Visualizing Characters and Evaluating their Balance in Competitive Video Games

by

Akash Saravanan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Akash Saravanan, 2023

Abstract

Many competitive online video games release new characters on a regular basis. Designing these characters requires significant effort on several aspects including art, story, music, and game balance. Thus automating the design of these aspects offers value in saving human effort. This thesis offers two major contributions to the open problem of video game character generation.

Our first contribution is a novel methodology for representing pixel art. We introduce the Pixel VQ-VAE, an enhanced VQ-VAE model with two enhancements to improve performance on pixel art. We demonstrate that our approach outperforms standard approaches in representational quality on two different datasets. We further apply the learned representations on two downstream tasks.

Our second contribution is a framework to evaluate changes to game balance via a process known as meta discovery. We use this meta discovery framework to simulate a large number of games and analyze the resultant game state. In developing this framework we train several reinforcement learning agents to approximate average human players and introduce a dynamic team-generation system. We demonstrate the effectiveness of the framework in approximating the results of balance changes on Pokémon Showdown, a competitive online battle simulator.

Preface

This thesis presents an original work by Akash Saravanan under the supervision of Dr. Matthew Guzdial. Parts of Chapters 2 and 3 have been published as Akash Saravanan and Matthew Guzdial, "Pixel VQ-VAEs for Improved Pixel Art Representation" in the AIIDE Workshop on Experimental AI in Games (EXAG), 2022. The remainder of this work may be restructured for publication in different research venues in the near future.

Acknowledgements

This thesis would not exist without the constant support, guidance and mentorship from my supervisor, Matthew Guzdial. He exposed me to the wonderful world of research and truly accelerated my growth as a researcher.

I would also like to thank Nathan Sturtevant and Osmar Zaïane for being on my examining committee and taking the time to understand and critique my work.

I am thankful for the GRAIL Lab and its members, both past and present, for the thought-provoking discussions, fun meet-ups, and interesting seminars we had.

The pandemic has offered many challenges in these years and I would be remiss to not thank my friends at Pappampatti Pro Racing who have helped me get through them. Special thanks to Tata Ganesh for always lending a ear to my ramblings.

Finally, I would like to express my profound gratitude to my family for their continuous love and support throughout my academic career.

Contents

Abstract ii

Preface iii

Acknowledgements iv

List of Tables ix

List of Figures xii

1 Introduction 1

1.1 Motivation: Visual Assets 1

1.2 Motivation: Game Balance 2

1.3 Thesis Statement 3

1.4 Thesis Contributions 4

1.5 Thesis Outline 4

2 Background & Related Work 6

2.1 Background 6

2.1.1 Pixel Art 6

2.1.2 Pokémon 7

2.1.3 Visual Attributes of Pokémon 7

2.1.4 Attributes for Competitive Pokémon Battles 8

2.1.5 Pokémon Battles 10

2.1.6 Metagames . . . 10
2.1.7 The Competitive Metagame of Pokémon . . . 12
2.1.8 Matchmaking Ratings . . . 13
2.2 Background: Machine Learning . . . 14
2.2.1 Artificial Neural Networks . . . 14
2.2.2 Convolutional Neural Networks . . . 15
2.2.3 Representations . . . 15
2.2.4 Autoencoders & Variational Autoencoders . . . 16
2.2.5 Image Generation . . . 17
2.2.6 Vector Quantized Variational Autoencoders . . . 17
2.2.7 Reinforcement Learning . . . 19
2.3 Pixel Art Representation . . . 19
2.3.1 Representing Pixel Art . . . 19
2.3.2 Downstream Tasks . . . 20
2.4 Video Game Character Balance . . . 20
2.4.1 Game-Playing Agents . . . 21
2.4.2 Team-Building . . . 22
2.4.3 Metagame Discovery & Game Balance . . . 22
3 Pixel Art Representation **24**
3.1 Pixel VQ-VAE . . . 25
3.2 Data . . . 27
3.2.1 Datasets . . . 27
3.2.2 Preprocessing . . . 28
3.3 Image Representation . . . 28
3.3.1 Pixel VQ-VAE Hyperparameters . . . 28
3.3.2 Baselines . . . 29
3.3.3 Embedding Quality . . . 29
3.3.4 Ablation Study . . . 32

3.4 Image Generation . . . 33
3.5 Image Transformation . . . 36
3.6 Conclusion . . . 38

4 Evaluating Game Balance Through Meta Discovery **39**
4.1 Application Domain . . . 42
4.2 Battle Agents . . . 42
4.2.1 Action Space . . . 43
4.2.2 State Representation . . . 44
4.2.3 Reward Function . . . 44
4.2.4 Baselines . . . 45
4.2.5 Reinforcement Learning Agents . . . 45
4.2.6 Model Architecture . . . 45
4.2.7 Hyperparameters . . . 46
4.2.8 Evaluation: Performance . . . 46
4.2.9 Results: Performance . . . 47
4.2.10 Evaluation: Speed . . . 49
4.2.11 Results: Speed . . . 49
4.2.12 Final Agent Selection . . . 50
4.3 Team-Building . . . 50
4.3.1 Components . . . 51
4.3.2 Algorithms . . . 52
4.4 Meta Discovery Environment . . . 54
4.5 Meta Discovery . . . 55
4.5.1 Evaluation Metrics . . . 56
4.5.2 Approaches . . . 57
4.5.3 Results . . . 57
4.6 Conclusion . . . 60

5 Conclusions and Future Work **61**

5.1 Conclusions 61

5.2 Future Work 62

5.3 Potential Impact 63

Bibliography **65**

A Pixel VAE **73**

B Impact of Sprite Rotations on the Pixel VQ-VAE **76**

C Best and Worst Case Reconstruction Analysis of the Embedding Space **77**

D Embedding Visualization of the Pixel VQ-VAE **82**

E Team-building Algorithm: Grid Search Results **87**

F Sample Outputs for the Pixel VQ-VAE Downsteam Tasks **89**

List of Tables

2.1 Examples of pixel art from the Pokémon video game franchise. 7

2.2 Examples of the different visual attributes of Pokémon. 8

2.3 Examples of the different sprites a single Pokémon can have. In the first row, there are significant differences in gameplay-mechanics between the two versions of the Pokémon. In the second and third rows, the changes are purely cosmetic. 9

3.1 Examples of sprites from both our datasets. These are images taken before the pre-processing step. Notice how the Pokémon exhibit high variations between individual Pokémon whereas the Sprites dataset is quite consistent. 27

3.2 Test set reconstructions of the Pixel VQ-VAE and the baselines. The VQ-VAE HiRes is not included as it cannot exist without the enhancements introduced in the Pixel VQ-VAE. The first two columns are from the Pokémon dataset, while the remaining are from the Sprites dataset. 30

3.3 Test set reconstruction metrics. MSE: Lower is better. SSIM: Higher is better. The VQ-VAE HiRes is not included as it cannot exist without the enhancements introduced in the Pixel VQ-VAE. 31

3.4 Ablation study comparing test set reconstruction metrics. MSE: Lower is better. SSIM: Higher is better. For HiRes models, the base VQ-VAE and the Adapter VQ-VAE cannot exist without further enhancements to the model. 32

3.5 Hand-picked generated samples for each model. All VQ-VAE results are generated from a PixelCNN model trained on the VQ-VAE embeddings. 34

3.6 FID of generated images. Lower is better. 35

3.7 Number of parameters of each model. 35

3.8 Example results for the image recolouring task. 37

4.1 Results of evaluating our agents against the three baseline agents. Columns indicate the number of battles won against a particular baseline. Higher is better for all metrics. Best agent is in **bold**, 2nd best is in *italics*. 47

4.2 Results of evaluating our agents against humans on the Pokémon Showdown random battle ladder. Higher is better for all metrics. Best agent is in **bold**, 2nd best is in *italics*. 48

4.3 Comparison of our reinforcement learning agents on time taken to complete 1000 random battles. All times are in seconds, averaged over 3 runs and rounded up to the nearest integer. All runs were performed on the same hardware and use a timeout of 3600 seconds. 49

4.4 The 4 scenarios we consider for evaluating the AST-Meta task. 56

4.5 Evaluation of the meta discovery framework using the Overlap metric on the 4 ABC-Meta scenarios. Higher is better, 1.00 is perfect. **Bold** values indicate the best methods. 57

4.6 Evaluation of the meta discovery framework using the Edit Distance metric on the 4 ABC-Meta scenarios. Values indicate the delta from the True Meta. Lower is better, 0.00 is perfect. **Bold** values indicate the best methods. 57

4.7 Results of Spearman's Rho on the Meta Discovery approaches. 58

4.8 Evaluation on the BSD-Meta task. Values are the percentage of the Smogon tiers in the discovered meta. 59

4.9 Composition of the metas discovered in the BSD-Meta task. Values are the percentage of the discovered meta that is classified in a particular Smogon tier. 59

A.1 Test set reconstruction metrics on the Pokémon dataset. MSE: Lower is better. SSIM: Higher is better. 73

A.2 Test set reconstruction comparison of the Pixel VAE on the Pokémon dataset. 75

B.1 Analysis on the impact of augmenting the Pixel VQ-VAE training data using random rotations. 76

C.1 Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Pokémon dataset according to the MSE metric. 78

C.2 Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Pokémon dataset according to the SSIM metric. 79

C.3 Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Sprites dataset according to the MSE metric. 80

C.4 Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Sprites dataset according to the SSIM metric. 81

E.1 Results of performing a grid search over the three domain knowledge components of our team-building algorithm. 88

List of Figures

3.1 Pixel VQ-VAE Architecture. 25

4.1 The components of our meta discovery framework. 41

D.1 t-SNE Plot of the latent space of the Sprites dataset in the VAE. 83
D.2 t-SNE Plot of the latent space of the Sprites dataset in the VQ-VAE MedRes. 84
D.3 t-SNE Plot of the latent space of the Sprites dataset in the Pixel VQ-VAE MedRes. . 84
D.4 t-SNE Plot of the latent space of the Pokémon dataset in the VAE. 85
D.5 t-SNE Plot of the latent space of the Pokémon dataset in the VQ-VAE MedRes. . . . 85
D.6 t-SNE Plot of the latent space of the Pokémon dataset in the Pixel VQ-VAE MedRes. 86

F.1 Sample images from our Pokémon dataset after pre-processing. 90
F.2 Sample generated Pokémon from our VAE baseline. 91
F.3 Sample generated Pokémon from our LowRes VQ-VAE baseline. 92
F.4 Sample generated Pokémon from our LowRes Pixel VQ-VAE. 93
F.5 Sample generated Pokémon from our MedRes VQ-VAE baseline. 94
F.6 Sample generated Pokémon from our MedRes Pixel VQ-VAE. 95
F.7 Sample generated Pokémon from our GAN baseline. 96
F.8 Sample generated Pokémon from our HiRes Pixel VQ-VAE. 97

Chapter 1

Introduction

The creation of characters in video games is a time consuming process. With games being an interactive blend of audio, video, and text, developing a single character requires significant effort on several fronts. This may include designing visual assets, writing backstories, determining character statistics and attributes, balancing the character in the context of the game, and more. In addition, it is an iterative process with dependencies between these different aspects. Thus a system capable of generating such interlinked features for a single character offers value in saving human effort and potentially improving the quality of the final character.

In this thesis, we specifically focus on two aspects of this system - visual asset design and character balance. We choose these aspects as they are aspects of a character that generally require significant human effort. This is especially true in the genre of games that we focus on - competitive multiplayer online games. In these games, new characters tend to reuse existing systems for several aspects of their design such as attacks, abilities and items. While the lore and writing behind a character are important, they may take a backseat to the character's balance and visual appearance. For instance, in games such as Pokémon, character lore is usually flavor text that is mostly irrelevant to gameplay. We thus focus on visual asset design and character balance in this thesis, as an initial exploration of balanced character generation.

1.1 Motivation: Visual Assets

When designing a character, their visual assets are one of, if not the most, important aspect. They are generally the first thing that players will actually see. From a design point of view, characters are restricted by certain constraints. Most importantly, the character's visual appearance must fit into the game both thematically and in terms of the game's visual identity. That is, the appearance must make sense within the context of the game and the art style must fit the game's visual aesthetic. For

instance, a robot would not make sense in a medieval farming simulator and photorealistic art would not fit into a cartoon world. Furthermore, creating art for a character does not stop at a single image. For instance, some content requires multiple images of the same character in different poses for animation purposes or marketing. In addition, many recent games have introduced “skins", which are alternate versions of a character that are purely cosmetic in nature. These may be anything from a simple recolour of the image to something more complex such as a reimagining of a character in a different setting. For the remainder of this work, we specifically consider pixel art, a visual aesthetic popular in video games, webcomics [85] and animations [35].

Given the recent popularity of diffusion based models such as Stable Diffusion [67], it might be natural to consider using such models for pixel art character generation. While training such a model from scratch is not feasible, there have been attempts at generating pixel art by finetuning these models [94]. However, these results are largely similar to the image generation results from our own work, as described in Chapter 3. In addition, our contributions better support secondary tasks such as image transformation, where we retain the same image but make specific changes such as modifying the colour palette. This is because our work offers finegrained control over individual pixels while this is more difficult in diffusion models.

In this thesis we propose a methodology that takes steps towards solving this problem - a system capable of both generating and transforming visual assets for video game characters. Specifically, we emphasise the use of representations in the machine learning domain. These representations are multi-dimensional information-rich vectors that can be used by machine learning models to build images. Once a model has learned these representations, we can then demonstrate the ability to transform images by modifying the learned representations. We elaborate further on this in Chapter 3.

1.2 Motivation: Game Balance

In recent years, several video games such as League of Legends, Overwatch and Pokémon have ballooned in popularity. In terms of raw viewers, they are comparable to regular sporting events and in 2018, the League of Legends World Championship was reported to have surpassed the Super Bowl in terms of number of viewers [56]. All these games fall under the umbrella of eSports and share a common element - they are all highly competitive multiplayer online video games. These games generally involve one or more players controlling one or more characters in competition with an opposing player or team to achieve some goal. In order to ensure balance and avoid staleness, these games are often updated to strengthen or weaken characters or to introduce new characters or gameplay mechanics. Games that follow this system are typically called “live service" games. The frequency of these updates can vary from a couple of weeks to multiple years. Regardless,

these changes must be made with care. A character that is vastly stronger or weaker than the other characters in the game could result in unsatisfactory experiences for many players. It is thus important to balance the game in terms of character power levels.

According to developers [66, 10], a game is said to be balanced if every character offers a fair experience to the players. That is, simply picking a character does not lead to a significant disadvantage to a player. For this thesis, we operate under this definition.

It is not uncommon to see updates (sometimes called patches) that have significant impact on a game. In League of Legends, the “Juggernaut" patch changed things so drastically that certain characters were present in nearly every professional match played [55]. In Pokémon, the introduction of a particular Pokémon necessitated the creation of an entirely new competitive tier to accommodate it [93]. Thus it is important for a developer to carefully evaluate the impact of any changes before public release. However, this is a difficult task to achieve.

Developers usually balance a game using a blend of their own domain knowledge and expertise, playtester feedback and user feedback [87]. The former is something that varies on an individual basis, while playtester feedback is acquired from a relatively small group compared to the much larger active playerbase. User feedback generally plays a large role in the determining balance changes since many popular competitive online multiplayer games have tens of millions of monthly active users [66]. While explicit feedback is important, implicit feedback such as statistics around the game such as what characters are picked the most and what characters win the most matches are equally important. By analyzing these statistics, developers can identify characters that have grown stronger as a result of changes they made. However, these statistics can be acquired only once the changes are pushed to a live server, that is, they cannot be acquired just from playtesters in an alpha or beta environment. Thus developers may not have a choice other than to push changes to the live version of the game and wait for these statistics. A bad change would then lead to an unsatisfactory player experience until the next patch is released.

To counteract this effect, we propose a new method to test the impact of changes in game balance. These changes could be small tweaks to certain characters or they could be larger, like testing the balance of a newly created character. Our meta discovery framework simulates a large number of battles in an alpha environment in order to arrive at an approximation of the statistics of the game after experiencing the developer's proposed changes. We then study these statistics and offer insights to developers on the effect of their changes. We explore this idea in more detail in Chapter 4.

1.3 Thesis Statement

We study the following hypotheses in this thesis.

Thesis Statement 1. *Representations for pixel art can be improved by utilizing VQ-VAEs due to their natural synergy with the artstyle.*

Thesis Statement 2. *The meta of a game can be approximated via a system that employs reinforcement learning to simulate a large number of game matches.*

1.4 Thesis Contributions

Our contributions in this thesis can be summarized as follows:

- We propose the usage of VQ-VAEs [90] to represent pixel art as a set of discrete embeddings, each mapped to groups of individual pixels. This focus on pixels synergizes well with pixel art where each individual pixel matters.
- We introduce the Pixel VQ-VAE, which uses two key enhancements, the PixelSight block and the Adapter layer, to improve performance on pixel art.
- We evaluate our Pixel VQ-VAE against several baselines on the quality of embeddings while also studying its performance on multiple downstream tasks.
- We demonstrate the superiority of Pixel VQ-VAE on pixel art tasks, especially for a high variance dataset.
- We propose a framework to evaluate changes to game balance via meta discovery.
- We introduce a dynamic team-generation system that adapts to changes in the metagame.
- We evaluate the usefulness of our framework in the environment of an online competitive game.
- We demonstrate the effectiveness of our framework in approximating the results of changes to game balance.
- We additionally explore the open problem of learning a metagame from scratch.

1.5 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2 we cover both the necessary background to understand this thesis as well as prior works related to our research. We follow this with Chapter 3, where we discuss our first major contribution, the Pixel VQ-VAE, a model for representing pixel

art. Chapter 4 describes our second contribution, a method to evaluate game balance through meta discovery. We then discuss the importance, impact and applications of both contributions, followed by our conclusions and a brief discussion on possible future work in Chapter 5.

Chapter 2

Background & Related Work

In this chapter we cover the necessary background required for understanding of this thesis. We then discuss prior and related work pertaining to both aspects of this thesis - representation of pixel art and video game character balance through meta discovery. In section 2.1, we offer the necessary background pertaining to pixel art, the idea of a metagame, and the relevant aspects of the Pokémon video game franchise. In section 2.2, we give a high level overview of the different aspects of machine learning that a reader needs to understand this work. We follow this up with a literature review for our work on pixel art representation (2.3) and video game character balance (2.4).

2.1 Background

In this section we first give brief overviews on pixel art (2.1.1) and the Pokémon video game franchise (2.1.2). Next we discuss the different visual attributes of Pokémon (2.1.3) which is important to the contents of Chapter 3. We then discuss the complex and interlinked attributes of Pokémon battles (2.1.4), followed by a high level overview of Pokémon battles (2.1.5). We then introduce the idea of a metagame (2.1.6) and discuss the competitive metagame of Pokémon (2.1.7) along with the Match-Making Rating (MMR) system used by the competitive community (2.1.8).

2.1.1 Pixel Art

Significant work in the computer vision domain focuses on photo-realistic art styles but there are several other styles used in popular media. Pixel art is one such art style, characterized by a restricted colour palette and discrete visible blocks of pixels. Originally created for 8-bit and 16-bit games which had limited memory and low resolutions, pixel art has remained popular, appearing in

Table 2.1: Examples of pixel art from the Pokémon video game franchise.

games like Minecraft, Pokémon, and Stardew Valley, as well as animations [35] and webcomics [85]. In the context of video games, pixel art images of characters, objects and other entities physically present in the game world are also called sprites. Table 2.1 depicts some examples of sprites from the Pokémon video game series. In each of the images we see visible individual pixels. Having these visible pixels is a defining characteristic of pixel art.

2.1.2 Pokémon

The Pokémon video game series is a popular set of Role-Playing Games (RPGs) that involves players capturing the titular creatures and fighting battles using them. There are nearly 1000 unique species of Pokémon with each possessing different appearances, attributes, characteristics and lore. We discuss Pokémon species further in the next section. Although we refrain from a comprehensive explanation of the Pokémon universe, we will highlight the key aspects of the games that are pertinent to this thesis. There are several different types of Pokémon games, but our focus is on the so-called "main-series" games. Generally released as a pair of games, they introduce a significant number of new Pokémon, a new area, new abilities, items, mechanics and more. Due to this, each pair of newly released games are considered to be a new "generation" of Pokémon. Fans and developers refer to the first pair of main-series games as "first generation" games, a term also used to refer to the re-releases of those games. At the time of this writing, the latest generation is 8, with generation 9 expected to release shortly.

2.1.3 Visual Attributes of Pokémon

Each Pokémon has several different factors impacting its appearance. Every Pokémon possesses 1 or 2 types which help define other properties of the Pokémon including their appearance and the moves they can learn. For example, a fire-type Pokémon might have a design that includes visible flames and/or a yellow, orange or red colour palette. In addition, they are generally capable of performing moves that manipulate fire. There are a total of 18 different Pokémon types. In addition, each Pokémon has an attribute that classifies it into one of 14 shapes, another attribute

for colour and one to two attributes denoting egg groups (used for breeding Pokémon). Table 2.2 demonstrates examples of these attributes. Every Pokémon species is distinct from other Pokémon species. However, within a particular species, there may be different variations. For instance, every Pokémon also has an alternate “shiny" version. These “shinies" are rare versions of Pokémon that are a near-exact copy of the base Pokémon, except that they use a different colour palette. Additionally, depending on the Pokémon in question, there may be changes in appearance due to in-game mechanics, narrative reasons, or attributes of the Pokémon such as their gender. Each of these different versions also possess a shiny sprite. Table 2.3 depicts some examples of these different sprites. We give this detail as it directly impacts the composition of our training data for the visual representation learning part of this thesis.

Pokémon	**Type 1**	**Type 2**	**Shape**	**colour**	**Egg Group 1**	**Egg Group 2**
	Rock	Ground	Serpentine	Gray	Mineral	None
	Grass	Poison	Bipedal, Tailless	Green	Fairy	Grass
	Fire	None	Quadruped	Yellow	Field	None

Table 2.2: Examples of the different visual attributes of Pokémon.

2.1.4 Attributes for Competitive Pokémon Battles

In terms of competitive battling, each Pokémon has several stats and attributes to take into account. While a complete understanding is not necessary to understand this thesis, they are all important

Table 2.3: Examples of the different sprites a single Pokémon can have. In the first row, there are significant differences in gameplay-mechanics between the two versions of the Pokémon. In the second and third rows, the changes are purely cosmetic.

aspects of competitive Pokémon battles and serve to illustrate the complexity and interlinked nature of these battles. Every Pokémon has a level (1-100), 6 base stats (Hit Points [HP], attack, defense, special attack, special defense, and speed) each in a range of 1-255, 1 to 4 moves (from over 800 options across 3 classes), one or two types (from 18 choices), Effort Values (EVs) ranging from 0 to 255 and Individual Values (IVs) ranging from 0-31 for each of the 6 base stats, a nature, 2 to 3 passive abilities (from over 250 options) and can hold up to 1 item (from over 500 options). Every move also has its own statistics such as Accuracy and Base Power, as well as one of the 18 types. There is also a 1 in 24 base chance of a move being a critical hit, which deals increased damage. Each Pokémon type has a set of other types that it is strong against, weak against, resistant to, and immune to. For example, a fire-type Pokémon will generally be weak to a water-type Pokémon and strong against a grass-type Pokémon. Further, Pokémon have a Same Type Attack Bonus (STAB) which causes them to deal more damage with moves that match their type. A Pokémon's overall stats are calculated using a combination of its base stats, effort values, individual values, level and nature. Some moves, abilities and items may also inflict one of 6 permanent status effects or over

150 temporary status effects each of which influence a Pokémon's attributes. In addition, certain moves, items, and abilities may temporarily increase or decrease a Pokémon's overall stats. The Pokémon's overall stats, in combination with a Pokémon's type, the move's base power, STAB, ability, item, stat changes and status effects determine the amount of damage a Pokémon can do with a single move. However, this damage is also affected by the same set of attributes for the opposing Pokémon.

2.1.5 Pokémon Battles

In a standard battle each player has a team of up to 6 Pokémon. These could be a team of Pokémon that the player has built, a team they obtained via other sources. In general, all Pokémon are usually level 100, utilize the maximum amount of EVs and IVs, have 4 moves, and equip an item. At the start of the battle, each player chooses one of their 6 Pokémon to send out. In some battle formats this choice will always be the first Pokémon in the roster and in others it can be random. Battles proceed in a turn-based fashion where both players perform an action simultaneously, the results are calculated and the players once again take an action. On a player's turn they may either choose one of the 4 moves that a Pokémon knows or they may switch to one of the available Pokémon on their team. If a player's Pokémon has fainted (its HP has been reduced to 0), the Pokémon will no longer be available for the battle. In this case, the player must switch to another Pokémon on their team. This is a free action, that is, the opponent does not take an action until a new Pokémon has been sent in (the player's next turn). This continues until a player forfeits or all 6 of a player's Pokémon have fainted. There are also alternate formats such as the Doubles format, where each player fields two Pokémon at a time. However this format is outside the scope of this thesis and we do not discuss it in further detail.

The team-building aspect of Pokémon adds in another layer of complexity. As seen in the previous section, just considering all the attributes of a single Pokémon is a significant amount of work. When teams come into play, this needs to be repeated 6 times. In addition, other factors need to be taken into account. These include how well each Pokémon synergizes with the team, major or minor weaknesses in the team against certain Pokémon, and the likelihood of such threats appearing in a battle, and others. Additionally, there are several distinct Pokémon archetypes and overall play-styles [40]. Thus this team-building aspect of Pokémon is an entire research topic of its own with several examples of prior work [79, 22].

2.1.6 Metagames

The metagame, or the meta, is a collection of knowledge that goes beyond the rules of the game itself. Sometimes referred to as the "Most Effective Tactical Advantage", in competitive games the

meta simply refers to the most popularly picked characters or teams. However, this also means that the metagame is not constant. It shifts and changes over time as players work to beat other players. Thus, if the meta is what is popular, a player would want to use a team strong against the meta teams as they would offer a better chance of victory. This in turn causes a rise in popularity of these teams that counter the current meta. Eventually, these counter-teams themselves become the new meta. In time, counters to this metagame arise and the cycle repeats.

Although the meta changes over time, this is not a rapid process. Considering that the meta is, in essence, a measure of popularity, the large number of active players in these games means that it may take time for new strategies and teams to enter the spotlight. Another effect of the popularity of such games is that players may be of wildly varying skill-levels [99]. This in turn impacts the metagame. Teams that require a high level of skill to utilize effectively are unlikely to be as popular as teams that are easier to use. To further complicate things, a single character could play entirely different roles in different teams [39].

The meta can also shift as a result of patches by the developers or external factors such as a community-wide decision to ban a character [93]. Regardless of the cause, these changes can influence the power levels of characters or even introduce new characters or gameplay mechanics. These changes generally result in larger shifts of the metagame as characters become more or less powerful than before. In many games, such as in competitive Pokémon or in League of Legends, developers use certain statistics such as the pickrate (percentage of matches a character is picked in) and winrate (percentage of matches a character actually wins) to help determine balance changes [76, 77, 66].

From a game theoretic perspective, it is interesting to consider if these metagames could ever be perfectly balanced. From our definition of balance in Chapter 1.2, we want a metagame where no character offers a disadvantage when selected. In theory, a metagame where every character offers no disadvantage is possible. Such a metagame could be like rock-paper-scissors, where each character wins against and loses to the same number of characters. However, this is not realistic for a number of reasons. First, these games are enormous in scope and complexity. Making a single change to a single character affects nearly every other character to some extent. Second, picking a character is not enough. Players must also be capable of utilizing that character. Every player has a different skill level and psychological state while playing the game. A character that wins in the majority of cases in the hands of one player could end up losing in the same scenarios when utilized by a different player. This is further complicated in games where there are multiple players on a single team. Third, developers operate upon the assumption that a perfectly balanced metagame is unrealistic [54, 25]. Thus they rarely make enormous sweeping changes to the entire game, but focus on smaller and more focused changes that target certain aspects of the game [66, 87].

2.1.7 The Competitive Metagame of Pokémon

Pokémon Showdown is a popular online simulator for Pokémon battles, which is sponsored by Smogon [73], a fan-run competitive community. There are several different battle formats or “tiers" used in Pokémon Showdown. Each of these tiers have their own metagame with varying strategies and characters. The most popular format according to publicly available statistics [74] is the random battle format where players do not build their own team. Instead, both players use a pseudo-randomly generated team. However, from a competitive standpoint, only tiers where users bring their own teams are considered. In addition, nearly every tier has several clauses put into place for both game balance and fun. These include a “species clause" which prevent multiple of the same Pokémon species from being used along with several others that ban particular moves or abilities. In addition, each tier also has its own particular banlist of Pokémon, moves, abilities or items. The tiering system, bans and clauses are all created and maintained by Smogon.

The most popular competitive tier is the OverUsed (OU) tier. Below OU is the UnderUsed (UU) tier. This continues on with the NeverUsed (NU), RarelyUsed (RU), and PU (no full form) tiers. In each case, all Pokémon with a certain weighted usage rate are classified as being a Pokémon of that tier. A Pokémon from a higher tier is banned from usage in lower tiers, although lower tier Pokémon can be used in any higher tier. In addition, each tier has an associated BanList (BL) which consists of Pokémon that are banned from that particular tier but do not have enough usage to be classified into a higher tier. For instance, the UnderUsed BanList (UUBL) consists of Pokémon banned from UU but under the required usage rate to be classified as OU. The one exception to this is the OU tier itself. All Pokémon banned from OU are placed in an entirely different tier - the Ubers tier. There is also one other competitive tier with special rules that does not fall under a usage-based system. Known as Little Cup (LC), it has several restrictions on the usable Pokémon and is generally separate from the other competitive tiers. Finally, although not considered a truly competitive tier, the Anything Goes (AG) tier is the highest and most broad of all tiers, allowing for any Pokémon to be picked and has only a single clause to prevent endless battles. It was created as a result of certain Pokémon being banned from the Ubers tier and thus has only two members. While out of the scope of this work, we mention it for completeness. In order of highest tier to lowest, the tiers are Anything Goes, Ubers, OU, UUBL, UU, NUBL, NU, RUBL, RU, PUBL, PU, and LC.

All bans are handled by a community-run council of high-level players [77]. We do not go into further detail on the mechanics of these bans as they are outside the scope of this thesis. More relevant is the mechanism through which Pokémon move up or down tiers [76]. There are two scenarios to consider. The first is at the start of a new generation of games which may introduce significant changes including adding or removing Pokémon, new game mechanics, items, abilities, moves etc. The second is a month-to-month maintenance of competitive integrity and balance. We

note that this is relevant only to the usage-based tiers, so the Anything Goes, Ubers, and Little Cup tiers are exempt from these systems. In the first scenario, all Pokémon are initially considered to be OU for the first month. Certain Pokémon, usually Pokémon from the Ubers tier in the previous generation, are banned by the council and the standard Smogon-wide clauses still apply. After 1 month of play, all Pokémon above 4.52% usage rate remain in OU with all other Pokémon being demoted to the UU tier. This percentage corresponds to having a greater than 50% chance of encountering a Pokémon at least once over 15 battles [78]. After another month, all Pokémon above the same usage threshold stay in UU while the remaining go down to NU. This is repeated every month until the final tier, at which point all tiers have been established. For the first month of a tier's existence, it is considered to be open, that is, there are no bans during this period. One month after a tier has been established, Pokémon are allowed to move up or down tiers based on usage and the council is allowed to ban Pokémon. From this point, tier shifts happen in 3 month cycles. For the first month, all Pokémon below a usage rate of 1.53% drop to the lower tier. For the second month, the overall usage rate is calculated while weighing both months equally with all Pokémon below a usage rate of 2.28% drop to a lower tier. For the third month, this percentage increases to 4.52% with all three tiers being weighed equally[1]. In general, Pokémon can rise in tier(s) only in the third month while they can fall in tier on any month. This is a complex, involved, and lengthy process, and is similar to the process used by industry game companies to establish game balance. If we could instead reasonably predict how the balance of a game would develop given some change, we could greatly speed up this process

Finally, we note that these percentages mentioned are not just the raw usage percentages for that tier. Rather, only the usage rate of players with a certain matchmaking rating are considered, specifically an ELO rating of 1695 for OU and 1630 for all other tiers [76]. In the next section we discuss the different matchmaking ratings on Pokémon Showdown and their impact on this work.

2.1.8 Matchmaking Ratings

Pokémon Showdown is the most popular competitive online Pokémon battle simulator with millions of battles played monthly [74]. In every battle format, there is a competitive ladder which acts as a leaderboard of the best players in that particular battle format. For every win, a user's rating increases and for every loss it decreases. Users cannot decide who they face. Rather, the matchmaking algorithm pits a user in a battle against a different user who has a similar rating.

There are 3 different ratings available on Pokémon Showdown - ELO, Glicko-1 and GXE [75]. The ELO system is similar to that used in chess and is the rating used to sort the competitive ladder on Pokémon Showdown. All players start at a rating of 1000 and gain or lose points for every win or loss respectively. The amount of points lost or gained depends upon the rating of the player's

[1] Previous generations used a different weighing scheme but this is no longer in effect.

opponent. Losing to a lower ranked opponent results in losing more points than would be obtained by beating them. Similarly, beating a higher rank opponent results in more points gained. The second metric, Glicko-1, is a modification of the ELO system that uses a Rating Deviation (RD) which adds in the level of uncertainty in the Glicko-1 estimate of the user's rating. Finally, the GXE measures the odds of a player winning a battle against a randomly selected opponent from the ladder. This is based on Glicko-1 and is to be considered a means of interpreting the Glicko-1 estimate.

2.2 Background: Machine Learning

In this section we cover, at a high level, the machine learning background required to understand this thesis. We begin with an introduction to artificial neural networks (2.2.1) followed by an exploration of convolutional neural networks (2.2.2). We then discuss representations in machine learning (2.2.3) along with Autoencoder and Variational Autoencoders (VAE) (2.2.4). Following this we examine some approaches towards image generation (2.2.5). We then discuss the Vector Quantized Variational Autoencoder (VQ-VAE) (2.2.6) in detail due to its relevance in Chapter 3. Finally, we give a brief introduction to reinforcement learning (2.2.7).

2.2.1 Artificial Neural Networks

Artificial Neural Networks or ANNs [1] are the fundamental machine learning model. Typically just referred to as a neural network or a feedforward neural network, they are used in almost every domain to make predictions about some given information. This happens in two phases. In the first phase, the training phase, a large amount of input and output pairs are passed through the model. More specifically, a model receives an input and attempts to predict the output corresponding to that input. This is done by learning a set of parameters which in conjunction with non-linear functions transform a given input into a prediction. A node or a "neuron" is responsible for this operation. A "loss" function is used in order to calculate the distance between the predicted class and the true class. This loss value is then utilized in order to update the model via some optimizer, with the most common being gradient descent-based approaches [82]. In the second phase, the testing phase, the model is passed in inputs and the predictions are compared with the true outputs in order to gain an understanding of the model's performance.

ANNs generally consist of 3 components, an input layer, one or more hidden layers, and an output layer. Each layer consists of a number of neurons. The number of neurons determines the size of the output and imposes a requirement on the size of the input. The input layer transforms the inputs which are then fed to the first hidden layer. Each hidden layer transforms the input matrix and feeds it to the next layer. Finally, the output layer transforms the output of the final

hidden layer into a prediction vector (the size of which varies based on the task). For instance, if the task is to predict if the sentiment of some text is positive, neutral or negative, the output layer would have 3 nodes, where the value of each node would correspond to the confidence of one particular sentiment.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks [42] are a class of neural networks primarily used for tasks associated with images, such as image classification. It operates on the idea that an image is simply a matrix of pixel values. That is, an RGB image of size 64x64 is simply a 64x64x3 matrix where each dimension corresponds to the height of the image, the width of the image, and the number of image channels, respectively. At a high level, CNNs process clumps of neighboring pixels together in order to "see" the image as a whole. This is done in a progressive manner where only small lines and edges may be visible to the model initially and over subsequent layers, the overall image comes into focus.

More specifically, a single layer in a CNN defines a number of kernels (or filters) each having the same size but different parameters. Each of these filters are then convolved over the entire image to obtain a transformed matrix. The number of filters determines the number of channels in the output while the size of the filter determines the dimensions of the output. For instance, a 3x3 filter that is convolved across a 5x5 image would result in an output of shape 3x3x1. If there were 32 filters, the output shape would then be 3x3x32. Each layer may also use padding (adding a row and column of some value, usually 0, at the edges of the image) and stride (moving the convolutional kernel by more than one step). In a CNN, the parameters that are learned are the weights of the kernels themselves. Each kernel can be considered similar to a neuron in an ANN.

CNNs generally use odd filter sizes such as 3x3, 5x5 and 7x7, with 3x3 filters being the most popular. 1x1 filters are generally used in specific scenarios such as reducing model complexity [84]. They do so by reducing the number of channels in the image while retaining the same height and width. The idea here is that the 1x1 filter will allow for dimensionality reduction of the matrix. Specifically, the information is compressed into a smaller number of channels.

2.2.3 Representations

Neural networks have been used for a variety of different tasks. These include classification [60, 34], translation or transformation [41, 45, 83], and generation [14, 59, 88] of both text and images. While these were generally treated as independent tasks, the advent of embeddings [47] introduced an alternate way forward - using shared representations of content as a common starting point. These representations are information-rich multi-dimensional vectors learned by machine learning models for use in downstream tasks. Each of the dimensions in these vectors correspond to some aspect

of the content being represented. When taken in totality, they form the model's understanding of that piece of content.

The primary advantage of learning representations of content is their usefulness in saving human effort. This is especially true in the case of Natural Language Processing (NLP) where word embeddings [47] completely redefined the field, with many NLP systems today relying on learned embeddings. For a more concrete example, consider the following scenario. We have a large collection of images and wish to perform two tasks - image generation and image transformation. That is, we want to generate new images similar to the ones in the collection and we want to be able to transform the images, say change their colour scheme. Although we could certainly approach each task independently and achieve a certain degree of success, there is a common thread between the two tasks that could be shared. They both require an understanding of the images themselves. Thus by instead focusing on learning a good representation of these images, the results of both tasks could potentially be improved.

2.2.4 Autoencoders & Variational Autoencoders

Autoencoders (AE) [97] are a class of neural networks that are used to learn representations of data. They consist of two components, an encoder which maps the input to a point in the latent space and a decoder which maps the point in the latent space to an approximation of the original input. Thus the input and output to an autoencoder are the same. It is the representation learned which is of primary importance. However, autoencoders are not generative models and the learned latent spaces do not hold much information. Thus they are generally used for dimensionality reduction instead.

On the other hand, Variational Autoencoders (VAEs) [31] and their variants dominate the field due to their improved representational capabilities [13, 9, 90]. Like autoencoders, they also consist of an encoder and a decoder. However, unlike an autoencoder, VAEs do not simply map an input to the latent space. Rather, the encoder learns a probability distribution of the latent space while the decoder samples from this distribution to recreate the original input. Although the representations learned by VAEs have achieved success across many domains, in the case of images, VAEs tend to produce blurry and unclear outputs [32]. For pixel art in particular, this is a major problem as having discrete, visible pixels is an essential characteristic of the art style. There have been several works [9, 36, 90, 61] that have demonstrated success in enhancing the quality of the generated images, though none have been adapted to pixel art.

2.2.5 Image Generation

Image generation, as the name suggests, is the generation of new images based on some existing images or signals. A popular approach towards image generation is the Generative Adversarial Network (GAN) [14] alongside its numerous variants that focus on improving the quality of generated images [59, 45]. A GAN has two components - a generator and a discriminator. The generator learns to generate images similar to the training data while the discriminator learns to distinguish between a real image and an image created by the generator. Training progresses in an adversarial manner until it reaches a point where the discriminator is unable to differentiate between a real and generated image. However, GANs generally require large corpora of data to achieve good results. On the other hand, most art styles that lack natural images (such as pixel art) have limited data available as they are generally hand-authored. This is further complicated in other approaches like diffusion models [51, 67] which require labeled data. Further, while these diffusion models have been used to generate pixel art, the low resolution and varied structure of pixel art would make it difficult to generate sprites for specific games [6]. Another class of image generation models, PixelCNNs [88, 89], are auto-regressive in nature. That is, they generate images one pixel at a time, which makes their use for pixel art appealing. However, the learning problem grows more complex due to modeling relationships both between individual pixels and their colour channels. We handle this complexity further in this thesis by introducing the Pixel VQ-VAE which replaces multi-dimensional (RGB) pixels with single-dimensional encodings.

2.2.6 Vector Quantized Variational Autoencoders

Similar to a VAE, the Vector Quantized VAE (VQ-VAE) [90] consists of an encoder, decoder and a latent space. However, unlike the VAE, it learns a discrete latent space. This is done via a codebook which maps discrete encodings (integers) to continuous embeddings (vectors) in the latent space. More specifically, the encoder takes in an image and outputs a grid of high-dimensional vectors. For each of these vectors, the closest embedding in the codebook is identified and the corresponding encoding is returned. That is, the output of the encoder is quantized. The decoder then reconstructs the original image back from these quantized encodings. The number of unique encodings to learn is a hyperparameter while the embeddings are learned during the training process. Specifically, there are two related learning problems here. The first is to learn codebook vectors that are close to the encoder outputs and the other is to learn encoder outputs close to the codebook vector. The loss function is thus

$$L = logp(x|z_q(x)) + ||sg[z_e(x)] - e||_2^2 + \beta||z_e(x) - sg[e]||_2^2$$

Where sg is the stopgradient operator which stops the flow of a gradient, z_e is the encoder, z_q is the decoder, and e_i refers to the embeddings.

The first term in the loss function is the standard reconstruction loss. In the forward pass, rather than the encoder output, the nearest embedding is passed to the decoder. During the backward pass, the decoder's loss is passed back to the encoder in an unaltered fashion. The idea here is that since both the encoder output and the decoder input share the same latent space, the gradients will still contain useful information. The second term is called the codebook loss and is used to train the codebook vectors. There are no gradients that flow through the encoder output simply because it is used only to update the codebook. Essentially, the stopgradient operator causes its operand to be treated as a constraint. Thus the codebook loss treats the encoder output as a constant in order to bring the nearest embedding closer to it. Similarly, the final term, the commitment loss, treats the nearest embedding as a constant in order to bring the encoder outputs closer to the nearest embedding. Note that the decoder optimizes only for the reconstruction loss due to the stopgradient operators. The encoder optimizes for both the reconstruction loss and the commitment loss while the embeddings are optimized only by the codebook loss.

We further note that the latent representation of an image in a VQ-VAE is not just a single discrete encoding. Rather, the VQ-VAE produces a representation consisting of a grid of encodings. Consider a 64x64 image that we pass into a VQ-VAE. Taking an extreme example, our encoder could output 64x64 high-dimensional vectors which are then quantized into 64x64 encodings. Each of these encodings would then correspond to a single pixel in the reconstructed image. If instead the encoder output 32x32 high-dimensional vectors then each encoding would correspond to a 2x2 block of pixels in the reconstructed image. More broadly, each encoding in the grid of discrete encodings obtained from the quantization process corresponds to some group of pixels in the reconstructed image. This also means that the position of each encoding is important. The first encoding, in the top-left position of the grid, will correspond to a group of pixels at the top-left corner of the final image. This property of VQ-VAEs allows us to have fine-grained control over individual blocks of pixels, thus forming a natural synergy with pixel art. Thus the VQ-VAE is the basis of our work on improving pixel art representation. To the best of our knowledge, we are the first to leverage this synergy for pixel art.

Like traditional embedding models [47], a VQ-VAE is generally used to generate embeddings while a different model, such as a PixelCNN or a Transformer, uses this representation in downstream tasks. Thus the quality of downstream tasks are dependent on the downstream model as well, not just the learned representations. However, unlike traditional embedding models which learn embeddings that display characteristics of an entity when analyzing the latent space [24], VQ-VAEs do not have such latent spaces. This is primarily due to the fact that the learned embeddings correspond to individual patches of an image and not the image as a whole. Thus a traditional method of embedding analysis such as t-SNE does not reveal anything about the quality of the learned representation.

2.2.7 Reinforcement Learning

Reinforcement learning [3] is an area of machine learning which deals with training entities (called agents) to interact with environments in order to reach some goal. An environment can be considered the world where the agent acts. A state represents one moment or location of an environment where an agent can make some decision, one of a number of options, called as an action. Whenever an agent takes an action it receives the next state (the environment after the action completes) and a reward. This reward serves as feedback on the value of taking that action in the state, which the agent then learns from. A discount factor γ is generally applied to the reward to cause future rewards to weigh exponentially less than present rewards. Agents select actions according to a learned policy, that is, a function that takes in a state and outputs an action to take. In Deep Reinforcement Learning (DRL), this function is a deep learning model like a neural network [48]. The goal of all reinforcement learning algorithms is to learn a maximum policy, one that maximizes the cumulative rewards over time.

In the episodic scenario considered later in this thesis, all agents start in a starting state S_0 and keep interacting with the environment until they reach a terminal state S_T. At each timestep t, the agent uses the current state S_t to take an action A_t and receives the next state S_{t+1} and the reward R_t. Reinforcement learning models learn by playing out a large number of such episodes while periodically updating the policy so as to take into account the rewards and the associated transitions. We note that we do not contribute to reinforcement learning as a research area, but employ it to study metagame discovery.

2.3 Pixel Art Representation

In this section we cover prior and related work on representing pixel art (2.3.1). We additionally discuss various downstream tasks for representations, specifically image generation and image transformation, in section 2.3.2.

2.3.1 Representing Pixel Art

Pixel art appears commonly in video games, thus work on representing video game content often requires representing pixel art. The Video Game Level Corpus [81] represents game levels as a set of discrete symbols. Jadhav & Guzdial [24] utilized a VAE to learn embeddings of game level components. Karth et. al [29] used a VQ-VAE to learn encodings which were then used for map generation through a secondary process. These approaches do not focus directly on the pixel art representation. Closer to this work, Gonzalez, Guzdial and Ramos [13] used a convolutional VAE to learn representations of Pokémon art, albeit not pixel art, along with their associated type

attributes. Then, by changing these attributes, they were able to generate variations of a given Pokémon. Although their work resulted in modifications to both the colour and the appearance of the Pokémon, their VAE suffered from the aforementioned blurriness issue. Additionally, while this work dealt with image transformation using a latent representation, it did not explore the possibilities of generation.

2.3.2 Downstream Tasks

There have been several approaches for tasks that we consider downstream to learned embeddings. Serpa and Rodrigues [62] used GANs to convert 2D line-art of sprites into grey and coloured versions. Pokemon2Pokemon [95] used CycleGAN [98, 23] to modify the overall colour palette of a Pokémon based on a user-specified Pokémon type. Liapis [43] studied the relationship between a Pokémon's visual appearance and its type attributes. They then used an evolutionary algorithm to alter the Pokémon's colour palette using type information. All of these fall under image transformation, specifically a palette swapping task which we demonstrate in Chapter 3.

Considering image generation, Horsley and Perez-Liebana [20] used a Deep Convolutional GAN (DCGAN) to generate new pixel art characters. However the generated images were blurry and lacked the characteristic property of pixel art (discrete visible pixels). Both Onsager [53] and Japeal [27] used a hand-crafted algorithm that combined multiple Pokémon to generate new "fused" Pokémon. Despite good results, the generations are formulaic and require a large amount of manual effort to implement, both issues that could be solved via machine learning. Finally, while there are several other open-source, unpublished works that work on Pokémon generation [38, 33], they primarily focus on the generation aspect and not the representations themselves.

Pokérator [12] aims to generate new Pokémon names and descriptions based on user input. While this is a potential downstream task to learned embeddings, we do not cover this aspect of character generation as generating coherent and sensible text that fits into an established world is a separate research problem of its own.

2.4 Video Game Character Balance

We now cover research related to our work on video game character balance. In section 2.4.1, we examine approaches towards training an AI agent to play a game. We then cover the different aspects of game balance through meta discovery. Specifically, in section 2.4.2, we examine systems to generate teams. In section 2.4.3 we study prior work examining the impact of balance changes in games as well as approaches surrounding the idea of meta discovery.

2.4.1 Game-Playing Agents

Significant prior work has focused on developing AI agents capable of playing games, with some such as MuZero [69] even achieving superhuman performance on certain board games. On the topic of video games, OpenAI Five [4] played Dota 2 and defeated professional players and teams in a limited game environment. However, these methods require a large amount of computation to train and thus may not be the most feasible in a game development scenario. Closer to our work, there have been several approaches towards training agents to play Pokémon. Given that team-building is an entirely different aspect of Pokémon, the majority of these works use randomly or pseudo-randomly generated teams. The works of pmariglia [57], and Norström [52] along with Technical Machine [79], and Percymon [17] use search-based algorithms for competitive battling. Of these approaches, pmariglia fared the best, achieving a peak ELO of 1461 (66.9% GXE) on the Random Battle ladder and an ELO of 1450 (65.9% GXE) on the OU ladder (using a user-defined team). As discussed in Section 2.1.8, GXE indicates the odds of a player winning a battle against a randomly selected opponent from the ladder.

The Showdown AI Competition [40] made a case for training agents for Pokémon Showdown and introduced several baselines. There are also several works [65, 28, 30] that used reinforcement learning techniques to train battle agents. The Pokémon Battle Environment [72] introduced an alternative to Pokémon Showdown built to support reinforcement learning. They demonstrated and evaluated two agents based on asynchronous Q-Learning algorithms. However, all these systems were evaluated only against their own baselines and not on a competitive ladder. This makes it difficult to objectively ascertain their quality. Huang and Lee [21] developed the state-of-the-art agent for Pokémon battling. They used Proximal Policy Optimization (PPO), a reinforcement learning algorithm, and trained for nearly 4 million battles via self-play. They evaluate their system against several baselines including pmariglia as well as on the Pokémon Showdown ladder. They obtained a Glicko-1 Rating of 1677 (72% GXE) over 1500 random battles. Finally, Future Sight AI [22] uses a predictive algorithm to determine the winner at any point during the battle. They use this in conjunction with a search based algorithm in order to play battles. This system was also evaluated on the Pokémon Showdown ladder and on the OU ladder achieved an average ELO of 1547 and peaked at 1630, beating pmariglia in both cases.

We note that while the achievements of Future Sight AI, and Huang and Lee's work are the state of the art, they are nowhere close to a human elite. At the time of writing, the number 1 rank on the random battle ladder and the OU ladder are at 2497 ELO (93.9% GXE) and 2070 ELO (89.1% GXE) respectively. The lowest publicly available ranking for both ladders is rank 500, which in OU is 1717 ELO (77.7% GXE) and in random battles is 2115 ELO (84.5% GXE). We mention this not to downplay prior work, but rather to illustrate the difficulty of the task at hand.

2.4.2 Team-Building

Team-building is essential in team-based competitive games. Given that the meta consists of popular picks, teams must be capable of adequately competing against the meta. In addition, when attempting to simulate the metagame, the teams that are generated directly form the meta. Thus team-building is an essential component of meta discovery. Several approaches have analyzed this aspect of competitive games. Summerville, Cook and Steenhuisen [80] developed a system to predict the heroes picked during the drafting phase in Dota 2 while Hong, Lee and Yang [18] did the same for League of Legends. Both Crane et. al [7] and Oliveira et. al [8] studied team-building in Pokémon GO. The former tested several team generation systems to generate teams capable of regularly winning in an analyzed metagame while the latter tested several optimization algorithms to counter a given team. However, due to Pokémon GO being distinct from and far simpler, the system each paper used to determine the results of a battle are not applicable to us. Focusing on Pokémon Showdown in particular, Rejim [63] developed a collaborative system to recommend additions to a team based on user specified inputs and usage statistics. Future Sight AI [22] also generates teams based on usage statistics but has no publicly available algorithm or implementation. Technical Machine [79] either steals teams from opponents it loses to or uses a weighted random selection based on usage statistics. Our final team-building methodology uses techniques from Rejim and Technical Machine in order to dynamically generate teams based on the metagame state.

2.4.3 Metagame Discovery & Game Balance

Several works have studied the modification of game rules and their resulting impact. These can be treated as changes to gameplay mechanics instead of changes to characters. One such work [86] focused on the game of chess, defining several alternate rulesets to the game and analyzing the resultant game after achieving near-optimal performance using AlphaZero. Though effective for chess, competitive Pokémon battling has no agent that approaches such a level. Jaffe et al. [26] restricted particular actions in a simple competitive game to evaluate the effects on game balance. However, their tool was built for perfect-information games, while Pokémon is an imperfect-information game. Zook, Fruchter and Riedl [100] balanced parameters of a game by combining an automated playtester with an active learning framework. However, they use a flat set of parameters and acknowledge that alternate techniques might be required for more complex scenarios. Silva et. al [46] used an evolutionary algorithm to find a set of changes to Hearthstone cards such that decks approach a 50% winrate. While an effective methodology, as discussed in earlier sections, many online competitive games determine balance by a combination of pickrate as well as winrate. Finally, Hernandez et. al [15] developed a system to balance competitive games by finding a parameterization of the optimal metagame but required game balance to be represented as a parameterizable graph.

Prior work on meta discovery is scarce, though there has been some work on analyzing and identifying the meta. Lee and Ramler [39] worked on identifying and evaluating non-meta strategies in League of Legends while Peabody [55] analyzed the metagame shifts between patches in the same game. The only work that we are aware of to perform meta discovery, MEDIROMA [2] discovers the meta by clustering large numbers of League of Legends games. However, this approach only identifies if a particular hero is in the meta or not. While useful, this does not offer the statistics that a complete simulation would offer. Possibly the closest in theory to our work, the VGC AI Competition [64] presented a framework to balance the Pokémon metagame. They sought two types of agents, Player Agents to play battles and Balance Agents to manipulate Pokémon attributes in order to balance the game. However, at the time of writing, this is still only a call for agents with no baselines currently in existence. In addition, they use a different competitive ruleset (VGC) and eschewed the use of Pokémon Showdown in favor of another simulator [72] in order to meet their competition restrictions. However, due to Pokémon Showdown's popularity, it offers simpler means of evaluation against human opponents as well as more easily accessible information about the metagame.

Chapter 3

Pixel Art Representation

In this chapter we explore in detail the first major contribution of this thesis - an improved methodology for representing pixel art. We briefly examine the need for representation over simple generation in the context of this thesis - the generation of balanced characters in competitive video games. We then introduce the Pixel VQ-VAE and study the embeddings learned. We further examine the usage of these embeddings in two different downstream tasks.

Given our goal of generating characters, it would be natural to question learning representations over simply training a generative model. However, as discussed in Section 1.1, our task is not to simply just generate a single visual asset. Rather, we require a system that can not only generate assets but also modify them. Consider the case of Pokémon. As described in Section 2.1.3, there are nearly 1000 unique Pokémon species, each with their own particular appearances, characteristics, attributes and lore. Even at the most basic level, there are a minimum of two sprites required - the base sprite of the Pokémon and a shiny (alternate colour palette) sprite. However, in practice there may be many more sprites. Depending on the particular Pokémon, there may also be sprites for different animation frames, female and male versions, cosmetic changes, changes based on in-game mechanics, sprites of the Pokémon viewed from the back and shiny versions of each of these. Due to the nature of generative models, there is no guarantee that the model can generate the same image twice with only minor changes. At least, not without utilizing a latent representation of some kind. On the other hand, a shared common representation would allow for more controllability. Instead of attempting to generate a new sprite for each scenario, we instead learn a single representation shared across all sprites for a new Pokémon. Generating altered sprites then becomes a matter of simply modifying this representation as we demonstrate later in this work.

While there exist several machine learning models capable of learning representations, the VQ-VAE possess a special property that makes its usage appealing for pixel art. As discussed in Section 2.2.6, the embeddings learned by a VQ-VAE form a natural synergy with pixel art. We further

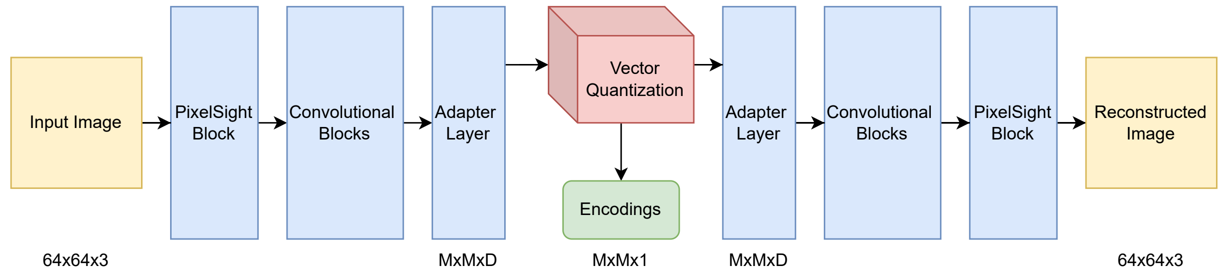

Figure 3.1: Pixel VQ-VAE Architecture.

modify the VQ-VAE to introduce the Pixel VQ-VAE which has two enhancements to boost its performance. We use the Pixel VQ-VAE to learn representations of pixel art and then demonstrate the applications of the learned representations for the tasks of image generation and image transformation.

We use Python for all code in this work. Specifically, we use the PyTorch, NumPy, and Pandas libraries for machine learning and some elements of data processing, Pillow for image processing, and Jupyter Notebook for initial experiments. We make all code publicly available for reproducibility.[1]

3.1 Pixel VQ-VAE

Our Pixel VQ-VAE, which makes use of two enhancements over a traditional VQ-VAE to better represent pixel art. Before we proceed with a detailed discussion of our enhancements, we overview some relevant properties and hyperparameters. Since the VQ-VAE performs a nearest neighbor search to discretize embeddings, it requires that the number of encoder output channels be equal to the dimensionality of the learned embeddings D. K is the number of unique discrete encodings (integers) associated with those embeddings (vectors). Furthermore, we define an additional hyperparameter M, the encoding-pixel correspondence. This parameter is implicitly set via the formula $M = I/2^L$ where I is the size of the input and L is the number of scaling convolutional blocks in the model. The encoder outputs embeddings of shape $I^2/M \times D$ which are discretized into I^2/M encodings that are then converted back into a $I \times I \times 3$ image by the decoder. As an example, given a $64 \times 64 \times 3$ image, if we set $D = 32$, $K = 128$, and $L = 2$, then $M = 16$. This gives us $64 * 64/16 = 256$ embeddings of dimensionality 32 to represent the entire image where each embedding corresponds to one of K (128) discrete encodings. We note that larger values of L result in lower values of M. That is, the number of layers in the model and the number of encodings are inversely correlated.

We introduce two new enhancements - the “PixelSight" block and the “Adapter" layer. These

[1] https://github.com/akashsara/fusion-dance

enhancements solve two distinct problems. The first is a problem that arises with the use of CNNs for pixel art. Specifically, CNNs tend to process clumps of neighboring pixels together. This property is a major reason for the popularity of CNNs. They progressively "see" more of an image, starting from lines and edges and over time identifying patterns and textures. However, this property is counter-intuitive to pixel art where we wish to specifically consider individual pixels. When taking into account the general low resolution of pixel art, processing the image in clumps leads to even further loss of information. The second problem arises due to the general VQ-VAE architecture. Specifically, the standard practice when using convolutional layers is to halve the input size and double the number of filters at each convolutional layer. However, since the encoder is restricted to an output dimensionality of D, the final convolutional layer must have D filters. At the same time, this parameter D has a significant impact on our model. Low values result in too few filters for the model to learn effectively while higher values led to giant models that have trouble converging.

As a solution to the first problem, we introduce the PixelSight block. It consists of a convolutional layer with filter size 1x1 and a stride of 1, a batch normalization layer and the ReLU activation function. While a convolutional layer with filter size 1x1 is not new, having traditionally been used for reducing model complexity [84], to the best of our knowledge it has never been used specifically for pixel art. When we slide this convolutional layer over the image, we essentially focus on a single pixel at a time. This allows the model to take an initially granular view of the input, giving each pixel due consideration. We note that our PixelSight block is not restricted to only the VQ-VAE but can be plugged in to improve the pixel art performance of any convolutional model. We demonstrate the improvements gained on a traditional VAE in Appendix A.

In order to solve our second problem, we introduce the Adapter convolutional layer. This layer uses a 1x1 with stride 1 in the same manner as traditional literature [84] to reduce the number of filters down to D while retaining the remaining shape of the vector. This thus allows us to use an arbitrary number of filters for the preceding layers. More specifically, this allows for the standard approach of halving the input size and doubling the number of filters to be used in the preceding layers.

Figure 3.1 illustrates our general model architecture. The encoder consists of the PixelSight block, L convolutional blocks, and the Adapter layer. Each of the convolutional blocks consist of 2x2 convolutional layers with stride 2 like in [20, 13], a batch normalization layer and ReLU activation as per standard practice. The Adapter layer uses a linear activation function to leave the embedding space unrestricted. We calculate the number of filters for the remaining layers such that the final one has F filters, with each preceding one having half as many. The decoder is a mirrored version of the encoder that starts with the Adapter layer to increase the number of filters back to F, followed by L transposed convolutional blocks, a PixelSight block with a transposed convolutional layer and a sigmoid activation.

3.2 Data

In this section we discuss the datasets that we use along with the preprocessing and data augmentation steps taken for use in our experiments in the next few sections.

Dataset	Example 1	Example 2	Example 3
Pokémon			
Sprites			

Table 3.1: Examples of sprites from both our datasets. These are images taken before the preprocessing step. Notice how the Pokémon exhibit high variations between individual Pokémon whereas the Sprites dataset is quite consistent.

3.2.1 Datasets

We perform our experiments on two different datasets. The first is compiled from several different Pokémon games [49], and consists of Pokémon sprites in a wide variety of shape, size and colour, with no consistent features between them. In total we have 649 unique Pokémon species, but as discussed in Section 2.2, each Pokémon can have multiple sprites. Adding in to consideration that different games may have different sprites for the same Pokémon, we have a total of 7937 sprites before preprocessing. We chose to use Pokémon not only due to its usage in prior work [43], but also due to the high variance present in the dataset. This variance is important since many models are capable of generating consistent structures in low variance environments but struggle with high variance. In contrast, our second dataset, the Sprites Dataset [96], consists of sprites of extremely consistent style. It is made up of 1296 humanoid character sprites in a variety of poses and orientations, with a total of 93,312 images. It is based on the Liberated Pixel Cup [2], an open source video game artwork competition. Table 3.1 depicts some images from both datasets.

[2] https://lpc.opengameart.org/

3.2.2 Preprocessing

For the Pokémon dataset, we perform augmentation on our training data in a manner similar to prior work [13] by using 4 different backgrounds - black, white and two randomly generated noisy backgrounds. The first two are to deal with Pokémon sprites having very dark or very light colour palettes being difficult to view in a similarly coloured background. The remaining two noisy backgrounds are used only for our training data, to both act as a form of regularization for the models and to help the models learn to ignore the background. We also perform horizontal flips and 4 random rotations (up to 30 degrees) in either direction. We acknowledge that these rotations may cause harmful effects in terms of pixel art style in exchange for a much larger dataset. However, based on our experiments (Appendix B), we found that this trade-off worked in our favor as it led to significant improvement in all results for all models. Due to different Pokémon games having images of different sizes, we resized them to 64x64 using bicubic interpolation. For the Sprites dataset, we only perform horizontal flips. We do not perform the background-based augmentation step as the Sprites dataset consists of images with an existing black background by default. Additionally, we avoid rotations since we have a sufficiently large dataset already. We split the data using a standard 85:5:10 train:validation:test split. While splitting the data, we ensure that no sprites of the same entity are duplicated in different splits by using the sprite's ID. Thus, the specific percentages for each dataset differ slightly as certain entities had more sprites associated with them. Our final Pokémon dataset has 168,334 training, 3,046 validation and 6,999 test images while the Sprites dataset consists of 129,600 training, 14,400 validation and 42,624 test images.

3.3 Image Representation

In this section we describe the implementational details of our Pixel VQ-VAE as well as several baseline models. We then compare these different models on the quality of the learned embeddings to understand if our work better models pixel art. We additionally perform an ablation study to validate our enhancements.

3.3.1 Pixel VQ-VAE Hyperparameters

We train 3 models of decreasing encoding-pixel correspondence by setting $M = 16, 4, 1$ ($L = 2, 1, 0$). As this results in more detail due to the larger number of encodings (I^2/M), we term the three models as Pixel VQ-VAE LowRes, MedRes and HiRes respectively with a note that the LowRes model has the most layers ($L = 2$) and thus the most parameters. We empirically determined that the hyperparameters $K = 256$, $F = 512$, offered the best performance while still retaining a low value for K. We also found that $D = 64$ worked best for the LowRes Pokémon model while $D = 64$

worked best for all other models. All models use an Adam optimizer with learning rate 0.0001 and batch size 64. We use a Mean Squared Error (MSE) reconstruction loss term and train the models to convergence (25 epochs).

3.3.2 Baselines

Our first baseline is a standard VAE consisting of a mirrored encoder-decoder architecture each with 4 convolutional blocks. We train it to full convergence (25 epochs) using the Adam optimizer with a batch size of 64 and a default learning rate (0.0001). We use the standard VAE training objective of the sum of the Mean Squared Error and the KL-Divergence. In addition, we also train two VQ-VAEs that do not use our enhancements. Specifically, we train a LowRes version and a MedRes version. We emphasize that a HiRes VQ-VAE is not feasible since it requires the number of scaling convolutional blocks to be $L = 0$. The HiRes version is only possible for the Pixel VQ-VAE due to the enhancements we use. This is because our enhancements add layers that do not affect the size of the image (1x1 convolutions). In the case of these baseline VQ-VAEs, we performed a hyper-parameter search for D and found that $D = 32$ worked best, with the exception of the Sprites LowRes model where $D = 64$ worked best.

3.3.3 Embedding Quality

We evaluate all models in terms of the Mean Squared Error (MSE) and Structural Similarity Index Metric (SSIM) of their reconstructions against the ground truth images of the test set. MSE is a per-pixel loss function which compares every pixel between the ground truth image and the reconstructed image while SSIM compares the structural similarity between two images based on statistical measures. We use these metrics as they complement each other well. MSE focuses on individual pixels while ignoring overall image structure while SSIM focuses on the structure of the image over the specific colours used. Thus we can evaluate the images on both fronts. Table 3.2 gives a visual comparison of the baseline models and our Pixel VQ-VAE. All images were randomly chosen except for the first column (Pokémon) which was used as a validation instance during development. Table 3.3 compares the performance of the different models in terms of our evaluation metrics. We also include comparisons of the best and worst reconstructions of each model according to each loss function in Appendix C.

We first consider the performance of the models on the Pokémon dataset (first two rows). Right off the bat, we see that all VQ-VAEs surpass the VAE which is blurry and lacks detail. While both the VQ-VAEs and our Pixel VQ-VAEs do well, our Pixel VQ-VAE offers better detailing in the precision of the colour palette. While slightly noticeable in the case of the LowRes models, it is more evident when comparing the VQ-VAE MedRes and the Pixel VQ-VAE MedRes. Additionally,

Model	Pokémon	Pokémon	Sprites	Sprites
Original				
VAE				
VQ-VAE LowRes				
Pixel VQ-VAE LowRes				
VQ-VAE MedRes				
Pixel VQ-VAE MedRes				
Pixel VQ-VAE HiRes				

Table 3.2: Test set reconstructions of the Pixel VQ-VAE and the baselines. The VQ-VAE HiRes is not included as it cannot exist without the enhancements introduced in the Pixel VQ-VAE. The first two columns are from the Pokémon dataset, while the remaining are from the Sprites dataset.

Dataset	Pokémon		Sprites	
Model	**MSE**	**SSIM**	**MSE**	**SSIM**
VAE	0.01815	0.64272	0.00192	0.65942
VQ-VAE LowRes	0.01076	0.60198	0.00685	0.66051
Pixel VQ-VAE LowRes	0.01040	0.78669	0.00175	0.76395
VQ-VAE MedRes	0.00584	0.63973	0.00647	0.77871
Pixel VQ-VAE MedRes	0.00421	**0.91210**	0.00224	**0.83261**
Pixel VQ-VAE HiRes	**0.00070**	0.82967	**0.00070**	0.79415

Table 3.3: Test set reconstruction metrics. MSE: Lower is better. SSIM: Higher is better. The VQ-VAE HiRes is not included as it cannot exist without the enhancements introduced in the Pixel VQ-VAE.

note the presence of a gray-ish background learned by the VQ-VAE that does not show up in the Pixel VQ-VAE. This gray background is due to the training data having images with both a white and a black background which the Pixel VQ-VAE has, for the most part, learned to ignore. The Pixel VQ-VAE HiRes visually offers the best performance, being almost indistinguishable from the original. The metrics from Table 3.3 also reflect these observations.

Now we consider the results on the Sprites dataset (last two rows). We can immediately see that the performance of the VAE on this dataset is significantly better. Although there are some visible artifacts in the image, it visually looks better than the VQ-VAE LowRes. However, like before, our Pixel VQ-VAE surpasses all the baselines models. We see that in both the LowRes and MedRes models, the Pixel VQ-VAE has less blocky textures and has learned a sharper set of encodings. However, we do see that the difference between the models is not as large as in the Pokémon dataset. This is because, as discussed previously, the Sprites dataset is highly consistent in nature, making it easier for models to better represent it, though we note that our Pixel VQ-VAE still outperforms the other models. Like the Pokémon dataset, the metrics agree with these observations once more.

Now considering the metrics from Table 3.3 in general, we see that in all cases our Pixel VQ-VAE beats the baseline models, especially in terms of SSIM. Comparing our three Pixel VQ-VAEs, we see that the MSE is better for models with lower values of M. However, the MedRes model has a higher SSIM score than the HiRes version. This is because the HiRes model has a 1:1 encoding-pixel correspondence, meaning that the model focuses heavily on individual pixels, and not overall structure. When taken in consideration with the relatively low value of K in proportion to the possible range of values, this discrepancy is understandable. We note that despite this, both models far outperform all baselines.

Since embeddings represent information about an entity, a good embedding generally exhibits characteristics of this information in the latent space. This information can be helpful in downstream tasks [24]. However, for a VQ-VAE, the latent space cannot be analyzed in the same manner. This

Dataset	Model	Pokémon		Sprites	
Model	**Size**	**MSE**	**SSIM**	**MSE**	**SSIM**
Base VQ-VAE	LowRes	0.01076	0.60198	0.00685	0.66051
Adapter VQ-VAE	LowRes	0.01049	0.77879	**0.00540**	0.75536
PixelSight VQ-VAE	LowRes	0.01079	**0.79586**	0.00704	0.71360
Pixel VQ-VAE	LowRes	**0.01040**	0.78669	0.00647	**0.77871**
Base VQ-VAE	MedRes	0.00584	0.63973	0.00175	0.76395
Adapter VQ-VAE	MedRes	0.00489	0.66650	0.00252	0.80916
PixelSight VQ-VAE	MedRes	0.00504	0.78298	**0.00160**	0.78589
Pixel VQ-VAE	MedRes	**0.00421**	**0.91210**	0.00224	**0.83261**
PixelSight VQ-VAE	HiRes	0.00133	0.73070	0.00087	**0.81525**
Pixel VQ-VAE	HiRes	**0.00070**	**0.82967**	**0.00070**	0.79415

Table 3.4: Ablation study comparing test set reconstruction metrics. MSE: Lower is better. SSIM: Higher is better. For HiRes models, the base VQ-VAE and the Adapter VQ-VAE cannot exist without further enhancements to the model.

is due to the fact that the learned embeddings correspond to individual patches of an image and not the image as a whole. We explore this in further detail in Appendix D. However, this does not mean that our Pixel VQ-VAE's embeddings are without use. Later in this work we demonstrate the usefulness of these embeddings in two different downstream tasks.

3.3.4 Ablation Study

We next demonstrate an ablation study to verify the effectiveness of the enhancements used in our model. We compare our model to the baseline VQ-VAE as well as versions that use only one of our two enhancements. We denote these as "Base" referring to the baseline, "Adapter" referring to the use of the Adapter layer alone and "PixelSight" referring to the use of the PixelSight block. We run this comparison for all versions of the Pixel VQ-VAE (LowRes, MedRes, HiRes). As described earlier, a HiRes version of the baseline VQ-VAE cannot exist without our enhancements. Similarly, the adapter requires at least one other layer in the model, thus a HiRes Adapter VQ-VAE does not exist either. Table 3.4 compares model performance on MSE and SSIM across the three different models types (LowRes, MedRes, HiRes) on both datasets. In the Sprites dataset where all images have a very consistent structure, all the enhanced models do better than the base model, particularly on SSIM. However, within the enhanced models there is some variation, though the Pixel VQ-VAE is generally best or second-best. This is due to the highly consistent structure of the Sprites dataset, which limits the efficacy of the enhancements. Thus all the enhanced versions are close to each other in terms of raw values. We note that this is not observed in a more varied dataset like Pokémon. In the Pokémon dataset, the enhanced VQ-VAEs always do better than the baseline VQ-VAE. Further, in nearly every case the Pixel VQ-VAE is the clear winner. The sole

exception is the LowRes model where all three enhanced models exhibit very similar values. This can be attributed to the LowRes models having a large encoding-pixel correspondence (M=16). This leads to each individual encoding corresponding to a larger portion of the final image, thus leading to less control over the finer aspects. In the more controlled MedRes and HiRes cases, the Pixel VQ-VAE significantly outperforms the other models.

3.4 Image Generation

Image generation is perhaps the most common use for learned representations. We thus evaluate our Pixel VQ-VAE in terms of its image representation capabilities. For our baselines, we use two common approaches to image generation, namely a VAE and a DCGAN [20]. We use the same VAE from above and a standard DCGAN. For the Pokémon dataset, we increased the number of parameters in the generator to have a similar number of parameters to our final model. We do not do this on the Sprites dataset due to poor convergence. The DCGAN architecture we use is similar to unpublished works that have explored Pokémon generation [33]. Both the VAEs and GANs were trained on the same dataset as our VQ-VAE. As discussed earlier, VQ-VAEs are like traditional word embedding models in that the model that learns the embeddings (VQ-VAE) is distinct from the model that uses the embeddings for downstream tasks. We use the PixelCNN [88], a common downstream model used with VQ-VAEs [90]. For the simpler Sprites dataset, this model sufficed. However, the Pokémon dataset is far more complex and required additional effort. Specifically, we use a Conditional PixelCNN [89] which has several optimizations for better generation, including conditional image generation. For both datasets, the generative model was trained on the embeddings generated by the Pixel VQ-VAE. In the case of Pokémon, we conditioned the model on both the Pokémon's shape attribute and its two type attributes. These attributes were selected as they generally affect a Pokémon's silhouette and colours.

The PixelCNN was trained over 25 epochs with the Adam optimizer with a learning rate of 0.0001 and a batch size of 32. It uses 7 gated convolutional layers, each with 3x3 filters (256 for Pokémon, 128 for Sprites). We trained one version of this model for each of our Pixel VQ-VAEs (LowRes, MedRes, HiRes). We repeat the same procedure with a set of VQ-VAEs without our enhancements. This is repeated for the Pokémon dataset with the Conditional PixelCNN. All the models have the same set of hyperparameters with the architecture being modified slightly to work with the specific VQVAE. Specifically, we follow an approach similar to the original VQ-VAE paper [90], setting the number of discrete classes as the number of VQ-VAE embeddings K. For complete fairness, we trained a PixelCNN model directly on our training dataset (that is, without using VQ-VAE embeddings). However, the model failed to converge and generated only blank images. We suspect that this is due to the much larger output space in this scenario. For any given pixel there are 256^3 or over 16 million possibilities. In contrast, even the largest Pixel VQ-VAE has only K

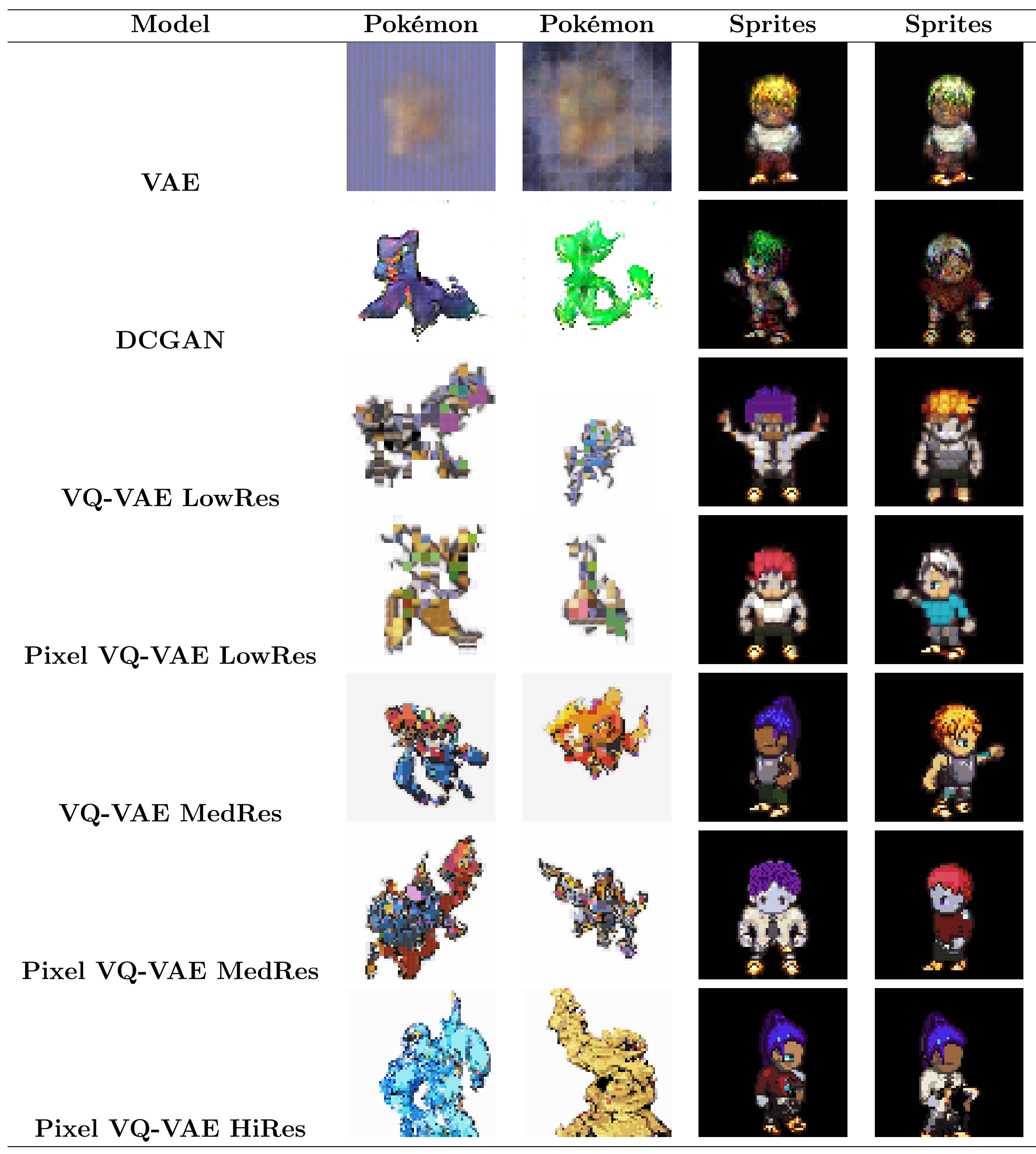

Table 3.5: Hand-picked generated samples for each model. All VQ-VAE results are generated from a PixelCNN model trained on the VQ-VAE embeddings.

Model	Pokémon	Sprites
VAE	322.746	196.877
GAN	101.759	210.485
VQ-VAE LowRes	167.464	140.565
Pixel VQ-VAE LowRes	154.735	126.869
VQ-VAE MedRes	118.253	106.698
Pixel VQ-VAE MedRes	124.302	111.184
Pixel VQ-VAE HiRes	**98.575**	**102.801**

Table 3.6: FID of generated images. Lower is better.

Model	Pokémon	Sprites
VAE	7,680,774	7,680,774
GAN	15,424,000	6,342,272
VQ-VAE LowRes	20,998	50,182
Pixel VQ-VAE LowRes	1,413,065	1,363,881
VQ-VAE MedRes	17,222	17,222
Pixel VQ-VAE MedRes	1,102,121	1,102,121
Pixel VQ-VAE HiRes	54,313	54,313
PixelCNNs (for all VQ-VAEs)	13,444,864	3,367,040

Table 3.7: Number of parameters of each model.

(256 in our case) possibilities for any given pixel.

Table 3.6 compares the models in terms of the Fréchet Inception Distance (FID) [16]. FID is a metric that tells us how close a generated sample is to the training data distribution. This metric was created to compare photo-realistic images and we use it only in the absence of a better metric. We note that the FID only tells us the similarity of a given image to the training distribution, not if the images themselves are appealing to a human. That aspect is highly subjective and would require a comprehensive user study. However, our focus at this time is to study the effectiveness of the representation as a whole and not just a single component. We thus leave a human subject study for future work. In each case, we generate 10,000 images and compute the FID score between the model and the training data. Table 3.5 depicts selected outputs from each model while Table 3.7 depicts the number of parameters used by each model.

Let us first consider the Pokémon dataset. Our first baseline, the VAE, suffers from extreme blurriness with no recognizable images whatsoever, which its FID score reflects. Although the DCGAN achieves an FID score close to our best model (Pixel VQ-VAE HiRes), we see a clear distinction in the generated images. While the GAN does generate interesting shapes with a good spectrum of colours, the generated images no longer retain the pixel art style that we desire. On the other hand, our Pixel VQ-VAE generates interesting shapes and employs colour gradients while still retaining the pixel art style. In the case of the Sprites dataset, both the VAE and the GAN generate

body shapes in the right style. This discrepancy is due to the highly consistent nature of the Sprites dataset. However, we see that there are a lot of artifacts in the generated images and unlike the VQ-VAEs, features such as the eyes are not readily visible. The VAE seems to do slightly better than the GAN, having fewer artifacts. This is reflected in the FID scores as well. As discussed in Section 3.3.1, the Pixel VQ-VAE HiRes model has the least parameters due to the low value of L. While smaller in comparison to the GAN, it must be considered in tandem with the respective PixelCNN which is responsible for generation. We reiterate that a HiRes VQ-VAE cannot exist without our enhancements. Our Pixel VQ-VAEs have far more parameters than the baselines for the same reason. Although we tried tuning the hyperparameters of the baselines in order to match the number of parameters, we found that this led to very poor convergence. Comparing the Pixel VQ-VAE and the baseline VQ-VAE across both datasets, we see that our Pixel VQ-VAE works better in the LowRes case while in the MedRes case the baseline beats it out by a small margin. Our Pixel VQ-VAE LowRes and MedRes models are larger than the baselines due to the addition of the PixelSight block and Adapter layers. However, taking into consideration both the quality of the embeddings and the FID scores, our Pixel VQ-VAEs are still the best performing models overall. Now considering the generations themselves, using the Sprites dataset leads to generations that are quite similar to our existing dataset. We believe that this is due to the limited output space of the Sprites dataset which may deter more novel attempts at generation. It is also evident that none of the generations appear representative of new Pokémon. Pokémon, due to their high variance, are difficult to generate. Despite our enhancements to the VQ-VAE, a PixelCNN is insufficient on its own to completely solve this task. We provide further examples of randomly generated outputs in Appendix F.

3.5 Image Transformation

We now consider the task of image transformation. While there are several different subtasks in this area such as recolouring/palette swapping, pose modification, creating animation frames etc., we specifically consider the recolouring/palette swapping task due to its popularity in prior work [43, 95, 62, 13]. This task involves modifying colour palettes of images while retaining their shape and structure. In a recolouring task, this can be as simple as just changing some of the colours being used. In a palette swap, we specifically swap the colours between two images. Both of these techniques can be used to generate different variations of a particular character, perhaps with different attributes associated with them. For instance, we could create a water-type version of a fire-type Pokémon by changing the palette to use blues instead of oranges. Using a palette from an existing character is appealing as it allows us to try out several colour palettes without manually handcrafting them.

We specifically demonstrate the application of the learned embeddings in this palette swap

Source Image	Target Image	Hand-Authored	Pixel VQ-VAE

Table 3.8: Example results for the image recolouring task.

scenario. We follow a relatively simple process to generate a colour swapped image, given a source and a target image. For each image, we first compile a map of the encodings present in the image in descending order of the number of their occurrences. That is, we take the I^2/M encodings associated with each image and compile a frequency map. We then simply replace the encodings of the target image with that of the source. Specifically, the source image's most frequently used encoding would replace the target image's most frequently used embedding. We note that while this is a simple method of performing this palette swap, it is still reasonably effective. We further note that this same methodology can be used to recolour images to custom colours by simply mapping the encodings of the Pokémon to the encodings corresponding to the desired colours. This has two requirements - the user must know the palette they wish to use and the palette must exist within our K learned encodings. However, this restriction actually aligns well with the limited colour palette of pixel art.

We note that this is a highly subjective task with no real metrics apart from a visual evaluation. Different artists may use the same colour palette but utilize the colours in different manners. Table 3.8 shows some examples of a hand-crafted [58] colour swap and one generated by our HiRes Pixel VQ-VAE. The hand-authored Pokémon sprites are from Poke-Colors [58]. We could not find any

recolours on the Sprites dataset at this point and so the author of this thesis created the recolours manually. In the case of Pokémon (first two rows) we see that our model actually produces images quite close to the human-authored version. In the case of the Sprites dataset (last two rows), the first generated recolour is significantly different from the hand-authored version. However, we believe that while it is different, it is still an interesting generation that could realistically still be used. The second recolour on the other hand is much closer to the hand-authored version. We note that despite not performing any special training procedure for this task, our model is still capable of arriving at a reasonable approximation of a human-authored image without ever seeing one. While we leave the implementation to future work, we believe that reaching near-human results on this task would be achievable.

3.6 Conclusion

In this chapter, we introduced the Pixel VQ-VAE, a VQ-VAE model enhanced specifically for representing pixel art. We trained Pixel VQ-VAE models of three different sizes on two distinct datasets. We then evaluated the quality of the learned representations in terms of two metrics and additionally performed an ablation study. We then demonstrated the applications of these learned embeddings in two downstream tasks - image generation and image transformation.

Chapter 4

Evaluating Game Balance Through Meta Discovery

In this chapter we explore in detail the second major contribution of this thesis - a framework for evaluating game balance through meta discovery. We start with a discussion on the importance of a balanced metagame. We then describe the primary motivation for meta discovery in the context of this thesis, the generation of balanced characters in competitive video games, before detailing our approach to meta discovery.

As discussed in Section 2.1.6, the metagame is a collection of knowledge that goes beyond the rules of the game. In competitive games like Pokémon or League of Legends, it refers to a collection of the most popular characters. Since players naturally want to win games, the meta characters also tend to be the strongest characters in the game. In many games, the metagame, when left alone, is self-balancing to some extent. As characters grow stronger and hence more popular, other characters are used more often to counter them. As those characters grow in popularity, different characters that counter them may in turn grow in popularity. Thus the metagame constantly shifts over time. This may offer more engagement from a player's perspective. However, there are also avenues of frustration as the self-balancing nature of the meta is not perfect. For instance, characters that are very weak may not see much usage at all. On the other hand, some characters may be far too strong and never go out of the meta, thus causing strategies to revolve around handling that character. This over-centralization is not ideal and may lead to stagnation within the meta. This is undesirable as players might find the game unfun or boring. Thus in order to adjust power levels of characters, prevent stagnation, and to encourage diversity in character selection, external influences may change the metagame. These could be in the form of regularly released patches that adjust character power levels and/or introduce new gameplay content, or community-voted bans on characters.

Developers need to be careful with changes they make to a game's balance. There are several instances of developer changes causing major shifts in the metagame such as the infamous Juggernaut patch of League of Legends [55] and the creation of the Anything Goes tier in competitive Pokémon [93]. In both cases, an over-powered character(s) had too much influence over the meta. Another factor to consider is the popularity of characters. A small number of players winning a lot with a certain character can be attributed to them being very good at using that character. However, a large number of players winning in the majority of matches with a character might imply undue strength. Thus when balancing games, many developers consider the combination of a character's winrate and their pickrate [66]. The pickrate is simply the percentage of matches a character is picked in while the winrate is the percentage of matches that the character actually wins. We note that while this information offers value in identifying areas that may need changes, they do not tell us what these changes are. In practice, developers use their expertise and domain knowledge to make balance changes. The meta merely serves as a pool of knowledge from which the target of these changes can be identified.

Most live service video game companies have an alpha and/or beta environment where changes are tested and iterated upon before public release. While the combination of internal metrics and playtester feedback works to an extent, as the Juggernaut patch showed, it is not perfect. One reason for this is that statistics like winrates and pickrates in the playtester environment may not be reflective of the live version of the game. This is because the alpha or beta environments are much smaller in playerbase than the live version of the game. We thus propose a balance evaluation framework through meta discovery in a competitive metagame. Specifically, we discover a metagame by simulating a large number of matches while also considering factors such as team-building. Such a framework could allow developers to better understand the impact that balance changes would have on the competitive metagame. In order to build this framework, we require several components.

First, we need a battle agent to play matches. A heuristic agent that uses hand-crafted rules would be sufficient for this task. However, significant work exists on utilizing reinforcement learning to train such agents (as described in Section 2.4.1). We note that our goal here is not to create a state-of-the-art or expert-level agent. Rather, we simply want a relatively fast agent capable of approximating an average human player. We thus verify the performance of our agents by evaluating them on matches against humans.

Next, we need a team-building component. Since the majority of competitive games utilize teams [63, 22, 18], team-building is a crucial aspect of any solution to this problem. Specifically, this component must generate new teams for the agents to use. Since teams are generally composed of characters that are in the meta or characters that do well against characters in the meta, team generation cannot be completely randomized. Rather, it must take into account the current meta to generate a team. In addition, since the meta changes over time, this component must also adapt

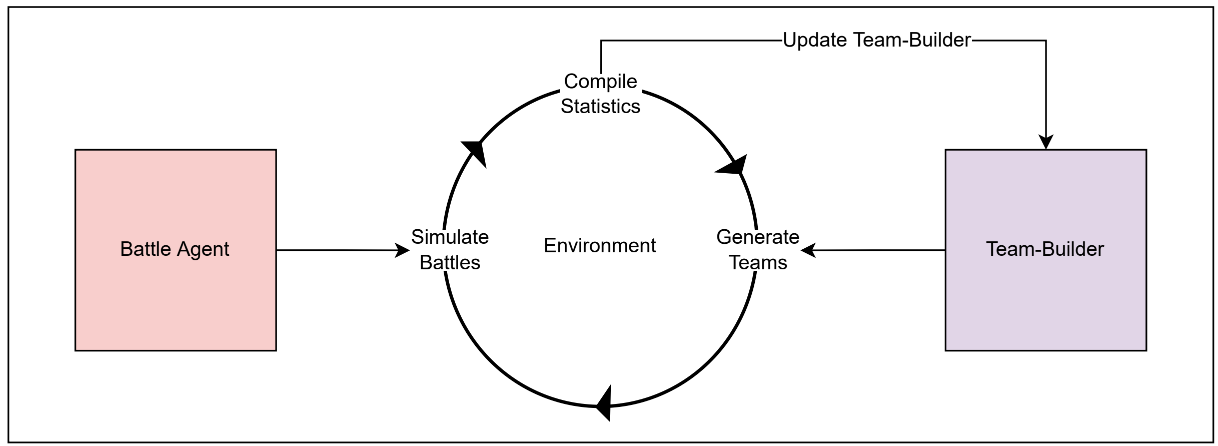

Figure 4.1: The components of our meta discovery framework.

to such changes.

Finally, we need to create the environment in which meta discovery occurs. This environment houses the battle agent and team-building components and is responsible for simulating battles. It also tracks statistics such as pickrates and winrates. Once the simulation is complete, these statistics can be extracted and analyzed to form an estimation of the new metagame.

Figure 4.1 depicts the relationship between these components visually.

In this thesis, we consider a common scenario in games like Pokémon and League of Legends [76, 77, 19]. Specifically, a character in a competitive online game has been banned, removed or otherwise disabled, thus preventing its use in matches. How does the metagame change? In other words, given the current metagame and a change to be applied, can the new metagame be approximated? We term this task as **ABC-Meta**, or Analyzing Balance Changes on the Metagame. Additionally, while it is not the focus of our research, we also explore a related but distinct task that we term Blank Slate Discovery of Metagames or **BSD-Meta**. It refers to the discovery of a metagame from scratch under the assumption that there is no prior metagame in existence. This scenario is common in games like Pokémon, where a new set of battle formats are created for every generation of games [76, 77].

We consider this additional scenario primarily due to its similarity in technical implementation to the ABC-Meta task. Both tasks require all three components of our framework. However, they differ in their implementation. Specifically, in the case of the ABC-Meta task, we have access to the current metagame to base our initial team-building algorithms on. In the BSD-Meta task, this information is not used as we start from scratch. Thus, the BSD-Meta task is a special case of the ABC-Meta task where we do not have prior information on the metagame.

4.1 Application Domain

We implement and evaluate our system on the Pokémon video game series, specifically on Pokémon Showdown. Pokémon Showdown is an online Pokémon battle simulator with millions of monthly matches [74] and as we discussed in 2.1.7, it has an extensive, complex metagame. We use Pokémon over other games because it is more technically feasible due to an easily accessible API. In addition, the matches are generally much faster. This is important as it allows us to simulate a large number of matches in a relatively short amount of time, thus allowing us to experiment with different techniques for meta discovery. Furthermore, due to our work in the same video game series in Chapter 3, continuing to work on it seemed a natural choice. Other games such as League of Legends and Dota 2 are more technically complex and do not have as easily accessible APIs. We elect to use the Pokémon Showdown metagame in particular over other options such as the VGC format [64] due to its popularity.

Although much of our work on reinforcement learning agents is based on the work of Huang and Lee [21], we do not use their codebase due to compatibility issues. Specifically, the code was developed several years ago and there have been several changes to Pokémon Showdown since then. We instead use the poke-env [68] API to interface with Pokémon Showdown through Python. This API builds off of the OpenAI Gym environment which was built for training reinforcement learning agents. Thus it lets us train reinforcement learning agents and also allows us to easily access all aspects of a battle. This in turn allows us to direct our attention to the problem of meta discovery rather than setting up infrastructure. In addition, it also supports the use of non-reinforcement learning, heuristic agents via a "General API“, which does not use the Gym environment and is hence faster. We note that this API is in active development and specify that future versions of this API might break our codebase. We detail specific API versions in our codebase but use Python for all aspects of our code. We release all code publicly[1]. We use a local version of Pokémon Showdown (which is open source) to simulate our battles but connect to the online version for evaluating our agents.

We note that several aspects of the following sections require some knowledge of the Pokémon video game series and thus recommend readers to refer to Section 2.1.4 for more information on these aspects.

4.2 Battle Agents

We preface this section by restating that our objective here is not to create a state of the art agent. Although this is an interesting research problem on its own (Section 2.4.1), our goal is to only have

[1] https://github.com/akashsara/meta-discovery

an agent capable of approximating an average human at the game. This is because they have the most influence over the metagame, given that the meta is a measure of popularity and the majority of players in competitive online games are of average skill. We additionally note that the metagame itself may differ based on a player's matchmaking rating, but leave an exploration of this for future work. We thus train and evaluate several different agents. Our final selection, however, is largely dependent on the speed of the agent. This is because our goal is to simulate a large number of matches, thus even small differences in speed can have large impacts on running time.

We use the Random Battle format, specifically "gen8randombattle", for training agents. This format comprises of two randomly generated teams of six Pokémon in a singles battle format. That is, each player sends out one Pokémon at a time and they battle until a player either forfeits or has no remaining Pokémon. We describe this system in more detail in Section 2.1.5. We note that the random battle format is different from the standard Smogon tiers that we analyze for meta discovery. We train our agents on this format for two reasons. First, this format does not require us to provide teams, instead pseudo-randomly generating them based on hand-crafted rules. In contrast, if we used the standard Smogon tiers, we would have to compile teams ourselves or create our own team-building algorithm. Second, the pseudo-random nature of the teams allows for more generalization in the reinforcement learning model since it offers a wider variety of teams to the model. We do note that some attributes of the generated team are dependent on the current meta. Thus we cannot use the team generation system used by gen8randombattle for our own meta-discovery process as it would bias our system.

4.2.1 Action Space

All our agents have a discrete action space of size 22. The first four actions correspond to each of the four moves that a Pokémon can have. The next 12 actions correspond to Z-Move variations, Mega Evolution variations, and Dynamax variations of the same 4 moves. Each of these are once-per battle gameplay mechanics and are associated with the first four actions. The final six actions correspond to the ability to swap out the current Pokémon for the Pokémon in the corresponding position within the team. This does mean that at all times, at least one of these 6 actions will be illegal as one Pokémon is always on the field. Similarly, it is not possible to switch to a fainted Pokémon nor is it possible to use a move if the current Pokémon has fainted. We note that in the case of all our agents, we perform action masking, thus preventing such illegal actions from being taken. At every step, actions are passed to the environment as an integer corresponding to one of the 22 possible actions.

4.2.2 State Representation

We create three different state representations - the Simple state, the Full state and the Flattened state. We use these different state representations for two reasons. First, Pokémon battles are highly complex as discussed in Section 2.1.4. Thus, the learning problem might be more difficult in more complex states. So we use states of varying complexity such that some agents may learn to do well. We reiterate that our goal is to have an agent capable of approximating an average human, not a state-of-the-art agent. Second, there is an overhead cost in time for extracting states from the environment. This is important as we wish to simulate a large number of matches.

The Simple state is, as the name suggests, a simple representation of the battle. It is based on the default state representation available in the poke-env API. It is of size 10, with the first 4 elements corresponding to the base power of the Pokémon's 4 moves. We normalize this base power by dividing it by 100 to facilitate learning. The next 4 elements correspond to the type effectiveness of each move against the opponent's currently active Pokémon. The final two elements offer an indication of how many Pokémon are still alive on each team. We normalize this value by dividing it by 6 as a team can have a maximum of 6 Pokémon.

Our Full state is almost entirely based off of the state representation used in Huang and Lee's work [21]. At a high level, this involves learning a latent representation of each of the six Pokémon on a team and combining them with team-specific battle information. This is repeated for the opposing team and then the two embeddings are combined with global battle information before being passed through a model. We follow this approach as it is the state-of-the-art work in this area. The only change we make is that we do not use the Weather Time Left and Weather Min Time Left attributes as they are not available through the poke-env API. The Flattened state uses the exact same representation as the Full state but flattens out the hierarchical structure into a single vector of size 2789. This offers a middle-ground between the state representations, taking into account all the information from Huang and Lee's work but allowing for a faster training time.

4.2.3 Reward Function

Our reward function is a weighted sum of the game state. Specifically, we weigh fainted Pokémon and overall HP fractions at 0.0125 and 0.1 respectively and give out a reward of 1 and -1 for victories and defeats respectively. These are default parameters recommended by the author of the poke-env API.

4.2.4 Baselines

We employ three baseline agents for comparison purposes. The first is the Random agent that takes uniformly random actions. The next agent is the Max Damage agent which attempts to do the most damage to the opponent based on the base power of the moves that are available. This does not take into account domain knowledge like type effectiveness or STAB. Our final baseline, the Heuristic agent, uses hand-crafted rules and domain knowledge. This agent was developed by the authors of poke-env. It is capable of estimating stats of opponents, determining the use of specific types of moves and the use of one-time battle mechanics, analyzing type effectiveness and STAB, and determining when to switch out to a different Pokémon.

4.2.5 Reinforcement Learning Agents

We investigate two different reinforcement learning algorithms for our task - a Double Deep Q-Network (DDQN) [92] and Proximal Policy Optimization (PPO) [70]. We utilize DDQN due to the popularity of its variants across the literature [40, 28, 72]. The DDQN uses a target network and experience replay like in [92]. We additionally use the PPO algorithm with Generalized Advantage Estimation due to its success in prior work [21]. For both algorithms, we trained agents using both the Simple and Full state representations against each of the three baselines. In all cases, our preliminary results indicated that the agents trained against the random baseline performed best. We thus train all future agents against this baseline. In addition, we also trained each agent via self-play. In the case of the PPO algorithm, we also trained agents on the Flattened state representation. We do not train the DDQN agents on this state representation as our preliminary results indicated that the PPO agents generally performed better.

To summarize, we train two reinforcement learning algorithms (DDQN, PPO). The DDQN agents are trained on the Simple and Full states against two training partners (random baseline and self-play). The PPO agents are trained on each of the three states (Simple, Full, Flattened) against the two training partners. In total, we investigate 10 reinforcement learning agents related to our goal of achieving an agent capable of approximating an average human.

4.2.6 Model Architecture

Our Simple agents use a model consisting of two feedforward layers with 128, and 64 nodes respectively. We use a standard ELU [5] activation function following each layer. Our Flattened state agents uses a similar architecture but instead has four layers with 512, 256, 128, and 64 nodes respectively. Our Full state agents follow the same architecture as Huang and Lee's work [21].

The models vary slightly based on the algorithm they are used for. In the case of the DDQN,

there is a single output layer which is mapped to the 22 actions in the action space. For the PPO agents, there are two output layers corresponding to the actor-head and the critic-head. The former is mapped to the 22 actions in the action space while the latter gives a single value.

4.2.7 Hyperparameters

For our DDQN models, we use a Tau (frequency at which the target model is updated) value of 0.001, a train interval of 1, 1000 warmup steps, a memory size of $10,000$, and a discount factor of 0.95. We use a batch size of 128, an embedding dimension of 128 and an exponentially decaying Epsilon-Greedy policy which decays from 0.95 to 0.05 with a decay factor of $10,000$. The models are trained for $1,000,000$ steps using an Adam optimizer with learning rate 0.00025. The beta parameter for the L1 loss function is set to 0.01. We performed a hyperparameter search to arrive at these values. This was done over $10,000$ steps and was optimized over the performance against the baseline agents.

For our PPO models, we use a discount factor of 0.95, a Generalized Advantage Estimation Lambda value of 0.9, a surrogate clipping parameter of 0.1, a value function clipping parameter of 0.1, loss constants of 0.5 and 0.002. All these hyperparameters are based off Huang and Lee's work. We perform $10,000$ steps per rollout, train for 10 epochs after each rollout and train for a total of $1,024,000$ steps. We use a batch size of 128, an embedding dimension of 128 and an Adam optimizer with a learning rate of $2e-4$.

In all cases, we use a fixed random seed of 42.

4.2.8 Evaluation: Performance

For our first set of evaluations about this approach we look at our agents performance against other agents and human players. We would ideally like our agents to perform at the level of average human players, which means consistently beating the simpler baselines and achieving a roughly 50% win rate against humans. While training the agent, we first simulate 1000 battles each against our three baseline agents. We use this to given an initial estimate for the quality of an agent. In order to have a more comprehensive evaluation in line with several prior works [57, 21], we also play 100 battles against real humans on the Pokémon Showdown random battle ladder. These battles are done on new accounts for each agent so that each agent can be treated independently. We examine these battles on three metrics - ELO, Glicko-1, and GXE as discussed in Section 2.1.8. Like [21], we primarily look at the Glicko-1 rating. We also interpret it using GXE which tells us the probability of winning a battle against an opponent randomly selected from the ladder. We would want this number to be close to 50% to approximate an average player unlike a state-of-the-art player which would want it to be as high as possible. We note that our evaluation on Pokémon Showdown is a

State	Algorithm	Training Partner	Battles	Random	Max Damage	Heuristic
-	Random	-	1000	529	128	8
-	Max	-	1000	883	497	38
-	**Heuristic**	-	**1000**	**993**	**957**	**509**
Simple	DDQN	Random	1000	961	823	267
Simple	DDQN	Self-Play	1000	925	752	209
Simple	*PPO*	*Random*	*1000*	*977*	*865*	*265*
Simple	PPO	Self-Play	1000	978	822	214
Full State	DDQN	Random	1000	406	134	6
Full State	DDQN	Self-Play	1000	536	145	13
Full State	PPO	Random	1000	799	323	41
Full State	PPO	Self-Play	1000	799	323	41
Flattened	PPO	Random	1000	759	360	43
Flattened	PPO	Self-Play	1000	629	177	17

Table 4.1: Results of evaluating our agents against the three baseline agents. Columns indicate the number of battles won against a particular baseline. Higher is better for all metrics. Best agent is in **bold**, 2nd best is in *italics*.

time-consuming process and we thus only evaluate our best models in each category.

Overall, we test two different reinforcement learning algorithms (DDQN, PPO) across three distinct state representations (Simple, Full, Flattened) and two different training partners (Random baseline, self-play).We evaluate the 10 agents over six metrics. The first three compare an agent's performance against the three baselines. The next three metrics compare are the metrics from the human evaluation described above.

4.2.9 Results: Performance

Table 4.1 depicts the results of our final set of agents after 1000 battles against the baseline agents. We immediately see that while all the agents do reasonably well against the Random baseline, the larger state representations (Full State, Flattened) are significantly worse than the other agents and are only slightly better than the random baseline. Given the better performance from the simple state agents and the complexity of the state representation, we believe that this is a factor of the lack of training time. We train for 1 million steps which is approximately equivalent to 50,000 battles. For comparison, one of the best published agents [21] was trained for $3,840,000$ battles over 6 days, which is two orders of magnitude higher. Given our goal of using a reasonably effective agent, we did not experiment further on the larger state agents.

We now turn our attention to the Simple state agents. These are significantly better than the larger state agents. However, while they offer excellent performance against the random baseline

State	Algorithm	Training Partner	Played	Won	Lost	ELO	GXE	Glicko-1
-	Random	-	100	14	86	1000	15.7%	1182 ± 39
-	Max	-	100	24	76	1050	24.0%	1282 ± 36
-	**Heuristic**	-	**100**	**51**	**49**	**1235**	**49.4%**	**1495 ± 35**
Simple	DDQN	Random	100	37	63	1027	34.4%	1378 ± 36
Simple	DDQN	Self-Play	100	38	62	1031	33.2%	1368 ± 36
Simple	PPO	Random	100	42	58	1049	36.4%	1395 ± 35
Simple	*PPO*	*Self-Play*	*100*	*46*	*54*	*1124*	*42.4%*	*1442 ± 35*
Full State	DDQN	Random	100	15	85	1040	18.2%	1216 ± 36
Full State	DDQN	Self-Play	100	22	78	1000	21.6%	1256 ± 36
Full State	PPO	Random	100	13	87	1000	16.5%	1194 ± 36
Full State	PPO	Self-Play	100	21	79	1000	20.5%	1244 ± 36
Flattened	PPO	Random	100	23	77	1000	22.1%	1262 ± 36
Flattened	PPO	Self-Play	100	19	81	1000	20.8%	1247 ± 36

Table 4.2: Results of evaluating our agents against humans on the Pokémon Showdown random battle ladder. Higher is better for all metrics. Best agent is in **bold**, 2nd best is in *italics*.

and do reasonably well against the Max Damage baseline, they do not fare very well against the Heuristic agent. We believe that although these models would benefit from increased training time, the performance is primarily restricted by this small state representation we use. Since this representation does not consider several aspects of the battle, the performance of this agent is bound to plateau at some point. A different state representation that strikes a balance between the complexity of the larger state agents and the speed of the simple agent is a problem we leave for future work. We see that overall the best performing agent is the baseline Heuristic agent. Next, we further evaluate the same set of agents against human players on the Pokémon Showdown ladder.

Table 4.2 depicts the results of the agents after 100 battles on the Pokémon Showdown random battle ladder. As a reminder, we aim to achieve a winrate of 50%. The results are largely as expected, with the Heuristic baseline performing the best. It achieves a Glicko-1 rating of 1495 which corresponds to a 49.4% (GXE) chance of beating a random opponent on the ladder. The best reinforcement learning agent is the Simple Self-Play PPO model which is not too far behind at a Glicko-1 rating of 1442 or a 42.4% chance of beating a random opponent. The performance of all these agents are in-line with the previous table for the same reasons discussed above. Interestingly, we do note that the Simple Random-Opponent PPO did the best against our baselines while the Simple Self-Play PPO performed the best against human opponents. This may indicate that the self-play agent learned slightly better tactics than the agent trained on the random baseline.

Agent	API	Time
Full	Gym	3600
Flattened	Gym	3600
Simple	Gym	854
Full	General	2132
Flattened	General	1746
Simple	General	292
Simple-Concurrent	General	98
Heuristic-Concurrent	General	99

Table 4.3: Comparison of our reinforcement learning agents on time taken to complete 1000 random battles. All times are in seconds, averaged over 3 runs and rounded up to the nearest integer. All runs were performed on the same hardware and use a timeout of 3600 seconds.

4.2.10 Evaluation: Speed

As discussed previously, the speed of our agents is important to us as it directly affects the time required to simulate a large number of battles. For each agent, we measure the time taken to play out 1000 battles. We average these times over 3 runs on the same hardware and report the time taken in seconds, rounded up to the nearest integer. We use a maximum timer of 3600 seconds at which point we cancel the run.

We found that running the agents on the poke-env Gym API that we used for training was not very time-efficient. Hence we consider the poke-env General API, which does not use any reinforcement learning components, as an alternate option. The lack of reinforcement learning components does not pose a problem since we only want to use the trained models, that is, we don't need to train the models while performing meta discovery. This also means that the training partner and reinforcement learning algorithm do not matter in this scenario. Thus we only need to consider the three different state representations (Simple, Full, Flattened). We arbitrarily use the Self-Play PPO agents in each case. We compare these three agents on both the Gym API and the General API, for a total of six comparisons.

The implementation of the General API also supports concurrent battles. After some tuning, we found that performing 25 battles concurrently led to the biggest increase in speed. We consider this as a separate version of the agent for the purposes of this evaluation. We test this method only for the fastest agent from the six discussed above.

4.2.11 Results: Speed

Table 4.3 compares the agents discussed above. Comparing the run-time of the agents on the General API, we can see that the Simple Agent is an order of magnitude faster than both the Full

and Flattened agents. This is primarily because the state representations in the latter agents are far more complex. Extracting each individual part of the state is thus a time-consuming process. To put things into perspective, we must extract information about 6 Pokémon on a team, pass each Pokémon's information through a model to acquire a latent representation and then combine this with the battle information that is specific to that team. This is repeated for the other team as well. Then, these two embeddings are combined with global battle information before being passed through the model. Thus, while we believe that, given sufficient time and effort, the larger agents would achieve performance similar to [21], we do not explore them further. On comparing the Heuristic Agent and the Simple Agent, we note no significant difference in time taken. We believe that this is due to the similar technical overhead present in both agents for selecting an action.

4.2.12 Final Agent Selection

Our goal was to select an agent capable of approximating an average human player in order to explore our desired meta discovery tasks. Our baseline heuristic agent outperformed all other agents according to our two evaluations. We additionally also consider the Simple Self-Play PPO model as it was the best reinforcement learning agent. Since there was no significant difference in speed between the two agents, we utilize them both in our meta discovery experiments. Specifically, we use the versions where we run multiple battles concurrently.

4.3 Team-Building

As discussed in Section 2.1.5, Pokémon battles generally consist of 2 teams, each having 6 Pokémon. Since the overall meta is dependent on the Pokémon that are used in battles, the method through which we build teams will influence our estimation of the meta. Thus we need to generate teams in a manner similar to that used by human players. Prior work [63, 79, 22] has analyzed this team-building process, as discussed in Section 2.1.4.

Before we build teams however, we need to setup a Pokémon's moveset. That is, we must set all the aspects of a Pokémon such as its moves, abilities, item etc. as discussed in Section 2.1.4. As a solution, we scrape publicly available Smogon data that compiles such movesets based on usage statistics.

In general, prior work tends to represent the team-building process sequentially, with algorithms picking one member of the team at each step. We term the team that is being built as the **Current Team** and the set of available Pokémon as the **Pool**. Initially the Current Team will be empty and at each step of the team-building process, one Pokémon is added to it. In order to select a Pokémon to add to the team, several components have been considered. In the next section, we define these

components and also introduce our own.

4.3.1 Components

Prior work has considered the use of both domain knowledge (type synergies, base stats) and metagame knowledge (pickrates, winrates, popularity). Our approach takes inspiration from these works to both modify existing statistics and define new ones. All the statistics we consider have a value ranging from 0.0 to 1.0 and are computed on a per-Pokémon basis in a vectorized manner.

We first consider three statistics that use metagame knowledge. The first is the pickrate which refers to the fraction of teams in which a Pokémon has been picked. Since both teams in a battle can have the same Pokémon, the pickrate for a Pokémon is defined as the total number of picks divided by twice the number of battles.

$$\text{Pickrate(X)} = \frac{\text{Num. Picks(X)}}{2 * \text{Num. Battles}} \tag{4.1}$$

Next, the winrate refers to the fraction of battles that a Pokémon wins. If a Pokémon has never been picked, the winrate is automatically set to 0 to avoid divide-by-zero errors.

$$\text{Winrate(X)} = \frac{\text{Num. Wins(X)}}{\text{Num. Picks(X)}} \tag{4.2}$$

Finally, the popularity as defined by Rejim [63] is based on how frequently Pokémon are used together. We alter this to be a measure of how frequently a Pokémon wins when used alongside the Pokémon in the Current Team. This is used only when the Current Team has at least one Pokémon. The number of wins every Pokémon has when used with every other Pokémon is tracked in a popularity matrix. When team-building, this matrix is normalized. For each Pokémon, we obtain the normalized value associated with every Pokémon in the Current Team. These values are then averaged to arrive at the overall popularity for that Pokémon.

$$\text{Popularity(X)} = \text{Mean(PopularityMatrix[X][CurrentTeam])} \tag{4.3}$$

We now consider the three statistics that utilize domain knowledge or a combination of domain knowledge with metagame knowledge. First, we use the Base Stat Total or BST, which is the normalized base stat total of a Pokémon. This value is effectively a constant as the BST cannot be changed by players.

Next, we define two values based on type effectiveness in Pokémon, the Meta Type Value and the Type Value. The former prioritizes Pokémon with types that are strong against the current

meta while the latter prioritizes Pokémon with types that are strong against the types that can beat the Current Team. That is, the Type Score is higher for Pokémon that counter the counters to the Current Team. To obtain these values, we first pre-compute a matrix detailing the effectiveness of every type against every type. This type chart is assigned values of -2, -1, 0, and 1 for type immunities, resistances, neutralities and weaknesses respectively. Thus, given a type we can extract a type vector that details the effectiveness of all types against it. We extend this to a group of types by summing up the type vectors for the entire group. We then normalize this vector to obtain a value for each type and assign these values to Pokémon based on their type. In cases where Pokémon have two types, we take the average of these values.

To calculate the Meta Type Value, we compile the types of all the Pokémon in the current meta and calculate the value using the process described above. To calculate the Type Value, we first compile the types of all Pokémon in the Current Team. We then compile the group of types that are strong against the types in the current team and calculate the group of types that can beat that group. We pass this final group into the process described above. We represent this as an equation below:

$$\text{Meta Type Value} = \text{TypeEffectivenessCalculator}(\text{MetaTypes}) \tag{4.4}$$

$$\text{TeamWeaknesses} = \text{ExtractTypes}(\text{TypeEffectivenessCalculator}(\text{CurrentTeamTypes})) \tag{4.5}$$

$$\text{Type Value} = \text{TypeEffectivenessCalculator}(\text{TeamWeaknesses}) \tag{4.6}$$

Where TypeEffectivenessCalculator takes in a group of Pokémon types and returns a type vector and ExtractTypes extracts the most effective types from a type vector.

4.3.2 Algorithms

Our team-building algorithm differs between the ABC-Meta task and the BSD-Meta task due to differences in the availability of statistics. In both cases, we compute a score for every Pokémon in the Pool. We then use this value to compute an approximation of the probability of a player picking a Pokémon. We then sample from this to pick a Pokémon to add to the Current Team. The score function differs based on the task while our probability function is defined below:

$$\text{Probability} = \frac{\text{Score}}{\text{Sum(Score)}} \tag{4.7}$$

We note that Pokémon that are banned are included in the initial calculation of the score function, though the value is zeroed out before we compute the probability.

Since the ABC-Meta task wishes to approximate the true meta, we restrict ourselves only to those components which a human player would have access to. Specifically, these are pickrates, BST, Type Values and Meta Type Values. We do not use winrates or popularity as Pokémon Showdown does not track winrates and the popularity measure is not easily accessible. We run a grid search over these 4 components using the metrics defined in the Section 4.5.1 to determine our final team-building algorithm. We discuss the results of this grid search in Appendix E. Our final score function for the ABC-Meta task is:

$$\text{Score}_{ABCMeta} = \text{Pickrates} * \text{ BST} \tag{4.8}$$

For the BSD-Meta task where we have no prior knowledge of the metagame, we simulate the system from scratch thus allowing us to use both the popularity and winrates. We thus use both the winrates and popularity as a replacement for prior knowledge of the meta. In addition, we do not use the pickrates. If we used the pickrates, then the team-building algorithm would be dependent on the pickrates and the pickrates (in this scenario) would be entirely dependent on the team-building algorithm. This could cause an undesirable feedback loop which would result in a small set of Pokémon constantly being selected. Using the winrate instead breaks this feedback loop while also allowing the algorithm to prioritize stronger picks. Our score function is also different from the ABC-Meta task in that we use a sum instead of a product to arrive at the overall score. We make this change because we want to encourage diversity in the meta. Due to the addition of the popularity term, multiplying all three quantities would generally result in smaller values. The exceptions would be those Pokémon with a high value in multiple terms. This poses a problem as it would result in the use of a smaller set of Pokémon without sufficient exploration of the others. Thus we use a sum to increase the effect of each individual term. Our final score function for the BSD-Meta task is:

$$\text{Score}_{BSDMeta} = \text{Winrates} + (\text{c1} * \text{ BST } + \text{c2} * \text{ Meta Type Value } + \text{c3} * \text{ Type Value }) + \text{Popularity} \tag{4.9}$$

Where c1, c2, and c3 are constants that we set to 0.50, 0.25, and 0.25 respectively based on preliminary results.

For both the ABC-Meta and BSD-Meta tasks, we utilize an epsilon-greedy policy to pick Pokémon. We greedily select a Pokémon based on the probability derived above but there is an epsilon chance of selecting a Pokémon using the inverse pickrate, that is, selecting a less frequently used

Pokémon. This introduces an element of exploration in the team-building system that ensures that the system does not always generate the same set of teams. In the case of the ABC-Meta where we already have an idea of the meta, our epsilon value is fixed at 0.001. When calculated over the 12 Pokémon in a battle, this means there is an approximately 1% chance of picking a Pokémon based on the inverse pickrate. For the BSD-Meta where both the winrates and the popularity start with a default value of 0, we set the epsilon value to 1.0 and linearly decay it down to 0.001 over $20,000$ battles. This allows for an initialization process where the system can obtain statistics for every legal Pokémon.

4.4 Meta Discovery Environment

The final component of our meta discovery framework is the simulator environment. This environment must simulate battles using the battle agent, generate teams using our team-building system and also track statistics such as pickrates and winrates.

Recall that our final agents as discussed in Section 4.2.12 are setup to be able to run multiple battles concurrently. This may cause a race condition if we update our battle statistics at the end of each battle. Instead, we perform this update at regular intervals. To help reduce technical overhead, we also elect to generate a large number of teams at this point and simply sample a team at the start of a battle. The number of teams generated and the interval at which we update battle statistics is a parameter. We use an arbitrary value of 2500 teams generated and an update frequency of 1000 battles.

Since our goal is to approximate the metagame on Pokémon Showdown, we must follow the three-month system that it works on (Section 2.1.7). Hence, we determine the number of battles we need to simulate for one month. The most popular non-random competitive metagame on Pokémon Showdown is the OU ladder [74]. This averages about 1.45 million battles a month. The second most popular metagame is comparatively dwarfed at an average of 130,000 battles for the Ubers tier. We elect to use a fixed size number of battles per month for simplicity and pick 150,000 battles per month. In total, we simulate 3 months worth of changes for each battle and thus we perform 450,000 battles in total.

In addition, while presence in a tier is determined by usage rates, in practice this means 34 to 40 Pokémon were officially classified as a member of a tier. Thus we define the meta as the top 40 Pokémon in a tier.

Finally, we also implement a blanket ban of the LC tier in our system. That is, the Pokémon in the LC tier are not eligible to be picked by the team-building algorithm. We do this for two reasons. First, the Pokémon in the LC tier are extremely weak in relation to the rest of the Pokémon and can thus be easily identified. Second, they make up over a quarter of all the available Pokémon (210 out

of 740). As a result, our team-building algorithm would spend unnecessary time simulating battles between these Pokémon.

4.5 Meta Discovery

We now have all three components of our meta discovery system ready. We have two battle agents, the Simple Self-Play PPO agent and the Heuristic agent, a team-building algorithm for both our tasks, and a simulator environment to run meta discovery.

Our objective in the ABC-Meta task to analyze the changes to the metagame after a change, specifically a Pokémon being banned. This is a common occurrence in Pokémon Showdown [77] with such bans occurring every few months. To better evaluate our work, we specifically target instances where only a single ban is applied. In each scenario, we take the weighted usage statistics over the 3-month period immediately preceding the ban and average them with an equal weightage. We use this as the initial metagame, that is, the 40 most popular Pokémon in this list are classified as the meta Pokémon. We additionally also acquire the 3-month period after the ban to compare our framework against. In an ideal scenario, our discovered metagame would significantly overlap with this true metagame. We note that in general bans on Pokémon Showdown are enacted in the middle of a month. Since only statistics for the entire month are available, we do not consider the month in which the Pokémon is banned. Rather, we take the 3 months before the ban and the 3 months after the ban. At the time of writing, the latest available statistics are for October 2022. Thus in some cases this will limit the amount of data we have after the ban.

To show that our system is not suited to just one metagame, we identify four scenarios spanning four tiers - OU, UU, NU, and PU. In each case, the initial metagame is relatively stable with no major changes to the meta in a 3-month period. Following this period, a single Pokémon is banned, thus causing a shift in the metagame. We identify these scenarios by crawling through the Smogon forums. We note that these bans are not available in a structured format and identifying any changes requires significant manual effort. Table 4.4 depicts the 4 scenarios that we consider in terms of the tier, Pokémon banned and the month of the ban.

In the BSD-Meta, we have no prior metagame to work with. Rather, our aim is to analyze the discovered metagame when starting from a blank slate. In an ideal scenario, the discovered metagame should primarily contain the strongest Pokémon, that is, the Pokémon in the highest tiers of Smogon (Anything Goes, Ubers, OU). Here, we remove all bans (including the ban we impose on LC) and simulate 3 months of battle. We thus have only one scenario to consider which we term “Blank Slate".

Scenario	Tier	Bans	Month
Smogon OU	OU	Kyurem	2021-12
Smogon UU	UU	Aegislash	2022-06
Smogon NU	NU	Blastoise	2022-09
Smogon PU	PU	Vanilluxe	2022-08

Table 4.4: The 4 scenarios we consider for evaluating the AST-Meta task.

4.5.1 Evaluation Metrics

To evaluate our meta discovery framework on the ABC-Meta task, we develop two different metrics. Let us call the pre-ban metagame A, the post-ban metagame B and our discovered metagame B'.

Our first metric, Overlap, considers the overlap between B and B'. This tells us how close the two metagames are by comparing the specific Pokémon in the meta. A value of 0% would mean that none of the Pokémon in our discovered metagame B' are present in B while a value of 100% would mean that all the Pokémon in B' are present in B.

However, just identifying whether Pokémon are present in a metagame is not sufficient to fully evaluate our framework. The relative ordering of the Pokémon is also important. More specifically, we must compare the change in ranking of Pokémon from A to B compared to the change in ranking of Pokémon from A to B'. We do this using the Edit Distance between the Pokémon in the two metagames. We calculate the Edit Distance as the mean absolute difference in ranking for all Pokémon present in both A and B. This value tells us how much of a shift has occurred in the rankings of the Pokémon as a result of the change in meta. We can then compute the Edit Distance between A and B' and compare the two quantities. We note that even when the two quantities are close, it only means that the two metas have had a similar amount of change. It does not tell us that the same changes have occurred. Thus we use Spearman's Rho to understand if these changes correlate.

For the BSD-Meta task, we cannot use either the Overlap or the Edit Distance metrics from above as the pre-ban metagame A does not exist. So instead, we analyze the discovered meta by comparing the distribution of Pokémon in B' to the current (October 2022) tiers on Smogon. Specifically, we compute the percentage of the top 3 Smogon tiers that have been captured by the discovered metagame. We use the top 3 tiers as they consist of approximately 80 Pokémon (Anything Goes has only two Pokémon as discussed in Section 2.1.7). Thus we would expect the majority of, if not the entire, discovered metagame to be within these tiers.

Scenario	Blank Slate Discovery	Naive Baseline	Meta Discovery Simple Agent	Meta Discovery Heuristic Agent
Smogon OU	0.350	0.950	**0.975**	**0.975**
Smogon UU	0.375	0.975	**1.000**	**1.000**
Smogon NU	0.350	0.925	**0.950**	**0.950**
Smogon PU	0.300	**0.925**	**0.925**	**0.925**

Table 4.5: Evaluation of the meta discovery framework using the Overlap metric on the 4 ABC-Meta scenarios. Higher is better, 1.00 is perfect. **Bold** values indicate the best methods.

Scenario	Blank Slate Discovery	Naive Baseline	Meta Discovery Simple Agent	Meta Discovery Heuristic Agent
Smogon OU	108.77	3.04	**1.03**	1.09
Smogon UU	82.48	4.60	**1.30**	1.50
Smogon NU	79.85	5.57	2.74	**2.49**
Smogon PU	77.74	6.84	0.33	**0.11**

Table 4.6: Evaluation of the meta discovery framework using the Edit Distance metric on the 4 ABC-Meta scenarios. Values indicate the delta from the True Meta. Lower is better, 0.00 is perfect. **Bold** values indicate the best methods.

4.5.2 Approaches

For the ABC-Meta tasks, we utilize two baselines. The first is what we term a naive baseline. That is, we take the pre-ban usage statistics and simply remove the entry for the Pokémon that we wish to ban. Thus all the rankings below it would be shifted up by one. For our other baseline, we use the same system we use for the BSD-Meta (using the Simple Agent). We expect this to perform the worst as it does not use prior knowledge of the metagame, but include it to demonstrate the importance of having this knowledge. Since we use two agents as discussed in Section 4.2.12, we have two versions of our meta discovery framework. One uses the Simple Self-Play PPO Agent and the other which uses the Heuristic Agent.

For the BSD-Meta task, there are no existing baselines for this task due to the lack of prior work. Instead, we elect to craft a baseline that is just a sorted list of Pokémon based on their BST, as the BST plays a large role in determining a Pokémon's strength. We utilize the same two versions of our meta discovery framework as in the ABC-Meta task.

4.5.3 Results

Table 4.5 depicts the results on the Overlap metric. We see that in all four cases, our meta discovery approach meets or outperforms the baselines. The Naive Baseline performs reasonably well since the bans caused small shifts in the meta that moved a few Pokémon in and out of the meta. The Blank Slate Discovery method performs the worst, achieving only about 35% Overlap in most cases.

Scenario	Agent	Rho	p-value
Smogon OU	Simple	0.1861	0.2502
Smogon OU	Heuristic	0.1827	0.2589
Smogon UU	Simple	-0.1455	0.3702
Smogon UU	Heuristic	-0.1304	0.4224
Smogon NU	Simple	-0.0645	0.6960
Smogon NU	Heuristic	-0.0686	0.6779
Smogon PU	Simple	0.1152	0.4787
Smogon PU	Heuristic	0.1482	0.3613

Table 4.7: Results of Spearman's Rho on the Meta Discovery approaches.

This could be due to it having no prior knowledge of the metagame.

Table 4.6 depicts the results of our Edit Distance metric. Specifically, we show the delta from the True Meta. Even here, the Blank Slate Discovery baseline is significantly worse than the others, having discovered a vastly different metagame. The naive baseline is relatively much better since it effectively just shifts some percentage of the meta up by one ranking. However, our meta discovery approach once again performs the best, with Edit Distances that are very close to the true meta.

Looking at the results of our Meta Discovery approach between the Simple and Heuristic agents, we see that there is no significant difference in terms of their overall metrics. This is to be expected to some extent as both agents were quite close to each other in terms of overall performance. However given that the heuristic agents require a lot more domain knowledge, we prefer the Simple agent which only requires a larger time commitment.

Considering both metrics together, we see that though the Naive Baseline has a high level of overlap, it does not account for the shifts within the meta itself. Further, the Blank Slate Discovery baseline suffers heavily due to the lack of metagame knowledge. Our approach on the other hand has a greater overlap while also simulating the shifts within the metagame. We note that despite outperforming the baselines, our Meta Discovery approach is still some way from the True Meta. This tells us that there is scope for improvement in terms of our team-building approach as well as in our trained agents. Despite this, we present clear evidence that our approach is able to approximate the effects of a change in the meta to a high degree.

As discussed in Section 4.5.1, the Edit Distance only tells us how similar the quantity of changes to the meta are, not how close the two metas area. To study this, we compute Spearman's Rho over our meta discovery approaches in Table 4.7. We see that the two agents are once again quite similar. More importantly, we see that there is a weak positive correlation in two of the scenarios (Smogon OU, Smogon PU). However, due to the high p-value in all these cases, we cannot reject the null hypothesis for any of them. Although this is not ideal, it serves to re-emphasize how difficult this problem of predicting a Pokémon's placement in the meta is, especially without human expertise

Agent	AG	Ubers	OU
BST Baseline	100%	71.79%	22.86%
Simple Agent	100%	61.54%	17.14%
Heuristic Agent	100%	64.10%	17.14%

Table 4.8: Evaluation on the BSD-Meta task. Values are the percentage of the Smogon tiers in the discovered meta.

Agent	AG	Ubers	OU	Below OU
BST Baseline	5.0%	70.0%	20.0%	5.0%
Simple Agent	5.0%	60.0%	15.0%	20.0%
Heuristic Agent	5.0%	62.5%	15.0%	17.5%

Table 4.9: Composition of the metas discovered in the BSD-Meta task. Values are the percentage of the discovered meta that is classified in a particular Smogon tier.

guiding it. Despite this, there are still positive signs as we have a weak positive correlation in half of the scenarios.

Table 4.8 depicts the results of our system on the BSD-Meta task. As discussed previously, we look at the percentage of the top 3 Smogon tiers that our system has predicted. In all cases, the Anything Goes tier is captured in its entirety and over 60% of the Ubers tier is captured. In Table 4.9, we instead examine the composition of our discovered meta in terms of the Smogon tiers. In all cases, at least 80% of the discovered meta is within the top 3 Smogon tiers. However, we see that our framework performs worse than the baseline, capturing certain Pokémon in lower tiers. We suspect that this happens for two reasons. First, the framework may be finding certain Pokémon that work especially well against the higher tier Pokémon but are not necessarily strong Pokémon when considered independently. Second, our framework might be performing too much exploration and not enough exploitation. In standard Smogon tiers, anywhere between 50 to 80 percent of picks are from the meta, that is, the top 40 most popular Pokémon. We observed the same effect in our ABC-Meta task results. However, in the case of the BSD-Meta task where we don't have prior knowledge of the meta, we are forced to explore more often. We found that the top 50% of picks in the discovered meta span across 300+ Pokémon. This tells us that we need a more complex system to pick Pokémon for this task. On comparing our two different agents, the results are quite close, with the meta discovered by the Heuristic agent picking one Pokémon more in the Ubers tier. As discussed earlier, this is largely expected due to the similarity in performance between the agents. Considering these results, we believe that the reinforcement learning agent is the preferable option due to the high domain knowledge required to build a heuristic agent.

4.6 Conclusion

In this chapter, we introduced a framework for evaluating changes to game balance via meta discovery. We developed several reinforcement learning agents to approximate human players and created a team-generation system that adapts to changes in the meta. We implemented this system in the Pokémon Showdown competitive online game and evaluated the effectiveness of this system on two separate tasks. Our results show that the framework is effective in approximating the results of balance changes to the metagame.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we have presented two major contributions towards our ultimate goal of a system capable of automating the design of all aspects of a character. Specifically, we work on a method for improving representation of pixel art and a framework to evaluate changes to game balance via meta discovery.

We presented a novel methodology for representing pixel art. Specifically, we introduced the Pixel VQ-VAE, an enhanced VQ-VAE model specialized for pixel art. We evaluated the Pixel VQ-VAE against several baselines on the quality of the embeddings and performed an ablation study on each of our enhancements. We demonstrated the applications of the Pixel VQ-VAE on two downstream tasks - image generation and image transformation across two distinct pixel art datasets. We found that the Pixel VQ-VAE works better on high variance data in comparison to other approaches. Given our results, we can justify our hypothesis (Thesis Statement 1) that VQ-VAEs could learn a pixel art representation.

Our second contribution is a framework to evaluate changes to game balance via meta discovery. To do this, we trained several reinforcement learning agents to approximate an average human player and introduced a dynamic team-generation system that adapts to changes in the metagame. We implement this framework on the Pokémon Showdown competitive online game and evaluate its usefulness by approximating the results of bans applied to the character roster. We additionally explored the open problem of learning a metagame from scratch. Though our results are not perfect, we can confirm our hypothesis (Thesis Statement 2) that a meta discovery system to evaluate the impact of balance patches would help improve game balance.

To the best of our knowledge, we are the first to utilize the VQ-VAE's encoding-pixel corre-

spondence for the representation of pixel art. In addition, we also believe that we are the first to implement a framework for evaluating game balance via meta discovery. We hope that our work acts as a starting point for further work in both domains.

5.2 Future Work

The ultimate goal of our work is to build a comprehensive system that can generate all aspects of a character. Thus furthering existing work in aspects such as music or text generation is an important step towards this. Another important area of future work is in evaluation metrics for both contributions. In the case of pixel art, existing metrics that evaluate generated images are not optimal. The FID score we use is built for photorealistic images. Thus one area of consideration is in the development of such metrics to evaluate generated pixel art. In the case of meta discovery, no prior work exists in this area. Thus we developed our own. However, further study on the effectiveness of these metrics is required. There are still several avenues for future work for each of our two contributions.

For instance, our Pixel VQ-VAE is based off the VQ-VAE. However there are several other models that build from and have improved upon the VQ-VAE such as the VQ-VAE 2 [61] and the VQ-GAN [11]. In addition, there are also several other downstream tasks such as image classification or text generation that might benefit from our learned representations. Further, a more comprehensive evaluation of our generative models, especially in terms of appeal, could be useful. A human subject study, like in [71], would be a good metric to study such a subjective problem. Finally, one area of particular interest to the author is to employ some form of conditioning during the training of the VQ-VAE in a manner similar to [13].

In the case of our meta discovery framework, there are avenues for improvement on each component. At present, our battle agent roughly approximates an average human player. However, the Pokémon Showdown ladder consists of different players of varying skill levels. In addition, different players might have different approaches to the game. Some might prefer to constantly be on the attack while others might prefer to take things slower. Thus training a pool of different agents to battle against each other might result in a more accurate simulation of the meta. In this scenario, the team-building approach would also have to generate teams based on the user's requirements. Given the dynamic nature of this system, a reinforcement learning system to generate teams for each user might work. We envision an ideal scenario where both the team-building system and the battle agent learn as they battle, thus organically adjusting to the perceived meta. In our work we evaluate our system on 5 different scenarios. In the future, we would like to build upon this by evaluating the system on more complex scenarios such as a group of multiple bans together or a sequence of several bans being enacted over consecutive months. This would also handle the prob-

lem discussed in Section 4.2 where we train agents on a different battle format from the one used in the simulation. Finally, we would also like to use our system to handle entirely different changes to the metagame such as the addition of new characters or the modification of existing characters. Identifying such scenarios would itself be a contribution towards future research.

There are other areas for future work that are more technical in nature. One major drawback of the poke-env API was the overhead cost of extracting elements of the state from the environment. To utilize more complex agents, solving this issue is of great importance. A simpler solution might also involve utilizing several instances of the agent to perform battles while a central brain handles the tracking of statistics and the generation of teams. This would allow for significantly more battles to be run in parallel.

5.3 Potential Impact

Each of our contributions can be built upon independently. For instance, our pixel art representation system could be used as-is for learning representations of any collection of images that utilize pixel art, not just games. These representations could then be used to train downstream models for image generation and transformation.

Our meta discovery framework also offers value in several situations. For instance, an iterative balancing process where developers can make small changes to the metagame, determine their impact and make adjustments on those changes. Another case would be in the introduction of new characters. By simulating the competitive metagame after adding new characters, developers could attain better insights as to the character's overall balance in the meta. Both scenarios would make use of the same system as our ABC-Meta task. Another interesting application is in the generation of a new metagame from scratch. For instance, if a large number of changes are introduced (as new Pokémon games tend to do), developers could use this framework to get an idea of what the meta could look like. They could then potentially make changes so that the meta approaches their desired state. In this scenario, we would use our meta discovery system for the BSD-Meta task.

Our contributions could also be used together in the generation of new characters for games. The Pokémon video game series which we utilize in this thesis is a prime example where each of these components would be useful. These components could further be complemented with other aspects of a character including text generation in the form of lore generation, music generation for character themes or sounds and character attribute generation. Although we do not work on these areas, prior work does explore them [12, 37, 50].

However, our ultimate goal is to develop a system capable of automating the design of all aspects of a character including images, sounds, abilities, text alongside automated balancing. In essence, we wish to orchestrate [44] several generative models for the specific purpose of character generation.

Such a system does not yet exist, and we thus encourage future researchers to study this problem.

Bibliography

[1] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., and Arshad, H. State-of-the-art in artificial neural network applications: A survey. *Heliyon 4*, 11 (2018), e00938.

[2] Aninda, A. Meta discovery and role-based matchmaking (mediroma).

[3] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine 34*, 6 (2017), 26–38.

[4] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[5] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[6] Coutinho, F., and Chaimowicz, L. On the challenges of generating pixel art character sprites using gans. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 18*, 1 (Oct. 2022), 87–94.

[7] Crane, D., Holmes, Z., Kosiara, T. T., Nickels, M., and Spradling, M. Team counter-selection games. In *2021 IEEE Conference on Games (CoG)* (2021), pp. 1–8.

[8] da Silva Oliveira, S., Silva, G. E. P. L., Gorgônio, A. C., Barreto, C. A. S., Canuto, A. M. P., and Carvalho, B. M. Team recommendation for the pokémon go game using optimization approaches. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (2020), pp. 163–170.

[9] Dai, B., and Wipf, D. Diagnosing and enhancing VAE models. *7th International Conference on Learning Representations, ICLR 2019* (2019).

[10] David Cole, R. G. How we balance valorant. https://playvalorant.com/en-us/news/dev/how-we-balance-valorant/, 2020.

[11] Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis, 2020.

[12] Geissler, D., Nguyen, E., Theodorakopoulos, D., and Gatti, L. Pokérator - unveil your inner pokémon. In *Proceedings of the 11th International Conference on Computational Creativity (ICCC'20)* (2020), pp. 500–503.

[13] González, A., Guzdial, M. J., and Ramos, F. Generating gameplay-relevant art assets with transfer learning. *Proceedings of the Experimental AI in Games (EXAG) Workshop, AIIDE* (2020).

[14] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. In *Generative Adversarial Networks* (oct 2020), vol. 63-11, Association for Computing Machinery, p. 139–144.

[15] Hernandez, D., Toyin Gbadamosi, C. T., Goodman, J., and Walker, J. A. Metagame autobalancing for competitive multiplayer games. In *2020 IEEE Conference on Games (CoG)* (2020), pp. 275–282.

[16] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), NIPS'17, p. 6629–6640.

[17] Ho, H., and Ramesh, V. Percymon: A pokemon showdown artifical intelligence.

[18] Hong, S.-J., Lee, S.-K., and Yang, S.-I. Champion recommendation system of league of legends. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)* (2020), pp. 1252–1254.

[19] Hore, J. Orianna has been disabled for league of legends worlds 2022. https://www.theloadout.com/league-of-legends/orianna-disabled-lol-worlds-2022, 2022.

[20] Horsley, L., and Perez-Liebana, D. Building an automatic sprite generator with deep convolutional generative adversarial networks. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)* (2017), pp. 134–141.

[21] Huang, D., and Lee, S. A self-play policy optimization approach to battling pokémon. In *2019 IEEE Conference on Games (CoG)* (2019), pp. 1–4.

[22] III, A. Future sight ai. https://www.pokemonbattlepredictor.com/home.

[23] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[24] Jadhav, M., and Guzdial, M. Tile embedding: A general representation for level generation. In *Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2021), AIIDE'21.

[25] Jaffe, A. Metagame balance for esports & fighting games. https://www.youtube.com/watch?v=miu3ldl-nY4, 2015.

[26] Jaffe, A., Miller, A., Andersen, E., Liu, Y.-E., Karlin, A., and Popović, Z. Evaluating competitive game balance with restricted play. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2012), AIIDE'12, AAAI Press, p. 26–31.

[27] Japeal. Fakemon maker. https://japeal.com/pkm/, 2017.

[28] Kalose, A., Kaya, K., and Kim, A. Optimal battle strategy in pokémon using reinforcement learning.

[29] Karth, I., Aytemiz, B., Mawhorter, R., and Smith, A. M. Neurosymbolic map generation with vq-vae and wfc. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021* (2021), FDG'21, Association for Computing Machinery.

[30] Kevin Chen, E. L. Gotta train 'em all: Learning to play pokémon showdown with reinforcement learning.

[31] Kingma, D. P., and Welling, M. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014).

[32] Kingma, D. P., and Welling, M. *An Introduction to Variational Autoencoders*. 2019.

[33] Kleiber, J. Pokegan. https://github.com/jkleiber/PokeGAN, 2020.

[34] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., and Brown, D. Text classification algorithms: A survey. *Information 10*, 4 (2019), 150.

[35] Kuo, M.-H., Yang, Y.-L., and Chu, H.-K. Feature-aware pixel art animation. *Comput. Graph. Forum 35*, 7 (oct 2016), 411–420.

[36] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning* (2016), PMLR, pp. 1558–1566.

[37] Latitude. Ai dungeon 2. https://aidungeon.io/.

[38] Lazarou, C. Autoencoding generative adversarial networks.

[39] Lee, C.-S., and Ramler, I. Identifying and evaluating successful non-meta strategies in league of legends. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (New York, NY, USA, 2017), FDG '17, Association for Computing Machinery.

[40] Lee, S., and Togelius, J. Showdown ai competition. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)* (2017), pp. 191–198.

[41] Li, B., Qi, X., Lukasiewicz, T., and Torr, P. H. S. *Controllable Text-to-Image Generation.* 2019.

[42] Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems* (2021), 1–21.

[43] Liapis, A. Recomposing the pokémon color palette. In *International Conference on the Applications of Evolutionary Computation* (2018), Springer, pp. 308–324.

[44] Liapis, A., Yannakakis, G. N., Nelson, M. J., Preuss, M., and Bidarra, R. Orchestrating game generation. *IEEE Transactions on Games 11*, 1 (2019), 48–68.

[45] Liu, Y., De Nadai, M., Cai, D., Li, H., Alameda-Pineda, X., Sebe, N., and Lepri, B. *Describe What to Change: A Text-Guided Unsupervised Image-to-Image Translation Approach.* Association for Computing Machinery, 2020, p. 1357–1365.

[46] Mesentier Silva, F. d., Canaan, R., Lee, S., Fontaine, M. C., Togelius, J., and Hoover, A. K. Evolving the hearthstone meta. In *2019 IEEE Conference on Games (CoG)* (2019), IEEE Press, p. 1–8.

[47] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (2013), Y. Bengio and Y. LeCun, Eds.

[48] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[49] Munroe, L. veekun sprite packs. https://veekun.com/dex/downloads, 2017.

[50] N. Ferreira, L., Mou, L., Whitehead, J., and Lelis, L. H. S. Controlling perceived emotion in symbolic music generation with monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 18*, 1 (Oct. 2022), 163–170.

[51] Nichol, A., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., and Chen, M. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models.

[52] Norström, L. *Comparison of artificial intelligence algorithms for Pokémon battles*. Master's Thesis, Department of Space, Earth and Environment, Chalmers University of Technology, 2019.

[53] Onsager, A. Pokemon fusion. https://pokemon.alexonsager.net/, 2013.

[54] Pardo, R. Making a standard (and trying to stick to it!): Blizzard design philosophies. https://www.gdcvault.com/play/1012291/Making-a-Standard-(and-Trying, 2010.

[55] Peabody, D. P. Detecting metagame shifts in league of legends using unsupervised learning. In *University of New Orleans Theses and Dissertations. 2482* (2018).

[56] Pei, A. This esports giant draws in more viewers than the super bowl, and it's expected to get even bigger. https://www.cnbc.com/2019/04/14/league-of-legends-gets-more-viewers-than-super-bowlwhats-coming-next.html, 2019.

[57] pmariglia. Showdown.

[58] pokecolours. Pokemon palette swaps. https://pokecolours.tumblr.com/, 2016.

[59] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (2016), Y. Bengio and Y. LeCun, Eds.

[60] Rawat, W., and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation 29*, 9 (2017), 2352–2449.

[61] Razavi, A., van den Oord, A., and Vinyals, O. *Generating Diverse High-Fidelity Images with VQ-VAE-2*. 2019.

[62] Rebouças Serpa, Y., and Formico Rodrigues, M. A. Towards machine-learning assisted asset generation for games: A study on pixel art sprite sheets. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (2019), pp. 182–191.

[63] Reijm, H. A time-efficient competitive pokemon team-building algorithm.

[64] Reis, S., Reis, L. P., and Lau, N. Vgc ai competition - a new model of meta-game balance ai competition. In *2021 IEEE Conference on Games (CoG)* (2021), pp. 01–08.

[65] Rill-Garcia, R. Reinforcement learning for a turn-based small scale attrition game.

[66] Riot Repertoir, R. G. Champion balance framework. https://www.leagueoflegends.com/en-us/news/dev/dev-champion-balance-framework/, 2019.

[67] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2022), pp. 10684–10695.

[68] Sahovic, H. poke-env. https://github.com/hsahovic/poke-env, 2019.

[69] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature 588*, 7839 (2020), 604–609.

[70] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[71] Shields, S., Mawhorter, R., Melcer, E., and Mateas, M. Searching for balanced 2d brawler games: Successes and failures of automated evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2022), vol. 18, pp. 189–198.

[72] Simões, D., Reis, S., Lau, N., and Reis, L. P. Competitive deep reinforcement learning over a pokémon battling simulator. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)* (2020), pp. 40–45.

[73] Smogon. Smogon. https://www.smogon.com/.

[74] Smogon. Stats. https://www.smogon.com/stats/.

[75] Smogon. Everything you ever wanted to know about ratings. https://www.smogon.com/forums/threads/everything-you-ever-wanted-to-know-about-ratings.3487422/, 2013.

[76] Smogon. Gen 8 smogon university usage statistics discussion thread. https://www.smogon.com/forums/threads/gen-8-smogon-university-usage-statistics-discussion-thread.3657197/, 2019.

[77] Smogon. Ou forum rules, ou council info, and announcements. https://www.smogon.com/forums/threads/ou-forum-rules-ou-council-info-and-announcements.3656260/#post-8284243, 2019.

[78] Smogon. Tiering for generation 8. https://www.smogon.com/forums/threads/tiering-for-generation-8.3657121, 2019.

[79] Stone, D. Technical machine. https://github.com/davidstone/technical-machine, 2010.

[80] Summerville, A., Cook, M., and Steenhuisen, B. Draft-analysis of the ancients: Predicting draft picks in dota 2 using machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 12*, 2 (Jun. 2016), 100–106.

[81] Summerville, A. J., Snodgrass, S., Mateas, M., and n'on Villar, S. O. The vglc: The video game level corpus. *Proceedings of the 7th Workshop on Procedural Content Generation* (2016).

[82] Sun, S., Cao, Z., Zhu, H., and Zhao, J. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics 50*, 8 (2019), 3668–3681.

[83] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems 27* (2014).

[84] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9.

[85] Thoss, J., Ensslin, A., and Ciccoricco, D. Narrative Media: The Impossibilities of Digital Storytelling. *Poetics Today 39*, 3 (09 2018), 623–643.

[86] Tomašev, N., Paquet, U., Hassabis, D., and Kramnik, V. Assessing game balance with alphazero: Exploring alternative rule sets in chess. *arXiv preprint arXiv:2009.04374* (2020).

[87] Toolkit, G. M. How games get balanced. https://www.youtube.com/watch?v=WXQzdXPTb2A, 2019.

[88] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, p. 1747–1756.

[89] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. Conditional image generation with pixelcnn decoders. In *Proceedings of*

the 30th International Conference on Neural Information Processing Systems (2016), NIPS'16, p. 4797–4805.

[90] van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), NIPS'17, p. 6309–6318.

[91] van der Maaten, L., and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research 9*, 86 (2008), 2579–2605.

[92] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (2016), vol. 30.

[93] Vertex. The smog, smogon university. https://www.smogon.com/smog/issue39/anything-goes, 2015.

[94] Whitehouse, C. clip-guided-diffusion-pokemon. https://replicate.com/cjwbw/clip-guided-diffusion-pokemon, 2022.

[95] Wong, R. N. pokemon2pokemon: Using neural networks to generate pokemon as different elemental types. https://www.rileynwong.com/blog/2019/5/22/pokemon2pokemon-using-cyclegan-to-generate-pokemon-as-different-elemental-types, 2019.

[96] Yingzhen, L., and Mandt, S. Disentangled sequential autoencoder. In *Proceedings of the 35th International Conference on Machine Learning* (10–15 Jul 2018), vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 5670–5679.

[97] Zhai, J., Zhang, S., Chen, J., and He, Q. Autoencoder and its various variants. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2018), IEEE, pp. 415–419.

[98] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)* (2017).

[99] Zielke, G. Gotta win'em all: How expert play in the online community of smogon changes pokemon.

[100] Zook, A., Fruchter, E., and Riedl, M. O. Automatic playtesting for game parameter tuning via active learning. In *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3-7, 2014* (2014), Society for the Advancement of the Science of Digital Games.

Appendix A

Pixel VAE

Model	**MSE**	**SSIM**
VAE	0.01815	0.64272
Pixel VAE	0.01257	0.74339
VQ-VAE LowRes	0.01076	0.60198
Pixel VQ-VAE LowRes	0.01040	0.78669
VQ-VAE MedRes	0.00584	0.63973
Pixel VQ-VAE MedRes	0.00421	**0.91210**
Pixel VQ-VAE HiRes	**0.00070**	0.82967

Table A.1: Test set reconstruction metrics on the Pokémon dataset. MSE: Lower is better. SSIM: Higher is better.

We believe that the PixelSight block introduced in Section 3.1 could work to improve performance on pixel art in any CNN-based model. Thus, we test this in a traditional VAE. Taking the same baseline described in Section 3.3.2, we add a PixelSight block to the start of the encoder and another PixelSight block with transposed convolutions at the end of the decoder. In addition, we modified the training objective, specifically, the reconstruction loss, to use a combination of the Mean Squared Error (MSE) and the Structural Similarity Index (SSIM). We reasoned that the MSE, a per-pixel loss function, does not directly contribute to the structure of the reconstructed image. As such, including the SSIM which focuses purely on the structure of the image might help in improving reconstructions. In an attempt to give a greater focus to the reconstruction loss, we also experimented with a weighted KL-Divergence. After some tuning we found that a weight of 0.1 gave us the biggest improvement in terms of both MSE and SSIM. We trained this model for 25 epochs with a batch size of 64 with the Adam optimizer and a learning rate of 0.0001.

Table A.1 compares the results against the other baselines and the Pixel VQ-VAE. We see that there is a significant improvement over the baseline VAE, with both the MSE and the SSIM seeing significant jumps in score. However, this performance is still much worse than the VQ-VAEs,

especially in terms of the MSE. Table A.2 gives a visual comparison of the same.

Model	Pokémon 1	Pokémon 2
Original		
VAE		
Pixel VAE		
VQ-VAE LowRes		
Pixel VQ-VAE LowRes		
VQ-VAE MedRes		
Pixel VQ-VAE MedRes		
Pixel VQ-VAE HiRes		

Table A.2: Test set reconstruction comparison of the Pixel VAE on the Pokémon dataset.

Appendix B

Impact of Sprite Rotations on the Pixel VQ-VAE

Model	Rotations	MSE	SSIM
Pixel VQ-VAE LowRes	No	0.01075	0.65624
Pixel VQ-VAE LowRes	Yes	0.01040	0.78669
Pixel VQ-VAE MedRes	No	0.00443	0.72544
Pixel VQ-VAE MedRes	Yes	0.00421	0.91210
Pixel VQ-VAE HiRes	No	0.00155	0.73377
Pixel VQ-VAE HiRes	Yes	0.00070	0.82967

Table B.1: Analysis on the impact of augmenting the Pixel VQ-VAE training data using random rotations.

In this section, we examine the impact that performing random rotations on our training data has on our Pixel VQ-VAE's performance. To evaluate this, we train another version of each of our Pixel VQ-VAE models but do not perform any rotations while preprocessing the data. All other hyperparameters remain the same. Table B.1 details the results of this experiment. We see that in all cases, using the rotations results in a significant improvement in the SSIM and a minor improvement in the MSE.

Appendix C

Best and Worst Case Reconstruction Analysis of the Embedding Space

Here we include images of the best and worst case reconstructions for the Pixel VQ-VAE and the baselines on both metrics (MSE and SSIM). We include this to help the reader better understand the quality of each model. We note that in many of these cases the same image appears to be repeated across the different models. Although this does mean that the models had some difficulty with these images, we note that in many of these cases multiple images had the same performance and thus the first such instance was picked. Tables C.1 and C.2 depict the results on the Pokémon dataset while Tables C.3 and C.4 depict the results on the Sprites dataset.

Looking at the table, we can see that our Pixel VQ-VAEs do not vary significantly in the quality of reconstructions. This is not as true in some of the baseline such as the VAE, which exhibits a large difference between the best and worst reconstructions such as in Tables C.1 and C.2. This is in line with the metrics depicted in Chapter 3.

Model	Best Case Original	Best Case Reconstructed	Worst Case Original	Worst Case Reconstructed
VAE				
VQ-VAE LowRes				
Pixel VQ-VAE LowRes				
VQ-VAE MedRes				
Pixel VQ-VAE MedRes				
Pixel VQ-VAE HiRes				

Table C.1: Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Pokémon dataset according to the MSE metric.

Model	Best Case Original	Best Case Reconstructed	Worst Case Original	Worst Case Reconstructed
VAE				
VQ-VAE LowRes				
Pixel VQ-VAE LowRes				
VQ-VAE MedRes				
Pixel VQ-VAE MedRes				
Pixel VQ-VAE HiRes				

Table C.2: Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Pokémon dataset according to the SSIM metric.

Model	Best Case Original	Best Case Reconstructed	Worst Case Original	Worst Case Reconstructed
VAE				
VQ-VAE LowRes				
Pixel VQ-VAE LowRes				
VQ-VAE MedRes				
Pixel VQ-VAE MedRes				
Pixel VQ-VAE HiRes				

Table C.3: Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Sprites dataset according to the MSE metric.

Model	Best Case Original	Best Case Reconstructed	Worst Case Original	Worst Case Reconstructed
VAE				
VQ-VAE LowRes				
Pixel VQ-VAE LowRes				
VQ-VAE MedRes				
Pixel VQ-VAE MedRes				
Pixel VQ-VAE HiRes				

Table C.4: Test set reconstructions of the Pixel VQ-VAE and the baselines. Columns indicate the best and worst case reconstructions on the Sprites dataset according to the SSIM metric.

Appendix D

Embedding Visualization of the Pixel VQ-VAE

As discussed in Section 3.3.3, The VQ-VAE latent space cannot be analyzed in the same manner as traditional latent spaces due to its discrete nature. Since the learned embeddings correspond to individual patches of an image and not the image as a whole, attempting to analyze individual embeddings would not be very helpful. Instead, we obtain an embedding for the image as a whole by considering the group of all embeddings for that image.

We plot the results of a t-SNE [91] visualization in three cases (baseline VAE, VQ-VAE MedRes, Pixel VQ-VAE MedRes) for both the datasets. In all cases we used PCA initialization and varied the perplexity from 10-50, using the best visualization in each case.

We use 10% of the Sprites dataset (due to its large size). The results for the baseline VAE, VQ-VAE MedRes, and our Pixel VQ-VAE MedRes are in Figures D.1, D.2, and D.3. The points are colored based on the orientation of the sprite - left, right or front-facing.

We see that in the case of the VAE, the three categories are separated out to some degree, though there is significant overlap. While both VQ-VAEs separate out the three categories, our Pixel VQ-VAE groups each category into a single, separable cluster. However, we suspect that these results are largely due to the consistent nature of the Sprites dataset. Since all the images have the same basic template, it is relatively easy for the model to separate them into groups due to their common nature.

This is not the case for Pokémon. For this dataset, the results for the baseline VAE, VQ-VAE MedRes, and our Pixel VQ-VAE MedRes are in Figures D.4, D.5, and D.6. The points are colored based on the Pokémon's type attribute. We use only the first type as many Pokémon do not have a second.

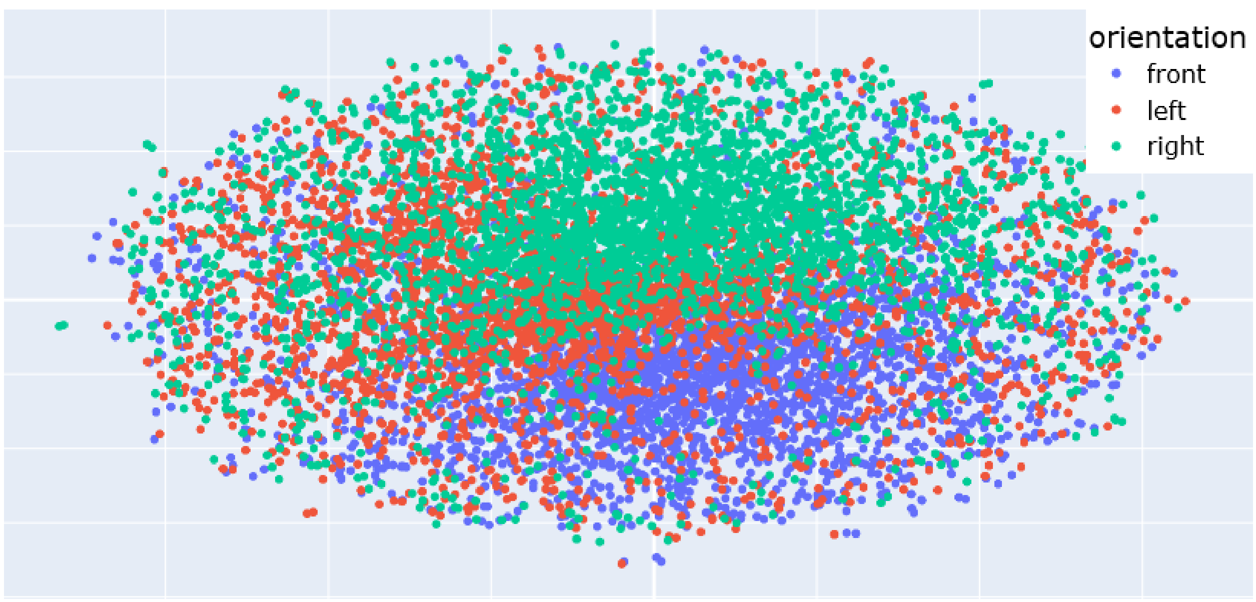

Figure D.1: t-SNE Plot of the latent space of the Sprites dataset in the VAE.

We reiterate that the Pokémon dataset is a far more complicated dataset than the Sprites dataset. It has several hundreds of unique monsters each with their own distinct size and appearance. These monsters are all classified into one of 18 types. While their color is a decent heuristic for their type, this is not always true. From the plots we see that none of the models do very well in terms of separating the latent space.

However, we do point out one specific area of interest - the cluster of orange dots found in all 3 plots. This cluster consists of one particular Pokémon with 17 different forms, all which are nearly identical. We see that this cluster (middle-right) is quite close to most of the other points in the VAE. When we examine the base VQ-VAE we see that this cluster is more separate from the other points. Finally, the Pixel VQ-VAE puts even more distance between this cluster and the other points. Thus we can see that the models do identify some meaning in the latent space here. While not easily visible in a static plot, there are a number of similar clusters in the images that exhibit this behavior such as the small purple cluster in the Pixel VQ-VAE plot. We suspect that although the models are learning some information in the latent space, there exists too much variation between individual Pokémon for the model to capture it consistently. Training these models along with some specific gameplay properties of different Pokémon like in González et al [13] might lead to a richer latent space.

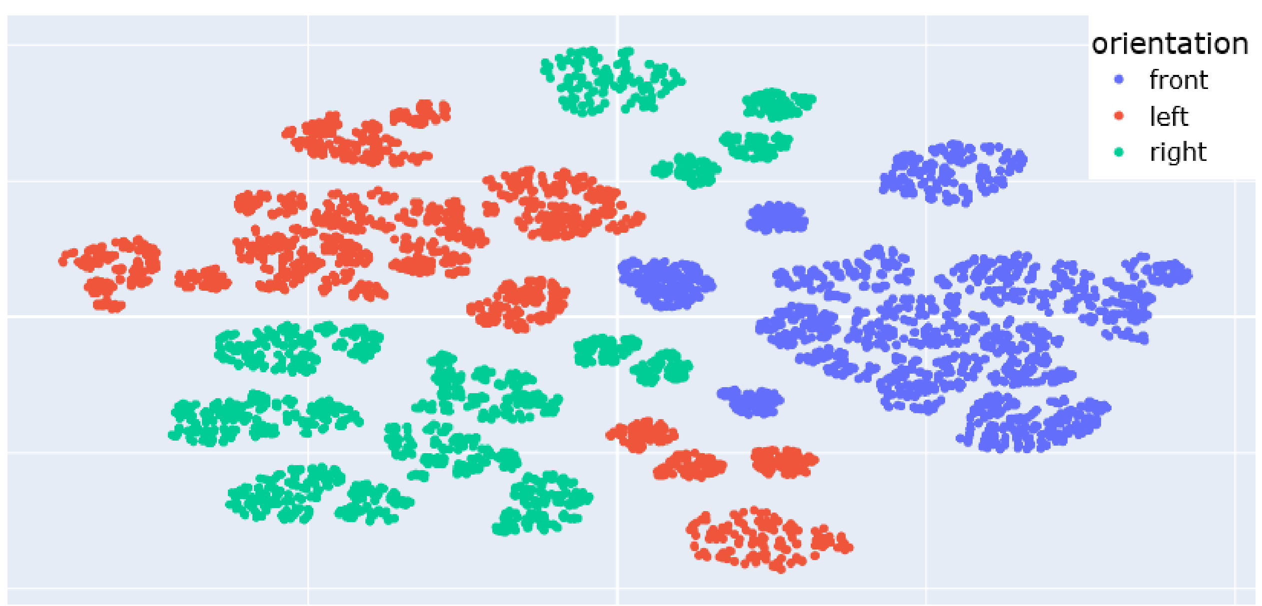

Figure D.2: t-SNE Plot of the latent space of the Sprites dataset in the VQ-VAE MedRes.

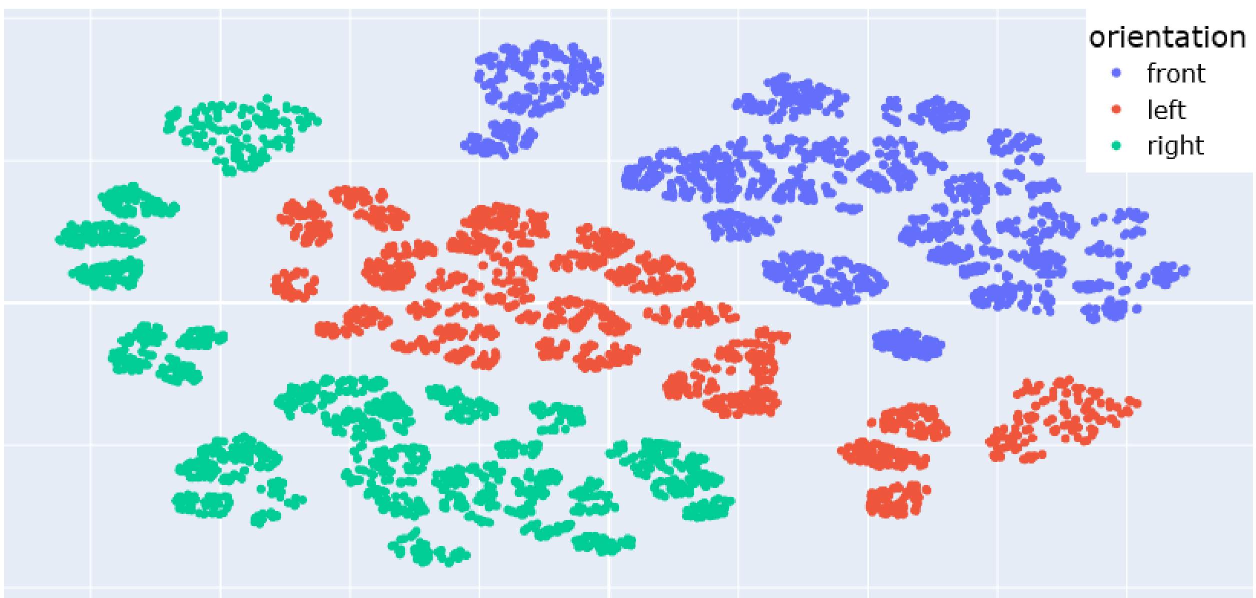

Figure D.3: t-SNE Plot of the latent space of the Sprites dataset in the Pixel VQ-VAE MedRes.

VAE

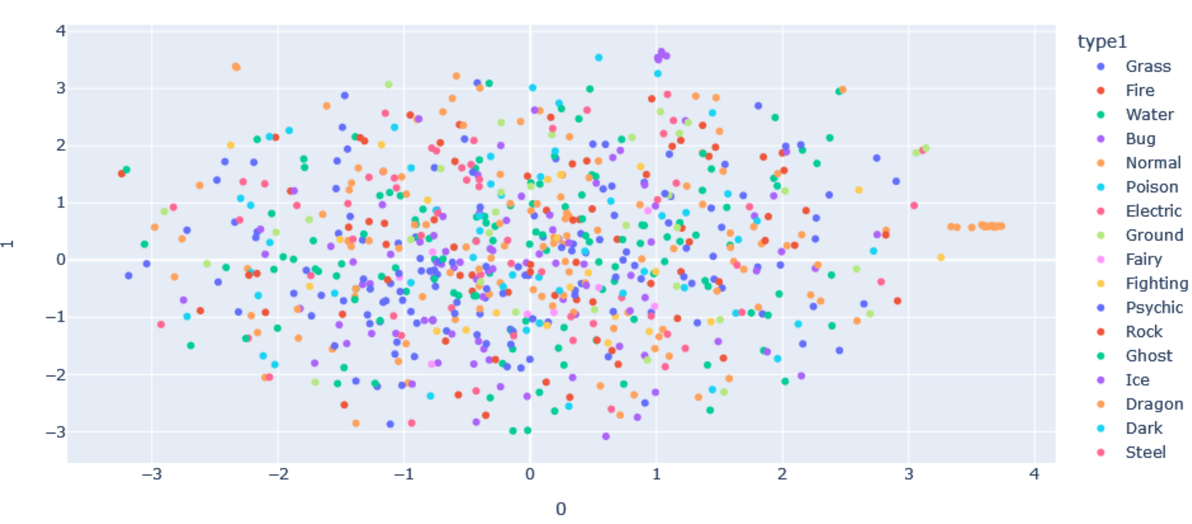

Figure D.4: t-SNE Plot of the latent space of the Pokémon dataset in the VAE.

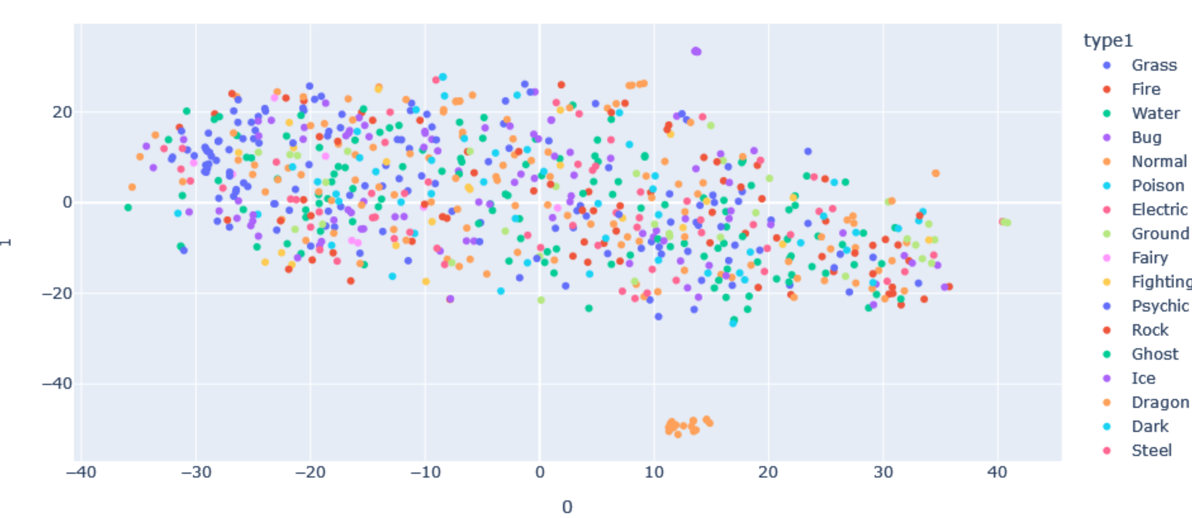

Figure D.5: t-SNE Plot of the latent space of the Pokémon dataset in the VQ-VAE MedRes.

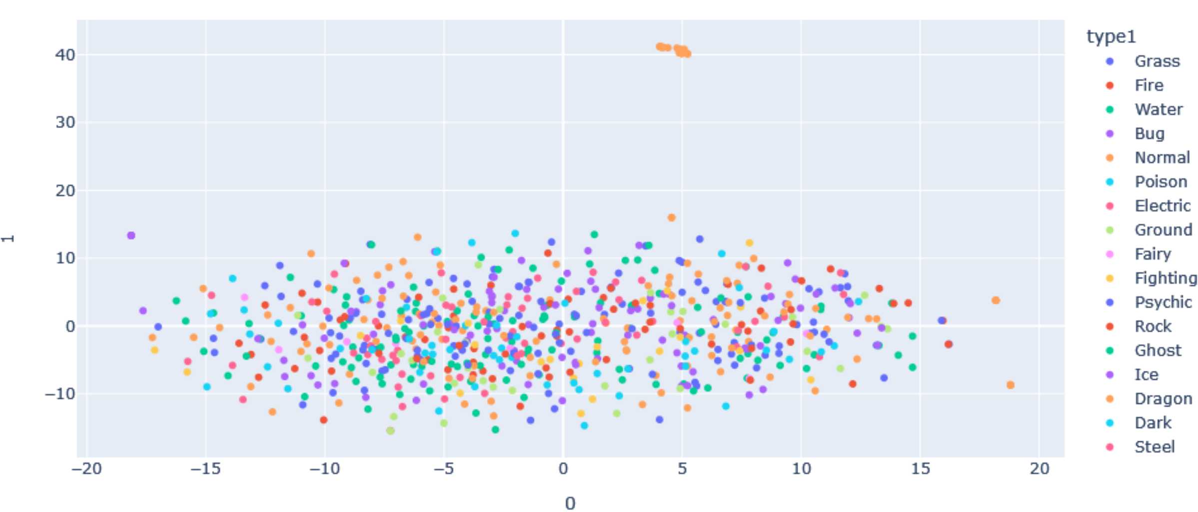

Figure D.6: t-SNE Plot of the latent space of the Pokémon dataset in the Pixel VQ-VAE MedRes.

Appendix E

Team-building Algorithm: Grid Search Results

As discussed in Section 4.3.2, we perform a grid search across 4 components to determine the best team-building algorithm for the ABC-Meta task. These are the pickrate, the BST, The Meta Type Value and the Type Value. We use the following basic algorithm:

$$\text{Score}_{ABCMeta} = \text{Pickrate} * (\text{c1} * \text{ BST } + \text{c2} * \text{ Meta Type Value } + \text{c3} * \text{ Type Value }) \tag{E.1}$$

This function has two component terms. The first term, the pickrates, allows us to use the current metagame statistics to impact our team selection. These pickrates range from a value of 0 to 1, with a value of 1 indicating that every team uses a particular Pokémon. We weight this term by our second term, which combines domain knowledge (BST), meta knowledge (Meta Type Score) and team-building principles (Type Score). This second term also ranges from 0 to 1 but each sub-term is weighted differently.

We use weights of 0.50, 0.25, and 0.25 respectively for c1, c2, and c3. Since the BST is a strong indicator of a Pokémon's strength, we weigh it higher. The two type scores do not consider the strength of a Pokémon, just their types. Since they operate in similar areas, we weigh them equally. While we experimented with additional weighing schemes, our preliminary results indicated that these worked best.

For the purposes of this grid search, we use the Simple Self-Play PPO agent and evaluate using the same metrics discussed in Section 4.5.1. We consider only the Smogon OU scenario from Section 4.5 and run battles for a 3-month period, to a total of $450,000$ battles. We fix the usage of the pickrates and toggle the use of the remaining three components in all combinations. In total, we

compare eight approaches. Table E.1 depicts these results.

BST	Meta Type Value	Type Value	Edit Distance	Overlap
0	0	0	2.83	95.0%
1	0	0	1.09	97.5%
0	1	0	8.64	82.5%
0	0	1	0.18	92.5%
0.5	0.5	0	1.82	92.5%
0.5	0	0.5	1.58	92.5%
0	0.5	0.5	2.33	90.0%
0.5	0.25	0.25	0.85	92.5%

Table E.1: Results of performing a grid search over the three domain knowledge components of our team-building algorithm.

Examining the results, we see that the use of the BST has a noticeable impact on the Overlap, achieving a minimum of 92.5% Overlap and around 1 in Edit Distances. Using the Meta Type Value on the other hand has a significant negative impact on both metrics, with an Overlap of only 82.5%. This seems to be true even when it is used in conjunction with the other components, with the versions using the Meta Type Value performing worse than the ones that do not. Using only the Type Values has the closest Edit Distance, but as the Overlap shows, it does not accurately predict the overlap meta. Using only the pickrate (the first row) achieves slightly higher overlap at the cost of a much higher Edit Distance. Overall, the version which uses only the pickrate and the BST seems the best choice. It has the highest Overlap in the table and does reasonably well in terms of Edit Distances.

Appendix F

Sample Outputs for the Pixel VQ-VAE Downsteam Tasks

We include a selection of randomly sampled images generated by the models described in the image generation section of the main paper. All VQ-VAEs use a PixelCNN for generation and all models were trained on the same dataset. Note that the VQ-VAE models learned to mask out the noise while the GAN does not. All generations are sampled from 10,000 generated images.

Sample Pokemon

Figure F.1: Sample images from our Pokémon dataset after pre-processing.

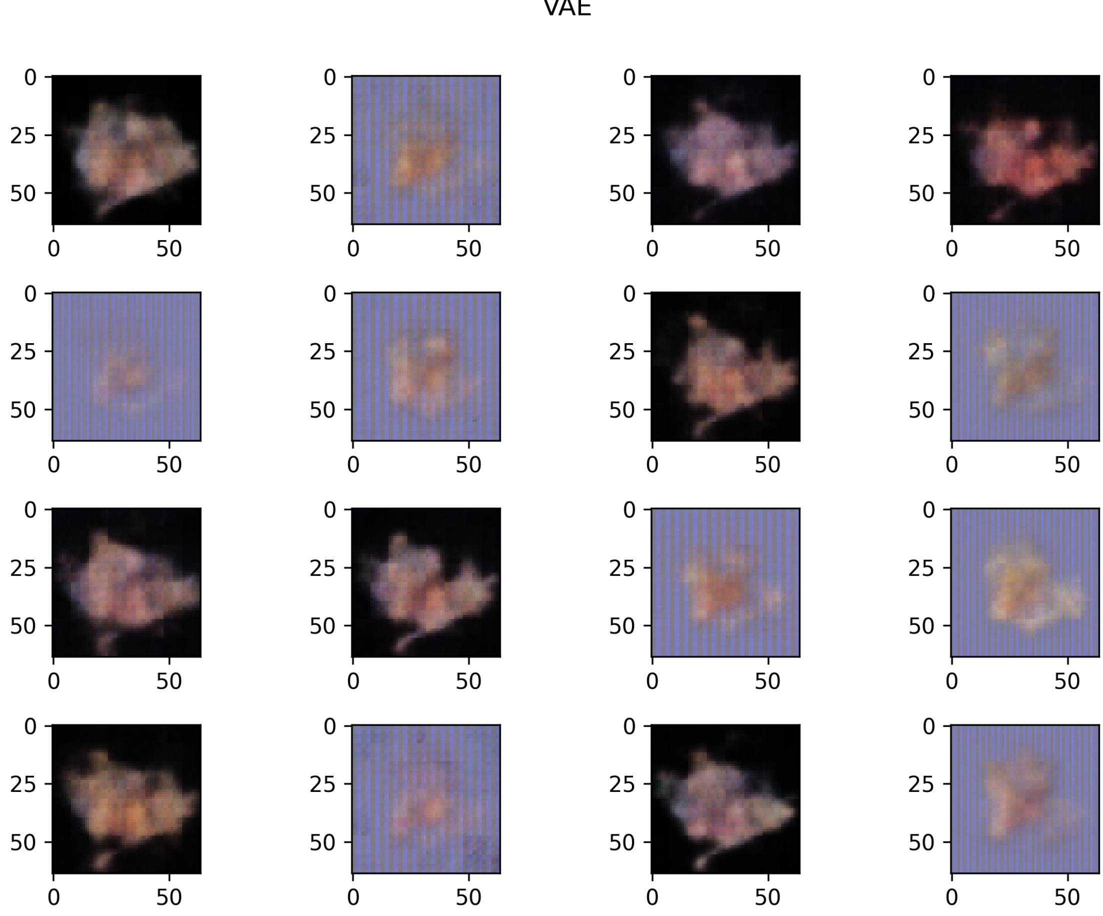

Figure F.2: Sample generated Pokémon from our VAE baseline.

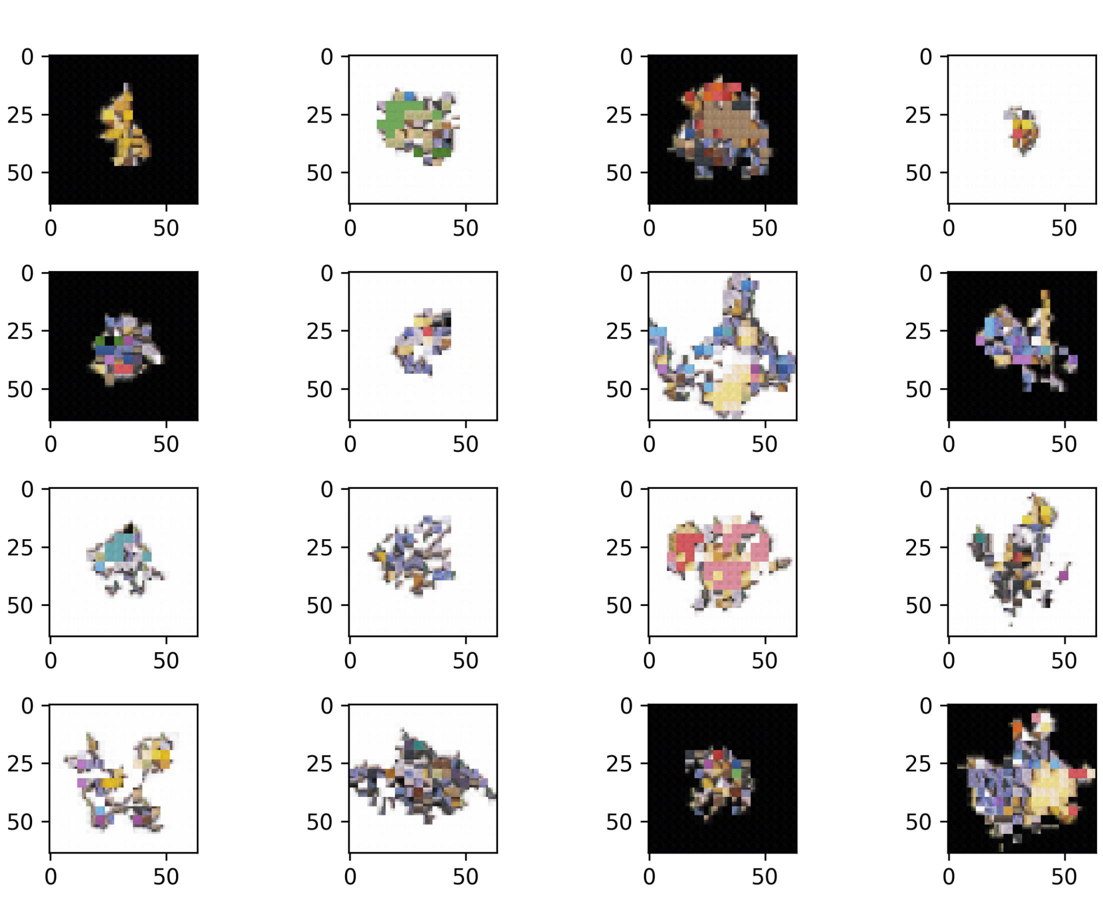

Figure F.3: Sample generated Pokémon from our LowRes VQ-VAE baseline.

Pixel VQ-VAE LowRes

Figure F.4: Sample generated Pokémon from our LowRes Pixel VQ-VAE.

VQ-VAE MedRes

Figure F.5: Sample generated Pokémon from our MedRes VQ-VAE baseline.

Pixel VQ-VAE MedRes

Figure F.6: Sample generated Pokémon from our MedRes Pixel VQ-VAE.

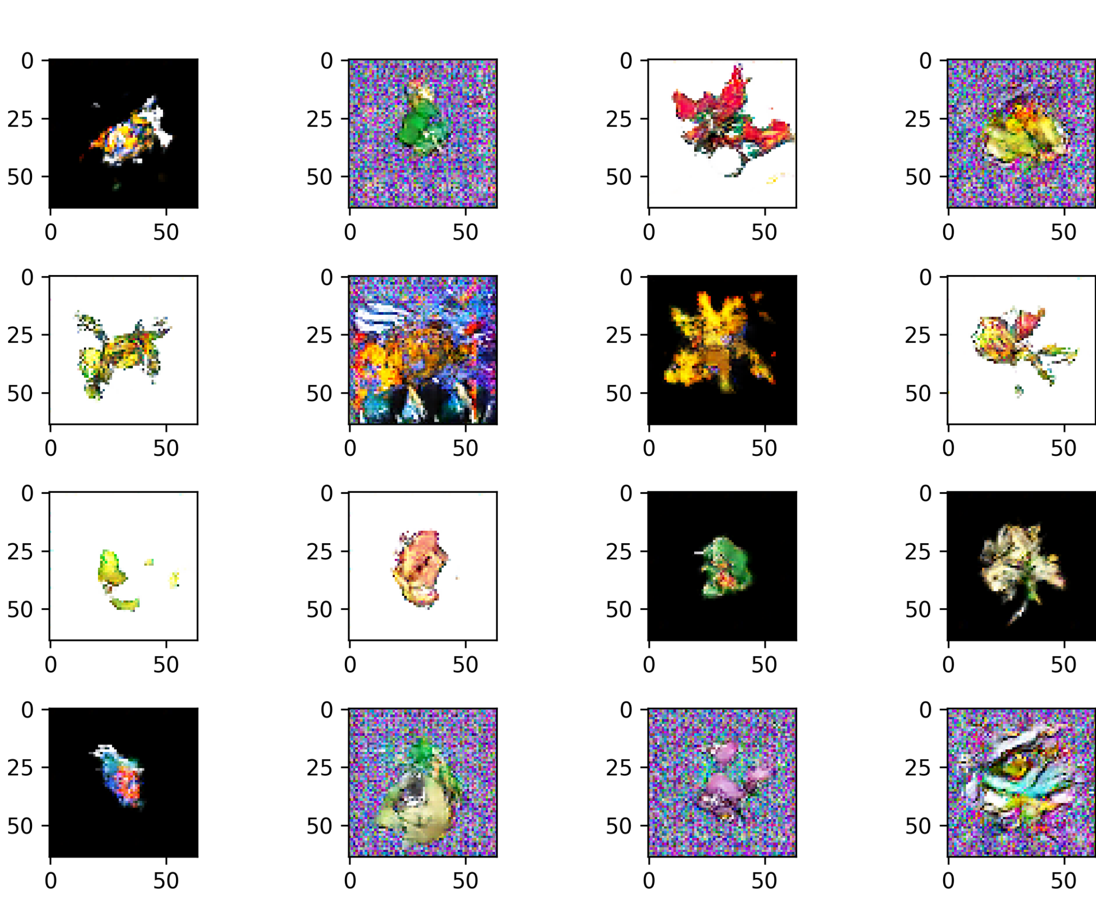

Figure F.7: Sample generated Pokémon from our GAN baseline.

Pixel VQ-VAE HiRes

Figure F.8: Sample generated Pokémon from our HiRes Pixel VQ-VAE.