# Enhancing Hearthstone deck building with a Generative Adversarial Network (GAN)

Callum Roberts

April 17, 2021

**Abstract**

## Acknowledgements

I would like to thank, in particular, my project supervisor Mark Bartlett, whose wisdom proved invaluable to the completion of this project. This thanks extends to the rest of the RGU staff who went beyond expectations in their support during these tough times.

Thanks are also owed to the Hearthpwn team who allowed me to collect their data and to the HearthSim team for their simulator to run the tests.

# Contents

# 1. Introduction

## 1.1 Overview

Bringing Artificial Intelligence to Hearthstone: Heroes of Warcraft is not a new concept, it has been subject to many studies in the field, where it be playing the game, suggesting moves to the player or building decks. Focusing on the field of deck building, it is a concept that exists across multiple trading card games such as Magic: The Gathering, Pokemon, Yu-Gi-Oh! and of course Hearthstone. Papers have studied these games more or less, but all seem to gravitate around similar deck building techniques, details of which will be expanded later on. These techniques seemingly been saturated, newer studies apply small scale changes to already defined methods [1]. Exploration of different techniques has been touched on but they seem to be outliers and of limited number [2], experimenting with other algorithms could show better, more interesting results instead of assuming that one saturated method is the best because of its popular usage.

## 1.2 Motivation

Using a Artificial Intelligence algorithm that has not been utilised nor researched in the field of deck building could provide insight into the Generative Adversarial Network (GAN) applications in other fields of research, as it is mostly limited to image generation. Along with this discovering another method for deck building that could bear fruit to similar or improved results, opening the way for more techniques and deeper research into the use of recorded data for deck generation.

## 1.3 Aims and Objectives

The aim of this project is to create a deck building artificial intelligence using a Generative Adversarial Network (GAN) and testing the viability of it, this will be achieved by completing the following objectives:

- Collecting user deck data from reliable sources

- Cleansing of user deck data
- Converting user deck data to vectors and back to human readable after training
- Implement GAN for single dimensional vector generation
- GAN hyper-parameter and layer optimization
- Simulating results against user decks
- Performing evaluations on resulting decks

## 1.4 Key Findings

This project has exhibited that it is possible to use Generative Adversarial Networks to generate decks for Hearthstone, a technique that seemingly no one has attempted to use, the AI demonstrates similarities to user created decks such as card types, type percentage, card duplicates, synergies and card spread. The decks tested have an average of 55% win rate, varying from class to class, match ups against other classes greatly influenced the outcome of a match. Although the win rate is heavily influenced by the AI playing it, considering that the simulator game playing AI will not play as well as a human player. The training process is much faster than anticipated (around 10 minutes), however the testing was much longer at around 1 - 2 hours depending of the number of decks created.

## 1.5 Structure

The structure of the report is as follows, first a literature review which presents an overview of Hearthstone, some of the important game mechanics and existing AI that have been used. Then a discussion of the requirements of the project and the methodology to implement them. Following this is a discussion of the potential legal and ethical issues with the project and what has been done to overcome them. The implementation of the project, a deep dive into how the project was completed. A testing and evaluation section describing the strategy and the results of the generated decks. Finally, a conclusion which summarises the project is given.

# 2. Literature Review

The purpose of this literature review is to define the technologies used in the field of artificial intelligence for building a deck in Hearthstone. To introduce, and compare previous works to determine their strengths and weaknesses. A review of the literature is valuable in understanding important aspects of a research area [3]. The structure of this literature review is as follows: the initial section will detail the background of the project, explaining the fundamentals of Hearthstone and deck building. Following that will be an introduction to the project, motivations, and research questions. Finally, we will have the core technologies used for similar projects.

## 2.1 Background

For the benefit of the reader, this section will introduce the basics of Hearthstone. It will emphasize the deck building aspect of the game including practices used by players.

### 2.1.1 Collectible Card Games

Collectible Card Games (CCG)[1] are a sub-genre of card games introduced in 1993 by Magic: The Gathering [2]. They require players to make a custom decks to play, they mix trading cards with strategy and deck building features. CCGs are usually defined as a turn-based game, where each player acquires their own collection of cards through the purchasing of "starter decks" for beginners or "booster packs" containing a small number of random cards from a *pool* of cards usually referred to as an expansion. The aim is to build an efficient deck that can account for the inconsistency that comes from the nature of card games, to predict and play around your opponent's actions to ultimately beat them. Some CCGs can prove to be lucrative for players as cards have a value intrinsic to their rarity and demand [3], this makes building the perfect deck rather difficult and usually costly.

[1] https://en.wikipedia.org/wiki/Collectible_card_game
[2] https://en.wikipedia.org/wiki/Magic:_The_Gathering
[3] https://www.cardmarket.com/en/Magic/

### 2.1.2 Hearthstone

Hearthstone: Heroes of Warcraft is a CCG developed by Blizzard Entertainment in 2013 [4], but with the twist of it being entirely digital, there is no physical version of the game. This choice unlocks potential for gameplay features that could not be implemented, in exchange for the tradability of cards.


Figure 2.1: Example of a Legendary Hearthstone card[4]

Two players face off wielding each a deck of their own making. Decks consist of exactly 30 cards. Players then take it in turns to play their cards, the objective being to reduce the other player's health to zero. On each player's turns that player draws a card and gains a "Mana Crystal" up to a maximum of 10 (crystals are refreshed every turn), these crystals are expended to cast a card from the player's hand. Before a match each player chooses to embody a class (such as Mage, Warrior, Rogue, Druid, etc...), each class has specific cards only they can add into their deck, these are adequately named "Class Cards", these are accompanied by "Neutral Cards" that any class can use. Classes also have access to an ability unique to them called a "hero power".

Each card in the game has a "mana" cost which shows how many mana crystals are needed to cast that card. They also have a card type, rarity, and an effect. In a deck, players can put duplicates of the same card (up to 2) except for "Legendary Cards" (figure 2.1) that are limited to a single copy due to their powerful effects. Players have a "Collection", where the cards they own are stored, to get new cards players can buy card packs with gold, the in-game currency of Hearthstone. Gold is earned slowly through quests, winning, and events, however this process can be sped up through the purchase of gold with real-life currency. Players can also choose to "Disenchant" their duplicate cards to gain another in-game currency called "Dust" which can be used to create a

[4] https://www.pinterest.fr/pin/573716440004576557/

[5] https://bothgunsblazingblog.wordpress.com/2014/06/22/hearthstone-analysis-and-deconstruction/


Figure 2.2: Example of a Hearthstone board[5]

card of the players choosing[6].

### 2.1.3 Deck Building

In the world of CCGs, there is a long-standing debate on how to measure the skill of a player. Although card games involve luck and circumstance, it is believed that there is a degree of strategy in the building and execution of decks whether it is just a slight increase in win probability or a fundamental to winning [5]. However, the debate stems from which is the most important, the building aspect or the execution aspect of CCGs. [6]

#### 2.1.3.1 Metagame

Hearthstone is a game with lots of complex systems that are influenced by many factors, mainly due to a large number of cards and different playable heroes. In a game where there are lots of variables, players try to rank cards, heroes, and combinations to increase their chances to win. This phenomenon creates decks from a "pool" of top-rated cards, leaving out the mediocre, forcing players to use these top-rated decks in order to have a better chance of winning or be put at a disadvantage. The result is what is called the "Metagame" or *meta* for short[7]. Blizzard release updates to the game frequently through "expansions"[8] which add a variety of new cards to keep the game fresh. Shifts in the meta occur when these expansions are added and players experiment to

[6] `https://hearthstone.gamepedia.com/Crafting`
[7] `https://www.hearthstonetopdecks.com/hearthstones-best-standard-ladder-decks/`
[8] `https://hearthstone.gamepedia.com/Expansion`

find better combinations over time. However better cards may not be added, and changes in the meta may not occur, this dissuades players from continuing or returning to play knowing that they have already experienced all that they can. To avoid that Hearthstone has implemented two-game modes[9], one in which only cards added over the past two years are available, and another mode that allows all cards. Whilst this method has helped, it still does not put a stop to the possibility of a stale meta. Researchers have done studies on how to evolve the meta through AI means, by *balancing* powerful cards (Fernando et al.) [7]. Balancing a card means to adjust the power of said card to make it more or less viable in the current game environment. Fernando et al. discussed the idea that around 50% of Hearthstone's meta is derived from match-ups which is the win probability two decks have against each other, a favorable match-up being the one with the highest win probability or known in the community as *win rate*.

#### 2.1.3.2 Mana Curve

Theorycrafting is a term used widely in many video games, it designates a mathematical analysis of a game's core mechanics to attempt to discover new strategies or combinations that could rival the current ones. Hearthstone is one such game, a large portion of the player base enjoys theorycrafting new decks that may *break* the meta, as in cause a fundamental shift of the current metagame. These players rely on fundamentals or schematics that are used as a guideline in building a new deck. The *mana curve* [10] is one such fundamental, it exists in all decks built in the game. Every card in Hearthstone has a cost, this cost determines the power of the card, a low-cost card will be weaker than a higher cost card since it would cost fewer resources to cast. This *mana curve* is a histogram of each card plotted by cost, it allows players to visualize how expensive in resources their deck is and to determine the deck's archetype.


Figure 2.3: Example of a Mana Curve[11]

[9] https://hearthstone.gamepedia.com/Game_format
[10] http://hearthstone.gamepedia.com/Mana_curve
[11] https://hearthstone.judgehype.com/deck-mage-tempo-ladder-legendaire-tgt-gvg/

#### 2.1.3.3 Archetype

The word archetype is derived from the Greek word *archétypon* which means "beginning, origin", applied in the psychology field to categorize complex human behaviour called "Jungian Archetypes" [8]. This term was transposed into the deck building field of CCGs, a deck's archetype is meant to categorize and describe the behaviour of the deck from a high-level perspective, forgoing the need to play the deck to learn its strategy. In Hearthstone, most decks can be categorized by three main archetypes [9]:

- *Aggro* decks are the aggressive decks meant to defeat an opponent as quickly as possible, as a consequence the mana curve of such decks is focused towards the cheaper side of the histogram. Their power comes in the early turns, but they quickly become weaker to the other archetypes as the turns go on.
- *Control* decks are meant to control the state of the board through the use of expensive cards, they tend to generate a lot of cards and have a wide range of card choices. The mana curve of such an archetype is towards the expensive end of the histogram. They tend to have a few cards to play early in the game but have a multitude of win conditions in later turns.
- *Mid-range* decks are situated in the middle of the two other archetypes, focusing mainly on the mid-game, their win conditions are stronger than the aggro decks but weaker than the control decks. Their mana curve peaks in the middle of the histogram.

Hearthstone also has other archetypes that cover a shorter scale, they are usually introduced in the newer expansions and rotate out of the normal game after a couple of years [10]. For example, a highlander archetype is a deck with 30 unique cards. Archetypes are formed around specific cards with win conditions[12], meaning that they have the power to win the game. So in the example given in a highlander deck, there would be a card that has an effect that triggers from having only unique copies in the deck.

#### 2.1.3.4 Resource Cost

In CCGs, cards have a certain cost to use, in Hearthstone that cost is mana which regenerates every turn. The cost of a card is determined by the power of said card, if it has a powerful effect, has decent attack and defence values, or even both. This cost will determine how late into the game a card can be played. However, a card that costs a lot can be considered weak and a low-cost card can be considered strong. The power of a card is determined through the resource cost, an invisible value that is hard to calculate and a subject of study [11]. Although the Zuin et al. study was used to predict the cost of a card in Magic: The Gathering, the resource calculation is still present in Hearthstone. It poses

[12] `https://playhearthstone.com/en-us/news/21363038`

a good solution to the balancing of the metagame and would be adaptable to Hearthstone. A card is considered efficient if the theoretical resource cost is higher than the current mana cost, and would be inefficient if the resource cost were to be lower than the mana cost. The resource cost of a card is something that may need to be considered when developing an AI for building decks. Stiegler et al. [12] applied a similar theory to design a deck-building AI based on a utility system that classified cards based on resource cost-effectiveness, mana curve, and synergies.

## 2.2 Project Introduction

The rising popularity of Hearthstone has attracted a lot of new players reaching over 100 million accounts in 2018 [13], however, due to the nature of collecting cards in the game some will not have the required cards to build the most popular decks. The game is free, anyone can download it, however, a lot of the content is locked behind a paywall[13]. Whilst it is possible to earn cards by earning "gold", it becomes a time-consuming ordeal that requires a lot of spare time to invest. With new expansions being added regularly, the game seems to become a never-ending grind, unless you decide to pay real money to acquire currency. This is where the term "pay-to-win" is used to describe Hearthstone [14], meaning that to get the most enjoyment and the highest chance to win, the player must spend money or be disadvantaged. The goal of this honours project is to create an AI that builds decks from a collection of cards, incomplete or otherwise in order to improve the game experience for players that are unable or not willing to pay. For players that do own a large collection of cards, it can also provide fresh new decks to play that differ from the more popular ones.

### 2.2.1 Related Works

Video games are the ideal tool for the training of Artificial Intelligence. The virtual space that a game provides is a realistic environment with a limited amount of information available [15] allowing control and knowledge over the behaviour of the AI. Hearthstone is a game that provides a platform for a wide variety of AI that differs from AI-benchmark games such as Chess or Go. Hoover et al. [16] classifies Hearthstone AI into specialized categories:

- Game Playing AI, rather self-explanatory, this form of AI is designed to play the game. Generally, tree search algorithms are used, Monte Carlo Tree Search (MTCS) in particular. However, this method is rather ineffective in Hearthstone due to the amount of hidden information and limited visibility of the AI. Developers of Hearthstone simulators, such as *MetaStone*[14] tend to use a greedy approach to compensate [17]. Some researchers attempt to use variations of MCTS and heuristics to work around the limited information [18][19][20].

[13] `https://en.wikipedia.org/wiki/Paywall`
[14] `http://www.demilich.net/`

- Developer Assisting AI, this AI help with certain issues that developers could have. Since Hearthstone has hundreds of cards, it is challenging to design cards with new flavour that are not identical to previously printed cards. Could there be a way to generate inspiration? Woolf "minimaxir" Max[15] created an API that generates Magic: The Gathering cards[16] using a transformer language model[17] for such a purpose. Another possible use is for balancing the game, since maintaining game balance when creating additional cards may create unfair combinations, or render some cards useless[7].

- Deck Building AI, these create decks for the player or another AI to use, most commonly created with Evolutionary Algorithms[1]. It has the inherent advantage of being usable in conjunction with other AI. Such combinations help ascertain potential balance issues without human bias involved. Since this is the main topic of this paper, a further in-depth explanation will be provided in the body.

Whilst all these AI are used in the context of Hearthstone, they are utilized for different aspects of the game, therefore, proving that Hearthstone is a platform with a constant influx of AI challenges to be met, a prime example is the additional *battlegrounds* gamemode[18], a variation of the game where the creatures attack on their own automatically, then completing your board as you progress between rounds. This alteration of the way the game is played will surely become the subject of a paper in the future.

### 2.2.2 Research Questions

Research Questions are essential to any methodical research, it is the first step in any project and fundamental to any successful project. Kowalczyk[21] described Research Questions as a metaphor for a house: "Your data collection forms the walls and your hypothesis that guides your data collection is the foundation. So, what is the research question? It is the ground beneath the foundation. It is what everything in a research project is built on. Without a question, you can't have a hypothesis. Without the hypothesis, you won't know how to study what you're interested in." The research questions in this literature review are defined as:

- **RQ1:** What are the current best deck-building techniques in Hearthstone?

- **RQ2:** What are the strengths and weaknesses of the different techniques?

- **RQ3:** With our findings, what techniques can be applied to optimize the deck-building problem in Hearthstone?

[15] `https://minimaxir.com/apps/gpt2-mtg/`

[16] For example cards: `https://github.com/minimaxir/mtg-gpt-2-cloud-run/tree/master/generated_card_dumps`

[17] `https://openai.com/blog/tags/gpt-2/`

[18] `https://hearthstone.gamepedia.com/Battlegrounds`

### 2.2.3 Assistance Systems

Despite AI being widely used in Hearthstone for research purposes, it is against Blizzard's Terms of Service (ToS) to use game-playing AI in Hearthstone (Section 1.C.II)[19]. However, a surge of Hearthstone deck tracking software [20] is being used by players without being banned. So how do players use this kind of software without violating ToS? It was revealed that turning on debug logs would provide enough information for these systems without breaking ToS [22] which birthed a whole sub-genre of AI coined as "Assistance Systems" designed to be used to assist the player without it being considered cheating. This brought on the creation of deck trackers, which mentioned above track which cards each player has used and tracks statistics. Whilst deck trackers are not AI since they just read logs, Bursztein [23] used this system to create an AI that predicted what cards the opponent would play in future turns, and used this predictor AI to climb to *legend* rank (the highest rank in competitive mode[21]). While it was not against ToS to use it, when they presented the tool, Blizzard reached out to them and asked them not to release the code as it was *game breaking*. The effectiveness of the tool was however limited to later turns, the accuracy is much lower (going as low as 50%) in first turns and becomes more accurate each turn. Some of the most crucial turns for some archetypes are in those early turns, so this AI would only maximise effectiveness for the *Control* Archetype (2.1.3.3).

Other assistance algorithms include Hearthstones Arena game mode[22], a mode in which the player drafts a deck one card at a time by selecting 1 of 3 possible choices from a pool of cards, using Apriori algorithms [24] such as HearthArena[23] to make suggestions based on data from a diverse range of high-quality decks created by player and/or deck building algorithms[25]. However, this sort of algorithm would only be useful in an environment where the player cannot select their cards.

The use of assistance systems is interesting but still requires the interactivity of a third party to function. The advantage to assistance systems is the ethical and legal implementation from it. The disadvantages such as the low accuracy rate of the prediction tool in earlier stages of the game, or the limited usability of the Apriori algorithm makes it difficult to be used in a standard deck-building format.

[19] `https://www.blizzard.com/en-us/legal/fba4d00f-c7e4-4883-b8b9-1b4500a402ea/blizzard-end-user-license-agreement`

[20] Some software examples: `https://hsreplay.net/downloads/?hl=en`
`https://go.overwolf.com/firestone-app/`
`http://hearthstonetracker.com/`
`https://trackobot.com/`

[21] `https://hearthstone.gamepedia.com/Ranked`

[22] `https://hearthstone.gamepedia.com/Arena`

[23] `https://www.heartharena.com/`

## 2.3 Genetic Algorithm

Developed AI algorithms often draw inspiration from biology[26], Genetic Algorithms (GA) is an example of this. GAs are a subset of Evolutionary Algorithms (EA) that base their training process the same way nature does, biological evolution through natural selection (Figure 2.5). A population of solutions each with a set of properties (chromosomes in nature) that are mutable is randomly generated. Each iteration or *generation* the algorithm selects the fittest individuals of the population using a fitness function, then the most fit are used to form the next generation. Some mutations of properties may occur in some of the population. The algorithm ends when either the correct fitness level is achieved or when the number of set generations is reached[27]. GAs are stochastic in nature, meaning that a single iteration would not be sufficient to provide significant statistical results [28]. The process is reminiscent of Charles Darwin's theory of evolution[24] and proves to be effective in optimization problems[26]. This algorithm is the most frequently used for deck building

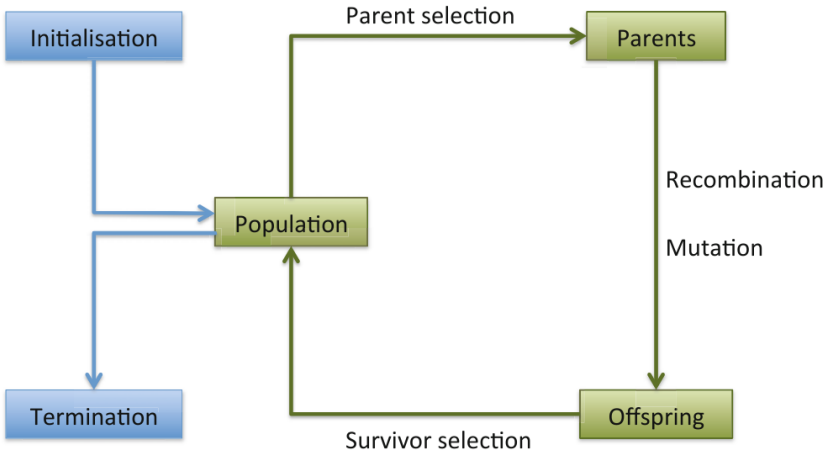



Figure 2.4: Evolutionary Algorithm Flowchart [26]

problems [29][30][31], although they are optimized in different ways. In GAs, there exists a fitness function that determines the fitness score of an individual that is used to create the next generation, and there is the mutation function which will randomly mutate some individuals (it may or may not improve the fitness of said individual).

Bjørke and Fludal [29] used a genetic algorithm to construct decks for Magic: The Gathering based on a certain pool of cards. Instead of using a fitness function that would calculate the score of a deck, they pitted each deck in that gener-

[24] https://en.wikipedia.org/wiki/Natural_selection

ation against each other in a tournament format using a Magic: The Gathering game simulator, each deck had the same number of games to play and they would select the fittest decks based of their win rate in the tournament. While theoretically, the idea is sound, the time that it took was substantial for an unremarkable win rate (less than 60%). With 50 matches per deck over 350 generations, it took 43 hours to execute. Due to the time it takes, it would be unusable for players as it would simply take too long.

Garcia-Sanchez et al. had a similar approach using lexicographical fitness with a Hearthstone game simulator called *MetaStone*[25], separating the fitness evaluation into three parts: one part counted the number of victories of in 16 games, another part which determined the deck correctness (no more than 2 duplicates, only 1 legendary, etc...), and the last part was applying standard deviation to the number of victories, it being optimal if the deck won against every opponent [30]. The results were achieved faster much faster and were of a better standard than Bjørke and Fludal's work[29], however, the results were not as high as they could have been, mainly due to the fitness function using *MetaStone's* greedy AI to play. The decisions made by the AI would be greedy and different from that of a human player. The mutation function could also have been touched on, allowing the mutation to make smarter decisions about which cards to mutate.

Garcia-Sanchez et al. tackled the problem once again and touched on the mutation function [31], they developed a *smart mutation* function that would replace a card in a deck with another of a similar cost (roughly $\pm$ 1). The results with the smart mutation were overall better than without it. This may solve one of the weaknesses of their previous work[30] but still uses the same greedy simulator heuristic.

## 2.4 Machine Learning

Machine Learning is a subset of Artificial Intelligence that constructs systems that can learn and improve without the need to be explicitly programmed. Burkov described it as "a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon." [32]

This section of the review will present Machine Learning techniques that have been used or could possibly be used for deck building.

### 2.4.1 Artificial Neural Network

Artificial Neural Networks (ANN) also known as Neural Networks (NN) for short is an algorithm that was inspired by the biological neural networks seen in brains. They are a group of interconnected nodes or *neurons* typically organised into multiple layers, an input layer, an output and $n$ amount of hidden layers. Through many loops known as *epochs*, the NN trains itself using data

[25] `https://github.com/demilich1/metastone`

from datasets to ultimately make a prediction based on given data as an input. Each node is weighted, and these weights are updated throughout the training process increasing the weight of positive output and decreasing on a negative one, allowing the system to make more accurate predictions [33]

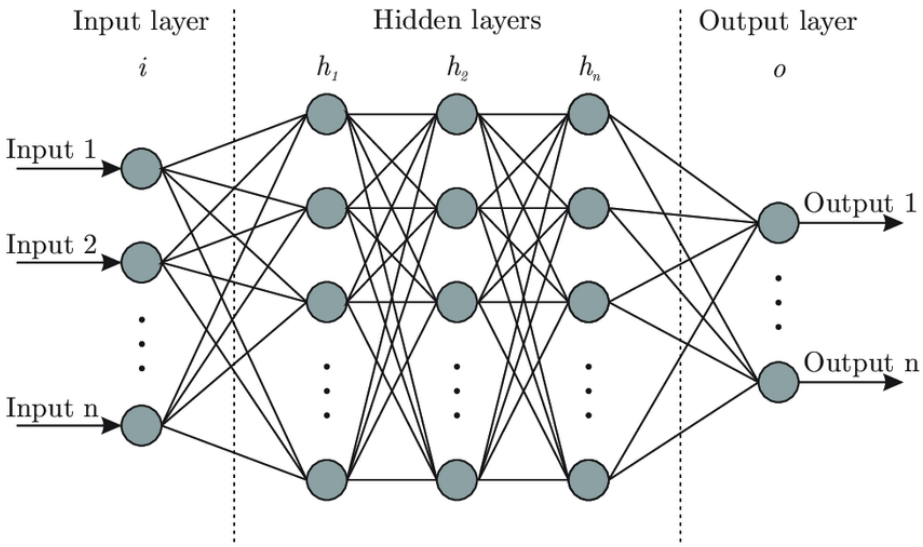



Figure 2.5: Artificial Neural Network architecture [34]

The approach to developing a deck-building AI is done differently than a GA. Ward et al. created a NN that emulated the choices a human would make when drafting a deck from a pool of cards for Magic: The Gathering [2]. The idea was to select the cards that were chosen by a human in the dataset (target variable). The resulting accuracy on the test set was 65.7%. The main drawback to this technique is that it requires the data to be clean, since the dataset used was a lot of decks drafted by human players, the data within may be suboptimal or purposefully tampered with, which would cause the AI to build weaker decks.

Jakubik attempted a different method by using a NN to predict the win rate of a Hearthstone deck learned by using the results of observed matches [35]. This was a proposed solution to the AAIA'18 data mining challenge and came second. However, Jakubik's solution was subject to over-fitting, which Hieu Vu et al. tackled during the same challenge, which brought them the winning solution to the AAIA'18 data mining challenge[36]. They tackled over-fitting by doubling the number of input features (or nodes) to account for the opposing player, they added an extra hidden layer and more test cases to compensate. Jakubik [35] and Hieu Vu [36] rely on data of games that have already been played meaning that this method is not suitable for the construction of the decks rather the prediction of the win rate based on games that have been played. Ward et al. paper is the better technique of the three papers to build good decks, however,

it requires clean and uncorrupted data.

### 2.4.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) is an application of generative modeling originally described in 2014 by Goodfellow[37], which is an unsupervised learning task that automatically discovers patterns in the input data that the model can then use to output new examples that could have been drawn from the original data. A GAN uses a generative model to generate a similar output and a discriminator model uses supervised learning techniques to determine the probability of the generative output to be part of the training data[37]. The goal of the generative model or *generator* is to create an output that can trick the discriminator into thinking it was part of the dataset. Goodfellow et al. use a fake money analogy: "We can think of the generator as being like a counterfeiter, trying to make fake money, and the discriminator as being like police, trying to allow legitimate money and catch counterfeit money"[38]. The GAN trains itself like this until the generator can create something that tricks the discriminator. Figure 1.6 shows the architecture of a GAN

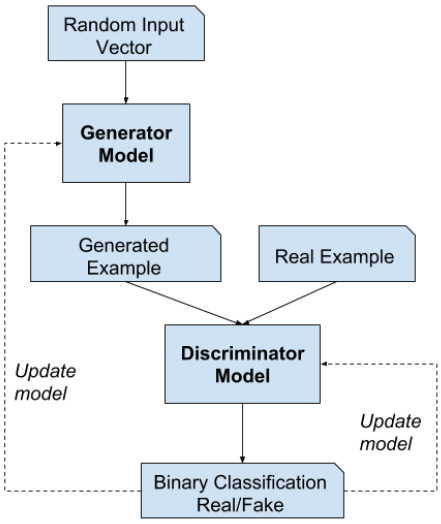



Figure 2.6: Generative Adversarial Network architecture[26]

As of writing this, there have not been any applications of a GAN for building decks. However, Rodriguez Torrado et al. combined a GAN with a procedural content generator to create different playable levels that were randomly generated from the generative model[39]. The GAN struggled to make playable and unique levels when provided a small amount of data. With this paper, it

[26] https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

is theoretically possible to apply a GAN in a deck-building context but it will require a large amount of data to provide malleable results.

## 2.5 Hybrids

In AI, aspects of a problem to be solved can be optimised through the use of multiple AI. This section of the review discusses papers that have used a combination of machine learning and evolutionary algorithms. The reason why GAs are combined with Machine Learning techniques is that GAs do not require lots of data to function, this allows for uses such as parameter optimization, Sehgal et al.[40] uses GAs to speed up the learning process of their Deep Reinforcement Learning (RL) agent and provided the maximum success rate faster. Such et al. [41] used it for similar work, and stated that is was a competitive alternative to popular RL training techniques. It has also been observed that in the chemical field that the GA could outperform generative models in a discriminator neural network [42], exploring the possibility of an application in GANs. The use of genetic algorithms in hybrid neural networks is mainly for achieving results faster [40][41], however in highly parametrized systems the GA tends to struggle to find the optimal combination. [43]

## 2.6 Conclusion & Final Thoughts

This literature review has covered existing deck building solutions from Hearthstone and Magic: The Gathering, including evolutionary algorithms, neural networks, and various assistance systems. Potential solutions such as hybrid algorithms and Generative Adversarial Algorithms have also been discussed. Based on this literature review, it has been determined that evolutionary algorithms are the most popular method of tackling the problem therefore the most saturated of new solutions to explore. Machine Learning is a more obscure usage, focusing mainly on win rate prediction and card ranking. Hybrid algorithms and GANs have not yet been used in a deck-building context. In the context of GAs and NNs, they attempt to solve the deck building problem from different perspectives, the GA attempt to build a deck directly and test it in a simulated environment to determine the power of the deck in real-time, whereas the NN focuses on previous data that it uses to predict the power of the deck. They achieve the same goal, however, GAs are more direct and provided concrete results whereas the NN focuses on an indirect prediction from previous use cases. Moving forward this paper will explore solutions within those methods.

# 3. Methodology

## 3.1 Overview

The project implemented will be written in Python, in the form of three python notebooks, each with a specific task. The first notebook collects card data from an API and creates a card dataset, the second notebook scrapes deck data to create datasets for each class, and finally the last notebook will house the AI and the simulator testing. A user simply needs to run the notebooks to get their results, an input is required when deciding what class the user wants the deck to be. With the second notebook, additional datasets can be created and used in the place of the old ones for training the AI.

This project outputs JSON files for the card and deck data, one file for the card data and ten files for the deck data (one for each class). When executing the AI, the user is presented with a certain number of decks, the number varies on how well the GAN trained, these decks are tested and the user is shown graphs and charts of deck composition, overall winrate and winrate per class. They can see the cards within the deck to use at their own leisure.

## 3.2 Functional Requirements

a. AI **MUST** output a legal deck following the rules of Hearthstone. Meaning that a deck should be composed of 30 cards, with no more than 2 duplicates of any card (excluding legendaries, which can only have a single copy).

b. AI **MUST** output a deck following the rules of the "Standard" game mode. Meaning that the AI can only use the classic card set and the latest expansions from the past 2 years.

c. AI **MUST** use a generative adversarial network in order to generate a deck. The network will include a generator to create a similar and a discriminator to determine if the deck is part of the dataset.

d. AI **MUST** use a dataset of user created decks from the latest expansions in standard format (Rise of Shadows, Saviors of Uldum, Descent of Drag-

ons, Ashes of Outland, Scholomance Academy, Madness at the Darkmoon Faire) collected from Hearthpwn.

e. AI created decks **MUST** be tested in a simulator to output a predicted win-rate and determine the playability of the deck versus user created decks. A total of 16 games should be played for optimal results. This will allow for the completion of requirement "f".

f. The generated deck **SHOULD** have a calculated win-rate of over 50%. Ideally around 60%, so that the AI can make worthwhile decks. Meaning that the created decks should win at least half of the time, and in an ideal scenario they would win more often than lose.

g. AI **SHOULD** output a dataset of decks it generated following the same format as the training data, as to log the output and determine a possible bias.

h. AI **SHOULD** be able to generate a deck from a limited collection of cards, so it can be useful to players without access to all cards in Hearthstone.

i. AI **COULD** be used in the wild format, making use of all expansions to create competitive decks from a larger pool of cards.

j. An interface **COULD** be created to accommodate the AI and assist in feature implementation and accessibility of users.

## 3.3 Non-Functional

a. **MUST** have written consent from Hearthpwn to scrape data for dataset

b. Data collected **MUST** conform to GDPR regulations and be anonymous, since the identities of decks created by users is not needed.

c. **MUST** use Python as the programming languages of choice for the access to expansive libraries for data manipulation and AI creation.

d. **MUST** use Jupyter notebook as the software for coding the AI. It provides a platform for Python, and the cells are useful for making small changes without having to re-execute the whole program.

e. Dataset size **MUST** not be more than 100,000 rows as to decrease time to train and prevent overfitting.

f. The training process of the AI **SHOULD** take less than 3 hours, to allow a faster uptime if the AI needs to retrain after an update.

g. AI **SHOULD** use card data from the The Hearthstone API on RapidAPI if needed, to collected data such as card effects, images and tribal tags based of the name of the cards.

h. AI **SHOULD** use deck data scraped from Hearthpwn to train from using a coded web scraper.

i. Creation of a deck after training **SHOULD** not take more than 10 minutes. As to allow the to be quick to use by others whilst allowing for testing time with the simulator and the generation of graphs

j. **COULD** generate graphs of performance of decks outputted by the AI and simulators to visualise the training and testing process including the results of the created decks.

## 3.4 Approach

The project was designed using a agile methodology, equipped with logging, initially from a Trello board for the literature review and then migrated to github commit logs for the implementation of the project. The aim was to implement a requirement, testing it, fixing bugs and integration issues and adding additional features, this process was then repeated for each requirement until completion.

Testing was done on a per feature basis, once a feature was implemented, a series of tests examples were created to ensure the functionality of it, if any errors or problems arose the focus would be shifted to fixing that issue. Evaluation was done after testing, once deemed ready, the AI was executed 10 times to create decks for each class which were evaluated over 100 games across all 10 classes, graphs then display these results.

## 3.5 Libraries

This section describes what libraries have been used in this project, the aim is to show transparency and understand some more ambiguous parts of the implementation. Note that all of these libraries are Python libraries.

### 3.5.1 Web Scaper and API

#### 3.5.1.1 Requests

The requests library is a simple HTTP library that allows the user to send HTTP requests, whether it be GET or POST. This library was selected above others because over 500,000 others use this library aswell meaning that the library is stable and enables long term projects to stay active. This is used for both web scraping and api calls as both require to get a uri to get the information needed for the datasets.

#### 3.5.1.2 Json

The json library allows for the encoding and decoding of json data, this is used for creating the outputs of the API calls and Web scraping, it is also used as an input for loading these data files and processing the data.

#### 3.5.1.3 Beautiful Soup

Beautiful Soup is a library for web scraping, it pulls out data from HTML and XML files. There are three main libraries for web scraping, Selenium, Scrapy and Beautiful Soup. Selenium is mainly used for AJAX requests and Web Application testing, so it is not ideal for this project as the request library is being used to make http calls. Scrapy whilst having more features, and being faster than Beautiful Soup, it is meant for complex web systems with middleware which are not needed in the scope of my project. Beautiful Soup was chosen for its simplicity and flexibility on small scale projects, since this will mainly be used for scraping data to create datasets. Originally Beautiful Soup was used for the collection of card data, it is no longer used for that as the resulting output was missing crucial data such as class, cost and image links, so an API for collecting the card data was used instead, which allowed the data to be stored in JSON format.

### 3.5.2 GAN

#### 3.5.2.1 Keras

Keras is a deep learning API that runs using Tensorflow, a machine learning platform which focuses on enabling speedy experimenting, it is simple to use and very well documented making it easy to implement models. This is what was used for the creation of the Generative Adversarial Network, specifically the implementation of the generator and the discriminator.

#### 3.5.2.2 Numpy

Numpy is a python library designed for working with arrays, it aims to provide array objects that are 50x faster than standard python lists, since the GAN will be using a lot of arrays this is ideal for achieving faster training times. This library was used throughout the training process of the GAN, from sample generation to final deck results.

### 3.5.3 Evaluation

#### 3.5.3.1 Matplotlib

Matplotlib is a data visualization library which allows data to be displayed through graphs and charts, this is used throughout the evaluation process of the project.

#### 3.5.3.2 Time

Time is a python library that is used to measure execution time of functions, in this project it is used to calculate execution time of the GAN training and testing process.

#### 3.5.3.3 Fireplace

Fireplace is the python library for the Hearthstone simulator, this is used for testing the generated decks against ones from the dataset. There are two other simulators worth considering, Sabberstone and Hearthbreaker. Hearthbreaker was used by the Google Deep Mind team for card generation however the project is no longer maintained, it has been for many years so that one cannot be used to fulfil our requirements. Sabberstone is written has a full interface but for testing that creates unnecessary bloat, so Fireplace is the better choice, it is maintained, and plays games in seconds, the main downside however is that only about half of the cards in the game are implemented. This is because some cards have tricky mechanics that are hard to implement and that the developers are to write interactions manually.

# 4. Legal and Ethical Considerations

Since data is a fundamental aspect to the success of the project, it is subject to certain legal and ethical implications that need to be addressed whether it be with the handling of the data or the collection of the data.

## 4.1 Legal

There are two main legal consideration for this project, the protection of data through GDPR and obtaining data through web scraping.

### 4.1.1 Data Protection

When dealing with personal data, developers are bound by laws to keep that data safe, the first most obvious law that needs to abided by is GDPR, which applies to the project since we are using EU data (among other countries aswell), certain protections must guaranteed. Some ways of protecting personal data are, pseudonymisation and anonymisation, in essence the user's identity is protected by a pseudonym or with nothing at all which means that there will be no trace back to them, this could be represented as an ID or a username. If the data can be reversed back to personal data then data protection obligations will still apply, however if it is not possible then that data is no longer personal [44]. Data minimisation should also be practised when dealing with personal data, this means that data should be lawful, fair, transparent and it should only involve data that is necessary for the task at hand, no more. So in the case only the essential information has been collected (deck data) to train the AI, it is completely anonymous. Data minimisation also extends to how the data is used, it should only be used for its intended purpose only and not to be shared to others. One final important legal implication I need to consider is usage of childrens data, since anyone can use my website I also need to consider the possibility of minors using the web system. GDPR covers specific safeguards for children, which include parental/guardian consent to children under the age of 16.

### 4.1.2 Data Scraping

The laws surrounding data scraping is a grey area, whilst not technically illegal, there can be issues around collecting personal data from open tabs or sites that require a log in. To stay on the right side of the law, best practise is to check their Terms of Service or equivalent, worst case of a breach could be a court case so it is in the best interests of every party to verify before scraping. The site that needed to be scraped for this project was Hearthpwn[1], the data that needed to be scraped was public and pseudonymisation, since it was not behind a login page or any hidden pages it was not considered illegal. Before even writing the script, checking the Terms of Service of their site it said that data scraping was prohibited (Appendix A), this means that despite the data being public they do not want anyone using their data freely without permission. To get around this I sent an email to a representative of the site requesting permission to collect data for research purposes, after detailing the concept of the project and describing how the data would be used, the team at Hearthpwn granted me permission to scrape their site for data (Appendix B).

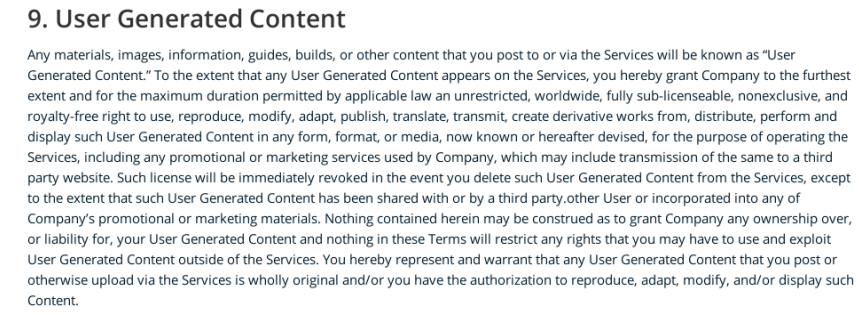
## 4.2 Ethical

This project raises certain ethical issues, raised from the collection of the user data, these issues are discussed below:

- Users may not know that their data is being used: Users may not be informed about their data being used to train the AI for this project.
- Users may not realise the scale of which their data is being collected: Users may not know that a web scraping script was used to mass collect deck data to create a training dataset.
- Users may not know how much information can be gained from analysing the data they have made available: Users may not realise that their data can be used to generate decks and data visualizations.
- Users may not realise the rights a company has regarding their public data: Users may not be informed about the amount of control the company has on their data.

These ethical issues concern mostly the users of the Hearthpwn platform, other data used in this project is not user related therefore are not concerned by these problems. The terms and conditions of the Hearthpwn site state that any User Generated Content can be used, reproduced, modified, adapted, published, translated, transmitted, create derivative works from, distributed, performed and displayed by the Company or third parties in any way that they choose so long as the data is not deleted.

[1] `https://www.hearthpwn.com/`



**9. User Generated Content**

Any materials, images, information, guides, builds, or other content that you post to or via the Services will be known as "User Generated Content." To the extent that any User Generated Content appears on the Services, you hereby grant Company to the furthest extent and for the maximum duration permitted by applicable law an unrestricted, worldwide, fully sub-licenseable, nonexclusive, and royalty-free right to use, reproduce, modify, adapt, publish, translate, transmit, create derivative works from, distribute, perform and display such User Generated Content in any form, format, or media, now known or hereafter devised, for the purpose of operating the Services, including any promotional or marketing services used by Company, which may include transmission of the same to a third party website. Such license will be immediately revoked in the event you delete such User Generated Content from the Services, except to the extent that such User Generated Content has been shared with or by a third party.other User or incorporated into any of Company's promotional or marketing materials. Nothing contained herein may be construed as to grant Company any ownership over, or liability for, your User Generated Content and nothing in these Terms will restrict any rights that you may have to use and exploit User Generated Content outside of the Services. You hereby represent and warrant that any User Generated Content that you post or otherwise upload via the Services is wholly original and/or you have the authorization to reproduce, adapt, modify, and/or display such Content.

Figure 4.1: User Generated Content section of Hearthpwn Terms of Service[2]

The users of the platform need to be made aware of this, some of the issues listed above can be addressed but not all issues have a full solution. The ethical issues mentioned above depends on the user awareness to the extend of which their data is being used and the impact it may have, however, it would prove difficult to communicated to the whole user base of Hearthpwn that their data is being used. The best course of action would be to either used an official training dataset that is not considered personal data to allow for more public use or limit the extent of this AI to research and academic purposes, the latter being the best solution at this current time as an updated public dataset does not exist as of the creation of this project.

[2] `https://www.magicfind.us/terms/`

# 5. Implementation

This section showcases the implementation steps of the project in three distinct sections: Card Data, Deck Data and Generative Adversarial Network. Each section will chronicle their respective development process, whilst discussing potential alternatives and limitations.

## 5.1 Card Data

### 5.1.1 Introduction

Before the development of the GAN can begin, data needs to be collected and formatted so that the GAN can be trained. The GAN can only take dimensional arrays as input, in the case of this project a vector, a single dimension array of numbers that identify specific cards to form a deck. From those deck vectors the GAN can learn, refine and generate new deck vectors that can be converted back into human readable form, resulting in a GAN created deck. In order to correctly identify the cards inside the deck vectors, going in and coming out of the GAN, a list of cards need to be compiled, each assigned with a unique numerical id that represents them.

### 5.1.2 Initial Method

#### 5.1.2.1 Text File

The initial method was to start of by creating a text file, this text file would contain a list of each cards that exists in the standard game mode of Hearthstone, each card would be separated by a line break within the text file. The aim is to make the file readable by a Python method, so that a dictionary can be created to identify the card by a integer key.

On the official Hearthstone website they have an online collection of every card in the game, these cards can be filtered and sorted by class, game mode, cost etc... This is a good source of information to gather the card data as it is official, reliable and up-to-date, this seemed useful as a list of card names could be compiled from scraping the site which could then be used to vectorize the cards for the GAN. The requirement for this project is that the GAN must

create standard decks from the past 2 years, so filtering to standard is necessary.


Figure 5.1: Hearthstone Collection Page[1]

Collecting the card data using the Beautiful Soup library proven futile, the site has a maximum card display setting meaning that only a small portion of the cards are displayed at one time, the rest are loaded by continuously scrolling to the bottom of the page. Due to the simple nature of the web scraper, it could not load the rest of the cards by scrolling (if the page was using different pages instead then it would have been possible to collect them), resulting in it only collecting a tiny portion of the standard cards. A solution was found by using a firefox web extension called "Open Web Scraper"[2].

Open Web Scraper is a scraper add-on that appends onto the web developer tools of a web browser. The user can then create a sitemap by entering a URL, then from this sitemap, the user can query information from the entered URL. After looking at the source code of the Hearthstone page, a query was written to get the name of every card on the page that was loaded beforehand with all the standard cards loaded. The query was `tr:nth-of-type(n)div.CardTableLayout__CardCropCell-sc-1jy3g9y-`, in short the query gets all names from the CardTableLayout table. The results are then highlighted in red on the page. These highlighted areas were then saved to a text file called cards-standard.txt (which can be view in Appendix C).

[1] `https://playhearthstone.com/en-us/cards`

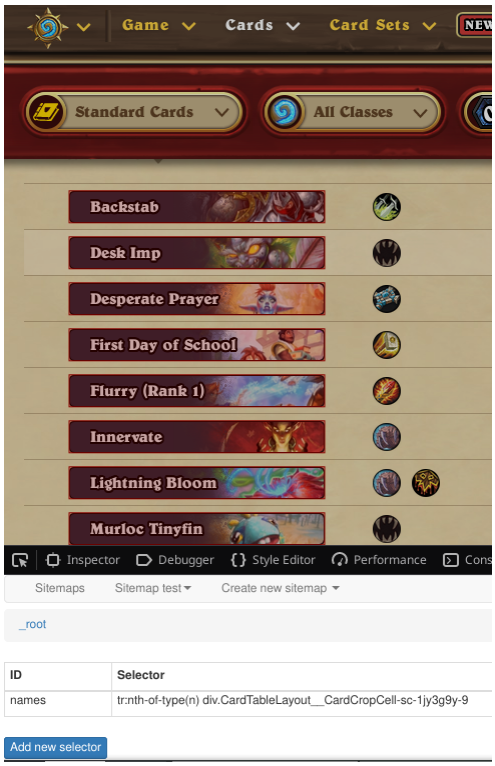



Figure 5.2: Open Web Scraper Results

#### 5.1.2.2 Implementation of Initial Method

Once the `cards-standard.txt` file was created, the implementation of the card vectorization could begin. Using Python the file was opened and with the `readlines()` method, an array of each card name was created. Two dictionaries were then created for the purpose of converting the cards to id form and back, the first dictionary called `cards_key` housed the names of the cards which could be searched by id and the second dictionary called `cards_value` did the inverse, allowing the id to be found by searching the name.

[2] `https://webscraper.io/`

```
In [2]: #open file for read
        file1 = open('cards-standard.txt', 'r')
        #read lines into array
        Lines = file1.readlines()
        #declarations
        cards_key = {}
        cards_value = {}
        i=1
        #for each card name
        for line in Lines:
            #add i as key and name as value
            cards_value[i] = line.rstrip("\n")
            #add name as key and i as value
            cards_key[line.rstrip("\n")] = i
            #increment id
            i += 1

        print(cards_key["Arcane Breath"])
        print(cards_value[32])
        print(len(Lines))

        32
        Arcane Breath
        1254
```

Figure 5.3: Code Extract of dictionary initialization

With the two dictionaries created, the functions that allow created decks to be vectorized or converted to readable form need to be integrated. Two functions, one to vectorize called `deck_to_vector_csv` , another to de-vectorize called `vector_to_deck_csv`. Both use a loop but they use different dictionaries to get the card in the desired form then an array of the converted deck is returned. Although there is some text pre-processing done beforehand when vectorizing the decks, unwanted white spaces can causes bugs in the future so by using `lstrip` and `rstrip`, excess white spaces on the left and the right of the card name can be removed (this does not include spaces between words).

```
In [4]: deck_vector = [32,64,98,1000,1250,1,2,4,5,70,97,12,67,98,33,33,43,54,76,
        print(len(deck_vector))

        #no need to filter based of card class or legendaries etc.. because this

        def deck_to_vector_csv(deck):
            deck_vector = []
            for elem in deck:
                #remove any potential white spaces from strings
                if type(deck) is not list:
                    elem = elem.lstrip()
                    elem = elem.rstrip()
                deck_vector.append(cards_key[elem])
            return deck_vector

        def vector_to_deck_csv(deck_vector):
            deck = []
            for elem in deck_vector:
                deck.append(cards_value[elem])
            return deck

        deck = vector_to_deck_csv(deck_vector)
        print(deck)

        new_deck_vector = deck_to_vector_csv(deck)
        print(new_deck_vector)
```

Figure 5.4: Code Extract of deck conversion functions

The result is a functioning card and deck vectorization and de-vectorization system that will allow the deck data to be converted into a format that the GAN can train with, and return the generated decks to human readable deck without loss of information throughout the process.

#### 5.1.2.3 Limitations

Whilst this method functions without issue, there are certain limitations it. The main limitation is that the only information displayed about the cards is their id and their name. This is not enough information to do evaluations, sorting, displaying or even class selecting, meaning that with this method there would be no way of determining the validity of the deck, check for mana curves, check type split. Another limitation is that there are some libraries like the simulator that use official Hearthstone card id's to identify cards and use them to play simulated games. So a new method would need to be adapted to incorperate these changes. This would also allow for easier, more flexible expansions on the project in future works.

### 5.1.3 Improved Method

The new method will make use of third party platforms that already have the data needed to train the AI. The format of this file will be JSON, it is

the easiest format to parse, there are many libraries in Python that enable the processing of JSON data. There are many ways of obtaining this data, this next section will describe three potential platforms to get the card data from, and then select one based on their pros and cons. Once the data is collected into a file, then it is simply a matter of loading the file, all the data will be correctly structured and available. Some data cleaning will likely be needed.

#### 5.1.3.1 Blizzard API

The Blizzard API is a platform that allows a user to gather information about any of their games, the one that interests the project is the Hearthstone section. On this the user can make card searches, gather metadata and even use their deck code algorithm that turns a deck of cards into a string that can be copied and used to instantly build a deck in the game.

The upside to this API is that it is official, so it will likely be maintained and updated frequently far into the future, however, the downside is that the API is designed for web application so it requires OAuth with a secret client to generate a token to use the API. This would disallow anyone other than the client user to make use of the implemented code, since the client ID is limited to a single person, and for security reasons that client ID cannot be given, this process makes it awkward for other developers to make use of the code.

Another downside is that the API is extremely limited in the data that can be collected from it, the data needed for the project is a list of cards, but the Blizzard API only allows a single card search at a time meaning that a list of card names would need to be compiled then an API call will need to be repeated for every card (which at the moment with current standard cards is 1254 times). Although thanks to the initial method a list of card names has already been compiled, so it would be feasible to gather the data from this API is a way around the OAuth can be found. The API throttles at requests at a rate of 36,000 requests an hour, albeit many request if exceeded it would block the API from being used further until the next hour, this can be problematic if the secret client ID were to be shared along with the code to a public platform, if too many users were to make requests at once it could block users out of the API.

#### 5.1.3.2 HearthstoneJSON

HearthstoneJSON is a web site that considers itself an API, it functions differently to an API because on the site there are a series of file uploads each one after an update (like an expansion release) with the new cards, there are two files that are uploaded each time, one with all the cards and other with just the collectible cards. The cards that interest us are the collectible ones as the other non-collectible cards are tokens or cards that other cards generate and cannot be added manually to a deck. This API does not allow the user to make queries for specific card data, it is simply a file with everything in it, this is both a good and a bad thing, it makes the data collection easy since all that is

needed is to download the file but it means that cards will have to be manually sorted, wild from standard, specific sets, etc...

Furthermore, it means that no code is required to gather the data, however, a user would need to download the file if they want the new cards and it heavily relies on the developers staying up to date with it. As of now, the api has not been updated since 2018 signifying that that it does not have the latest cards making it tricky to fulfill the standard cards only requirement.

#### 5.1.3.3 Hearthstone API

Hearthstone API is another web platform that makes use of RapidAPI for their endpoints. RapidAPI is a site that hosts thousands of API's that can be accessed with a single key, it allows users to make queries either on their site or through a library to gather data. Hearthstone API relies on this platform to allow users to make queries, unlike the Blizzard API, there search options are limitless, a user can search for specific expansions, collectible, card name, card class and much more. The API is constantly updated, so that even the latest released cards are added, there still is the issue with the access token, but it is more manageable since a developer can use their RapidAPI key from their account, or an account can be created, no OAuth needed. The biggest downside is that the API has quite high latency, meaning that it would take longer to make API calls.

#### 5.1.3.4 Choice

All three platforms have their pros and cons for their usage, however the better one to choose would be the Hearthstone API using RapidAPI. The reasons are because, it is updated and actively maintained unlike HearthstoneJSON, it also allows to do queries to limit the range of data gathering to what is needed so in theory no pre-processing will need to be done. Access to Hearthstone API is also easier to its Blizzard counterpart because of the RapidAPI platform and the lack of OAuth which is not needed in the context of this project. The JSON data that the Hearthstone API provides will encourage sorting and filtering to improve the GAN training. To summarise, for this project the best choice is Hearthstone API due to how updated it is and how much easier it is to use compared to its respective counterparts.

#### 5.1.3.5 Implementation of Improved Method

To get the data from Hearthstone API, the `requests` library was used to access the RapidAPI platform. The data was collected using the Card Sets query built into the API which allows the user to get all the cards from an expansion. An array of these expansions was created and iterated through to make a query for each card set, the result was an array of card lists per expansion. Once that was created, the card lists were concatenated into a single array whilst adding an integer id that will be used for the GAN model. Finally, the JSON list was written into a json file using the `json.dump()` method from the `json` library.

```python
import requests
import json

data = []
final_json = []
sets=['Madness%20At%20The%20Darkmoon%20Faire', 'Scholomance%20Academy', 'Demon%20Hunter%20Initiate', 'Ashes%20of%20O

for expansion in sets:
    url = "https://omgvamp-hearthstone-v1.p.rapidapi.com/cards/sets/{}".format(expansion)
    print(url)
    querystring = {"collectible":"1"}
    headers = {
        'x-rapidapi-key': "6f7facb290msh83edc592c0e6ed3p1dbcbajsne1d012ba6886",
        'x-rapidapi-host': "omgvamp-hearthstone-v1.p.rapidapi.com"
        }

    response = requests.request("GET", url, headers=headers, params=querystring)
    json_data = json.loads(response.text)
    data.append(json_data)
    print(len(json_data))

i = 0
for card_set in data:
    for card in card_set:
        card['id'] = i
        i += 1
        final_json.append(card)


with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(final_json, f, ensure_ascii=False, indent=4)
```

Figure 5.5: Getting card data with Hearthstone API

The file created is a json file called `data.json`, an extract of the contents can be viewed in Appendix – . Loading this file for the GAN is straightforward, simply use the in-built Python `open()` function. As seen in figure 5.6 below the amount of information provided by a single card is a big improvement to just the name of the card.

```
import json

file = open("data.json", "r")

cards = json.load(file)

cards[1]
```

```
{'cardId': 'DMF_221',
 'dbfId': '61127',
 'name': 'Felscream Blast',
 'cardSet': 'Madness At The Darkmoon Faire',
 'type': 'Spell',
 'rarity': 'Common',
 'cost': 1,
 'text': '<b>Lifesteal</b>. Deal $1 damage to a minion and its neighbors.',
 'flavor': 'All the souls start running when they see the felscream truck.',
 'artist': 'K. Lashley & K. Turovec',
 'collectible': True,
 'playerClass': 'Demon Hunter',
 'img': 'https://d15f34w2p8l1cc.cloudfront.net/hearthstone/7a73e2ee3bbff1f040e6
701d6a90.png',
 'locale': 'enUS',
 'mechanics': [{'name': 'Lifesteal'}],
 'id': 1}
```

Figure 5.6: Loading card data

Whilst working with the collected data, some minor inconsistencies appeared in the JSON, firstly the name of hero portraits appeared, these are special personas a class can take on, with different animations, audio queues, however they are not cards, those outliers need to be removed for the code to function. Another issue stemmed from the latest expansion **Madness at the Darkmoon Faire**, this expansion added a new type of card, the dual class card, allowing two different classes to use the card, the problem is that due to the structure of the JSON there can only be a single class under the key `playerClass` (which can be seen in figure 5.6). To fix these issues, a function was written to remove the hero portrait outlier cards, and append both classes of dual class cards to a variable.

```
def remove_impurities(cards):
    i = 0
    j = 0
    fixed = 0
    to_del = []
    for card in cards:
        #remove hero portraits
        if 'cost' in card:
            i+=1
        else:
            to_del.append(card)
            print('Removed:', card['name'])
            j +=1
        #check is card is a dual class card, then appends the other missing class to the classes key
        if card['playerClass'] == 'Neutral' and 'classes' in card:
            card['playerClass'] = card['classes'][0]
            fixed += 1
            print('Fixed playerClass of ', card['name'], ' from Neutral to ', card['classes'][0])
    for elem in to_del:
        cards.remove(elem)
    print('Number of impurities removed: ', j, '/', i+j)
    print('Number of card impurities fixed: ', fixed)
    return cards
```

Figure 5.7: Removing impurities in card data

## 5.2 Deck Data

### 5.2.1 Introduction

The core necessity for the success of the project is data, the GAN requires a lot of data to train, a lot of deck data. Datasets do exist and can be found on various platforms, however they tend to be outdated or limited in size. The GAN takes the datasets as input for the discriminator to use to determine if the generated decks are similar. So where can recent datasets for Hearthstone decks be found? Well unfortunately ready made datasets for the specific purpose of this project, however, platforms exist in the hearthstone community where the users post their decks online, these decks can be rated, sorted and viewed. Using a web scraper would enable the gathering of the deck data needed for the GAN. Web scraping is a technique for harvesting data on a website, the tool used to accomplish the task is called a web scraper among other names. Legal issues surrounding web scraping can be found in Section 4.1.2 of the report.

### 5.2.2 Platform Choice

HSReplay and Hearthpwn are the two most popular sites in the community, Hearthstone players frequent these sites to see what decks are strong currently or to find niche archetypes to accommodate the use of certain cards they like. This sub section will describe both platforms and which one is the better fit in the context of this project.

#### 5.2.2.1 HSReplay

HSReplay is a community run website owned by HearthSim (the same people that have developed the Hearthstone simulators), the most notable feature is their game replay feature, allowing users to view and share their previously played games. They also provide extensive statistics about cards and decks based on user played games.

The site has a lot of features, graphs and statistics, but all of this is not really useful as the dataset only needs the decks themselves. The biggest downside to this website is how they display their data. After sending a request to scrape data from their site, getting data from their site proved tricky, due to all the features and displays, the data does not seem to be loaded straight away, meaning that the web scraper can not seem to find certain information, namely the deck lists. The deck lists are requested after the page has loaded to prevent the web page from taking to long to load, whilst this is good practice to improve the sites usability, it impedes the web scraper. Before delving any deeper into the issue, testing the second platform might prove to be more straightforward.

#### 5.2.2.2 Hearthpwn

Hearthpwn is an alternative to HSReplay, it acts as a database for cards and decks rather than statistics and replays. Decks can be sorted by expansion, popularity and class. This site's features can actually assist a web scraper in collecting the data that is needed. Because of the sites minimal design it runs smoothly and loads fast, so from first glance it seems ideal for data harvesting. After receiving permission to collect data, initial testing was successful and much less hassle than HSReplay, because of this the best course of action is to select Hearthpwn, for the in depth sorting and simplicity to scrape. The next sections will detail methods to scraping data from Hearthpwn.

### 5.2.3 Initial Method

Before speaking about the implementation of the web scraper, the first thing that needs to be discussed is the way it was formatted initially and how it was improved. Since the web scraper implementation is the exact same for both the initial and the later improved method, the web scraper code can be explored further on.

#### 5.2.3.1 Text

The initial method made use of the `cards-standard.txt` file that was reviewed before, this is because the web scraper was initially implemented with the text file method in mind then was later changed to accommodate the improved system. The idea is that the web scraper collects the deck data and then stores it into two separate arrays, the first one to store the deck list, so every deck with their card names inside them and the second one is the class of the

deck. The reason why there are two arrays was to create a dataset for each class, in order to organize the training of the GAN. These arrays would then be sorted by class, then a `csv` file containing the deck data would be saved. For safety, the datasets were saved in a vectorized and non-vectorized form, it was unknown which one would be better or easier to use so both forms were saved.

#### 5.2.3.2 Implementation

```
def concatenate_deck_class(decks, classes):
    concatenated_result = []
    for i in range(len(decks)):
        concatenated_result.append([decks[i],classes[i]])
    return concatenated_result

deck_type = ['Demon Hunter','Druid','Hunter','Mage','Paladin','Priest','Rogue','Shaman','Warlock','Warrior']

#for every class
for cl in deck_type:
    print("Scraping information: ", cl)
    #calls web scraper
    decks, deck_classes = scrape_decks(cl)
    print("Creating ", cl ," decks")
    #concatenate the class and deck arrays
    collected_decks = concatenate_deck_class(decks, deck_classes)
    collected_decks_vectored = concatenate_deck_class(vectorize_deck_list(decks), deck_classes)
    #write to csv file
    print("Creating ", cl, ".csv")
    pd.DataFrame(collected_decks, columns=['cards', 'class']).to_csv("collected_decks_"+deck_type+".csv",index=False
    pd.DataFrame(collected_decks_vectored, columns=['cards', 'class']).to_csv("collected_decks_vectored_"+cl+".csv",
```

Figure 5.8: Writing scraped data to csv file

The implementation follows after the collection of the data into the two arrays, a concatenate function was then created to append the class label to the deck list, the file writing was done using `pandas` to create a dataframe to then save the file as a csv. An example output csv file can be seen in Appendix E

With the csv files created and filled with data, the decks within need to have a way to be extracted, so that they can be used in the GAN. When reading the file, anomalies were checked for: problematic characters, excess brackets and deck length. Since the standard size of a legal deck is 30 cards, any that do need reach that will not be used for training, the result is an array with the deck lists and their class labels. However it was later on discovered that the class labels would not be enough to correctly build the GAN, since there can be other class cards or neutral cards in a deck list, hence why the JSON method was implemented later on.

```
def read_vector_decks(file):
    #read the file
    df = pd.read_csv(file)
    cards_from_csv = df['cards']
    converted_decks = []
    #every deck in the dataset
    for cards in cards_from_csv:
        if len(deck) == 30: #len(cards)
            deck_to_convert = []
            #remove problematic characters
            cards = cards.strip("[]")
            cards = cards.split(", ")
            #every card in deck
            for card in cards:
                if card == '':
                    continue
                deck_to_convert.append(int(card)-1)
            #convert vector decks to human readable
            converted_decks.append(vector_to_deck(deck_to_convert))
            class_from_csv = df['class']
            final = concatenate_deck_class(converted_decks, class_from_csv)
    return final

deck_list = read_vector_decks("collected_decks_vectored.csv")
```

Figure 5.9: Reading csv file

### 5.2.4 Improved Method

In the previous section, the project was making use of already vectorized deck data from the web scraper. The problem with that is that if the card list is changed in any way, for example when sorting by a certain class, or removing impurities, the indexes of the vectors will be wrong. So it would be better to save them in a human readable format, then vectorize them when needed. However using `csv` file format, data will be harder to read, even with just the integer numbers, a decent amount of pre-processing was required to just read the data. Using another method may save time and processing power, good thing is that there is a better alternative with the JSON card data that was collected previously. By loading the `data.json` card data file, with just the names of the cards a JSON dataset can be created by converting the web scraped card names to full fledged JSON data format.

#### 5.2.4.1 Implementation of Improved Method

To begin, the code for the `csv` conversion needs to be adapted, it will be longer than with the csv. Instead of vectorizing the deck data and then writing to file, the scraped data is appended to a `to_convert` array. An example of a scraped deck looks like: `['Lab Partner', 'Lab Partner', 'Primordial`

Studies', 'Primordial Studies', 'Wand Thief', 'Wand Thief', 'Astromancer Solarian', 'Cram Session', 'Cram Session', 'Runed Orb', 'Runed Orb', 'Wildfire', 'Wildfire', 'Firebrand', 'Fireball', 'Fireball', 'Reckless Apprentice', 'Reckless Apprentice', 'Refreshing Spring Water', 'Refreshing Spring Water', 'Ring Toss', 'Ring Toss', 'Jandice Barov', 'Ras Frostwhisper', 'Flamestrike', 'Flamestrike', 'Mordresh Fire Eye', 'Far Watch Post', 'Far Watch Post', 'Taelan Fordring']

The JSON card data is also loaded for the conversion later on.

```
deck_type = ['Demon Hunter','Druid','Hunter','Mage','Paladin','Priest','Rogue','Shaman','Warlock','Warrior']
#read json card data
file = open("data.json", "r")
cards = json.load(file)

to_convert = []
#for every class
for cl in deck_type:
    print("Scraping information: ", cl)
    #scrape data
    decks, deck_classes = scrape_decks(cl)
    print("Appending ", cl, " data")
    #append to array for conversion
    to_convert.append(decks)
```

Figure 5.10: JSON adaptation of previous code

To convert the scraped data to a JSON format, the name of the cards need to be searched through the `cards` array, then a new list needs to be created to add the converted cards into. Figure 5.11 is a code extract of this process. For each card in a classes deck list, strip excess white spaces then find the name of the card in the JSON card list, once found append to a new array. Once the conversion is complete, the data is then written into a JSON file using `json.dump()`. Then the creation of the dataset is complete. An extract of the a converted JSON dataset be found in Appendix F.

```
#strip of spaces
def strip(elem):
    elem = elem.rstrip()
    elem = elem.lstrip()
    return elem

#returns json of card name
def get_json(val):
    val = strip(val)
    for card in cards:
        card['name'] = strip(card['name'])
        if card['name'] == val:
            return card
    return '{}'

class_json = []
for class_ in to_convert:
    final_json=[]
    for deck in class_:
        deck_list=[]
        for card in deck:
            json_card = get_json(card)
            deck_list.append(json_card)
        final_json.append(deck_list)
    class_json.append(final_json)
```

Figure 5.11: JSON adaptation of previous code

Since the file type for the datasets has changes from `csv` to JSON, it means that the file loading needs to be modified to work with the updated datasets. Luckily it is straightforward to load JSON data, with the `json` library, simply load the file (as seen in figure 5.12), there is also a check to only register decks of legal length.

```
def read_json_decks(file):
    file = open(file, "r")
    deck_list = json.load(file)
    cleaned_deck_list = []
    for deck in deck_list:
        if len(deck) == 30:
            cleaned_deck_list.append(deck)
    return cleaned_deck_list

deck_list = read_json_decks('data_Warrior.json')
```

Figure 5.12: Read JSON file

### 5.2.5 Implementation of Web Scraper

Now that the methods surrounding the web scraper have been detailed, the implementation of the web scraper can begin. Firstly, the library that is used to get the data is `requests`, and the library that is used to parse through the data is `Beautiful Soup`. Before harvesting the data, the url needs to be determined for the request method to work, since there are 10 classes in Hearthstone, 10 separate urls will be needed. The urls are uncomplicated to find, figure 5.13 shows the filter section of the deck search on Hearthpwn. Every class has a unique id when attempting to filter: `'Demon Hunter' : 16384, 'Druid' : 4, 'Hunter' : 8, 'Mage' : 16, 'Paladin' : 32, 'Priest' : 64, 'Rogue' : 128, 'Shaman' : 256, 'Warlock' : 512, 'Warrior' : 1024`
So a dictionary was created to store these values so that when the web scraper goes to collect a certain class decks, it can find the values easily. An example url will look like: `https://www.hearthpwn.com/decks?filter-show-standard=1&filter-show-constructed-only=y&filter-deck-tag=1&filter-class=4`). This link will only show standard and constructed decks, `filter-deck-tag` sorts the decks buy `"hot"`, which means popular decks and of course the class filter, in this example it is Druid decks.


Figure 5.13: Hearthpwn deck search

The links though will only display the first page of decks, which is around 25 decks per page, the are many more but the number is not fixed, so before the decks can be collected, the number of pages there are needs to be found, `page_max` does just that (figure 5.14).

```
#gets number of pages
def page_max(deck_type):
    #request html from url
    r = requests.get('https://www.hearthpwn.com/decks?filter-show-standard=1&filter-show-constructed-only=y&filter-
    soup = BeautifulSoup(r.content,"html.parser")
    #parse through html
    rows = soup.find('ul', class_="b-pagination-list paging-list j-tablesorter-pager j-listing-pagination")
    links = rows.find_all('li')
    return int(links[len(links)-2].text)
```

Figure 5.14: Function that finds the number of deck pages there are

The scraping needs to be done in two parts, this is because the results shown on the deck search page does not include the cards that are use to build the deck, that is hidden behind another link to the deck information. This means that initially, the links to the decks needs to be collected, then another part needs to go within those collected links to harvest the card data. The `scrape_decks` function takes a class as an argument and creates the starting links for the selected class and gets every deck link, from all the pages (figure 5.15).

```
def scrape_decks(deck_type):
    start_urls = []
    #get number of pages
    n = page_max(deck_type)
    print('Number of Pages: ',n)
    #create the initial urls for deck_type
    for i in range(n):
        start_urls.append("https://www.hearthpwn.com/decks?filter-show-standard=1&filter-
    deck_urls = []
    deck_classes = []
    #get every deck link on each page deck_type
    for url in start_urls:
        print(url)
        r = requests.get(url)
        soup = BeautifulSoup(r.content,"html.parser")
        rows = soup.find_all('table', class_="listing listing-decks b-table b-table-a")
        for row in rows:
            decks = row.find_all('span', class_="tip")
            for deck in decks:
                links = deck.find_all('a', href=True)
                for link in links:
                    deck_urls.append("https://www.hearthpwn.com" + link['href'])

            deck_class = row.find_all('td', class_="col-class")
            for cl in deck_class:
                deck_classes.append(cl.text)
    #call function that gets card data within the collected urls
    return cards_from_url(deck_urls), deck_classes
```

Figure 5.15: Gets all deck links from class page

After the deck links are collected, another function called `cards_from_url` is called, this take the urls as an argument and returns the deck lists in their default format (figure 5.16). It navigates thorough the page into the html table containing the list of cards and gets the names. Finding the correct tables and containers was found through trial and error, but is consistent throughout every page on the site.

```
#scrape cards in deck from separate url
def cards_from_url(deck_urls):
    decks = []
    for url in deck_urls:
        print(url)
        deck=[]
        r = requests.get(url)
        soup = BeautifulSoup(r.content,"html.parser")
        table = soup.find_all('table', class_="listing listing-cards-tabular b-table b-table-a")
        #navigating the page table to get card names
        for rows in table:
            row = rows.find_all('td', class_="col-name")
            for i in row:
                card = i.find('a')
                name = card.text
                number = card['data-count']
                #remove "\n" for later functions
                for num in range(int(number)):
                    deck.append(name.replace('\n',''))
        decks.append(deck)
    return decks
```

Figure 5.16: Gets cards from deck links

## 5.3 Generative Adversarial Network

### 5.3.1 Introduction

When researching GANs, the main application that they were used for was image generation with human and artistic applications, creating realistic people that do not exist from many images of people. The GAN made use of libraries to convert the images into two dimensional arrays, single dimensional arrays seemed scarce at best. Only a few basic tutorials about vectors in a two dimensional space could be observed using single dimensional input. Making use of what little information about single dimension GANs was available, the conversion from 2D to 1D started. The vectorization of decks idea was thought of when the conversion of images was observed. The point is that by feeding the GAN many vector inputs, the resulting output will be a vector that is similar to the training vectors but different enough for it to create unique vectors, whilst demonstrating "thinking" on the GANs part. (talk about tutorial that inspired me and reference it)

### 5.3.2 Implementation

The structure of the implementation is as follows, to begin the generator and discriminator models will be define, then how it combines into a GAN, after that the training process will be discussed, finally the performance summary will be detailed.

#### 5.3.2.1 Generator

The generator is a sub-model of the GAN that will generate the deck vectors. The generator is composed of layers, an input layer, an output and activation layers between. A dropout layer is also added to the generator to help prevent over-fitting by randomly setting inputs to 0 during training time. The number of activation layers added to the generator is the result of many hours of optimization, when training graphs are displayed showing how similar the generated deck is to a real deck. After hours of tweaking, one input layer, one output layer and two activation layers (of 200 activations) seems to produce the best outcomes. All the layers except the output layer are `ReLU` activated, this limits the vectors from going into the negative numbers. The output layer is set to `linear` otherwise if there are any card ids generated that do go into the negative, they will automatically be set to 0 which is a card id itself. The generated negatives numbers and how they are dealt with is discussed later on. The loss is calculated through `binary cross-entropy`, because it categorizes a deck to be either 0 (a fake deck) or 1 (a real deck). Figure 5.17 shows the code implementation of what was described, the `Sequential` function is part of the `keras` library function to declare the model, `Dense` and `Dropout` as also part of `keras`.

```
# define the standalone generator model
def define_generator(latent_dim, n_outputs=30):
    model = Sequential()
    #input layer
    model.add(Dense(200, activation='relu', kernel_initializer='he_uniform', input_dim=latent_dim))
    #activation layers
    model.add(Dense(200, activation='relu'))
    model.add(Dense(200, activation='relu'))
    #droupput to prevent overfitting
    model.add(Dropout(0.3))
    #output layer
    model.add(Dense(n_outputs, activation='linear'))
    model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])
    return model
```

Figure 5.17: Define generator model

#### 5.3.2.2 Discriminator

The discriminator is the second sub-model that constitutes the GAN, the purpose of the discriminator is to determine if a deck vector created by the generator resembles the deck vectors from the training dataset. The model is smaller than a generator model with just an input and an output layer. The input layer has 30 activations and take a generated vector with 30 inputs (the amount of cards within deck) with `ReLU` activation. The output layer has a single output and a sigmoid activation, this is because the output of the discriminator is a binary result, 0 for fake, 1 for real. This sub-model is also reliant on `binary cross-entropy`. Figure 5.18 is a code snippet of the discriminator illustrating the explanation.

```
# define the standalone discriminator model
def define_discriminator(n_inputs=30):
    model = Sequential()
    model.add(Dense(30, activation='relu', kernel_initializer='he_uniform', input_dim=n_inputs))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Figure 5.18: Define discriminator model

#### 5.3.2.3 GAN

The GAN section is where both the generator and the discriminator sub-models are added to the GAN model. There is also a setting that needs to be changed, the discriminator needs the weights to not be trainable or the criteria of the discriminator will change and disrupt the training process.

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(generator, discriminator):
    # make weights in the discriminator not trainable
    discriminator.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(generator)
    # add the discriminator
    model.add(discriminator)
    # compile model
    model.compile(loss='binary_crossentropy', optimizer='adam')
    return model
```

Figure 5.19: GAN model

#### 5.3.2.4 Training

This section is where the training of the GAN happens. The training process starts off by preparing the real and fake decks, the real ones are sampled from the dataset randomly and the fake decks are generated randomly (not by the generator). The results from these functions return an `x` and a `y` variable, so `x_real`, `x_fake`, `y_real` and `y_fake`. The `x` variable contains the array of deck vectors and the y their labels, their labels being a binary to help the GAN distinguish if they are a real or a fake deck. After the real and fake decks are gathered, they are used to update the discriminator model, so that it can distinguish real decks from fake decks.
Once the discriminator model has trained, the generator creates a deck vector to attempt to trick the discriminator. The deck is then trained on the GAN model to update the generator based on the discriminator's error.
The whole process is repeated for the number of epochs given, for this GAN model it is 10,000 epochs. Every thousand epochs a performance summary is

done to evaluate the GAN. Which is the subject of the next sub section. Figure 5.20 shows the code of what has been described above. When a summarize performance function is called, a generated GAN is then appended to an array, these will be the resulting decks, the reason why this is done is because, depending on how the model started the model tends to overfit towards the end of the training process but the number of epochs is never the same so by saving each deck, a function can then be called to sort the illegal decks and identify the better deck.

```
# train the generator and discriminator
def train(g_model, d_model, gan_model, latent_dim, class_, deck_list ,n_epochs=10000, n_eval=1000, n=100):
    # determine half the size of one batch, for updating the discriminator
    print("Training a",class_, "deck")
    final_gans = []
    #limit deck list and card list to chosen class
    new_cards, new_deck_list = limit_card_range(copy.deepcopy(cards), copy.deepcopy(deck_list), [class_, 'Neutral'])
    # manually enumerate epochs
    for i in range(n_epochs):
        # prepare real samples
        x_real, y_real = generate_deck_samples(n, class_, new_deck_list)
        # prepare fake examples
        x_fake, y_fake = generate_fake_decks(g_model, n, new_cards)
        # update discriminator
        d_model.train_on_batch(x_real, y_real)
        d_model.train_on_batch(x_fake, y_fake)
        # prepare points in latent space as input for the generator
        x_gan, y_gan = generate_fake_sample(g_model, new_cards)
        # update the generator via the discriminator's error
        gan_model.train_on_batch(x_gan, y_gan)
        # evaluate the model every n_eval epochs
        if (i+1) % n_eval == 0:
            summarize_performance_gan(i, g_model, d_model, class_, new_cards, new_deck_list)
            final_gans.append(x_gan)
    return final_gans, new_cards, new_deck_list
```

Figure 5.20: Training process of the GAN

#### 5.3.2.5 Summarize Performance

This part of the training process is called every thousand epochs in the training, it is used to evaluate the training process to check the progress, and for any improvements, signs of over-fitting etc...
Just like in the training process a real and a fake deck is gathered, however instead of updating the models, instead they are evaluated and a accuracy is return. Since the process is using binary results, the discriminator will evaluate the deck with either a 0 or a 1. Theoretically the discriminator should always display 1 for real decks and then either 0 or 1 for fake ones. After, a graph is drawn displaying the both decks, this is to check their similarity. Refer to figure 5.21 for illustration in code form.

```
# evaluate the discriminator and plot real and fake points
def summarize_performance_gan(epoch, generator, discriminator, class_, cards, deck_list, n=1):
    # prepare real samples
    x_real, y_real = generate_deck_samples(n, class_, deck_list)
    # evaluate discriminator on real examples
    _, acc_real = discriminator.evaluate(x_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_decks(generator, n, cards)
    # evaluate discriminator on fake examples
    _, acc_fake = discriminator.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print(epoch, acc_real, acc_fake)
    # scatter plot real and fake data points
    real = []
    fake = []
    for i in x_real:
        for d in i:
            real.append(d)
    for x in x_fake:
        for d in x:
            fake.append(d)
    real.sort()
    fake.sort()
    plt.plot(real, color='red')
    plt.plot(fake, color='blue')
    plt.show()
```

Figure 5.21: Summarize performance of the GAN

### 5.3.3 Initial Results

Initially the results of the training process were less than ideal, despite having the decks having similar structure (figure 5.22) the issue was that the decks that were created had cards from all the classes. The reason why this happened is because when the GAN trains, it tends to predict cards around the same point as the real cards, but it never produce something identical, and the way the cards are identified is sorted by Hearthstone's default sorting which is cost, so the class cards end up spread across the array of cards the GAN is training from, resulting in multiclass decks that are considered illegal and thus cannot be played. In this case the deck training was limited to the Warrior class dataset, so a method would need to be devised to make a choice.

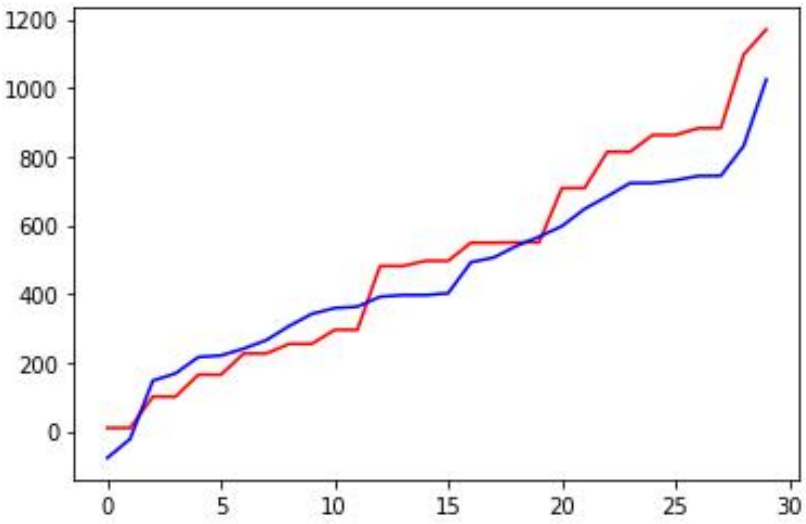



Figure 5.22: Similarity between generated and real deck, blue represents fake deck and red the real deck

Another problem that will need to be dealt with is the appearance of negative id's, because of the way the generator model creates decks, some of the cards that are low in the card array can cause the generator to predict negative identifiers for cards, and those cards do not exist. There are a few options that can be considered, such as removing the generated decks that have negative numbers, but will there be any generated decks at the end? The generator model could be changed to not allow negative identifiers to exist, but then there will be a lot of cards of id 0, meaning that there will be more than 2 duplicates of that card resulting in an illegal deck. Or a method that does an absolute of the negative number to make it positive, however if the negative number is too high then it might go outside the range of the card array.

Finally, the GAN may not guarantee the generation of legal decks, so a method would need to be written to filter out the illegal decks, such as over 30 cards, more than 2 duplicates of normal cards, duplicates of legendary cards, etc...

### 5.3.4 Improvements

This section will discuss the improvements that were made to the GAN in order to solve some of the issues discussed previously.

#### 5.3.4.1 Class Choice

The first improvement was to add the ability to select other classes to train a deck for, originally only Warrior decks were trained. A user input was added

to allow them to select one of the ten classes to generate a deck. For usability, the user can select by either typing the name of the class or the number it is associated to. This choice is then saved for when the AI training begins, the AI will select the appropriate dataset to train from and will also limit card data to that class, this will be explained in detail in the next section.

```
1. Warrior
2. Mage
3. Hunter
4. Priest
5. Druid
6. Rogue
7. Shaman
8. Warlock
9. Paladin
10. Demon Hunter
Enter which class you want to make a deck for (number or name): 3
You have chosen: Hunter
```

Figure 5.23: Choose class results

#### 5.3.4.2 Limit Card Data

Previously, the initial results of the GAN would produce decks with cards from multiple classes, this was a major issue as the GAN did not produce legal decks to play. To fix this, when the user selected a class to create a deck for, the dataset for that class was loaded for training, and the cards were filtered to only have cards of that class and the neutral class, in essence only the cards that the class can use. Once the cards were filtered, the cards identifiers were changed to fit the smaller scale to limit the bounds of the id's, otherwise there could be more predicted id's that do not correspond to cards. Because the card id's were changed, it meant that the id's of the loaded dataset had to be changed aswell. The figure 5.24 below shows the code of this process through the `limit_card_range` function.

```
def limit_card_range(cards, new_deck_list, classes):
    #filter the cards to return the selected classes
    new_cards = groupby_class(classes, cards)
    i=0
    #reID every card after filter
    for card in new_cards:
        card['id'] = i
        i+=1
    #reID cards in decks to fit new ID's of the filtered card list
    for deck in new_deck_list:
        print([x['name'] for x in deck])
        for card in deck:
            print(card['name'], card['id'])
            temp = next((new_card for new_card in new_cards if new_card['name'] == card['name']), None)
            if(temp != None):
                print(temp['name'], temp['id'])
                card['id'] = temp['id']
    return new_cards, new_deck_list
```

Figure 5.24: Limit Range of cards by chosen classes

After this was implemented into the GAN, the resulting decks were legal and showed synergistic card combinations.

#### 5.3.4.3 Deck Verification

When decks were generated by the AI, it was unclear if the GAN would produced consistently legal decks, for example some decks could have too many duplicates of a card or duplicates of legendaries.

The first check was to convert negative id's to positive ones and check if the converted id's were in legal range of the card list. The second verification needed was see if the created decks had exactly 30 cards. Note that the `created_gans` array is the array that is returned by the GAN with the generated decks.

```
final_decks = []
for decks in created_gans:
    deck = []
    #check for negative id's
    for card in decks[0]:
        #if absolute of card id is within card list range
        if abs(card) <= len(new_cards):
            deck.append(abs(card))
        else:
            break
    #check if deck has 30 cards
    if(len(deck) == 30):
        final_decks.append(deck)
```

Figure 5.25: Convert negative id's and check for legal deck size

Finally, to verify that there are not too many duplicate cards in the deck,

a method was written using an array to count the number of instances of a card in a deck. Removing the deck if the duplicates of a card went past 2 for non-legendary cards, and 1 for legendary cards.

```
duplicates = []
for decks in final_decks:
    print('DECK:')
    #convert float list to int list
    decks = [int(a) for a in decks]
    readable = vector_to_deck(decks, new_cards)
    for card in readable:
        #if duplicates of card is superior to 2 then deck is illegal
        if(readable.count(card) > 2):
            final_deck.remove(decks)
        #different duplicate check for legendary cards
        if(card['rarity'] == 'Lengendary'):
            if(card in duplicates):
                final_deck.remove(decks)
            else:
                duplicates.append(card)
```

Figure 5.26: Check for duplicate cards

# 6. Testing

## 6.1 Aims

The aim of the project is to generate playable Hearthstone decks from an AI. The secondary goal is to generate decks with a good win to loss ratio, so more wins than losses. The datasets used for training were collected with a created web scraper, one for each class. To test these decks, a simulator will be used to play the decks, the selection process of the simulator and how it was implemented is explained in the next section.
The testing should reflect the following statements:

- The overall win rate of a generated deck should be over 50%, to pertain the relevance of creating a deck
- A generated deck should have a similar **type split** to a real deck
- A generated deck should have a similar **neutral split** to a real deck
- A generated deck should have a similar **mana curve** to a real deck
- The criteria of a considered legal deck should not be breached, illegal decks should be excluded from the results.
- The above statements should not be affected by a change of dataset

## 6.2 Simulator

### 6.2.1 Fireplace

As explained previously fireplace is a github project run by HearthSim, it can be imported as a python library to be used for testing the generated decks. Fireplace is generally better because it is maintained, and speedy. However there are some changes that need to be made to the simulator as the base file that runs the games generates random decks to test.

### 6.2.2 Changes

There were two kinds of changes that needed to be made, first of all the actual file changes to the project to be able to feed the generate decks into, then extra code needed to be added externally to call the library

#### 6.2.2.1 File changes

Two files were changed, the `utils_fireplace.py` file which is a copy adaptation of `utils.py` file which was moved because of python file traversal limitations preventing users from navigation back through file systems effectively. The other file changed was `full_game.py` to accommodate more arguments and allow user created and AI generated decks to be played.

First the changes to `full_game.py`, initially the fireplace simulator did not accept arguments, the main was changed to be able to get them, since ideally the generated decks want to be played against real users decks, both deck lists and their classes need to be passed through. Then a function imported from the `utils_fireplace.py` file, which sets up the games and plays the decks given.

```python
def main():
    #For some reason sometimes my code can find imports from file start so I have to add imports in main
    from fireplace import cards
    from fireplace.exceptions import GameOver
    from utils_fireplace import play_full_game
    import importlib

    importlib.reload(cards)
    player_class = sys.argv[0]
    deck = sys.argv[1]
    player_name = sys.argv[2]
    opponents_deck = sys.argv[3]
    opponents_class = sys.argv[4]

    cards.db.initialize()
    try:
        play_full_game(player_class,deck,player_name,opponents_deck,opponents_class)
    except GameOver:
        print("Game completed normally.")

if __name__ == "__main__":
    main()
```

Figure 6.1: `full_game.py` main function

The changes to `utils_fireplace.py` are more substantial, the `play_full_game()` function is changed slighly to pass variables to the `setup_game()` function (figure 6.2). The rest does not need to be changed because the scope of our testing is focused on deck building rather than improving the AI playing the decks.

```
def play_full_game(player_class,deck,player_name,opponents_deck,opponents_class):
    #changed this to accommodate setup_game changes
    game = setup_game(player_class,deck,player_name,opponents_deck,opponents_class)

    for player in game.players:
        print("Can mulligan %r" % (player.choice.cards))
        mull_count = random.randint(0, len(player.choice.cards))
        cards_to_mulligan = random.sample(player.choice.cards, mull_count)
        player.choice.choose(*cards_to_mulligan)

    while True:
        play_turn(game)

    return game
```

Figure 6.2: `play_full_game` function

Before changing the `setup_game()` function it was coded statically, only a single class was chosen per player. Using the arguments sent by the main, the code was adapted to make it dynamic (figure 6.3), however a `class_converter()` function was written to convert the string inputs of the two players classes to an object so that the simulator can register them as that player (figure 6.4)

```
def setup_game(player_class,deck,player_name,opponents_deck,opponents_class):
    from fireplace.game import Game
    from fireplace.player import Player

    #function to convert class string to readable object for simulator
    p_hero = class_converter(player_class)
    o_hero = class_converter(opponents_class)

    #global so it can carry over to other functions
    global name
    name = player_name

    player1 = Player(name, deck, p_hero) #CardClass.WARRIOR.default hero
    player2 = Player("Oppenent", opponents_deck, o_hero)
    game = Game(players=(player1, player2))
    game.start()
    return game
```

Figure 6.3: `setup_game` function

```
def class_converter(class_):
    if class_ == "Warrior":
        return CardClass.WARRIOR.default_hero
    if class_ == "Mage":
        return CardClass.MAGE.default_hero
    if class_ == "Hunter":
        return CardClass.HUNTER.default_hero
    if class_ == "Priest":
        return CardClass.PRIEST.default_hero
    if class_ == "Druid":
        return CardClass.DRUID.default_hero
    if class_ == "Rogue":
        return CardClass.ROGUE.default_hero
    if class_ == "Shaman":
        return CardClass.SHAMAN.default_hero
    if class_ == "Warlock":
        return CardClass.WARLOCK.default_hero
    if class_ == "Paladin":
        return CardClass.PALADIN.default_hero
    if class_ == "Demon Hunter":
        return CardClass.DEMONHUNTER.default_hero
    return None
```

Figure 6.4: `class_converter` function

The simulator ends when a GameOver call is made, to save the results of the simulator in the `play_turn()` function, a catch exception was made which gets both players match results. Either `PlayState.WON` or `PlayState.LOST`. This is then written into a text file in order to save the results for further evaluation later.

```
except GameOver:
    print('Game End')
    #for some reason the players are sometimes switched (so player 1 becomes player 2),
    #to avoid this I have written this code
    if game.player1.name == name:
        print(game.player1.name, game.player1.hero, game.player1.playstate)
        with open('results.txt', 'w') as f:
            print(game.player1.playstate, file=f)
    elif game.player2.name == name:
        print(game.player2.name, game.player2.hero, game.player2.playstate)
        with open('results.txt', 'w') as f:
            print(game.player2.playstate, file=f)
```

Figure 6.5: `play_turn` function exception

### 6.2.3 External Code

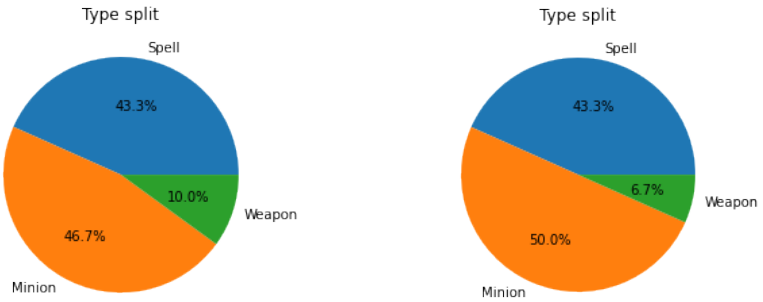
Originally, the execution of the simulator was going to be done through the library, however there was a problem encountered were the changed code would not get updated through the library. After many hours of attempting to get it working, another method was found to work. This method used Python's exec function, which essentially runs the code like it would be done in the terminal. The results of the simulator are then collected from the `results.txt`, which is then saved into an array to evaluate the win rate.

```
def run_sim_with_exec(file, args, num, enemy_class):
    results = []
    print("Running Generated deck against a", enemy_class, "\n", enemy_deck)
    for i in range(num):
        script_descriptor = open(file)
        a_script = script_descriptor.read()
        sys.argv = args
        exec(a_script)
        result = open("results.txt", 'r')
        line = result.readline()
        print(line)
        line = line.strip('\n')
        results.append(line)
    return results
```

Figure 6.6: function that runs the simulator

## 6.3 Comparing type split for real and fake decks

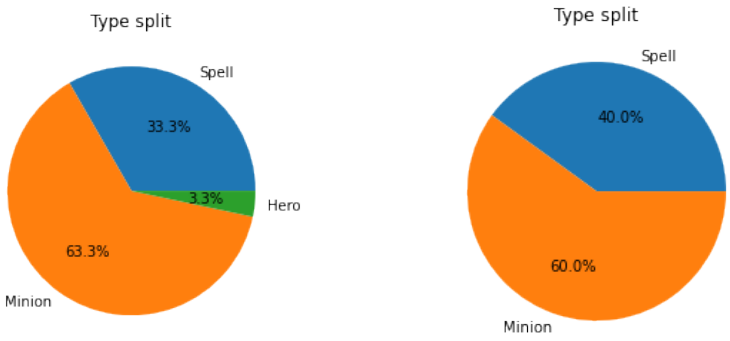
### 6.3.1 Test 1 - Hunter Deck




(a) Generated Deck (b) Real Deck

Figure 6.7: Type Split of real and generated deck

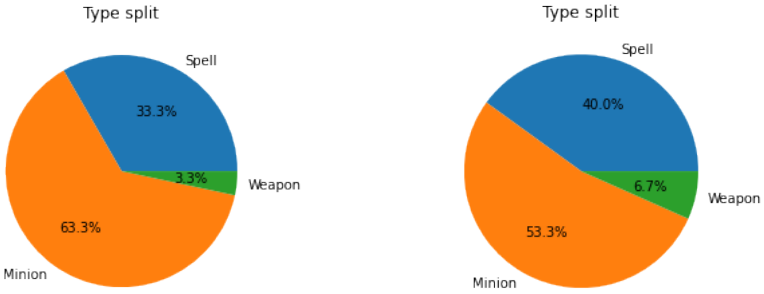
### 6.3.2 Test 2 - Warlock Deck




(a) Generated Deck (b) Real Deck

Figure 6.8: Type Split of real and generated deck

### 6.3.3 Test 3 - Shaman Deck

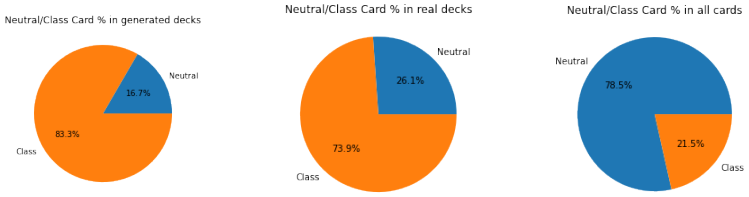



(a) Generated Deck (b) Real Deck

Figure 6.9: Type Split of real and generated deck

## 6.4 Comparing neutral split for real and fake decks

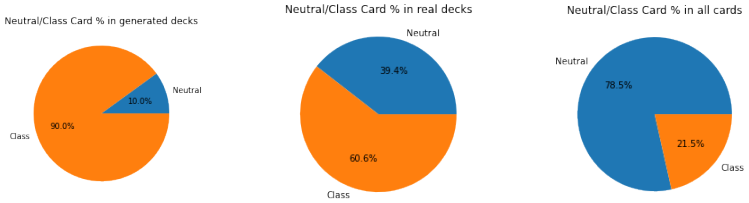
### 6.4.1 Test 1 - Hunter Deck




(a) Generated Decks (b) Real Decks (c) All Cards

Figure 6.10: Neutral split of real, generated deck and overall

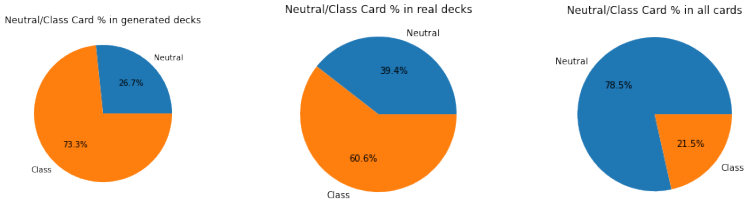
### 6.4.2 Test 2 - Warlock Deck




(a) Generated Decks (b) Real Decks (c) All Cards

Figure 6.11: Neutral split of real, generated deck and overall

### 6.4.3 Test 3 - Shaman Deck

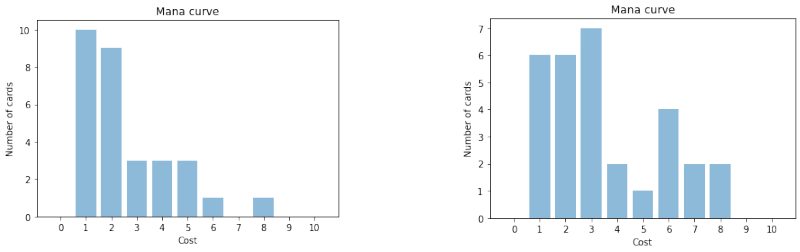



(a) Generated Decks (b) Real Decks (c) All Cards

Figure 6.12: Neutral split of real, generated deck and overall

## 6.5 Comparing mana curve for real and fake decks

### 6.5.1 Test 1 - Hunter Deck

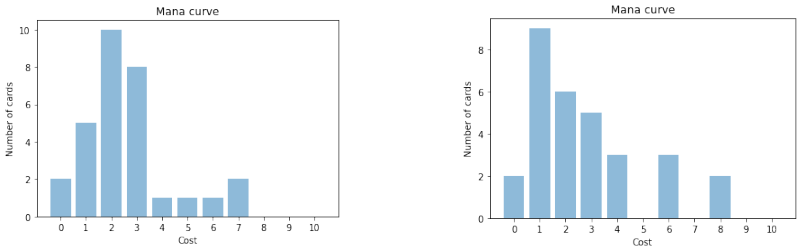



(a) Generated Deck

(b) Real Deck

Figure 6.13: Mana curve of real and generated deck

### 6.5.2 Test 2 - Warlock Deck

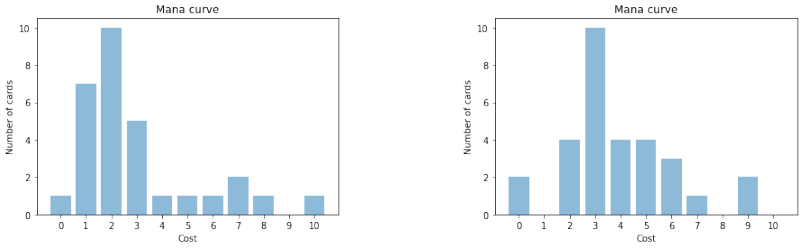



(a) Generated Deck

(b) Real Deck

Figure 6.14: Mana curve of real and generated deck

### 6.5.3 Test 3 - Shaman Deck

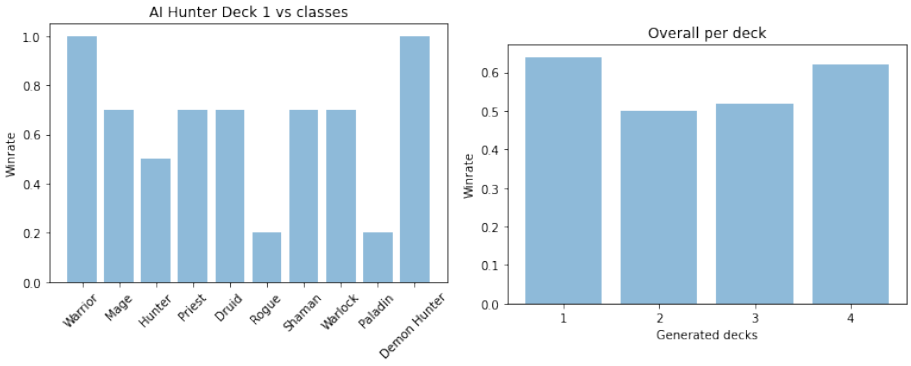



(a) Generated Deck (b) Real Deck

Figure 6.15: Mana curve of real and generated deck

## 6.6 Testing winrates versus real decks

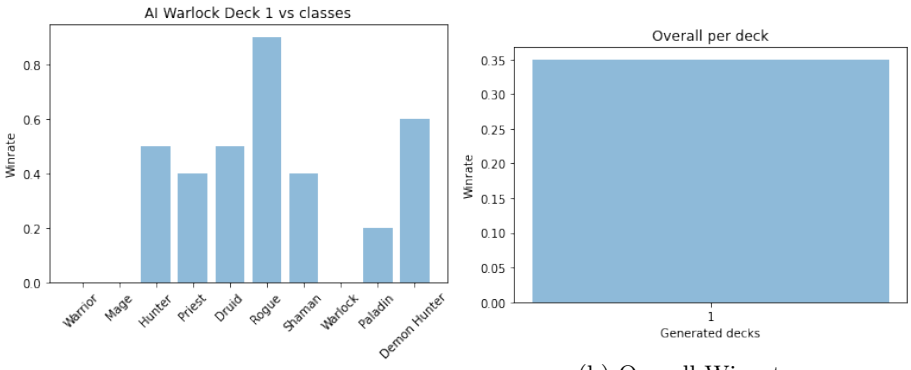
### 6.6.1 Test 1 - Hunter Deck




(a) Winrate Per class (b) Overall Winrate

Figure 6.16: Winrate of generated deck vs real decks

### 6.6.2 Test 2 - Warlock Deck

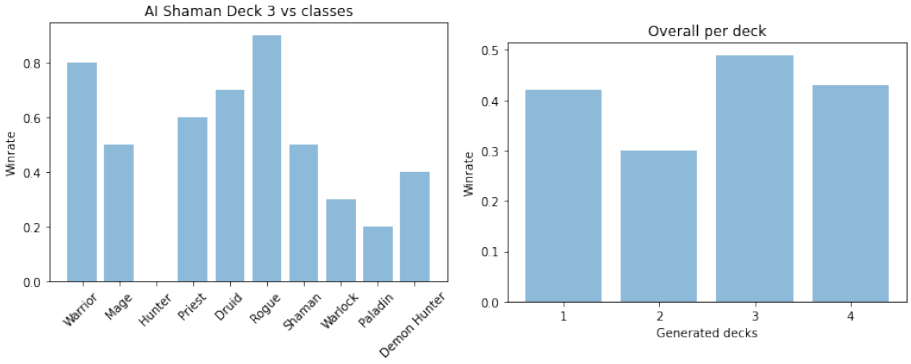



(a) Winrate Per class

(b) Overall Winrate

Figure 6.17: Winrate of generated deck vs real decks

### 6.6.3 Test 3 - Shaman Deck




(a) Winrate Per class

(b) Overall Winrate

Figure 6.18: Winrate of generated deck vs real decks

# 7. Evaluation

## 7.1 Overview

Over all, the project was a success, despite some limitation a new technique has been found to work for the building of Hearthstone decks. Although not as refined as other established techniques, another avenue has been opened for further study. Non-functional requirements have been fully accomplished, the most important functional requirements have been met save for a few such as wild format implementation, UI and build a deck with a limited collection of cards due to time constraints. Testing while inconclusive due to constraints has shown great promise in the structuring of a deck and provides interesting insights in the testing of a deck.

## 7.2 Limitations

Throughout the testing process there are major limitations preventing proper results from being recorded. The first limiting factor was the simulator, since only about half of the cards in the game have been implemented into the game, some of those cards would be present in the deck creation. Whilst the cards do exist in the simulator and can be played, there effects will not be registered by the simulator, for example a minion with an effect on board, if unimplemented will still have the health and the attack but will not have the effect of the card. For a spell, the mana will be spent but the effect will not trigger due to it being unimplemented, this ends up making cards that are considered good by community standards bad and actually becomes a detriment to the deck and lowers winrate. The second limiting factor is the AI that plays the deck, due to its simplistic nature, the cards within the decks are not played to their fullest potential, the AI seeminly plays cards randomly based on the amount of mana it has available, because of this decks that require a resource management will not have the most representative winrate than that of a human player. These combined factors prevent the testing results for winrates from being conclusive, should the project be continued the possibility of using a more advanced AI such as a Monty Carlo algorithm could bear fruitful results.

## 7.3 Results

### 7.3.1 Comparing type split for real and fake decks

The results of the comparison of the types split are a massive success, the differences between the real and generated decks are minor. Decks in Hearthstone generally have more minions than any other type of card, except for specific minionless archetypes. Then in close second comes spells, and finally some classes have weapon, others hero cards which are cards that transform the heros portrait into another with effects. From these results a user would not be able to distinguish a real deck from a fake (omitting that they can not see the cards).

### 7.3.2 Comparing neutral split for real and fake decks

The comparison of the split of neutral cards to class cards was important because it was the first graph that demonstrated that the GAN was thinking about what cards to have in the decks. This is because the split of neutral and class cards for all cards is majority neutral, there is close to 80% neutral cards and 20% class cards. Whereas with real decks there is a majority of class cards, the reason for this is due to class cards being more powerful than neutral cards, and generally are cornerstones to a decks archetype, neutral cards are used to fill in the extra spaces. This idea is reflected in the results of the generated decks, there is a majority class than neutral, even more so than in real decks. Overall, despite their being slightly more class cards than expected, the results were successful.

### 7.3.3 Comparing mana curve for real and fake decks

The mana curve of a deck is important, especially when talking about the archetype. Most aggressive deck mana curves tend to peak in the early cost cards, this is so that they can overwhelm the opponent quickly and finish off the enemy before they can get access to their powerful cards. More control oriented decks tend to peak in the middle part of the curve with more higher cost cards than lower cost cards. The results of the mana curves vary a lot, most of the generated decks tend to peak in the earlier mana cost cards meaning that the GAN prefers to create aggressive decks, this could be the case because of the classes chosen, Hunters tend to have aggressive decks over control decks, also the current Hearthstone meta could be aggressive decks meaning that the GAN would lean towards that from the datasets containing more aggressive decks. Despite the mana curves looking different, there is still a similar structure that can be observed between the real and fake decks.

### 7.3.4 Testing winrates versus real decks

This part tested the overall winrate of all the generated decks by the GAN, and their winrate on a per class basis. From the results, it can be noticed

that the winrates of the Hunter and Shaman class are over 50% which is good as it fulfils the requirement set, however it can be observed that Warlock has an unusually low winrate, a possible explanation is due to the AI playing the deck, the AI tends to use hero powers whenever it can, and the Warlock's hero power damages himself, meaning that the AI is more likely to die as a Warlock, whereas as a Hunter, the hero power damages the enemy meaning that the winrate would be higher. Also some classes have a better winrate versus some classes, this is mostly due to match ups, an agressive deck like a hunter deck will have an easier time dealing with control decks like a warriors hence why there is 100% winrate for Hunter in that category. However it has trouble versus other midrange type decks, like a Rogue or Paladin. Overall these results are skewed by the AI not being very smart, but some interesting insights have come from it such as the match up dependance and the fact that the AI performs better with aggressive decks than it does decks that require more thinking.

–simulator limited– – logging buffered messages which crashed program– – Aggro decks provide better results because they require playing as many cards as possible whereas more control decks require resource management–

# Bibliography


[1] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996. [Online]. Available: https://books.google.fr/books?hl=en&lr=&id=htJHI1UrL7IC&oi=fnd&pg=PR9&dq=evolutionary+algorithms&ots=fBl_1QSCiT&sig=g5AzYmN078gAvTVrxhGqfPW-Ieg&redir_esc=y#v=onepage&q=evolutionary%20algorithms&f=false

[2] H. N. Ward, D. J. Brooks, D. Troha, B. Mills, and A. S. Khakhalin, "AI solutions for drafting in Magic: the Gathering," 9 2020. [Online]. Available: http://arxiv.org/abs/2009.00655

[3] J. Isaacs, "Academic writing and the literature review. honours project. robert gordon university, 14th of october," 2020, [Accessed: 30/10/20]. [Online]. Available: https://liverguac-my.sharepoint.com/:v:/g/personal/j_p_isaacs_rgu_ac_uk/EWsECIDDZHNDuCcAMkCajlEBg513KeDA7VYZjzDdqZIerg?e=ihykeq

[4] "Hearthstone." [Online]. Available: https://playhearthstone.com/en-us

[5] "What is skill and luck - multiplayer discussion - hearthstone forums." [Online]. Available: https://us.forums.blizzard.com/en/hearthstone/t/what-is-skill-and-luck/4415

[6] CCGer, "Deck building vs skillfull play. 29th december." 2011, [Accessed: 27/10/20]. [Online]. Available: https://www.mtgsalvation.com/forums/magic-fundamentals/magic-general/327490-deck-building-vs-skillfull-play

[7] F. De, M. Silva, M. C. Fontaine, R. Canaan, J. Togelius, S. Lee, and A. K. Hoover, "Evolving the hearthstone meta." [Online]. Available: https://github.com/HearthSim/SabberStone

[8] R. Robertson, "Jungian archetypes: Jung, gödel, and the history of archetypes," 2016. [Online]. Available: https://books.google.fr/books?hl=en&lr=&id=tLJgDAAAQBAJ&oi=fnd&pg=PT7&dq=jungian+archetypes&ots=56eqtqa6K_&sig=Tmk2iRpe1XHy9l_kFD3Yc_N-Gnc&redir_esc=y#v=onepage&q=jungian%20archetypes&f=false

[9] J. Judlick, "Identifying deck archetypes - articles - tempo storm." [Online]. Available: https://tempostorm.com/articles/identifying-deck-archetypes

[10] Gamepedia, "Standard format," 2020, [Accessed: 04/11/20]. [Online]. Available: https://hearthstone.gamepedia.com/Standard_format

[11] G. Zuin and A. Veloso, "Learning a resource scale for collectible card games," vol. 2019-August. IEEE Computer Society, 8 2019.

[12] A. Stiegler, C. Messerschmidt, J. Maucher, and K. Dahal, "Hearthstone deck-construction with a utility system." Institute of Electrical and Electronics Engineers Inc., 5 2017, pp. 21–28.

[13] B. Entertainment, "Celebrating 100 million players!" 2018, [Accessed: 05/11/20]. [Online]. Available: https://playhearthstone.com/en-us/news/22636890

[14] K. T. Howard, "Free-to-play or pay-to-win? casual, hardcore, and hearthstone," *Transactions of the Digital Games Research Association*, vol. 4, pp. 147–169, 10 2019. [Online]. Available: http://todigra.org/index.php/todigra/article/view/103

[15] P. Sweetser and J. Wiles, "Current ai in games : a review," *Australian Journal of Intelligent Information Processing Systems*, vol. 8, no. 1, pp. 24–42, 2002. [Online]. Available: https://eprints.qut.edu.au/45741/

[16] A. K. Hoover, J. Togelius, S. Lee, and F. de Mesentier Silva, "The many ai challenges of hearthstone," *KI - Kunstliche Intelligenz*, vol. 34, pp. 33–43, 3 2020. [Online]. Available: https://link.springer.com/article/10.1007/s13218-019-00615-z

[17] G. N. Yannakakis and J. Togelius, *Artificial intelligence and games*. Springer International Publishing, 2 2018.

[18] A. Janusz, T. Tajmajer, and M. Świechowski, "Helping ai to play hearthstone: Aaia'17 data mining challenge," in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2017, pp. 121–125.

[19] A. Santos, P. A. Santos, and F. S. Melo, "Monte carlo tree search experiments in hearthstone," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 272–279.

[20] M. Świechowski, T. Tajmajer, and A. Janusz, "Improving hearthstone AI by combining mcts and supervised learning algorithms," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018, pp. 1–8.

[21] D. Kowalczyk, "Writing research questions: Purpose and examples," 2013, [Accessed: 07/11/20]. [Online]. Available: https://study.com/academy/lesson/writing-research-questions-purpose-examples.html

[22] Flipperbw, "Simple hearthstone logging - See your complete play history without TCP, screen capture, or violating the TOS," 2014, [Accessed: 08/11/20]. [Online]. Available: https://www.reddit.com/r/hearthstone/comments/268fkk/simple_hearthstone_logging_see_your_complete_play/

[23] E. Bursztein, "I am a legend: hacking hearthstone using statistical learning methods," 2016, pp. 1–8. [Online]. Available: https://elie.net/static/files/i-am-a-legend-hacking-hearthstone-using-statistical-learning-methods/i-am-a-legend-hacking-hearthstone-using-statistical-learning-methods-paper.pdf

[24] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," 1994.

[25] M. C. Fontaine, F. D. M. Silva, S. Lee, J. Togelius, L. B. Soros, and A. K. Hoover, "Mapping hearthstone deck spaces through map-elites with sliding boundaries." Association for Computing Machinery, Inc, 7 2019, pp. 161–169. [Online]. Available: https://dl.acm.org/doi/10.1145/3321707.3321794

[26] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-44874-8

[27] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 6 1994. [Online]. Available: https://link.springer.com/article/10.1007/BF00175354

[28] J. J. Merelo, F. Liberatore, A. F. Ares, R. García, Z. Chelly, C. Cotta, N. Rico, A. M. Mora, and P. García-Sánchez, "There is noisy lunch: A study of noise in evolutionary optimization problems," in *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, vol. 1, 2015, pp. 261–268.

[29] S. J. Bjørke and K. A. Fludal, "Sverre johann bjørke knut aron fludal deckbuilding in magic: The gathering using a genetic algorithm," 2017. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2462429

[30] P. Garcia-Sanchez, A. Tonda, G. Squillero, A. Mora, and J. J. Merelo, "Evolutionary deckbuilding in hearthstone," vol. 0. IEEE Computer Society, 7 2016.

[31] P. García-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, "Automated playtesting in collectible card games using evolutionary algorithms: A case study in hearthstone," *Knowledge-Based Systems*, vol. 153, pp. 133 – 146, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705118301953

[32] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. [Online]. Available: https://books.google.fr/books?id=0jbxwQEACAAJ

[33] 3blue1brown. (2017) But what is a neural network? — deep learning, chapter 1. [Accessed: 25/10/20]. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk

[34] F. Bre, J. Gimenez, and V. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, 11 2017.

[35] J. Jakubik, "A neural network approach to hearthstone win rate prediction," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2018, pp. 185–188.

[36] Q. H. Vu, D. Ruta, A. Ruta, and L. Cen, "Predicting win-rates of hearthstone decks: Models and features that won aaia'2018 data mining challenge," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2018, pp. 197–200.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[38] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. [Online]. Available: http://arxiv.org/abs/1701.00160

[39] R. Rodriguez Torrado, A. Khalifa, M. Cerny Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," in *2020 IEEE Conference on Games (CoG)*, 2020, pp. 41–48.

[40] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 596–601.

[41] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 12 2017. [Online]. Available: http://arxiv.org/abs/1712.06567

[42] A. Nigam, P. Friederich, M. Krenn, and A. Aspuru-Guzik, "Augmenting genetic algorithms with deep neural networks for exploring the chemical space," *arXiv*, 9 2019. [Online]. Available: http://arxiv.org/abs/1909.11655

[43] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," pp. 33–72, 1993. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4615-2740-4_3

[44] "Ethics and data protection," 2018, [Accessed: 03/04/21]. [Online]. Available: https://ec.europa.eu/info/sites/info/files/5._h2020_ethics_and_data_protection_0.pdf

# 8. Appendix

## Appendix A

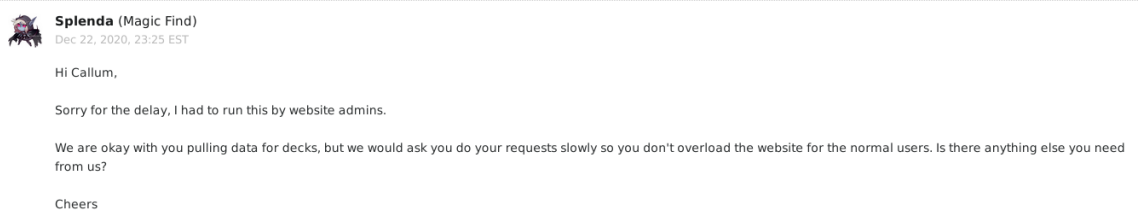
### 12. Prohibited Conduct

You are granted a non-exclusive, non-transferable, revocable license to access and use the Services, strictly in accordance with these Terms. As a condition of your use of the Services, you represent and warrant to Company that you will not use the Services for any purpose that is unlawful or prohibited by these Terms. Further, you agree that you will comply with these Terms and will not:

- Use the Services in any manner which could damage, disable, overburden, or impair the Services or interfere with any other party's use and enjoyment of the Services;
- Obtain or attempt to obtain any materials or information through any means not intentionally made available or provided for through the Services;
- Impersonate any person or entity, falsely claim an affiliation with any person or entity, or access the Services accounts of others without permission, forge another person's digital signature, misrepresent the source, identity, or content of information transmitted via the Services, or perform any other similar fraudulent activity;
- Harvest or collect the email addresses or other contact information of other users from the Services;
- Defame, harass, abuse, threaten or defraud users of the Services, or collect, or attempt to collect, personal information about users or third parties without their consent;
- Remove, circumvent, disable, damage or otherwise interfere with security-related features of the Services;
- Reverse engineer, decompile, disassemble or otherwise attempt to discover the source code of the Services or any part thereof, except and only to the extent that this activity is expressly permitted by the applicable law of your country of residence;
- Modify, adapt, translate or create derivative works based upon the Services or any part thereof, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation;
- Access any website, server, software application, or other computer resource owned, used and/or licensed by Company including but not limited to the Services, by means of any robot, spider, scraper, crawler or other automated means for any purpose, or bypass any measures Company may use to prevent or restrict access to any website, server, software application, or other computer resource owned, used and/or licensed to Company, including but not limited to the Services;
- Interfere with or disrupt the Services or servers or networks connected to the Services, or disobey any requirements, procedures, policies or regulations of networks connected to the Services;
- Attempt to indicate in any manner that you have a relationship with Company or that Company has endorsed you or any products or services for any purpose; and
- Use the Services for any illegal purpose, or in violation of any local, state, national, or international law or regulation, including, without limitation, laws governing intellectual property and other proprietary rights, data protection and privacy.

Prohibited Conduct section of Hearthpwn Terms of Service

# Appendix B



**Splenda** (Magic Find)
Dec 22, 2020, 23:25 EST

Hi Callum,

Sorry for the delay, I had to run this by website admins.

We are okay with you pulling data for decks, but we would ask you do your requests slowly so you don't overload the website for the normal users. Is there anything else you need from us?

Cheers

Permission email from Hearthpwn team[1]

# Appendix C

Ancestral Healing
Backstab
Blur
Circle of Healing
Desk Imp
Embiggen
First Day of School
Forbidden Words
Inner Rage
Innervate
Lazul's Scheme
Lightning Bloom
Moonfire
Mutate
Power Word: Shield
Preparation
Raise Dead
Sacrificial Pact
Shadowstep
Silence
Totemic Might
Totemic Surge
Whispers of EVIL
Wisp
Abusive Sergeant
Acornbearer
Activate the Obelisk
Adorable Infestation

[0] https://www.magicfind.us/terms/

```
Aldor Attendant
Angry Chicken
Animated Broomstick
Arcane Breath
Arcane Missiles
Arcane Shot
Argent Squire
Athletic Studies
Battlefiend
Bazaar Burglary
Beaming Sidekick
Bestial Wrath
Blackjack Stunner
Blazing Battlemage
Blessing of Might
Blessing of Wisdom
...
Plague of Death
Rattlegore
Sathrovarr
Ysera
Ysera, Unleashed
Ysiel Windsinger
Big Bad Archmage
C'Thun, the Shattered
Colossus of the Moon
Darkmoon Rabbit
Dimensional Ripper
Eye of the Storm
Jumbo Imp
Kalecgos
King Phaoris
Living Monument
Mind Control
N'Zoth, God of the Deep
Nagrand Slam
Nozari
Puzzle Box of Yogg-Saron
Pyroblast
Scrapyard Colossus
Sea Giant
Survival of the Fittest
The Amazing Reno
The Boom Reaver
Y'Shaarj, the Defiler
Yogg-Saron, Master of Fate
```

# Appendix D


```
[

    {
        "cardId": "DMF_119",
        "dbfId": "61648",
        "name": "Wicked Whispers",
        "cardSet": "Madness At The Darkmoon Faire",
        "type": "Spell",
        "rarity": "Rare",
        "cost": 1,
        "text": "Discard your lowest Cost card. Give your
            minions +1/+1.",
        "flavor": "When the Old Ones whisper, others will
            scream.",
        "artist": "E. Li & K. Turovec",
        "collectible": true,
        "playerClass": "Warlock",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/98
            c7c5af3e66f52772de7d15b86edf0f2e769c95b137e8e76a461e80791456fd
            .png",
        "locale": "enUS",
        "id": 15
    },
    {
        "cardId": "YOP_027",
        "dbfId": "61965",
        "name": "Bola Shot",
        "cardSet": "Madness At The Darkmoon Faire",
        "type": "Spell",
        "rarity": "Common",
        "cost": 2,
        "text": "Deal $1 damage to a minion and $2 damage
            to its neighbors.",
        "flavor": "On second thought, bolas aren't very
            good against flying worms.",
        "artist": "MAR Studio",
        "collectible": true,
        "playerClass": "Hunter",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/0
            c05d34fb1a4e9956326c776651f37649709559ffbf06747c6d7edf11f081f59
            .png",
```

```
    "locale": "enUS",
    "id": 16
},
{
    "cardId": "DMF_523",
    "dbfId": "61242",
    "name": "Bumper Car",
    "cardSet": "Madness At The Darkmoon Faire",
    "type": "Minion",
    "rarity": "Rare",
    "cost": 2,
    "attack": 1,
    "health": 3,
    "text": "<b>Rush</b>\\n<b>Deathrattle:</b> Add
       two 1/1 Riders with <b>Rush</b> to your hand
       .",
    "flavor": "Subject: Re: Unsafe Rides?\\n\\n\\
       nbump",
    "artist": "Ursula Dorada",
    "collectible": true,
    "race": "Mech",
    "playerClass": "Warrior",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/17
       cdc31567f5c47332f7e072e10678e608800d7fb453d98b96c4c51f2b49e62d
       .png",
    "locale": "enUS",
    "mechanics": [
        {
            "name": "Deathrattle"
        },
        {
            "name": "Rush"
        }
    ],
    "id": 17
},
{
    "cardId": "DMF_704",
    "dbfId": "61226",
    "name": "Cagematch Custodian",
    "cardSet": "Madness At The Darkmoon Faire",
    "type": "Minion",
    "rarity": "Common",
    "cost": 2,
    "attack": 2,
```

```
    "health": 2,
    "text": "<b>Battlecry:</b> Draw a weapon.",
    "flavor": "Cleans up after a dust up.",
    "artist": "Tang Ruiqian",
    "collectible": true,
    "race": "Elemental",
    "playerClass": "Shaman",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/
       fd7aff2cd493cba48b6673f2023bdee0b6e321483c1bd0a6314e160a5e43c104
       .png",
    "locale": "enUS",
    "mechanics": [
        {
            "name": "Battlecry"
        }
    ],
    "id": 18
},

. . .

{
    "cardId": "DMF_100",
    "dbfId": "61176",
    "name": "Confection Cyclone",
    "cardSet": "Madness At The Darkmoon Faire",
    "type": "Minion",
    "rarity": "Common",
    "cost": 2,
    "attack": 3,
    "health": 2,
    "text": "<b>Battlecry:</b> Add two 1/2 Sugar
       Elementals to your_hand.",
    "flavor": "\"Oh, I thought it was a desert
       elemental?\"",
    "artist": "Anton Zemskov",
    "collectible": true,
    "race": "Elemental",
    "playerClass": "Mage",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/9
       d7587b9799c607e3f1396b70c43a0a66b3956f55ced47a7cb6e9f7cc9915399
       .png",
    "locale": "enUS",
    "mechanics": [
```

```
            {
                "name": "Battlecry"
            }
        ],
        "id": 19
    },
    {
        "cardId": "YOP_019",
        "dbfId": "61957",
        "name": "Conjure Mana Biscuit",
        "cardSet": "Madness At The Darkmoon Faire",
        "type": "Spell",
        "rarity": "Common",
        "cost": 2,
        "text": "Add a Biscuit to your hand that
            refreshes 2 Mana Crystals.",
        "flavor": "So THIS is why she was AFK.",
        "artist": "Adam Byrne",
        "collectible": true,
        "playerClass": "Mage",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/
            b3f77a5edb992cee2b79e9e217375ffb7bd6c7ff6018c236ecf25c19583ae7ba
            .png",
        "locale": "enUS",
        "id": 20
    },
    {
        "cardId": "DMF_189",
        "dbfId": "61297",
        "name": "Costumed Entertainer",
        "cardSet": "Madness At The Darkmoon Faire",
        "type": "Minion",
        "rarity": "Common",
        "cost": 2,
        "attack": 1,
        "health": 2,
        "text": "[x]<b>Battlecry:</b> Give a random\\
            nminion in your hand +2/+2.",
        "flavor": "Just remember not to talk. It breaks
            character.",
        "artist": "Matt Dixon",
        "collectible": true,
        "playerClass": "Neutral",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/0599
```

```
            d607f4bc50a3327760e73f1839ec5ecce13b6f62e60b62316e31fba46234
            .png",
        "locale": "enUS",
        "mechanics": [
            {
                "name": "Battlecry"
            }
        ],
        "id": 21
    },
    {
        "cardId": "DMF_083",
        "dbfId": "61679",
        "name": "Dancing Cobra",
        "cardSet": "Madness At The Darkmoon Faire",
        "type": "Minion",
        "faction": "Neutral",
        "rarity": "Common",
        "cost": 2,
        "attack": 1,
        "health": 5,
        "text": "<b>Corrupt:</b> Gain <b>Poisonous</b>.",
        "flavor": "Snakes are just tentacles with fangs
            .",
        "artist": "Patrik Bjorkstrom",
        "collectible": true,
        "race": "Beast",
        "playerClass": "Hunter",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/84575
            c03daa3bd1698e0f9049e4a2cbbc2cafcc1037ee6722c390fd5606981f6
            .png",
        "locale": "enUS",
        "mechanics": [
            {
                "name": "Corrupt"
            },
            {
                "name": "Poisonous"
            }
        ],
        "id": 23
    }

]
```

# Appendix E

```
   cards,class
"[153, 153, 164, 226, 226, 230, 230, 250, 348, 348, 460, 472, 472,
477, 487, 497, 522, 522, 593, 652, 652, 841, 841, 891, 891, 1163, 1226,
940, 1232, 1254]",Warrior
"[32, 68, 228, 282, 296, 465, 502, 543, 567, 633, 699, 804, 815, 945,
964, 1013, 1060, 1107, 1114, 1130, 1239, 1246, 1251, 213, 234, 269,
453, 1205, 1213, 1254]",Mage
"[64, 64, 74, 74, 192, 249, 249, 312, 312, 509, 622, 622, 663, 663,
965, 965, 213, 304, 304, 550, 639, 639, 712, 712, 713, 713, 718, 718,
883, 883]",Hunter
"[149, 149, 318, 318, 364, 364, 483, 483, 553, 616, 616, 752, 752,
765, 765, 824, 900, 900, 1118, 1139, 1160, 1209, 1219, 1219, 1225,
1225, 696, 755, 1046, 1046]",Priest
"[17, 69, 69, 149, 149, 205, 205, 318, 318, 364, 364, 483, 483, 605,
616, 616, 707, 707, 765, 765, 1015, 1015, 1118, 1139, 1225, 972, 972,
1048, 1232, 1254]",Priest
"[164, 178, 206, 250, 295, 348, 472, 487, 497, 522, 591, 593, 629,
652, 772, 841, 891, 963, 1008, 1163, 1196, 1226, 453, 755, 935, 940,
1137, 1213, 1232, 1243]",Warrior

...

"[8, 8, 15, 15, 17, 17, 23, 23, 78, 78, 96, 96, 149, 149, 318, 318,
364, 364, 393, 393, 396, 396, 462, 462, 611, 611, 616, 616, 441, 441]",Priest
"[7, 7, 29, 29, 303, 303, 328, 328, 351, 400, 400, 556, 556, 717, 717,
761, 780, 780, 869, 869, 885, 885, 942, 942, 1037, 1127, 1128, 1175,
1216, 1216]",Paladin
"[10, 10, 12, 12, 13, 13, 61, 61, 126, 126, 302, 302, 334, 334, 411,
411, 448, 685, 748, 801, 801, 840, 840, 1032, 1229, 1034, 1034, 1180,
1218, 1254]",Druid
"[17, 17, 65, 65, 84, 84, 173, 173, 202, 202, 238, 284, 284, 289, 289,
321, 547, 547, 599, 599, 742, 742, 795, 795, 1038, 1038, 1062, 1062,
745, 745]",Warlock
"[224, 224, 303, 303, 351, 377, 377, 485, 485, 486, 486, 516, 691,
691, 761, 869, 869, 870, 870, 885, 885, 912, 912, 1216, 1216, 31, 31,
974, 974, 994]",Paladin
"[12, 12, 171, 171, 180, 180, 183, 183, 242, 242, 265, 265, 576, 645,
645, 664, 664, 684, 684, 911, 938, 107, 107, 189, 189, 449, 449, 441,
441, 782]",Shaman
"[2, 2, 16, 16, 19, 19, 133, 133, 140, 140, 159, 159, 204, 204, 281,
281, 292, 292, 427, 427, 444, 444, 497, 527, 521, 521, 635, 635, 939,
624]",Rogue
```

## Appendix F


```
[
    {
        "cardId": "EX1_277",
        "dbfId": "564",
        "name": "Arcane Missiles",
        "cardSet": "Basic",
        "type": "Spell",
        "faction": "Neutral",
        "rarity": "Free",
        "cost": 1,
        "text": "Deal $3 damage randomly split among all
            enemies.",
        "flavor": "You'd think you'd be able to control
            your missiles a little better since you're a
            powerful mage and all.",
        "artist": "Warren Mahy",
        "collectible": true,
        "playerClass": "Mage",
        "howToGet": "Unlocked at Level 1.",
        "howToGetGold": "Unlocked at Level 32.",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/10
            b306c7bb28614307e88ecad246a59d333085bab75e2e3d92a108b6756a18d3
            .png",
        "imgGold": "https://d15f34w2p8l1cc.cloudfront.net
            /hearthstone/
            b6f0e2ef6cc826bc3df7c9ff1ebe3c61db9931bda03db54a5d7274a0285ec101
            .png",
        "locale": "enUS",
        "mechanics": [
            {
                "name": "ImmuneToSpellpower"
            }
        ],
        "id": 1163
    },
    {
        "cardId": "EX1_277",
        "dbfId": "564",
        "name": "Arcane Missiles",
        "cardSet": "Basic",
        "type": "Spell",
        "faction": "Neutral",
```

```
    "rarity": "Free",
    "cost": 1,
    "text": "Deal $3 damage randomly split among all
       enemies.",
    "flavor": "You'd think you'd be able to control
       your missiles a little better since you're a
       powerful mage and all.",
    "artist": "Warren Mahy",
    "collectible": true,
    "playerClass": "Mage",
    "howToGet": "Unlocked at Level 1.",
    "howToGetGold": "Unlocked at Level 32.",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/10
       b306c7bb28614307e88ecad246a59d333085bab75e2e3d92a108b6756a18d3
       .png",
    "imgGold": "https://d15f34w2p8l1cc.cloudfront.net
       /hearthstone/
       b6f0e2ef6cc826bc3df7c9ff1ebe3c61db9931bda03db54a5d7274a0285ec101
       .png",
    "locale": "enUS",
    "mechanics": [
        {
            "name": "ImmuneToSpellpower"
        }
    ],
    "id": 1163
},

. . .

{
    "cardId": "DMF_100",
    "dbfId": "61176",
    "name": "Confection Cyclone",
    "cardSet": "Madness At The Darkmoon Faire",
    "type": "Minion",
    "rarity": "Common",
    "cost": 2,
    "attack": 3,
    "health": 2,
    "text": "<b>Battlecry:</b> Add two 1/2 Sugar
       Elementals to your_hand.",
    "flavor": "\"Oh, I thought it was a desert
       elemental?\"",
    "artist": "Anton Zemskov",
```

```
    "collectible": true,
    "race": "Elemental",
    "playerClass": "Mage",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/9
       d7587b9799c607e3f1396b70c43a0a66b3956f55ced47a7cb6e9f7cc9915399
       .png",
    "locale": "enUS",
    "mechanics": [
        {
            "name": "Battlecry"
        }
    ],
    "id": 19
},

{
    "cardId": "SCH_241",
    "dbfId": "59000",
    "name": "Firebrand",
    "cardSet": "Scholomance Academy",
    "type": "Minion",
    "rarity": "Common",
    "cost": 3,
    "attack": 3,
    "health": 4,
    "text": "<b><b>Spellburst</b>:</b> Deal 4 damage
       randomly split among all_enemy minions.",
    "flavor": "Too cool for school, too hot for
       thought.",
    "artist": "Mike Sass",
    "collectible": true,
    "playerClass": "Mage",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/
       fb1a7b1be4722b7a508287e4f3298a835b71f52acc92e6cf15d4e2596345be78
       .png",
    "locale": "enUS",
    "mechanics": [
        {
            "name": "Spellburst"
        }
    ],
    "id": 225
},
{
```

```
    "cardId": "CS2_029",
    "dbfId": "315",
    "name": "Fireball",
    "cardSet": "Basic",
    "type": "Spell",
    "faction": "Neutral",
    "rarity": "Free",
    "cost": 4,
    "text": "Deal $6 damage.",
    "flavor": "This spell is useful for burning
       things.  If you're looking for spells that
       toast things, or just warm them a little, you'
       re in the wrong place.",
    "artist": "Ralph Horsley",
    "collectible": true,
    "playerClass": "Mage",
    "howToGet": "Unlocked at Level 1.",
    "howToGetGold": "Unlocked at Level 44.",
    "img": "https://d15f34w2p8l1cc.cloudfront.net/
       hearthstone/27063
       a6c8d589362634acb41798c32ee6e04053842ccffd5b1f8e0c4cd9e8583
       .png",
    "imgGold": "https://d15f34w2p8l1cc.cloudfront.net
       /hearthstone/8
       e87a42598f8f41210ca04ff5ab99d1cc5d928809e9f4c9e2ab5ccf636a049f8
       .png",
    "locale": "enUS",
    "id": 1249
},
{
    "cardId": "CS2_029",
    "dbfId": "315",
    "name": "Fireball",
    "cardSet": "Basic",
    "type": "Spell",
    "faction": "Neutral",
    "rarity": "Free",
    "cost": 4,
    "text": "Deal $6 damage.",
    "flavor": "This spell is useful for burning
       things.  If you're looking for spells that
       toast things, or just warm them a little, you'
       re in the wrong place.",
    "artist": "Ralph Horsley",
    "collectible": true,
    "playerClass": "Mage",
```

```
        "howToGet": "Unlocked at Level 1.",
        "howToGetGold": "Unlocked at Level 44.",
        "img": "https://d15f34w2p8l1cc.cloudfront.net/
            hearthstone/27063
            a6c8d589362634acb41798c32ee6e04053842ccffd5b1f8e0c4cd9e8583
            .png",
        "imgGold": "https://d15f34w2p8l1cc.cloudfront.net
            /hearthstone/8
            e87a42598f8f41210ca04ff5ab99d1cc5d928809e9f4c9e2ab5ccf636a049f8
            .png",
        "locale": "enUS",
        "id": 1249
    }


]
```