate of t_1 and t_0 is $1 - w$. For individual *Pokémon* battles, the same rationale is used, replacing a pair of teams with a pair of individual *Pokémon*. Naturally, a candidate has 50% win rate against itself, and such team pairs t_0 and t_1, where $t_0 = t_1$, were excluded from the training dataset. Since it is iterated over nonrepeating pairs of teams, the actual number of inference computations of f_{θ_1} is less than $(|T| - 1)^2/2$ to fill a metagame $\mathbf{M}_t$.

Previously, a pyramid topology yielded good results [35] for match prediction between champions on League of Legends, alternating dropout, normalization, and dense layers. Betley et al. [36] considered two approaches to train a regression model predicting the win rate of each deck for a Heartstone dataset and a classification model predicting the result (win, lose). They opted for the latter since the result dataset had 300k samples versus 400 samples for win rate. We follow the pyramid architecture for f_{θ_1} and generated a dataset of 1M match results (500k augmented to 1M). A binary classification model was opted where it outputs both predicted win rates for row and column players. A rectified linear unit (ReLU) [37] activation function in the hidden layers was trained with Adam optimizer [38] and used Kaiming Ha initialization [39] with a bias component of 0.01. Four feedforward hidden layers are used, where the first has double the input size, each next has half the size of the previous, and the output layer has double size. It was trained with a learning rate $\alpha = 10^{-3}$ and mini-batches of size 128 during 50 epochs. It achieved a 92.5% accuracy over the training set (20k samples) and an 85.4% accuracy over the test set (200k samples). We also trained an equivalent predictor only

for individual *Pokémon* matchups, where we adjusted to smaller network layers (one-third compared to the team prediction) and a dataset of only 100k samples. It achieved a 97.9% accuracy over the training set (20k samples) and an 87.9% accuracy over the test set (2k samples).

C. Deep Q-Mapping

For Maximize Team Coverage, we also experimented to train a generator model f_{θ_2} similar to GANs [40] to recommend potential counter *Pokémon* or teams. Our original aim was to (by supervision or reinforcement) make the network output be entire teams. Our results were positive for small quantities of features but did not scale up. Instead, we tested what we refer to as Q-mapping, where a Q-network is trained with a number of outputs equal to the *Pokémon* constructive elements. Instead of using the reward from a discriminator and random noise/trajectories as input, constructive elements of strong counter teams are used as targets with high Q-values Q and the *Pokémon* or team we want to defeat as input. We are under a Markov decision process of a single step, so we verify that in the Q-learning target equation

$$Q(s, a) = r \tag{3}$$

where the target is calculated simply using the immediate reward r, and therefore, there is no need for a target network, replay memory, or other state-of-the-art tricks. State s corresponds to the opponent strategy (*Pokémon* or Team) and actions a to the constructive elements.

The Q-map network will rank the most desired constructive elements. We can look up among the available *Pokémon* in the roster to quantify their overall quality in terms of constructive elements or narrow the search space of a GA with just the most high-ranked *Pokémon*. Our tested Q-map network's structure is similar to our predictor network, only a Sigmoid output activation function is used. We have two hidden layers of size 256. The output layer has 87 outputs, corresponding to the VGC AI Competition *Pokémon*'s constructing elements, the *Pokémon* types (18), and the standard moves from the VGC AI Competition (69). It was trained with a learning rate of $\alpha = 5^{-5}$ and mini-batches of size 128 during 15 epochs.

To generate a counter *Pokémon* against individual *Pokémon*, in our first experiments, the dataset of the predictor network was reused. We found that using random teams with variable Q-values (values equal to the win rates) was good, but could be improved. During these experiments, we found that some power transitivity between teams was present; the GA search against random teams revealed that using some weaker moves resulted in configurations with higher hit points (HP) *Pokémon* and could obtain better win rates. Therefore, for our definitive dataset, we used the original counters as input (the team already found by searching) and countered the counter teams with a second GA search run, so we motivated our outputs to be strong against already strong teams.

We evaluate the trained model against individual *Pokémon* by human observation and by comparing the strength to the GA outputs. It was observed that the most recommended types were usually types that resisted well against the opponent *Pokémon*, while the moves contain, for the most part, weak moves to increase the *Pokémon* HP. The offensive moves are usually ranked more sparse, but at least with one good decisive move to close matches. Then, it was compared if countering three individual *Pokémon* with the Q-map network yields close results to simply using a GA against an entire team. Over 100 trial runs, the GA achieved a 90% win rate against a target opposing team, while Q-map achieved 72%. The roster and opponents were the same for both algorithms.

The result is overall positive, overcoming the first limitation in a team builder policy as an NN; a fixed *Pokémon*/team can have multiple counters, so we output promising constructive pieces (Q-values). Like Q-DeckRec, a good future approach may be to generate a team iteratively or approaches such as those of Muise et al. [41], where we could decompose the single-step games into ordered subdecisions, where we take the best microaction at each step and use it as observation for the next step. We believe such effort could be achieved as well with time series using recursive NNs or long short-term memory networks [42].

TABLE II
TEAM BUILDER AGENTS VERSUS RANDOM AGENTS

	IndvCtr	PkmCvg	Minmax	MetaCtr	TeamCvg
Random Roster, $\|R\| = 10$, $N = 20$					
Elo	1238	1269	1270	1264	1225
Δt	4s	11s	12s	103s	130s
Random Roster, $\|R\| = 20$, $N = 40$					
Elo	1276	1241	1252	1320	1215
Δt	4s	13s	12s	265s	280s
Random Roster, $\|R\| = 50$, $N = 100$					
Elo	1276	1287	1234	1392	1244
Δt	32s	21s	96s	593s	704s
Unbalanced Roster, $\|R\| = 11$, $N = 20$					
Elo	1200	1201	1237	1277	1246
Δt	8s	17s	18s	99s	156s
Unbalanced Roster, $\|R\| = 21$, $N = 40$					
Elo	1174	1195	1145	1302	1234
Δt	25s	145s	41s	232s	288s
Unbalanced Roster, $\|R\| = 51$, $N = 100$					
Elo	1212	1193	1204	1373	1264
Δt	31s	36s	28s	614s	776s

D. Evaluation and Discussion

The strength of the team-building agents is assessed by facing them off against each other. To identify which algorithms are making intelligent decisions, they first face against a baseline random agent. Two scenarios were considered: the roster was randomly generated, and the roster was purposely mildly unbalanced (see Section V-C; this scenario should penalize the random agent). We use the Elo rating provided by the team-building layer of the VGC AI Framework to compare the agent's strength. The results are illustrated in Table II. In the random roster scenario, every agent outclassed the random agent, indicating they are making intelligent decisions. They are also able to scale with the size of the roster and the number of played rounds N. Note that the algorithms that search via GAs run much slower and that both Meta Counter and Maximize Team Coverage construct a

TABLE III
TEAM BUILDER AGENTS' RELATIVE STRENGTH TABLE

	IndvCtr	PkmCvg	Minmax	MetaCtr	TeamCvg
Non-Balanced/Random Roster, $\|R\| = 20$, $N = 40$					
IndvCtr	–	1200	1200	1138	1200
PkmCvg	1200	–	**1235**	1119	1153
Minmax	1200	1164	–	1132	**1268**
MetaCtr	**1261**	**1280**	**1267**	–	1200
TeamCvg	1200	**1246**	1131	1200	–
Non-Balanced/Random Roster, $\|R\| = 50$, $N = 100$					
IndvCtr	–	1200	**1306**	1073	1164
PkmCvg	1200	–	1200	1064	1161
Minmax	1093	1200	–	1089	1151
MetaCtr	**1326**	**1335**	**1310**	–	**1225**
TeamCvg	**1235**	**1238**	**1248**	1174	–

The bold values correspond to the winning matchups, where the Elo is bigger than 1200.

TABLE IV
VGC AI TEAM-BUILDING TRACK COMPETITION RESULTS

	$\|R\| = 20$, $N = 40$	$\|R\| = 50$, $N = 100$
Random	1098 (sixth)	1109 (sixth)
IndvCtr	1214	1175
PkmCvg	1218	1188
Minmax	1158	1158
MetaCtr	**1325 (first)**	**1344 (first)**
TeamCvg	1183	1222

The bold values correspond to the winning algorithm, the one with the largest Elo.

team metagame table, and we configured to counter the 10 top meta teams as hyperparameter.

Next, the relative performance between the multiple algorithms is evaluated to evidence eventual (in)transitivity. The results are illustrated in Table III. We bold the winners and underlined the draws. At least five runs for each pair were made. Meta Counter outclasses all others. This goes against our initial expectations where Maximize Team Coverage would be the strongest as it adapts to every meta team and reasons about its choices, but Meta Counter shows to be powerful enough to find teams that can beat all meta teams.

Finally, a VGC AI Championship Track is simulated where every agent competes in the same ecosystem. The results for this final validation are illustrated in Table IV. We obtained positive results, as the random agent falls in the last place in most experiments and Meta Counter in first.

During the tests, great volatility was observed; it is possible for some of the algorithms to lose even to the random agent. This is due to the stochastic nature of the game, and the agents are sacrificing some wins to become unpredictable.

V. METAGAME AUTOBALANCING AGENT

The balance agent will adapt the metagame matrix by changing *Pokémon* attributes to incentivize a certain natural meta by the Team Builder agents (see Fig. 1 for an illustrated pipeline). First, we explore some previous and new metagame balance fitness functions and target distributions of usage under different scenarios.

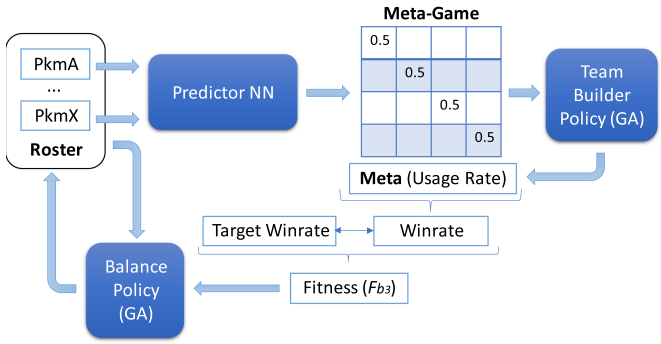

Fig. 1. Pipeline of the adversarial model. A predictor network fills the Metagame matrix with the win rate for each pair of *Pokémon* (or Teams). The Team Builder Policy will generate a distribution of *Pokémon* or Team usage, the Meta. Metagame and Meta can be used to get the effective win rate of each *Pokémon* or Team and the fitness stimulates a desired win rate.

A. Metagame Balance Fitness Function

To explore good fitness functions, we first studied simplified scenarios over the *Pokémon* Battle Environment [43], where *Pokémon* are composed of only one damaging move and, therefore, are only formed by five attributes: HP, type, speed, move's power, and move type. Therefore, we eliminate the dependence of a battle policy and study a more idyllic scenario (akin to an auto battler game). Also, offensive coverage gets diminished, and we can more easily observe uneven advantages between *Pokémon*s. There are two game variants: one with a deterministic outcome and a second where an accuracy parameter was conserved to turn the game stochastic.

At each experiment, different combinations of GA algorithm operators were carried out, but our discussion will focus on the design of the fitness function as we explore different formal definitions of metagame balance. In addition, in each experiment, we tried to escalate the difficulty of our problem by incrementing the number of generated *Pokémon*.

1) Experiment 1. Single-Handicap Nash Policy: In this experiment, the aim was to maximize metagame balance by forcing an NE $= \langle \pi_{M_1}, \pi_{M_2} \rangle$, to give incentive to every *Pokémon* to be selected by modifying the *Pokémon* move power as handicap. Chromosomes are a vector h of handicaps, one for each *Pokémon*. The desired optimal policy $\pi^*_M = \langle 1/|R|, \ldots, 1/|R| \rangle$ of the game is the one that all *Pokémon* have an equal chance to be selected. We want to minimize the distance of the current policy π_M with the target policy

$$F_{b_1}(\pi_M)\{\pi^*_M\} = -||\pi_M - \pi^*_M||. \tag{4}$$

2) Experiment 2. Single-Handicap Matchwise Win Rate: In our second approach, we remove the necessity of computing NE instead of incentivizing $\mathbf{M}$ (the meta-game or win rate table) to contain payoffs that will lead to pairwise balanced matches. As such our new fitness is based on measuring the distance between the desired $\mathbf{M}'$ with $\mathbf{M}$

$$F_{b_2}(\mathbf{M})\{\mathbf{M}'\} = -||\mathbf{M} - \mathbf{M}'||. \tag{5}$$

This was the approach followed to balance Hearthstone by de Mesentier Silva et al. [4], with the main difference being that

they operate over a set of a limited number of decks, and some cards are shared between those decks.

3) Experiment 3. Single-Handicap Pokémon *Average Win Rate:* Instead of considering balanced *Pokémon* as ones that draw against every opponent, our constraints are alleviated where each *Pokémon* just needs to have similar chances to be selected, by having a similar average win rate $E(\mathbf{M})$. The probability is given by each entry of the player meta policy π_M. The usage rate of a *Pokémon* is given by the player policy, but instead of computing the optimal policy by finding the NE, we assume that this one is fixed (the desired policy), and it is used as weights to incentivize $E(\mathbf{M}) = \mathbf{M} \cdot \pi_M$ to have payoffs that lead to a desired average win rate array $\mathbf{w} = \langle 0.5, \ldots, 0.5 \rangle$, translating to the fitness function

$$F_{b_3}(\mathbf{M}, \pi_M)\{\mathbf{w}\} = -||\mathbf{M} \cdot \pi_M - \mathbf{w}||. \quad (6)$$

4) Experiment 4. Multihandicap Pokémon *Average Win Rate:* This experiment is similar to the previous one, with the only difference being that our handicap adjustment is done by changing the *Pokémon* actual numerical attributes instead of a single-handicap coefficient. Concretely, the *Pokémon*'s numerical HP, move power, and speed attributes were changed.

5) Experiment 5. Balance + Pokémon *Similarity Single-Objective Optimization:* de Mesentier Silva et al. [4] explored minimizing the number of changes as a secondary objective of optimization in Heartstone. We also consider this aspect important because the game designers have a defined vision of the attributes' semantics and what they represent for the players, so in an application scenario, we want to minimize the number of changes of *Pokémon* in addition to maximizing the balance. In this experiment, the previous fitness is still considered, but combined with a weighted similarity fitness between two *Pokémon* $F_s = ||p_1 - p_2||$. We denote the weights of both fitness functions as δ_1 and δ_2

$$F_{b_4}(\mathbf{M}, \pi_M)\{\mathbf{w}\} = \delta_1 F_{b_3} - \delta_2 F_s. \quad (7)$$

We first perform single-objective optimization for F_{b_3}, and then, we use the solutions as the initial population solution for F_{b_4}. We consider that the balance of the game is more important than the similarity between *Pokémon*, as such we set $\delta_1 = 0.6$ and $\delta_2 = 0.4$.

6) Experiment 6. Balance + Pokémon *Similarity Multiobjective Optimization:* Instead of fixing a single-point solution, in this experiment, we used the multiobjective optimization algorithm NSGA2 [44] to find the Pareto front of our two simultaneous optimization objectives: Balance F_{b_3} and similarity F_s. Like the previous experiment, single-objective optimization was first performed just for F_{b_3}, and then, the found solutions are used as the initial solution for NSGA2. For an application setting, the game designers would have to opt for their preferred tradeoff between the two fitness functions.

7) Fitness Function Study Results: The main results of the preliminary experiments are illustrated in Table V.

The results for Experiment 1 were good since we obtained fitness function values near zero, which means that each *Pokémon* has an equal probability chance to be selected.

TABLE V
METAGAME BALANCE FITNESS FUNCTION COMPARISON STUDY RESULTS (SINGLE SIMPLIFIED *POKÉMON*)

	GA Parameters							
$\|R\|$	Gen	Pop	Sel	Cross	Mut	%	Fit	Δt (s)
Exp 1: Single-handicap Nash Policy								
5	500	8	rws	tp	rand	5	−0.01	11
11	500	8	rws	tp	rand	5	−0.27	12
21	500	8	rws	tp	rand	5	−0.24	35
Exp 2: Single- handicap Matchwise Win Rate								
5	500	8	tor	tp	rand	5	−2	0.46
11	500	8	tor	tp	rand	5	−5	1.23
21	2000	8	tor	tp	rand	5	−10	19
Exp 3: Single-handicap Pokémon Average Win Rate								
5	2000	8	tor	tp	rand	5	0.00	2.23
11	2000	8	tor	tp	rand	5	−0.55	2.48
21	2000	8	rws	tp	rand	5	−1.52	23
Exp 4: Multi-handicap Pokémon Average Win Rate								
5	1000	8	tor	tp	rand	5	0.00	1.12
11	1000	8	tor	tp	rand	5	−0.36	3.6
21	2000	8	rws	tp	rand	10	−1.61	180
Exp 5: Balance + Similarity Single-Objective Optimization								
5	1000	8	tor	tp	rand	5	−0.19	1.82
11	2000	8	tor	tp	rand	5	−1.37	9
21	7500	8	rws	tp	rand	15	−1.64	101

TABLE VI
EXPERIMENT 4.5: MULTIHANDICAP *POKÉMON* AVERAGE WIN RATE RESULTS WITH ACCURACY—STOCHASTIC VERSION WITH G SIMULATED GAMES

		GA Parameters							
$\|R\|$	G	Gen	Pop	Sel	Cross	Mut	%	F_{b_3}	t (s)
5	20	7.5k	16	tor	tp	rand	5	−0.52	72
11	30	7.5k	16	tor	tp	rand	5	−1.31	426
21	10	7.5k	8	tor	tp	rand	5	−3.32	331

For Experiment 2, we were unable to find good solutions. The problem with this approach is that such exact solutions for each entry of $\mathbf{M}$ may not exist or are very hard to find. In addition, this approach is of little interest for application as it makes the strategic choice irrelevant (perfectly balanced).

For Experiment 3, good solutions were found. When $|R| = 21$, we changed tournament selection to roulette wheel selection. Overall, it was an efficient approach not only in terms of quality but also in terms of execution time.

In Experiment 4, for $|R| = 21$, good solutions were also found to change different *Pokémon* numerical attributes and to achieve almost balanced games. As the number of *Pokémon* is increased, the number of generations for the GA also needs to be increased. For $|R| = 21$, changing tournament selection to roulette wheel selection yielded better results. For the second variant, i.e., the stochastic version (see Table VI), we observed that increasing the population number from 8 to 16 contributed positively; however, increasing the number of generations did not seem to have any effect. We suspect this may be due to the imprecision of the win rate values.

For Experiment 5, for $|R| = 21$, roulette wheel selection and 15% of mutation provided the best result.

For Experiment 6, we executed three runs of the algorithm for $|R| = 5$, $|R| = 11$, and $|R| = 21$, with 7500 generations, tournament selection, two-point crossover, and polynomial mutation. We concluded that the algorithm is able to find tradeoffs

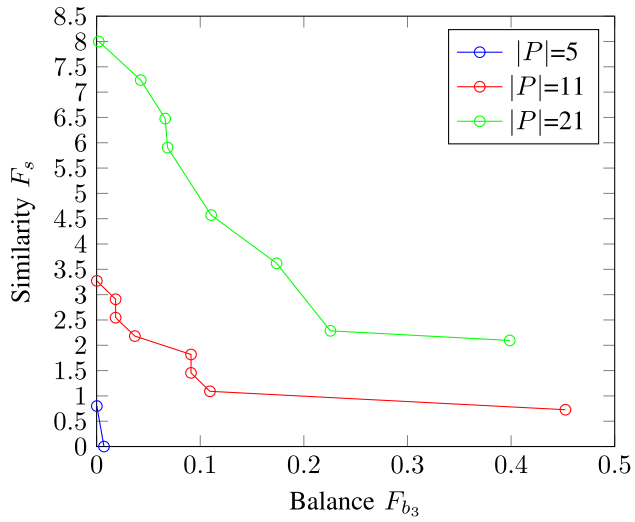

Fig. 2. Exp6: NSGA2 multiobjective optimization Pareto front of balance and similarity maximization multiobjective.

that maximize each individual metric, but can also find middle terms. The main results are listed in Fig. 2.

B. Team Balance Algorithm

For balancing teams, we 1) search for the most relevant team candidates to balance and 2) set the maximization of one of the studied fitness functions as an objective. We consider meta relevance as the summation of the average win rate of teams where the individual *Pokémon* is present. However, if only strong team builder agents compete in the ecosystem, we just need to measure the usage rates instead of accumulating win rates because strong team builders will narrow their choices to counter meta strategies.

The main issue with the algorithm is that simulating the creation of a new meta at each GA generation would be very computationally expensive, because our team builder algorithms scale poorly (multiple team-building GA search runs are done inside of a balance GA search run).

Instead, we propose a two-stage loop. First, strong team builders compete over several rounds, which should narrow the meta to the strongest *Pokémon*. Then, the most relevant teams are selected to be part of the team submetagame we balance. Some *Pokémon* may have less usage rate or none at all, not becoming targets of adjustment; therefore: 1) we preselect target teams containing all individual *Pokémon* among them, augmenting our target teams with the current unbalanced meta teams, and we soft unbalance toward turning our target diverse teams stronger; and 2) random teams can be generated with *Pokémon* with few usages in the meta teams, and we balance out every candidate so both meta teams with low diversity, and the augmented teams with unused *Pokémon* can balance out with similar strength (see Algorithm 5).

Algorithm 5: *Pokémon* Team Balance.

Result: Balanced Pokémon Roster R'
Data: Pokémon Roster R, Balance Iterations N, Gameplay Iterations L, Arbitrary Balance Fitness Function F_b, Team Build Policy π_{M_t}, target win rate $\mathbf{w}$

Iteration $i \leftarrow 0$
for $i < N$ **do**
  History $H \leftarrow \emptyset$
  Usage Rate $\mathbf{u}_t \leftarrow \langle 0.0, \ldots, 0.0 \rangle$
  Iteration $j \leftarrow 0$
  for $j < L$ **do**
    Sample Team $t \leftarrow \pi_{M_t}(R, H)$
    $H \leftarrow H \cup \{t\}$
  end
  if $|H| > h$ **then**
    Team Meta $\mathbf{u}_t \leftarrow \mathbf{u}_t[j]$ for each non-repeated permutation t_j of H
    Sample Meta Teams $T_M \leftarrow$ Softmax($\mathbf{u}_t$)
  else
    Sample Meta Teams $T_M \leftarrow$ MinimaxBuilder(R)
  end
  Augment T_M with random teams t_k with low usage rate $\mathbf{u}_t[k]$
  Team Meta-Game $\mathbf{M}_t \leftarrow$ pairwise win rates from T_M
  /* Pokémon Attributes */
  $R' \leftarrow$ GA.Optimize($F_b(\mathbf{w})$)
end

C. Evaluation and Discussion

Now, using *Pokémon* with four moves, we evaluate two different balance target variants: perfect balance and soft unbalance. The fitness function used is F_{b_3} or F_{b_4} depending on the use case. Balance agents are allowed to select the type and one of 69 standard moves for each *Pokémon*, and as such, the chromosomes will represent these constructive pieces. What we soon noticed was that without any kind of restrictions, randomly generated rosters are generally well balanced. For soft unbalance, we promote one-third of the roster with a desired average win rate array $\mathbf{w} = \langle 0.7, \ldots, 0.7, 0.3, \ldots, 0.3 \rangle$ (for example). This will guarantee that a subset of the pool of *Pokémon* is viable. Under soft unbalanced rosters, we also demonstrate that the proposed team builder agents will indeed shift to selecting the strongest *Pokémon* and filling our history H correctly. We also test whether only two moves and the type of *Pokémon* can be changed.

The results are illustrated in Table VII. In both perfect balance scenarios, the win rates are between $[0.48, 0.53]$ for the predicted metagame matrix $\mathbf{M}$ and $[0.27, 0.63]$ for the actual simulations, showing the error caused by the predictor network, and also, these values do not differ much from a randomly generated roster. For the soft unbalance scenario, win rates are between $[0.78, 0.94]$ for overpowered *Pokémon* and $[0.08, 0.68]$ for underpowered *Pokémon*. The large negative value of the fitness only translates the distance to the target of the underpowered, which is 0.35 and not 0.0.

For the similarity test alone, only with 2K generations of the GA, we obtained *Pokémon* similar (type, HP, and moves) to the originals when $|R| = 11$ and 5K generations when $|R| = 51$.

TABLE VII
METAGAME BALANCE TARGET COMPARISON STUDY (SINGLE COMPLETE *POKÉMON*)

	GA Parameters							
$\|R\|$	Gen	Pop	Sel	Cross	Mut	#	Fit	Fit*
Perfect Balance								
11	500	8	tor	two-point	rand	1	−0.06	−0.78
51	500	8	tor	two-point	rand	1	−0.82	−3.92
Soft Unbalance								
11	500	8	tor	two-point	rand	1	−3.31	−2.47
51	150	8	tor	two-point	rand	1	−14.17	−11.5
Soft Unbalance + Restricted								
11	500	8	tor	two-point	rand	1	−3.32	−2.18
51	100	8	tor	two-point	rand	1	−14.62	−11.39
Similarity								
11	2000	8	tor	two-point	rand	1	−0.09	–
51	5000	8	tor	two-point	rand	1	−0.18	–
Perfect Balance + Similarity								
11	1000	8	tor	two-point	rand	1	−0.64	−0.95
51	100	8	tor	two-point	rand	1	−2.40	−4.59
Soft Unbalance + Similarity								
11	1000	8	tor	two-point	rand	1	−1.82	−2.31
51	100	8	tor	two-point	rand	1	−8.44	−10.95
Perfect Balance + Restricted + Similarity								
11	200	8	tor	two-point	rand	1	−0.35	−0.75
51	100	8	tor	two-point	rand	1	−1.69	−3.73

TABLE VIII
VGC AI METADATA ANALYSIS AFTER 400 RUNS

	Usage Rate (%)								
Pokémon	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	...
Random	4.8	4.3	4.7	4.8	5.6	4.6	4.7	4.2	...
IndvCtr	4.2	5.2	5.1	4.7	4.6	4.9	5.3	3.9	...
PkmCvg	4.3	5.2	4.6	5.2	6.0	5.7	5.6	3.8	...
Minmax	2.2	4.1	4.5	2.7	6.9	4.0	2.9	6.2	...
MetaCtr	**10.2**	2.9	**18.8**	**9.3**	4.6	2.9	**18.4**	2.5	...
TeamCvg	7.5	4.9	6.6	5.2	5.2	7.4	6.7	4.1	...

The bold values illustrate Pokémon with a very high usage rate compared with the mean usage rate.

For the perfect balance + similarity testing a pair of weights $\delta_1, \delta_2 = 0.5, 0.5$ and $\delta_1, \delta_2 = 0.1, 0.9$, HPs are unchanged as are some types, but in terms of overall balance, the results are similar. Soft unbalance + Similarity presented the best results, where only one or two moves were changed, achieving our main aim of individual *Pokémon* balance. Restricted variants did not present any improvements.

Before analyzing the team balance algorithm, we must validate if our team-building algorithms narrow their selections to the most powerful *Pokémon* in a soft unbalanced meta. We evaluate their performance against random agents and check if they achieve a larger Elo compared to a more balanced meta (see Table II), but we did not observe any major difference by comparison. Then, we analyzed the metadata and checked if the usage rate is proportional to the average win rate of the *Pokémon*. We illustrate the results just for the smallest scenario in Table VIII. In a scenario of 21 *Pokémon* where seven were made more powerful, we can observe that the Meta Counter agent selected some *Pokémon* with almost four times (18.8%) the average usage rate (5%), evidencing its highest Elo rating in the previous experiments. Therefore, we elected Meta Counter to be used by the team balance adversarial process.

TABLE IX
POKÉMON TEAMS META-BALANCE RESULTS

$\|R\|$	6	11	21	51
Balance nondiverse with unused				
Fit	−0.30	−0.20	−0.18	−0.20
Pre Cnt	3	4	4	17
Post Cnt	3	5	8	16
Soft Unbalance to target diverse teams				
Fit	–	−1.14	−1.16	−1.05
Pre Cnt	–	3	8	13
Post Cnt	–	5	7	12

First, the balance of the *Pokémon* Teams was explored considering that only a fixed finite number of teams exists, and we can only just select those teams. Our results were very similar to balancing individual *Pokémon*, with the only major notable difference being that randomly generated teams tend to be quite unbalanced. As the result table was very similar to Table VII, we omit it.

Next, we balance a nonenumerable quantity of teams, maximizing the diversity in terms of the number of usage rates of individual *Pokémon*. The results of Algorithm 5 are illustrated in Table IX. We count pre and postbalance. Meta Counter narrows once again to a small group of used *Pokémon*. Most selected *Pokémon* change after balancing, but their overall quantity only increases marginally in the best case. We hypothesize that this could be a limitation of Meta Counter, or that a balance threshold may exist, or that more consistent results would require full searching by generating a new meta at each GA generation moving toward a more precise search direction. The latter requires fast team-building inference algorithms such as deep Q-mapping.

VI. CONCLUSION

In this article, we presented an initial takeaway to the VGC AI Competition, further formalizing metagame roster balance and a set of hybrid tabular deep solutions for both team building and roster balance, where we learn a pair of deep models to help the search for counterstrategies against a *Pokémon* metagame, and the adaptation of a *Pokémon* roster. We believe this work enhanced our understanding of the metagame autobalance paradigm. The studied techniques and the overall conceptual framework could be expanded to other game domains and inspire similar AI competitions. The current solutions will serve as new baselines for future editions in the VGC AI Competition, hoping to incentivize more participation.

SOURCE CODE

The source is publicly available.[1] We implemented our GA solutions using the library PyGAD [45] and the pymoo library [46] to implement the NSGA2 algorithm. The deep models were implemented and trained with PyTorch [47].

[1][Online]. Available: https://gitlab.com/DracoStriker/vgc-agents

REFERENCES

[1] A. Becker and D. Görlich, "What is game balancing?—An examination of concepts," *ParadigmPlus*, vol. 1, pp. 22–41, 2020.

[2] M. Carter, M. Gibbs, and M. Harrop, "Metagames, paragames and orthogames: A new vocabulary," in *Proc. Int. Conf. Found. Digit. Games*, 2012, pp. 11–17, doi: 10.1145/2282338.2282346.

[3] S. Reis, L. P. Reis, and N. Lau, "VGC AI competition—A new model of meta-game balance AI competition," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[4] F. de Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, "Evolving the hearthstone meta," in *Proc. IEEE Conf. Games*, 2019, pp. 1–8.

[5] Z. Chen, C. Amato, T.-H. Nguyen, S. Cooper, Y. Sun, and M. S. El-Nasr, "Q-DeckRec: A fast deck recommendation system for collectible card games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[6] H. N. Ward, D. J. Brooks, D. Troha, B. Mills, and A. S. Khakhalin, "AI solutions for drafting in magic: The gathering," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[7] T. Bertram, J. Fürnkranz, and M. Müller, "Predicting human card selection in magic: The gathering with contextual preference ranking," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[8] J. Kowalski and R. Miernik, "Evolutionary approach to collectible arena deckbuilding using active card game genes," *IEEE Congr. Evol. Computation*, pp. 1–8, 2020, doi: 10.1109/CEC48606.2020.9185755.

[9] A. Bhatt et al., "Exploring the hearthstone deck space," in *Proc. 13th Int. Conf. Found. Digit. Games*, 2018, pp. 1–10, doi: 10.1145/3235765.3235791.

[10] D. Crane, Z. Holmes, T. T. Kosiara, M. Nickels, and M. Spradling, "Team counter-selection games," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[11] S. da Silva Oliveira, G. E. P. L. Silva, A. C. Gorgônio, C. A. S. Barreto, A. M. P. Canuto, and B. M. Carvalho, "Team recommendation for the Pokémon Go game using optimization approaches," in *Proc. 19th Braz. Symp. Comput. Games Digit. Entertainment*, 2020, pp. 163–170.

[12] G. Andrade, G. Ramalho, A. Gomes, and V. Corruble, "Dynamic game balancing: An evaluation of user satisfaction," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2006, pp. 3–8.

[13] R. Leigh, J. Schonfeld, and S. J. Louis, "Using coevolution to understand and validate game balance in continuous games," in *Proc. 10th Annu. Conf. Genet. Evol. Comput.*, 2008, pp. 1563–1570, doi: 10.1145/1389095.1389394.

[14] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic, "Evaluating competitive game balance with restricted play," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, 2021, vol. 8, no. 1, pp. 26–31.

[15] D. Delaurentis et al., "Toward automated game balance: A systematic engineering design approach," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4768–4777.

[17] M. Moroşan and R. Poli, "Automated game balancing in Ms PacMan and StarCraft using evolutionary algorithms," in *Proc. Eur. Conf. Appl. Evolut. Comput.*, 2017, pp. 377–392.

[18] P. Beau and S. Bakkes, "Automated game balancing of asymmetric video games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.

[19] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 269–276, doi: 10.1145/2908812.2908913.

[20] A. Hansen, A. Cardona, J. Togelius, and M. Friberger, "Open trumps, a data game," in *Proc. 9th Int. Conf. Found. Digit. Games, Soc. Adv. Sci. Digit. Games*, 2014.

[21] M. Beyer et al., "An integrated process for game balancing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.

[22] M. Morosan and R. Poli, "Lessons from testing an evolutionary automated game balancer in industry," in *Proc. IEEE Games, Entertainment, Media Conf.*, 2018, pp. 263–270.

[23] N. Tomasev, U. Paquet, D. Hassabis, and V. Kramnik, "Assessing game balance with Alphazero: Exploring alternative rule sets in chess," 2020, *arXiv:2009.04374*.

[24] J. Pfau, A. Liapis, G. Volkmar, G. N. Yannakakis, and R. Malaka, "Dungeons & replicants: Automated game balancing via deep player behavior modeling," in *Proc. IEEE Conf. Games*, 2020, pp. 431–438.

[25] W. Kavanagh and A. Miller, "Chained strategy generation: A technique for balancing multiplayer games using model checking," in *Proc. 26th Autom. Reason. Workshop*, 2019, pp. 15–16.

[26] W. Wallis, W. Kavanagh, A. Miller, and T. Storer, "Designing a mobile game to generate player data—Lessons learned," in *Proc. 21st Int. Conf. Intell. Games Simul.*, 2021, pp. 13–15.

[27] W. Kavanagh, A. Miller, G. Norman, and O. Andrei, "Balancing turn-based games with chained strategy generation," *IEEE Trans. Games*, vol. 13, no. 2, pp. 113–122, Jun. 2021.

[28] W. Kavanagh and A. Miller, "Gameplay analysis of multiplayer games with verified action-costs," *Comput. Games J.*, vol. 10, no. 1, pp. 89–110, 2021, doi: 10.1007/s40869-020-00121-5.

[29] A. J. Liu and S. Marschner, "Balancing zero-sum games with one variable per strategy," in *Proc. 13th AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2017, pp. 57–65.

[30] D. Hernandez, C. T. Toyin Gbadamosi, J. Goodman, and J. A. Walker, "Metagame autobalancing for competitive multiplayer games," in *Proc. IEEE Conf. Games*, 2020, pp. 275–282.

[31] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *Proc. IEEE Congr. Evol. Comput.*, 2012, pp. 1–8.

[32] M. C. Fontaine et al., "Mapping hearthstone deck spaces through map-elites with sliding boundaries," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 161–169, doi: 10.1145/3321707.3321794.

[33] S. Lee and J. Togelius, "Showdown AI competition," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 191–198.

[34] R. Argue, "Supervised learning as a tool for metagame analysis," 2014. [Online]. Available: https://www.cs.umd.edu/sites/default/files/scholarly_papers/Argue.pdf

[35] T. D. Do, S. I. Wang, D. S. Yu, M. G. McMillian, and R. P. McMahan, "Using machine learning to predict game outcomes based on player-champion experience in league of legends," in *Proc. 16th Int. Conf. Found. Digit. Games*, 2021, pp. 1–5.

[36] J. Betley, A. Sztyber, and A. Witkowski, "Predicting winrate of hearthstone decks using their archetypes," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2018, pp. 193–196.

[37] A. F. Agarap, "Deep learning using rectified linear units (RELU)," 2018, *arXiv:1803.08375*.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.

[40] I. Goodfellow et al., "Generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[41] C. Muise, F. Dignum, P. Felli, T. Miller, A. R. Pearce, and L. Sonenberg, "Towards team formation via automated planning," in *Proc. Int. Workshop Coordination, Org., Inst., Norms Agent Syst.*, 2016, pp. 282–299, doi: 10.1007/978-3-319-426914_16.

[42] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. D: Nonlinear Phenomena*, vol. 404, 2020, Art. no. 132306, doi: 10.1016/j.physd.2019.132306.

[43] D. Simões, S. Reis, N. Lau, and L. P. Reis, "Competitive deep reinforcement learning over a Pokémon battling simulator," in *Proc. IEEE Int. Conf. Auton. Robot Syst. Competitions*, 2020, pp. 40–45.

[44] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[45] A. F. Gad, "PyGAD: An intuitive genetic algorithm python library," 2021, *arXiv:2106.06158*.

[46] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[47] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.