FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Artificial Intelligence Methods for Automated Difficulty and Power Balance in Games

Simão Paulo Rato Alves Reis

Doctoral Program in Computer Science of the Universities of Minho, Aveiro and Porto

First Supervisor: Luís Paulo Gonçalves dos Reis

Second Supervisor: José Nuno Panelas Nunes Lau

January 11, 2024

Artificial Intelligence Methods for Automated Difficulty and Power Balance in Games

Simão Paulo Rato Alves Reis

Thesis submitted to Faculty of Sciences of the University of Porto for the Doctor Degree in Computer Science within the Joint Doctoral Program in Computer Science of the Universities of Minho, Aveiro and Porto

Approved by the jury:

President: Carlos Miguel Ferraz Baquero-Moreno
Referee: João Alberto Fabro
Referee: Rui Filipe Fernandes Prada
Referee: Pétia Georgieva Georgieva
Referee: Luís Paulo Gonçalves dos Reis
Referee: Henrique Daniel de Avelar Lopes Cardoso

January 11, 2024

Resumo

Esta tese estuda o problema de equilíbrio no desenvolvimento de jogos, nomeadamente de jogos para dois jogadores. Equilíbrio é considerado uma das propriedades mais importantes em jogos pelos jogadores, mas das mais difíceis de afinar pelos desenvolvedores de jogos. Quando jogadores sentem que a dificuldade não se adequa ao esperado, ou que existem poucas decisões relevantes no jogo, isto causa a perda de entusiasmo e abandono. Portanto, existe a necessidade de facilitar esta tarefa aos desenvolvedores de jogo.

Pretende-se investigar a viabilidade da Inteligência Artificial (IA) como ferramenta auxiliar para corrigir propriedades de jogos. Dividimos esta investigação em duas linhas: Equilíbrio de Poder, onde o objetivo é ajustar as estratégias do jogo para que várias se tornem efetivas, mas não serem estritamente dominantes; Equilíbrio de Dificuldade, onde o objetivo é ajustar propriedades de jogos em tempo real para que jogadores mais fracos ou em desvantagem possam competir contra jogadores mais fortes, ou jogadores em vantagem. Ambos os domínios exigem afinações no jogo, mas diferem no tempo e no seu objetivo, um, lida com o desequilíbrio no desenho de jogos, enquanto o outro, lida com a desigualdade nas habilidades dos jogadores.

Para o Equilíbrio de Poder, a metodologia passou por definir um ecossistema completo de equilíbrio de meta-jogos baseado na série de vídeo jogos Pokémon e construir uma competição de IA onde as múltiplas tarefas associadas (batalha, previsão, construção de equipas e equilíbrio do meta-jogo) estão presentes e podem ser testados num domínio comum. Para equilibrar o meta-jogo, seguimos um modelo adversário onde os construtores de equipas pretendem restringir-se ao uso de Pokémon ótimos, enquanto os agentes equilibradores incentivam o máximo possível de Pokémon distintos a serem escolhidos pelos construtores de equipa. Isto resulta em agentes capazes de jogar, construindo equipas eficazes e afinar a lista de Pokémon ao longo do tempo. Discutimos como os nossos modelos podem ser estendidos noutros domínios de vídeo jogos.

Para o Equilíbrio de Dificuldade, propõe-se uma estrutura de Ajuste de Dificuldade Dinâmico Multijogador onde um agente Mestre de Jogo (MJ) é treinado e inserido num jogo, e dependendo do estado do jogo implementa mecanismos de *handicap*. O regime de treino segue uma ordem específica. Para generalizar situações de vantagem, perturbações parametrizadas nas ações de um agente de referência são usadas para simular vários graus de habilidade no jogo, e a vantagem de cada jogador é usada para analisar traços de vantagem, estas avaliadas para recompensar o MJ. Isto resulta na capacidade do MJ de otimizar um conjunto de critérios de desenho de jogo e criar oportunidades para o jogador atrás de recuperar.

Mostramos que existem ferramentas de IA adequadas para cada tarefa, e é razoável pensar em equilíbrio de poder e dificuldade como problemas separados, mas onde ambos podem ser assistidos automaticamente e facilitados, e ambos aumentam a nossa compreensão do campo de equilíbrio automatizado de jogos.

Abstract

This thesis studies the balance problem in game development, notably in two-player games. Balance is considered one of the most important properties of a game by players, but it is one of the most difficult to tune by game developers. When players feel that the difficulty does not match what is expected, or that there are few relevant decisions in the game, it causes a loss of enthusiasm and abandonment. Therefore, there is a need to make this task easier for game developers.

We aim to investigate the viability of Artificial Intelligence (AI) as an assisting tool to fix game properties. We split our research into two paths: Power Balance, where the goal is to adjust the game strategies so that several become effective but are not strictly dominant; Difficulty Balance, where the objective is to adjust game attributes on the fly so that weaker players or players at a disadvantage can compete with stronger players or players in advantage. Both domains require tuning the game, but they mainly differ in the timing and in their aim, one deals with the imbalance in game design, while the other deals with inequality in player skills.

For Power Balance, our methodology was to define a full meta-game balance ecosystem based on the Pokémon video game series and develop an AI competition where the multiple associated tasks (battling, team prediction and assembly, and meta-game balance) are present and can be tested in a common ground. To balance the meta-game, we follow an adversarial model where team builders aim to narrow the use of optimal Pokémon while balancer agents aim to maximize the diversity of Pokémon to be selected by team builders. This results in agents being able to play, build effective teams, and being able to tune the Pokémon roster over time. We discuss how our models can be extended to other video game domains.

For Difficulty Balance, we propose a Multiplayer Dynamic Difficulty Adjustment framework where a Game Master (GM) agent is trained and embedded into a game, depending on the game state it will deploy handicap mechanisms. The training regime follows a specific pipeline. To generalize advantage situations, parameterized perturbations on the actions of a reference player are used to emulate several degrees of playing skill, and the advantage for each player is used to analyze advantage traces which are evaluated as a reward for the GM. This results in the GM being able to optimize game design criteria and create opportunities for the player behind to recover.

We show there are suited AI tools for each task, and it is reasonable to think of power balance and difficulty as separate problems, where both can be automatically assisted and eased. Both further augment our overall comprehension of the automated game balance field.

Agradecimentos

O percurso do doutoramento foi desafiante, e o qual só consegui superar graças ao apoio dos meus mais próximos, que quero aqui agradecer a nível pessoal:

- Ao meu pai, José, que faleceu no início deste meu percurso, mas foi graças ao dele que tive a possibilidade de seguir os meus estudos superiores, e que me motivou fortemente a prosseguir o caminho longo que foi o doutoramento.
- À minha mãe, Raquel, que tal como o meu pai e eu também enfrentou um caminho difícil de doutoramento e inspirou-me em ir em frente no doutoramento.
- Ao meu companheiro, Miguel por estar sempre ao meu lado e ajudando-me a ultrapassar os momentos mais difíceis que enfrentei e sempre me motivou a não desistir.
- Ao meu primo, Adriano, que me apoio não só nalgumas revisões de escrita, mas principalmente colaborou em montar todos o artefactos gráficos dos jogos desenvolvidos e usados ao longo desta tese.
- Ao meu amigo, Lucas pelo apoio emocional que me deu e por rever profundamente a escrita de várias publicações que escrevi e oferecido as suas sugestões.
- Ao resto da minha família que sempre esteve próxima de mim, aos meus avós, Olga e Marcelino, as minhas tias João, Cecília e Olga e ao meu tio, Francisco.
- Ao meu amigo, David, como antigo colega de trabalho e por me orientar inicialmente nos fundamentos da área e pela nossa colaboração.
- À minha actual colega Rita, pelas colaborações que conseguimos neste último ano, relembrando-me o bom espírito de trabalho de equipa.
- Aos meus orientadores Luís e Nuno, que não só me deram esta oportunidade, mas também motivaram-me a continuar em momentos que quis desistir.

Quero também agradecer formalmente a nível institucional à FCT (Fundação para a Ciência e Tecnologia) pelo financiamento deste projecto de investigação através das bolsas individuais SFRH/BD/129445/2017 e COVID/BD/151875/2021, ao Projecto Intelwheels 2.0 (POCI-01-0247-FEDER-039898), também pelo co-financiamento, ao LIACC (Laboratório de Inteligência Artificial e Ciências da Computação) da Universidade do Porto e IEETA (Instituto de Engenharia Electrónica e Informática de Aveiro) da Universidade de Aveiro pelo acolhimento, e disponibilização de recursos para a realização desta tese.

Simão Reis

"On my business card, I am a corporate president.
In my mind, I am a game developer.
But in my heart, I am a gamer."

Satoru Iwata

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Introduction

1.1 Motivation

Games have as their goal to be fun, and to offer their players a pleasant experience while being played. Entertainment can be offered through many types of tasks and challenges to a player. In the case of multiplayer games, it can provide competition and cooperation, or both even, and games can also provide a spectacle for audiences. Certain games are designed to have a more casual audience, which does not commit so much time or effort into games, while other games are designed for those who desire a pure challenging experience, but games can be designed to be easy to learn and hard to master appealing to both audiences. Other factors, like the specific type of challenges the games provide for players (puzzle, strategy, dexterity, etc.), the setting, music, and environment, and all those characteristics are also many times important for players. Some games, like Serious Games, have scope outside pure entertainment, offering a medium for education, training, or rehabilitation, change of attitude and behavior, simulations of real-world processes, or persuasive purpose (Sajjadi et al., 2022). Like pure entertainment games, these should have their characteristics carefully tuned to their user, mainly for delicate treatments.

One particular game-defining characteristic cited by players and developers alike is game balance. Is the difficulty of a game properly balanced, and does it grow as a player progresses through the game, or does some game mechanics offer unfair advantages over other players, by being too weak to allow opposing adversaries, or too strong that the game becomes too easy against any opponent? These issues are important for a variety of reasons, for example, some games offer players character options and customization. If suddenly new characters are released making it very difficult or impossible for older characters to win, that may disengage some players; while other players may become bored due to stale meta-games, as new characters do not offer any type of new experience as they cannot fathom to compete as well as current strategies. Similarly, players may feel that the game is determined too much by chance, rendering their choices meaningless - this is usually criticized by hardcore players, while casual players may feel more motivated to play if they feel the game assists them to stay on level with their opponents through adaptive difficulty.

Indeed, there is not a single consensus on the definition of game balance. Becker and Görlich

(2020) examined reports from fourteen lead game designers and critics trying to construct some conceptual basis for the term. The authors conclude that the four elements more present in the reports are: Difficulty, balancing (as a purposive act), symmetry, and expected characteristics of a well-balanced game. The main expected characteristics of games are that:

1. Early mistakes should not prevent victory in a later stage;
2. Chance should not render skill irrelevant (skill here is either understood as good decision-making or dexterity in executing the actions, but either way players should feel the game progress is a consequence of their actions);
3. Multiple strategies should be viable;
4. Symmetry provides equal starting conditions for every player, and, although being a sufficient condition for fairness (as in giving equal chances to each player to win), it is not a sufficient condition for an actual deep and engaging game with meaningful decisions.

Difficulty in the game should keep a state of flow (Csikszentmihalyi, 2014), a psychological state where a player doesn't feel bored or frustrated with the difficulty of the task at hand. But while in single-player games, the line of difficulty may be adjusted to a player's needs without extra concerns, in multiplayer games the change of difficulty to one player may inversely affect the other(s). Finally, balancing as a process is not unique with a probable cause of the lack of consensus in the term, and most importantly, the process itself is difficult because all elements of a game work in a fragile equilibrium where small changes may drastically change how a game works, thus changing how the players see and feel the game, losing their attachment to it.

This motivates investigation for automatic tools that can ease the process of game balance, by efficiently exploring the game rule space, which is usually intractable for human designers. Traditional hand-tuned methods require many manual trial and error processes, including re-configuration, re-implementation, and re-testing, and on many occasions, a lot of human gameplay data is required for data analytics but not possible to collect (for example in the early stages of design and development). Such tools could also go beyond the boundaries of entertainment and enter the realm of Serious Games, games whose purpose is to serve as an engagement tool for educational, health, or training purposes. Automatically adjusting the task difficulty could enhance the serious purpose of those games.

1.2 Research Problem

Artificial Intelligence (AI) had a significant evolution over the last few years, which contributed to a good deal of advances in the Game AI field. However, most work done in academia specifically for automated game balance is still limited to specific game-oriented techniques, and mathematical or analysis tools - many of which are niche, and not in accessible state to be used by game designers and developers. AI now provides more tools than ever to assist practitioners in many fields, and never it has been more accessible: in particular, it can now easily be integrated into games and the

game development loop itself. Game balancing is not a trivial task and much less in multiplayer games, but we hypothesize that the processes involved in it can be formulated in ways that AI systems could tackle, which raised this thesis' main research question:

How can AI Automatically Balance Multiplayer Games?

There is a wide array of definitions and comprehensions for the game balance in terms of process and its desired characteristics, but we categorize two main distinct processes relative to their goals: Power Balance, which focuses on the strategy viability (in terms of advantage compared to other strategies) and the possibility of meaningful decisions; and Difficulty Balance, which tries to accommodate the game to players to compensate their faults. We observed, including from peer review processes during our scientific publications that, both Difficulty Balance and Power Balanced concepts are loosely mixed and used. Therefore, the main research question is split into two sub-questions:

1. *How can AI Automatically Power Balance Multiplayer Games?*
2. *How can AI Automatically Balance the Difficulty in Multiplayer Games?*

By answering both sub-questions, our overall main question is also answered. Therefore, this thesis tries to justify power and difficulty balance as the two major categories of the game balancing process and demonstrate AI as an assisting tool easing both tasks to game designers.

In particular, this work focuses on multiplayer game balance, which provides extra challenges, is less studied and comprehended, and is more required by the industry. In Power Balance, it is assumed that the true most powerful strategies are only revealed when played by skilled agents, and in multiplayer games, this becomes more challenging as players must adapt to their opponent's strategies to win, and small changes can have major impacts to how the game unrolls. While balancing the difficulty, players may present any degree of skill, and available mechanisms must be changed to compensate for players' mistakes on the fly, while not affecting negatively the experience of both players, which eventually have opposing objectives, and whose actions cannot be controlled directly.

Dynamic balance refers to processes made during the gameplay, and static balance to processes that happen before a game starts. It is important to note the scope of this study, which are dynamic difficulty balance and static power balance, but static difficulty can also be achieved, for example, by changing the symmetry or initial conditions of a game, and while power balance can be performed dynamically, it is usually released preemptively with game patches. Other influential aspects of a fun and functional game like solvability are not widely considered in this work. As we will see through this thesis, the distinction between the static and dynamic nature of these processes will determine the use of different tools to accommodate each, while showing some common relations.

1.3 Key Contributions

The overall main contribution of this thesis is:

- Explain the key differences and similarities in difficulty and power balance in multiplayer games, and advocate for AI as an assist tool for game developers in both tasks.

Several contributions for **Difficulty Balance** are presented. We give strong justification for the application of Dynamic Difficulty Adjustment (DDA) in real multiplayer game scenarios; our state-of-the-art revision showed that the work in the field of Multiplayer Dynamic Difficulty Adjustment (MDDA) is still limited. Furthermore, we present an automation framework for difficulty balance in two-player games, where the balance task is modeled as a Reinforcement Learning (RL) problem. A Game Master (GM) agent is trained to pilot handicap mechanisms in a three-phase pipeline, which is signaled by a reward function that measures the weighted combination of aesthetic criteria that values dramatization, scenarios where a lead player retraces while a behind player recovers. A trained GM promotes the frequency of rubber banding situations, compensating for skill gaps. We study each scenario over different populations of sub-optimal agents, agents that make a slightly worse choice than a model policy. The quality of the resulting games (with the trained GM embedded), is examined by measuring the same aesthetic criteria on the resulting games, by observing the changes in the game, and through user validation via a conducted user study. In sum, the main contributions of this topic are:

1. The application of Deep Reinforcement Learning (DRL) for MDDA where a GM agent is trained and embedded into a game to enable handicaps in a timely fashion, automatically creating rubber-banding effects where the advantage discrepancy between two opponents is lessened.

2. Demonstrating the application of our MDDA-RL framework in micro two-player adversarial games. We present how the games were conceived from the perspective of a game designer and how the GM improves the intensity in these games through computation analysis and a user study. Even if the framework application is general, each game still requires specifications.

Several contributions for **Power Balance** are presented. We design a framework for a new type of meta-game balance AI Competition based on Pokémon. The aim of the framework is to facilitate competitions in creating the most balanced meta-game possible: one where there is a large variety of Pokémon and moves to choose from, and many possible combinations that are effective. We explore 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 re-adapts the Pokémon's inner attributes to incentive the team builder agents to incorporate a greater variety of Pokémons into their teams, increasing the meta-game's overall diversity and balance. These agents make use of meta-gaming, linear programming, and evolutionary search to find strong combinations against the current meta-game. Deep learning is also employed to predict the outcome of matches and recommend constructive elements of teams, and build an initial Pokémon battling agent. The strongest team builder is faced against the team meta-game balance agent for its evaluation. In sum, the main contributions of this topic are:

1. The design of the Video Game Championship (VGC) AI Competition, the first meta-game balance AI competition, whose purpose is to compare multiple tasks (in this case Pokémon related tasks) related to power game balance in a common domain. We believe it is an inspiring source for future ventures in other similar game domains.

2. Demonstrating that the VGC AI Competition can be solved by mixing and applying in new ways current well-known AI techniques.

1.3.1 Scientific Publications

Several works were published during the writing of this thesis and are listed and summarized below. This thesis author was the main author in every, except for one where it was a co-author with equal contribution, and had the main contribution in two co-authored works.

1. **Reis, S.**, Reis, L.P., Lau, N. (2019). Player Engagement Enhancement with Video Games. In: Rocha, Á., Adeli, H., Reis, L., Costanzo, S. (eds) New Knowledge in Information Systems and Technologies. WorldCIST'19 2019. Advances in Intelligent Systems and Computing, vol 931. Springer, Cham. https://doi.org/10.1007/978-3-030-16184-2_26

    - This work surveys the main research sub-fields related to DDA. It discusses conceptual frameworks like GameFlow, and work and techniques done in DDA, MDDA, Procedural Content Generation (PCG) oriented to quality and user experience, Player Modeling, and Affective Computing.

2. **Reis, S.**, Reis, L.P., Lau, N. (2019). Automatic Generation of a Sub-optimal Agent Population with Learning. In: Rocha, Á., Adeli, H., Reis, L., Costanzo, S. (eds) New Knowledge in Information Systems and Technologies. WorldCIST'19 2019. Advances in Intelligent Systems and Computing, vol 931. Springer, Cham. https://doi.org/10.1007/978-3-030-16184-2_7

    - This work proposes the evolutionary property of RL to generate a population of surrogate agents with different skill degrees on a game task. The populations are evaluated as a trace of sorted agents, where the min-max gap and the skill density is evaluated.

3. **Reis, S.**, Reis, L.P. & Lau, N. Game Adaptation by Using Reinforcement Learning Over Meta Games. Group Decis Negot 30, 321–340 (2021). https://doi.org/10.1007/s10726-020-09652-8

    - This work proposes the initial MDDA framework using DRL, where a GM agent is trained and embedded in a game environment to adapt the difficulty for two player games, over a population of agents with different skill levels in a simple grid-world environment.

4. D. Simões, **S. Reis**, N. Lau and L. P. Reis, "Competitive Deep Reinforcement Learning over a Pokémon Battling Simulator," *2020 IEEE International Conference on Autonomous*

Robot Systems and Competitions (ICARSC), Ponta Delgada, Portugal, 2020, pp. 40-45, doi: 10.1109/ICARSC49921.2020.9096092.

- This work studies the feasibility of asynchronous adversarial DRL to train a Pokémon battle agent. For that purpose, was developed the first iteration of the Pokémon Battle Environment (PBE), a RL environment where on-the-fly generated Pokémon specimens can be used for generalized training purposes.

5. **S. Reis**, L. P. Reis and N. Lau, "VGC AI Competition - A New Model of Meta-Game Balance AI Competition," *2021 IEEE Conference on Games (CoG)*, Copenhagen, Denmark, 2021, pp. 01-08, doi: 10.1109/CoG52621.2021.9618985.

   - This work proposes the VGC AI Competition, an AI competition model and framework for team building and meta-game balancing.

6. **S. Reis**, R. Novais, L. P. Reis and N. Lau, "An Adversarial Approach for Automated Pokémon Team Building and Meta-Game Balance," in *IEEE Transactions on Games*, doi: 10.1109/TG.2023.3273157.

   - This work explores multiple team building and meta-game balance methods over the VGC AI Framework, mixing multiple AI techniques, to simulate an adversarial model where the team building agents narrow to the best strategies, while the balance agent maximize the viable strategies.

7. **S. Reis**, R. Novais, L. P. Reis and N. Lau, "Automatic Difficulty Balance in Two-Player Games with Deep Reinforcement Learning," *2023 IEEE Conference on Games (CoG)*, Boston, MA, USA, 2023, pp. 1-8, doi: 10.1109/CoG57401.2023.10333240.

   - This work extends our initial MDDA framework by proposing aesthetic game design criteria as an incentive factor to cause rubber-banding effects between two players, and validation is done through numerical game fitness and observation of game sessions.

1.3.2 Research Artifacts

The work developed in this thesis required the implementation of several environments to validate both the multiplayer power and difficulty balance frameworks and techniques, and can be used for future avenues to compare multiple solutions.

1. The MDDA-RL Test-Suit contains a set of RL environments where a triplet of agents act, two player agents and a GM agent, allowing to compare results of future MDDA endeavors and conduct user experiments, where humans can play versus AIs or other humans.

2. The VGC AI Framework, an ecosystem allowing to test and coordinate of three different types of agents involved in a meta-game balance process, allowing the comparison of multiple solutions, through an AI competition model with three main tracks.

1.3.3 Scientific Competitions

In the context of this thesis, the first edition of the VGC AI Competition was organized at the IEEE CoG23, Northeastern University, Boston, US. It had a total of six valid participants over the Battle Track of the competition. The solution included rule-based agents, and A* search algorithm and a DRL agent.

1.4 Thesis Overview

Given the duality of subjects, this thesis is organized into two main chapters covering every aspect of their respective topic: motivation, state-of-the-art, proposed methodology, and validation. Both chapters are first preceded by a background on Game Theory (GT) and various topics on AI with the main focus on RL in Chapter 2, covering the fundamental techniques applied in this thesis for less familiar readers. Our study on Difficulty Balance in multiplayer games is covered in Chapter 3. Our study on Power Balance in multiplayer games is covered in Chapter 4. Finally, in Chapter 5, we discuss our final conclusions, what lessons were learned in this research thesis, what are the answers to our main research sub-questions, what are the current limitations, what the future of game balance as a research field holds and how AI can contribute in future pursues.

Chapter 2

Background

2.1 Game Theory

Strategic interaction is one of the most present elements in multiplayer games. GT is a mathematical framework, a collection of analytical tools for optimal choices in interactional and decision-making problems (Sohrabi and Azgomi, 2018). The essential assumptions about GT are that decision-makers are rational and seek well-defined external objectives, and they use their knowledge or beliefs of the other decision-makers actions to reason strategically (Osborne and Rubinstein, 1994). A rational agent is aware of all possible outcomes and assigns probabilities over actions in order to maximize their expected utility. Utility measures how much an agent prefers one action over another. With multiple-agent interaction, the problem of infinite regress surges: one agent believes in the belief the other agent has of my belief, and so on. GT aims to model and understand strategic situations by modeling them as games and trying to predict their outcome.

2.1.1 Simultaneous Games

In simultaneous games, each player selects a strategy without knowing about the choices being made by other players. Players can make their choice in different instances of time but are always revealed simultaneously. These types of games have a normal-form representation. A normal-form game is the most basic description of a game in GT. These games are composed of players, the actions available to each, and the payoffs resulting from joint choices. A normal-form game is a tuple $G = \langle P, A, u \rangle$ where:

- P is a set with n players;
- $A = \langle A_i, \ldots, A_n \rangle$ is a tuple of sets of actions, one for each player, where $A_i = \{a_{1,i}, \ldots, a_{k,i}\}$ is the set with k actions of player i;
- $u = \langle u_i, \ldots, u_n \rangle$ is a tuple of utility functions, one for each player, where a utility function $u_i : A_1 \times \ldots A_n \rightarrow \mathbb{R}$ that given the joint action from every player returns player i's payoff.

In a two-player scenario, a normal-form game is represented by a bi-matrix. Each row is associated with an action the first player can take, and the same is true for each column of the second player. Each cell contains the payoffs for each of the players resulting from joint action. An example of a bi-matrix is represented in Table 2.1, illustrating the classic Prisoner's Dilemma game. Figure 2.1 illustrates the formal game description of the Prisoner's Dilemma.

Table 2.1: Prisoner's Dilemma Payoff Table.

	Cooperate	Defect
Cooperate	3,3	0,4
Defect	4,0	1,1

$$P = \{r, c\}, A = (A_r, A_c), u = (u_r, u_c)$$

$A_r = \{\text{Cooperate, Defect}\}$ $\quad A_c = \{\text{Cooperate, Defect}\}$

$u_r(\text{Cooperate, Cooperate}) = 3$ $\quad u_c(\text{Cooperate, Cooperate}) = 3$

$u_r(\text{Cooperate, Defect}) = 0$ $\quad u_c(\text{Cooperate, Defect}) = 4$

$u_r(\text{Defect, Cooperate}) = 4$ $\quad u_c(\text{Defect, Cooperate}) = 0$

$u_r(\text{Defect, Defect}) = 1$ $\quad u_c(\text{Defect, Defect}) = 1$

Figure 2.1: Prisoner's Dilemma Game Definition.

A pure strategy s must completely describe by itself how a player will play a game. The strategy set $S_i = \{s_1, \ldots, s_n\}$ is the set of all pure strategies available to the player i. In normal-form games, a pure strategy is simply constituted by the action chosen by the player, $s_i := \langle a_i \rangle$, since players make only one move and are solely described by the action they chose.

2.1.2 Strictly Dominated Strategies

A strategy a_1 of a player strictly dominates another strategy a_2 of the same player if the payoff of joint action a_1 paired with any actions from the opponents is greater than joint strategy a_2 paired with any actions from the opponents. A rational agent will never play strategy a_2. Formally, a strategy a_i is strictly dominated by a'_i if and only if:

$$u_i(a_i, a_{-i}) < u_i(a'_i, a_{-i}) \textbf{ for all } a_{-i} \in A_{-i}, \tag{2.1}$$

where a_{-i} is the joint action from all other players, and A_{-i} the set of every other player joint strategies. For example in Prisoner's Dilemma, Table 2.1, Cooperate strategy is strictly dominated by Defect strategy on row player case, $u_r(\text{Defect}, \cdot) > u_r(\text{Cooperate}, \cdot)$ and in column player case $u_c(\text{Defect}, \cdot) > u_c(\text{Cooperate}, \cdot)$.

2.1.3 Nash Equilibrium

Nash Equilibrium (NE) (Nash, 1950) is a central solution of GT. Joint actions are NE if they have the property that the change of strategy by a single player cannot improve its overall payoff, as

such no player has the incentive to deviate alone. If players are playing NE then each player is giving their best response to each other. A response is the best response $a_i^* \in BR$, with BR the best response set, if and only if:

$$u_i(a_i^*, a_{-i}) \geq u_i(a_i, a_{-i}) \textbf{ forall } a_i \in A_i. \tag{2.2}$$

A joint action $a = \langle a_i, \ldots, a_n \rangle$ is a pure strategy NE if and only if:

$$a_i \in BR(a_{-i}) \textbf{ forall } i. \tag{2.3}$$

Agents may play NE because:

- Rational reasoning;
- Correct beliefs: Pre-play communication, negotiation between the players before the game starts; Trial-and-error adjustment, accumulated experience over the same game or similar games.

Rationality together with correct beliefs makes players play NE. In Prisoner's Dilemma, Defect is the best response in both players' cases. The joint action ⟨Defect, Defect⟩ is the only NE in the Prisoner's Dilemma game.

2.1.4 Mixed Strategies

Mixed strategy s is a probability distribution over pure strategies of player i, $s_i = \langle p_i(a_1), \ldots, p_i(a_k) \rangle$, where $p_i(a_k)$ is the probability of player i choosing the action a_k. A mixed strategy is a pure strategy if a single strategy has a rate choice of one. If no strategy is null, then every pure strategy has a strictly positive probability of selection and is called a totally mixed strategy. The payoff of joint actions is given by the expected utility from decision theory:

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{j \in P} p_j(a_j). \tag{2.4}$$

A game with mixed strategies has NE as long as the number of players is finite, and each player has a finite number of pure strategies. To win, players must make themselves unpredictable, introducing randomness in their behavior. A response is the best response $s_i^* \in BR$, where BR is the best response set, if and only if:

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i}) \textbf{ forall } s_i \in S_i. \tag{2.5}$$

A joint strategy s is a pure strategy NE if and only if:

$$s_i \in BR(s_{-i}) \textbf{ forall } i. \tag{2.6}$$

S_i is the strategy set for player i. $S = \langle S_1, \ldots, S_n \rangle$ is the tuple of sets of strategies, one for each player.

For example, in Rock Paper and Scissor (see Table 2.2), there is no mutual best reply (there is no pure strategy NE). To be unpredictable, player i wants that $u_i(a_1) = \cdots = u_i(a_k)$. In the Rock Paper and Scissor game this results in the joint mixed strategy NE=$\left\langle \left\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right\rangle, \left\langle \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right\rangle \right\rangle$.

Table 2.2: Rock Paper and Scissor Payoff Table.

	Rock	Paper	Scissor
Rock	0,0	-1,1	1,-1
Paper	1,-1	0,0	-1,1
Scissor	-1,1	1,-1	0,0

2.1.5 Repeated and Stochastic Games

A repeated game consists of the repetition of some base/stage simultaneous game. The average payoff of player i in a repeated game is given by:

$$\frac{1}{T} \sum_{t=1}^{T} u_i \left(a_i^{(t)} \right), \tag{2.7}$$

where $a_i^{(t)}$ is the action from player i at iteration t and T is the number of iterations played in total.

Infinitely Repeated Games With Discounting consist in most real case scenarios, we do not know when the outcome of the game will be decided, so we care more about the early rounds/iterations of the game, the ones we are most sure that will be played and carry more weight on the game outcome. The future discounted reward of player i is given by:

$$\sum_{t \geq 0} \delta^t u_i \left(a_i^{(t)} \right), \tag{2.8}$$

where $\delta \in [0,1]$ is the discount factor. The discount factor could also be interpreted as the propensity of the game to end at any given round, with probability 1-δ.

Stochastic Games (Shoham and Leyton-Brown, 2008) are a generalization of repeated games. The game played at any iteration depends on the previous game played and all actions taken by the players. In multiple application settings, like single agent systems, a base game is called the game state or state. A Stochastic Game is a tuple $\langle \mathscr{G}, P, A, p, u \rangle$ such as:

- Set of some stage games $\mathscr{G} = \{G_1, G_2, \ldots\}$;
- P players;
- $A = A_1 \times \cdots \times A_n$, A_i is the finite set of actions available to player i;
- $p : \mathscr{G} \times A \times \mathscr{G} \to [0,1]$ probability of playing a game after taking an action while playing a certain game;

- $u = u_i, \ldots, u_n$, where $u_i : \mathscr{G} \times A \to \mathbb{R}$ is a utility function for player i.

Constant-sum games are games where the available resources are affected by the player's choices. In the case of zero-sum games, the gain of one player is the loss of another, hence, the total adds to zero, in chess, the victory of one player implies the loss of the other. In General-sum games, the gain of one player does not necessarily mean the loss of the other. Prisoner's Dilemma is a General-sum game, while Rock Paper Scissors is a zero-sum game.

2.2 Machine Learning

Machine Learning (ML) is a sub-field of AI that aims to model and solve very complex problems beyond human capability, i.e., build prediction or decision functions without being explicitly programmed. ML techniques have a statistical nature; by using past input samples, they try to find some pattern in the data to predict the outcome of future or unseen samples.

ML problems can be categorized by their task. In Supervised Learning, the data set is labeled, which means for each sample the model gets to know the right answer. In Unsupervised Learning, no labels are given to the data and the aim is to find some kind of structure in the data. Supervised Learning has two main applications: Regression and Classification. In Regression, we try to build a prediction model, a function with continuous domain and image. In classification, we try to label samples in a discrete number of classes. When the number of classes is greater than two, it is called Multi-Label Classification.

2.2.1 Deep Learning

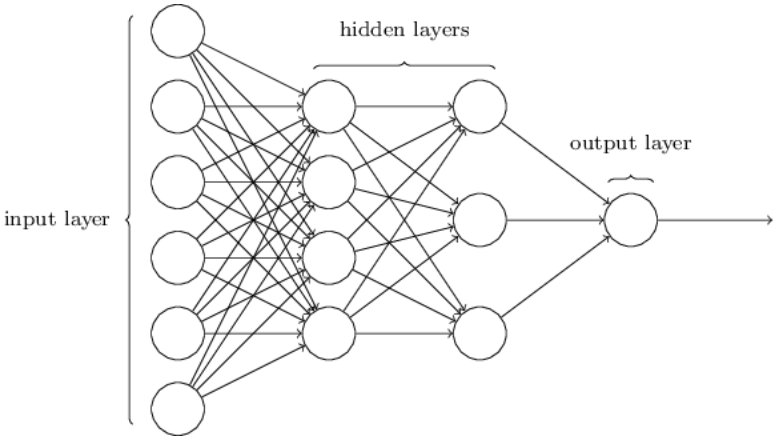


Figure 2.2: Deep Neural Network Architecture.
Source: Neural Networks and Deep Learning, Determination Press, 2015.

Artificial Neural Network (NN) are a ML model inspired by the biological brain structure, where perceptrons propagate signals through synapses (Duarte et al., 2020). A single neuron is the basic unit of processing in a NN. Perceptrons are organized in interconnected layers. Each

neuron receives input from all perceptrons from the previous layer, processes the input, and propagates its output to every neuron in the next layer. In addition, each neuron receives a bias input. Connections between two perceptrons i and j have an associated weight $\theta_{i,j}$.

The inference process of a neuron is constituted by a composition of an activation function f with a propagation function p, a linear combination of the previous neuron output, weighted at each connection,

$$p_{i,j} = f\left(\sum_k \theta_{j,k} p_{i,j-1}\right). \tag{2.9}$$

Nonlinear activation functions allow for the modulation of complex, non-trivial functions. The Universal Approximation Theorem states that a NN with at least one hidden layer is enough to represent any function (Cybenko, 1989), but it needs to be infeasible large and fails to converge in practice, hence modern results adapting for very deep architectures.

The input layer does not perform any computation, it is just set with the input values. The output layer returns the final predicted value. Any layer in between is called a hidden layer (Figure 2.2). A NN may have any number of layers, and each layer may have a different number of perceptrons, and its perceptrons may be from different types from one layer to another. There exist multiple NN and most are defined by their architecture and layer types. We describe some common examples:

- **Multi-Layer Perceptron (MLP)** is the simplest type of network. Defined by not having cycles, which means that each layer input depends only on the previous layer output.
- **Convolutional NN (CNN)** is known by shift/space invariant network (Li et al., 2022). Commonly used in visual learning. The first hidden layers are called convolutional layers, and using a filter they perform abstract feature extraction, while the remaining layers perform the usual classification.
- **Recurrent NN (RNN)** have memory by connecting perceptrons output to their input (Mousavi et al., 2018). Allows recognition of temporal patterns or sequences. Recursion connections act like memory.
- **Long-Short Term Memory (LSTM)** are an enhanced version of Recurrent networks, handling long-term dependencies (Graves, 2012).
- **Autoencoders** use the output labels equal to the input, and use smaller hidden layers, achieving dimensionality reduction from the data point samples (Burkov, 2019). Has a wide application in unsupervised learning.
- **Residual Neural Network (ResNet)** adds skip connections, connections that jump directly to two or three layers ahead (He et al., 2016). They were conceived to tackle the Degradation (or accuracy saturation) problem and the Vanishing Gradient problem, allowing previous layers to perform as well as current layers.

- **Transformers** are a network model which implement a self-attention mechanism (Vaswani et al., 2017). Attention in ML uses an encoder-decoder architecture, where not only the token vector is used but all token vectors of a sequence. Self-attention extends this concept but searching for meaningful associations, and transformer models use a stack of self-attention.

2.2.2 Regression, Optimization, and Backpropagation

NN, like other ML algorithms, need a cost function that defines the optimization problem, directing its search on the problem space. In the case of NN, we want to minimize the distance between the result from the output layer and the real labeled output (in the Supervised Learning case). We do so by changing the connection weights. During the inference stage, the weights are fixed.

Gradient Descent is an algorithm that aims to find the minimum of a function f. We iteratively navigate in the symmetrical direction to the gradient over the function domain until we converge to a minimum (see Figure 2.3). If we are on point x_t and perform one step, we reach the point $x_{t+1} := x_t - \alpha f(x_t)$, where α is the learning rate, a hyperparameter that can be used to configure the step size. We stop when reaching the point when $\alpha f(x_t) = 0$.

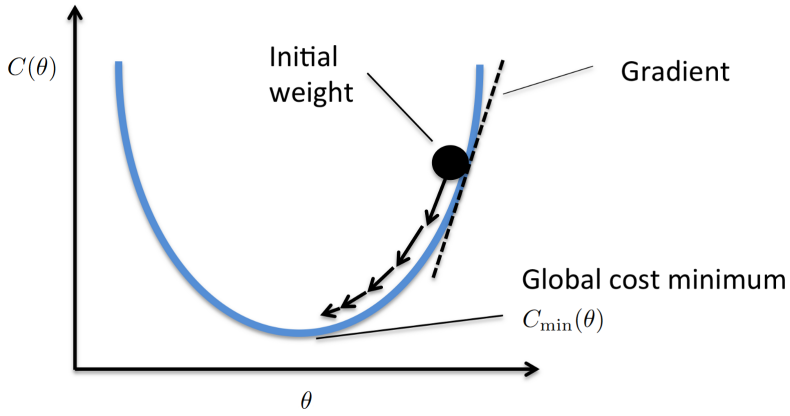


Figure 2.3: Gradient Descent Visual Representation.
Source: Machine Learning, Coursera, 2012.

When working with data samples (input and output point pairs) of a function, the minimization objective is the difference between the function's output and our hypothesis called the cost (or loss) function C. $\theta = \langle \theta_0, \theta_1, \ldots, \theta_k \rangle$ is the weight vector of our hypothesis, $x = \langle x_0, x_1, \ldots, x_m \rangle$ is a data sample, $y = \langle y_0, y_1, \ldots, y_n \rangle$ is the real output label of a sample, and $h \sim y$ is our hypothesis or function estimator of f given by the linear combination $h_\theta(x) = \sum_i^N \theta_i x_i$. The data has n features and m samples. A commonly used cost function for regression problems is the Mean Square Different:

$$C(\theta) = \frac{1}{2M} \sum_i^M \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2. \tag{2.10}$$

Note that $\nabla C(\theta) = \frac{1}{M} \sum_i^M h_\theta \left(x^{(i)} \right) - y^{(i)}$ and $\theta_{t+1} := \theta_t - \lambda \nabla C(\theta)$.

For problems of classification a logistic regression is used:

$$C(\theta) = \frac{1}{M}\sum_{i}^{M} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right). \tag{2.11}$$

Cost returns $-\log(h_\theta(x))$ if $y = 1$ or $-\log(1 - h_\theta(x))$ if $y = 0$.

Backpropagation is the algorithm used to update the weights of the perceptron connections over multiple layers, calculating and expanding the gradient of the cost function. The network weights are optimized by calculating partial derivatives (with respect to the weights) of the network loss function. The partial derivatives are propagated and accumulated through the layers (from the output to the input) as given by:

$$\theta_{i,j}(t+1) = \theta_{i,j}(t) + \alpha\frac{\partial C(\theta)}{\partial \theta_{i,j}}. \tag{2.12}$$

2.2.3 Challenges in Training Deep Models

Bias–variance trade-off refers to two sources of error in an estimation process. Variance relates to the disparity of data, while bias relates to the shift between the predicted value against the true value. Normally, we want a model to capture the patterns of the training data, but that can also generalize to unseen samples. High bias results from simpler models, and results in the model under-fitting, i.e., unable to capture the patterns of the data set, while high variance represents the training set well but fails to generalize to unseen samples, over-fitting. The objective is to minimize both. Classic techniques can be used in supervised learning to fight both high variance and high bias, for example, high variance can be tackled with regularization or larger data sets, while high bias can be tackled by extending the number of features or decreasing regularization.

The Curse of Dimensionality (Keogh and Mueen, 2017) refers to the exponential growth of required data needed with the growth of the number of features in the data samples. If no extra data is available, the usual method to tackle this issue is dimensionality reduction, normally divided into feature selection and feature extraction. Feature selection involves detecting features with high correlations, that have missing values or using techniques like Random Forrest. Feature extraction uses more automated techniques like Principal Component Analysis (PCA) (Jolliffe, 2011).

Vanishing Gradients refer to a phenomenon where a gradient in a backpropagation process becomes so small it becomes impossible to update, making it impossible for a network to further improve (Basodi et al., 2020). Multiple practical solutions exist, like Batch Normalization, where each layer is re-scaled and re-centered. Correct weight initialization can also tackle this issue. The use of proper activation functions like ReLU (Hara et al., 2015) or the use of residual networks allows gradients to propagate directly to deeper layers.

Another major factor is the selection of hyperparameters like the learning rate, and network architecture in terms of the number of layers, size of layers, activation functions, etc. Some techniques search automatically for good hyperparameters but are too computationally heavy. A good practice is to use similar hyperparameters of other domains with similar complexity.

2.3 Reinforcement Learning

RL is a class of ML algorithms oriented to complex goal-solving (Campesato, 2020). In RL an agent improves its performance in a trial-and-error fashion by exploring its surrounding environment (François-Lavet et al., 2018). Feedback in the form of reward (or punishment) is given as a consequence of the action taken by the agent, making it more likely to repeat or forego those behaviors in the future.

An intelligent agent system is one constituted by one or more agents that interact with an environment, as illustrated in Figure 2.4. An agent is an entity that can perceive the environment that surrounds it through sensors (gets a state s as input) and act through actuators (outputs an action a) (Sutton and Barto, 1998). An agent is rational if it tries to maximize its expected utility.

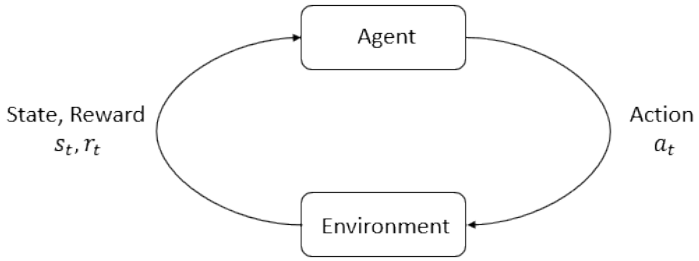


Figure 2.4: Agent-Environment Interaction Loop.
Source: Machine Learning For Beginners, Towards Data Science, 2018.

2.3.1 Markov Decision Process

Markov Decision Process (MDP) (Littman, 1994) is a common framework for single agent RL. A MDP is a tuple $\langle S, A, e, R, \delta \rangle$ where:

- S is the set of all possible states;
- A is the set of all possible actions;
- $e(s_{t+1}|s_t, a_t)$ is a world transition model;
- $R(s,a) : S \times A \rightarrow \mathbb{R}$ is the immediate reward, which is the utility that the agent receives by taking action a on state s;
- δ is a discount factor.

In the case of multiple agents, the model is generalized as a Stochastic Game.

Given s_t, the state of the environment at time-step t, and the set of all possible states S, an environment is fully observable if $o_t = s_t \forall t$, that is, the observation fully describes the state of the world. If the observation only describes part of the world then the world is partially observable.

Some environments have discrete action spaces, meaning that only a finite number of actions are possible. These can eventually be branched into multi-discrete actions, where multiple actions

are taken and chosen from different branches. Others, like physical control systems, have continuous action spaces, where actions are real-valued vectors. Some tasks require hybrid discrete and continuous actions.

A transition or world model specifies how the environment changes when an agent performs an action. If the transition is deterministic, then the pair $\langle a_t, s_t \rangle$ fully determines s_{t+1}. If the world is stochastic there is a probability distribution $e(s_{t+1}|s_t, a_t)$ that describes the probability of reaching each possible state s_{t+1} given the pair $\langle a_t, s_t \rangle$. This means that the agent assigns a probability of the true state to each possible s for each possible observation o. Partial observability may be the cause of uncertainty in the world. If actions are not known *a priori*, may also lead to uncertainty.

2.3.2 Policies, Trajectories, and Return

The mapping of observation to the next action is called policy π of the agent. Given all observations and actions $\langle s_0, \ldots, s_t, a_0, \ldots, a_{t-1} \rangle$ and the chosen action a_t then:

$$\pi(s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t) \rightarrow a_t. \tag{2.13}$$

Storing all observations and action historical may be computationally unfeasible. It may be opted to use a reactive model where only the current observation is used $\pi(s_t) \rightarrow a_t$ (Vlassis, 2007). Here the current state fully characterizes its story before time-step t. This is called the Markov Property. The world state may have a full description or may be represented by handcrafted features. A policy π can be deterministic $a_t = \pi(s_t)$, or stochastic $a_t \sim \pi(\cdot|s_t)$.

A trajectory τ (also called episode or rollout) is a sequence of states and actions in the world produced by a policy interaction with the environment:

$$\tau = \langle s_0, a_0, s_1, a_1, \ldots \rangle. \tag{2.14}$$

The initial state s_0 is randomly sampled from a start-state distribution denoted by ρ_0, $s_0 \sim \rho_0(\cdot)$. State transitions can be either deterministic $s_{t+1} = e(s_t, a_t)$, or stochastic $s_{t+1} \sim e(\cdot|s_t, a_t)$.

The reward function R depends on the current state of the world, the last action, and the next state of the world (frequently just depends on the current state or state action pair):

$$r_t = R(s_t, a_t, s_{t+1}) \tag{2.15}$$

The return $R(\tau)$ is the cumulative reward over a trajectory. The infinite-horizon discounted return, which is the sum of all rewards ever obtained by the agent, but discounted $\delta \in [0, 1]$ by how far off in the future they're obtained:

$$R(\tau) = \sum_{t=0}^{\infty} \delta^t r_t. \tag{2.16}$$

2.3.3 The Reinforcement Learning Problem

The goal in RL is to select a policy which maximizes expected return $E_{\tau\sim\pi}[R(\tau)]$ when the agent acts according to it (optimal policy π^*):

$$\pi^* = \arg\max_{\pi} E_{\tau\sim\pi}[R(\tau)]. \tag{2.17}$$

The expected return is the accumulated reward of traversing a trajectory:

$$E_{\tau\sim\pi}[R(\tau)] = \sum_{\tau} e(\tau|\pi)R(\tau). \tag{2.18}$$

The probability of a trajectory with T steps is:

$$e(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} e(s_{t+1}|s_t,a_t)\pi(a_t|s_t). \tag{2.19}$$

2.3.4 Value Functions and the Bellman Equations

Many RL algorithms make use of the value of a state, or state-action pair, corresponding to the expected return if starting in that state or state-action pair, and then act according to a particular policy forever after. On-Policy Value Function $V^{\pi}(s)$ gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)|s_0 = s]. \tag{2.20}$$

On-Policy Action-Value Function $Q^{\pi}(s,a)$ gives the expected return if starting in state s, take an arbitrary action a, and then forever after act according to policy π:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}R[(\tau)|s_0 = s, a_0 = a]. \tag{2.21}$$

Optimal Value Functions are similar, only we follow the optimal policy $\pi^*(s)$:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \tag{2.22}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.23}$$

There are two key relations between the Value Function and the Action-value Function. On-Policy, the value of being on state s is the expected value of following the policy π:

$$V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a)] \tag{2.24}$$

Off-Policy, the value of being on state s is the maximum value we can get from any action:

$$V^*(s) = \max_{a} Q^*(s,a) \tag{2.25}$$

Bellman Equations (Bellman, 1957) define Q and V updates alternatively and recursively following a given policy, obeying self-consistency rules. The Bellman equations for the On-Policy value functions are:

$$V^{\pi}(s) = E_{a\sim\pi,s'\sim e}[R(s,a) + \delta V^{\pi}(s')], \tag{2.26}$$

$$Q^{\pi}(s,a) = E_{s'\sim e}\left[R(s,a) + \delta E_{a'\sim\pi}[Q^{\pi}(s',a')]\right]. \tag{2.27}$$

The Bellman equations for the optimal value functions are:

$$V^{*}(s) = \max_{a} E_{s'\sim e}[R(s,a) + \delta V^{*}(s')], \tag{2.28}$$

$$Q^{*}(s,a) = E_{s'\sim e}[R(s,a) + \delta \max_{a'} Q^{*}(s',a')]. \tag{2.29}$$

2.3.5 Classic Approaches

The two main structured approaches are valued function estimation and direct policy search.

In value estimation through Dynamic Programming, it is assumed full knowledge of the MDP. Value iteration starts at $i = 0$ and V_0 as the initial guess of the value function. Compute V_{i+1} for all states s, and possible actions a using the Bellman Equations and update the policy for each state s if needed (policy Iteration):

$$V_{i+1}(s) \leftarrow \sum_{s'} e(s'|s,a)\left(R(s,a) + \delta V_i(s')\right), \tag{2.30}$$

$$\pi_{i+1}(s) \leftarrow \arg\max_{a} \sum_{s'} e(s'|s,a)\left(R(s,a) + \delta V_i(s')\right), \tag{2.31}$$

until V converges with the left-hand side equal to the right-hand side. Needs to compute expectations over the whole state space, which is impractical for most realistic scenarios.

In value estimation through Monte Carlo methods, policy iteration consists of two steps: policy evaluation and policy improvement. Given a stationary, deterministic policy π, the goal is to compute the function values $Q^{\pi}(s,a)$ for all state-action. A trajectory is produced using π, then for each state s, average the sampled returns. However, this procedure may spend too much time evaluating sub-optimal policies, using samples inefficiently and returns along the trajectories may have high variance, making convergence slow.

In value estimation through Temporal Difference (TD), bootstrapping is made from the current estimate of the value function. These methods sample from the environment, like Monte Carlo methods, and perform updates based on current estimates, like dynamic programming methods. The algorithm starts by initializing a table $V(s)$ arbitrarily, with one value for each state of the MDP. A positive learning rate α is chosen. We then repeatedly evaluate the policy π, obtain a reward r and update the value function for the old state using the rule below. The value $r + \gamma V(s')$ is known as the TD target.

$$V_{i+1}(s) \leftarrow V_i(s) + \alpha\left(r + \delta V_i(s') - V_i(s)\right). \tag{2.32}$$

SARSA (Sutton and Barto, 1998) is an On-Policy TD algorithm, with the update rule:

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha \left(r + \delta Q_i(s',a') - Q_i(s,a)\right). \tag{2.33}$$

Q-Learning (Watkins and Dayan, 1992) is an Off-Policy TD algorithm, with the update rule:

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha \left(r + \delta \max_a Q_i(s',a) - Q_i(s,a)\right).$$

2.3.6 Exploration and Exploitation

One of the main issues in RL is how agents should explore the environment. It is important for the agent to follow an exploration policy while learning. If an agent trusts too much early Q-values may not have the incentive to find better rewards. An optimal exploration policy would be one that accumulates many rewards as fast as possible, while at the same time exploring the environment as thoroughly as possible. The steps the agent performs near-optimally are classified as exploitation, and the steps far from optimally are classified as exploration (Hernandez-Leal et al., 2019).

An example of an exploration policy is called ε-greedy (Tokic, 2010). When in state s agent selects a random action with probability ε and action $a = \arg\max_a Q(s,a)$ with probability $1-\varepsilon$, with $\varepsilon < 1$ being a small number. ε values can be decreased over time as the agent becomes confident of the learned Q-values.

If the learning rate α is zero, the agent will only exploit current knowledge and will learn nothing new. If it is one, then will only consider the newest information, discarding all previous knowledge. The initial Q-values are often initialized with the first reward, allowing for some immediate learning, and are frequently initialized with high values, giving a high incentive for the agent to choose those unexplored actions.

2.3.7 Classic Multi-Agent Learning Approaches

In Multi-Agent Systems, multiple agents interact with the environment and each other. Adversarial games in particular often require agents to follow stochastic strategies (Busoniu et al., 2008). The transition model depends on the joint action of agents, and each agent receives an eventually different reward. Some assumptions, like the Markov property, do not hold (Tuyls and Weiss, 2012). In particular, the opponent's policy becomes the parameter of the value function which is not stationary. Thus, may fail when the opponent is learning or trying to adapt itself based on the history of choices (Shoham et al., 2007).

Classic multi-agent algorithms are derived from Q-Learning (Hernandez-Leal et al., 2019) and keep track of both Q-values and of a probability distribution in each state (Behzadan and Munir, 2017). They often update their Q-values in the same manner as Q-Learning and introduce different ways of calculating the policy π with only knowledge about their action and rewards. Generalized Infinitesimal Gradient Ascent using the WoLF principle (GIGA-WoLF) (Bowling, 2004) keeps track of two gradient updated strategies, one of which is updated faster than the other. *Regret* measures how much worse a policy performs compared to the best static strategy,

and GIGA-WoLF exhibits both no-regret and convergence properties. Weighted Policy Learner (WPL) (Abdallah and Lesser, 2008) has a variable learning rate and allows the agent to move towards the equilibrium strategy faster than moving away from it. Despite not having a formal proof of convergence due to the non-linear nature of WPL's dynamics, WPL's dynamics differential equations are solved numerically showing that it features continuous non-linear dynamics, while experimentally demonstrating it converges in several complex games. However, WPL is biased against pure strategies, and as such Simões et al. (2018a) proposed an extension to WPL, the Adjusted Bounded WPL (ABWPL) using a new update rule based on state counting. Many other algorithms can converge to NE but with unrealistic assumptions like already knowing the game's NE, game structure, or even rewards received by other players (Zhang and Lesser, 2010).

2.4 Deep Reinforcement Learning

Q-Learning memorizes Q-values in a table indexed by state-action pairs (s,a). It becomes unfeasible to store and compute when the state or action space becomes very large. Instead, modern approaches have moved towards approximating Q-values with NNs, parametrized by θ weights, $Q_\theta(s,a)$. States or observations are given as input to the NN model, and the output returns a value (in the case of Q-Learning) or policy. DRL is mainly motivated to tackle large state spaces.

MLP networks can be used to implement Deterministic Policies. To implement stochastic policies, a re-parameterization trick is used, where the NN outputs parameterize some distribution. For categorical policies, the output layer returns logits (raw output values) for each action, converted into probabilities using a softmax. A categorical policy acts like a classifier over discrete actions. A diagonal Gaussian policy has a NN that maps from observations to mean actions, $\mu_\theta(s)$. The neural network may also map from states to log standard deviations, $\log \sigma_\theta(s)$. A continuous policy can be seen acts like a regression and outputs multiple independent continuous actions.

We include a brief tree taxonomy of DRL algorithms from Open AI (see Figure 2.5), where modern fundamental algorithms are classified as model-free and model-based. Here the model is the world transition model, allowing for an agent to plan the next steps as it knows everything that can happen. The issue with this class of algorithms is that the world model is not always available. Model-free algorithms are more sample inefficient, but easier to tune and are currently more popular. Q-Learning or value-based algorithms are sample efficient as past samples can be used, but they are unstable since they rely on the Bellman self-consistency equations. Then, we have policy optimization where we directly find the policy, making the learning process usually more stable but more sample inefficient. Some algorithms mix the constructs of both families of algorithms, exploiting their strengths.

2.4.1 Value-Based Methods

Q-Learning may suffer from instability and divergence when combined with nonlinear Q-value function approximation and bootstrapping. As such, new approaches have been conceived to deal with these issues. Deep Q-Learning (DQN) (Mnih et al., 2015) uses NNs to generalize Q-values

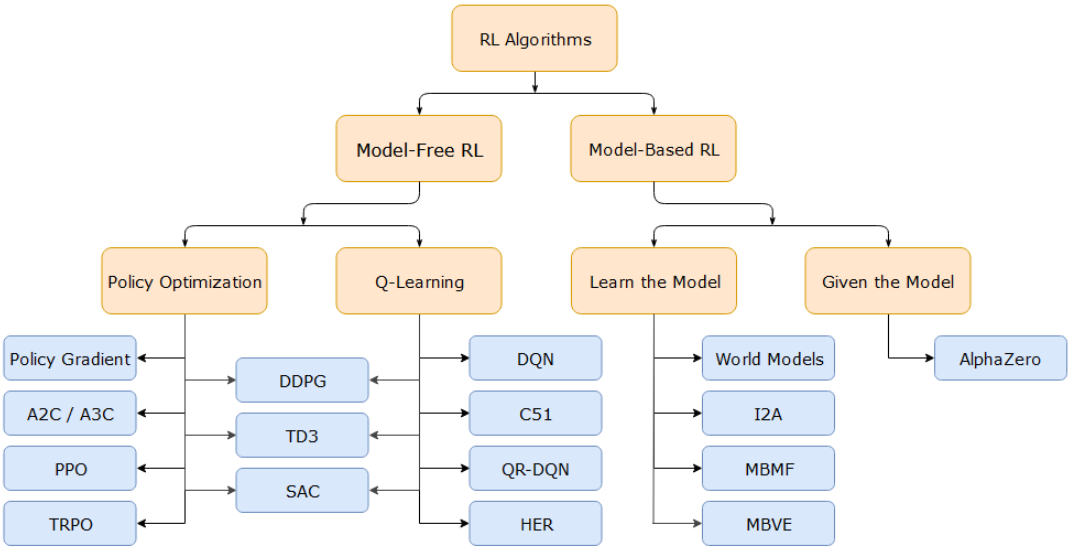


Figure 2.5: Taxonomy of Algorithms in Modern RL.
Source: Introduction to RL, Part 2: Kinds of RL Algorithms, OpenAI Spinning Up.

even in unseen states. It also greatly improves and stabilizes the training procedure with two mechanisms:

- Experience Replay: $\langle s_{t+1}, s_t, a_t, r_t \rangle$ tuples are stored in a replay memory. These samples are reused randomly multiple times in Q-values updates. Removes correlations in the observation sequences, and smooths over changes in the data distribution.

- Periodically Updated Target. Target Q-values are only updated periodically. The network is cloned and the clone is frozen. The clone becomes the new optimization target, eliminating short-term oscillations.

Double DQN (van Hasselt et al., 2016) is an Off-Policy method, where a different policy is used for value evaluation and to select the next action. The primary network is used to choose the action and the target network to generate Q-values for that action. This reduces the overestimation of Q-values, where sometimes higher Q-values were given to sub-optimal actions. Dueling DQN (Wang et al., 2016) tries to improve Double DQN by computing separately V-values and A-values. A-values are called advantage values and are defined as $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$. They are recombined into Q-values on the final layer. The main advantage of this architecture is that does not impose changes across actions.

2.4.2 Policy Optimization and Actor-Critic

Policy-based methods aim to learn a hypothesis distribution $\pi_{\theta}(a|s)$, parameterized by weights θ. This means to know with which frequency must an agent choose each possible action a knowing the current state s. Policy Gradient or Gradient-based search uses the Grad-Log-Prob of a

trajectory to optimize the policy:

$$\nabla_\theta E_{\tau\sim\pi}[R(\tau)] = E_{\tau\sim\pi}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t, s_t) R(\tau)\right]. \tag{2.34}$$

Knowing the value function can assist the policy update. Actor-Critic Learning is constituted of two main components or steps:

- Critic evaluation: updates parametrized Q_θ-values or V_θ-values depending of the algorithm. It reduces variance and stabilizes the learning more than pure policy gradient methods;
- Actor improvement: Updates the policy following the direction suggested by the critic.

Actor-Critic Learning is a hybrid On-Policy and Off-Policy method. The critic follows the expected optimal policy and thus Off-Policy, while the actor follows the current policy and thus On-Policy. Both actor and critic can have their own learning rate.

The main benefit of this architecture is to improve the weaknesses of the On-Policy and Off-Policy methods simultaneously, taking advantage of both method classes' strengths. Off-Policy methods like Q-Learning suffer from poor convergence. Because it operates over value space, any slight change of the value may affect tremendously the policy space. On-Policy methods have a more smooth convergence, but it suffers from high variance and local maximum convergence. Combining both methods allows for more efficient TD updates.

2.4.3 Advanced Policy Optimization Methods

Deterministic Policy Gradient (DPG) (Silver et al., 2014) assumes that the policy is a deterministic decision function. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019) extends Deep Q-Learning by combining it with DPG. DDPG allows Deep Q-Learning to learn in continuous spaces with an Actor-Critic framework. Distributed Distributional DDPG (D4PG) (Barth-Maron et al., 2018) improves DDPG to run in a distributed environment. Twin Delayed Deep Deterministic (TD3) (Dankwa and Zheng, 2020) like Deep Q-Learning and DDQN tries to prevent overestimation of the value function improving DDPG. It adds three new steps: Clipped Double Q-Learning; Delayed update of Target and Policy Networks; and Target Policy Smoothing.

Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) is optimized for parallel computing. Multiple critics learn the value function and multiple actors learn in parallel and are periodically synced with global parameters. A2C (Mnih et al., 2016) is a synchronous, deterministic version of A3C. It awaits that all works are done, updates global parameters, and sends workers again, with everyone following the same policy. Proximal Policy Optimization (PPO) (Schulman et al., 2017) like Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) tries to maximize its performance without changing too much at each update iteration. Unlike TRPO which solves this problem with a complex second-order method, PPO relies on simpler first-order methods. PPO has two variants. The most widely used in the literature is PPO-Clip, which relies on

clipping the policy targets bounding the new policy $\pi_\theta(a|s)$ to old policy $\pi_{\theta_k}(a|s)$:

$$C(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right). \quad (2.35)$$

The second variant, PPO-Penalty uses a KL constraint to hard stop updates and update over time the penalty coefficient. Phasic Policy Gradient (PPG) (Cobbe et al., 2020) tackles a common trade-off in Actor-Critic architecture, of a shared or separated network, while having a shared network may bring benefits by sharing features between value and policy, but may interfere with one another. PPG splits the optimization process of both into separated phases, allowing for greater sample reuse and achieving greater performance than PPO in some test beds.

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is a technique that incorporates entropy maximization to enable simultaneously stability and exploration. It is constituted by an Actor-Critic architecture and Off-Policy formulation that allow the use of precious data. It is based on Soft Q-Learning (Haarnoja et al., 2017).

2.4.4 Asynchronous Adversarial Deep Reinforcement Learning

Both GIGA-WoLF and WPL algorithms have been extended to the deep learning paradigm as GIGAθ and WPLθ (Simões et al., 2018b), based on Asynchronous Q-Learning (Mnih et al., 2016), which is a faster-distributed version of DQN. Asynchronous methods are adequate for multi-agent learning because they rely on parallel threads to gather samples to update their neural networks. DQN, on the other hand, uses a single worker and accumulates samples in a memory buffer. While this makes it more sample efficient, when used in a multi-agent setting, it causes agents to use outdated samples and adapt to opponent strategies that have since evolved.

An asynchronous deep version of GIGA-WoLF is described in Algorithm 1 as GIGAθ. Two policy networks with weights θ_π and $\theta_{\hat{\pi}}$ are used, but only a single policy update rate δ is required.

Algorithm 1: Pseudo-code for a worker thread running GIGAθ using ε-greedy exploration.

Require: Globally, target network update period τ, online and target Q-networks, online and target policy networks, online and target average policy network weights, exploration rate ε, and maximum iterations $T_{\max}$. Locally, online Q, policy, and average policy networks.

for iteration $T \leftarrow 0, T_{max}$ **do**
  Synchronize local online networks as copies of global online networks
  Sample initial state
  repeat
    Execute random action with probability ε, otherwise execute action from policy network
    Sample new state and reward
    Compute Q-network targets with target Q-network
    Compute policy networks' targets with GIGA-WoLF's update equations
    Compute loss of local online Q, policy, and average policy networks
    Accumulate gradients by minimizing the loss
  until terminal state
  Update global online networks weights with the accumulated gradients of local online networks
  Synchronize global target networks as copies of global online networks every τ time-steps
end for

Ensure: A converged Q-network to approximate the value function as $Q(s,a,\theta)$, and a converged policy network to approximate the policy function as $\pi(s,a,\theta_\pi)$.

An asynchronous deep version of WPL is described in Algorithm 2 as WPLθ. Unlike the previous algorithm, a single policy network θ_π is used in conjunction with a Q-network.

Algorithm 2: Pseudo-code for a worker thread running WPLθ using ε-greedy exploration.

Require: Globally, target network update period τ, online and target Q-networks, online and target policy networks, exploration rate ε, and maximum iterations $T_{\max}$. Locally, online Q and policy networks.

for iteration $T \leftarrow 0, T_{max}$ **do**
  Synchronize local online networks as copies of global online networks
  Sample initial state
  repeat
    Execute random action with probability ε, otherwise execute action from policy network
    Sample new state and reward
    Compute Q-network targets with target Q-network
    Compute policy networks' targets with WPL's update equations
    Compute loss of local online Q and policy networks
    Accumulate gradients by minimizing the loss
  until terminal state
  Update global online networks weights with the accumulated gradients of local online networks
  Synchronize global target networks as copies of global online networks every τ time-steps
end for

Ensure: A converged Q-network to approximate the value function as $Q(s,a,\theta)$, and a converged policy network to approximate the policy function as $\pi(s,a,\theta_\pi)$.

2.4.5 Other Learning Techniques

Some environments present sparse rewards. One way to tackle this is to explore new states to find rewards. Intrinsic Curiosity (Pathak et al., 2017) uses a model that is trained to recognize previously visited states, if we visit previously seen states then a penalization reward is given to the policy while visiting novel states is positively rewarded. GAIL (Ho and Ermon, 2016) is an imitation learning inspired by Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), where a discriminator learns to distinguish between actions learned by a demonstrator data set versus the trajectories produced by the policy. Imitation Learning (IL) allows us to rapidly find meaningful policies compared to its RL counterparts and usually is used as a pre-processing step to RL. Multiple reward signals can be combined for more effective learning (Juliani et al., 2020). Curriculum Learning (Soviany et al., 2022) is a technique where the training is organized in session, where we start with a simpler task or data set and increase the difficulty at each stage. For cooperative multi-agent learning, centralized critics and emergent communication are techniques that proved to help swarms of agents to learn how to coordinate (Simões et al., 2020a). For adversarial multi-agent learning, self-play combined with tree search achieved super-human performance achievements in classic board games like Chess, Go, and Shogi (Silver et al., 2017; Schrittwieser et al., 2020) were possible because they are perfect information games. Self-play makes policies train by competing against themselves and previous evolutions, exploiting some kind of transitivity over strategies and avoiding over-fitting against the current or latest policy.

2.5 Meta-Heuristic Search

Meta-Heuristic is a generic search procedure that aims to find a good enough solution to very complex optimization problems in a reasonable amount of time. The main characteristic of meta-heuristics is that they use little domain knowledge, and the procedures are generic enough to be easily applied to many domains. They do not usually offer guarantees of convergence or of finding the global optimum.

2.5.1 Genetic Search

Genetic Algorithm (GA)s are evolutionary techniques that are inspired by Darwin's biology theory of survival of the fittest in nature (Katoch et al., 2020) through natural selection and genetics. They are specific meta-heuristics search algorithms (Hassanat et al., 2019) used for optimization and search problems to find approximate solutions (Sohrabi and Azgomi, 2018), sometimes even optimal solutions to problems with a great number of potential solutions (Sheppard, 2016).

GAs are iterative algorithms. They are initialized with a population of random chromosomes. For each individual, a fitness function evaluates how good or bad they are, and from here the fittest solutions are selected and used to produce the next generations, which will be a mixture of chromosomes from the previous generation's parents. The effectiveness of GAs depends on some key elements of the chromosome representation and operations, they are: selection, crossover, mutation, and fitness function. Figure 2.6 illustrates the basic flow of a GA. Some of the most common selection operators are roulette wheel, random, rank, tournament, and stochastic universal selection. Some of the most popular crossover methods are single-point, two-point, uniform, and scattered. Finally, some of the most popular mutation methods are random, inversion, scramble, and adaptive.

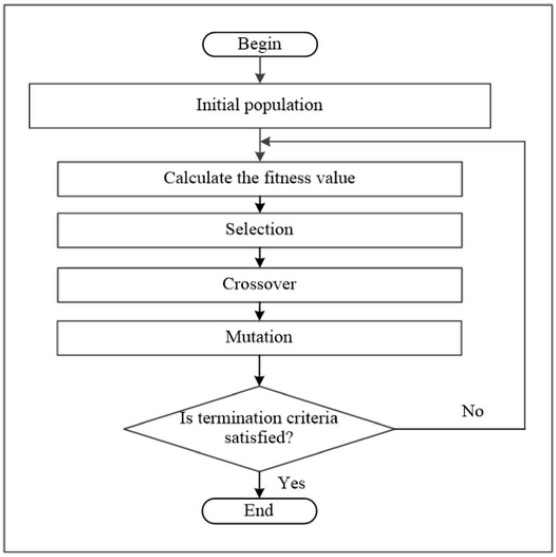


Figure 2.6: Flowchart of the Standard Genetic Algorithm.
Source: Höschel and Lakshminarayanan (2019)

2.6 Discussion

In this section, it is discussed the motivation behind the technological selection that was explored in this chapter and how it is applied to this thesis.

GT establishes the fundamental solutions for strategic interactions, covering topics like dominated strategies and optimal strategies, which are the base framework for our power balance analysis and strategies' restructuring. Our end goal is to find games that satisfy certain balance criteria, for example, having a mixed NE where some strategies have a reasonable chance of being selected. To search for game configurations that satisfy our balance criteria, GA is employed to search for game features. The GA fitness function requires filling out a payoff matrix or computing the NE of the game (distribution model) or sample from policies to discover the usage rate of each strategy. Instead of performing simulations to determine the win rate between each pair of strategies (which is computationally heavy), we apply supervised learning, where a model is trained to predict the win rate of each possible joint strategy of two players. A data set is generated on the fly with player choices' and the sample win rate from a finite number of game runs. Another type of semi-supervision or reinforcement is also explored to learn good matchups - counter strategies are found with GAs and supervision is done over the teams we want to defeat as input and the counter strategies as target labels.

RL is used to train both play-test agents and GM agents in our adaptive difficulty environments. Handicaps will be mechanisms parametrized by the actions of the GM agent, and play-test agents will have to adapt to any possible scenario of handicap configurations. Our research started in simple discrete environments which only required tabular variants of Q-Learning and multi-agent variants, and as the research progressed, it shifted to more complex environments with continuous state and action spaces that required the transition to the DRL paradigm. Some problems were tackled with multi-agent learning, for example with WPLθ and GIGAθ to achieve a battling agent which knows how to strategy over the core rules of Pokémon. When using regular self-play, we employed deep single-agent RL algorithms (PPO). The base RL had to be applied carefully to both our dynamic difficulty and power balance use cases, and in this thesis, we will explore how such can be done.

Chapter 3

Automated Difficulty Balance in Two-Player Games Through Deep Reinforcement Learning

3.1 Introduction

The video game industry is one of the largest entertainment industries in the world (Sepulveda et al., 2019) that has been growing and generating great market values. In 2021, the games market generated a total of 180.3 billion dollars in revenues, increasing 1.4% compared to 2020. Of the different devices to buy and play games, the mobile segment contributed 93.2 billion dollars (52%) to the global market (Wijman). Video games have become one of the main social devices and their consumption has steadily increased over the last few years. The produced content become more generic and the video game player base has become diverse. It is now more difficult than ever for games to offer an ideal experience to players (Oliveira and Magalhães, 2017). Content development now represents a major percentage of the development budget and time of production (Iosup, 2009). There is more incentive than ever to find ways to mitigate video game development costs (Oliveira and Magalhães, 2017).

One of the most important aspects to offer in a game is to provide the players with an optimal experience that is not too hard or too easy, aiming for a state where skill complies to challenge (Baumann et al., 2016; Baldwin et al., 2013), leading to a state of "flow" (Hendrix et al., 2019). Classical approaches for game difficulty design, go from allowing the player to manually choose the challenge level (easy, medium, or hard) to the conception of a difficulty curve by the game designer, where the challenge increases as the player progresses through the game levels (Yun et al., 2009). These static configurations do not consider player performance. This makes classic game design game-centered instead of user-centered (Yun et al., 2009).

DDA is a research field that studies methods to tailor the balance of skill and challenge in video games for human players using real-time software techniques (Hunicke, 2005), adjusting the resources, behaviors, and scenarios of a game so that the player is not bored or frustrated (Zohaib

and Nakanishi, 2018; Pezzera and Borghese, 2020), and in this sense maximizes their engagement (Gonzalez-Duque et al., 2020). Therefore, game designers can ffocus on the development process, leaving the adaptation to the DDA algorithm (Hendrikx et al., 2013). There is the risk that players may find the adjustments unfair, decreasing their engagement (Altimira et al., 2014). It is not easy for game designers and developers to map a complex game state into one variable.

MDDA is mostly unexplored and is harder to tackle (Zhu and Ontañón, 2019); helping or difficult one player may negatively impact the feelings of the players towards the game; also, the difficulty of a game depends not only on the challenges presented by the environment itself but from the skill of the other players which cannot be directly controlled. Given these challenges, we are investigating how can MDDA be implemented. MDDA could present great benefits in its potential applications, such as improving social interaction in Serious Games, games not developed with the sole focus of entertainment, but present enjoyable activities for training, education, rehabilitation, change of attitude and behavior, simulations of real-world processes or persuasive purposes (Adams, 2010; Novak, 2012; Sajjadi et al., 2022).

Game mechanics with the purpose of changing the difficulty of the games to equalize the chances of winning between players are called handicaps. Successful commercial multiplayer games already make use of handicapping systems, including Mario Kart 8 (Nintendo, 2014) where stronger items are given with greater chances to racers behind, or Left4Dead 2 (Valve, 2009) where the intensity of the game is automatically tuned accordingly with the player performance. Some of these games are considered casual (or non-competitive), games where the outcome does not entirely depend on skill but also on chance.

Current video game complexity justifies the use of ML to solve difficulty balancing tasks, many games are represented in a high dimensional number of features, also the variety of game scenarios, and the possible population of players piloting the game is innumerable. There is also a lack of general purpose MDDA tools, and many techniques are restricted by the game genre. We propose a framework for difficulty balance in adversarial two-player games, leveraging the capabilities of RL. Two surrogate agents represent the players of the game, each possessing a distinct skill proficiency and a GM agent trained to act strategically to best deploy handicap mechanisms to re-tune the relative difficulty of the game in the function of the players' performance.

The first iteration of this framework produced a family of surrogate agents exploiting the evolutionary property of RL where we stored early versions, and order the population by total accumulated reward. The intention was to use these agents to generate enough combination of situations for a GM agent to act over. The GM got an incentive to minimize the win rate of players by memorizing the wins and losses for each player as a meta feature. We formalize this problem as a game adaptation problem, where a base game is mapped to a target game complying with some criteria or constraints. However, this approach presented multiple limitations, for example, the surrogate player agents were not trained in states where handicaps were deployed. Just taking the win rate into account does not allow for the measurement of the in-game pacing and performance of each player. All this process was formalized in what we call the Implicit Model, a MDP for MDDA.

Therefore, some improvements were required to measure and consider the progression gap

between both players. In the improved framework, a game designer can better define a set of weighted aesthetic criteria that assert the quality of a gameplay trajectory. We opted to test for criteria that promote dramatization, leaning toward rubber-banding effects during gameplay. The new training pipeline consists of three steps: 1) production of a population of player agents with distinct skill levels; 2) training a player performance estimator that aesthetic criteria use to label the quality of game runs; 3) training of the GM agent. In this scenario, at step 1, the agents are trained with a deployed random GM policy, activating handicaps to the agents to be able to cope with them to a certain extent. Sub-optimal agents are also configured with another technique, instead of safekeeping previous iterations on training, we instead control the final policies by injecting errors or noise into the agents' actions. All this process was formalized in what we call the MDDA-RL framework, an enhanced MDP for MDDA.

Since the Implicit Model was tested early with simple grid-world or basic 3D settings, to evaluate MDDA-RL, we developed a test suit of MDDA game environments, comprised of player agents and a GM agent, the first being imperfect information turn-based card game and the second being a real-time strategy survival adversarial game where two avatars must defeat more enemies than each other as quickly as possible. Both games allow for Human vs Human (HvH), Human vs AI (HvA), and AI vs AI (AvA) gameplay.

The trained GMs are evaluated by measuring the resulting degree of the criteria achieved with different player agents and by conducting a user experiment, where small sessions were played, and participants were requested to fill their experience in a small questionnaire, to understand the impact of MDDA techniques during gameplay. The study provided some positive hints for such type of mechanisms but also contained some experimental design flaws, where we just had human users play against a strong AI surrogate agent, where handicaps were mostly deployed in favor of the human user, potentially creating a bias towards positive experience with handicaps. Some users reported the challenge given by the unfairness of the handicaps to be engaging.

The main contributions of this chapter are therefore:

- Study of two methods to create a population of sub-optimal agents.
- Design of a preliminary RL framework for MDDA, the Implicit Model.
- Extension of the Implicit model to our definitive MDDA-RL framework, where a GM learns to improve following aesthetic criteria, creating rubber banding effects.
- Design of a game test suit for MDDA tasks.
- Validation of our models with both surrogate AI agents and human users.

The remainder of this chapter is organized as follows: in Section 3.2 we discuss related work in DDA, MDDA and related fields like GameFlow theory, PCG oriented to quality of content, Player Modeling, and Affective Computing, and some previous DDA studies with users; in Section 3.3 we discuss our preliminary MDDA model, the Implicit Model, and we conduct a user experiment with a simple 3D game and understand the motivations to be improved; in Section 3.4 we present our

definitive model, the MDDA-RL framework, illustrating its architecture; in Section 3.5 we present our adapted aesthetic criteria set; in Section 3.6 we explain the training pipeline of the MDDA-RL framework; in Section 3.7 we present our game test suit environments; in Section 3.8 we discuss the results of MDDA-RL training of GM agents and their performance over our developed test suit; finally, in Section 3.9 we take our final conclusions of this chapter and do some further discussion.

3.2 Related Work

3.2.1 GameFlow

Flow Theory from the psychology field, defends that a state of Flow exists when performing an experience so gratifying that a person would be willing to do it for its own sake, as difficult or dangerous as it is for him or her. Flow happens when performing a goal-oriented task, bounded by a set of rules, requiring appropriate skills. This setting is comparable to games, where a player is required to have sufficient skill to play, must receive immediate feedback while playing, and has a clear comprehension of the game goals. GameFlow is a model that explains what enhancement through gameplay aims to achieve, it's the application of the Flow Theory to games (Sweetser and Wyeth, 2005). GameFlow maps eight different game elements to Flow Theory elements:

- The Game $\implies$ A task that can be completed.
- Concentration $\implies$ Ability to concentrate on the task.
- Challenge Player Skills $\implies$ Perceived skills should match challenges, and both must exceed a certain threshold.
- Control $\implies$ Allowed to exercise a sense of control over actions.
- Clear goals $\implies$ The task has clear goals.
- Feedback $\implies$ The task provides immediate feedback.
- Immersion $\implies$ Deep but effortless involvement, reduced concern for self and sense of time.

3.2.2 Dynamic Difficulty Adjustment

DDA approaches aim to provide a personalized experience, by adapting the difficulty of a game to each player's skill level. The adaptation with DDA algorithms require three basic needs (Andrade et al., 2005): i) The game needs to adapt and identify as soon as possible the player's level; ii) The performance of a player needs to be tracked; iii) The adaption of a game should not be understood by the player and must stay believable. The creation of DDA systems is not a trivial task, and their implementation is highly dependent on the game genres(s) (Dziedzic and Włodarczyk, 2018), so in the literature, there are several techniques. A common aspect among all techniques is that the

difficulty felt by the players is estimated at a given moment by explicit or implicit heuristic/challenge functions. These functions map a game state to a value that predicts whether a player is struggling at a specific moment (Zohaib and Nakanishi, 2018).

Some of the main techniques that exist to adapt difficulty in single-player games: 1) Probabilistic Methods, used to model complex systems with high uncertainty. They are related to reasoning tasks because probabilistic estimations are done about the random variables through the construction of joint distributions that represent their interrelationships (Koller and Friedman, 2009); 2) Rubber Banding, is an approach that aims to ensure that the players have equal chances of winning. The system regulates the performance of the player and his opponent(s) and helps the weakest player by giving him a boost and subjecting the stronger player to a disadvantage (Pagulayan et al., 2012; Missura, 2015; Kristan et al., 2020); 3) Hamlet System, is a method used in games that contain an inventory of items. The Hamlet system uses statistical metrics to monitor the performance of the player, and when detects/predicts that a player is having difficulties accomplishing the tasks, the game inventory is adjusted to control the difficulty of the game; 4) Dynamic Scripting, is an online, unsupervised technique for adapting the game AI to the human player's skill level (Spronck et al., 2006); 5) Monte Carlo Tree Search, is a method that combines the accuracy of tree search with random sampling. This approach builds in an incremental and asymmetric way a search tree based on the outcomes of random samples taken from the decision space to find the best decisions in a particular domain (Browne et al., 2012); 6) Reinforcement Learning, is a method that designs intelligent agents that, by trial and error experience in their own environment, learn or adjust new behaviors (Duarte et al., 2020); 7) Neuroevolution, is an approach that uses a NN to control agents and a genetic algorithm to refine the weights of the NN; 8) NN + UCT, is used to create adaptive intelligent agents, where simulation data from the UCT algorithm is used to train a NN; 9) Genetic Algorithms: are adaptive meta-heuristics search algorithms that follow Darwin's theory of the fittest in nature (Sohrabi and Azgomi, 2018; Katoch et al., 2020). They represent a good solution to solve complex problems because they have the advantage of consuming less computational resources and time (Duarte et al., 2020); 10) Affective computing: refers to the techniques that use biofeedback tools to track the emotional states of players and adapt the gameplay accordingly (Moschovitis and Denisova, 2022).

Demediuk et al. (2019) refers to three types of DDA agents aimed to achieve game fairness in human vs AI playing, trying to adapt their actions to human performance. The first is Challenge Sensitive Action Selection where an agent is trained with RL (Andrade et al., 2005) and actions are ranked using their Q-values and based on the performance of the opponent the most suited action is selected based on their raking. Missura and Gärtner (2008) also explores the ranking of available actions. It is considered that a player has a set of available actions/strategies and is playing against an AI agent. The actions have a natural ordering or raking, there are better strategies that the player can choose against the opponent AI, and the same for the AI agent. This means the AI agent can evaluate the player's performance by the actions taken by him and adapt its own actions to the player. The second is Reactive Outcome Sensitive Action Selection where an agent employs Monte Carlo Tree Search (MCTS), the score of the nodes is proportional to

a player metric (normalized between 0 and 1) and the agent policy is to choose a node with a score close to 0.5 (Demediuk et al., 2017). The third, Adaptive True Reactive Outcome Sensitive Action Selection, is an extension to the previous method, improving the exploration of weaker nodes (which sometimes are the most desired) because MCTS promotes stronger nodes the most. To solve this issue, the scoring metric is adjusted.

Beau and Bakkes (2016) focus on studying the application of MCTS by generating a population of surrogate agents to the generation of a population of agents in an asymmetrical game in which agents have a different action set, and use them to produce gameplay data to balance the asymmetric actions by approximation. Pratama and Krisnadhi (2019) proposes the use of a MCTS variant, Monte Carlo Exploring Start (MCES) for DDA. This work is evaluated in a Role Playing Game where each player controls three characters and their performance is evaluated by the differential of the summed hit points from the entire team of a player, the score is given as a reward to the MCES algorithm.

Missura and Gärtner (2011) models the DDA problem as a meta-game between the player and the GM, where the GM tries to predict the most appropriate setting playing over a partially ordered relation of 'more difficult than' over actions. The game proceeds in three steps:

1. GM chooses a game state with a certain difficulty;
2. Player plays one episode of the game;
3. GM receives feedback and adapts its choice in the first step.

The work is driven by the following three properties:

- Universality - should be applied in the most possible range of games;
- Non-intrusiveness - should work in a seamless way, transparent to the players;
- Feasibility - should have performance guarantees.

More recent approaches make specific use of RL. Zhang et al. (2018) proposed to extend the RL model, where the environment is not given, but controllable and learnable in a parameterized fashion. They model the problem as a dual MDP. The agent aims to maximize its cumulative reward, while the environment minimizes the reward for a given optimal policy from the agent. This work operates at an episodic level, achieving the generation of the initial condition of the environment vs DDA. Noblega. et al. (2019) makes the use of DRL in a 3D fighting game to develop an agent that adapts to the opponent's skill. They propose the use of a reward function that is based on a balancing constant. It creates multiple balance states, which in the case of the author's work they call Subjugate Punishment (when under-performing), Conservation Reward (balanced), and Rebellious Punishment (over-performing).

Mori et al. (2019) presented a framework that analyses multiple Experience Manager (EM), and identifies five common components in current computation techniques, Decision Constraint Function, Objective Function, Rollout Function, Estimated Player Policy, and Feature Vector. These components provide a way of making possible comparisons between the design of EMs.

3.2.3 Multiplayer DDA

Fairness in multiplayer games can be defined as a game where, for all players or teams, the winning ratio is statistically equal or nearly (Wu et al., 2016). MDDA explores to adapt in real-time a game considering the playing performance between two or more players. Furthermore, giving some sort of disadvantage to the highest performing player may be seen as punishment and may affect negatively that player's engagement, and improving, or giving some sort of advantage to the less skilled player may encourage the player without removing the rewarding feeling of the highest skillful player (Baldwin et al., 2013). DDA has received a lot of attention in single-player contexts, and so there is a huge gap in studies with multiplayer games (Baldwin et al., 2013; Silva et al., 2017). Multiplayer games offer challenges in their competitive and cooperative settings, and game balance/adaptation can be seen as more demanding than in a single-player game (Baldwin et al., 2013).

Cechanowicz et al. (2014) identified four methods of achieving video game balance in multiplayer scenarios: i) Matchmaking, which consists of grouping players with similar skill levels; ii) Asymmetric rules, each player has different roles or takes that are compatible with their level of skill; iii) Difficulty adjustment, the difficulty of the game matches to each player capabilities; iv) Assists, which either guide or compensate players for their lack of skill. Baldwin et al. (2013) proposed a MDDA framework to tailor the difficulty between two or more human players, by assisting less skilled players. The components allow for the abstraction of MDDA instances regardless of genre or mechanics. This framework is composed of seven components:

- Determination decides whether to use the MDDA instance using past performance or actual performance;
- Automation identifies if it is applied manually or automatically;
- Recipient indicates if the target of the instance is an individual or a team;
- Skill dependency indicates if it is required for a player to play with a certain degree of performance to activate the instance;
- Action required indicates if the instance must be initiated through some kind of interaction from the player;
- Duration identifies the number or timing of use of an instance if it can be used once per game, multiple times per game, or periodically;
- Visibility indicates which players are informed about the occurrence or use of an instance.

Zhu and Ontañón (2019) presented the open main challenges in Multiplayer EM: i) Player Model Aggregation, how can multiple player models be aggregated, dynamically group players, adapt to multiple player requirements, and evaluate the EM effectiveness; ii) Dynamic Multiplayer EM over Time with Dynamic Aggregation Mechanics (where for example a static strategy where

the majority preference prevails, and could disengage the minority) and or Dynamic Grouping, and infer the player's goals; iii) EM-driven Procedural Content Generation, in terms of "controllability", generating content to suit groups of players; iv) Evaluation Methods.

Karavolos et al. (2017) presents an algorithm that can classify a match as balanced or favorable to one of the teams by using convolutional neural networks, represented by an image using players' weapons as parameters. 39 level generators were used to distribute small and large objects through the map to generate sufficient data for ML. Also for the data volume needed for deep learning, AI agents with pre-built behaviors were used.

In terms of work and techniques, Silva et al. (2017) explore the use of an adversarial agent in a MOBA (Multiplayer Online Battle Arena) game that adapts its difficulty to a human player. Three different agents were built to detect the unbalancing moments in the game, and player performance was tracked by using game features that were evaluated using a heuristic function.

Current MDDA frameworks are still in a theoretical stage, and single-player DDA techniques extensions to the multiplayer setting remain unexplored. Our technique concretely makes use of deep RL to align a two-player game pacing using rubber-banding effects, by motivating the EM agent to deploy the most appropriate handicaps.

3.2.4 Procedural Content Generation

Moreover, other techniques can assist DDA (Reis et al., 2019a): PCG and Player Modeling. PCG, Player Modeling and Affective Computing are research areas that study the use of computational models to automatically produce game content and to obtain a description of the player in games.

PCG automatically generates game content through an algorithm (Togelius et al., 2011), but its application to game balance has been limited (Togelius et al., 2013). But more recently, PCG evolved to generate personalized content for players, motivated by the previous production of low-quality game content that led to player disengagement (Yannakakis and Togelius, 2011). PCG may be divided as experience-driven and search-based (Xia and Anand, 2016). Experience-driven PCG does a more controllable generation process (Yannakakis and Togelius, 2011). Search-based tries to use an evaluation function of the content and gradually choose the most appropriate content (Togelius et al., 2011). PCG methods are applied to generate procedural learning environments and new content using search-based approaches with an evaluation function (Shaker et al., 2016).

PCG applications of deep designer agents have emerged in games and are now called PCGML (PCG via ML) (Summerville et al., 2018). PCG methods are applied to generate procedural learning environments and new content using search-based approaches with an evaluation function (Shaker et al., 2016). PCG via RL (Khalifa et al., 2020) explores how RL can be used to train level-design agents. In Level design the content generator itself is learned, therefore PCG is done iteratively instead of in a one-shot fashion. Generative Playing Networks Bontrager and Togelius (2021) designs levels to play itself. It is constituted by an agent that learns to play game levels and a generator that learns to create playable game levels and does not require domain knowledge. Shi and Chen (2018) try to tackle two main issues on PCG: of quality assurance and DDA. They develop an algorithm to address both issues by using the classic Super Mario Bros (Nintendo,

1983) platform game as a test bed. They proposed a hybrid approach for generating Constructive Primitives (CPs) using simple rules and a learning-based method. DDA is done in response to the player's performance in the last played CP and affects the next CP accordingly.

Match balancing is one of the main design issues in multiplayer shooting games. An algorithm for matchmaking is usually used to form balanced teams considering their skill, but not in all games this can be applied, for example, when players form their teams. As such, Lanzi et al. (2014) proposed a technique based on search-based PCG with a genetic algorithm. They use an open-source shooter game, Cube 2[1] as a test-bed. It is used as a fitness function that evaluates the balance between the score of the players in the function of the number of deaths and self-deaths. Two bots were used in a series of experiments using the same or different shooting weapons (which have different characteristics).

3.2.5 Player Modeling

Player Modeling approaches can be used to express, predict, or detect players' behavior or emotions in a game (Farooq and Kim, 2015). Models of the players can be constructed by model-based/top-down or model-free/bottom approaches (Yannakakis and Togelius, 2018). Top-down approaches are built in solid theoretical frameworks to explain phenomena by hypothesizing models based on other domains such as Flow. Model-free approaches create player models that rely on the data left by the players in the game that are gathered and processed to make assumptions about the player's behavior.

Missura and Gärtner (2009); Mladenov and Missura (2010) explore the modeling of playing types based on a supervised learning method. Data is provided through gameplay during the testing stage. It is assumed that there is a finite number of player types. The methodology consists of three steps: cluster the recorded game traces; average the supervision over each cluster; and learn to predict the right cluster from a short period of gameplay. Oliveira and Magalhães (2017) aimed to develop a new methodology that helps developers to better understand the players. The work considers four major concepts: Developers, Game, Player, and Player Context. The player model is divided into player characteristics, which describe information about the player, containing information about the way that the player interacts with the game. For example, the information could answer if the player is casual or hardcore. The context model stores information about how the game is played. For example, player location in recent mobile augmented reality games. The game model has information about how the player plays the game, and what records he or she has achieved. Zook and Riedl (2012) explore tensor factorization in Action Role-Playing Game, developing a temporal player model for Content Tailoring. Content Tailoring generalizes DDA which in contrast performs online and offline optimization. A three dimension tensor is decomposed into player factors, spell type factors, and time factors. Similar to recommender systems, where user preference is stored for each item, here the users are the players and the items are the spell types. Bontchev and Georgieva (2018) present a model for in-game recognition of four

[1] `http://sauerbraten.org/`

playing styles (Competitor, Dreamer, Logician, and Strategist) based on Kolb ((1984) experiential learning theory. They use linear regression to classify players' gameplay styles. The model coefficients are found by heuristics and optimization and are effective to recognize play styles with high accuracy.

3.2.6 Affective Computing

Affective computing is a research field that studies how video games should react to players' emotions and also how they should provoke certain emotions in players (Lara-Cabrera and Camacho, 2018). The three main steps are: Measure human emotions, how to adapt the game to those emotions, and finally provoke them. A player model or profile can be used to predict the affective state of the player (Bakkes et al., 2014).

Lara-Cabrera and Camacho (2018) propose a taxonomy for affective games based on the type of feedback, direct and indirect, and the scope. Direct feedback is provided by biometric sensors. Indirect feedback is inferred by how the player is maneuvering the game controllers or how is playing the game. The first task is to detect and measure the affective state of the player. The second task is the adaptation of games to the affective state of the player.

3.2.7 Theoretical Frameworks

FlowAI (Cruz and Uresti, 2017) is a generic conceptual flow framework for Game AI, that unifies multiple approaches that suit game AI. The framework can be resumed in three modules: player modeling, dynamic difficulty scaling (DDS), and immersion. Player modeling is extracted from the player-game interaction. The DDS module adapts tangible gameplay features that might change the perceived level of difficulty. They split those features into four classes, NPC (Non-Playable Character) behavior, Quest or Scenario, Controls, and Game Mechanics. The immersion module focuses on breaking all the barriers that could break player immersion. For example, while DDS could choose the best mechanics for NPCs, the immersion module would make them have believable behavior.

PEAS (Snodgrass et al., 2019), is a recent theoretical framework for personalization through the adaptation of the Player, Environment, Agents, and System. It identifies and categorizes low-level components of games that can be personalized in terms of design: player interaction, feedback, rewards, and learning. They also discuss that in contrast, macro design worries about the game as a whole (genre). They propose guiding questions that will guide designers and researchers through their personalization choices: Why?, How? and What?

3.2.8 DDA Studies with Users

Regardless of the player mode, a major identified limitation is that most works do not evaluate the mechanisms implemented to adjust the difficulty of a game with human players or if they do relevant conclusions cannot be stated due to the small sample size of humans under study, and therefore it becomes very challenging and hard to compare the effectiveness and efficiency of the

works for practical application. The objective of DDA strategies should be to study the stimulus of activity in human players.

Rogers et al. (2018), conducted experiments with human players in an augmented reality tabletop game to study the effect of different balancing mechanisms on player experience. The authors concluded that players prefer some control over the game balance, that player experience is not influenced by the perception of fairness, and that information in the game space can cause a negative effect on a weaker player. Ang and Mitchell (2019) performed a similar study to test how difficulty adjustment should be presented to the players, 1) by the computer system or 2) by providing self-control over difficulty; and when these choices should be available to the players (once, periodically or constantly). The results demonstrated that the integration of choices in the game mechanisms improved the player experience and that these choices should be given periodically.

DDA was studied over digital domains (video games) and physical domains (sports). Altimira et al. (2017) uses augmented reality over a table tennis game, and studies how static and dynamic adaptations affect player engagement, contributing to the field of Human-Computer Interaction. In our interpretation, the questions and results remain pertinent to other video games in general, and therefore worthy of discussion. The authors try to understand how adjustments over sports equipment change player engagement. The main question the authors aimed to answer was the following: *How does game adjustment design that alters the sports equipment statically and dynamically affect game balancing and player engagement in a non-parallel game?* In our interpretation, instead of sports equipment, game-related artifacts, rules, or other balancing questions could be replaced in the question and easily adapted to other video games. The main question was then split into sub-questions by the authors:

- Do different game adjustments impact game balancing differently?
- Do different frequencies of the updates in-game adjustments impact game balancing differently?
- Do different game adjustments impact player engagement differently?
- Do different game adjustments impact player engagement differently depending on players' skill status (more skilled player or less skilled player)?
- Do different game adjustments impact player engagement differently depending on the frequency of the adjustment, i.e. static and dynamic?
- Is there an interaction effect between the different game adjustments, frequency of updates, and the player's skill status?

The conclusions from this work are that frequency can impact player engagement in physical non-parallel games. Another important result is that players with similar skills are engaged by the increasing game difficulty, for example, reducing the table size (or a valid area size with augmented reality) when increased performance by both players. Cechanowicz et al. (2014) also ask relevant questions for the evaluation of an automatic balancing system with human players:

- Does balance work?
- Does balance improve novice experience?
- Does balance detract from the expert experience?
- Is balancing preferred overall?
- How do the adaptation schemes differ?

They conclude that balancing techniques can be successful in racing games, and reducing the difference in performance was positive for novice and expert players. For our line of work, we extract important questions to evaluate our work in the human player domain for future work.

3.3 Preliminary Work

In this section, we discuss our earlier studies towards building a general-purpose RL framework for the automatic deployment of handicaps in two-player games.

Our first step was to formalize the problem of game adaptation (or transformation) as a mapping between a base game to a target game restricted to a set of adaptation constraints. This differs from generational content, where the input of the game content is none or random noise while adapting a game we must input a base game and the output game will share most characteristics with the base game but with newly added or changed functionality. For the purposes of difficulty balance, we perform game adaptation with the incorporation of a meta agent that assumes the role of GM and in some cases the incorporation of new handicap mechanisms. The adaptation will result in a game that incorporates DDA maintaining skill-challenge balance.

Initially, the GM performs no valuable actions. Therefore, we abstract the difficulty balance task as a meta-game, a game that when played changes the rules of another base game. We can observe this new meta-game as a traditional MDP and RL algorithms can be applied to learn how to solve this task. Inside the base game are two player agents that will act traditionally, and the GM must balance their chances of winning. We call this the Implicit Model, as the GM is a meta agent abstracted of the base environment while acting.

The validation of the Implicit Model is performed over a simple 2D grid world labyrinth environment and a simple 3D capture game. The latter was used to conduct a small user study to get preliminaries on how intelligent handicapping affects the experience of human users.

3.3.1 The Game Adaptation Problem

Games can be characterized by their gameplay, which we consider depends mainly on the following traits: rules, tools, aims, and feedback (Weitze, 2014). Rules define the structure of the game. Rules may comprehend meta-rules, which define how other rules may change, and immutable rules, rules that cannot be changed. If a game adaptation complies with every game's meta-rules, the resulting game may provide a more seamless sensation of change to human players. For an

adapted game and its base game to be similar, the adapted game must not change any existing immutable rules. It is usual that meta-rules are immutable rules. Tools are components a player manipulates to interact with the game, like physical objects, tokens like playing cards, or intangible items such as scored points. Aims are middle-term objectives that the player must achieve to progress in the game (Arnab et al., 2014).

We consider that games can be compared in a general sense with a degree of closeness d of their properties. Games are similar if they share similar rules and aims. Tabletop Chess and Tabletop Checkers are similar, and football and chess are less similar.

A game adaptation $\mathscr{A}$ can be formally defined as a mapping between two games G, G', $\mathscr{A} : G \xrightarrow{C,R} G'$, that should preserve every immutable rule R of G, $R(G) = R(G')$, meaning $d(G,G') < \varepsilon$, for some small ε and optimize adaptation criteria C. We can also define the optimal adaptation problem as an adaptation problem where G' must be as similar as possible to G while still complying with R. In our particular case, we aim to use game adaptation where C promotes the presence of balance states over G'. Adaptation strategies have two primary goals according to Xia and Anand (2016): i) adjust behavior rules; ii) generate adaptive game contents. First, numerical values or rules can be changed. In the second, new game content is generated, for example, using PCG algorithms. In our definition, we instead consider adaptation when content is added or changed from a base game, versus a game generated from zero.

For example, the input game G can be a maze game where the player has the goal to reach the exit, and the resulting output game G' just has the map altered (some new paths opened on the maze). Both G and G' compromise the same rules win condition, and basic structural components, just the maze map differs in both games. If G rules do not compromise a statically defined maze (or if G compromised a rule that would allow changes to the maze) both are the same game and therefore similar.

3.3.2 Implicit Model

We formalize Meta MDP as a base MDP with the inclusion of a meta state space $M = \{s_{m_1}, \ldots, s_{m_k}\}$, which exists alongside the base states $S = \{s_{p_1}, \ldots, s_{p_n}\}$. We have set of joint states from player agents base states and meta states $S' = S \times M = \{s_{p_1}, \ldots, s_{m_1}, \ldots, s_{m_k}\}$. M must be observable by at least one meta agent. A reward function for a meta agent r_m computes the reward function and its output depends on the meta state and base states alike, $r_m(s', a) : S' \times A \to \mathbb{R}$. The meta agent perceives the meta environment as a standard environment (see Figure 3.1). This results in a heterogeneous Multi-Agent System (MAS) since the reward functions and space states from the base agents and the meta agent are distinct.

Our Meta MDP follows what we define as the Implicit Model (Reis et al., 2020) (see Figure 3.2), the meta agent acts over an extended meta environment, where inside runs a traditional MDP, which the meta agent is abstracted of. The base agents observe just the traditional state space S, and their environment reward function depends on just from S, while the meta agent receives both S and M states, abstracted to him as just S', and its reward function also depends on M besides S. The interior base environment receives the joint action from all agents plus the meta

agent to update the world model. The base MDP agents perceive also the meta agent as part of the environment.

Our proposal for difficulty balance is to perform two sequential adaptations. First couple to an input game G a meta agent resulting in G', with $\mathscr{A} : G \xrightarrow{\emptyset,\emptyset} G'$. In G' the meta agent performs no actions, a null policy, so G' will be functionally identical to G. Then the meta agent will be trained on how to best perform changes by interacting with meta-game G', resulting in a new game G'', $\mathscr{A} : G' \xrightarrow{R,C} G''$, complying with rules R, and optimizing C. In this sense, G' can be seen as the meta-game where the meta agent plays over. From here after we will denominate the meta agent for difficulty balance as the GM agent.

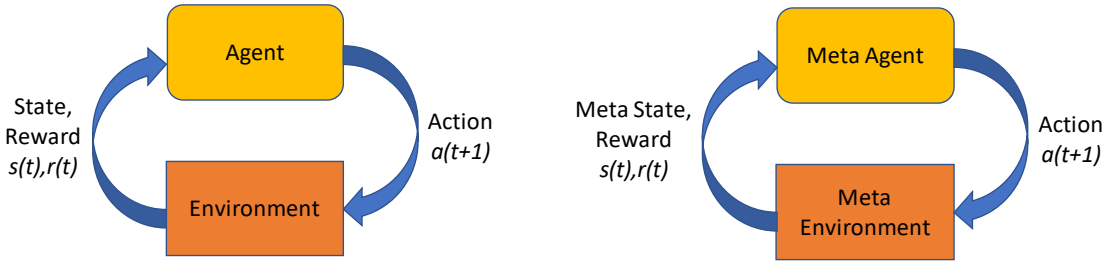


Figure 3.1: Traditional MDP vs Meta MDP. In practice, a Meta MDP has the same structure as a traditional MDP.

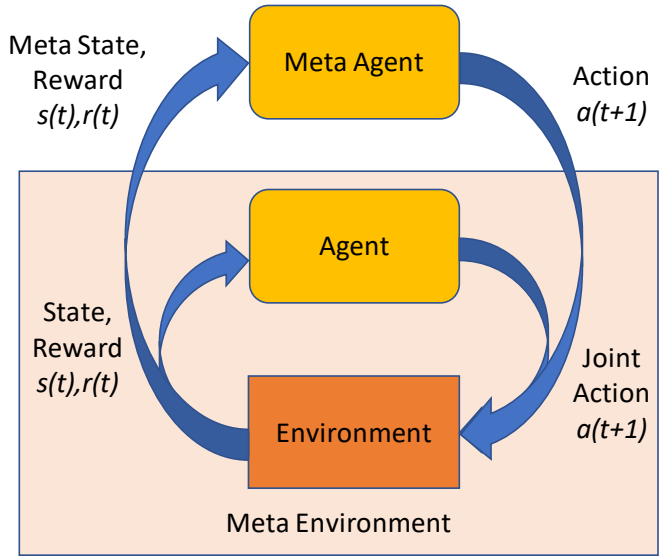


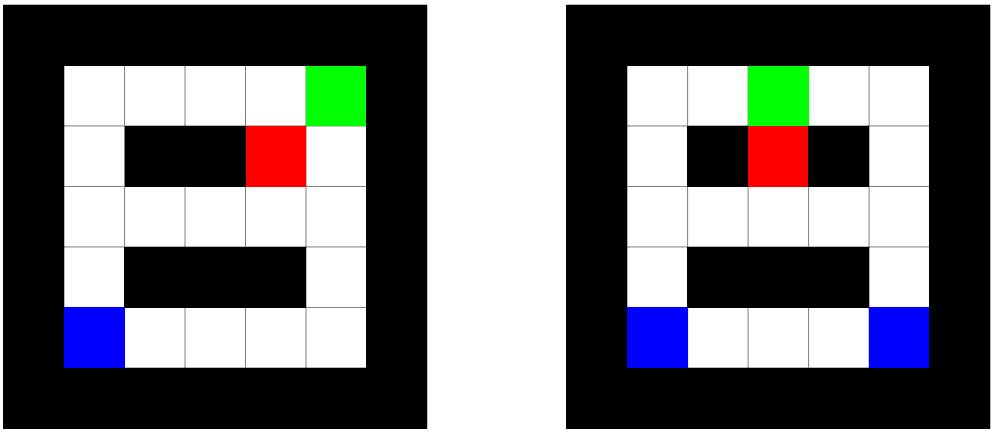
Figure 3.2: Implicit Model MDP. The traditional MDP is abstracted to the meta agent inside a meta environment. The meta states are only propagated to the meta agent. The environment receives the joint action from agents and the meta agent.

3.3.3 Training Regime

ML techniques require large volumes of data, and the scenario of difficulty game balance is no different. We propose the automatic generation of a population of sub-optimal surrogate agents, where each member has a distinct skill in beating the game, and agents are used in multiple combinations to reproduce multiple scenarios resulting in synthetically generated gameplay data (Reis et al., 2019b). RL agents learn in a trial and error fashion, and evolutionary phases of the agent training are preserved. To describe and compare the generated population, we use a population trace. It is a monotonic growing succession indexed by the agents' ID numbers, and the co-domain is the skill level of each agent. Two main attributes of the population are of interest to maximize: Δ or the difference between max and min skill; and Γ or the skill density or variety. The purpose of using an agent population is to be able to stage scenarios in the most possible combinations of possible states, creating a generalized training regime for the GM.

3.3.4 Case Study 1 - Balanceable Maze

The Balanceable Maze environment (see Figure 3.3) is a simple grid world labyrinth environment, where the player (blue) has the objective of reaching the goal state (green), navigating through free spaces (white) while avoiding walls (black). The GM will control a moving wall (red) to restrict the player's movement (see Figure 3.3a). In the multi-player scenario, two players compete to reach the goal state first (see Figure 3.3b). In both single and multi-player cases, the players have a step limit to reach the goal state.



(a) Single Player Setting. (b) Multi Player Setting.

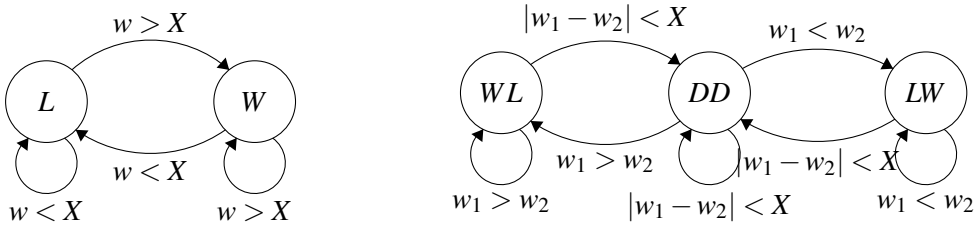
Figure 3.3: Balanceable Maze Environment.

In the single-player scenario, we consider two balance objectives: i) if the win rate w of the player is lower or equal to arbitrary threshold X the GM should remain still; ii) if w is more significant than X the GM should make it more difficult for the player to reach the goal state. This is translated to two balance states, Lose L when the win rate is equal or lower than X, and Win W when the win rate is higher. The balance state machines result as illustrated in Figure 3.4a.

Table 3.1: Balance Reward Function (Single Player Setting).

State W	State L	
t	t	$t \wedge c$
	$s = s_0$	
-1.0	1.0	-0.5

In the multi-player scenario, we consider instead three balance states which correspond to a game fairness state where the win rate of both players is similar and two states where the win rate of one player is higher than the other. The balance state machine is illustrated in Figure 3.4b, with states WL or Win Lose, DD or Draw, and LW or Lose Win.



(a) Single Player Setting. (b) Multi Player Setting.

Figure 3.4: Balanceable Maze's Meta-State Machine.

In the single-player scenario, we want to reward the GM when it remains still in the L balance state, and punish the GM if the player reaches the goal state in the W balanced state. Given $s \leftarrow$ GM state, $s_0 \leftarrow$ GM initial state, $c \leftarrow$ collision event with the wall, $t \leftarrow$ terminal event, $r \leftarrow 0.0$ balance reward, GM agent gets a non-null reward under Table. 3.1 state, events, and conditions. In the multi-player scenario, in the DD balanced state we want to reward the GM for being still. If the first player has a greater win rate than player two (WL state), we want to reward the GM for blocking player one and allow player two to reach the goal state and punish him if he lets player one reach the end. The case where player two has a greater win rate than player one (LW state) works analogously. Given $g_0 \leftarrow$ goal event player 0 (player one reaches the goal state), $g_1 \leftarrow$ goal event player 1, GM agent gets a non-null reward under Table. 3.2 state, events, and conditions. In terms of the GM's action space, just like the players, is allowed to move up, down, left, and right.

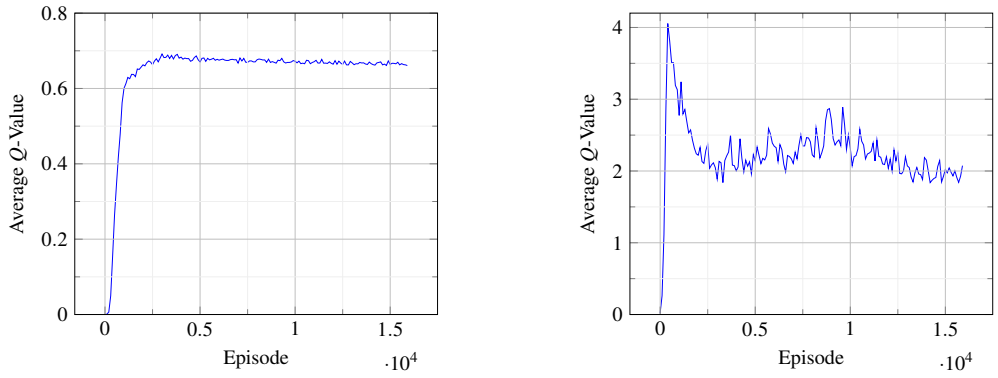
The objective of our experiment is to train the GM in both single-player and multi-player settings and observe the desired behaviors from the GM. For each case we first generate a population of player agents with different proficiency, resulting in the evolutionary stages of player agent

Table 3.2: Balance Reward Function (Multi-Player Setting).

State WL			State LW			State DD		
		c			c			c
	g_1		g_0	$\neg g_0 \wedge g_1$			$s = s_0 \wedge (g_0 \vee g_1)$	
-1.0	1.0	-0.5	-1.0	1.0	-0.5	-1.0	1.0	-0.5

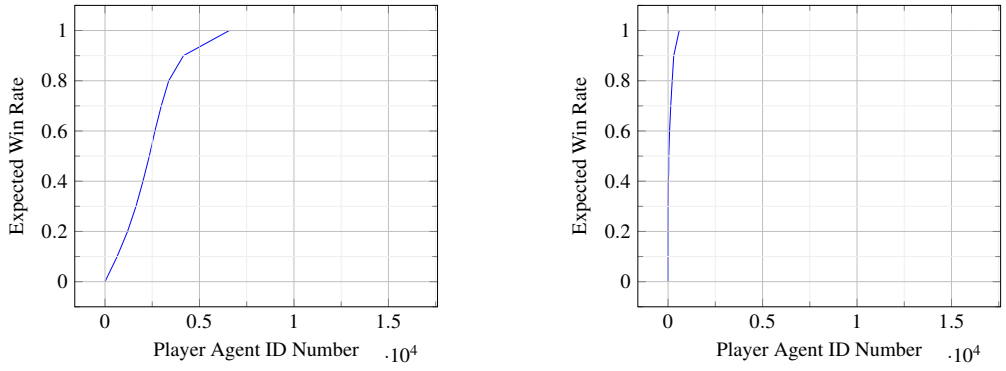
training. The player agent is trained with ABWPL algorithm with the following hyperparameters: learning rate $\lambda = 0.1$, the policy learning rate of $\lambda_\pi = \lambda/100$, future discount $\gamma = 0.9$, exploration rate $\varepsilon = 0.1$, minimum exploration rate 10^{-4}, and max number of steps of 25. The result convergence of Q-values can be observed for single and multi-player scenarios respectively in Figure 3.5a and Figure 3.5b respectively. ABWPL was chosen because it allows for mixed strategies, it is not biased against pure strategies like WPL, which is important in a maze where many states have a single best path.

Afterward, we evaluate each player agent's win rate by making them replay ten episodes. The successful agents form a new subset of the original population. This step is done because agents that cannot complete the task become impossible to assist without controlling their actions directly. For the single-player case the resulting population is illustrated in Figure 3.6a. For the multi-player case, we evaluate each member of the population by making them play against the weakest player. The resulting population is illustrated in Figure 3.6b.



(a) Single Player Setting. (b) Multi Player Setting.

Figure 3.5: Balanceable Maze Player Average Q (training).

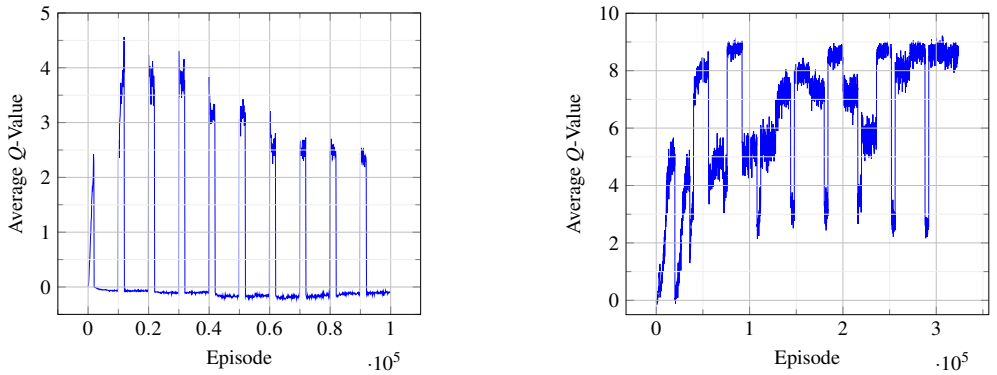


(a) Single Player Setting. (b) Multi Player Setting.

Figure 3.6: Balanceable Maze Population Trace. Both left and right have $\Delta = 1$, as the lowest skill player is 0 and the maximum is 1. The left has a higher Γ than right, as almost every agent has performance 1, where in left almost half of the agents present multiple distinct win rates.

Finally, we train the GM agent by making him play against sampled combinations of player

agents. The GM agent uses ABWPL as a learning algorithm with the same hyperparameters as the player agents. For the single-player case, we sample an agent with each distinct win rate performance of 0.1, 0.2... 0.9 to 1.0. For the multi-player case, we sample three players, one weak player, one average player, and one strong player, and make them play against one another in every combination of opponents and balance states. The GM state is the joint state of its own position, each player's position, and the balance state. In the end, the GM successfully learned the desired behaviors. The result convergence of Q-values can be observed for single and multi-player scenarios respectively in Figure 3.7a and Figure 3.7b. The observed slopes are resulting from the switch (unlock followed by a lock) of the game balance state and player agents.



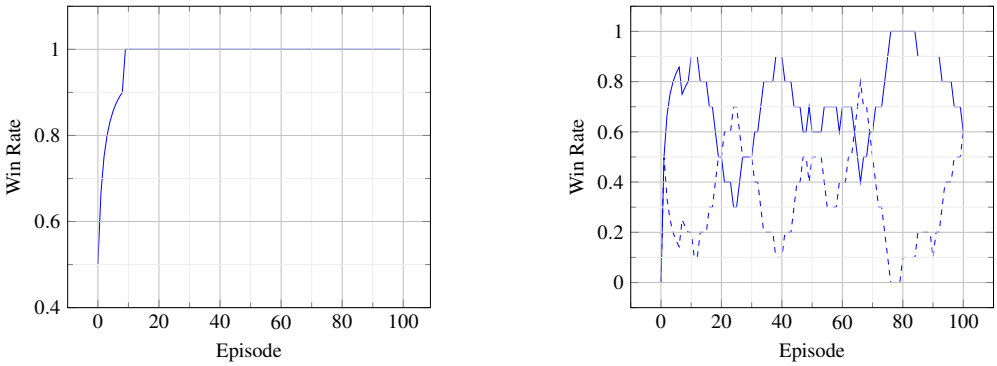
(a) Single Player Setting. (b) Multi Player Setting.

Figure 3.7: Balanceable Maze GM Average Q (training).

The evolution of the win rate before the deployment of the GM agent can be observed for single and multi-player scenarios respectively in Figure 3.8a and Figure 3.8b. In the single-player case, the game was played by the most skillful agent. In the multi-player case, the game was played by an average-skill agent competing against the most skillful agent. The evolution of the win rate after the deployment of the GM agent can be observed for single and multi-player scenarios respectively in Figure 3.9a and Figure 3.9b. In the multi-player cases, the solid player has greater skill than the dashed player. In every case, after the deployment of the GM agent, the win rate converges close to 0.5 for each player.

3.3.5 Case Study 2 - RollerCatcher

RollerCatcher (RoCa) is a two-player game where each player controls a sphere with physical dynamics above an arena, trying to catch a crystal that re-spawns after being caught at a new random position within the bounds of the platform (see Figure 3.10). The spheres do not collide with each other. When a sphere catches the crystal, the controller gets one point. The first player to accumulate 50 points wins the round. Although a chance factor exists, RoCa rewards high control skills, and a weaker player may easily be outmatched by a stronger opponent. To increase player engagement, we install a GM agent that can deploy several handicap mechanisms during



(a) Single Player Setting.

(b) Multi Player Setting.

Figure 3.8: Balanceable Maze Player Win Rate over time.

gameplay. By observing the game state, the GM must deploy the most appropriate handicap, at the most appropriate timing.

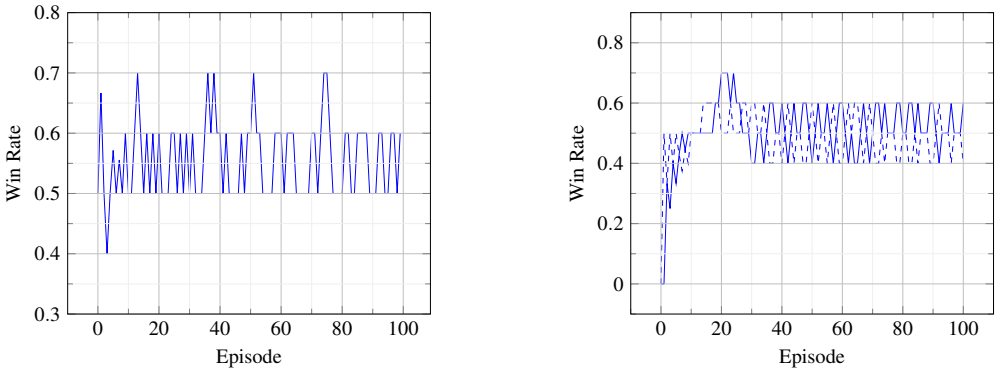
We selected three distinct handicaps of different natures. The first is the configuration of points received by the player. The GM may change the color of the crystal to blue. If a player catches a blue crystal, it gains three points instead. This handicap works directly with the player's score, not affecting the gameplay control mechanics. The second handicap is spawning a wall between the best-performing player and the crystal. This handicap makes environmental changes. The last handicap adapts the speed attribute of one of the spheres, making the best-performing sphere heavier, slower, and harder to control. Contrary to the previous, it instead idolizes changing players' avatar attributes. The color of each sphere is darker when heavier.

Initially, the GM does not know how to best deploy the handicaps to comply with the criteria enforced. Therefore, it must learn how to best deploy handicaps to achieve game fairness. In this case study, we improve the sub-optimal population generation by training in situations against all combinations of handicap deployment. The score handicap does not change nor does the environment nor the player's avatar as such disregarded in this step. In sum, the surrogate agent will know how to respond to the GM handicap deployment. In the Balanceable Maze, the agents were fully stopped by the GM. We also validate the management of multiple balance strategies simultaneously instead of just one.

Our objective is to observe less discrepancy in the score differential and be assured that the reduced differential is being caused by the intelligent decision of the GM agent. Just to name a few situations that would help verify the two requirements:

- Situation 1: The two spheres are far from each other, with a high score differential. The GM could spawn a wall to create an obstacle that would be difficult for the best-performing player without weakening the least-performing player.

- Situation 2: Similar to the previous situation but with both players close to each other, the wall would disturb the weaker player too, and as such slowing down the player with a greater score would not disturb the weaker player, proving to be a better strategy.



(a) Single Player Setting.

(b) Multi Player Setting.

Figure 3.9: Balanceable Maze Player Win Rate over time with GM intervention.

- Situation 3: Both players have a similar score, therefore giving three points could not be best suited for this situation. As such, it would be a good decision to not enable the score handicap, making the differential smaller in the next step.

We rapidly observed a good balance between surrogate agents during the GM training, but the handicaps changed at a high frequency, causing a visual flickering effect, the wall being constantly enabled and disabled, and we found it inadequate for human players. We instead allowed the GM to choose when to activate the handicaps, and remain active until the target is caught, achieving a good compromise of visualization, but then we observed that the balance was not being caused by rational decisions of the GM. For example, when the score of both players was the same, the better strategy would be not to change the value of the crystal, which causes a greater score differential. The third option was to follow the cycle of action, decision, action, and reward at a fixed rate of one action per second (frame-skipping).

To evaluate the trained GM, we compare it against a Null GM agent that does not perform any action, a Random GM agent that deploys handicaps randomly, and a Heuristic GM which follows a hand-coded algorithm and can achieve close to zero score differential quite reasonably. First, we describe our learning agent and heuristic agent and then present our comparative results in games where AI agents in competition and user tests with human players to check how fun users found the game with and without handicaps.

The handicaps are a set of discrete actions the GM can perform, more formally {`SetWall`, `SetScore`, `SetHeavier`}. The Observation Space of the GM is composed by the Euclidean distance d from player one and two $\delta_{12} = d(p_1, p_2)$, where p_1 is the position of player one, p_2 is the position of player two, the distance from player one to the crystal $\delta_{1t} = d(p_1, t)$, where t is the position of the crystal, the distance from player two to the target $\delta_{2t} = d(p_2, t)$, and the score differential from both players $\Delta_s = s_1 - s_2$, with s_1 the score of player one and s_2 the score of player two. The reward received by the GM during training is given by $R = -|\Delta_s|$. A RoCa player agent observes its position, velocity, wall position, target position, and speed handicap status. The GM observes the distance of player one to the target crystal, the distance of the crystal to player

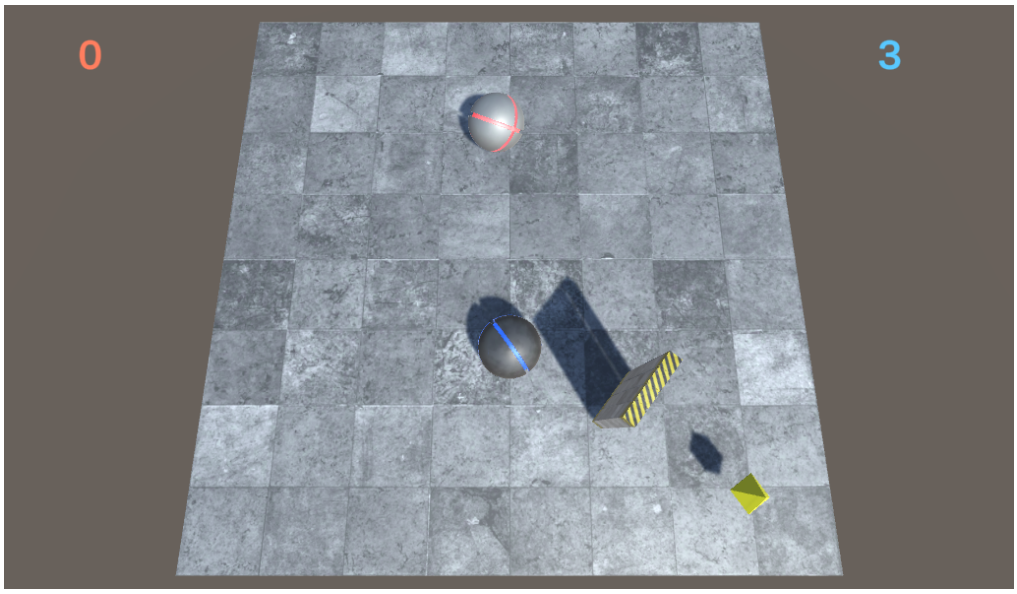


Figure 3.10: RollerCatcher Micro Game.

two, and the distance between both players, it observes if each handicap is enabled and observes the score differential of both players.

We observed an interesting property of the RoCa environment from the perspective of the GM agent, the game always starts with each player having null scores $s_1 = s_2 = 0$, maximizing the reward obtained by the GM, giving a very high reward to random actions. Intuitively, without previous knowledge of the players' performance, we can hardly evaluate if both teams have the same skill level. Therefore, we compared our previous reward model with a normalized reward given by

$$R^{(n)} = -|s_1 - s_2| \frac{\max\{s_1, s_2\}}{\text{MAX}_s},$$

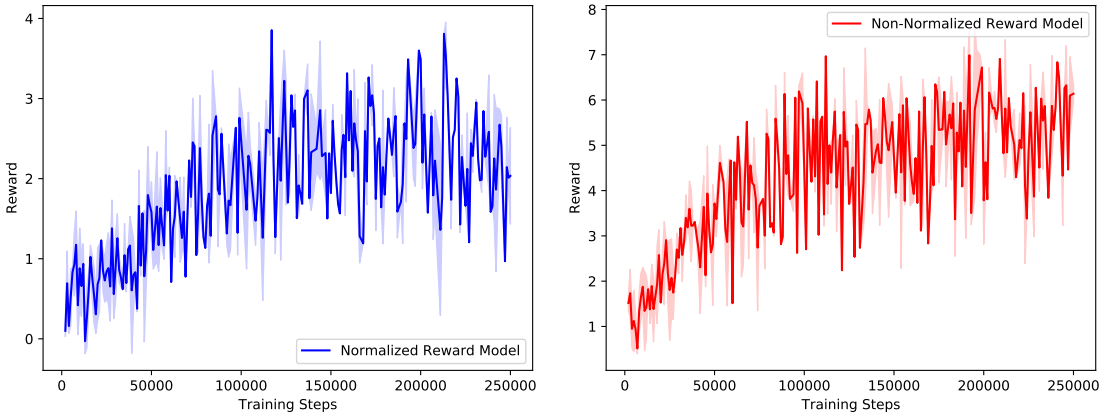
where $\text{MAX}_s = 50$ it's the maximum possible score to obtain by a player in the game. The intuition of this function is that the maximum reward is still given when the scores are equal $s_1 = s_2$, but at early stages of the game when the score of the players is lower the rewards are decreased by a weight factor which increases with the greatest score of the player over time, making early plays less import than later plays.

For the learning GM the following hyperparameters were used for the PPO trainer: $\beta = 5.0^{-3}$, 2 hidden layers with 128 units, a future discount reward of $\gamma = 0.99$, a learning rate $\alpha = 3.0^{-4}$, exploration rate $\varepsilon = 0.2$, batch size of 10, buffer size of 100, the memory size of 256. It is a good practice for immediate rewards to be normalized to stabilize training, as such R and $R^{(n)}$, are both adjusted. In our training sessions, the reward functions used were

$$R = 1 - \frac{|s_1 - s_2|}{25} \text{ and } R^{(n)} = \left(1 - \frac{|s_1 - s_2|}{25}\right) \frac{\max\{s1, s2\}}{\text{MAX}_s}.$$

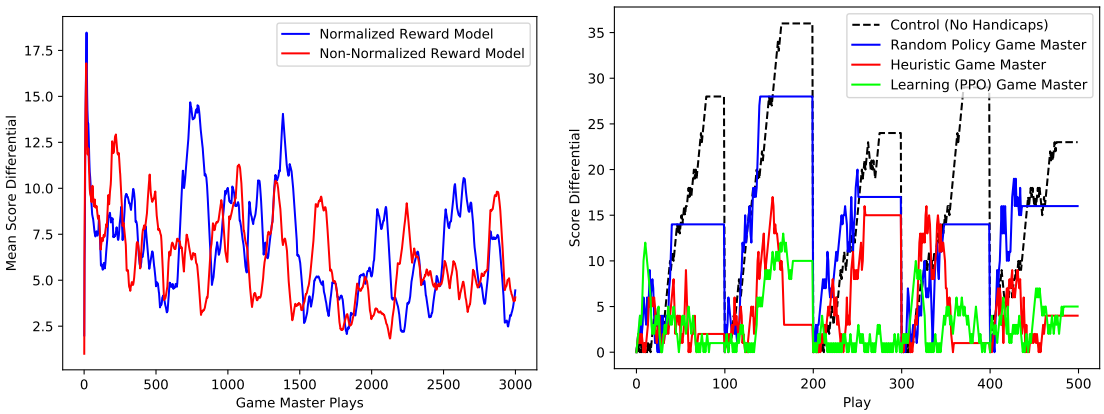
The training evolution of both normalized and non-normalized reward models can be observed

respectively in Figure 3.11a and Figure 3.11b, by observing the average cumulative reward at every 1k steps. The training curves present similar shapes both converging after 250k training steps. As observed by the maximum rewards received, the values received by the normalized version are lower due to the weight factor. Therefore to compare train stability between both reward models we observed the average score differential, as can be observed in Figure 3.12a, which once again does not present a significant difference, becoming inconclusive that the normalized reward model has any significant influence on training.



(a) Normalized Reward Model. (b) Non-Normalized Reward Model.

Figure 3.11: Training Reward of the GM.



(a) Score Differential during Training. (b) Score Differential after Training.

Figure 3.12: GM Multi-Agent RollerCatcher Statistics.

Our rule-based heuristic GM acts as follows. The same observation space and action space for our heuristic GM as the learning agents use adding the score. We first check if $\Delta_s = 0$, which means the score of both players is the same. If no we proceed to the subroutine `DisableHandicaps` which set all handicap flags to false. Else, we check heuristic conditions for each handicap and enable the handicap flag of H. If the difference in score is higher or equal to three, $\Delta_s > 3.0$, and the losing player is closer to the crystal we set the crystal blue, subroutine `SetScore`, to reward

him with three points. If the distance between both players is three or more, $\delta_{12} > 3.0$, we use the wall to make an obstacle for the winning player without affecting the losing player, subroutine `SetWall`. Otherwise, if both are close, $\delta_{12} < 3.0$, we instead deploy the slow handicap, turning the winning player heavier and therefore slower, subroutine `SetHeavier`.

We display the results of the multiple GM in Figure 3.12b. For each of the four scenarios, Control with no handicaps, Random Policy GM, Heuristic GM, and Learning GM, we made five complete games between an optimal RL agent and a human Behaviour Cloning agent, which could have in max 100 plays. At every 100 plays, we can compare the results in each case. As observed with no handicaps deployed, the discrepancy between the player agents reached in the worst case 35 points. With the random policy, the discrepancy maintains higher and does not benefit the game quality. The latter two were able to reduce the score differential during gameplay. The heuristic GM had, in general, a lower performance in comparison to the deep learning agent which obtained the lowest score differential. By our observations of the behavior of both agents, we concluded that this is due to our heuristic agent being more benevolent, for example not deploying the wall obstacle and changing the weights of the spheres simultaneously, while the RL learned a more aggressive strategy where attacks the winning player with both handicaps and more efficiently reducing the score differential. In terms of our expected behaviors, the heuristic agent complies with Situations 1-3. Situations 1 and 3 are observed by the learning agent, although, Situation 3 rarely happened, as it uses the more aggressive strategy, and the score differential is less than three most of the time. Situation 2 did not fully occur, the learning agent deploys the wall even if both players are close, but as it deploys the speed/weight handicap, the losing player can get around the wall faster.

3.3.6 User Evaluation

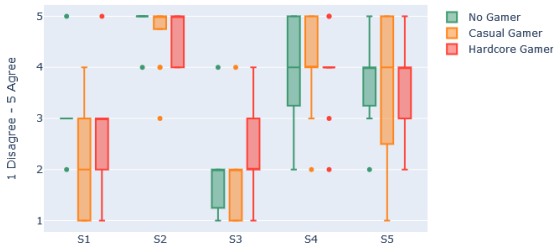
There is the risk that players may find the adjustments unfair, decreasing their engagement, or even of an ill-intended player exploiting the vulnerabilities of the DDA algorithm by losing purposely to gain the advantage. In this section, we validate our work with a vector of user tests, to find out to what degree human players found the automatic handicaps fair and engaging.

Each participant was first made to play an assisted learning tutorial where they learn how to control the sphere and catch the target crystal. In the first experiment (Experiment 1) we made the participant play against an optimal RL agent (representing a very skilled and unfair opponent). The experiment was followed by a second tutorial for the player to learn how to deal with the deployed handicaps (change in weights and wall spawning). After completing the second tutorial, the participants played again against the same opponent, the RL agent, but with automatic deployment by the PPO GM (Experiment 2). In the final experiment (Experiment 3) we change the frequency of the handicap deployment from 1Hz to 1/3Hz, to give both players more reaction time, and see the impact it caused on the human participants.

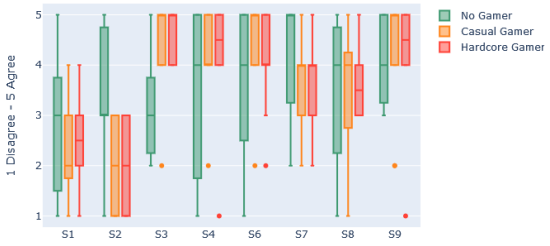
The inquiry consisted of a population of 30 volunteers, where 23,3% declared to not play video games (No Gamer), 43,3% of the volunteers declared playing video games casually (Casual Gamer), and 23,3% declared to play video games competitively (Hardcore Gamer). 80% of the

Table 3.3: User Questionnaire

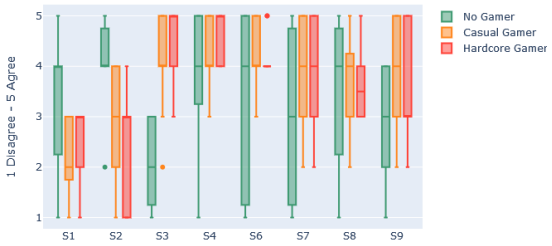
S1	I found it difficult to control the sphere.
S2	I found it difficult to keep up with the opponent's sphere.
S3	I found the challenge balanced.
S4	I had fun playing.
S5	The game would have been more fun if I could better keep up with the opponent's sphere.
S6	The wall was helpful in balancing the game.
S7	The weight change was helpful for fame balancing.
S8	Score manipulation was helpful for game balancing.
S9	The game was more fun than in the previous test.



(a) Experiment 1.



(b) Experiment 2.



(c) Experiment 3.

Figure 3.13: Test statistics from users.

population was male and 20% female. 93% of the population had an age between 18 and 35. Table 3.3 enumerates all the statements made in the inquiry where each volunteer must assign an ordinal value between 1 and 5, where 1 means disagree and 5 means agree:

From Figures 3.13a, 3.13b, 3.13c we can extract multiple conclusions. Without handicap deployment, sphere control depends fully on the player's skill, the majority of volunteers found it hard to keep up with the RL agent opponent and found the challenge unbalanced, but still had fun playing, although the majority would rather keep up with the opponent sphere. With handicap deployment, most volunteers found it easier to compete with the opponent sphere, found the challenge more balanced, remained fun playing, and found both wall and weight handicaps helpful in balancing the game. The score handicap was less conclusive, mostly because of the lack of its presence in the experiments. The majority of volunteers found the game more fun and entertaining with handicaps than with their absence. In Experiment 3, with lower frequency handicap deployment, both human and RL agents had more reaction time, and players found it a little harder to keep up with the opponent sphere, but they had more time to catch the blue crystal

gaining more frequently 3 points. Finally, the users found Experiment 3 to be the most entertaining setting overall. We also notice a great discrepancy between the group No Gamer, compared with the opinion of both other groups, showing a gap between Gamer and No Gamer perception of gaming and handicapping.

In Experiment 1 the average result of human vs RL agent was 20-50 while in the handicap deployment experiments, the average result was 45-49, an increase of 25 in average score compared to the setting without handicaps. Some players found an exploit in the trained GM strategy. When close to the 50 score objective, if the player had the biggest score, they would suffer from negative handicaps, and the opponent would catch up and win. Therefore, some players that noticed this pattern would try to stay behind the opponent until the final phase, and then the opponent would suffer the negative handicaps instead and the human could catch up and win the game. In two cases, the player felt unmotivated to continue playing. These cases are not unexpected, as our objective was to balance when players are trying to win and reduce the skill dependence, which does not cover a case where the player does not like the game and felt unmotivated to play, being out of the scope of this work, but an important issue to tackle in the future nonetheless. More competitive players found Experiments 1 and 3 more entertaining as their skill was put to the test. It was inconclusive if the score handicap was not a viable strategy.

3.3.7 Limitations and Discussion

In this preliminary study, we designed and explored the Implicit Model for the application of DRL to achieve game fairness in matches with players having uneven skill. A DRL GM was installed and trained over the base game and evaluated against other GMs. The DRL GM succeeded in achieving a more reliable, scalable, but aggressive strategy against hand-coded or random GM. To assess how fun the game feels with the dynamic adaptations controlled by the DRL GM we inquired several human volunteers who conducted a series of settings, which overall positive reactions, but most likely due to having a much stronger opponent resulting only in receiving positive handicaps. Competing against weaker adversaries could have resulted in more negative reactions due to the negative handicaps, so there is the risk our data is biased by the experimental setting. By using just the win rate or score differential as a metric to optimize, we do not make any in-game pacing guarantees, just that the outcome will be even. The simplicity of the test environments allowed us to hand-tune a heuristic agent for comparison, but more complex scenarios could motivate us to better improve our model and empirically reinforce the framework capabilities.

Regarding the limitations of our surrogate agent sampling technique, we would want to guarantee more diversity and parameterization of the behaviors. Quality diversity algorithms could encourage finding a large variety of solutions. A singular evolutionary line may not be enough to ensure population diversity, mainly in games with intransitive game mechanics. The use of explicit meta-states also becomes harder with more complex environments, so it would be preferable just to rely on the game base attributes.

3.4 MDDA-RL Framework

Our proposed MDDA-RL Framework is inspired by two previous models, and both are discussed first.

Our Implicit Model formulates the game adaptation problem for two-player games as a mapping from a base game to a target game complying with a set of restrictions. By embedding a GM agent in the game, which initially takes no practical actions, the agent is trained to play a meta-game, a game where actions performed change the base game rules. It optimizes a reward function where the win rate between two players is balanced. The aim of this framework is to convert a game into one that complies with similar win rate criteria independent of each player's proficiency but also limits games to predictable outcomes without any dramatic effects.

The Joint Perspective (Thue and Bulitko, 2018) incorporates a GM agent in a Factored-state MDP (FMDP). Each state s is split into n factors. The Joint Perspective is built over six properties: i) Manager Policy, which performs decision-making for the GM. ii) Tunable Parameters which are the features the GM is able to configure; iii) Opportunity to act, the GM may act simultaneously or sequentially to players' actions; iv) game mechanics unaffected by the GM; v) the space state, consisting of tunable and non-tunable features; vi) the GM action space.

The Joint Perspective was conceived to study engineered GM agents in games. At the same time, the implicit model proposes a reward function to train the GM agent but the resulting games lack gameplay quality. We combine both approaches' strengths and extend them to MDDA-RL where a Synthetic Design Module is added to reward an in-training GM following a set of game design criterion modules that assert gameplay quality. The architecture is illustrated in Fig 3.14.

Compared to a vanilla MDP we: add A^m the action space of the GM agent; we separate the features of S into S^g the game features observable by any agent and S^m the meta-features only observable by the GM; add $E : S \times A \rightarrow \mathscr{P}$ an estimator (or challenge function in the literature) of the performance of a player on a given game state-action pair, $\mathscr{P}$ is an arbitrary performance space (e.g. $[0,1]$ corresponding to a player's score with a bounded real value); the objective function r^m that returns the reward signals for the GM. The formal comparison with the Implicit Model and the Joint perspective is illustrated in Table 3.4.

3.5 Aesthetic Criteria

The application of MDDA-RL requires finding a reward function for the GM that aligns with our design criteria. We use a weighted function that only attributes a single reward at the end of each training episode. A game designer can delegate a set of game design criteria, each with a distinct weight.

For this work, it was selected aesthetic criteria based on the ones originally used by the Ludi Framework (Browne, 2008), which follow good game design practices. These criteria were used to evaluate and synthesize board games using evolutionary methods. The aesthetic criteria were classified as follows:

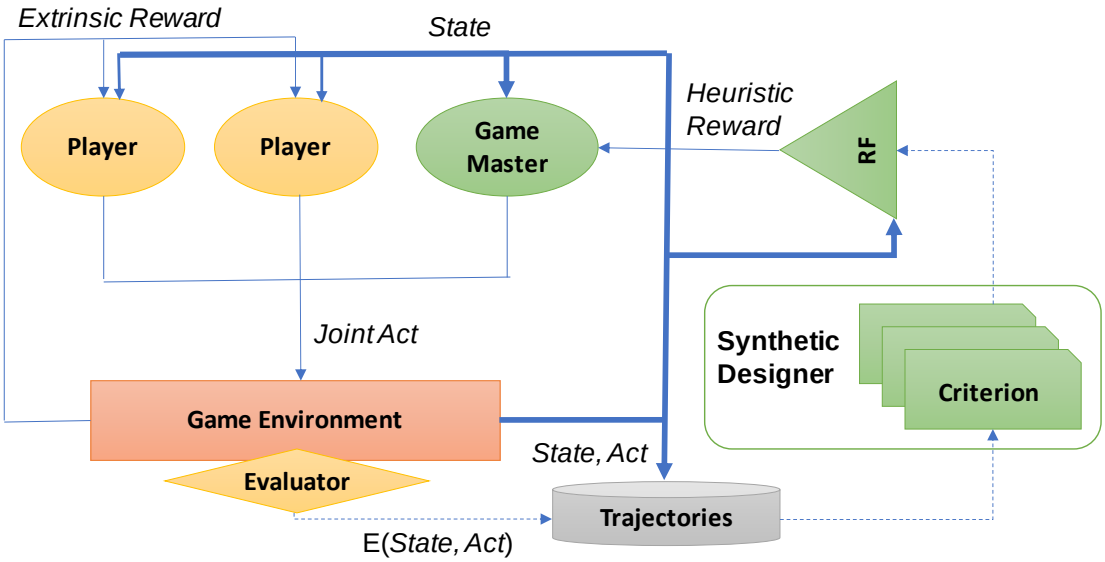


Figure 3.14: MDDA-RL Architecture has two main processes: (i) the traditional MDP cycle, the players are provided with extrinsic rewards, and the GM with the predicted external reward; (ii) the performance estimator estimates each player advantage and pairs them by trajectories. The trajectories are fed to the Synthetic Designer, which asserts the quality of the gameplay trajectory and labels the trajectories with their quality, to reward the GM, through a Reward Function (RF).

- **Intrinsic Criteria** are related to the inherent game artifacts and rules structure and are used to compare different games. These criteria are not suitable for our use case since the artifacts of the adapted games are the same as the original game.
- **Extrinsic Criteria**, sub-categorized into Validity Criteria, evaluate game resolution in terms of win and loss rates; Quality Criteria, assess the pacing of the game by evaluating each player's performance during gameplay, which can indirectly translate to players' experience or skill.

We opted for Quality Criteria because they can help to detect changes in the advantage of each player during gameplay, signaling the rubber-banding effects we aim for.

To apply these criteria in MDDA-RL, consider the performance of a player p denoted by $E_p(m_t) \in [0,1]$ at move or step m_t with step number t. It indicates how close each player is to a winning state (see Fig 3.15 for a visual example). Then consider aesthetic criteria set $\mathscr{C} = \{C_i\}_i$ and corresponding weight set $\mathscr{W} = \{w_i\}_i$. A criterion $C_i(E_{\tau_{p_1}}, E_{\tau_{p_2}}) \to [0,1]$ maps the evaluation of a pair of player trajectories (τ_{p_1}, τ_{p_2}) of the same gameplay to a value bonded between 0 and 1. The reward function returns

$$r(s_T) = 2\left(\sum_i w_i C_i(E_{\tau_{p_1}}, E_{\tau_{p_2}}) - 0.5\right), \tag{3.1}$$

with $r(s_T) \in [-1,1]$, where s_T is the terminal state. If fairness were to be considered, where win rates over a sequence of games converge to 50%, it would be required to consider Validity Criteria,

	Implicit Model	Joint Perspective	MDDA-RL
MDP	$\langle A, S, M, e, r, r^m \rangle$	$\langle A, S, \phi, e \rangle$	$\langle A, S, M, \phi, e, r, E, r^m \rangle$
Action Space	$A = \langle A^g, A^m \rangle$	$A = \langle A^g, A^m \rangle$	$A = \langle A^g, A^m \rangle$
State Space	$S = S^g \times S^m$	S^g	$S = S^g \times S^m$
Reward Function	$r : S \times A \to \mathbb{R}$	-	$r : S \times A \to \mathbb{R}$
Game Mechanics	$e : S \times A \to S$	$e : S^g \times A \to S^g$	$e : S \times A \to S$
Tunable Parameters	-	$S^t \subseteq \phi(S^g)$	$S^t \subseteq \phi(S^g)$
Manager Policy	$\pi^m : S \to A^m$	$\pi^m : \phi(S^g) \times A^g \to A^m$	$\pi^m : \phi(S^g) \times S^m \to A^m$
Opportunities to Act	Simultaneous	Any	Any
Performance Estimator	-	-	$E : \phi(S^g) \to \mathscr{P}$
Objective Function	$r^m : S \times A \to \mathbb{R}$	-	$r^m : \mathscr{P} \to \mathbb{R}$

Table 3.4: MDDA-RL Framework contrasted with the Implicit Model and Joint Perspective. MDDA-RL introduces the performance estimator, which is used by the GM reward function. It maintains most FMDPs characteristics but maintains the Implicit Model GM policy.

by concatenating multiple gameplay trajectories and counting the number of wins and losses from each player. However, this makes rewards only attributed after each N games and become more sparse, and the training becomes relatively inefficient.

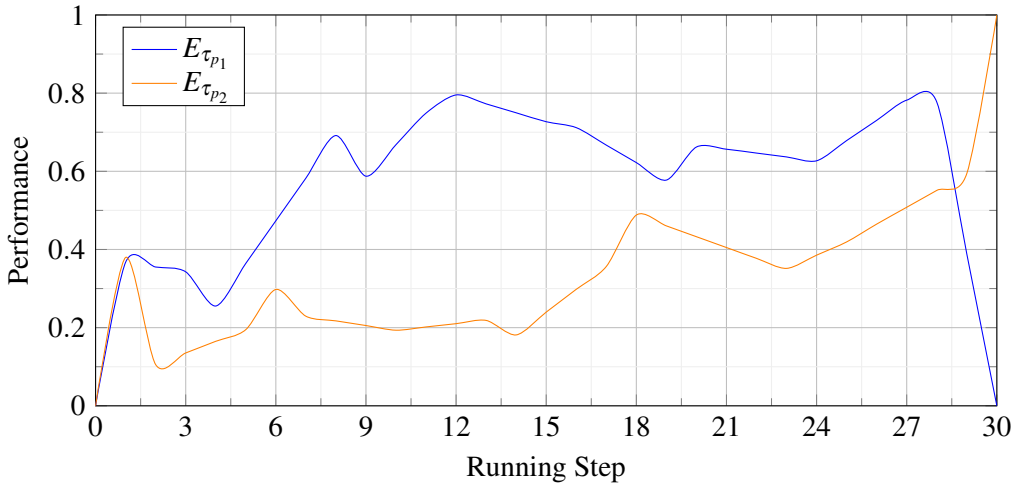


Figure 3.15: Example of the performance of each player over a gameplay trajectory. Player $p1$ showed to be more well-positioned for most of the gameplay. Each player starts with a null performance, the winner always finishes with 1 and the loser with 0. The example gameplay was overall dramatic as player $p1$ maintained an overall advantage but ended up losing.

The Uncertainty Quality Criterion C_1 asserts how long a game's outcome was until when a decisive move was made. Consider E_w the performance at each move of the eventual winner and E_l the performance of the eventual loser, the uncertainty can be measured as the distance from the final step to the last lead change. Consider L the winning player at move m_t, this criterion returns:

$$C_1 = \frac{\max_t \{t : L_t \neq L_{t-1}\}}{T}. \tag{3.2}$$

This criterion as the name suggests promotes games where the outcome is decided at a later stage of the game. If the game outcome is decided too early, it could bore the losing player, as its chances of recovering are minimal. In Figure 3.15 the uncertainty is high, only on step 28 out of 30 the decisive move was made.

The Drama Quality Criterion C_2 determines how many opportunities an eventual loser had to recover. Consider $\#_i$(condition) an operation that counts the number of instances a condition verifies. The gameplay drama can be measured by the number of moves where the lead $\Delta_{l,w}E(m_t) = E_l(m_t) - E_w(m_t)$ was positive:

$$C_2 = \frac{\#_{1<t<T}(\Delta_{l,w}E(m_t) > 0)}{T}. \tag{3.3}$$

Drama is the number of steps where the eventual loser had the lead, showing that there was a certain degree of dramatization. In Figure 3.15 drama occurred from step 2 until step 29 where a change of leading occurred.

The Maximum Drama Quality Criterion C_3 is similar to the previous, but only the greatest lead gap is measured, which demonstrates dramatization:

$$C_3 = \max\left(0, \max_{1<t<T} \Delta_{l,w}E(m_t)\right). \tag{3.4}$$

A game where a big gap between game advantages exists is another sign of dramatization. In Figure 3.15 the maximum drama occurred at step 12, where the losing player presented a performance of 0.8 while the winning player had a performance of 0.2, resulting in a maximum drama of 0.6.

Lead Change Quality Criterion C_4 counts how many times the lead changes over the course of the game, another signal of dramatization:

$$C_4 = \frac{\#_{1<t<T}(L_t \neq L_{t-1})}{T}. \tag{3.5}$$

In Figure 3.15 only two changes of lead happened, at step 2 and step 28. Depending on the game, a certain number of expected changes of lead can translate to two changes being a high or low value. In the case of our game test suit, this would be considered a low value.

Game Length C_5 is another essential quality criterion for MDDA-RL as will be discussed further ahead. But from a basic perspective, the game length is relative to the type of game, as some games are intended to be fast-paced while others are oriented towards long sessions. How the length is measured also depends on the type of game, for turn-based games, we consider steps the moments where agents make moves, having unlimited decision time, and the expected length is measured in plays, while in action or real-time-based games, the expected length is measured in frames. In Figure 3.15, the duration is 30, and the criterion value will depend on the intended game duration.

3.6 MDDA-RL Training Pipeline

3.6.1 Player Agent Population

To generate gameplay data, we inspire from our preliminary work, where using a population Π of sub-optimal agents with distinct levels of proficiency competing against each other provides generalized training settings for the GM training. In this work, instead of storing the evolutionary phases of a training agent, was opted to just train a reference (eventually optimal) player agent π^*, and then perturbations are exerted on its actions to which degree can be controlled in a parameterized fashion, so the inequality scenarios between two players can be controlled. Since at this step, the GM policy π_θ^m is not yet trained, a random policy is installed over the game environments, so the reference player agent learns to play with the presence of handicaps or would be otherwise hard to cope with unseen scenarios.

3.6.2 Player Performance Evaluator

Our performance evaluator is a Q-Critic $E_\theta^*(s,a) = Q_\theta^*(s,a)$ for actions a taken by player agents in state s assuming we are following the reference policy π^*. To train E_θ^*, it is targeted the lead assuming it grows linearly until victory (from 0 to 1) complying with our progress definition for aesthetic criteria. We assume each agent receives 0 or 1 in the final step and 0 otherwise (regardless of the environment reward function), and we linearly interpolate the reward at each step in reverse. To create appropriate training scenarios, the reference agent π^* will play against itself and weaker variants. To estimate sub-optimal plays, it can optionally be increased over time a ε chance of making a mistake (take random actions to explore sub-optimal plays akin ε-greedy). When there is a mistake at step t, we bootstrap $E^*(s_t,a_t)$, assuming that the lead corresponds to the maximum between the mistake's value (which does not follow π^*) and the bootstrapped value. We interpolate intermediary steps between each two mistake steps (see Algorithm 3).

Algorithm 3: Performance Estimator Training

Result: Player estimator E_θ^*
Data: Player policy population Π
$\pi_\varepsilon^* \leftarrow$ Elect model player from Π with ε-exploration
while *not reached number of training steps* **do**
  Set uniform GM policy π_r
  $\pi_\theta \leftarrow$ Sample adversary policy from Π
  $\varepsilon \leftarrow$ Set exploration coefficient linearly given *step*
  Trajectory $\tau_{p_1} \leftarrow$ Run episode with $(\pi_r, \pi_\varepsilon^*, \pi_\theta)$
  Set targets $E_{\tau_{p_1}} \leftarrow \{E_j = \sum_i^j \frac{i}{N_g} r_{\tau_{p_1},i}\}_j$ bootstrapping when next $step_{i+1}$ was random
  Train E_θ^* with targets $E_{\tau_{p_1}}$ and inputs τ_{p_1}
end

3.6.3 Game Master Agent

Our objective is to train the GM's policy π_θ^m such that the trajectories $(\tau^m, \tau_{p_1}, \tau_{p_2})$ maximize $\sum_i w_i C_i(E_{\tau_{p_1}}, E_{\tau_{p_2}})$. The process is sequenced in three phases: i) collect gameplay trajectories, estimating the performance of each player using E_θ^*; ii) evaluate the gameplay quality with aesthetic criteria; iii) Fed π_θ^m with the rewards regarding the aesthetic criteria (see Algorithm 4).

Algorithm 4: Game Master Training

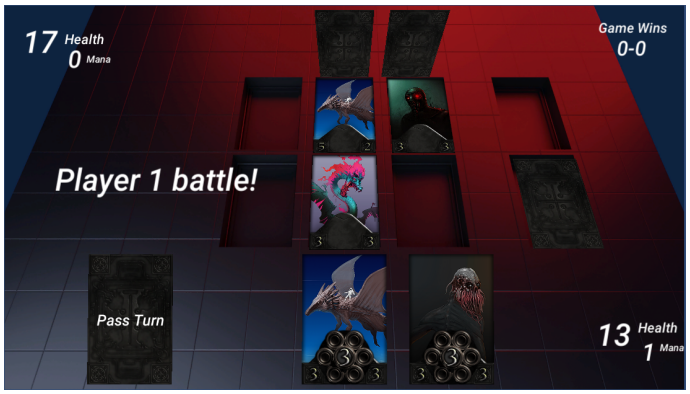
Result: GM policy π_θ^m
Data: Estimator function E_θ^*, Player policy population Π, Criteria set $\mathscr{C}$, Weight set $\mathscr{W}$
while *not reached number of GM training steps* **do**
  $\pi_{1\theta}, \pi_{2\theta} \leftarrow$ Sample a pair of policies Π
  Trajectories $\tau^m, \tau_{p_1}, \tau_{p_2} \leftarrow$ Run episode with $(\pi_\theta^m, \pi_{1\theta}, \pi_{2\theta})$
  $E_{\tau_{p_1}} = \{E_\theta^*(o_{p_1}, a_{p_1}), \forall o_{p_1}, a_{p_1} \in \tau_{p_1}\}$
  $E_{\tau_{p_2}} = \{E_\theta^*(o_{p_2}, a_{p_2}), \forall o_{p_2}, a_{p_2} \in \tau_{p_2}\}$
  $r_{n-1} \leftarrow 2\left(\sum_i w_i C_i(E_{\tau_{1,p_1}}, E_{\tau_{1,p_2}}) - 0.5\right), C_i \in \mathscr{C}, w_i \in \mathscr{W}$
  Set τ^m rewards as $\langle 0, \ldots, 0, r_{n-1} \rangle$
  Train π_θ^M with τ^m
end

Players' and GM's trajectories may be asynchronous, with different act opportunities. Therefore, there will be ghost steps where agents did not act, and it cannot be directly evaluated $E_p(m_t)$. To compare players' performance at every step is done linear interpolation over ghost steps performed by the player (except on GM exclusive steps), i.e., it is assumed that their game advantage should tend linearly from the value from the last move to the value of the next move. An example is illustrated in Fig. 3.16, where is presented a triplet trajectory from two player agents and a GM agent with some ghost steps.

	Trajectory				
t	0	1	2	3	4
τ^m	$s^{(*)}$	$s^{(*)}$	s_2, a_2	$s^{(*)}$	$s^{(*)}$
τ_{p_1}	s_0^1, a_0^1	s_1^1, a_1^1	$s^{(*)}$	s_3^1, a_3^1	s_4^1, a_4^1
τ_{p_2}	s_0^2, a_0^2	s_1^2, a_1^2	$s^{(*)}$	s_3^2, a_3^2	s_4^2, a_4^2
$E_{\tau_{p_1}}$	0.40	0.50	<u>0.65</u>	0.80	1.00
$E_{\tau_{p_2}}$	0.20	0.30	<u>0.60</u>	0.90	0.00

Figure 3.16: A triplet trajectory with different opportunities to act. The states indicated as $s^{(*)}$ are ghost states, states where a given agent did not have an opportunity to act. Ghost states are intermediate-stage consequences of other agents acting in those instants. Ghost states are not used for their respective agent training trajectory. However, they are used for aesthetic criteria with $E_{\tau_{p_1}}, E_{\tau_{p_2}}$. For players p_1 and p_2 their trajectories are $\tau_p = \langle (s_0^i, a_0^i), (s_1^i, a_1^i), (s_3^i, a_3^i), (s_4^i, a_4^i) \rangle$, and for the GM is $\tau^m = \langle (s_2, a_2) \rangle$. Underlined values are interpolated.

3.7 MDDA-RL Test Suit



(a) HeartOfTheCards Strategy Card Game.

(b) Right2Live Action Survival Game.

Figure 3.17: MDDA-RL Test Suit.

3.7.1 HeartoftheCards

HeartoftheCards (HotC) is a turn-based strategy card game, where the players have the objective of reducing the Hit Points (HP) of the opponent to 0, playing minion cards that remain on the field to attack the opponent (see Fig 3.17a). Cards in hand can summon minions, deal direct damage, interact with the field and the opponent's hand, draw cards from the deck, or recover life. At the end of each turn, the active player's minions damage opponent minions in the same column as them with a value equal to its attack, and defeat it if it reduces its defense to 0; otherwise, it deals damage to the opponent. Players gain three Mana Points (MP) each turn (5 max) and each card has a cost of up to 5 MP. Players may play cards that cost less than their MP. The GM selects from 13 possible card types to draw from each player's deck. The initial hands are set randomly.

The game state is constituted by the three minion field slots and the three-hand slots for each player, the player's HP, and MP. Each card has one or multiple effect types, all of these as categorical dimensions while attack, defense, and cost as continuous dimensions. The observation space for player agents is 49 and 76 for the GM (the GM has revealed the hands of both players). The player's categorical action space corresponds to the card-to-play and the target field position,

so two branches of discrete actions totaling 7 dimensions. In addition, players can play multiple cards, until they deplete their MP or make an invalid move.

3.7.2 Right2Live

Right2Live (R2L) is a real-time survival action game, where two players compete for survival, with the objective of defeating more undead enemies than the opponent (see Fig 3.17b). There are multiple types of undead, each with its own unique quirks. Each player must defeat the undead or grab treasure chests to score. The first player to reach a score of 10 wins. If a player's HP reaches 0, it loses. The GM can set more defense or offense for the player, and change the spawn rate in terms of overall spawns and the ratio of enemies spawns.

The player agents have a continuous action space of 5 dimensions, corresponding to 3 degrees of movement and 2 attacks. HP can only be recovered by grabbing spawned potions. The observation space corresponds to fifteen uniformly distributed ray casts that measure from the player's position the distance, type of enemy and attributes, and its own stats like position, HP, and stamina with a total of 248 dimensions. The GM has an action space of 8 corresponding to 4 discrete branches, 2 with 3 options of spawn rates, and one option for a multiplier over the player defense. The GM observes hand-selected intensity parameters like players' HP, player's score, the average distance to undead, and the number of undead in a total of 20 dimensions.

3.8 MDDA-RL Evaluation

Every setting was trained in commodity hardware, with an AMD Ryzen 9 5900HX at 3.30 GHz, and 16.0 GB RAM.

3.8.1 Player Agent Training and Play Performance

The reference agent for HotC used a fully connect architecture with 2 hidden layers of 256 units, a learning rate $\alpha = 0.0003$, 3 batch passes for optimization of 128 with a buffer size of 2048, a future discount $\delta = 0.99$, a GAE (Schulman et al., 2018) discount $\lambda = 0.99$ and an entropy regularization coefficient $\beta = 0.01$ (Ahmed et al., 2019). For R2L we used the same architecture but with 3 hidden layers of 512 units, optimization passes of 2048 samples with a buffer size of 20480, a future discount $\delta = 0.995$ and an entropy regularization coefficient $\beta = 0.005$.

The two environments used a PPO trainer, initialized with a kaiming normal initialization (He et al., 2015), and a swish activation (Ramachandran et al., 2017) is used for every hidden layer. HotC was trained in a curricula regime, where it first optimized against a uniform policy over 1M steps achieving a 0.95 score, and then trained over 3M steps in a self-play regime, fighting 25% of the episodes against a random agent, 25% of the episodes against its 1M step version, and 50% of the episodes against itself (4 hours), achieving a $95 \pm 5\%$ win rate against the random policy. R2L was trained over 20M steps, with an average cumulative reward of 2.3 (16 hours). It achieves a

$90 \pm 5\%$ win rate against weaker agents, giving priority to killing enemies, then catching treasures and potions.

3.8.2 Sub-optimal Agents

HotC sub-optimal player agent ranks the probability of each possible action, then the quality of the action to take is sampled following a beta distribution, with either a or b set to 2 and the other in the $[2, 20]$ interval. This results in sub-optimal card choices and field positioning. R2L sub-optimal player agent strength is configured by a perturbation η, and the noise is used to make small changes to the sampled actions from the model policy (clipped between $[-1, 1]$). To ensure some actions are provided from the reference policy, an ε probability parameter can be set in the two cases. The movement follows sub-optimal directions, and agents will not be best oriented to catch treasures or beat enemies.

3.8.3 Performance Estimator Training and Accuracy

The performance evaluator for each of the environments followed the same network architecture as their respective policy ones, but only with a head of only one output with sigmoid activation. HotC was trained in 250 k steps (half an hour) achieving a loss of 0.02. R2L trained in 350k steps (half an hour) achieving a loss of 0.012. For R2L, being a real-time environment, and a parallel game (opponent interaction does not affect our progress), it was made two small changes to the estimator training regime: 1) skip the bootstrapping step on mistakes; 2) having access to the completion progress through the player score, it is used as a final reward instead. These changes yielded more accurate predictions.

To assess the accuracy of the performance estimators, it is used heuristic functions for each environment with input features we considered relevant to determine the game's lead player, and compared the graphs from the heuristic and the trained estimators. For HotC our heuristic measured the distance of the opponent's HP to reach 0. For R2L, was considered the score and HP of both players. The curves were also contrasted in real time to the played games by a human expert over dozens of replays for each game.

In HotC it was observed that a player whose opponent has more HP but has more powerful cards on the field may be closer to victory. This shows that HP and score are important features in measuring player advantage, but not the only ones. In R2L, the accumulated score is a major advantage factor, and its value increases during gameplay, but it was also observed to drop when the player's HP gets close to 0, showing that it detects when the player is losing.

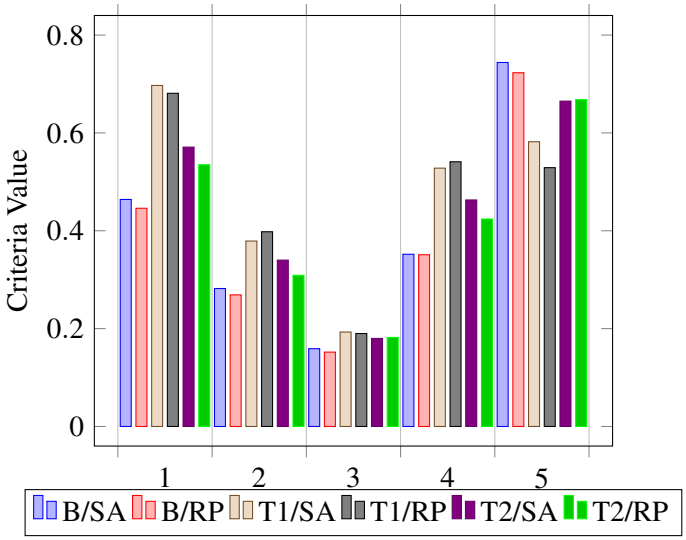
3.8.4 Game Master Training and Game Synthesis Evaluation

To evaluate the influence of the GM over the games, we experiment with multiple settings: different combinations of opponents; different criteria weights as maximization objectives; and modified actions spaces. We also describe the observed gameplay changes due to handicap management.

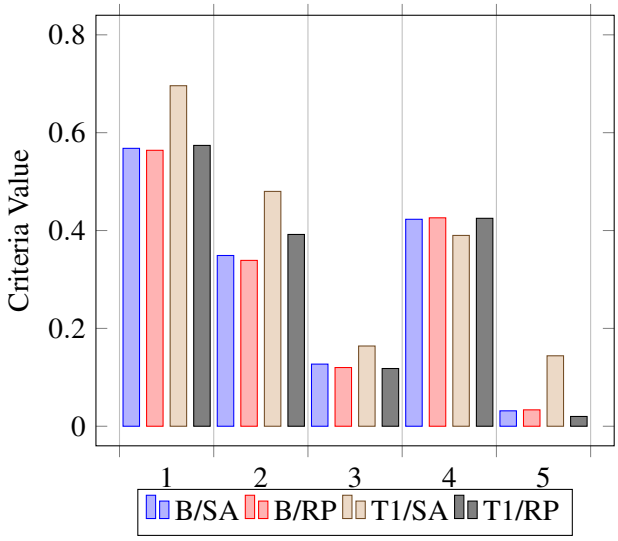
In HotC it was noticed a particular limitation. The GM agent converges for greedy strategies with the consequence of game states tending to be similar causing the GM to always select the same cards and causing returning to repeated states, making every game practically the same, i.e., follow a similar sequence of cards. We hypothesized that the same phenomena were not observed in R2L because the handicap mechanics are inherently stochastic and the GM only has to choose the best configuration depending on the different scenarios. As such, HotC was modified, and as previously described, the action space of the GM agent was adapted to select from a pile of cards (which sample randomly from a subset) instead of single cards. Each pile has a theme, one of the low-cost minions, one of the high-cost minions, cards with card draw effects, damage effects, and miscellaneous. By removing some control from the GM, we add randomness to visited game states.

The results for HotC measured in criteria value are illustrated in Figure 3.18a. The same hyper-parameters as the player agent were used, with the difference of using hidden layers of size 128. A base random GM with the reference player agent yielded overall criteria of 0.35 and with a random population of players, agents yielded 0.34, showing no significant difference. The first trained version of the GM gives 0.51 for both a random population and a reference player in self-play, showing it can learn card selection patterns that increase the overall criteria value, with no significant differences from the tested populations. What was observed is that the trained GM found another shortcut, selecting general cards with no immediate impact, like power buff cards or pure card draws, so it could turn the games longer and have more opportunities to create lead changes. So Hotc was again modified by adding the fifth criterion, Game Length with weight $w_5 = 0.5$, so it would motivate the agent to make games no larger than 40 plays. This resulted in overall criteria of 0.44 and 0.41 for the reference player in self-play and random population. What was observed in terms of card selection is that the agent still chooses cards very repetitively but selects in general cards that help to advance in the game progress, like strong minions.

The results for R2L measured in criteria value are illustrated in Figure 3.18b. Some hyper-parameters were changed in comparison to the player agent - a batch size of 128, a buffer size of 2048, hidden layers of size 64, and $\delta = 0.995$. In terms of results, the baseline random GM shows an average of 0.422 of criteria value over a random population and 0.426 with only the model agent in self-play. With the trained GM, these values changed respectively to 0.438 and 0.497. Since the best case was the trained GM over the scenario with the model agent in self-play, we observed its behavior over some games. We hypothesize that because the handicaps have a random nature, games do not feel as repetitive as in the HotC, and usually, players compete with a similar score. Corresponding to our small increase in criteria score, we were unable to observe meaningful changes in gameplay with a trained GM, as barriers are deployed with high or low levels of intensity and enemy frequency is also deployed without any patterns depending on the intensity of the game. We firmly believe that such a case is due to the lack of control that the selected handicaps give the GM, contrarily to the HotC where most drawn cards are controlled by the GM.



(a) HeartoftheCard criteria average over 100 games.



(b) Right2Live criteria average over 100 games.

Figure 3.18: MDDA-RL Results: B - Base case, or with non-trained GM; T1 - GM trained with criteria set 1; T2 - GM trained with criteria set 2; SA - Only strongest agent acted in self-play; RP - Random pairings over a Population of agents.

The hyper-parameter configurations in terms of criteria values for the GM and parameters for sub-optimal agent populations can be found in Table 3.5.

3.9 Conclusions

In this chapter, we explored DRL architectures for MDDA processes. The base design was maintained through multiple iterations: a GM agent would be trained to automatically deploy handicap mechanisms, supporting disadvantaged players or hindering players in front. To create situations

Table 3.5: Criteria and player agent noise settings during GM training.

Game	Case	w_1	w_2	w_3	w_4	w_5	ab_1	ab_2	η_1	η_2	ε_1	ε_2
HotC	T1/SA	0.40	0.30	0.10	0.20	0.00	35	38	-	-	0.90	1.00
HotC	T2/SA	0.20	0.15	0.05	0.10	0.50	35	38	-	-	0.90	1.00
R2L	T1/SA	0.35	0.25	0.10	0.30	0.00	-	-	0.00	0.15	0.90	1.00

for generalized training, games would be simulated using a population of sub-optimal agents, scenarios where weak versus weak, strong versus strong, and weak versus strong would be created, or where the agents showed different game-playing abilities. Initial solutions were conceived without taking into account good game design principles, creating awkward scenarios, like where for example, the opponent player would be completely blocked and unable to continue playing the game. Therefore, we extend our initial Implicit Model to MDDA-RL, contemplating the use of aesthetic criteria that promotes dramatization during gameplay, creating a desired rubber-banding effect with the appropriate use of handicap mechanisms. The architectures were tested initially in a simplistic 2D and 3D game. Then we further pushed our validation to medium complexity environments: a turn-based card game, and a real-time action survival game in order to show the generality of our framework.

The GM was able to promote the desired criteria in the test bed scenarios. However, the criteria by themselves do not guarantee the resulting game will be considered fun to play. For example, in the case of HotC, the base case resulted in boring games where weak cards were chosen and made the games too long or uneventful. This leads us to consider extra parameters such as a time constraint criterion so we could reach playable games. After this, a human assessment was necessary. For these, we realized a user study where the participants had to play the game under different conditions, comparing play with or without the handicap mechanisms activated. The study was done over the Implicit Model system in a simple 2D game, where it was easy to understand the handicap application. Some participants reported not playing games on a regular basis, and others played them as one of their main hobbies. The reception was overall good, in a scenario where players competed only against an unfairly strong opponent, and where negative handicaps were given in most cases to the opponent, which likely generated positive emotions from human players. It is also notable that the challenge of negative handicaps was also felt by the most proficient players.

The original MDDA-RL framework was inspired by the AI Alignment Problem (Gabriel, 2020). AI alignment is a field that discusses the alignment of what humans intent from an AI agent, versus the translation we give of the problem to the AI. Reward functions are prompt to human error, since humans have difficulty describing micro tasks involved in a greater and more complex task. Christiano et al. (2017) had proposed an architecture for reward modeling, where the reward function itself was a deep model. To train such a model, short trajectories of around one second were stored, and they would be shown to humans in pairs, where humans could vote on which of the two footage they preferred, as it is easier for a human to compare and prefer one thing over the other than to describe in detail what is the intention. MDDA-RL was originally designed

to do this process: the synthetic designer using the aesthetic criteria would vote between pairs of gameplay trajectories sampled from a data set of trajectories, and the reward function trained. In this version it was also concatenated multiple gameplay trajectories, so we could make use of validity criteria, for example, to have a 50% win rate over some sequence matches. Due to the negative results obtained, this element from the training pipeline was discarded, but nonetheless, we find its discussion worthwhile, as a similar process could be used not only for game balance but in general for quality improvement of games, by using human preferences over a given task.

Programming an agent for this type of task is not a trivial task, and as we showed in the RoCa case, a RL agent can more easily learn a strategy to promote criteria versus creating a rules base agent, which will only become harder as the complexity of the game increases, like HotC and R2L. However, making the GM only able to deploy handcrafted handicap mechanisms at the most opportune moments also has its limitations. The GMs trained in this chapter are able to promote situations for players behind to recover, but these mechanisms only work if players are already able to play the game at an acceptable level. This becomes more noticeable in continuous game problems, where action dexterity becomes more important to maintain gameplay, compared to discrete action environments such as card games where dexterity is mostly limited to being able to select cards. This indicates the need to explore ways where the handicap mechanisms can themselves be more automatically designed and generated, and that they must be able, depending on the game, to know how to compensate for certain weaknesses of the human player. It also brings us to the problem of how to represent human difficulties and progression with surrogate agents so that a GM would be able to learn how to create a curriculum for the specific player. This would be applied, for example, for Serious Games scenarios where we adapt to focusing on a specific person instead of a general audience that will play the game. Comparing the approaches, generating a population with evolutionary methods can guarantee some diversity, but will not cause agents to fail or to perform at a certain expected degree in certain sub-tasks. Instead, we can parameterize noise in each of the surrogate agents' actions. Exploring more dynamic deep models to represent human difficulty, together with the generation of suitable handicap mechanisms and the AI alignment problem in game design, will continue to be some of the greater challenges for the MDDA field.

In the context of single-player games, there is a lot of evidence that player experience can be improved dynamically. However, little is known about multiplayer settings. For future validations on developments similar to our framework, and to evaluate the impact of having a human player playing against an AI or human with MDDA, a questionnaire inspired by other accepted and used standards methods should be conceived to: measure flow (Jackson and Marsh, 1996), game experience (IJsselsteijn et al., 2013), perceived challenge (Denisova et al., 2020), and intrinsic motivation (Choi et al., 2010), making the experiments as rigorous and comparable as possible. The motivation is that a lot of questions are made to individual tasks with no intervention from other participants. To better understand and compare the impact of handicaps on human players' gameplay experience, an extensive classification corpus on handicaps could be formulated. This classification would be used to better understand user feedback from handicap mechanisms.

The source code to the Balanceable Maze[2] environment as all test vectors and data sets run in these experiments can be found are openly available. The source code for MDDA-RL[3] is available open-source. The game environments were implemented using Unity Game Engine. To provide interprocess communication between the game environments and our Pytorch (Paszke et al., 2019) implementation, we used the Unity MLAgents Toolkit (Juliani et al., 2020) low-level Applicational Programming Interface (API).

[2] `https://gitlab.com/DracoStriker/balanceable-maze-env`
[3] `https://gitlab.com/DracoStriker/multiplayer-difficulty-balance-rl`

Chapter 4

Algorithms and Ecosystem for Automated Pokémon Team Building and Meta-Game Balance

4.1 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 that 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 of enforcing such requirements in a game is known as balance.

This aspect of pre-game preparation is present in multiple genres of multiplayer games: in collectible card games, players must assemble a deck, choosing strategically the cards from the available pool to compete against other players, with notorious examples Magic: The Gathering (Wizards of the Coast, 1994) both in tabletop and online variants and Hearthstone (Blizzard Entertainment, 2014); in fighting games, players must choose from an array of available fighters and confront the opponent to compare their fighting skill, like in Street Fighter series (Capcom, 1987) and Tekken (Bandai Namco, 1994); in real-time strategic games, players or teams control one or multiple units to defeat the opponent using a vast set of abilities, some unique to certain champions, like Dota 2 (Valve, 2013) and StarCraft (Blizzard Entertainment 1998). Finally, Pokémon (Game Freak, Creatures, Nintendo, 1997) is the largest entertainment franchise in the world, having at its core a role-playing video game series. In Pokémon battles, the trainer's goal is to defeat the opponent's Pokémon team before they defeat his or hers. Then, in VGCs, matches are done as best of three. There is a limited but extensive roster of allowed Pokémon units, which players may select from to build their team.

Strategic interest in the above games results from the fact that some units are good at some things while others at other things, or they are good at different times in the games; which allows

for greater player choice and gameplay variety. If all units were identical, it would make a boring game. However, competitive games are critically evaluated on how balanced they are and on how much diversity of gameplay they can offer, and it is a decisive factor for players' retention.

Carter et al. (2012) discusses the definition of the term meta-game. 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 opponents' behavior and their strategic choices before a game starts. It is important to distinguish between meta-game balance and in-game balance: Meta-game 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 towards victory, without being strictly dominating in a significant quantity of game states.

During the meta-game balance phase of game development, game designers must perform exhaustive testing and gameplay analysis to properly tune the mechanics of each type of unit (cards, fighters, champions) allowing for a greater variety of playable mechanics and strategies in the meta-game. Game balance is easily prone to human error, which frequently causes flawed designs, for example, overpowered units that have a high win rate independent of the opponent unit(s), hence removing diversity from the meta-game. AI agents could assist developers by playtesting thousands or millions of games, detecting the cause of eventually overpowered units and game mechanics to propose or enforce fixes. It is, however, not obvious how to train functions to value units or the composition of units in this type of scenario. Game fixes can result in unpredicted changes to the game dynamics, combined with the high combinatorial of possible unit selections some multiplayer games present, and the fact that multiplayer games usually present intransitive game mechanics, and what makes good units is context-dependent. A balancing agent has to predict and weigh the impact of its adaptation proposals.

AI competitions are a driving mechanism to awake new interest in AI research domains. This was an incentive to design a new model of AI competition where agents must perform meta-game balancing. We called it the VGC AI Competition (Reis et al., 2021), providing a common framework to build and test Pokémon battle agents, team builder agents, and meta-balance agents in Pokémon. The framework runs a self-contained meta-game 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. For the VGC AI Competition, three Pokémon must be selected from a roster generated at the beginning of the competition with an arbitrary size. So, for team-building, the roster is unknown before the balance competition starts; this adds enough complexity that humans are required to perform meta-gaming, observing opponents' team-building trends from public statistics to anticipate their opponent's choices. Inside the VGC AI Competition, this data is provided from the framework itself to team builder agents, containing usage rates of individual Pokémon and teams. For meta-game balance, the roster has to be fine-tuned periodically while a team builder population competes, requiring adapting to the builder agents. We selected Pokémon VGCs as the domain of test for this chapter because it complies with our requirements for a meta-game balance, where teams of Pokémon are a considerably

large but finite number of available combinations (meta-game strategies a player can employ).

One of the core modules of the VGC AI Framework is the PBE, which preserves many of the core elements of Pokémon battling. The environment focuses on the tradeoff between immediate damage rewards against the advantages of delayed rewards through switching or other special abilities. As a competitive multi-agent environment, it has a partially-observable, high-dimensional continuous and categorical state-space, adheres to the Gym (Brockman et al., 2016) *de facto* standard RL interface, and is performance-oriented, achieving thousands of interactions per second in commodity hardware. We partially determine whether deep competitive RL algorithms, WPLθ and GIGAθ, can learn successful policies in this environment.

Previous work assumes the strategies to be balanced already found or that they do not change over time (de Mesentier Silva et al., 2019), which is an unrealistic setting during game design or maintenance stages. We explore a new meta-game 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 (Reis et al., 2023).

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, and a neural network that predicts the win rate of matches. The training schema for the predictor network consists of generating a data set 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 data set 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 main contributions of this chapter are therefore:

- A RL environment that replicates the core gameplay mechanics of Pokémon battles;
- Demonstration of the feasibility of multi-agent deep learning in Pokémon battles;
- The formalization of meta-game and the meta-game balance as an optimization problem;
- Definition and implementation of a multi-track AI competition to develop, train and evaluate all decision processes involved in the meta-game balance process;
- Development and evaluation of multiple baseline team builder agents;
- Development and evaluation of multiple baseline meta-game balance agents (for single Pokémon and team games) using an adversarial approach.

The remaining of this chapter is organized as follows: in Section 4.2 we discuss the related work on Game Design AI competitions, existing Pokémon AI environments, previous Pokémon

Battle Agents, Deck/Team Builder algorithms and automated meta-game balance algorithms; in Section 4.3 we present our developed Pokémon battle RL environment; in Section 4.4 we present our developed VGC AI Competition, its framework's architecture, competition tracks, evaluation model and competition rules; in Section 4.5 we study the application of multi-agent RL algorithms to our Pokémon battle environment; in Section 4.6 we formalize our definition of meta-game and meta-game balance; in Section 4.7 we discuss the developed team builder agents for the VGC AI Competition; and finally in Section 4.8 we apply the strongest team builder agent to develop and evaluate the balance agents for the VGC AI Competition as well. We give our final conclusions and discussion in Section 4.9.

4.2 Related Work

4.2.1 Game Design AI Competitions

We can observe the recent proposals for the annual IEEE Conference on Games (CoG) where most entries are based on real-time video games with perfect information like the StarCraft AI Competition (Čertický et al., 2019), or the Fighting Game AI Competition (Lu et al., 2013). However, some competitions started giving the first steps towards game design and balance tasks, but with major limitations which are discussed below.

The Hearthstone-AI Competition (Dockhorn and Mostaghim, 2019) is an AI Competition based on the popular digital game Hearthstone. Not only does the card game has challenging properties like partially observable states, as the players do not know which cards they are going to draw each turn or which cards are in the opponent's hand and deck, of which there are over 2000 cards each with different attributes and game-changing effects that drastically increases the complexity of the game. Currently, the competitions proposed a game track and a deck-building track, where humans may design the team prior to the competition and optimize their agents to the designer team. The work suggests an automatic deck-build track, which is hard since the cards' power depends on synergies and combinations between themselves during gameplay; and a game-balance track where card rate usage is crucial to choose what are the best deck-building options. The balance could be done at the card pool level or at the rules level (game mechanics). Our target domain coincides with the first, as Pokémons have the same role as units, similar to cards in card games. The Strategy Card Game AI competition (Kowalski and Miernik, 2019) serves the purpose of a more lightweight version of the Hearthstone-AI Competition.

The General Video Game AI competition (Perez-Liebana et al., 2016, 2019) poses the challenge of creating AI solutions that can play a wide range of games. It promotes the devising of algorithms that are able to play any game they are given, even if the game is unknown *a priori*. This highly contrasts with all other competitions which focus on a single game domain. It is built upon the Video Game Description Language (Schaul, 2013), accommodating two-player games, and, more interestingly, provides a challenge for PCG in terms of level (Khalifa et al., 2016) and

rule generation (Khalifa et al., 2017), which is one of the few domains being proposed in automatic game design competitions. The next benchmark for the Pokémon domain would be to also be able to extend game rules (or mechanics) in a balanced fashion, conserving the core gameplay.

Ludii (Piette et al., 2020) is a framework for general board games, where new games can be synthesized. Ludii was proposed as a platform for agent-based competitions, but also for Procedural Content Generation competitions (Stephenson et al., 2019). By using the Ludii framework it is possible to run competitions for game generation, or certain aspects of games (rules, puzzles, or tutorials). It can enforce multiple generational requirements and proposes evaluation by human opinion or agent validation. This contrasts with our game balance competition, where we restrict ourselves to an original Pokémon roster (rule set) and try to convert it into a high-quality roster.

In conclusion, there exist studies for the automatic design of games or game balance, but these are still limited in scoop since they lack a formal execution model and formal evaluation model, which is crucial to further enhance research in these fields through AI competitions.

4.2.2 Pokémon AI Environments and Competitions

Multiple fan-made Pokémon simulators have been developed over the years like the Pokémon Showdown Battle Simulator[1] and the Pokémon Online[2], which focuses on human user-experience. Given the rising interest in ML research in the Pokémon Battling field, an AI competition (Lee and Togelius, 2017) has recently been proposed for the IEEE Conference on Games, known as Showdown AI Competition. The Showdown AI Competition API is a modified version of the Showdown battle simulator, providing methods for agents to act over the full roster and mechanics, but does not propose a game balance track. The authors identify eight main properties that contrast Pokémon Battling with other games:

- Branching Factor - with (on average) nine possible actions per turn (there are four possible moves and five possible switches), planning multiple steps is computationally heavy. This of course is the assumption that in competitive Pokémon battles, the use of items is not allowed;
- Infinite Looping - both agents may choose to switch endlessly without causing any damage to the opponent Pokémon. Although this rarely happens in human games, agents may not identify this cycle to break it;
- Turn Atomicity - although Pokémon is a turn-based game, choices are made simultaneously, and agents only observe both moves after selection;
- Categorical Dimensions - although in a clean state, HP (Hit Points) and statistics are sufficient to evaluate the actions' reward, over-time conditions like *burned* or *paralyzed*, or other field effects like *sandstorm* or *stealth rock* are hard to quantify (delayed rewards);

[1] `https://pokemonshowdown.com/`
[2] `http://pokemon-online.eu/`

- Stochasticity - moves' damage calculation has random parameters, and may miss and do zero damage;
- Hidden Information - this environment is partially-observable at two levels. The opponent's active Pokémon's move set, statistics, and abilities (although this can be minimized with domain knowledge), and at the team level, the unawareness of the opponent's party makes it harder to plan and predict the best long-term strategy;
- Deception - caused by a single ability, where a Pokémon appears in battle disguised as another Pokémon from the opponent's party;
- Computational Cost - a Pokémon simulator, due to having low graphical requirements, can probably outperform many other video-game simulators.

The Showdown AI Competition also identifies two main categories of domain knowledge, which can help the complexity of the large state space. The first is Pokémon typing, which helps to estimate the damage and move value. The other is archetypes, wherein formats where players are allowed to build their teams, knowing synergies between moves, or Pokémons themselves can help to predict the opponent team, but this consideration is out of the scope of this work.

4.2.3 Pokémon Battle Agents

Pokémon Battling agents were developed using multiple AI approaches. Lee and Togelius (2017) tested several baseline agents based on search and learning techniques where Minimax, One Turn Lookahead, and Pruned BFS outclassed the others. Huang and Lee (2019) explores a self-play RL with an actor-critic architecture using embedding layers to capture differences in different move abilities. The agent had success against random teams and learned to pilot three different meta-teams, and the authors mention the use of a strong agent as a helpful tool in procedurally generated teams. The methodology resulted in a good performance against the random, greedy agents and tree-search agents, and human players alike.

4.2.4 Deck/Team Builder Agents

Ward et al. (2021) collected a data set 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. (2021) discusses 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 instead of combinations, and suggests a Contextual Preference Ranking framework, using a Siamese network trained with a triplet loss. Kowalski and Miernik (2020) 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. (2018) explores evolutionary strategies in 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 (Chen et al., 2018) learns a deck search policy and solves deck building with less computation against baseline agents.

Crane et al. (2021) define Team Counter-Selection Games, 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. (2020) explore three optimization techniques, Iterative Local Search (ILS), Genetic Algorithm, 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 meta-game and roster, requiring new approaches for Pokémon team building. The conceived approaches are inspired by the latter two. GA is used to find counter strategies and perform some form of epistemic reasoning over a team selection normal form meta-game, whose number of entries may be highly dimensional. Past choices from a population can be queried.

4.2.5 Automated Game Balance

Andrade et al. (2006) 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 RL agent; and a trained Challenge-Sensitive RL agent. The results showed that the agents with game-balancing approaches performed close to the human player and received better feedback in terms of user satisfaction - demonstrating that game balance is highly correlated with user satisfaction.

Leigh et al. (2008) studied the application of co-evolutionary 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. (2021) 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. (2021) explore how game design methodologies can be applied to engineering design applications.

An unbalanced game is identified through perturbations on important features identified using an explainable AI technique called SHapley Additive exPlanations (Lundberg and Lee, 2017). Moroşan and Poli (2017) focused on tailoring the prowess of dominant strategies using evolutionary techniques, testing in both Pac-Man and StarCraft. Beau and Bakkes (2016) 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. (2016) evaluate multi-objective 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 (Cardona et al., 2014), a simulator and data set of procedurally generated decks. Beyer et al. (2016) discusses automated game balancing in the Business Process Management domain. Moroşan and Poli (2018) discuss 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 define a specification language based on JSON and test GA search over a vector of parameters.

Tomašev et al. (2020) 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. (2020) proposed Deep Player Behavior Modeling (DPBM), where a dataset generated over 6 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 (2019) introduce 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 (Wallis et al., 2021), which was extended to a mobile game and provides an available gameplay data set. Kavanagh et al. (2021) extend this work to a more complex version of RPGLite. Kavanagh and Miller (2021) use state and action lookup tables and action costs, quantifying how much is lost by performing a given action, allowing the evaluation of a player's ability and accounting that information for the game balance process with CSG.

Some of the work indicates 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 is left for future work.

4.2.6 Automated Meta-Game Balance

To address the problem of achieving a balance between the available strategies in a game, Liu and Marschner (2017) modeled this problem considering zero-sum games and proposed a game-theoretic approach to automatically balancing strategies. The authors defined handicap functions that affected the game's payoff table 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 consider a single attribute for each Pokémon. 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. (2019) propose a method for representing the meta-balance and aims to minimize the number of changes through evolutionary search while maximizing the balance. They conclude 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. (2020) present a tool for balancing multiplayer games during game design, using a graph representation of their

target meta-game (they define meta-game 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. (2012) explore 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 author applies an integer-valued genetic algorithm with two fitness functions to promote interestingness and balance. Fontaine et al. (2019) develop a technique called MAP-Elites with Sliding Boundaries, a quality diversity algorithm, and apply it to the design and re-balancing of Hearthstone, as it can uncover complex relationships in gameplay.

Our meta-game scenario differs in that changes are done by adapting to a team-building agent population over a fixed and also non-enumerable number of teams. The major novelty of this work resides in the proposed balance adversarial model: while rational team builders naturally narrow their choices to the strongest teams inside a meta-game context, the balance agents must search for desirable Pokémon attributes to increase motivation for more diverse play.

4.3 Pokémon Battle Environment

In Pokémon battles, a trainer controls one active Pokémon at a time, fighting in a simultaneous turn-based game against the opponent's active Pokémon. Each Pokémon possesses a set of modifiers such as HP, ATTACK, DEFENSE, and SPEED. Each turn, each trainer may select one of four moves for his active Pokémon to damage the opponent's active Pokémon, or the trainer may switch the active Pokémon with one on the bench. When a Pokémon's HP depletes, it becomes unusable for the remainder of the battle and is forcibly switched out. The first player to knock out the opponent's entire team wins. A core mechanic is type advantage, Pokémons and moves possess a type, and types have different effectiveness against different types. A FIRE move is super effective against GRASS Pokémon, and WATER Pokémon resists FIRE moves. The type chart damage multiplier can be found in Figure 4.1. New mechanics were added to the Pokémon battles over the years, but the core mechanics and goals remained the same. Some of the most notable changes of more recent Pokémon generations are the addition of passive abilities, triggered abilities activated under a certain game condition, and various equipable items with a wide range of effects.

4.3.1 Adapted Game Rules

PBE is inspired but has simplified rules in contrast to the official Pokémon video game series. Some remain intact, while others were changed. The rules are as follows:

- The Pokémon type advantage chart follows the official one.
- Pokémon are single type.
- Same Type Attack Bonus (STAB) applies.

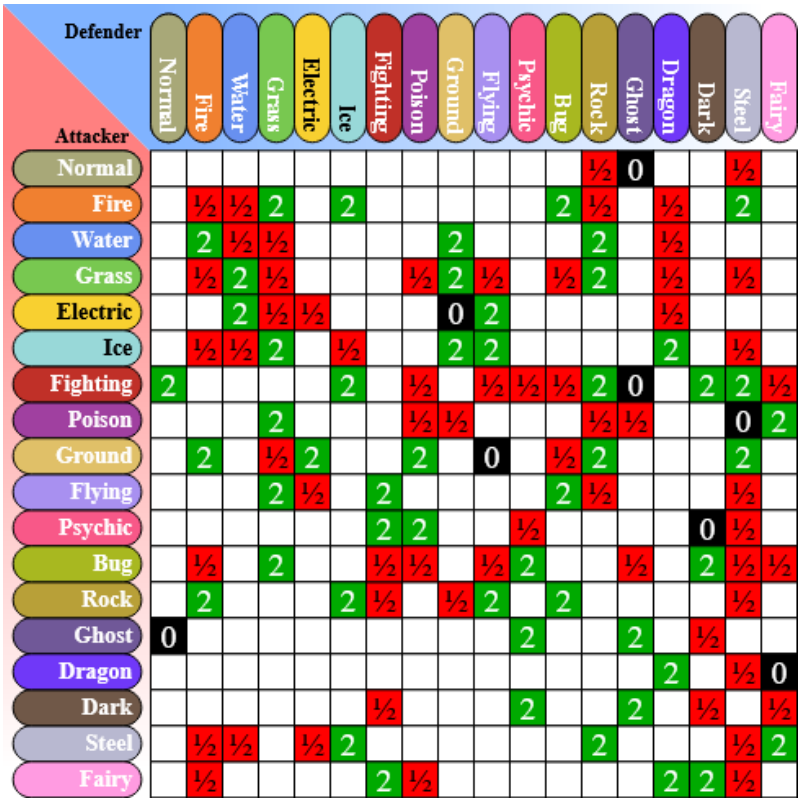


Figure 4.1: Official Pokémon Type Chart Table.
Source: Wikipedia.

- Switch action takes priority over moves.
- Team battle teams are composed of three members.
- Full teams are composed of three Pokémon (however this value is configurable).
- A global weather condition can be set to CLEAR, SUNNY, RAIN, SANDSTORM, HAIL
- Status condition can be inflicted on the opposing active Pokémon. Its possible states are NONE, PARALYZED, POISONED, CONFUSED, SLEEP, FROZEN, BURNED
- While not switched the active Pokémon can have a stage from -5 to 5 in its ATTACK, DEFENSE, and SPEED attributes.
- An entry hazard that deals damage to the new opposing active Pokémon can be set, with possible states NONE and SPIKES.
- Pokémon with the same SPEED stage attacks randomly.
- A Pokémon can faint from an entry hazard process and will repeat until the new active Pokémon does not faint from the hazard damage.
- A Pokémon is guaranteed to wake up, to be confused no longer, or to thaw after 5 turns. It can randomly have his status removed earlier.

- Confusion is the only status that is removed upon switching the Pokémon. Other statuses prevail in a bench Pokémon, but do not perform their effects each turn.
- A move with zero PP is replaced by Struggle (a move with no type).
- When a move is used its PP is decremented.
- If a switch action targets a fainted bench Pokémon, it does nothing, wasting a turn.

In terms of processing, after switching to the new active Pokémon, all entry hazard damage is applied. Then pre-battle effects are processed. These include if status conditions should be removed from active Pokémons. Pokémon status conditions are checked and if they can move, they do so in order. If the first moving Pokémon causes the second one to faint, the latter is not allowed to move. The same logic is applied if their fainting was caused by Entry Hazard damage. Then post-battle effects are processed. These include clearing the weather after finishing the countdown and applying status damage. Then fainted Pokémon are recursively switched, applying Entry Hazard damage and switching again until all Pokémons of one player faint or both active Pokémon are still not fainted. On reset, all special conditions are cleared, and stats reset to the maximum. The full turn execution of PBE is described in Algorithm 5.

Algorithm 5: Pokémon Battle Engine 2.0 Turn Execution

Result: Game State Encoding $E(S)$, Agent rewards R, Terminal T, Game State S
Data: Agents' Actions A

$R \leftarrow (0,0)$, $T \leftarrow$ False;
SetMovesOrder();
$S \leftarrow$ PerformSwitchActions(S,A);
$S,R \leftarrow$ DealEntryHazardDamage(S,R);
$S,R \leftarrow$ ProcessPreBattleEffects(S,R);
$S,R \leftarrow$ PerformMoves(A,S,R);
$S,R \leftarrow$ ProcessPostBattleEffects(S,R);
$T \leftarrow$ AllFainted(S);
while *active Pokémon fainted and not* T **do**
    $S \leftarrow$ PerformSwitchAction(S);
    $S,R \leftarrow$ DealEntryHazardFamage(S,R);
    $T \leftarrow$ AllFainted(S);
end

4.3.2 Pokémon Battle RL Task

Games have been used as test beds for artificial intelligence algorithms because they provide a controlled environment with widely different sets of rules and levels of complexity. They allow researchers to demonstrate properties of their algorithms in single-agent environments (Mnih et al., 2015) (where a single agent competes to successfully finish the game) and in multi-agent systems (where a team of agents must coordinate to achieve a goal (Simões et al., 2018c, 2019b,a), or a

group of enemies must compete for supremacy (Vinyals et al., 2017; Firoiu et al., 2017; Silver et al., 2018)). Pokémon battling falls under the latter category, a competitive 1v1 environment, where each agent (or trainer) controls a Pokémon, turn by turn, to attempt to defeat the enemy.

The properties of the Showdown AI Competition motivates the existence of a Pokémon battle simulator that is compliant with standard machine learning interfaces. It is partially observable since each trainer only knows information about their team and some visible stats regarding the opponent's active Pokémon. Unlike chess or Go, the environment is stochastic, and attacks have a chance to hit and do damage based on an interval formula. It is a multi-agent system, and as the trainer learns and optimizes his strategies, the opponent is learning and adapting as well, a situation known as the moving target problem (Bowling and Veloso, 2001). The reward scheme is zero-sum (Busoniu et al., 2008), since the players' rewards are symmetrical, and the player's only way to win is through the opponent's defeat. The state space is continuous and high-dimensional. To achieve successful policies in a reasonable amount of time, algorithms must generalize to new unseen states, since exploring the entire state-space is intractable.

PBE (Simões et al., 2020b) complies to the *de facto* standard RL API Gym and enables generalized training through random team configurations. Each team is composed of an active Pokémon, and two benched Pokémon. The environment provides an encoded game state. Categorical dimensions are encoded in one-hot vectors and continuous dimensions are normalized. The total dimension of the main objects of the VGC ecosystem is the following (supposing 3 members per team):

- Pokémon Move Encode Length is 42.
- Pokémon Encode Length is 195.
- Pokémon Team Encode Length is 591.
- The full Game State Encode Length is 1188 (2 teams plus a global weather condition).

The 6-dimensional action space of each agent corresponds to the four moves from the active Pokémon and two switch options for the benched Pokémon. A main learning goal in PBE is the optimization of a long-term strategy in favor of a worse strategy with immediate rewards. As such, for RL based algorithms, the environment provides a default reward function given by

$$R = R_d + R_f + R_v - R_t, \tag{4.1}$$

where R_d represents the damage dealt as a fraction of the opponent's maximum HP, $R_f, R_v \in \{0,1\}$ is a bonus if the opponent fainted or victory over the opponent team was achieved, and R_t represents the damage taken as a fraction of Pokémon's maximum possible HP. With unclean states (effects are active), recovery moves, and recoil moves, R_t is the sum of all damage taken, by moves, recoil, status condition, weather condition, or entry hazards.

Comparing the PBE with the one from the Showdown AI Competition, in terms of AI challenge, we identify a set of differences and similarities. The branching factor was decreased, and

the Deception ability does not exist, which creates a simpler environment. However, stochasticity is heavily exacerbated with a random set of generated Pokémon and moves, instead of a fixed set of viable combinations and teams. PBE also maintains multiple similarities, since turn atomicity is preserved, infinite looping remains possible, and the environment remains partially observable.

Overall, the most important components of the core gameplay of Pokémon battles are replicated in a controlled environment, and for the VGC AI Competition where a roster must be tuned, balance agents must be able to propose fixes, and the roster must be unpredictable before the competition, as balance fixes may be hard coded by human competitors. The Pokémon Showdown and the Pokémon Online work with the official Pokémon roster, not complying with our requirements, motivating us to integrate PBE into the VGC AI Framework, where randomly generated Pokémon can be deployed in battle.

4.4 Pokémon VGC AI Competition

The VGC AI Competition aims for the development of player agents and meta-game balance agents and their comparison through individual or composed tasks. Player agents compete in an ecosystem based on human VGC competitions, and balance agents must tune the roster so to increase strategic expressiveness. The VGC AI Framework is the software structure that allows running each component of the VGC AI Competition, its main purpose is to allow players to design and train their models in the same environment where the competition itself is run. The framework emulates the main tasks of competitive Pokémon:

- Team Selection;
- Pokémon Battling;
- Team Building;
- Meta-game Balance.

Each task differs in nature, and there may be a need to coordinate different techniques. The framework abstracts all the logic behind managing the ecosystem, allowing the user to focus on the development of each task. Each task may be developed and tested in isolation. The motivation behind this design decision is that isolated contributions to each task can further advance new knowledge on other tasks as well. The ecosystem is illustrated in Figure 4.2. In this illustration, the Battle Layer constitutes both team selection and Pokémon battling tasks sequentially.

We segmented the framework description into three parts. First, we introduce the core data structures (or artifacts) of the framework, which are manipulated by the system and software agents. Next, we describe the architecture, the main modules, how they interconnect, and how they can be run in isolation. Lastly, we describe the framework API, so competitors know how they can develop the player and balance agents.

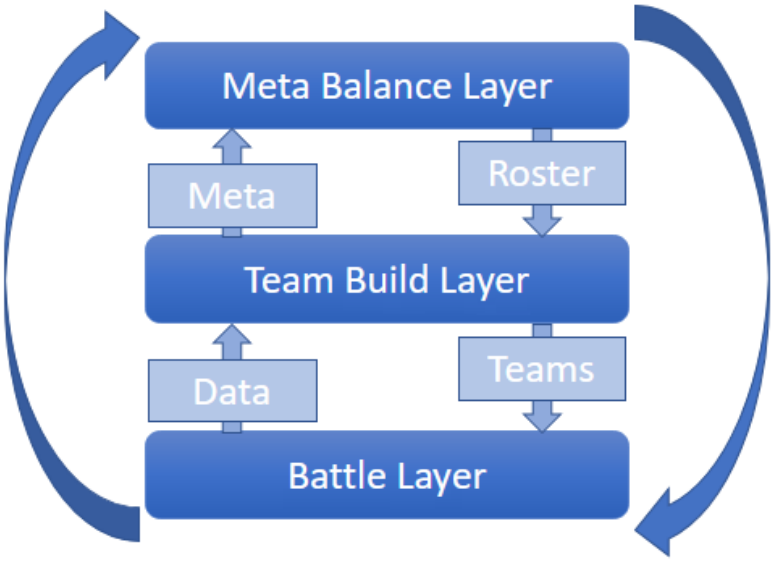


Figure 4.2: VGC AI Framework Ecosystem.

4.4.1 Data Model

The core concepts of the framework and their relationship are illustrated in Figure 4.3. A Pokémon Template defines the possible moves a Pokémon specimen may have, what is the range of values their attributes can have, and so on. A Pokémon is an instance of a template with concrete moves and attributes configured. A Pokémon Team is composed of three different Pokémon (a team may not contain two or more instances of the same template). The Move Roster defines which moves are legal as the Pokémon Roster defines which Pokémon Templates are legal for the duration of a competition. A Meta-Data contains the usage history of team/individual Pokémon and moves usage rates for the decision-making of players and balance agents. The Game-Data structure, during a competition, is automatically updated by the framework. Pykalitics[3] is a free online analytic platform that aggregates information from VGCs to assist human players in building their teams. It provides Pokémon usage statistics and moves, items, abilities most used for a given Pokémon, and most common team compositions and was the main inspiration for this meta-data model.

4.4.2 Architecture

We name each major part of the ecosystem a module, which runs sequentially and/or concurrently with other modules, depending on the case. Each module is incorporated with logic to manipulate the core data artifacts through processes, whose outcome is dependent on behaviors' decisions. Behaviors and processes are connected through interactions, where the process gives the AI behavior observation, and the AI module replies with an action. Processes are connected by a logic flow and by I/O channels between modules. The four major modules from the framework are:

[3] `https://www.pikalytics.com/`

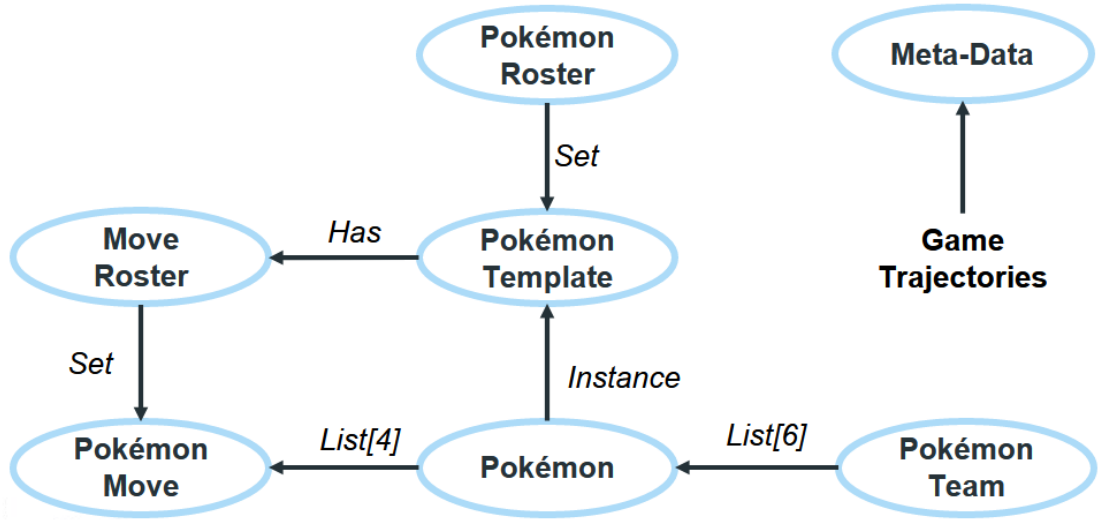


Figure 4.3: VGC Framework Data Model. Game trajectories here correspond to the sampling of teams used in battles.

- Selection Phase;
- Battle Phase;
- Team Building Phase;
- Meta-Game Balance Phase;

In the Selection Phase, given the full-player team, partial information of the opponent team (in human VGC competition the players have information about the specimens the opponent has before selecting a sub-team to compete; to simulate such a process in a system where Pokémons are randomly generated some of their information acts like a signature, for example, typing and HP, and is revealed at this stage) and meta-data the aim is to choose the most optimal sub-team selection. This is done in two steps. First, by using Meta-Data and partial information (view) of the opponent's team, we predict the opponent units. Given the predicted team, we choose the team that best suits the opponent. The Selection Phase is illustrated in Figure 4.4.

In the Battle Phase, in which a player agent competitor, entering with their selected team, must defeat the opponent. In sum, we make a desired number of runs of the PBE (here denoted the Pokémon Battle Engine). Turns snapshots are stored in the form of trajectories that are used to update knowledge we have about the opponent's team. The ecosystem's meta-game data is updated with the Playing Team of each player. The Battle Phase is illustrated in Figure 4.5.

Both previous modules make use of a Team Predictor behavior, which allows us to reduce the workload of the Battle Policy, where it can assume it has perfect information about the game, while we leave the burden of filling the hidden information to the Team Predictor, which can be reused in the Selection Phase as well because, in matches done at the best of three, we can re-select our sub-team before each battle.

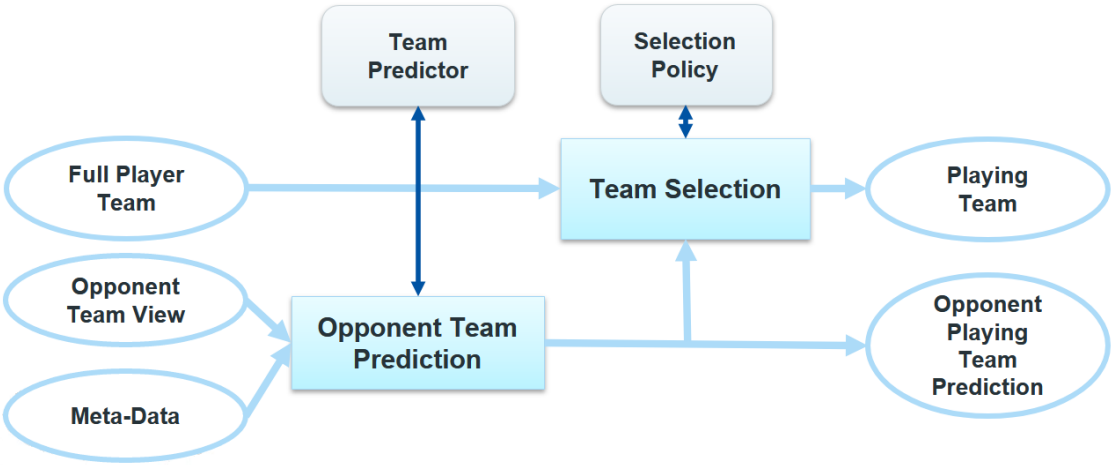


Figure 4.4: The Selection Phase is constituted of two processes, each interacting with a respective behavior. The Team Predictor behavior predicts the opponent team structure. The prediction is provided to the Selection Policy to choose strategically the initial active Pokémon and party (non-active Pokémon team members) against the opponent. This process is a one-shot run where there are no loops in the logic flow.

The next is Team Building, where between battles each player is given the opportunity to analyze the meta-game and restructure their team. When ready the player can engage in the Competition Ecosystem. The Team Building module runs concurrently with the Competition Ecosystem, where players are paired with other players to perform multiple times sequentially the Selection Phase and the Battle Phase. Competitors are put in a match queue in the Competition Ecosystem, and after being paired, they battle, and meta-game information is updated after the fact. When allowed, a competitor may leave the battle queue and proceed to a team-building phase. The Team Building is illustrated in Figure 4.6.

Lastly, in the Meta-Game Balance module, we aim to tune the Pokémon roster by analyzing an ever-evolving Championship Ecosystem (consisting of the Team Building Phase) where player agents build teams and battle against each other. Using that information, a balancing agent makes changes to the Roster in an online fashion aiming to maximize some balance metrics while subject to design constraints, both must be set by the organizers before the competition. When the balance agent changes the roster, every player agent is interrupted and moved to the Team Building Phase with their teams reflecting the changes of the roster, i.e., changed moves or even banned Pokémon. The Meta-Game Balance is illustrated in Figure 4.7.

4.4.3 Competition Tracks and Rules

To compete, participants must submit an agent that abides by the VGC Framework Competitor API containing the multiple categories of behaviors previously discussed. As illustrated in Figure 4.8, depending on the role of the agent (player or balance/designer), it must contain specific behaviors and will act at different levels of abstraction. For a detailed description of every method and class of the VGC API refer to Section A.3.

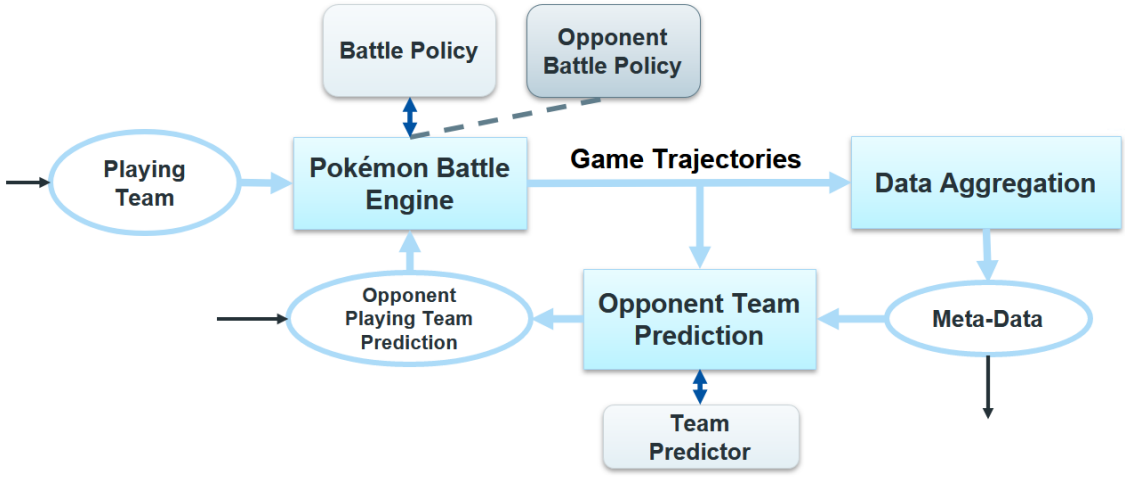


Figure 4.5: The Battle Phase is composed of three processes, the Pokémon Battle Engine which interacts with a pair of battle policies, a Data Aggregation process, and the Opponent Team Prediction process. Game states and joint actions are propagated in trajectories and delivered to Team Predictors to update their opponents' team predictions. At the end of the battle, the used teams are sent to the Data Aggregation process which will update the meta-data (usage rates of Pokémons, moves, teams, etc). There is a loop in the information flow, the Pokémon Battle Engine runs until the game reaches a terminal condition.

Participants may choose to compete in one or more of the proposed tracks, where each tackles different challenges on the VGC AI Competition. In the Battle Track player agents only competes with battle policies. In the Championship Track, competitor agents must have mastered the skills of battle, team selection, and team building. In the Balance Track, balance agents must tune the roster better than their opponents which manage other VGC AI Framework instances. Participants only need to submit a competitor agent with the minimum required behaviors for the tracks they are participating in as specified in Table 4.1. Modules can be reused for multiple tracks.

The Battle Track focused only on the battling aspect of Pokémon. The submitted player agents must be able to play with any given random team against any challenging random team. The winner is determined by the outcome of sequential battles organized in a tree format. Teams are randomly generated at each round for each match. To assert fairness, generated teams are switched during a match, so players can compete under the same conditions. There is a time limit for deciding a move action during a Pokémon battle (value still under consideration). The

Table 4.1: Competition Requirements.

	Battle Track	Championship Track	Meta-Game Balance Track
Team Predictor	-	X	-
Team Selection Policy	-	X	-
Battle Policy	X	X	-
Team Builder Policy	-	X	-
Game Balance Policy	-	-	X

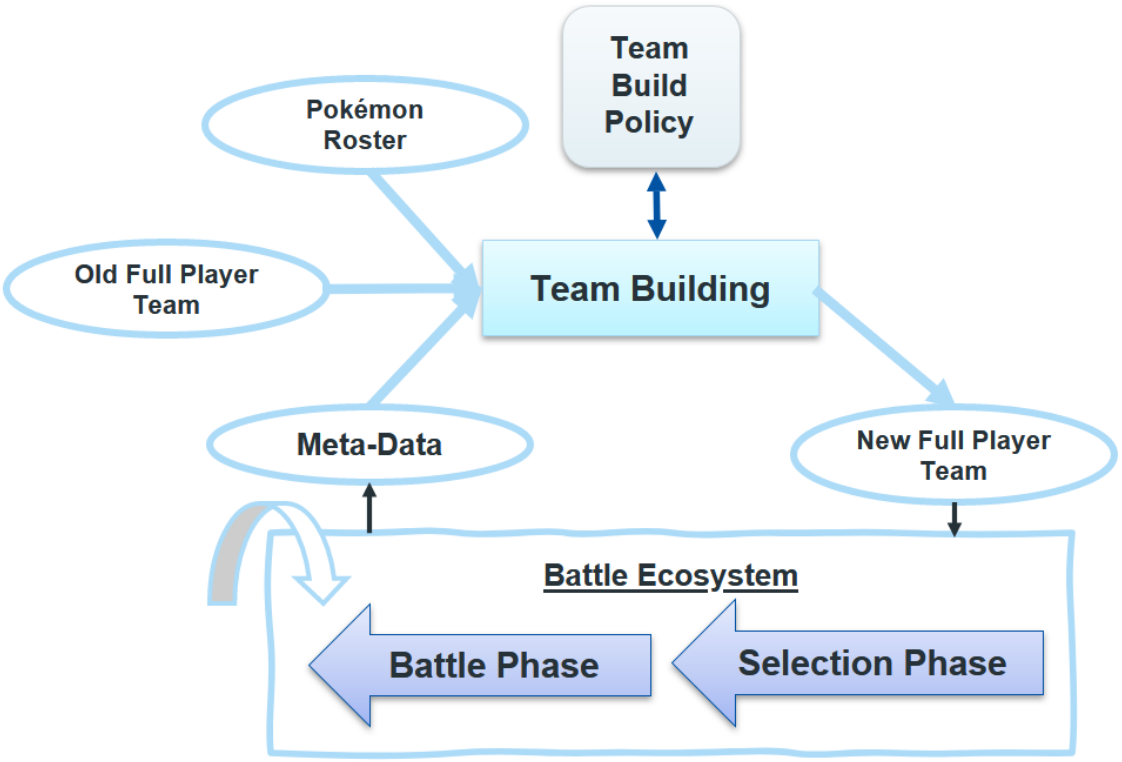


Figure 4.6: The Team Building Phase has only a Team building process and Team Builder Policy module which has access to the Pokémon Roster, Meta-Data, as it needs to know what changes can be made to their team and its current team for reference. It outputs a (possibly changed) team. The framework provides mechanisms to validate the output team, which may be rejected and changed back to its previous configuration in the case an illegal team is outputted. Is a one-shot run where there are no loops in the logic flow.

randomly generated teams will follow all the bellow rules for legal teams in the Championship Track.

The Championship Track focuses on team-building skills in the presence of an ever-evolving meta-game. Player agents must be able to adapt as quickly as possible to the meta-game by choosing the best teams to compete against the teams they believe they will be facing. A fixed roster is randomly generated at the beginning of the competition that dictates the legal team compositions. There is a preparation phase where agents are given the opportunity to engage in battle during many epochs and may regularly change their team. As for the rooster not becoming fully random and unpredictable, some rules were established that define the set of possible Pokémon and templates:

- Pokémon teams are not permitted to have duplicate specimens.
- Individual Pokémon must have points distributed along stat slots so that the total sum of move powers and effects does not exceed a predetermined maximum.

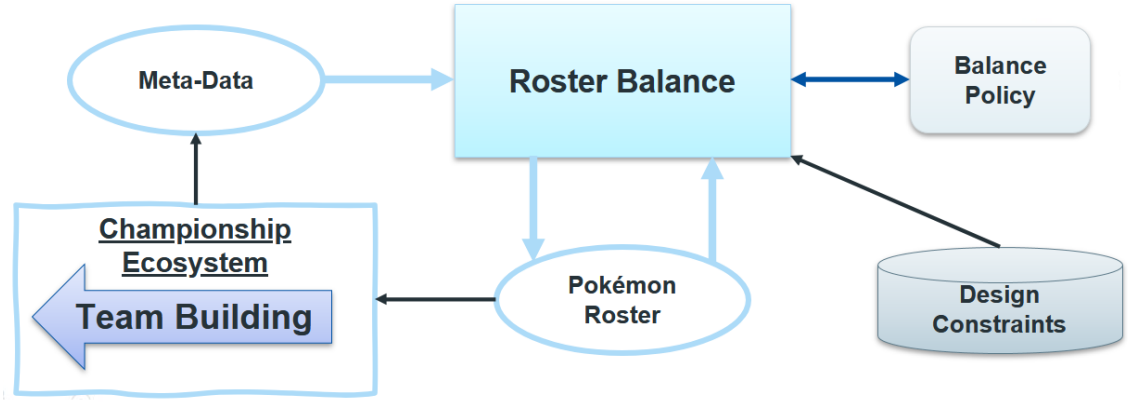


Figure 4.7: The Meta-Game Balance Phase. The roster balance runs concurrently with a Championship Ecosystem. Organizers can impose a set of design restrictions on the agents, i.e., cannot fully change the Pokémon properties or quantity.

  - The type of the Pokémon and the type of each of its moves are not considered for the total points.
  - Each move with an effect is worth one point.
  - Each 30 HP and each 30 power among the moves from the Pokémon are worth one point.
  - Each Pokémon can be given a maximum of 11 points.
  - Each Pokémon will have base 120 HP and base 30 Power for each move that does not count toward point distribution.
  - Each Pokémon maximum HP is truncated to 240.
  - Each move only has a maximum of one effect.

- For the Battle Competition Track, the randomly generated teams will follow all the above rules.

- For the Championship Competition Track, a roster of 51 Pokémon.

The evaluation in this track is through Elo rating, players will play a defined number of rounds and can be paired randomly or by their rating, and the winner will be the player with the highest Elo. The initial Elo for every player is set to 1200.

Being a fairly new type of problem some complexities have been alleviated and depending on the feedback received in future runs of the competition they may become the norm:

- Pokémon Full teams are only composed of three members, and team selection and prediction are skipped. This means that Battle Teams will directly correspond to their Full Teams. Battle agents compete with their constructed teams, with team order prevailing during Championship Ecosystems as they are built. Player entries are therefore not required to have a Team Predictor or a Team Selection Policy.

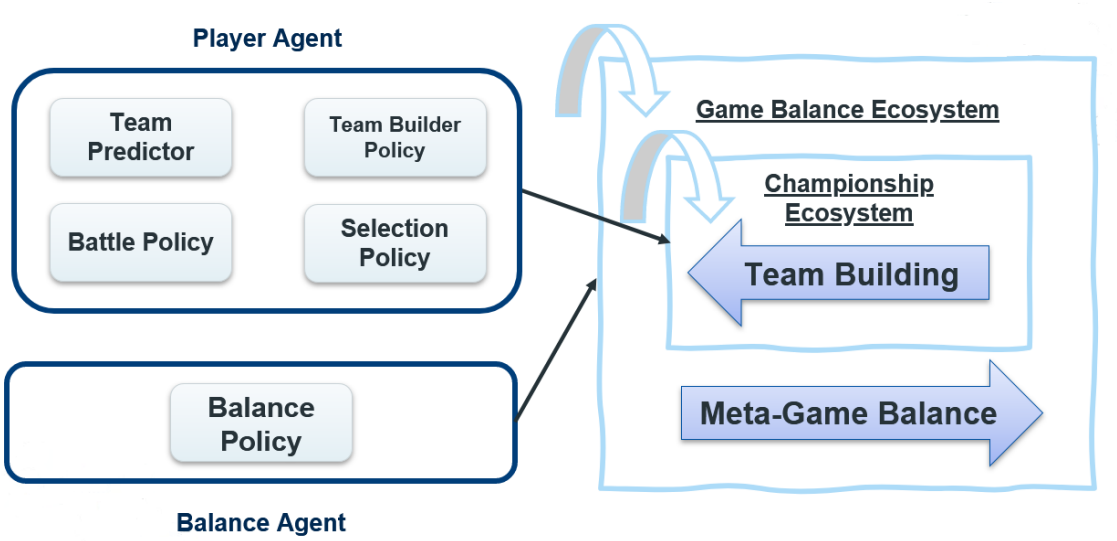


Figure 4.8: The VGC Framework Competitor API. A player agent acts in the VGC Ecosystem environment, while a balancing agent acts in the Game Balance Ecosystem environment. Depending on the role, only a selection of behaviors is needed to be integrated into the agent.

- Pokémon Templates generated for this competition will only contain a selection of already defined moves. This means the generated Pokémons will be set with a certain Type, HP, and a subset of four moves from the standard move roster. Since the moves are predetermined and only four moves will be available for each Pokémon Template the HP stat will be determined by calculating the remaining points from the sum of the moves' points. The team build will be reduced to just select 3 specimens from the 100 that will be available, no configuration of their individual moves is needed.

In the Meta-Game Balance Track, a competitor enters with a balancing agent and is assigned to maintain a roster in order to provide a diverse meta-game. Each competitor is given the same number of run epochs, and evaluation points are accumulated by the end of each epoch with a balance fitness function (refer to Section 4.6 and to Section 4.8). The roster must at all times abide by a set of randomly generated constraint rules before the competition starts (in this case the set of rules from Championship Track is included but more can be added, see Section 4.4.4). To assert fairness and ease of evaluation, each contestant will be given the same initial roster and design rule set. The player who scores the most points over time will be the winner. Each violation on the design roster will be ranked down. Competitors with the same number of violations are tiebreak by the most score. Organizers must provide the player agents, which should be as close as possible to theoretically optimal, to run inside the Championship Ecosystem.

4.4.4 Design Constraints

Restrictions have the role of game semantics for players, i.e., the changes should not make a Pokémon unrecognizable. Human players' perception of balance is usually correlated with the meta-game's diversity, of which there are many types of specific team configurations, Pokémon

usage moves usage, type predominance, types of moves, abilities, what archetypes are most relevant, etc. A balancing agent must perform global optimization over many parameters, eventually some with priority over others. Therefore, we need formal metrics to evaluate the balance task performance.

Typical design changes are done by adding new elements, removing existing ones, or tuning the parameters of existing ones. In the case of the VGC AI Competition, moves' attributes can be tuned, moves can be added or removed from a Pokémon template, or attributes from the template could be re-adjusted. But, fixes to the roster could result in uninteresting results, we could want Pokémons to maintain their main characteristics, so only moves could be changed but not their type (due to some aesthetic characteristic).

For the purposes of enabling competition organizers to define the design restrictions and balance metrics, we implemented the Design Constraint Language. Restriction rules can be applied to every Pokémon or only to specific templates. These rules can, for example, disable changes on move sets, types, and other attributes, or limit how much a Pokémon can be modified (the degree of change can be measured by an information distance metric). We can also define global parameters like the Pokémon roster size and move roster size. See Figure 4.9 for an example.

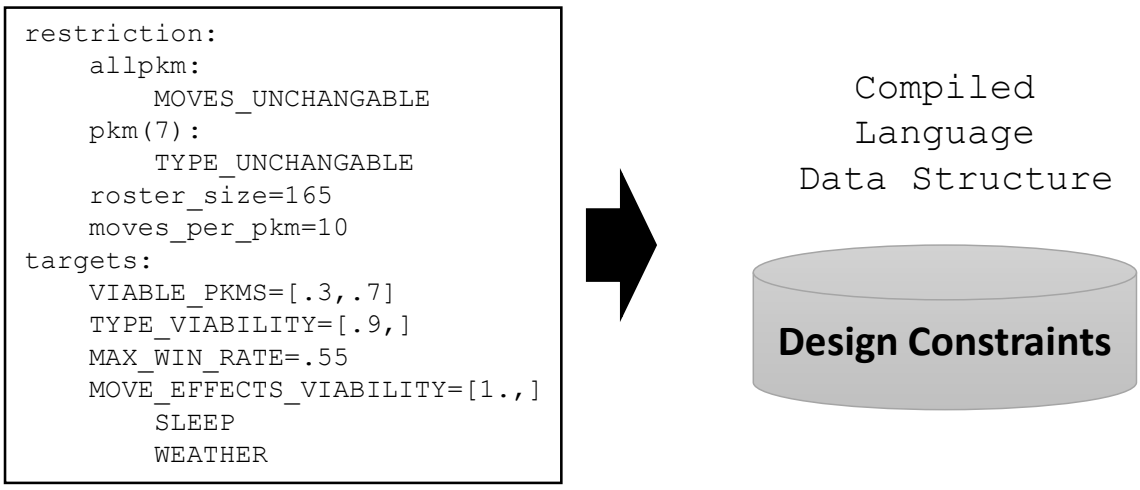


Figure 4.9: Design Constraint Language Mock Example: Moves cannot be changed for any Pokémon, and template 7 cannot have its type changed. The target of balance is to have a viable population between 30% and 70%, type viability of 90%, max win-rate for Pokémons and/or teams is 55%, and Sleep and Weather effects must be present at least in 10% of teams used. The configuration language is compiled for a data structure for the VGC framework.

4.5 Asynchronous Adversarial RL in Pokémon Battling

Experiments were conducted for agents to learn strategies that include type effectiveness and tactical switches over an early version of PBE. Here, each team is composed of six Pokémon has at least one move of its type and three moves of random types that are not effective against the Pokémon and that the Pokémon is not effective against. For example, a FIRE Pokémon will not have

a WATER attack (effective against FIRE) or a GRASS attack (which FIRE is effective against). While some specific Pokémon have attacks with unconventional types, removing these from the PBE leads to more strategic policies learned by agents than if a Pokémon could have fully-random type moves. Finally, all moves are pure damaging moves with no additional effects.

The GIGAθ and WPLθ algorithms are competitive mixed-policy deep-learning algorithms that require only local information to converge to NE policies, specifically targeted at competitive environments and have been shown to allow players to converge to the NE strategy in simple games (Simões et al., 2018b). The algorithms have no prior knowledge about the environment.

The asynchronous mixed policy training sessions trained in self-play for two million episodes, used 12 concurrent workers, a neural network with three hidden layers of 150 nodes with ELU activations (Clevert et al., 2016), a learning rate $\alpha = 10^{-4}$, a policy update rate $\gamma = \frac{\alpha}{200}$, and a future reward importance discount $\delta = 0.9$. The ε-annealing technique was used, following the scheme used originally by Mnih et al. (2016), with each worker annealing the exploration rate from 1 to one of $0.5, 0.1, 0.01$ over 1.3 million episodes. Weights were initialized with the Glorot initializer (Glorot and Bengio, 2010) and optimized with the Adam optimizer (Kingma and Ba, 2017).

The learning results are shown in Figure 4.10 and compared against the hard-coded baseline that followed the guidelines given by two human experts. The hard-coded solution emulates an expert player's decision-making, maximizing the damage output of the team, based on the known information about the opponent's team. Both GIGAθ and WPLθ achieve good results in the 7-type PBE, but GIGAθ converges faster and can match the performance of the hard-coded baseline. In the complete version with 18 types present, WPLθ is no longer able to learn adequate policies. On the other hand, GIGAθ has a slower evolution with higher variance, likely due to the increased complexity of the environment, but still matches the performance of the hard-coded solution.

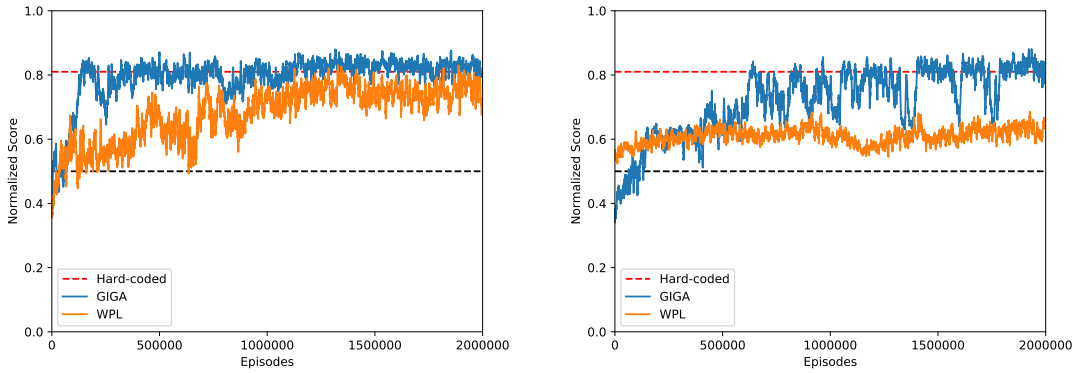


Figure 4.10: Results of GIGAθ and WPLθ for the PBE, in an unfair test scenario. The plots represent the normalized score of each algorithm, trained in self-play but tested against a random agent, over training episodes. Because the initial state places the agents in a disadvantageous position, a weak agent (such as a random agent) would obtain a normalized score below 0.5. The plots also show the hard-coded baseline given by expert players.

Both algorithms were tested in a fixed scenario, where a trained agent in a disadvantageous position played against a random enemy, initially favoring the opponent with a FIRE/NORMAL roster, while the trainer has a GRASS/WATER roster. Because FIRE is effective against GRASS, and WATER against FIRE, the allied trainer's first move should be to switch Pokémon, favoring long-term rewards. After the switch, the WATER Pokémon has both WATER and FIGHTING attacks (effective against FIRE and NORMAL Pokémon, respectively), and should pick those moves with high priority. An example is shown in Figure 4.11, demonstrating how the converged GIGAθ trainer behaves in the unfair scenario. The trainer prioritizes strategic choices and exploits type advantages, by following the policy described previously. Without any previous domain knowledge, it learned how FIRE is vulnerable to WATER, WATER to GRASS, and NORMAL to FIGHTING, and makes strategic decisions while using effective moves when possible.

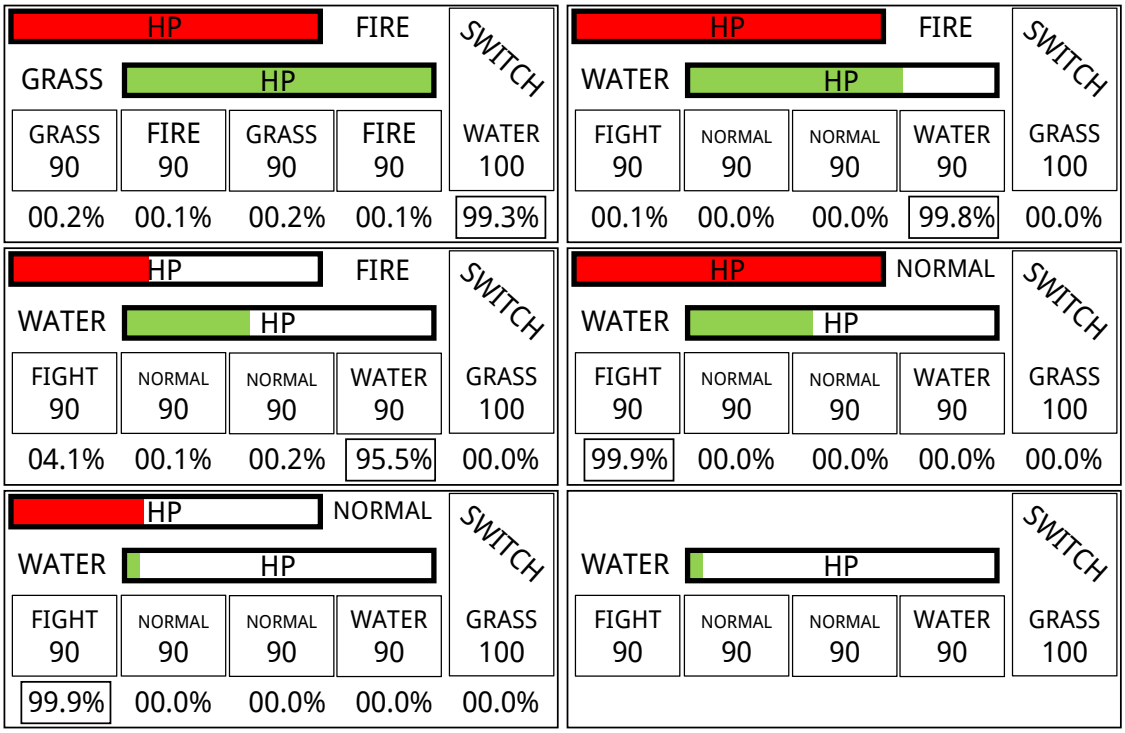


Figure 4.11: A converged GIGAθ agent's policy in a fixed test scenario. The initial setting is as follows: Opponent Pokémon is FIRE-type, and own type is GRASS, and we can observe our Pokémon moves and type of the benched Pokémon. The policy of the agent for each state is shown below, with the chosen action highlighted. The opponent's Pokémon uses attacks of their own type every turn, but the initial switch gives the local agent an advantage (switch from GRASS-type to WATER-type to fight the opponent FIRE-type), and it wins the battle with no casualties.

Adversarial deep RL algorithms were able to learn type-advantage and long-term strategies without any *a priori* domain knowledge. Switching to a benched Pokémon which has better typing against the opponent's active Pokémon may lead to a more efficient defeat of the opponent Pokémon, even though the short-term reward is negative. WPLθ has been shown to outperform GIGAθ in heavily stochastic environments, but PBE commonly provides agents with a deterministic optimal choice, and GIGAθ was able to learn faster than WPLθ. GIGAθ was also the only

algorithm that learned a successful policy with all 18 Pokémon types, achieving a 0.8 normalized score and a 100% win rate in our test scenario. For future work, we aim to train a deep RL agent with the same level of performance in the current version of PBE, whose observation space dimensionality increased 10-fold, and the game is much more complex with the introduction of status effects, weather and the other discussed special conditions.

4.6 Meta-Game Balance: 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), 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 meta-data 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 meta-game $\mathbf{M}_{i,j} = \mathbf{M}[i,j] = w(\psi_i, \psi_j)$ 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 meta-game of individual Pokémon and $\mathbf{M}_t$ as the meta-game whose row and column entries are teams.

Assuming the meta-game 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 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 (2020), 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 NE (a joint strategy where players have no advantage to deviate alone from) a significant number of pieces has a reasonable chance of being picked (de Mesentier Silva et al., 2019). 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.

4.7 Team Builder Agents

In this work, the battle policy π is set as a search algorithm based on Minimax from Lee and Togelius (2017). The advantage of such an agent is that allows us to change the core game without needing to re-train the battle agent. The disadvantage is the computation time, slowing down battle simulations.

The outcome of Pokémon matches depends on a meta-gaming 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 sub-optimal plays of the opponent. We set all the battle agents as the mentioned 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. (2018) where a small set of pre-trained agents with specific game styles were 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 super-human agent able to perform all the mentioned skills, which is outside the scope of this work.

4.7.1 Nash Equilibrium and Linear Programming

Since we are dealing with a two-player symmetric constant-sum game (of value equal to one, the sum of win rates), linear programming can be employed to find the min-max solution in polynomial time, with theoretical guarantees of the uniqueness of the solution. We use the `linprog` procedure of SciPy (Virtanen et al., 2020), which solves problems in the form:

$$\begin{aligned} \max_{x} \quad & c^T x \\ \text{such that} \quad & A_{ub} x \leq b_{ub} \\ & A_{eq} x = b_{eq} \\ & lo \leq x \leq up, \end{aligned} \tag{4.2}$$

with $x = \langle w, s_1, \ldots, s_n \rangle^T$ is the vector containing the decision variables, w in particular is the expected payoff of the strategy profile $\langle s_1, \ldots, s_n \rangle$ and corresponds to the win rate. $c, b_{ub}, b_{eq}, lo, up$ are vectors and A_{ub}, A_{eq} are matrices. A_{ub}, b_{ub} define the upper bound inequalities constraint terms, A_{eq}, b_{eq} define the equality constraint terms and lo, up define the lower and upper bounds that x can take in the problem space, c defines the coefficients of the linear objective. We want to optimize $c^T x = \mathbf{w}$. The sum of the strategy profile $\langle s_1, \ldots, s_n \rangle$ must give $\sum_i s_i = 1$. To convert this problem

to find the NE of a normal-form meta-game we set the following:

$$\begin{aligned} c &= \langle 1, 0, \ldots, 0 \rangle^T \\ A_{ub} &= \left[\begin{bmatrix} 1 & \ldots & 1 \end{bmatrix}^T | \mathbf{M} \right] \\ b_{ub} &= \langle 0, \ldots, 0 \rangle \\ A_{eq} &= \begin{bmatrix} 0 & 1 & \ldots & 1 \end{bmatrix} \\ b_{eq} &= \langle 1 \rangle \\ lo &= \langle \infty, 0, \ldots, 0 \rangle \\ up &= \langle \infty, \ldots, \infty \rangle, \end{aligned} \tag{4.3}$$

Since the game is symmetric, $\mathbf{M}$ corresponds to one of the players' payoff matrices of the normal-game bi-matrix.

4.7.2 Team-Build Algorithms

Our first team builder agent is an extension of Weighted Probability Team Generator (Crane et al., 2021) (which we denominate Individual Pokémon Counter). It employs epistemic reasoning, maximizing its win rate while trying to remain unpredictable (see TABLE 4.2, where p_0, p_1, p_2 are individual Pokémon). The usage rates are weighted with the win rate table between pairs of teams and summed. The final weights are used for a Softmax policy.

Table 4.2: Individual Pokémon Counter.

Roster R	Meta-Game $\mathbf{M}_p$			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

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 6) 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 match-up against. When it selects the next Pokémon it weights each Pokémon match-up vector (or meta-game 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, and the threshold τ determines which is the minimum win rate for strategies to be considered for a reduction in the coverage vector.

Algorithm 6: 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 $M_p[i,j] \geq \tau$ **then**
        $\mathbf{c}[j] \leftarrow \rho \mathbf{c}[j]$
    end
end

Minimax Builder (see Algorithm 7), assumes opponents are playing rationally, and so the NE of the normal-form meta-game $\mathbf{M}_p$ 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.

Algorithm 7: 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

Individual Pokémon Counter, Maximize Pokémon Coverage and Minimax Builder do not exploit more intrinsic game properties like teammate synergies besides individual Pokémon pairwise match-up advantages. Our second set of algorithms considers full teams as pieces of the meta-game. A meta-heuristic search method like 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 8 solutions and 2k generations resulting in 16k look-ups.

Meta Counter (see Algorithm 8) 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 meta-

game row entries with each team usage rate under entries T_M,

$$F_{t_1}(t_i)\{T_M\} = \sum_{t_j \in T_M} \mathbf{M}_t[i,j] \times \mathbf{u}_t[j]. \tag{4.4}$$

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,

$$\mathbf{u}_t[i] = \frac{1}{|t_i|} \sum_{p_j \in t_i} \mathbf{u}_p[j]. \tag{4.5}$$

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 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 instead a meta-independent algorithm like Minimax Builder is used during the first iterations of the team building cycle to select the opponent meta teams.

Algorithm 8: 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 u_t ← u_t[j] for each non-repeated permutation t_j of H
    Sample Meta Teams T_M ← Softmax(u_t)
else
    Sample Meta Teams T_M ← MinimaxBuilder(R)
end
t ← GA.Optimize(F_{t_1}(T_M))
```

Maximize Team Coverage (see Algorithm 9), instead of finding an overall counter team against all teams, it searches separated optimal teams against each meta team. Then fills $\mathbf{M}_t$ with the pairwise win rates of all meta teams and counter teams and computes the NE solution as selection policy.

4.7.3 Deep Match-up Predictor

The win rates of the meta can be determined empirically by running an arbitrary number of game-play 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 parametrized predictor function f_{θ_1} that given two teams should output

Algorithm 9: 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 $t \leftarrow \pi_{M_t}$

the predicted win rate of each. Previous tries showed success in predicting the outcome using supervised learning in competitive games (Argue, 2014). To train this model, we generate a data set of random team pairs piloted by the battle policy π, and determine their average win rate. For function f_{θ_1} we input team pairs t_0, 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, t_1 is w then the win rate of t_1, 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 as such team pairs t_0, t_1 where $t_0 = t_1$ were excluded from the training data set. Since it is iterated over non-repeating pairs of teams, the actual number of inference computations of f_{θ_1} is less than $(|T|-1)^2/2$ to fill a meta-game $\mathbf{M}_t$.

Previously a pyramid topology yielded good results (Do et al., 2021) for match prediction between champions on League of Legends, alternating dropout, normalization, and dense layers. Betley et al. (2018) considered two approaches, training a regression model predicting the win rate of each deck for a Hearthstone data set and a classification model predicting the result (win, lose). They opted for the latter since the result data set had 300k samples vs 400 samples for win rate. We follow the pyramid architecture for f_{θ_1} and generated a data set 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 ReLU (Agarap, 2019) activation function in the hidden layers, trained with Adam optimizer (Kingma and Ba, 2017) and used Kaiming Ha initialization (He et al., 2015) with a bias component of 0.01. Four feed-forward 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 size two. 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 match-ups, where we adjusted to smaller network layers (one-third compared to the team prediction) and a data set 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).

4.7.4 Deep Q-Mapping

For Maximize Team Coverage we also experimented to train a generator model f_{θ_2} similar to GANs (Goodfellow et al., 2014), 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 MDP of a single step, so we verify that in the Q-Learning target equation

$$Q(s,a) = r, \tag{4.6}$$

where the target is calculated simply using the immediate reward r, and therefore no need for a target network, replay memory, or other state-of-the-art tricks (see Figure 4.12). State s corresponds to the opponent strategy (Pokémon or Team) and actions a to the constructive elements.

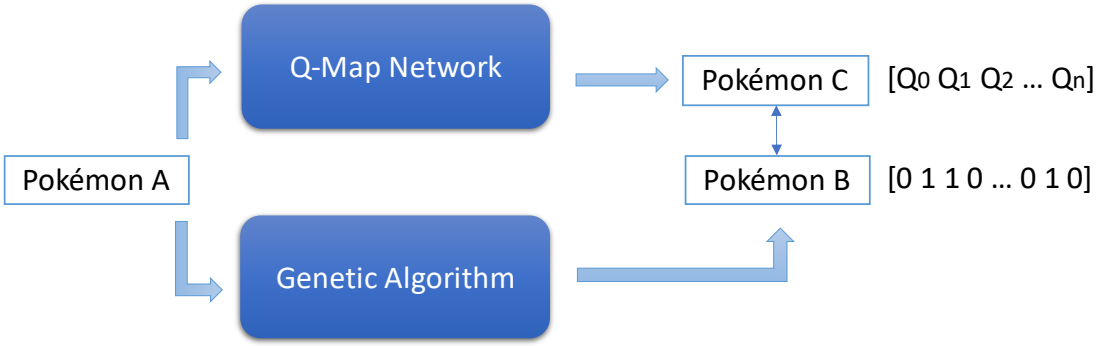


Figure 4.12: Q-MAP Network Training Process. Pokémon A is encoded as input for Q-MAP Network and outputs its gene representation fed to a Genetic Algorithm. The genetic algorithm will output a Pokémon B that beats A. We then try to minimize the error between the Pokémon B constructive elements and the Q-Values output of the network maximizing present 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 (see Figure 4.13). Our tested Q-Map network's structure is similar to our predictor network, only a Sig-

moid 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) (see Table A.1). Was trained with a learning rate $\alpha = 5^{-5}$ and mini-batches of size 128 during 15 epochs.

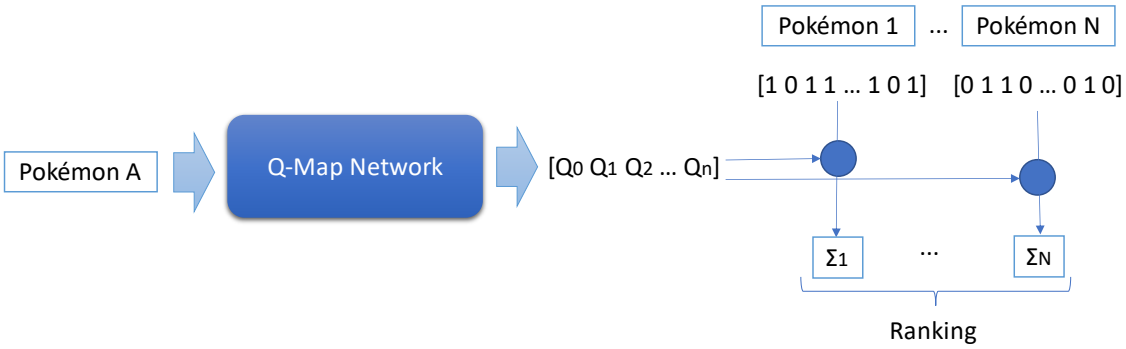


Figure 4.13: Q-MAP Lookup. We input the Pokémon we want to defeat, and the Q-MAP network will return the Q-values representing the most desirable constructive elements. Each candidate Pokémon will be crossed with the Q-Values (elementwise product) and summed, resulting in the total Q-value that can be used to rank the preferred Pokémon.

To generate a counter Pokémon against individual Pokémon, in our first experiments, the data set 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 HP Pokémon and could obtain better win rates. So for our definitive data set we used the original counters as input (the team already found by searching), and counter 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 a neural network, 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. (2016), where we could decompose the single-step games into ordered sub-decisions, where we take the best micro-action 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 Neural Networks or Long-Short Term Memory networks (Sherstinsky, 2020).

4.7.5 Evaluation and Discussion

The strength of the team build 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 4.8.3, 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 4.3. 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 team meta-game table, and we configured to counter the 10 top meta teams as hyper-parameter.

Table 4.3: Team Builder Agents vs 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

Next, the relative performance between the multiple algorithms is evaluated to evidence eventual (in)transitivity. The results are illustrated in TABLE 4.4. 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 an average team that can beat all meta teams.

Table 4.4: Team Builder Agents Relative Strength.

	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	-

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 4.5. We obtained positive results, as the random agent falls in last place in most experiments and Meta Counter in first.

Table 4.5: VGC AI Championship Track Competition Results.

	$\|R\| = 20, N = 40$	$\|R\| = 50, N = 100$
Random	1098 (6^{th})	1109 (6^{th})
IndvCtr	1214	1175
PkmCvg	1218	1188
Minmax	1158	1158
MetaCtr	**1325 (1^{st})**	**1344 (1^{st})**
TeamCvg	1183	1222

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.

4.8 Meta-Game Auto Balancing Agent

The balance agent will adapt the meta-game matrix by changing Pokémon attributes to incentive a certain natural meta by the Team Builder agents (see Figure 4.14 for an illustrated pipeline). First, we explore some previous and new meta-game balance fitness functions and target distributions of usage under different scenarios.

4.8.1 Meta-Game Balance Fitness Function

To explore good fitness functions it was first studied simplified scenarios over the Pokémon Battle Environment (Simões et al., 2020b), where Pokémon are composed only by one damaging move and therefore are only formed by 5 attributes: HP, type, speed, move's power, and move type.

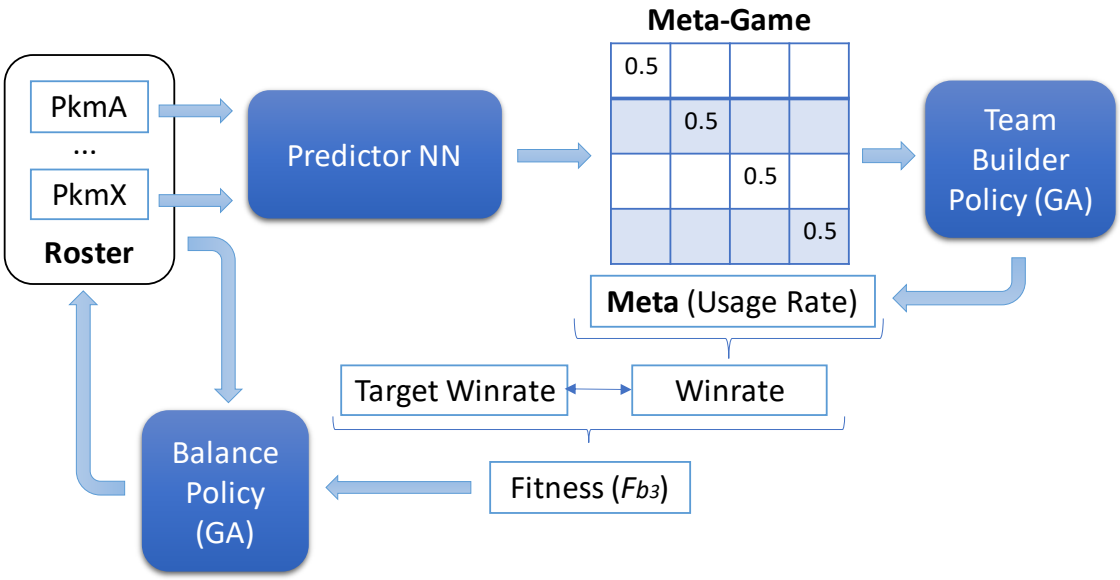


Figure 4.14: Pipeline of the adversarial model. A predictor network fills the Meta-Game matrix with the win rate for each pair of Pokémon (or Teams). The Team Builder Policy will generate a distribution of Pokémon or Team usage, the Meta. Meta-Game and Meta can be used to get the effective win rate of each Pokémon or Team and the fitness stimulates a desired win rate.

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 Pokémons. 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 meta-game balance. In addition, at each experiment, we tried to escalate the difficulty of our problem by incrementing the number of generated Pokémon.

4.8.1.1 Experiment 1: Single-Handicap Nash Policy

In this experiment, the aim was to maximize meta-game balance by forcing a 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,

$$F_{b_1}(\pi_M)\{\pi^*_M\} = -||\pi_M - \pi^*_M||. \tag{4.7}$$

4.8.1.2 Experiment 2: Single-Handicap Match wise Win Rate

In our second approach, we remove the necessity of computing NE by instead incentive **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 $\mathbf{M}$,

$$F_{b_2}(\mathbf{M})\{\mathbf{M}'\} = -||\mathbf{M} - \mathbf{M}'||. \tag{4.8}$$

This was the approach followed to balance Hearthstone by de Mesentier Silva et al. (2019), with the main difference they operate over a set of a limited number of decks, and some cards are shared between those decks.

4.8.1.3 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 this one is fixed (the desired policy) and it is used as weights to incentive $E(\mathbf{M}) = \mathbf{M} \cdot \pi_M$ 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

$$F_{b_3}(\mathbf{M}, \pi_M)\{\mathbf{w}\} = -||\mathbf{M} \cdot \pi_M - \mathbf{w}||. \tag{4.9}$$

4.8.1.4 Experiment 4: Multi-Handicap 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, it was changed the Pokémon's numerical HP, move power, and speed attributes.

4.8.1.5 Experiment 5: Balance + Pokémon Similarity Single-Objective Optimization

de Mesentier Silva et al. (2019) 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 the 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, δ_2,

$$F_{b_4}(\mathbf{M}, \pi_M)\{\mathbf{w}\} = \delta_1 F_{b_3} - \delta_2 F_s. \tag{4.10}$$

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 $\delta_2 = 0.4$.

4.8.1.6 Experiment 6: Balance + Pokémon Similarity Multi-Objective Optimization

Instead of fixing a single point solution, in this experiment, it was used the multi-objective optimization algorithm NSGA2 (Deb et al., 2002) 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 trade-off between the two fitness functions.

4.8.1.7 Fitness Function Study Results

The main results of the preliminary experiments are illustrated in TABLE 4.6.

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

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

In Exp 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 it is also needed to increase the number of generations for the GA. For $|R| = 21$ changing tournament selection to roulette wheel selection yielded better results. For the second variant, i.e., the stochastic version (see TABLE 4.7) 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 Exp 5, for $|R| = 21$, roulette wheel selection and 15% of mutation provided the best result.

For Exp 6, we executed three runs of the algorithm for $|R| = 5$, $|R| = 11$, and $|R| = 21$, with 7500 generations, tournament selection, two-points crossover, and polynomial mutation. We concluded that the algorithm is able to find trade-offs that maximize each individual metric but can also find middle terms. The main results are listed in Figure 4.15.

4.8.2 Team Balance Algorithm

For balancing teams we: (i) search for the most relevant team candidates to balance; (ii) set as an objective the maximization of one of the studied fitness functions. 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.

Table 4.6: Meta-Game 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

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 sub-meta-game we balance. Some Pokémon may have less usage rate or none at all, not becoming targets of adjustment, therefore: (1) it can be pre-select target teams containing all individual Pokémon among them, augmenting our target teams with the current unbalanced meta teams, and we soft unbalance towards turning our target diverse teams stronger; (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 10).

4.8.3 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

Table 4.7: Exp 4.5: Multi-Handicap 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

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 correctly our history H. We also test whether only two moves and the type of Pokémon can be changed.

The results are illustrated in TABLE 4.8. In both perfect balance scenarios, the win rates are between $[0.48, 0.53]$, for the predicted meta-game 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 under-powered Poémon. The large negative value of the fitness only translates the distance to the target of the under-powered, 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 $|R| = 51$. 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 4.3), 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 4.9. In a scenario of 21 Pokémon where 7 were made more powerful, we can observe that the Meta Counter agent selected some Pokémon with almost 4 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.

Firstly, 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

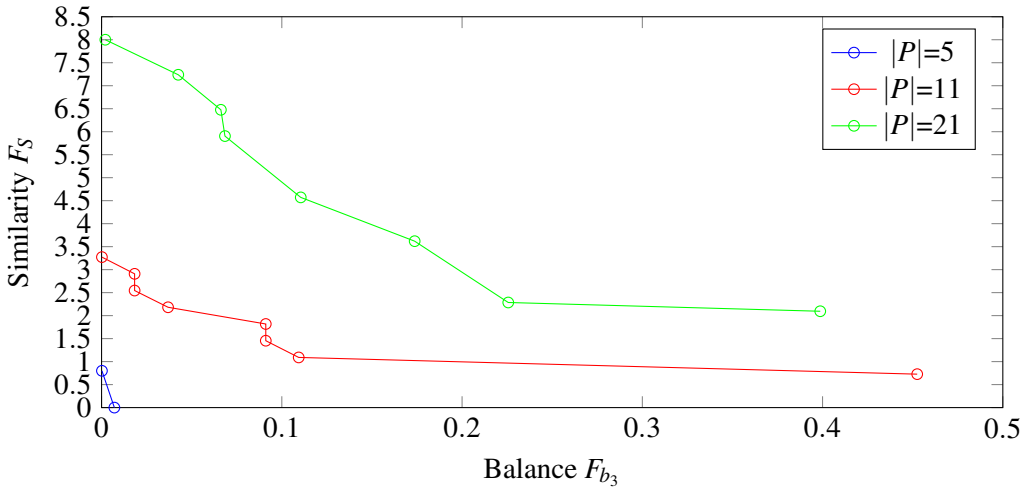


Figure 4.15: NSGA2 Multi-Objective Optimization Pareto Front of Balance and Similarity maximization multi-objective.

generated teams tend to be quite unbalanced. As the result table was very similar to TABLE 4.8, we omit it.

Next, we balance a non-enumerable quantity of teams, maximizing the diversity in terms of the number of usage rates of individual Pokémon. The results of Algorithm 10 are illustrated in TABLE 4.10. We count pre and post-balance. 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 towards a more precise search direction. The latter requires fast team build inference algorithms like Deep Q-Mapping.

4.9 Conclusions

In this chapter, an AI meta-game balance competition was formalized, the VGC AI Competition, and its implementation, the VGC AI Framework. In the VGC AI Framework, player agents compete in Pokémon-like battles, team-builder agents must anticipate the teams their opponents will use and select their own counter strategies, balancing agents must adjust a roster of available Pokémons to allow enough diversity via viable team-building strategies to give human players meaningful choices and variety of gameplay. Each layer of the competition has a corresponding track, and not merely are the multiple tracks complex by themselves, but the tasks are interdependent: to make good teams one needs to how to maximize their chances of winning battles and how to predict the outcome of matches, and in the same fashion to correctly balance a roster one needs to predict which teams will be played. Since the official Pokémon series uses a fixed roster of Pokémon we also implemented a simulator of Pokémon battles, the PBE, where randomly generated Pokémon can compete. Pokémon battling also demonstrates by itself interesting properties

Algorithm 10: Pokémon Teams 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

in terms of competitive AI research. This environment allows researchers to test various learning objectives. Further specifications of the Framework can be found in Chapter A.

Our initial takeaway to the VGC AI Competition was also presented, where meta-game roster balance is formalized and a set of hybrid tabular-deep solutions for Pokémon battling, team building, and roster balance were implemented. We trained deep battle agents with adversarial asynchronous DRL over an early version of PBE with only type advantage as a learning objective, achieving human-level performance. Then itwas trained by supervision a pair of deep models to help the search for counter-strategies against Pokémon teams' meta-game, and the tuning of a Pokémon roster. These models speed up meta-heuristic techniques by rapidly evaluating the fitness of solutions. The rationale behind this design choice of our algorithms is that while Pokémon battling is a dynamic process, team-building and roster tuning are static processes, making search algorithms that produce static results fitting and cost-efficient. Our match prediction models achieved high accuracy, but still have room for further improvement. To do so, not only other NN architectures must be explored, and we also must increase our data set volume, but to do so we need faster inference Pokémon battle algorithms to sample match results, requiring to extend our DRL battling policies to the current version of the PBE. In the same way, we would like to improve the generation time of counter teams, to search more efficiently for balanced rosters. Ideally, it would require a deep model whose input would be the entire pool of meta-teams, or alternatively

Table 4.8: Meta-Game 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

improve single team counter win rate (our current Deep Q-Map model), but the results of Maximize Team Coverage algorithm versus Meta Counter suggest that even if Q-Mapping were able to achieve optimal counter teams against individual teams and assist Maximize Team Coverage it would not still surpass Meta Counter. Nevertheless, the idea of generating teams or guiding the search direction of Meta Counter seems the most promising research avenue. This idea could be expanded even to the balancing process itself, but this would require much more high dimensional output and larger data sets, to the point that it is hard to evaluate if there is a practical benefit, and in addition, there exist multiple possible solutions to tune certain base rosters and GA are able to rapidly encounter multiple solutions.

We also mention alternative problems in the meta-game balance process that were not touched on in this chapter, but we think are worth discussing. The first is how to transpose our meta-game balance models to just game balance. The difference is that we want the in-game mechanics to be relevant in some strategies. In the VGC AI Competition, we incentivized certain Pokémon or teams to be used by team-builders. However, even if all Pokémon are correctly balanced does not imply that certain moves will be used, and each move has many defining characteristics which translate very distinctly semantically and strategically to the game, and some moves may never be used because a Pokémon only needs one or two of its moves to win, and some moves have no

Table 4.9: 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	...

direct impact in damage output, like weather conditions, so we would have to tune a particular move like SUNNY DAY, which boosts FIRE type moves and debuffs WATER type moves, but it also has secondary effects like automatically charging SOLARBEAM, and the spending of one turn to activate the SUNNY condition may not reward enough, making it a strictly dominated strategy against rational opponents. This requires not only being able to predict matches but to dynamically perform in-game predictions and understand the impact of each decision on the game result, and tune the moves without destroying the current equilibrium (without large power shifts) of other moves and Pokémons. Another interesting balance constraint not discussed in this chapter is strategy entropy maximization, i.e., instead of adapting a roster trying to maximize both balance and similarity to the balanced roster, we could not have a base roster but instead, wish to generate one from the ground up. In this case, a trivial case is to generate multiple copies of the same Pokémon, as such, we would have to put as secondary criteria that the generated Pokémon differ as much as possible from each other, creating more interesting team building decisions and gameplay variety. Finally, often games are augmented with new game mechanics, which means that monolithic deep models would no longer apply to the expanded game, but could through transfer learning maintain the knowledge for a team battling, win rate prediction, and so on, would be a useful tool to predict the impact of the introduction of those new mechanics. All these challenges could incentive new tracks or entirely new related competitions to the VGC AI Competition, with blocks reused.

The current solutions will serve as the baselines for future editions of the VGC AI Competition, hoping to incentive future participation. Participation is an integral part of the evolution model of the competition, as new battle agents are perfected so can they be used by team-building agents for the Championship track, and, in the same way, perfecting team-building agents will allow for more correct roster balance following our proposed adversarial model, where team-builders must narrow while maintaining themselves as unpredictable as possible, and the balancing agents try to maximize the entropy of Pokémon usage. This means access to agents from the previous iteration would be given to participants, and organizers could use stronger battling and team-builder agents for the meta-game balance track each edition. Our initial published proposal for the competition also evolved, evolved from its initial conceptual model. As we started developing our baseline solutions, we got the motivation to remove some of the planned behaviors that had to be implemented and to reformulate the objective of the Meta-Game Balance track, where the score is

Table 4.10: Pokémon Teams Meta-Balance Results

$\|R\|$	6	11	21	51
Balance non diverse 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

proportional to the distribution of Pokémon and team usage. The distribution can be measured in a one-shot, or it can be accumulated over multiple iterations. The score should can, as discussed, be affected by the similarity degree between each Pokémon pair and their similarity to a base roster. The objective can be re-adjusted for each edition, following the recommendations discussed here.

The studied techniques and the overall conceptual framework could be expanded to other game domains and inspire similar AI competitions. As stated in this chapter's introduction, many games require assembling armies, like in Dota 2 or Starcraft, or card decks by anticipating the current meta-game, like in Magic: The Gathering or Hearthstone, or may just be required selecting a single fighting unit, like in Street Fighter series or Tekken. A Pokémon corresponds to a game unit, as cards and fighters or champion avatars do. Deep agents like in Pokémon can help generate data sets to train predictors of the outcome of matches. Games that just require selecting a single unit do not require models for unit assembly, but card games usually have decks from 30 to 60 cards, making it more challenging than Pokémon, although the main series Pokémon allows for individual Pokémons to be configured, while cards are usually immutable objects. The balance paradigm would remain the same. In single-unit games, we can balance a finite-dimensional meta-game matrix, while in games that require army assembly or deck construction, we fall under the quasi-infinite combinatorial case of Pokémon. Given this, and since the assembly process of each game differs, and the base game also differs greatly, we believe that similar competitions may indeed be designed following similar paradigms, but requires other types of tasks to be learned and that our processes can be transposed to other game domains.

We released the source code for WPLθ and GIGAθ with the original PBE[4], VGC AI Competition Framework[5], Team Builder and Meta-Game Balancing agents (VGC Agents)[6]. We implemented our GA solutions using the library PyGAD (Gad, 2021), and the pymoo library (Blank and Deb, 2020) to implement the NSGA2 algorithm. The deep models were implemented and trained with Pytorch (Paszke et al., 2019).

[4] https://gitlab.com/DracoStriker/simplified-pokemon-environment
[5] https://gitlab.com/DracoStriker/pokemon-vgc-engine
[6] https://gitlab.com/DracoStriker/vgc-agents

Chapter 5

Conclusion

5.1 Thesis Summary

In this thesis, it was tackled automatic game balance. Balance is a property of games where they operate under certain expectations. Finding appropriate mechanisms or configurations to reach those expectations is not trivial, and games that require continuous expansions suffer from the risk that new mechanics can easily break down the previous expectations. We defend that the two main dimensions where game balance can be characterized are player difficulty and power (as in how a mechanic will be preferred over other available). Given the modern evolution of AI as an accessible toolbox, it was proposed to research how AI techniques could assist game designers and developers in tuning a game to fill under the expectations. Since difficulty balance and power balance differ in cause and aim, it was hypothesized that different techniques could be found more suitable to tackle both aims. Both issues become even more complex in multiplayer scenarios, where strategic play depends on the opponent's strategy, and in the case of difficulty balance, tailoring the difficulty for one player has the risk of disengaging the other.

We started by revisiting GT, whose concepts of game equilibrium, optimal strategies, and dominated strategies in interactive strategic games were essential for the conceptualization and analysis of power balance, and also as a requirement for the understanding of adversarial multi-agent RL. Then, it is explored the core concepts of ML, specifically on Artificial NNs, their common architectures, and the learning process. As it is required the production of intelligent agents, we then shifted focus to the study of RL fundamentals and its extension to DRL which is a combination of NNs and classic RL, allowing to tackle large and continuous action and state spaces, which was found suitable for the dynamic difficulty balancing challenges. To reach desired GT static solutions for power balance, it is also covered the basics of GAs that are a form of metaheuristic search inspired by biological evolution, and linear programming to find out NE in two-player normal-form games.

For **Difficulty Balance**, the related fields and their current state of the art are identified, including psychological models like GameFlow, DDA techniques, multiplayer variants, PCG applied to the creation of quality content, Player Modeling, Affective Computing, and experience studies

with human users. The work done for this topic was split into two parts. First, it was conceived our initial RL framework for MDDA, by defining the Game Adaptation Problem, and our Implicit Model, where the modification of a game's rules is abstracted as a game, in this case, a meta-game. A GM is trained in a generalized fashion by creating situations with a population of agents with different skill levels playing agents in each training episode. We validated the Implicit Model in two case studies, where it was performed a user study, where data was collected about the users' feelings while playing a game with intelligent handicaps. Second, it was analyzed the Implicit Model limitations, which motivated us to extend it to MDDA-RL framework, where dramatization effects were used to reinforce the GM. During the evaluation of these agents, we make a per-design study while understanding practical considerations in designing MDDA training environments and surrogate agents.

For **Power Balance**, Pokémon was chosen as the test domain and related fields were identified, and reviewed their current state of the art, including Game Design AI competitions, AI applied to Pokémon battling, automated deck building on card games and team building, and automated game and meta-game balance. We then describe our PBE a RL environment for Pokémon, where general combinations of attributes on the base units can be simulated. This environment was extended to a full framework to simulate human meta-game competitions and allow for the development and comparison of AI agents for Pokémon Battling, Pokémon Team Building, and Roster Meta-Game Balance. The rules of each track were defined and the framework architecture and API. Then the expected characteristics of a balanced meta-game were defined. For the Battle Track, adversarial asynchronous DRL was explored in an initial version of PBE where agents were able to learn type advantage and through delayed rewards when best to perform switch actions. For the Championship Track, multiple algorithms were conceived mixing GT, predicting the NE of recent meta-games, and using deep NNs to predict the outcome of matches, so it could be rapidly computed a fitness function for GAs finding the most suitable team meta-counter and determined the best through our VGC AI competition model. It was also explored a deep recommender system for team elements which we called Deep Q-Mapping, where each constructive element is an atomic construction action. Finally, we used the same tools to change elements of a roster, where multiple balance fitness functions are compared.

5.2 Answering the Research Question

Our main research question was *'How can AI Automatically Balance Multiplayer Games?'*. Not only the term game balance is loosely applied in scientific research and practitioners alike, but we also extended the question to two cases: Multiplayer Difficulty Balance and Multiplayer Power Balance. The answer to our question was different in each case.

To answer *'How can AI Automatically Balance the Difficulty in Multiplayer Games?'*, it was proposed a dynamic solution by training and embedding an intelligent agent in a game whose purpose is to react to the game state configuring handicap mechanics, i.e., deciding what are the best strategies to employ to diminish the distance between players' advantage, giving weaker or

unlucky players a possibility to recover. Our results showed that the agent maximized desired game design criteria. Our framework was tested in two very different game scenarios, a real-time action survival game, and a turn-based adversarial strategy card game, demonstrating its generality. It was also conducted a user experiment, showing that humans can react positively to both receiving positive handicaps against a strong opponent suffering from negative ones, while some hardcore players also exhibit satisfaction from the challenge of negative handicaps.

To answer *'How can AI Automatically Power Balance in Multiplayer Games?'*, it was first identified that there was no common tool for comparing solutions of meta-game balance and related tasks, so it is proposed a novel model of meta-game AI competition. In this competition, AI agents assume the role of game players who play Pokémon battles and construct teams predicting their opponents' strategies, and finally, another agent assumes the role of game designer, performing changes to the game in epochs. Our competition was accepted for the IEEE CoG 2022, however, was not run given that not enough participants registered. This motivated us to develop the first batch of baseline algorithms for both team-building and meta-game balance, demonstrating our proposed adversarial model where the player agents try to optimize their win rate, narrowing their choices to the strongest strategies, while the balance agent tries widening the possible relevant choices, achieving situations where neither there is a single dominant strategy, and where not every strategy is equally strong.

Our methodology was from an application standpoint, aiming to find AI solutions to perform game balance for both difficulty management and power management. As expected, given the different nature of the problems, distinct solutions were devised since difficulty management was explored as a dynamic process, DRL was concluded to be the most fitting tool, while for power balance, static search methods like GAs were found to be better suited, with the downside of having a high computational demand due to the number of raw game simulations required to determine the outcome of games. Deep learning was used as a pre-processing step for lightweight search by predicting game outcomes on the fly. We also found deep learning as a potential tool to provide direct search in team building or to speed up static search algorithms like GAs by narrowing their search space with potential candidates. Having answered both our sub-questions, we answer our main question as well, AI can be a fitting tool to assist human designers with game balance, by automatically deploying handicaps for difficulty balance and by exploring a meta-game space and achieving expected game theoretical policies by rational players for power balance.

5.3 Limitations and Open Work

As already discussed, the used criteria for MDDA-RL are still limited to two-player games, and it will be interesting to see how the mechanism can be extended to other settings like cooperative games or hybrid games. In addition, the tested games are micro games, so how our framework scales to more heavyweight games remains to be tested since the middle-complexity games are still not fully playable. Even with entropy regularization and other tricks, it was found to be hard

to reach strategies that resulted in a variety of games while simultaneously optimizing our criteria, even requiring in the case of HeartoftheCards to set not only our initial hand of cards randomly, but also to perform intermediary random actions, so the GM did not converge to singular games during training and inference. The action space of the GM had to be adapted multiple times. The early versions would select fixed cards, which resulted in repetitive games, so we changed again so it would pick groups instead of individual cards. Right2Live showed us the issues of having handicaps that are mostly stochastic and outside the control of the GM. Also, contemplating every type of player difficulty is not trivial and may require handicap mechanism generation, and not just coordinated predesigned ones.

Regarding power balance in meta-games, the current major limitation is the fact that we still cannot output teams on the fly without some search and our final team-balance algorithm was set to be performance reliable, trading off in accuracy. If on-the-fly team generation becomes possible, then we could accelerate the search process for roster balancing that maximized the entropy of our team-builder policy and would be interesting to investigate to which extent it could be dropped search methods and think about generating a full roster using deep models given a base roster and the desired balance characteristics, performing fast discovering of balanced meta-games. Another untested experiment is of maximizing Pokémon usage entropy, i.e., finding the most balanced roster while making each element as different from the others as possible. And of course, we hope to see in the future more game balance competition applied to other game domains.

Since this work focused on dynamic difficulty balance and static power balance, the complement scenarios in multiplayer settings are other interesting avenues of research, mainly finding interesting asymmetries in initial conditions where fairness prevails, taking the players' skill as a factor, and finding powerful strategies in-game dynamically. As we stated, we believe DDA can have a positive impact in Serious Games, and one possible method could be to perform deep player modeling and predict which are the best mechanisms to deploy for specific difficulties, as the current MDDA-RL just fit for a population of players, without any memorization of the players' capabilities from past games provided one of the players remains the same, and how can their improvements or degrading be considered.

5.4 A Final Take

I hope that automatized game balance tools can soon support game developers and teams of all sizes, that this thesis and its produced scientific publications and shared artifacts serve for future research, and that a better distinction of power and difficulty balance also enhances the field in the form of new solutions, surveys, and taxonomies, increasing the quality of future discussions.

My vision is that by alleviating the balance tasks, the game designers can focus on the creative side of game design and development, and the assistant tools can recommend parameterization on the game attributes, or manage on the fly the game flow, mitigating risks of unfair game scenarios or with inappropriate difficulty, helping in the overall quality of games, which ultimately will be played and enjoyed by human players.

References

Sherief Abdallah and Victor Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008. doi: 10.1613/jair.2628.

Ernest Adams. Fundamentals of game design. In *Choice Reviews Online*, volume 47, pages 47–4462–47–4462. 2010. ISBN 978-0-321-92967-9. doi: 10.5860/choice.47-4462.

Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), February 2019.

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 151–160. PMLR, June 2019. doi: 10.48550/arXiv.1811.11214.

David Altimira, Florian 'Floyd' Mueller, Gun Lee, Jenny Clarke, and Mark Billinghurst. Towards understanding balancing in exertion games. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ACE '14, pages 10:1–10:8, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2945-3. doi: 10.1145/2663806.2663838.

David Altimira, Florian Floyd Mueller, Jenny Clarke, Gun Lee, Mark Billinghurst, and Christoph Bartneck. Enhancing player engagement through game balancing in digitally augmented physical games. *Int. J. Hum.-Comput. Stud.*, 103(C):35–47, July 2017. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2017.02.004.

Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 7–12, 2005.

Gustavo Andrade, Geber Ramalho, Alex Gomes, and Vincent Corruble. Dynamic game balancing: An evaluation of user satisfaction. pages 3–8, January 2006. doi: 10.1609/aiide.v2i1.18739.

Dennis Ang and Alex Mitchell. Representation and frequency of player choice in player-oriented dynamic difficulty adjustment systems. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '19, pages 589–600, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6688-5. doi: 10.1145/3311350.3347165.

R. Argue. Supervised learning as a tool for metagame analysis, 2014.

Sylvester Arnab, Theodore Lim, Maira B. Carvalho, Francesco Bellotti, Sara Freitas, Sandy Louchart, Neil Suttie, Riccardo Berta, and Alessandro De Gloria. Mapping learning and game

mechanics for serious games analysis. *British Journal of Educational Technology*, 46(2):391–411, 2014. doi: 10.1111/bjet.12113.

S. Bakkes, S. Whiteson, G. Li, G. V. Vişniuc, E. Charitos, N. Heijne, and A. Swellengrebel. Challenge balancing for personalised game spaces. In *2014 IEEE Games Media Entertainment*, pages 1–8, October 2014. doi: 10.1109/GEM.2014.7047971.

A. Baldwin, D. Johnson, P. Wyeth, and P. Sweetser. A framework of Dynamic Difficulty Adjustment in competitive multiplayer video games. In *2013 IEEE International Games Innovation Conference (IGIC)*, pages 16–19, September 2013. doi: 10.1109/IGIC.2013.6659150.

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed Distributional Deterministic Policy Gradients, April 2018.

Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020. doi: 10.26599/BDMA.2020.9020004.

Nicola Baumann, Christoph Lürig, and Stefan Engeser. Flow and enjoyment beyond skill-demand balance: The role of game pacing curves and personality. *Motivation and Emotion*, 40(4): 507–519, August 2016. ISSN 1573-6644. doi: 10.1007/s11031-016-9549-7.

Philipp Beau and Sander Bakkes. Automated game balancing of asymmetric video games. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016. doi: 10.1109/CIG.2016.7860432.

Alexander Becker and Daniel Görlich. What is game balancing? - an examination of concepts. *ParadigmPlus*, 1(1):22–41, April 2020. doi: 10.55969/paradigmplus.v1n1a2.

Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *CoRR*, abs/1712.09344, 2017. doi: 10.48550/arXiv.1712.09344.

Richard Bellman. *Dynamic Programming*. Dover Publications, 1957. ISBN 978-0-486-42809-3.

Timo Bertram, Johannes Fürnkranz, and Martin Müller. Predicting human card selection in magic: The gathering with contextual preference ranking. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021. doi: 10.1109/CoG52621.2021.9619134.

Jan Betley, Anna Sztyber, and Adam Witkowski. Predicting Winrate of Hearthstone Decks Using Their Archetypes. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 193–196, September 2018.

Marlene Beyer, Aleksandr Agureikin, Alexander Anokhin, Christoph Laenger, Felix Nolte, Jonas Winterberg, Marcel Renka, Martin Rieger, Nicolas Pflanzl, Mike Preuss, and Vanessa Volz. An integrated process for game balancing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016. doi: 10.1109/CIG.2016.7860425.

Aditya Bhatt, Scott Lee, Fernando de Mesentier Silva, Connor W. Watson, Julian Togelius, and Amy K. Hoover. Exploring the hearthstone deck space. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, FDG '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 978-1-4503-6571-0. doi: 10.1145/3235765.3235791.

Julian Blank and Kalyanmoy Deb. Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2990567.

Boyan Bontchev and Olga Georgieva. Playing style recognition through an adaptive video game. *Computers in Human Behavior*, 82:136–147, 2018. ISSN 0747-5632. doi: 10.1016/j.chb.2017.12.040.

Philip Bontrager and Julian Togelius. Learning to generate levels from nothing. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021. doi: 10.1109/CoG52621.2021.9619131.

Michael Bowling. Convergence and no-regret in multiagent learning. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 1021–1026, San Francisco, CA, USA, August 2001. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-812-2.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016.

Cameron Bolitho Browne. *Automatic Generation and Evaluation of Recombination Games*. PhD thesis, Queensland University of Technology, 2008.

Cameron Bolitho Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. ISSN 1943-0698. doi: 10.1109/TCIAIG.2012.2186810.

A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. ISBN 978-1-9995795-1-7.

L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Trans. Sys. Man Cyber Part C*, 38(2):156–172, March 2008. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.913919.

O Campesato. *Artificial Intelligence, Machine Learning, and Deep Learning*. Mercury Learning & Information, 2020. ISBN 978-1-68392-467-8.

Andrew Borg Cardona, Aske Walter Hansen, Julian Togelius, and Marie Gustafsson Friberger. Open Trumps, a Data Game. In *Foundations of Digital Games, Ft. Lauderdale FL USA, Aboard: Royal Caribbean's Liberty of the Seas (2014)*. Society for the Advancement of the Science of Digital Games, 2014.

Marcus Carter, Martin Gibbs, and Mitchell Harrop. Metagames, paragames and orthogames: A new vocabulary. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 11–17, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 978-1-4503-1333-9. doi: 10.1145/2282338.2282346.

Jared E. Cechanowicz, Carl Gutwin, Scott Bateman, Regan Mandryk, and Ian Stavness. Improving player balancing in racing games. In *Proceedings of the First ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '14, pages 47–56, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 978-1-4503-3014-5. doi: 10.1145/2658537.2658701.

M. Čertický, D. Churchill, K. Kim, M. Čertický, and R. Kelly. StarCraft AI competitions, bots, and tournament manager software. *IEEE Transactions on Games*, 11(3):227–237, 2019. doi: 10.1109/TG.2018.2883499.

Zhengxing Chen, Christopher Amato, Truong-Huy D. Nguyen, Seth Cooper, Yizhou Sun, and Magy Seif El-Nasr. Q-DeckRec: A fast deck recommendation system for collectible card games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, Maastricht, Netherlands, 2018. IEEE Press. doi: 10.1109/CIG.2018.8490446.

Jimmy Choi, Tamiko Mogami, and Alice Medalia. Intrinsic Motivation Inventory: An Adapted Measure for Schizophrenia Research. *Schizophrenia Bulletin*, 36(5):966–976, September 2010. ISSN 0586-7614. doi: 10.1093/schbul/sbp030.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *International Conference for Learning Representations*, 2016. doi: 10.48550/arXiv.1511.07289.

Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic Policy Gradient, September 2020.

Dawson Crane, Zachary Holmes, Taylor Tadziu Kosiara, Michael Nickels, and Matthew Spradling. Team counter-selection games. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021. doi: 10.1109/CoG52621.2021.9619157.

Christian Arzate Cruz and Jorge Adolfo Ramirez Uresti. Player-centered game AI from a flow perspective: Towards a better understanding of past trends and future directions. *Entertainment Computing*, 20:11–24, 2017. doi: 10.48550/arXiv.1803.08375.

Mihaly Csikszentmihalyi. Toward a psychology of optimal experience. In *Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi*, pages 209–226. Springer Netherlands, Dordrecht, 2014. ISBN 978-94-017-9088-8. doi: 10.1007/978-94-017-9088-8_14.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274.

Samuel da Silva Oliveira, Guilherme Eglé P. Lima Silva, Arthur C. Gorgônio, Cephas A. S. Barreto, Anne M. P. Canuto, and Bruno M. Carvalho. Team recommendation for the pokémon GO game using optimization approaches. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 163–170, 2020. doi: 10.1109/SBGames51465.2020.00030.

Stephen Dankwa and Wenfeng Zheng. Twin-delayed DDPG: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, ICVISP 2019, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-7625-9. doi: 10.1145/3387168.3387199.

Fernando de Mesentier Silva, Rodrigo Canaan, Scott Lee, Matthew C. Fontaine, Julian Togelius, and Amy K. Hoover. Evolving the hearthstone meta. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019. doi: 10.1109/CIG.2019.8847966.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Daniel A. DeLaurentis, Jitesh H. Panchal, Ali K. Raz, Prajwal Balasubramani, Apoorv Maheshwari, Adam Dachowicz, and Kshitij Mall. Toward automated game balance: A systematic engineering design approach. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021. doi: 10.1109/CoG52621.2021.9619032.

Simon Demediuk, Marco Tamassia, William L. Raffe, Fabio Zambetta, Xiaodong Li, and Florian Mueller. Monte Carlo tree search based algorithms for dynamic difficulty adjustment. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 53–59, August 2017. doi: 10.1109/CIG.2017.8080415.

Simon Demediuk, Marco Tamassia, Xiaodong Li, and William Raffe. Challenging AI: Evaluating the effect of MCTS-Driven dynamic difficulty adjustment on player enjoyment. pages 1–7, January 2019. ISBN 978-1-4503-6603-8. doi: 10.1145/3290688.3290748.

Alena Denisova, Paul Cairns, Christian Guckelsberger, and David Zendle. Measuring perceived challenge in digital games: Development & validation of the challenge originating from recent gameplay interaction scale (CORGIS). *Int. J. Hum.-Comput. Stud.*, 137(C), May 2020. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2019.102383.

Tiffany D. Do, Seong Ioi Wang, Dylan S. Yu, Matthew G. McMillian, and Ryan P. McMahan. Using machine learning to predict game outcomes based on player-champion experience in league of legends. In *Proceedings of the 16th International Conference on the Foundations of Digital Games*, FDG '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 978-1-4503-8422-3. doi: 10.1145/3472538.3472579.

Alexander Dockhorn and Sanaz Mostaghim. Introducing the Hearthstone-AI Competition, May 2019.

Fernando Fradique Duarte, Nuno Lau, Artur Pereira, and Luis Paulo Reis. A survey of planning and learning in games. *Applied Sciences*, 10(13), 2020. ISSN 2076-3417. doi: 10.3390/app10134529.

Dagmara Dziedzic and Wojciech Włodarczyk. Approaches to measuring the difficulty of games in dynamic difficulty adjustment systems. *International Journal of Human-Computer Interaction*, 34(8):707–715, 2018. ISSN 15327590. doi: 10.1080/10447318.2018.1461764.

Sehar Shahzad Farooq and Kyung-Joong Kim. Game player modeling. In Newton Lee, editor, *Encyclopedia of Computer Graphics and Games*, pages 1–5. Springer International Publishing, Cham, 2015. ISBN 978-3-319-08234-9. doi: 10.1007/978-3-319-08234-9_14-1.

Vlad Firoiu, William F. Whitney, and Joshua B. Tenenbaum. Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning, May 2017.

Matthew C. Fontaine, Scott Lee, L. B. Soros, Fernando De Mesentier Silva, Julian Togelius, and Amy K. Hoover. Mapping hearthstone deck spaces through MAP-Elites with sliding boundaries. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 161–169, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6111-8. doi: 10.1145/3321707.3321794.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018. ISSN 1935-8237. doi: 10.1561/2200000071.

Iason Gabriel. Artificial intelligence, values, and alignment. *Minds Mach.*, 30(3):411–437, September 2020. ISSN 0924-6495. doi: 10.1007/s11023-020-09539-2.

Ahmed Fawzy Gad. PyGAD: An Intuitive Genetic Algorithm Python Library, June 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, May 2010. PMLR.

Miguel Gonzalez-Duque, Rasmus Berg Palm, David Ha, and Sebastian Risi. Finding game levels with the right difficulty in a few trials through intelligent trial-and-error. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2020-Augus:503–510, 2020. ISSN 23254289. doi: 10.1109/CoG47356.2020.9231548.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-24796-5 978-3-642-24797-2. doi: 10.1007/978-3-642-24797-2.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1352–1361, Sydney, NSW, Australia, 2017. JMLR.org.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870. PMLR, July 2018.

Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015. doi: 10.1109/IJCNN.2015.7280578.

Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and V. B. Surya Prasath. Choosing mutation and crossover ratios for genetic Algorithms—A review with a new dynamic approach. *Information-an International Interdisciplinary Journal*, 10(12), 2019. ISSN 2078-2489. doi: 10.3390/info10120390.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1): 1:1–1:22, February 2013. ISSN 1551-6857. doi: 10.1145/2422956.2422957.

Maurice Hendrix, Tyrone Bellamy-Wood, Sam McKay, Victoria Bloom, and Ian Dunwell. Implementing adaptive game difficulty balancing in serious games. *IEEE Transactions on Games*, 11 (4):320–327, 2019. doi: 10.1109/TG.2018.2791019.

Daniel Hernandez, Charles Takashi Toyin Gbadamosi, James Goodman, and James Alfred Walker. Metagame autobalancing for competitive multiplayer games. In *2020 IEEE Conference on Games (CoG)*, pages 275–282, 2020. doi: 10.1109/CoG47356.2020.9231762.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity, March 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Kaspar Höschel and Vasudevan Lakshminarayanan. Genetic algorithms for lens design: A review. *Journal of Optics*, 48(1):134–144, March 2019. ISSN 0974-6900. doi: 10.1007/s12596-018-0497-3.

Dan Huang and Scott Lee. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, 2019. doi: 10.1109/CIG.2019.8848014.

Robin Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pages 429–433, New York, NY, USA, 2005. ACM. ISBN 1-59593-110-4. doi: 10.1145/1178477.1178573.

W.A. IJsselsteijn, Y.A.W. de Kort, and K. Poels. *The Game Experience Questionnaire*. Technische Universiteit Eindhoven, Eindhoven, 2013.

Alexandru Iosup. POGGI: Puzzle-Based Online Games on Grid Infrastructures. In Henk Sips, Dick Epema, and Hai-Xiang Lin, editors, *Euro-Par 2009 Parallel Processing*, Lecture Notes in Computer Science, pages 390–403, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-03869-3. doi: 10.1007/978-3-642-03869-3_39.

Susan A. Jackson and Herbert W. Marsh. Development and Validation of a Scale to Measure Optimal Experience: The Flow State Scale. *Journal of Sport and Exercise Psychology*, 18(1): 17–35, March 1996. ISSN 1543-2904, 0895-2779. doi: 10.1123/jsep.18.1.17.

Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 8(1):26–31, June 2021. doi: 10.1609/aiide.v8i1.12513.

Ian Jolliffe. Principal component analysis. In Miodrag Lovric, editor, *International Encyclopedia of Statistical Science*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_455.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A General Platform for Intelligent Agents, May 2020.

Daniel Karavolos, Antonios Liapis, and Georgios Yannakakis. Learning the patterns of balance in a multi-player shooter game. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, pages 70:1–70:10, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5319-9. doi: 10.1145/3102071.3110568.

Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, October 2020. doi: 10.1007/s11042-020-10139-6.

William Kavanagh and Alice Miller. Chained strategy generation: A technique for balancing multiplayer games using model checking, 2019.

William Kavanagh and Alice Miller. Gameplay analysis of multiplayer games with verified action-costs. *The Computer Games Journal*, 10(1):89–110, June 2021. ISSN 2052-773X. doi: 10.1007/s40869-020-00121-5.

William Kavanagh, Alice Miller, Gethin Norman, and Oana Andrei. Balancing turn-based games with chained strategy generation. *IEEE Transactions on Games*, 13(2):113–122, 2021. doi: 10.1109/TG.2019.2943227.

Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 314–315. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_192.

Ahmed Khalifa, Diego Perez-Liebana, Simon M. Lucas, and Julian Togelius. General video game level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 253–259, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908920.

Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. General video game rule generation. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, August 2017. doi: 10.1109/cig.2017.8080431.

Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. PCGRL: Procedural content generation via reinforcement learning. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'20, pages 95–101. AAAI Press, October 2020. ISBN 978-1-57735-849-7.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

D. A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall P T R, New Jersey, (1984), 1984.

Daphne Koller and Nir Friedman. Probabilistic graphical models - principles and techniques, 2009.

Jakub Kowalski and Radosław Miernik. Strategy card game AI competition COG 2019. Web: https://jakubkowalski.tech/Projects/LOCM/COG19/, 2019.

Jakub Kowalski and Radostaw Miernik. Evolutionary approach to collectible arena deckbuilding using active card game genes. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, Glasgow, United Kingdom, 2020. IEEE Press. doi: 10.1109/CEC48606.2020.9185755.

David Kristan, Pedro Bessa, Ricardo Costa, and Carlos Vaz de Carvalho. Creating competitive opponents for serious games through dynamic difficulty adjustment. *Information-an International Interdisciplinary Journal*, 11(3), 2020. ISSN 2078-2489. doi: 10.3390/info11030156.

P. L. Lanzi, D. Loiacono, and R. Stucchi. Evolving maps for match balancing in first person shooters. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, August 2014. doi: 10.1109/CIG.2014.6932901.

Raúl Lara-Cabrera and David Camacho. A taxonomy and state of the art revision on affective games. *Future Generation Computer Systems*, 2018. ISSN 0167-739X. doi: 10.1016/j.future.2017.12.056.

S. Lee and J. Togelius. Showdown AI competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 191–198, August 2017. doi: 10.1109/CIG.2017.8080435.

Ryan Leigh, Justin Schonfeld, and Sushil J. Louis. Using coevolution to understand and validate game balance in continuous games. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 1563–1570, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 978-1-60558-130-9. doi: 10.1145/1389095.1389394.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, December 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2021.3084827.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, pages 157–163, San Francisco, CA, USA, July 1994. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-335-6.

Albert Julius Liu and Steve Marschner. Balancing zero-sum games with one variable per strategy. In *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'17, Little Cottonwood Canyon, Utah, USA, 2017. AAAI Press. ISBN 978-1-57735-791-9.

F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas. Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 320–323, 2013. doi: 10.1109/GCCE.2013.6664844.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. Evolving card sets towards balancing dominion. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012. doi: 10.1109/CEC.2012.6256441.

Olana Missura. *Dynamic Difficulty Adjustment*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2015.

Olana Missura and Thomas Gärtner. Online adaptive agent for connect four. In *Proceedings of the 4th International Conference on Games Research and Development CyberGames*, pages 1–8, 2008.

Olana Missura and Thomas Gärtner. Player Modeling for Intelligent Difficulty Adjustment. In *Proceedings of the 12th International Conference on Discovery Science*, DS '09, pages 197–211, Berlin, Heidelberg, October 2009. Springer-Verlag. ISBN 978-3-642-04746-6. doi: 10.1007/978-3-642-04747-3_17.

Olana Missura and Thomas Gärtner. Predicting dynamic difficulty. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

Martin Mladenov and Olana Missura. Offline learning for online difficulty prediction. In *Workshop on Machine Learning and Games at ICML*, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, June 2016. PMLR.

Giulio Mori, David Thue, and Stephan Schiffel. A structured analysis of experience management techniques. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15(1):174–180, October 2019. doi: 10.1609/aiide.v15i1.5241.

Mihail Moroşan and Riccardo Poli. Automated game balancing in ms PacMan and StarCraft using evolutionary algorithms. pages 377–392, March 2017. ISBN 978-3-319-55848-6. doi: 10.1007/978-3-319-55849-3_25.

Mihail Moroşan and Riccardo Poli. Lessons from testing an evolutionary automated game balancer in industry. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pages 263–270, 2018. doi: 10.1109/GEM.2018.8516447.

Paraschos Moschovitis and Alena Denisova. Keep Calm and Aim for the Head: Biofeedback-Controlled Dynamic Difficulty Adjustment in a Horror Game. *IEEE Transactions on Games*, pages 1–1, 2022. ISSN 2475-1510. doi: 10.1109/TG.2022.3179842.

Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: An overview. In Yaxin Bi, Supriya Kapoor, and Rahul Bhatia, editors, *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 426–440, Cham, 2018. Springer International Publishing. ISBN 978-3-319-56991-8.

Christian Muise, Frank Dignum, Paolo Felli, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Towards team formation via automated planning. In *Lecture Notes in Computer Science*, pages 282–299. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42691-4_16.

John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, January 1950. doi: 10.1073/pnas.36.1.48.

Ashey Noblega., Aline Paes., and Esteban Clua. Towards adaptive deep reinforcement game balancing. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, pages 693–700. SciTePress / INSTICC, 2019. ISBN 978-989-758-350-6. doi: 10.5220/0007395406930700.

Jeannie Novak. *Game Development Essentials: An Introduction*. Delmar Cengage Learning, third edition, 2012. ISBN 978-1-111-30765-3.

S. Oliveira and L. Magalhães. Adaptive content generation for games. In *Encontro Português de Computação Gráfica e Interação (EPCGI)*, pages 1–8, October 2017. doi: 10.1109/EPCGI.2017.8124303.

Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, USA, 1994. ISBN 0-262-65040-1.

Randy Pagulayan, Kevin Keeker, Thomas Fuller, Dennis Wixon, Ramon Romero, and Daniel Gunn. User-centered design in games. In *Human Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*. CRC Press, third edition, May 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2778–2787, Sydney, NSW, Australia, 2017. JMLR.org.

Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon Lucas. General Video Game AI: Competition, Challenges and Opportunities. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), March 2016. ISSN 2374-3468. doi: 10.1609/aaai.v30i1.9869.

Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms. *IEEE Transactions on Games*, 11(3):195–214, September 2019. ISSN 2475-1510. doi: 10.1109/TG.2019.2901021.

Manuel Pezzera and N. Alberto Borghese. Dynamic difficulty adjustment in exer-games for rehabilitation: A mixed approach. *2020 IEEE 8th International Conference on Serious Games and Applications for Health, SeGAH 2020*, 2020. doi: 10.1109/SeGAH49190.2020.9201871.

Johannes Pfau, Antonios Liapis, Georg Volkmar, Georgios N. Yannakakis, and Rainer Malaka. Dungeons & replicants: Automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)*, pages 431–438, 2020. doi: 10.1109/CoG47356.2020.9231958.

Eric Piette, Dennis J.N.J. Soemers, Matthew Stephenson, Chiara F. Sironi, Mark H.M. Winands, and Cameron Browne. Ludii - the ludemic general game system. In Giuseppe De Giacomo, Alejandro Catala, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 : 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 411–418, Netherlands, 2020. IOS Press. doi: 10.3233/FAIA200120.

Hafiz Adhiyasa Pratama and Adila Alfa Krisnadhi. Representing dynamic difficulty in turn-based role playing games using Monte Carlo tree search. In *2018 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2018*, 2018 International Conference on Advanced Computer Science and Information Systems, ICACSIS 2018, pages 207–212, United States, January 2019. Institute of Electrical and Electronics Engineers Inc. doi: 10.1109/ICACSIS.2018.8618167.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions, October 2017.

Simão Reis, Luís Paulo Reis, and Nuno Lau. Player engagement enhancement with video games. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *New Knowledge in Information Systems and Technologies*, pages 263–272, Cham, 2019a. Springer International Publishing. ISBN 978-3-030-16184-2. doi: 10.1007/978-3-030-16184-2_26.

Simão Reis, Luís Paulo Reis, and Nuno Lau. Automatic generation of a sub-optimal agent population with learning. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *New Knowledge in Information Systems and Technologies*, pages 65–74, Cham, 2019b. Springer International Publishing. ISBN 978-3-030-16184-2. doi: 10.1007/978-3-030-16184-2_7.

Simão Reis, Luís Paulo Reis, and Nuno Lau. Game adaptation by using reinforcement learning over meta games. *Group Decision and Negotiation*, January 2020. ISSN 1572-9907. doi: 10.1007/s10726-020-09652-8.

Simão Reis, Luís Paulo Reis, and Nuno Lau. VGC AI competition - A new model of meta-game balance AI competition. In *2021 IEEE Conference on Games (CoG)*, pages 01–08, 2021. doi: 10.1109/CoG52621.2021.9618985.

Simão Reis, Rita Novais, Luís Paulo Reis, and Nuno Lau. An Adversarial Approach for Automated Pokémon Team Building and Meta-Game Balance. *IEEE Transactions on Games*, pages 1–11, 2023. ISSN 2475-1510. doi: 10.1109/TG.2023.3273157.

Katja Rogers, Mark Colley, David Lehr, Julian Frommel, Marcel Walch, Lennart E. Nacke, and Michael Weber. KickAR: Exploring Game Balancing Through Boosts and Handicaps in Augmented Reality Table Football. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–12, New York, NY, USA, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173740.

Pejman Sajjadi, Ahmed Ewais, and Olga De Troyer. Individualization in serious games: A systematic review of the literature on the aspects of the players to adapt to. *Entertainment Computing*, 41:100468, March 2022. ISSN 1875-9521. doi: 10.1016/j.entcom.2021.100468.

Tom Schaul. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8, 2013. doi: 10.1109/CIG.2013.6633610.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1889–1897. PMLR, June 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, October 2018.

Gabriel K. Sepulveda, Felipe Besoain, and Nicolas A. Barriga. Exploring Dynamic Difficulty Adjustment in Videogames. In *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–6, November 2019. doi: 10.1109/CHILECON47746.2019.8988068.

Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer International Publishing, Cham, 2016. ISBN 978-3-319-42714-0 978-3-319-42716-4. doi: 10.1007/978-3-319-42716-4.

Clinton Sheppard. *Genetic Algorithms with Python*. 2016. ISBN 978-1-5403-2400-9 1-5403-2400-1.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, March 2020. ISSN 0167-2789. doi: 10.1016/j.physd.2019.132306.

Peizhi Shi and Ke Chen. Learning Constructive Primitives for Real-Time Dynamic Difficulty Adjustment in Super Mario Bros. *IEEE Transactions on Games*, 10(2):155–169, June 2018. ISSN 2475-1510. doi: 10.1109/TCIAIG.2017.2740210.

Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0-521-89943-5.

Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, May 2007. ISSN 0004-3702. doi: 10.1016/j.artint.2006.02.006.

Mirna Paula Silva, Victor do Nascimento Silva, and Luiz Chaimowicz. Dynamic difficulty adjustment on MOBA games. *Entertainment Computing*, 18:103–123, 2017. ISSN 18759521. doi: 10.1016/j.entcom.2016.10.002.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, June 2014. PMLR.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017. ISSN 1476-4687. doi: 10.1038/nature24270.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science (New York, N.Y.)*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

David Simões, Nuno Lau, and Luís Paulo Reis. Adjusted bounded weighted policy learner. In *RoboCup 2018: Robot World Cup XXII*, pages 324–336, Berlin, Heidelberg, 2018a. Springer-Verlag. ISBN 978-3-030-27543-3. doi: 10.1007/978-3-030-27544-0_27.

David Simões, Nuno Lau, and Luís Paulo Reis. Mixed-Policy Asynchronous Deep Q-Learning. In Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, and Carlos Cardeira, editors, *ROBOT 2017: Third Iberian Robotics Conference*, Advances in Intelligent Systems and Computing, pages 129–140, Cham, 2018b. Springer International Publishing. ISBN 978-3-319-70836-2. doi: 10.1007/978-3-319-70836-2_11.

David Simões, Nuno Lau, and Luís Paulo Reis. Guided Deep Reinforcement Learning in the GeoFriends2 Environment. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2018c. doi: 10.1109/IJCNN.2018.8489372.

David Simões, Nuno Lau, and Luís Paulo Reis. Multi-Agent Deep Reinforcement Learning with Emergent Communication. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2019a. doi: 10.1109/IJCNN.2019.8852293.

David Simões, Nuno Lau, and Luís Paulo Reis. Multi-agent Neural Reinforcement-Learning System with Communication. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors, *New Knowledge in Information Systems and Technologies*, Advances in Intelligent Systems and Computing, pages 3–12, Cham, 2019b. Springer International Publishing. ISBN 978-3-030-16184-2. doi: 10.1007/978-3-030-16184-2_1.

David Simões, Nuno Lau, and Luís Paulo Reis. Multi-agent actor centralized-critic with communication. *Neurocomputing*, 390:40–56, 2020a. ISSN 0925-2312. doi: 10.1016/j.neucom.2020.01.079.

David Simões, Simão Reis, Nuno Lau, and Luís Paulo Reis. Competitive deep reinforcement learning over a pokémon battling simulator. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 40–45, 2020b. doi: 10.1109/ICARSC49921.2020.9096092.

Sam Snodgrass, Omid Mohaddesi, Jack Hart, Guillermo Romera Rodriguez, Christoffer Holmgård, and Casper Harteveld. Like PEAS in PoDS: The player, environment, agents, system framework for the personalization of digital systems. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-7217-6. doi: 10.1145/3337722.3337756.

Mohammad Sohrabi and Hossein Azgomi. A survey on the combined use of optimization methods and game theory. *Archives of Computational Methods in Engineering*, 27, November 2018. doi: 10.1007/s11831-018-9300-5.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565, June 2022. ISSN 0920-5691. doi: 10.1007/s11263-022-01611-x.

Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006. ISSN 08856125. doi: 10.1007/s10994-006-6205-6.

Matthew Stephenson, Éric Piette, Dennis J.N.J. Soemers, and Cameron Browne. Ludii as a competition platform. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019. doi: 10.1109/CIG.2019.8848084.

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, 10(3):257–270, September 2018. ISSN 2475-1510. doi: 10.1109/TG.2018.2846639.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning - an Introduction*. Adaptive Computation and Machine Learning. MIT Press, 1998. ISBN 0-262-19398-1.

Penelope Sweetser and Peta Wyeth. GameFlow: A model for evaluating player enjoyment in games. 3(3):3–3, July 2005. ISSN 1544-3574. doi: 10.1145/1077246.1077253.

David Thue and Vadim Bulitko. Toward a unified understanding of experience management. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'18, pages 130–137, Edmonton, Alberta, Canada, November 2018. AAAI Press. ISBN 978-1-57735-804-6.

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, September 2011. ISSN 1943-0698. doi: 10.1109/TCIAIG.2011.2148116.

Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, Georgios N. Yannakakis, and Corrado Grappiolo. Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines*, 14(2):245–277, June 2013. ISSN 1573-7632. doi: 10.1007/s10710-012-9174-5.

Michel Tokic. Adaptive ε-Greedy exploration in reinforcement learning based on value differences. In Rüdiger Dillmann, Jürgen Beyerer, Uwe D. Hanebeck, and Tanja Schultz, editors, *KI 2010: Advances in Artificial Intelligence*, pages 203–210, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16111-7.

Nenad Tomašev, Ulrich Paquet, Demis Hassabis, and Vladimir Kramnik. Assessing Game Balance with AlphaZero: Exploring Alternative Rule Sets in Chess, September 2020.

Karl Tuyls and Gerhard Weiss. Multiagent Learning: Basics, Challenges, and Prospects. *AI Magazine*, 33(3):41–41, September 2012. ISSN 2371-9621. doi: 10.1609/aimag.v33i3.2426.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2094–2100, Phoenix, Arizona, 2016. AAAI Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A New Challenge for Reinforcement Learning, August 2017.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Nikos Vlassis. *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Morgan and Claypool Publishers, 1st edition, 2007. ISBN 1-59829-526-8.

Vanessa Volz, Günter Rudolph, and Boris Naujoks. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 269–276, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908913.

William Wallis, William Kavanagh, Alice Miller, and Tim Storer. Designing a mobile game to generate player data – lessons learned, January 2021.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1995–2003, New York, NY, USA, 2016. JMLR.org.

Henry N. Ward, Bobby Mills, Daniel J. Brooks, Dan Troha, and Arseny S. Khakhalin. AI solutions for drafting in Magic: The Gathering. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021. doi: 10.1109/CoG52621.2021.9619100.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

Charlotte Lærke Weitze. Learning, education and games. chapter Developing Goals and Objectives for Gameplay and Learning, pages 225–249. ETC Press, Pittsburgh, PA, USA, 2014. ISBN 978-1-312-54285-3.

Tom Wijman. The Games Market and Beyond in 2021: The Year in Numbers.

M. Wu, S. Xiong, and H. Iida. Fairness mechanism in multiplayer online battle arena games. In *2016 3rd International Conference on Systems and Informatics (ICSAI)*, pages 387–392, November 2016. doi: 10.1109/ICSAI.2016.7810986.

Wen Xia and B. Anand. Game balancing with ecosystem mechanism. In *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*, pages 317–324, March 2016. doi: 10.1109/SAPIENCE.2016.7684145.

G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, July 2011. ISSN 1949-3045. doi: 10.1109/T-AFFC.2011.6.

Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63518-7 978-3-319-63519-4. doi: 10.1007/978-3-319-63519-4.

Chang Yun, Dvijesh Shastri, Ioannis Pavlidis, and Zhigang Deng. O' game, can you feel my frustration?: Improving user's gaming experience via stresscam. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2195–2204, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. doi: 10.1145/1518701.1519036.

Chongjie Zhang and Victor Lesser. Multi-Agent Learning with Policy Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):927–934, July 2010. ISSN 2374-3468. doi: 10.1609/aaai.v24i1.7639.

Haifeng Zhang, Jun Wang, Zhiming Zhou, Weinan Zhang, Ying Wen, Yong Yu, and Wenxin Li. Learning to design games: Strategic environments in reinforcement learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 3068–3074, Stockholm, Sweden, 2018. AAAI Press. ISBN 978-0-9992411-2-7.

Jichen Zhu and Santiago Ontañón. Experience Management in Multi-player Games. In *2019 IEEE Conference on Games (CoG)*, pages 1–6, August 2019. doi: 10.1109/CIG.2019.8848030.

Mohammad Zohaib and Hideyuki Nakanishi. Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction*, 2018, January 2018. ISSN 1687-5893. doi: 10.1155/2018/5681652.

Alexander E. Zook and Mark O. Riedl. A temporal data-driven player model for dynamic difficulty adjustment. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'12, pages 93–98, Stanford, California, USA, 2012. AAAI Press.

Appendix A

Appendix - VGC AI Framework

A.1 Standard Pokémon Moves

Table A.1: Standard Pokémon Moves Table

Name	Power	Acc	PP	Type	Effect	Spd
Recover	0	1.	5	NORMAL	Recovers 80 HP	No
Double-Edge	120	1.	10	NORMAL	Recovers -40 HP	No
Extreme Speed	80	1.	10	NORMAL		Yes
Slam	80	.75	20	NORMAL	1/3 of causing PARALYZED	No
Tackle	40	1.	20	NORMAL		No
Sunny Day	0	1	5	FIRE	Sets weather SUNNY	No
Fire Blast	110	.85	5	FIRE	1/3 of causing BURNED	No
Flamethrower	90	1.	15	FIRE	1/10 of causing BURNED	No
Ember	40	1.	20	FIRE	1/10 of causing BURNED	No
Rain Dance	0	1	5	WATER	Sets weather RAINY	No
Hydro Pump	110	.8	5	WATER		No
Aqua Jet	40	1	20	WATER		Yes
Bubble Beam	65	1	20	WATER	1/10 of decreasing SPEED.	No
Thunder Wave	0	1	20	ELECTRIC	Causes PARALYZED	No
Thunder	110	.7	10	ELECTRIC		No
Thunderbolt	90	1.	15	ELECTRIC	1/10 of causing PARALYZED	No
Thunder Shock	40	1.	20	ELECTRIC	1/10 of causing PARALYZED	No
Spore	0	1	20	GRASS	Causes SLEEP	No
Giga Drain	75	1.	15	GRASS	Recovers 30 HP	No
Razor Leaf	55	.95	20	GRASS		No
Energy Ball	90	1.	10	GRASS	1/10 of decreasing DEFENSE	No
Hail	0	1	5	ICE	Sets weather HAIL.	No

Continued on next page

Table A.1 – continued from previous page

Name	Power	Acc	PP	Type	Effect	Spd
Blizzard	110	.7	10	ICE	1/10 of causing FROZEN	No
Ice Beam	90	.1	10	ICE	1/10 of causing FROZEN	No
Ice Shard	40	1	20	ICE		Yes
Bulk Up	0	5	20	FIGHTING	Increases self ATTACK	No
Mach Punch	40	1	20	FIGHTING		Yes
Close Combat	120	5	5	FIGHTING	Decreases self DEFENSE	No
Dynamic Punch	100	.5	5	FIGHTING	1/10 of causing CONFUSION	No
Poison	0	1	20	POISON	Causes POISONED	No
Gunk Shot	110	.8	5	POISON	1/3 of causing POISONED	No
Poison Jab	80	1.	5	POISON	1/3 of causing POISONED	No
Acid Spray	40	1.	20	POISON	Decreases DEFENSE	No
Spikes	0	1	20	GROUND	Increases SPIKES	No
Earthquake	100	1	10	GROUND		No
Mud Shot	55	.95	15	GROUND	Decreases SPEED	No
Earth Power	90	1.	10	GROUND	1/10 of decreasing DEFENSE	No
Roost	0	1.	5	FLYING	Recovers 80 HP	No
Chatter	65	1.	20	FLYING	Causes CONFUSION	No
Hurricane	110	.7	10	FLYING	3/10 of causing CONFUSION	No
Wing Attack	60	.7	20	FLYING		No
Calm Mind	0	1	5	PSYCHIC	Increases self DEFENSE	No
Psychic	90	1	10	PSYCHIC	1/10 of decreasing DEFENSE	No
Psycho Boost	140	.9	5	PSYCHIC	Decreases self DEFENSE	No
Psybeam	65	1	10	PSYCHIC	1/10 of causing CONFUSION	No
String Shot	0	1	5	BUG	Decreases SPEED	No
Bug Buzz	90	1	10	BUG	1/10 of decreasing DEFENSE	No
Leech Life	80	1.	10	BUG	Recovers 40 HP	No
Sandstorm	0	1	5	ROCK	Sets weather SANDSTORM	No
Power Gem	80	1	20	ROCK		No
Rock Tomb	60	.95	15	ROCK	Decreases SPEED	No
Stone Edge	100	.8	5	ROCK		No
Night Shade	0	1	10	GHOST	Deals 100 damage	No
Shadow Ball	80	1	15	GHOST	1/5 of decreasing DEFENSE	No
Shadow Sneak	40	1	20	GHOST		Yes
Dragon Rage	0	1	10	DRAGON	Deals 40 damage	No
Draco Meteor	130	.9	5	DRAGON	Decreases self ATTACK	No
Dragon Breath	60	1	20	DRAGON	Causes PARALYZED	No
Outrage	120	1	10	DRAGON	Causes self CONFUSION	No

Continued on next page

Table A.1 – continued from previous page

Name	Power	Acc	PP	Type	Effect	Spd
Nasty Plot	0	5	20	DARK	Increases self ATTACK	No
Crunch	80	1	15	DARK	1/5 of decreasing DEFENSE	No
Snarl	55	.95	15	DARK	Decreases ATTACK	No
Iron Defense	0	5	20	STEEL	Increases self DEFENSE	No
Iron Tail	100	.75	15	STEEL	3/10 of increasing self DEFENSE	No
Steel Wing	70	.9	20	STEEL	1/10 of increasing self DEFENSE	No
Bullet Punch	40	1	20	STEEL		Yes
Sweet Kiss	0	.75	5	FAIRY	Causes CONFUSION	No
Play Rough	90	.9	10	FAIRY	1/10 of decreasing ATTACK	No
Moonblast	95	1	15	FAIRY	3/10 of decreasing ATTACK	No

A.2 Pokémon Battle Special Conditions

The Pokémon Battle Special Conditions are defined in Table A.2. There are four categories of special conditions: Weather, Status, Stats, and Entry Hazard. The Weather condition is a global state, with the default value being CLEAR. When a move changes the Weather it will remain in that Weather condition for five turns, until it becomes CLEAR again. If a move changes the Weather while a condition is active, the countdown will be restarted to 5. The Status condition is attached to a specific Pokémon and shall remain until the end of the battle, with the exceptions of CONFUSED, SLEEP, and FROZEN which may be removed after some turns with a probability of 1/3, remaining a maximum of five turns. Any condition that causes damage is only applied to the active Pokémon. The status conditions countdown is only applied to the active Pokémon. The stat stage levels are only applied to the active Pokémon and are reset-ed on the switch. The Entry Hazard SPIKES damage is calculated accordingly to its stage level $H \in \{0,1,2\}$, increased by the times a move employs this condition. The hazard damage D_h is given by

$$D_h = \begin{cases} \text{MAX\_HP/8}, & H = 0 \\ \text{MAX\_HP/6}, & H = 1 \\ \text{MAX\_HP/4}, & H = 2 \end{cases} \tag{A.1}$$

A.3 Application Programming Interface

A.3.1 Battle Game State

The VGC Framework API lets agents collect the multiple features of the current game state. This API was designed, so that a view from the game state is provided to agents, without enabling the

Table A.2: Pokémon Battle Special Game Conditions.

Name	Effect
Weather	
CLEAR	*(by default)*
SUNNY	Boosts FIRE moves by 1.5 and nerfs WATER moves by 1.5
RAIN	Boosts WATER moves by 1.5 and nerfs FIRE moves by 1.5
SANDSTORM	Deals 1/8 of max HP damage to non-ROCK/GROUND/STEEL Pokémon
HAIL	Deals 1/8 of max HP damage to all non-ICE Pokémon
Status	
NONE	*(by default)*
PARALYZED	1/3 of chance the Pokémon doesn't move
POISONED	Deals 1/8 of max HP damage to Pokémon each turn
CONFUSED	1/3 of chance the Pokémon doesn't move and takes 1/8 of max HP damage
SLEEP	1/3 of chance the Pokémon doesn't move
FROZEN	1/3 of chance the Pokémon doesn't move
BURNED	Deals 1/8 of max HP damage to Pokémon each turn
Stats	
ATTACK	Each stage increases the attack damage multiplier
DEFENSE	Each stage decreases the taken damage multiplier
SPEED	If the speed stage is greater than the opponent, the player moves first
Entry Hazards	
SPIKES	Deals damage when non FLYING Pokémon switches in

same to manipulate the game state. The API is abstracted into four layers. The `GameState` is composed of two opposing teams, where the self team is the `idx = 0`, the opponent is `idx = 1`, and composed by a global weather condition.

```
class GameState:
    teams: Tuple[PkmTeam, PkmTeam]
    weather_condition: Weather
```

Listing A.1: Game State Data Structure

A Pokémon Team object stores the active Pokémon and party Pokémons. Also stores the active Pokémon exclusive battle stats:

- Stage level of every stat increase or decrease of the active Pokémon.
- Flag indicating if the active Pokémon of this team is confused.
- How many turn the active Pokémon of this team is confused?
- Flags indicating if there are entry hazard conditions on this team's side of the field.

```
class PkmTeam:
     active: Pkm
     party: List[Pkm
     stage: List[int]
```

```
    confused: bool
    n_turns_confused: int
    entry_hazard: List[int]
```

Listing A.2: Pokémon Team Data Structure

A Pokémon object stores its move set, type, hit points, status (SLEEP, PARALYZED, etc.) and the number of turns it has been asleep. Its information is hidden from the opponent until switched in.

```
class Pkm:
    type: PkmType
    max_hp: float
    hp: float
    status: PkmStatus
    n_turns_asleep: int
    moves: List[PkmMove]
    pkm_id: int
```

Listing A.3: Pokémon Data Structure

A Pokémon Move is composed of a vast array of attributes, its type, accuracy, power points, power, and many effect attributes. Its information is hidden to the opponent until used by the active Pokémon.

- Flag indicating if the move has priority.
- The probability of the move's effects.
- Flag indicating if the move recovers hit points.
- How much-fixed damage the move deals.
- What status the move inflicts?
- What stats it changes its stats.
- How many stages do the move increase or decrease?
- What weather the move sets.
- Who is the target of the move effect (self or opponent)?

```
class PkmMove:
    power: float
    acc: float
    pp: int
    type: PkmType
    priority: int
    prob: float
    target: int
```

```
    recover: float
    status: PkmStatus
    stat: PkmStat
    stage: int
    fixed_damage: float
    weather: WeatherCondition
    hazard: PkmEntryHazard
```

Listing A.4: Pokémon Move Data Structure

Only public information about the opponent can be checked. When a Pokémon is switched to be the active Pokémon or the opponent Pokémon uses a move it becomes public information until the end of the match, also accessible to the competitor Team Hypothesizer and Selection Policy.

A `GameState` pair is returned to the `info` *Gym* field, while the `state` field returns an encoding pair for deep agents. Every API parameter is encoded with our proposed standard encoding.

A.3.2 Ecosystem Meta-Data

For team building, the legal Pokémon roster can be accessed. A Pokémon roster is a set of Pokémon templates, where each template defines which specimens are legal to use. A Pokémon template is constituted by the Pokémon type, max hit points, and its move roster.

```
class PkmTemplate:
     move_roster: PkmMoveRoster
     type: PkmType
     max_hp: float
     pkm_id: int

PkmRoster = Set[PkmTemplate]
```

Listing A.5: Pokémon Template and Roster Data Structures

The meta-data compresses usage and win statistics of individual Pokémon, moves, and teams. The `evaluate` method is the fitness function that evaluates the healthiness of the Meta-Game and is used by the VGC AI Framework to evaluate the performance of the Balance agents. There are two methods influential for the meta-game representation. The `update_with_team` updates the usage and win rates. The `update_with_delta_roster` updates the similarity between Pokémon and moves to be used by the `evaluate` method, but it does not change usage or pairing rates, so agents must be careful to take that into consideration when making use of the meta-game data.

```
class MetaData:
    def update_with_team(self, team: PkmFullTeam, won: bool)
    def update_with_delta_roster(self, delta: DeltaRoster)
    def get_global_pkm_usage(self, pkm_id: PkmId) -> float
    def get_global_pkm_winrate(self, pkm_id: PkmId) -> float
    def get_global_move_usage(self, move: PkmMove) -> float
    def get_global_move_winrate(self, move: PkmMove) -> float
    def get_pair_usage(self, pair: Tuple[PkmTemplate, PkmTemplate]) -> float
```

```
    def get_team(self, t) -> Tuple[PkmFullTeam, bool]
    def get_n_teams(self) -> int
    def evaluate(self) -> float
```

Listing A.6: Meta-Data Data Structure

A.3.3 Competition Behaviours

To compete, participants must submit multiple categories of behaviors. To be compatible with the VGC AI Framework, they must abide by the Python Programming Language interfaces. All behaviors follow a generic interface. There exists a method `required_encode` that allows the agent developer to require if the agent needs our predefined encoding of the game state or structured view on the game state.

```
class Behaviour(ABC):
    def get_action(s) -> Any
    def requires_encode() -> bool
    def close()
```

Listing A.7: Behaviour Interface

A `BattlePolicy` must return an integer value corresponding to the action selected depending on the game state. If the action is less than four, it corresponds to one of the four active Pokémon moves. Greater or equal to four corresponds to a switch action.

```
class BattlePolicy(Behaviour):
    def get_action(self, s: Union[List[float], GameState]) -> int
```

Listing A.8: Battle Policy Data Structure

A `TeamBuilderPolicy` has access to the available Pokémon roster, all moves each Pokémon may learn, and stats they can manipulate. The engine will verify the integrity of all Pokémon before running a tournament.

```
class TeamSelectionPolicy(Behaviour):
    def get_action(self, s: Tuple[PkmFullTeam, PkmFullTeam]) -> Set[int]
```

Listing A.9: Team Build Policy Data Structure

A `BalancePolicy` receives the current Pokémon Roster and outputs a new Pokémon Roster.

```
class BalancePolicy(Behaviour)
    def get_action(self, s: Tuple[PkmRoster, MetaData, VGCDesignConstraints]) ->
    DeltaRoster
```

Listing A.10: Team Build Policy Data Structure

A.3.4 Framework Ecosystems

For ease of use, there are provided objects that simulate each individual layer of the framework architecture.

The Battle Ecosystem Simulates a series of battles between a population of opposing battle agents. The ecosystem is run for `n_epochs` iterations. At each iteration, each player does a match at best of `n_battles`. The pairing can be done with a `pairings_strategy` random or ELO based. The full battle teams will remain constant through the epochs.

```
class BattleEcosystem(meta_data: MetaData,
                      debug: bool,
                      render: bool,
                      n_battles: int,
                      pairings_strategy: Strategy)
    def register(self, cm: CompetitorManager)
    def run(self, n_epochs: int)
```

Listing A.11: Battle Ecosystem Data Structure

The Championship Ecosystem, between `n_league_epochs` iterations of a Battle Ecosystem, it is offered the opportunity for battle agents to change their full teams. This process is repeated `n_epochs` times.

```
class ChampionshipEcosystem(roster: PkmRoster,
                            meta_data: MetaData,
                            debug: bool,
                            render: bool,
                            n_battles: int,
                            strategy: Strategy)
    def register(self, cm: CompetitorManager)
    def run(n_epochs: int, n_league_epochs: int)
```

Listing A.12: Championship Ecosystem Data Structure

In The Game Balance Ecosystem, between `n_vgc_epochs` iterations of a Championship Ecosystem, it is offered the opportunity for the balance agent to change the roster. This process is repeated `n_epochs` times. A population of `surrogate_agent` must be provided to fill the role with battle agents.

```
class GameBalanceEcosystem(competitor: Competitor,
                           surrogate_agent: List[CompetitorManager],
                           constraints: VGCDesignConstraints,
                           base_roster: PkmRoster,
                           meta_data: MetaData,
                           debug: bool,
                           render: bool,
                           n_battles; int,
                           strategy: Strategy)
    def run(self, n_epochs: int, n_vgc_epochs: int, n_league_epochs: int)
```

Listing A.13: Game Balance Ecosystem Data Structure

A.4 Graphical User Interface

We provide a simple GUI so human players and researchers can test their battle capabilities against AI agents and other human players (see Figure A.1). We also allow rendering graphically of the output battle (see Figure A.2). Since Pokémon specimens are automatically and randomly generated, we use generic game sprites, one sprite for each Pokémon type.

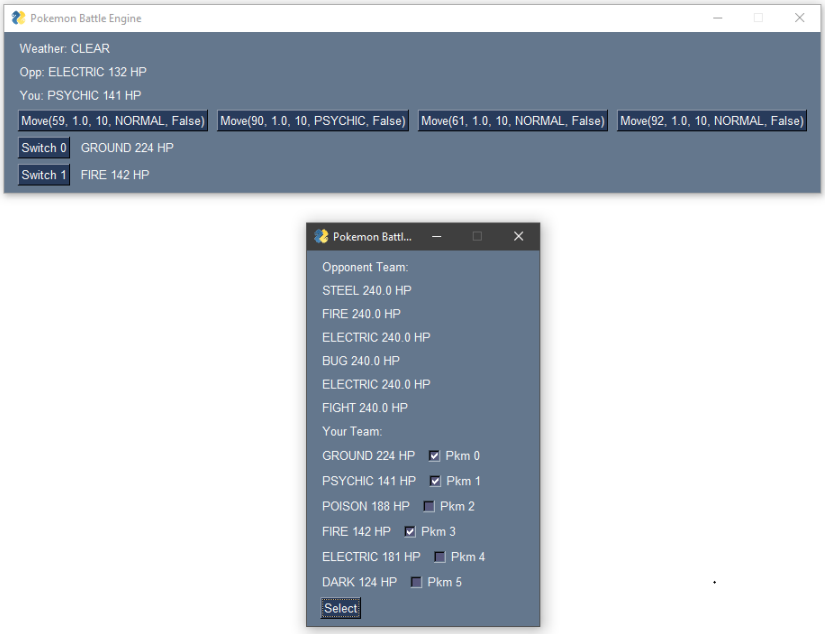


Figure A.1: GUI Battle Agent (up) and GUI Selector Agent (down).

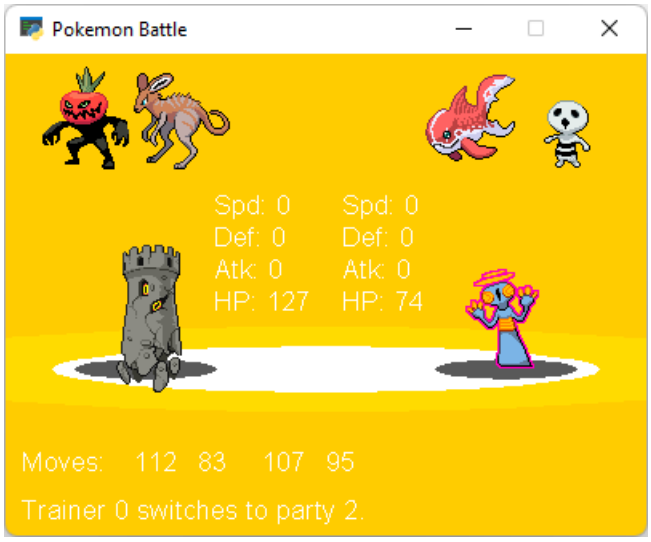


Figure A.2: Visual View of a Pokémon Battle.

A.5 Generated Teams and Balanced Rosters

In this section, we illustrate some resulting game artifacts produced from the competition in the form of the teams that were built and the tuned rosters. We also illustrated examples from our prediction network and Deep Q-Map network.

A.5.1 Q-Map Recommendations

```
# Pokemon 1
Pkm(ELECTRIC, HP=240, Moves={Thunder Shock, Bullet Punch, Mud Shot, Recover, })
['Earth Power', 'String Shot', 'Mud Shot', 'Earthquake', 'Acid Spray', 'Dragon Rage', 'Poison', 'Rain Dance', 'Rock Tomb']
['GROUND', 'POISON', 'FAIRY', 'ELECTRIC', 'ICE', 'PSYCHIC']
# Pokemon 2
Pkm(GROUND, HP=180, Moves={Spikes, Slam, Thunder, Night Shade, })
['String Shot', 'Energy Ball', 'Spikes', 'Earth Power', 'Bubble Beam', 'Tackle', 'Earthquake', 'Razor Leaf', 'Recover']
['FLYING', 'BUG', 'GRASS', 'GROUND', 'STEEL', 'NORMAL']
# Pokemon 3
Pkm(POISON, HP=210, Moves={Poison, Nasty Plot, Ember, Draco Meteor, })
['Psychic', 'String Shot', 'Earth Power', 'Acid Spray', 'Rain Dance', 'Shadow Ball', 'Crunch', 'Night Shade', 'Psycho Boost']
['STEEL', 'POISON', 'GROUND', 'NORMAL', 'GHOST', 'ROCK']
# Pokemon 4
Pkm(FIRE, HP=210, Moves={Ember, Hydro Pump, Dragon Rage, Mach Punch, })
['Earth Power', 'String Shot', 'Rock Tomb', 'Mud Shot', 'Rain Dance', 'Acid Spray', 'Snarl', 'Dragon Rage', 'Recover']
['FIRE', 'WATER', 'ROCK', 'DRAGON', 'GROUND', 'POISON']
# Pokemon 5
Pkm(GROUND, HP=210, Moves={Earth Power, Wing Attack, Outrage, Sandstorm, })
['String Shot', 'Energy Ball', 'Earth Power', 'Mud Shot', 'Thunder Wave', 'Sandstorm', 'Hail', 'Sunny Day', 'Spikes']
['FLYING', 'GRASS', 'BUG', 'STEEL', 'DRAGON', 'WATER']
# Pokemon 6
Pkm(WATER, HP=150, Moves={Aqua Jet, Night Shade, Hurricane, Ice Beam, })
['Acid Spray', 'String Shot', 'Rock Tomb', 'Sandstorm', 'Dragon Rage', 'Snarl', 'Leech Life', 'Rain Dance', 'Energy Ball']
['GRASS', 'WATER', 'DRAGON', 'GROUND', 'GHOST', 'POISON']
# Pokemon 7
Pkm(WATER, HP=210, Moves={Bubble Beam, Iron Tail, Play Rough, Razor Leaf, })
['String Shot', 'Acid Spray', 'Rock Tomb', 'Energy Ball', 'Snarl', 'Sandstorm', 'Bulk Up', 'Calm Mind', 'Leech Life']
['GRASS', 'DRAGON', 'WATER', 'BUG', 'NORMAL', 'DARK']
# Pokemon 8
Pkm(GHOST, HP=150, Moves={Mach Punch, Fire Blast, Tackle, Hydro Pump, })
['Snarl', 'String Shot', 'Shadow Ball', 'Crunch', 'Acid Spray', 'Sweet Kiss', 'Tackle', 'Rain Dance', 'Sandstorm']
['NORMAL', 'DARK', 'GROUND', 'STEEL', 'POISON', 'FAIRY']
```

```
# Pokemon 9
Pkm(POISON, HP=150, Moves={Poison, Hurricane, Crunch, Iron Tail, })
['Psychic', 'String Shot', 'Earth Power', 'Crunch', 'Snarl', 'Mud Shot', 'Acid
    Spray', 'Rock Tomb', 'Night Shade']
['STEEL', 'GROUND', 'POISON', 'PSYCHIC', 'WATER', 'NORMAL']
# Pokemon 10
Pkm(FAIRY, HP=90, Moves={Play Rough, Double-Edge, Stone Edge, Hurricane, })
['Acid Spray', 'String Shot', 'Psychic', 'Night Shade', 'Rock Tomb', 'Poison', '
    Roost', 'Dragon Rage', 'Poison Jab']
['POISON', 'STEEL', 'FIRE', 'FAIRY', 'GHOST', 'GROUND']
# Pokemon 11
Pkm(FIRE, HP=210, Moves={Flamethrower, Razor Leaf, Ice Beam, Dragon Rage, })
['Earth Power', 'String Shot', 'Rock Tomb', 'Acid Spray', 'Mud Shot', 'Rain Dance',
     'Sweet Kiss', 'Crunch', 'Snarl']
['FIRE', 'WATER', 'ROCK', 'DRAGON', 'GROUND', 'NORMAL']
# Pokemon 12
Pkm(DRAGON, HP=150, Moves={Dragon Breath, Poison Jab, Psybeam, Mach Punch, })
['Acid Spray', 'String Shot', 'Rock Tomb', 'Rain Dance', 'Sandstorm', 'Crunch', '
    Iron Defense', 'Dragon Rage', 'Play Rough']
['FAIRY', 'GROUND', 'DARK', 'ICE', 'WATER', 'BUG']
# Pokemon 13
Pkm(ROCK, HP=180, Moves={Stone Edge, Aqua Jet, Acid Spray, Double-Edge, })
['Earth Power', 'String Shot', 'Mud Shot', 'Earthquake', 'Energy Ball', 'Dragon
    Rage', 'Spore', 'Razor Leaf', 'Sandstorm']
['STEEL', 'GROUND', 'FIGHT', 'GRASS', 'POISON', 'DRAGON']
# Pokemon 14
Pkm(GROUND, HP=180, Moves={Earth Power, Moonblast, Aqua Jet, Mach Punch, })
['String Shot', 'Energy Ball', 'Razor Leaf', 'Sandstorm', 'Dragon Rage', 'Rain
    Dance', 'Spikes', 'Thunder Wave', 'Tackle']
['FLYING', 'GRASS', 'BUG', 'POISON', 'STEEL', 'GHOST']
# Pokemon 15
Pkm(GRASS, HP=240, Moves={Energy Ball, Play Rough, Thunder Wave, Tackle, })
['Acid Spray', 'String Shot', 'Wing Attack', 'Rock Tomb', 'Iron Defense', '
    Sandstorm', 'Bulk Up', 'Thunder Wave', 'Rain Dance']
['FLYING', 'BUG', 'POISON', 'GRASS', 'FIRE', 'STEEL']
```

Listing A.14: Q-MAP Examples

In Listing A.14 we print several input target Pokémon to defeat, and the Deep Q-Map network outputs. Pokémon 1 is ELECTRIC, so Earth Power, Mud Shot, and Earthquake are some of the primary recommended moves, and since GROUND is also immune to ELECTRIC is also recommended primary for the Pokémon Type. For Pokémon 3, Psychic is a strong move against POISON as is Earth Power, and STEEL is immune to POISON. Earth Power is one of the most recommended moves, showing the offensive power GROUND has, Rock Tomb seems to be very selected as well as a move that can reduce the Speed of the opponent giving a strong advantage to the user. Non-damaging moves in the competition rules allow for Pokémon to have more HP and many are recommended together with type advantage moves. Of these String Shot is one of the most recommended moves, most likely because also reduces speed like Rock Tomb, allowing for

turn priority for the rest of the battle. We can observe in general that immunity is given priority to resistance, like FAIRY being immune to DRAGON, the type of Pokémon 12, but strangely Play Rough as a strong offensive FAIRY attack was not recommended with high priority. In Pokémon 15, Wing Attack is recommended against GRASS over Hurricane, this may be due to Hurricane reducing the Pokémon total HP state or if it is the low accuracy of the move.

A.5.2 Balanced Rosters

```
# Target Win Rate:
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
# Final Roster:
Pkm(GROUND, HP=240, Moves={Extreme Speed, Calm Mind, Bubble Beam, Sunny Day, })
Pkm(NORMAL, HP=240, Moves={Moonblast, Rain Dance, Bubble Beam, Poison, })
Pkm(POISON, HP=240, Moves={Power Gem, Rain Dance, Acid Spray, String Shot, })
Pkm(PSYCHIC, HP=240, Moves={Leech Life, Rain Dance, String Shot, Nasty Plot, })
Pkm(GHOST, HP=240, Moves={Ice Beam, Sunny Day, Hail, Tackle, })
Pkm(POISON, HP=240, Moves={Snarl, String Shot, Recover, Tackle, })
Pkm(FAIRY, HP=240, Moves={Play Rough, Calm Mind, Thunder Wave, Sunny Day, })
Pkm(PSYCHIC, HP=240, Moves={Poison, Hurricane, Dynamic Punch, Chatter, })
Pkm(GRASS, HP=240, Moves={Hail, Nasty Plot, Slam, Mud Shot, })
Pkm(PSYCHIC, HP=240, Moves={Mach Punch, Fire Blast, Thunderbolt, Thunderbolt, })
Pkm(FIGHT, HP=240, Moves={Sunny Day, Giga Drain, Razor Leaf, Steel Wing, })
Pkm(FLYING, HP=240, Moves={Ice Shard, Hurricane, Thunder, Play Rough, })
Pkm(ICE, HP=240, Moves={Sandstorm, Earthquake, Mach Punch, Mach Punch, })
Pkm(FAIRY, HP=240, Moves={Calm Mind, Draco Meteor, Outrage, Extreme Speed, })
Pkm(PSYCHIC, HP=240, Moves={Roost, Play Rough, Draco Meteor, Hurricane, })
Pkm(ELECTRIC, HP=240, Moves={Ember, Hurricane, Steel Wing, Dynamic Punch, })
Pkm(FAIRY, HP=240, Moves={Sweet Kiss, Double-Edge, Hurricane, Fire Blast, })
Pkm(FLYING, HP=240, Moves={Rain Dance, Blizzard, Blizzard, Thunderbolt, })
Pkm(FIRE, HP=240, Moves={Night Shade, Poison Jab, Shadow Ball, Outrage, })
Pkm(ELECTRIC, HP=240, Moves={Thunder Wave, Extreme Speed, Extreme Speed, Psybeam,
    })
Pkm(ICE, HP=240, Moves={Thunder Wave, Bug Buzz, Psycho Boost, Blizzard, })
# Matchup Table
0.5 0.5 1.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
0.5 0.5 0.3 0.3 0.8 0.  0.1 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
0.  0.7 0.5 0.8 0.6 0.  0.7 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.  0.7 0.2 0.5 0.1 0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.  0.2 0.4 0.9 0.5 0.  0.2 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.  1.  1.  1.  1.  0.5 0.7 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
1.  0.9 0.3 1.  0.8 0.3 0.5 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.5 1.  0.  1.  0.  0.  1.  0.  1.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  1.  0.  1.  0.  0.5 0.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  1.  1.  0.5 1.  0.4 0.  1.  1.  1.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.2 0.  0.  0.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  1.  1.  0.6 1.  0.5 0.  1.  0.  1.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  1.  0.  1.  0.8 1.  0.5 1.  0.  0.  1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  1.  0.  0.  0.5 1.  0.3 1.  1.  0.  1.  1.
```

```
0.  0.  0.  0.  0.  0.  0.  1.  0.5 0.  1.  1.  1.  0.  0.5 1.  1.  1.  0.8 1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  1.  0.  1.  0.7 0.  0.5 1.  1.  0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.6 0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.4 0.5 0.  1.  1.
0.  0.  0.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  0.2 1.  1.  1.  0.5 1.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5
# Average Win Rate:
0.76190476 0.78571429 0.82380952 0.78571429 0.81904762 0.96190476 0.8952381
    0.4047619  0.33333333 0.51904762 0.22380952 0.48095238 0.44285714 0.37142857
    0.51428571 0.39047619 0.14761905 0.13809524 0.6047619  0.07142857 0.02380952
```

Listing A.15: Unbalanced Roster With 21 Pokémon for Individual Matchups

In Listing A.15 we print a solution to unbalance a roster, promoting the first seven Pokémon to be equally strong. As we can observe in the average win rate, the first seven Pokémon have an average win rate above 75%. There are two other interesting observations we can do in this example. This solution is rather unstable since Pokémon 6 is a dominant Pokémon overall with almost 100% win rate, the cause being we put as a target 100% win rate for the strongest Pokémon. This could be mitigated by reducing the value of the targets for more conservative values, like 60%-70%. The last observation is that this unbalance target alone doesn't guarantee that the seven strongest Pokémon will be equally strong and if we reduce the Matchup table to the first seven rows and columns Pokémon 6 remains a dominant strategy over the others. A secondary objective could be to set the seven strongest Pokémon to be equally strong among themselves. Note that the displayed matchup table is the one resulting from battle simulations and not the predicted values, as we only use the predicted values at run time, and evaluate with the more close ground truth win rate values.

```
# Target Win Rate:
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
# Final Roster:
Pkm(ELECTRIC, HP=240, Moves={Night Shade, Bug Buzz, Sunny Day, Mud Shot, })
Pkm(BUG, HP=240, Moves={Hurricane, String Shot, Chatter, Night Shade, })
Pkm(ICE, HP=240, Moves={Psychic, Spore, Ice Beam, Thunder, })
Pkm(DARK, HP=240, Moves={Ice Beam, Aqua Jet, Sandstorm, Gunk Shot, })
Pkm(GHOST, HP=240, Moves={Poison Jab, Steel Wing, Tackle, Steel Wing, })
Pkm(FIGHT, HP=240, Moves={Dynamic Punch, Spore, Snarl, Shadow Ball, })
Pkm(BUG, HP=240, Moves={Wing Attack, Stone Edge, Stone Edge, Poison, })
Pkm(PSYCHIC, HP=240, Moves={Outrage, Wing Attack, Psybeam, Slam, })
Pkm(ROCK, HP=240, Moves={Psycho Boost, Shadow Ball, Aqua Jet, Flamethrower, })
Pkm(WATER, HP=240, Moves={Wing Attack, Poison Jab, Dragon Breath, Spikes, })
Pkm(FLYING, HP=240, Moves={Poison Jab, Close Combat, Bubble Beam, Razor Leaf, })
Pkm(DARK, HP=240, Moves={Ember, Acid Spray, Tackle, Aqua Jet, })
Pkm(FAIRY, HP=240, Moves={Thunder Shock, Spore, Nasty Plot, Aqua Jet, })
Pkm(FAIRY, HP=240, Moves={Earth Power, Crunch, Blizzard, Moonblast, })
Pkm(STEEL, HP=240, Moves={Thunder, Mach Punch, Roost, Thunderbolt, })
Pkm(ICE, HP=240, Moves={Iron Tail, Dragon Rage, Giga Drain, Wing Attack, })
Pkm(FIRE, HP=240, Moves={Gunk Shot, Play Rough, Mach Punch, Ice Shard, })
```

```
Pkm(ICE, HP=240, Moves={Earthquake, Dynamic Punch, Mud Shot, Mach Punch, })
Pkm(GROUND, HP=240, Moves={Poison, Wing Attack, Poison, Mach Punch, })
Pkm(GROUND, HP=240, Moves={Gunk Shot, Acid Spray, Leech Life, Psychic, })
Pkm(FIRE, HP=240, Moves={Ice Beam, Thunder, Thunder Shock, Snarl, })
# Predicted Matchup Table
0.5 1.  0.5 0.1 0.8 0.8 0.9 0.5 0.3 0.9 0.1 0.8 0.9 0.1 1.  0.6 0.1 0.2 0.5 0.4 0.1
0.  0.5 0.7 0.4 0.4 1.  0.5 0.8 0.5 0.2 0.4 0.2 0.4 0.9 0.5 0.5 0.4 1.  0.7 0.3 0.3
0.5 0.3 0.5 0.2 0.7 0.5 0.9 0.2 0.6 0.6 0.7 0.  0.4 0.6 0.1 0.5 0.5 0.6 0.6 0.7 0.6
0.9 0.6 0.8 0.5 0.7 0.2 0.5 0.6 1.  0.  0.9 0.9 0.1 0.5 0.3 0.1 0.1 0.  0.9 0.8 0.1
0.2 0.6 0.3 0.3 0.5 1.  0.6 0.7 0.1 0.5 0.6 0.3 0.8 0.7 0.  0.6 0.7 0.3 0.  0.3 0.5
0.2 0.  0.5 0.8 0.  0.5 0.  0.2 0.7 0.5 0.2 0.6 0.1 0.6 0.8 0.6 0.7 0.9 0.1 0.3 0.6
0.1 0.5 0.1 0.5 0.4 1.  0.5 0.9 0.4 0.1 0.7 0.2 0.6 1.  0.2 0.6 0.4 1.  0.9 0.5 0.4
0.5 0.2 0.8 0.4 0.3 0.8 0.1 0.5 1.  0.5 0.6 0.5 0.  0.  0.2 0.8 0.8 0.7 0.7 0.8 0.5
0.7 0.5 0.4 0.  0.9 0.3 0.6 0.  0.5 0.6 1.  0.  0.2 0.3 0.3 0.6 1.  0.2 0.8 1.  0.5
0.1 0.8 0.4 1.  0.5 0.5 0.9 0.5 0.4 0.5 0.3 1.  0.  0.7 0.  0.7 0.2 0.7 0.1 0.4 0.9
0.9 0.6 0.3 0.1 0.4 0.8 0.3 0.4 0.  0.7 0.5 0.8 0.9 1.  0.  0.6 0.4 1.  0.4 0.1 0.1
0.2 0.8 1.  0.1 0.7 0.4 0.8 0.5 1.  0.  0.2 0.5 0.3 0.5 1.  0.8 0.  0.9 0.5 0.3 0.
0.1 0.6 0.6 0.9 0.2 0.9 0.4 1.  0.8 1.  0.1 0.7 0.5 0.6 0.8 0.1 0.2 0.4 0.  0.  0.5
0.9 0.1 0.4 0.5 0.3 0.4 0.  1.  0.7 0.3 0.  0.5 0.4 0.5 0.9 0.4 0.7 0.6 0.9 0.5 0.9
0.  0.5 0.9 0.7 1.  0.2 0.8 0.8 0.7 1.  1.  0.  0.2 0.1 0.5 0.5 1.  0.  0.  0.1 0.8
0.4 0.5 0.5 0.9 0.4 0.4 0.4 0.2 0.4 0.3 0.4 0.2 0.9 0.6 0.5 0.5 0.5 0.8 0.6 0.6 0.7
0.9 0.6 0.5 0.9 0.3 0.3 0.6 0.2 0.  0.8 0.6 1.  0.8 0.3 0.  0.5 0.5 0.3 0.2 0.1 0.9
0.8 0.  0.4 1.  0.7 0.1 0.  0.3 0.8 0.3 0.  0.1 0.6 0.4 1.  0.2 0.7 0.5 0.9 0.3 1.
0.5 0.3 0.4 0.1 1.  0.9 0.1 0.3 0.2 0.9 0.6 0.5 1.  0.1 1.  0.4 0.8 0.1 0.5 0.9 0.1
0.6 0.7 0.3 0.2 0.7 0.7 0.5 0.2 0.  0.6 0.9 0.7 1.  0.5 0.9 0.4 0.9 0.7 0.1 0.5 0.
0.9 0.7 0.4 0.9 0.5 0.4 0.6 0.5 0.5 0.1 0.9 1.  0.5 0.1 0.2 0.3 0.1 0.  0.9 1.  0.5
# Predicted Average Win Rate
0.51642578 0.5095803  0.48671115 0.50179125 0.46161928 0.43281462 0.51529055
    0.50279878 0.49270069 0.51107965 0.49907708 0.50456765 0.49672315 0.52626958
    0.51086213 0.50582539 0.49664847 0.48514378 0.49613808 0.51836584 0.52956681
# Matchup Table
0.5 0.7 0.2 0.3 0.1 0.5 1.  0.5 0.3 1.  0.  0.5 1.  0.  0.9 0.2 0.3 0.  0.4 0.1 0.3
0.3 0.5 0.6 0.6 0.7 0.9 1.  0.5 0.1 0.  0.5 0.2 1.  0.8 0.1 0.6 0.3 1.  0.9 0.3 0.7
0.8 0.4 0.5 0.  0.5 0.5 0.5 0.4 0.8 0.6 0.5 0.  1.  0.5 0.3 0.  0.7 0.6 0.9 0.6 1.
0.7 0.4 1.  0.5 0.4 0.1 0.6 0.8 1.  0.  1.  1.  0.9 0.7 0.1 0.  0.2 0.  1.  1.  0.
0.9 0.3 0.5 0.6 0.5 1.  0.7 0.5 0.  0.2 0.8 0.8 1.  1.  0.  0.2 0.9 0.3 0.4 0.4 0.7
0.5 0.1 0.5 0.9 0.  0.5 0.  0.2 0.2 0.1 0.  1.  0.4 0.2 0.8 0.8 0.9 0.8 0.7 0.9 0.7
0.  0.  0.5 0.4 0.3 1.  0.5 0.4 0.  0.  0.2 0.  0.9 1.  0.1 0.3 0.4 1.  1.  0.3 0.4
0.5 0.5 0.6 0.2 0.5 0.8 0.6 0.5 1.  0.5 0.3 0.4 0.  0.  0.2 0.6 0.8 0.5 1.  0.2 0.5
0.7 0.9 0.2 0.  1.  0.8 1.  0.  0.5 1.  0.7 0.  0.9 0.5 0.2 0.3 0.8 0.  1.  0.6 0.2
0.  1.  0.4 1.  0.8 0.9 1.  0.5 0.  0.5 0.6 1.  0.  0.9 0.  0.7 0.4 0.7 1.  0.6 1.
1.  0.5 0.5 0.  0.2 1.  0.8 0.7 0.3 0.4 0.5 1.  1.  1.  0.  0.4 0.7 1.  0.8 0.1 0.
0.5 0.8 1.  0.  0.2 0.  1.  0.6 1.  0.  0.  0.5 0.7 0.6 1.  0.6 0.  1.  0.3 0.4 0.
0.  0.  0.  0.1 0.  0.6 0.1 1.  0.1 1.  0.  0.3 0.5 0.5 0.5 0.  0.1 0.1 0.  0.  0.
1.  0.2 0.5 0.3 0.  0.8 0.  1.  0.5 0.1 0.  0.4 0.5 0.5 1.  0.2 0.6 0.3 0.4 0.  1.
0.1 0.9 0.7 0.9 1.  0.2 0.9 0.8 0.8 1.  1.  0.  0.5 0.  0.5 0.6 1.  0.1 0.  0.  0.7
0.8 0.4 1.  1.  0.8 0.2 0.7 0.4 0.7 0.3 0.6 0.4 1.  0.8 0.4 0.5 0.3 1.  1.  0.8 0.7
0.7 0.7 0.3 0.8 0.1 0.1 0.6 0.2 0.2 0.6 0.3 1.  0.9 0.4 0.  0.7 0.5 0.  0.6 0.  0.9
1.  0.  0.4 1.  0.7 0.2 0.  0.5 1.  0.3 0.  0.  0.9 0.7 0.9 0.  1.  0.5 1.  0.4 1.
```

```
0.6 0.1 0.1 0.  0.6 0.3 0.  0.  0.  0.  0.2 0.7 1.  0.6 1.  0.  0.4 0.  0.5 0.2 0.
0.9 0.7 0.4 0.  0.6 0.1 0.7 0.8 0.4 0.4 0.9 0.6 1.  1.  1.  0.2 1.  0.6 0.8 0.5 0.
0.7 0.3 0.  1.  0.3 0.3 0.6 0.5 0.8 0.  1.  1.  1.  0.  0.3 0.3 0.1 0.  1.  1.  0.5
# Average Win Rate:
0.41904762 0.55238095 0.52857143 0.54285714 0.55714286 0.48571429 0.41428571
    0.48571429 0.53809524 0.61904762 0.56666667 0.48571429 0.23333333 0.44285714
    0.55714286 0.65714286 0.45714286 0.54761905 0.3        0.6        0.50952381
```

Listing A.16: Balanced Roster With 21 Pokémon for Individual Matchups

In Listing A.16 we print a solution to balance a roster, promoting equally all 21 Pokémon to be as strong. We focus our analysis on the differences that can be observed between the predicted matchup table and the actual matchup table. The predicted matchup table corresponds to the table using the predicted win rates, while the actual table uses simulated battles and statistics to fill. We can see that the GA does a nearly perfect job for the predicted values, but since there is around 10% error in the predictor network, we can observe some disparity in the actual average win rate, where the values go from .23 to .66 instead of .43 to .53 in with the predicted win rates.

```
# Target Win Rate:
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
# Base Roster
Pkm(FIGHT, HP=240, Moves={Mach Punch, Aqua Jet, String Shot, Close Combat, })
Pkm(DARK, HP=210, Moves={Crunch, Earth Power, Spore, Iron Tail, })
Pkm(GRASS, HP=210, Moves={Razor Leaf, Thunder Shock, Psybeam, Play Rough, })
Pkm(NORMAL, HP=120, Moves={Slam, Roost, Thunder, Chatter, })
Pkm(GHOST, HP=240, Moves={Shadow Ball, Mach Punch, Roost, Night Shade, })
Pkm(FAIRY, HP=30, Moves={Moonblast, Thunder, Double-Edge, Dynamic Punch, })
Pkm(FIGHT, HP=180, Moves={Bulk Up, Dynamic Punch, Psybeam, Recover, })
Pkm(GHOST, HP=210, Moves={Shadow Ball, Mach Punch, Nasty Plot, Steel Wing, })
Pkm(PSYCHIC, HP=120, Moves={Psybeam, Aqua Jet, Earth Power, Hurricane, })
Pkm(POISON, HP=210, Moves={Poison, Steel Wing, Ice Beam, Snarl, })
Pkm(WATER, HP=210, Moves={Rain Dance, Dragon Breath, Psycho Boost, Calm Mind, })
Pkm(FIGHT, HP=150, Moves={Mach Punch, Mach Punch, Close Combat, Thunderbolt, })
Pkm(STEEL, HP=120, Moves={Iron Tail, Gunk Shot, Recover, Moonblast, })
Pkm(DRAGON, HP=240, Moves={Dragon Rage, Wing Attack, Flamethrower, Dragon Breath,
    })
Pkm(ELECTRIC, HP=180, Moves={Thunder, Spore, Spikes, Close Combat, })
Pkm(GHOST, HP=210, Moves={Shadow Ball, Ice Shard, Thunderbolt, Iron Defense, })
Pkm(GRASS, HP=180, Moves={Razor Leaf, Hydro Pump, Steel Wing, Energy Ball, })
Pkm(FAIRY, HP=240, Moves={Play Rough, Mach Punch, Sweet Kiss, Poison, })
Pkm(DARK, HP=120, Moves={Snarl, Draco Meteor, Extreme Speed, Chatter, })
Pkm(DARK, HP=180, Moves={Crunch, Hail, Chatter, Steel Wing, })
Pkm(GROUND, HP=180, Moves={Earth Power, Thunderbolt, Rain Dance, Stone Edge, })
# Final Roster:
Pkm(FIRE, HP=240, Moves={Close Combat, Aqua Jet, String Shot, Close Combat, })
Pkm(DARK, HP=210, Moves={Snarl, Shadow Ball, Fire Blast, Iron Tail, })
Pkm(GRASS, HP=210, Moves={Energy Ball, Thunder, Psychic, Psybeam, })
Pkm(NORMAL, HP=120, Moves={Poison Jab, Recover, Thunder, Poison, })
Pkm(GHOST, HP=240, Moves={Snarl, Ember, Roost, Hurricane, })
```

```
Pkm(WATER, HP=30, Moves={Moonblast, Thunder Shock, Slam, Earth Power, })
Pkm(GROUND, HP=180, Moves={Mud Shot, Mach Punch, Chatter, Chatter, })
Pkm(FAIRY, HP=210, Moves={Dragon Breath, Tackle, Leech Life, Steel Wing, })
Pkm(FAIRY, HP=120, Moves={Psychic, Bubble Beam, Play Rough, Blizzard, })
Pkm(POISON, HP=210, Moves={Calm Mind, Iron Defense, Hail, Snarl, })
Pkm(WATER, HP=210, Moves={Giga Drain, Thunder, Calm Mind, Iron Defense, })
Pkm(FIGHT, HP=150, Moves={Bubble Beam, Night Shade, Energy Ball, Thunder Shock, })
Pkm(BUG, HP=120, Moves={Iron Tail, Snarl, Extreme Speed, Bug Buzz, })
Pkm(DRAGON, HP=240, Moves={Leech Life, Extreme Speed, Flamethrower, Snarl, })
Pkm(ICE, HP=180, Moves={Blizzard, Razor Leaf, Giga Drain, Bulk Up, })
Pkm(FAIRY, HP=210, Moves={Acid Spray, Hail, Thunderbolt, Draco Meteor, })
Pkm(GRASS, HP=180, Moves={Mud Shot, Hydro Pump, Calm Mind, String Shot, })
Pkm(DRAGON, HP=240, Moves={Play Rough, Earth Power, Moonblast, Thunder Shock, })
Pkm(STEEL, HP=120, Moves={Mach Punch, Dragon Rage, Mach Punch, Wing Attack, })
Pkm(DARK, HP=180, Moves={Rock Tomb, Ice Shard, Razor Leaf, Bullet Punch, })
Pkm(BUG, HP=180, Moves={Iron Defense, String Shot, Rain Dance, Rock Tomb, })
# Matchup Table:
0.5 1.  1.  1.  0.  0.9 0.4 0.  0.  0.  1.  0.4 0.4 1.  1.  0.  0.1 1.  1.  0.4 0.
0.  0.5 0.5 0.2 1.  0.1 0.8 0.  1.  1.  0.5 0.1 0.5 0.  0.2 0.  1.  0.  1.  0.3 1.
0.  0.5 0.5 0.  0.7 1.  1.  0.3 0.6 0.  1.  1.  0.3 0.  0.2 0.  1.  0.1 0.  0.4 0.
0.  0.8 1.  0.5 0.1 0.1 0.4 0.6 1.  0.  0.5 0.6 0.6 0.5 0.5 0.7 1.  0.1 1.  0.4 0.9
1.  0.  0.3 0.9 0.5 0.9 0.8 0.3 0.  1.  0.2 0.  0.6 1.  0.  0.2 1.  0.6 0.8 0.2 1.
0.1 0.9 0.  0.9 0.1 0.5 0.6 0.8 0.5 0.7 0.  1.  0.9 1.  0.9 0.7 0.4 1.  0.6 1.  1.
0.6 0.2 0.  0.6 0.2 0.4 0.5 0.5 0.8 1.  0.  0.  0.  1.  0.  1.  0.  0.1 1.  1.  0.
1.  1.  0.7 0.4 0.7 0.2 0.5 0.5 0.  0.7 0.6 0.5 0.1 1.  0.3 0.  0.9 0.7 0.4 0.5 1.
1.  0.  0.4 0.  1.  0.5 0.2 1.  0.5 1.  0.1 1.  0.2 1.  0.  0.  0.4 0.4 0.  0.  0.8
1.  0.  1.  1.  0.  0.3 0.  0.3 0.  0.5 1.  0.4 0.5 1.  0.4 0.  0.  1.  1.  1.  0.4
0.  0.5 0.  0.5 0.8 1.  1.  0.4 0.9 0.  0.5 1.  0.7 0.  0.9 0.6 0.6 0.1 1.  1.  0.
0.6 0.9 0.  0.4 1.  0.  1.  0.5 0.  0.6 0.  0.5 0.6 0.  0.4 0.2 0.  0.  1.  1.  0.8
0.6 0.5 0.7 0.4 0.4 0.1 1.  0.9 0.8 0.5 0.3 0.4 0.5 0.7 0.9 0.9 1.  0.3 0.6 0.2 0.6
0.  1.  1.  0.5 0.  0.  0.  0.  0.  0.  1.  1.  0.3 0.5 0.2 0.  1.  0.  1.  1.  1.
0.  0.8 0.8 0.5 1.  0.1 1.  0.7 1.  0.6 0.1 0.6 0.1 0.8 0.5 0.2 1.  0.8 0.5 0.2 0.7
1.  1.  1.  0.3 0.8 0.3 0.  1.  1.  1.  0.4 0.8 0.1 1.  0.8 0.5 1.  0.4 0.  0.6 1.
0.9 0.  0.  0.  0.  0.6 1.  0.1 0.6 1.  0.4 1.  0.  0.  0.  0.  0.5 0.6 1.  0.3 0.
0.  1.  0.9 0.9 0.4 0.  0.9 0.3 0.6 0.  0.9 1.  0.7 1.  0.2 0.6 0.4 0.5 0.  0.9 0.7
0.  0.  1.  0.  0.2 0.4 0.  0.6 1.  0.  0.  0.  0.4 0.  0.5 1.  0.  1.  0.5 0.  0.
0.6 0.7 0.6 0.6 0.8 0.  0.  0.5 1.  0.  0.  0.  0.8 0.  0.8 0.4 0.7 0.1 1.  0.5 1.
1.  0.  1.  0.1 0.  0.  1.  0.  0.2 0.6 1.  0.2 0.4 0.  0.3 0.  1.  0.3 1.  0.  0.5
# Average Win Rate:
0.52857143 0.46190476 0.40952381 0.53809524 0.53809524 0.64761905 0.42380952
    0.55714286 0.45238095 0.51428571 0.54761905 0.45238095 0.58571429 0.45238095
    0.57142857 0.66666667 0.38095238 0.56666667 0.31428571 0.48095238 0.40952381
```

Listing A.17: Balanced Roster With 21 Pokémon for Individual Matchups conserving Similarity

In Listing A.17 we also conserve the similarity between Pokémon. As we can observe many types are conserved, and although some moves are changed they still preserve some characteristics like typing. Pokémon 1 move 1 had its move changed from Close Combat to Mach Punch, and both are FIGHTING. As we discussed, the issue with balance and similarity is that randomly

generated rosters tend to be quite balanced, so similarity can be better observed by unbalancing the roster, where we can see the individual Pokémon getting stronger while maintaining some properties. In the example, 1250 generations were run, with weights 0.9 and 0.1 for balance and similarity respectively.

```
# Target Win Rate:
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
# Base Roster
Pkm(POISON, HP=240, Moves={Poison, Mach Punch, Iron Defense, Rock Tomb, })
Pkm(ICE, HP=120, Moves={Blizzard, Nasty Plot, Gunk Shot, Mach Punch, })
Pkm(FAIRY, HP=180, Moves={Sweet Kiss, Blizzard, Dragon Breath, Play Rough, })
Pkm(ELECTRIC, HP=120, Moves={Thunder Shock, Ember, Thunder, Double-Edge, })
Pkm(ICE, HP=210, Moves={Ice Beam, Wing Attack, Steel Wing, Snarl, })
Pkm(ICE, HP=240, Moves={Ice Shard, Bubble Beam, Poison, Tackle, })
Pkm(DRAGON, HP=90, Moves={Dragon Breath, Psycho Boost, Double-Edge, Psybeam, })
Pkm(NORMAL, HP=150, Moves={Double-Edge, Sandstorm, Power Gem, Fire Blast, })
Pkm(NORMAL, HP=60, Moves={Slam, Double-Edge, Dynamic Punch, Close Combat, })
Pkm(FIRE, HP=210, Moves={Ember, Chatter, Thunder Shock, Snarl, })
Pkm(ELECTRIC, HP=150, Moves={Thunder Shock, Crunch, Dynamic Punch, Bullet Punch, })
Pkm(FIRE, HP=210, Moves={Ember, Bubble Beam, Recover, Flamethrower, })
Pkm(GHOST, HP=240, Moves={Night Shade, String Shot, Thunder Shock, Leech Life, })
Pkm(DRAGON, HP=240, Moves={Dragon Rage, Iron Defense, Mach Punch, Night Shade, })
Pkm(PSYCHIC, HP=90, Moves={Psychic, Draco Meteor, Chatter, Hydro Pump, })
Pkm(DRAGON, HP=210, Moves={Dragon Breath, Mach Punch, Snarl, Poison Jab, })
Pkm(DARK, HP=210, Moves={Crunch, Flamethrower, Bubble Beam, Nasty Plot, })
Pkm(POISON, HP=240, Moves={Poison, Power Gem, Steel Wing, Acid Spray, })
Pkm(ICE, HP=120, Moves={Ice Beam, Dragon Breath, Shadow Ball, Psycho Boost, })
Pkm(GHOST, HP=90, Moves={Mach Punch, Slam, Hurricane, Chatter, })
Pkm(FLYING, HP=180, Moves={Roost, Recover, Draco Meteor, Double-Edge, })
# Final Roster
Pkm(POISON, HP=240, Moves={Gunk Shot, Bullet Punch, Nasty Plot, Rock Tomb, })
Pkm(ICE, HP=120, Moves={Power Gem, Bulk Up, Poison, Shadow Ball, })
Pkm(FAIRY, HP=180, Moves={Play Rough, Hail, Wing Attack, Moonblast, })
Pkm(ELECTRIC, HP=120, Moves={Thunderbolt, Ember, Thunderbolt, Tackle, })
Pkm(ICE, HP=210, Moves={Crunch, Wing Attack, Steel Wing, Crunch, })
Pkm(ICE, HP=240, Moves={Acid Spray, Aqua Jet, Poison, Tackle, })
Pkm(DRAGON, HP=90, Moves={Snarl, Psycho Boost, Double-Edge, String Shot, })
Pkm(NORMAL, HP=150, Moves={Recover, Sandstorm, Play Rough, Fire Blast, })
Pkm(NORMAL, HP=60, Moves={Tackle, Slam, Dynamic Punch, Close Combat, })
Pkm(FIRE, HP=210, Moves={Ember, Psybeam, Thunder, Giga Drain, })
Pkm(ELECTRIC, HP=150, Moves={Thunder Wave, Snarl, Dynamic Punch, Bullet Punch, })
Pkm(FIRE, HP=210, Moves={Ember, Bubble Beam, Double-Edge, Ember, })
Pkm(GHOST, HP=240, Moves={Dragon Rage, Leech Life, Thunderbolt, Double-Edge, })
Pkm(ICE, HP=240, Moves={Dragon Rage, Steel Wing, Mach Punch, Night Shade, })
Pkm(PSYCHIC, HP=90, Moves={Psychic, Psycho Boost, Hurricane, Hydro Pump, })
Pkm(DRAGON, HP=210, Moves={Sunny Day, Bullet Punch, Snarl, Bubble Beam, })
Pkm(PSYCHIC, HP=210, Moves={Thunder Shock, Fire Blast, Aqua Jet, Snarl, })
Pkm(POISON, HP=240, Moves={Thunder Wave, Power Gem, Bullet Punch, Acid Spray, })
Pkm(ICE, HP=120, Moves={Psybeam, Dragon Breath, Mach Punch, Psychic, })
```

```
Pkm(GHOST, HP=90, Moves={Mach Punch, Double-Edge, Hurricane, Wing Attack, })
Pkm(FLYING, HP=180, Moves={Wing Attack, Tackle, Draco Meteor, Double-Edge, })
# Matchup Table:
0.5 0.7 1.  0.4 0.6 0.9 0.9 1.  0.9 1.  1.  1.  1.  1.  0.4 1.  1.  1.  0.4 1.  0.7
0.3 0.5 0.2 0.  1.  1.  0.4 1.  1.  1.  1.  0.9 1.  1.  1.  1.  1.  1.  1.  1.  1.
0.  0.8 0.5 0.7 1.  0.1 1.  1.  1.  1.  1.  0.8 1.  1.  1.  1.  1.  1.  0.9 0.9 0.8
0.6 1.  0.3 0.5 0.7 0.2 0.1 1.  1.  1.  1.  1.  1.  1.  0.9 1.  1.  1.  1.  1.  1.
0.4 0.  0.  0.3 0.5 0.6 0.5 1.  1.  0.6 1.  0.1 1.  1.  1.  1.  1.  1.  1.  1.  0.6
0.1 0.  0.9 0.8 0.4 0.5 1.  1.  1.  0.6 1.  0.9 1.  1.  1.  1.  1.  1.  1.  1.  1.
0.1 0.6 0.  0.9 0.5 0.  0.5 1.  1.  1.  1.  1.  1.  0.8 1.  1.  1.  1.  0.9 1.  0.4
0.  0.  0.  0.  0.  0.  0.  0.5 0.6 1.  1.  0.4 1.  0.4 0.  1.  0.8 1.  0.1 1.  0.
0.1 0.  0.  0.  0.  0.  0.  0.4 0.5 0.5 1.  0.  0.  0.  0.4 1.  0.3 1.  0.  0.4 0.
0.  0.  0.  0.  0.4 0.4 0.  0.  0.5 0.5 1.  0.  0.4 1.  0.5 1.  0.3 1.  0.4 0.  0.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.
0.  0.1 0.2 0.  0.9 0.1 0.  0.6 1.  1.  1.  0.5 1.  1.  1.  1.  1.  1.  1.  0.5 0.
0.  0.  0.  0.  0.  0.  0.  0.  1.  0.6 1.  0.  0.5 0.  0.4 1.  0.  1.  0.  0.  0.
0.  0.  0.  0.  0.  0.  0.2 0.6 1.  0.  1.  0.  1.  0.5 0.4 1.  1.  1.  0.  0.5 0.
0.6 0.  0.  0.1 0.  0.  0.  1.  0.6 0.5 1.  0.  0.6 0.6 0.5 1.  0.1 1.  0.6 0.1 0.2
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.5 0.7 1.  0.  0.  0.
0.  0.  0.  0.  0.  0.  0.  0.2 0.7 0.7 1.  0.  1.  0.  0.9 0.3 0.5 1.  1.  0.1 0.1
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.  0.
0.6 0.  0.1 0.  0.  0.  0.1 0.9 1.  0.6 1.  0.  1.  1.  0.4 1.  0.  1.  0.5 0.5 0.1
0.  0.  0.1 0.  0.  0.  0.  0.  0.6 1.  1.  0.5 1.  0.5 0.9 1.  0.9 1.  0.5 0.5 0.
0.3 0.  0.2 0.  0.4 0.  0.6 1.  1.  1.  1.  1.  1.  1.  0.8 1.  0.9 1.  0.9 1.  0.5
# Average Win Rate:
0.82857143 0.82380952 0.83333333 0.82380952 0.6952381  0.81904762 0.74761905
    0.41904762 0.26666667 0.35238095 0.07142857 0.61428571 0.26190476 0.39047619
    0.4047619  0.15238095 0.35714286 0.02380952 0.46666667 0.45238095 0.6952381
```

Listing A.18: Balanced Roster With 21 Pokémon for Individual Matchups conserving Similarity

In Listing A.18 we illustrate a case of unbalance with similarity conserved. The first seven Pokémon have a higher win rate while in general, every Pokémon maintains their type and most moves conserve their type, just changing for a weaker or stronger option. Pokémon 1 changes Poison to Gun Shot, both are POISON, Mach Punch transforms into Bullet Punch, both have priority, Iron Defense and Nasty Plot both boost a stat.

```
# Target Win Rate:
0.7 0.7 0.7 0.7 0.7 0.7 0.7 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.
# Final Roster:
Pkm(DARK, HP=240, Moves={Dragon Breath, Recover, Extreme Speed, Earth Power, })
Pkm(WATER, HP=240, Moves={Night Shade, Roost, Rock Tomb, Spore, })
Pkm(NORMAL, HP=240, Moves={Mud Shot, Sunny Day, Sweet Kiss, Double-Edge, })
Pkm(FLYING, HP=240, Moves={Ice Shard, Calm Mind, String Shot, Hail, })
Pkm(POISON, HP=240, Moves={Hydro Pump, Thunderbolt, String Shot, Chatter, })
Pkm(GHOST, HP=240, Moves={Psychic, Psybeam, Sandstorm, Blizzard, })
Pkm(FIRE, HP=240, Moves={Double-Edge, Roost, Rock Tomb, Hail, })
Pkm(DARK, HP=240, Moves={Dynamic Punch, Hurricane, Dynamic Punch, Sweet Kiss, })
Pkm(FIGHT, HP=240, Moves={Iron Tail, Moonblast, Hydro Pump, Moonblast, })
```

```
Pkm(DRAGON, HP=240, Moves={Nasty Plot, Thunder Shock, Iron Defense, Chatter, })
Pkm(DARK, HP=240, Moves={Razor Leaf, Giga Drain, Tackle, Hydro Pump, })
Pkm(GHOST, HP=240, Moves={Shadow Ball, Blizzard, Fire Blast, Draco Meteor, })
Pkm(ICE, HP=240, Moves={Ice Shard, Earth Power, Slam, Psycho Boost, })
Pkm(POISON, HP=240, Moves={Sunny Day, Giga Drain, Thunder Shock, Sunny Day, })
Pkm(FAIRY, HP=240, Moves={Poison, Hurricane, Rain Dance, Rain Dance, })
Pkm(GROUND, HP=240, Moves={Spikes, Psybeam, Shadow Ball, Thunder Shock, })
Pkm(STEEL, HP=240, Moves={Double-Edge, Sunny Day, Thunderbolt, Chatter, })
Pkm(DRAGON, HP=240, Moves={Snarl, Hydro Pump, Draco Meteor, Energy Ball, })
Pkm(FIGHT, HP=240, Moves={Fire Blast, Draco Meteor, Stone Edge, Bulk Up, })
Pkm(FLYING, HP=240, Moves={Power Gem, Fire Blast, Bullet Punch, Mach Punch, })
Pkm(FLYING, HP=240, Moves={Nasty Plot, Poison Jab, Bullet Punch, Bug Buzz, })
# Matchup Table
0.5 1.  0.8 1.  0.5 1.  0.  1.  0.9 1.  0.9 1.  1.  1.  0.  1.  0.1 1.  0.9 0.9 1.
0.  0.5 0.  1.  1.  0.2 0.  1.  1.  1.  0.  1.  1.  1.  1.  1.  0.6 1.  1.  0.5 1.
0.2 1.  0.5 0.  1.  0.9 0.3 0.9 1.  1.  1.  1.  1.  1.  0.9 1.  1.  0.3 1.  0.  0.
0.  0.  1.  0.5 0.1 0.  0.  1.  0.9 1.  1.  1.  0.  1.  0.9 1.  0.  1.  0.9 1.  1.
0.5 0.  0.  0.9 0.5 0.  0.4 1.  1.  0.6 1.  0.9 0.7 1.  1.  1.  0.9 0.  1.  0.6 0.9
0.  0.8 0.1 1.  1.  0.5 1.  0.  0.9 1.  0.  0.9 1.  1.  1.  1.  1.  0.3 1.  0.8 1.
1.  1.  0.7 1.  0.6 0.  0.5 1.  1.  1.  1.  0.  1.  1.  1.  1.  0.4 1.  1.  0.6 1.
0.  0.  0.1 0.  0.  1.  0.  0.5 0.4 0.5 0.2 0.  0.1 0.1 0.  1.  0.4 0.5 0.3 0.  0.2
0.1 0.  0.  0.1 0.  0.1 0.  0.6 0.5 1.  0.1 0.6 0.8 1.  0.6 1.  0.  0.9 0.7 0.7 1.
0.  0.  0.  0.  0.4 0.  0.  0.5 0.  0.5 1.  0.  0.  1.  0.  1.  0.  0.  0.2 0.  1.
0.1 1.  0.  0.  0.  1.  0.  0.8 0.9 0.  0.5 1.  1.  1.  0.8 1.  0.  0.  0.6 0.  0.
0.  0.  0.  0.  0.1 0.1 1.  1.  0.4 1.  0.  0.5 0.8 1.  0.  1.  1.  0.1 0.6 0.1 1.
0.  0.  0.  1.  0.3 0.  0.  0.9 0.2 1.  0.  0.2 0.5 1.  0.  1.  0.  1.  0.1 0.  1.
0.  0.  0.  0.  0.  0.  0.  0.9 0.  0.  0.  0.  0.  0.5 0.4 1.  0.  0.  0.  0.  0.
1.  0.  0.1 0.1 0.  0.  0.  1.  0.4 1.  0.2 1.  1.  0.6 0.5 1.  0.  1.  0.3 0.  1.
0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.  0.  0.  0.
0.9 0.4 0.  1.  0.1 0.  0.6 0.6 1.  1.  1.  0.  1.  1.  1.  1.  0.5 0.6 0.3 0.7 1.
0.  0.  0.7 0.  1.  0.7 0.  0.5 0.1 1.  1.  0.9 0.  1.  0.  1.  0.4 0.5 1.  0.5 1.
0.1 0.  0.  0.1 0.  0.  0.  0.7 0.3 0.8 0.4 0.4 0.9 1.  0.7 1.  0.7 0.  0.5 0.9 1.
0.1 0.5 1.  0.  0.4 0.2 0.4 1.  0.3 1.  1.  0.9 1.  1.  1.  1.  0.3 0.5 0.1 0.5 1.
0.  0.  1.  0.  0.1 0.  0.  0.8 0.  0.  1.  0.  0.  1.  0.  1.  0.  0. 0.  0.  0.5
# Average Win Rate:
0.78571429 0.7047619  0.71428571 0.63333333 0.66190476 0.72857143 0.8
    0.25238095 0.46666667 0.26666667 0.46190476 0.46190476 0.39047619 0.13333333
    0.48571429 0.02380952 0.65238095 0.53809524 0.45238095 0.62857143 0.25714286
```

Listing A.19: Unbalanced Roster With 21 Pokémon for Individual Matchups (70% win rate).

In Listing A.19 we print a solution to unbalance a roster, promoting the first seven Pokémon to be equally strong, but instead of putting their target win rate at 1 with set 0.7 (70% win rate). The first seven Pokémon have a close desired match-up win rate, but some of the Pokémon which was desired to have a low win rate have. We don't consider this a critical aspect since we still achieve a good distribution between stronger and weaker Pokémon. This result is to be expected, compared to the first scenario we weakened the seven stronger Pokémon, and as consequence, some of the weaker Pokémon have to become stronger in the meta.