Computing Equilibrium Strategies for Collectible Card Games with Counterfactual Regret Minimization

修　士　論　文

Computing Equilibrium Strategies for Collectible Card Games with Counterfactual Regret Minimization

（Counterfactual Regret Minimization を用いたトレーディングカードゲームの戦略計算）

指導教員　鶴岡 慶雅 教授

東京大学大学院情報理工学系研究科
電子情報学専攻

氏 名　48-186457 張昊宇

提 出 日　2020 年 1 月 30 日

Abstract

Research on two-person zero-sum imperfect information games has always been very important, and especially focusing on finding equilibrium strategies has received much attention. Researchers mostly choose poker games as representative objects of imperfect information games and have made excellent contributions in this test-bed. On the other hand, Collectible Card Games (CCGs) are another popular and more complicated type of card games because of the diversity of its gameplay. Same as two-person zero-sum imperfect information games, the research on CCGs is still in the exploratory stage, and researchers have proposed various approaches, such as reinforcement learning, supervised learning, and so on. This thesis proposes to provide a new aspect of exploring CCGs‘ equilibrium strategies via Counterfactual Regret Minimization (CFR). Besides, how to apply CFR algorithm to those more complex card games is discussed. Moreover, following the idea of solving poker cards, the convergence of the algorithm is verified first, and then tested on a small model. Finally, we gradually expand and enrich the game model, and bulid a smart agent who performed well in digital collectible card game.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

For a long time, people's exploration of artificial intelligence has never stopped, always thinking about various methods to create more powerful artificial intelligence (AI). Among them, naturally, the game has become an excellent place to develop and test artificial intelligence. Computer scientists have always been curious about game AI, not because they want to improve chess or poker skills, but because they want to continuously improve artificial intelligence algorithms and the ability to handle complex problems in the process. In fact, the history of game AI is almost as long as the history of artificial intelligence. A lot of research on artificial intelligence originates from studying how to build agents that can complete games or even defeat humanity under same conditions. The evolution of game AI has always been accompanied by advances in AI research.

The reason why game AI research chooses chess or card games as experimental objects, such as backgammon, checkers, chess, poker, and go, is mainly because they often have the following characteristics:

1) Each has a simple and clear rule, with clear judgment conditions and action rules;

2) Mastery of these chess and card games is often a manifestation of intelligence to a certain extent.

In fact, long before Alan Turing outlined the picture of artificial intelligence, computer scientists had begun to test their "intelligent" programs on games.

In 1928, John Vonn Neuman published the Minimax algorithm, and in 1949 Claude Shannon reorganized the algorithm and used it to solve chess problems. In 1956, the Dartmouth Conference was held, and artificial intelligence was established as a discipline. In the same year, Arthur Samuel invented an algorithm that could overcome the Checkers game through self-learning, which is now called Reinforcement Learning. By 1992, Gerald Tesauro wrote TD-Gammon, which uses an artificial neural network as a model and uses the TD-lambda algorithm for training. Through a large number of self-games, TD-Gammon has reached the level of top humans. The significance of TD-Gammon is not only that it uses reinforcement learning for training, but also that it does not require any feature engineering. Simply using the position of the chess piece as the input of the neural network can also train agents that reach the level of top human players. In the following 1990s, checkers and chess

AI surpassed human beings. In comparison, the state of Go is far more complicated than the above-mentioned chess games (each step can be selected from 19 * 19 types), and the strategy of playing go is very dependent on the evaluation of the game, so Go has always been considered more difficult than chess. After a series of efforts, in 2015, the DeepMind team developed AlphaGo, a program based on deep reinforcement learning on the basis of their predecessors, and successfully defeated European Go champion Fan Yan, which makes AlphaGo becomes the first program that defeats professional Go players on a 19-way board without giving up.

Compared to the above board games, poker, bridge, mahjong and other card games are considered to be another type of game. The players in these games often have asymmetric information, so these games are called imperfect information games. Due to asymmetric information and higher complexity of imperfect information games, AI that can cope with imperfect information games are also more difficult to design and implement. For example, in poker games such as Texas Hold'em, players can mislead their opponents by bluffing. Generally, it is believed that top human players have already mastered this technology (art), but AI is difficult to learn this skill. But researchers at various universities have been promoting the development of Texas Hold'em AI, from simulating the bluffing behavior of Texas Hold'em players to introducing game theory based methods for AI to learn how to master bluffing skills. Finally, in 2015, the CFR+ algorithm was introduced based on the previous solution to solve the two-player Limited Texas Hold'em, proving that the computer can beat humans under limited bets condition. In 2017, with the successive release of Libratus and DeepStack, AI successfully defeated the world's top human players in two-player No-Limit Hold'em. In 2019, Carnegie Mellon University in conjunction with Facebook AI released the successor version of Libratus, Pluribus, which successfully defeated professional poker players in six-player No-Limit Texas Hold'em. This breakthrough far exceeds the previous research on two-player zero-sum poker and a key milestone of research of imperfect information games.

Compared to "perfect information" board games such as chess and go, Texas Hold'em, bridge and mahjong such "imperfect information" card games are more challenging because they not only have "asymmetric information" but also have more hidden states space. The situation of imperfect information makes the complexity of the game strategy higher, which in turn makes the calculation complexity of the system based on the tree search and the CFR algorithm greater. Such properties make them closer to the decision-making process in human real life. Therefore, the breakthrough of this kind of game AI may be a milestone for the next game AI research. Thus, researchers have gradually expanded their attention to more different types of imperfect information games. For example, a class of very important imperfect information games, Real-Time Strategy (RTS) games, has become an excellent soil recently, and some fruitful results have been born. Deepmind and OpenAI, which we are familiar with in this research field, have tried to do research on the famous imperfect information real-time games Starcarft II and DOTA 2 respectively. Finally they both bulit an game artificial intelligence of level of human professional players.

In this paper, we chose to focus on a special type of card game, Collectible Card Game (CCG). Collectible Card Games (CCGs) are a variant type of ordinary card games, which means they are also intuitive rational decision-making environments with multiple players during game process. As same, they are constantly changing systems, in which agents interact with both environment and each participant. Further more because of the diversity of their game-play, CCGs are more complex than poker games and also make them closer to the decision-making process in human real life.

Although CCGs are very different from poker games, they are still essentially a multi-player zero-sum card game. Based on the above discussion, we already know that if an excellent set of strategies can be obtained, the agent will not lose to the top human players. Therefore, in this task, the goal is to find an equilibrium point and a Nash equilibrium strategy that each other cannot obtain by changing their own strategies. Deep reinforcement learning is also successful in this area, but there are still some problems left. By using reinforcement learning for self-play, if the learning converges, this Nash equilibrium strategy can be obtained. However, there is no known evidence shows that learning can be purely self-play through reinforcement learning. Therefore, the obtained agents are only experimentally strong, and there is no guarantee that a Nash equilibrium strategy can be universally obtained.

In order to effectively obtain an equilibrium strategy, since CCGs are still a zero-sum game, it is reasonable to follow the idea of solving Texas hold'em. On the premise of ensuring the convergence of the CFR algorithm, the CFR algorithm is applied to a complex CCGs game model, and optimization is required to ensure that the calculation is completed in an acceptable time.

1.2 Goals

The purpose of this study is to find a way to get Nash equilibrium strategies for adversarial multi-agent tasks which is two-player imperfect information games without using prior knowledge. First, verify the convergence of CFR algorithm on a complex action space, which is effective in ordinary card games. This method is considered to be a useful method, and it is tried to apply it to the CCGs battle model. A new ideas for solving games that is closer to human gaming behavior is provided.

1.3 Contribution

The main contributions of this research are the following:

- The convergence and divergence of the CFR algorithm on complex action space is verified. When game models and player actions are no longer as intuitive as poker games, the question of whether counterfactual regret is still applicable is confirmed.
- The problem of the establishment of the infoset tree of Collectible Card Games is raised and optimized. In the Collectible Card Games, a single player can make multiple legal actions in one turn, which makes this card game's exploration tree too deep. Here, the idea of replacing a single action with an action sequence of actions has been proposed. Also we create or rewrite several game engine to satify our propose approach which makes exsiting game simulator suitable for CFR algorithm.
- Using the infoset tree of imperfect information games, an agent capable of obtaining an equilibria strategy in Collectible Card Games is trained. Using the CFR algorithm, which has played an important role in poker games, we have found a way to guarantee a equilibria strategy.

1.4 Structure of This Thesis

In Chapter 2, we first briefly introduce the video game Hearthstone: Warcraft, which will give readers an understanding of its goals, mechanics, and technical challenges. This description needs to be made first, as other chapters will depend on some concepts of the game. Since this is a commercial, closed-source video game, the same chapter will introduce the simulator and related communities. The simulator is considered to be part of the behavior and mechanics of Hearthstone, and the related game community is where the programmers who tried to solve the autonomous agent problem of this game were gathered, and they also developed and open sourced these simulators. On this basis, we briefly introduced the perspective of existing research methods and their solutions.

Chapter 3 introduces the theories we rely on for the versatility of this game's autonomous agent. These theories have been greatly developed and fully applied in games with imperfect information game, and have been sufficiently verified in poker games, such as defeating top human players in the annual poker game competition. Based on these theories we applied a counterfactual regret based approach in an attempt to solve the problem.

This theory is then applied in Chapter 4 to a simplified version of the video game Hearthstone: Warcraft. This chapter discusses the design of simplified game models, which are hard coded logical game mode designed to verify whether this theory can be applied to the real game itself. This section also points out the difficulties we encountered when trying to make a simplified version of Hearthstone: Heroes of War, and how we can circumvent them to finally implement our ideas in more complex models. Then train in a model with more game mechanics added, and conduct qualitative and quantitative experiments on this algorithm. Here will also introduce how we rewrite the mature game engine to meet our experimental needs. Results and datasets were obtained by using the rewritten simulator.

Finally, we highlight the advantages and disadvantages of this theory, as well as its application in Hearthstone: Warcraft, and summarize the completed and unfinished work to give a general overview of the work. This section also introduces the development trajectory we follow and describes future work on improving the feasibility of the algorithm.

Chapter 2

Prerequisite knowledge

2.1 Collectible Card Games

Since the founding of Magic: The Gathering in the 1990s, collectible card games (CCGs) have been one of the most popular and profitable products in the entertainment industry. Recently the digital version also appeared, HearthStone: Hero Warcraft is one of the most popular digital CCGs. CCGs are turn-based card games in which players set their decks in advance and carefully select cards to have the opportunity to take advantage of powerful combinations during real matches. Because each card has a specific function, there are always complex and diverse game modes.

In addition to the game modes, cards are obviously a major component of the game. However, as the number of cards increases, the number of interactions or 'combos' also increases, which makes the strategy of such games dependening on the player's constructed deck. Therefore, most of the previous research focuses on the strategic exploration of fixed decks. That will be further explained in the subsequent exsiting methods section.

2.2 Hearthstone

$HearthStone : Heros of Warcraft$ (HS) is a an online 2-player turn-based zero-sum large strategic collectible card game with imperfect information created and developed by Blizzard Entertainment, released in March 2014 for PCs and April 2015 for iOS and Android devices. This free-to-play game usually let players against other human players but also have solo mode against AI bots. According to statistics from September 2014, more than 20 million users have registered, and just four months later, this number has increased by 5 million. Usually Blizzard's games are developed and produced by a team of 60 to 100 people, while the developer team behind Hearthstone consists of only five programmers. It was an extraordinary success.

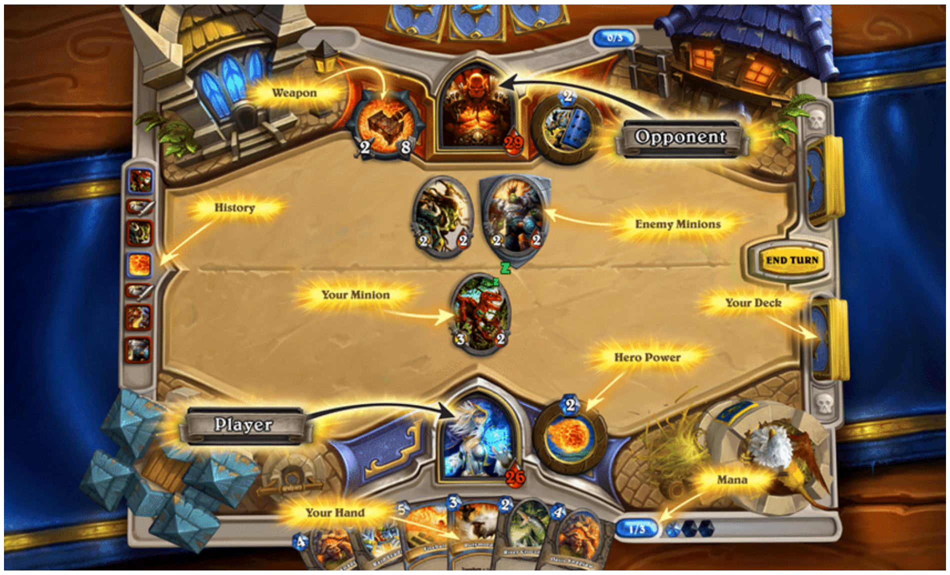


Figure 2.1: Screenshot of Hearthstone match

2.2.1 Game Description

The game has two boards on each side, with one player on each side. Each player chooses a hero. The hero has special power with no attack points (ATK), and starts the game with 30 HP. Throughout this article, we will use the words hero or player at the same time, which can be considered interchangeable game terms.

The game starts by tossing a coin to determine which player takes the shot first. The player then draws the starting card from his deck of 30 cards. The player who actions first draws three cards, the player who actions next draws four cards and gets a special card called The Coin. Before the game begins, both players can swap any of their starting cards with other cards from the top of their deck. Then put the cards they swapped back into the deck. This operation is defined to be completed during the card selection phase before the real battle starts. The goal of the game is to reduce the health of enemy heroes to zero or lower. To this end, players constantly choose different cards from their hands to play, these cards can be spells or characters (we call the latter one minion in this game). The player's hand is invisible to the opponent, that is, it is hidden, and the opponent can only see the card back. This is the embodiment of the inperfect information card game.

Hearthstone is a turn-based game means that two players will never take actions at the same time. At the beginning of a player's turn, if possible, current player would draw a card from his deck to his hand. He can use any number of cards from his hand throughout the this turn. If he placed any minions in previous rounds and it is alive, he can order them to attack enemy characters (heroes or minions). Players can also use the hero's special power once per round. Finally, players can choose to end their round at any time during thier round time. Players usually end turns when the goal has been achieved or there is no more action available. If a player cannot draw because his deck is empty, he will receive fatigue damage. This unavoidable situation will cause the current player's hero damage equal to the number of times he tries to draw a card when his deck is already empty. This mechanism prevents endless games from happening. There is a screenshot of Hearthstone match shown in Fig. 2.1 which shows the various features of the battle. Many game terms have appeared above. In order not

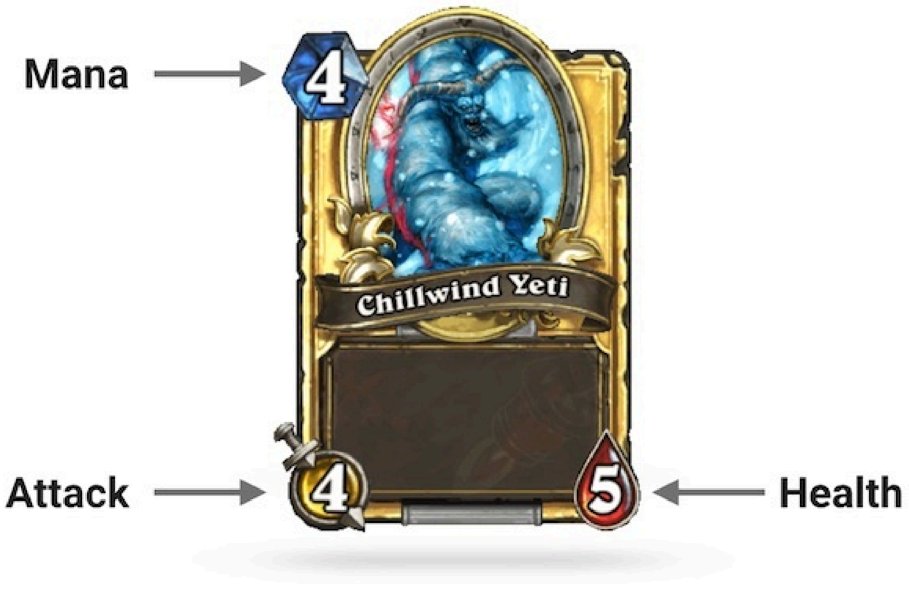
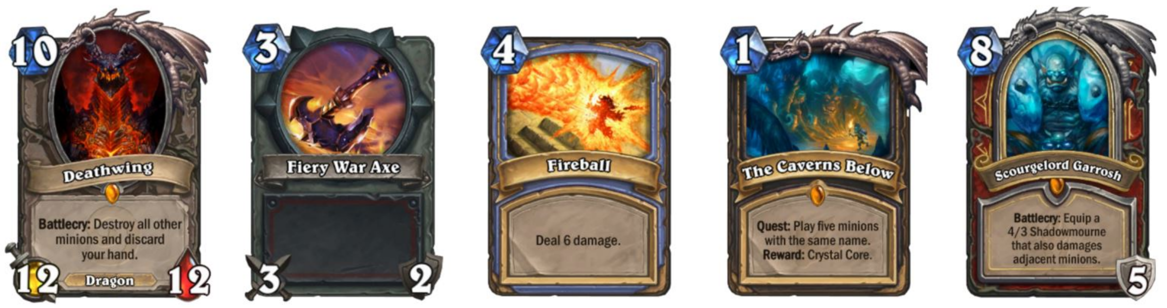


Figure 2.2: Example of minion

to cause misunderstanding, these terms will be explained immediately below:

- $ManaPoints$

  To use cards or heroes' special power, players need mana points (MP). Mana is a resource inherent to the player. Mana points also called mana crystal is required to successfully use the card in hand. In the first round, each player has one mana crystal. At the beginning of each round, the maximum number of mana crystals of current player increases by 1 until it reaches a maximum of ten, and all mana crystals are replenished. Special power and cards (whether they are spells or minions or weapons) require mana cost. For example, to be able to play cards with a mana cost of N, a player must have at least N MPs available. In other words, if he chooses to do so, he can successfully use this card, which will consume N MP in his mana reserve. If there are remaining MPs after deducting N MPs, players can continue to choose to consume crystals to play cards or other actions. An example of minion is shown in Fig. 2.2 which cost mana of four with four attack and five health points.If the player chooses to end the current round at this time, the remaining mana crystals will be ignored and will not be accumulated in the next round that means he does not want to use those MPs and waste them.


10
Deathwing
Battlecry: Destroy all other minions and discard your hand.
12
Dragon
12
3
Fiery War Axe
3
2
4
Fireball
Deal 6 damage.
1
The Caverns Below
Quest: Play five minions with the same name.
Reward: Crystal Core.
8
Scourgelord Garrosh
Battlecry: Equip a 4/3 Shadowmourne that also damages adjacent minions.
5

(a) Minion (b) Weapon (c) Spell (d) Quest (e) Hero

Figure 2.3: Example of different type of cards

- *Cards*

  The collectible Hearthstone card is the core of its gameplay and one of its most attractive features, because powerful cards can give players a huge advantage. The card represents a kind of behavior in a sense. Players use the card and consume the corresponding mana crystal to produce effects, and these effects will affect the game itself. Each card is associated with descriptions and effects or abilities. There are three main types of cards: minions, spells, and weapon cards. In newer version, quest cards and hero character cards have also been added. To win, players need to use several types of cards to reduce their opponent's health from 30 to 0. Since cards are the core of this type of card game, we will introduce them in detail in a later section. Here is an intuitive example. An example is shown in Fig. 2.3 including a representative of different type of cards.

- *Decks*

  Building decks is one of the most important parts of the game's experience. Although using the best cards is obviously important and ideal, players are limited to building a deck of 30 cards, and only can play cards already in the deck. Although there are virtually unlimited Hearthstone decks, only a few can perform well enough in professional at least semi-professional competitions. In contrast, some other decks performed much worse. The player must first start with the public available card pool and the hero's unique card pool, and carefully choose the deck that the player wishes to use in the game. It is required that no more than two cards can be added to the same deck. Apparently deck-building is a complex activity that requires understanding the current version of the game and evaluating other players' deck. So it's no surprise that this topic accounts for almost the vast majority of articles and discussions among participants. Most of them discuss how to build a strong deck and how to use it properly. The latter is the game strategy we will specifically introduce below.

- *Heroes*

  When the player try to build a deck, the game will ask the player to choose a character, which we call a hero. As mentioned earlier, players can only choose cards from the public card pool and the hero's unique card pool to build a deck. The hero's unique card pool here is determined by seven different heroes. They are *Druid*, *Mage*, *Hunter*, *Paladin*, *Priest*, *Hooligan*, *Shaman*, *Warlock*, and *Warrior*. In addition, each hero has different hero skills (consumes 2 crystals), which is also a special power we mentioned. Combined with its unique card pool, each hero has a different prototype of the deck with characteristics. For example, the Priest's healing ability is a very powerful choice for the control deck, but not so convenient for the aggro deck. We would talk about control deck and aggro deck later. On the other hand, we can also treat heroes as a kind of special minions with 30 health and 0 attack. In the game model program we wrote for confirming the convergence of our proposed algorithm, it did exactly the same. That makes game objects simple, and makes it easy for us to write properties and methods of Class. Normally, heroes do not have attack, so they cannot attack. However, spells or other special abilities will enable them to do so. For example, a hero can attack with a weapon, and the number of attacks depends on the weapon. The game ends if and only if a hero's health is less than or equal to 0.

- *Board*

  The board is a visual interface that the game provides to players. Of course, it can also be understood as a battlefield linking players on both sides. On the board we can get all information about the progress of the game. First and foremost, we can know the status of the enemy, such as the remaining health of the opponent hero, the number of minions of opponent, along with attack power and health, and so on. These are decisive messages on how we should act in the game. Minions already on the battlefield can attack each other, and heroes can use hero skills or cards on the battlefield. In addition, it is important to evaluate who is leading the board, because in most CCGs, the strategy that leads to win is to control the board by trading minions, and then use the minions on the board to kill the opponent's hero. Strategies is also the focus of this paper.

2.2.2 Cards

As we mentioned earlier, cards are the core of its gameplay and one of its most attractive features. And from a game perspective, it is a behavior that can affect the game itself. It is essentially similar to the bet action in Texas hold'em, except that the use of cards has a more complex effect because the cards themselves have some characteristics. These features are discussed below.

2.2.2.1 Cards Description

Cards usually have descriptive text on them. (Of course, there are cards without any description, we call them "whiteboard cards.") This text expresses the different functions of the card: either the effect when it is played, or the reaction to an event, or how to have effect on the game when the card is on the board. The description can be almost anything. It must be mentioned here that an implicit rule in Hearthstone is that any rule can be broken by the cards' description. This is not the same as most games are restricted by rules. Hearthstone's game behavior depends on the card on the board and its description. For example, in general game rules, players can choose to attack opponent's heroes or

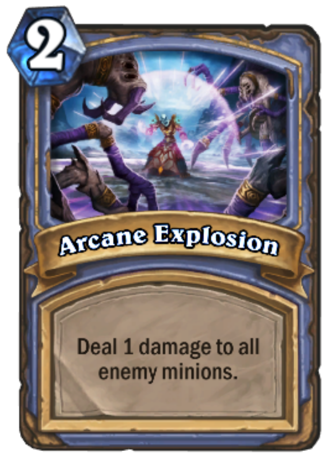


Figure 2.4: A spell card with direct damage effect

minions, but some cards will make the heroes unable to be selected. Therefore, as long as the card is on the board, this modification of the game rules will cause players can only choose to attack minions, which makes the action space smaller. When the effects described by multiple cards overlap, unless otherwise specified by other cards, the card with the most effective card prevails, or several effects are superimposed linearly. It can be expected that this can be a big challenge for game modeling and algorithm implementation. Examples of card description can be seen in Fig. 2.4-2.6.

We briefly introduce several of these features here, because we have added all these features to subsequent experiments.

- *Battlecry*

  Battlecry has always been one of the most common abilities in Hearthstone. When a card with the Battlecry effect is used, a specific description is activated. Although most Battlecry are limited to minions, some weapons also carry a battle cry. Battlecry represents the easiest way for a card to bring additional effects.It have many different effects and is determined by specific description.

- *Deathrattle*

  Deathrattle refers to the ability to produce the effect described when destroying minions or weapons. Cards with Deathrattle show skulls and crossbones in the game.

- *Taunt*

  Taunt is one of the first and most powerful abilities in Hearthstone. It forces opponent players to attack objects with this ability, and cannot make melee attacks on any other objects that do not have this ability. Minions with taunts protect their allies by forcing the enemy to fight them first, preventing them from attacking other friendly minions or protecting heroes until the

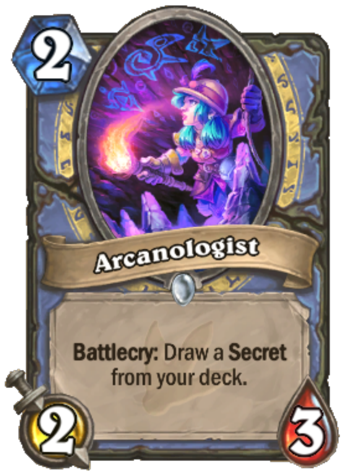


Figure 2.5: A minion card with *Battlecry* ability

taunt is removed. However, taunt has no effect on spells, hero skills, or other destructive effects. Minions with taunts appear on the battlefield in shields.

- *Aura*

  Auras provide a continuous change to the affected objects, which is removed at some specific events such as minion is removed. Auras also have many different effects and is determined by specific description.

- *Effect*

  A spell card always can trigger a one-time effect or ability, as described in the card's description. This effect works immediately, of course, under the rules formed by all cards on the battefield. In contrast to minion cards or weapon cards always have a continue effect to the game, one-time effect occurs and discarded immediately after use.

- *Silence*

  Silence is an ability which removes all card description, whether it's deathrattle, taunts or frozen. In short, it negates all effects but does not minion type or battlecry which is already happend.

- *Frozen*

  Freezing is the ability to mark a minion or hero as frozen. The frozen character will miss the next round, which means that the object cannot take any action in the next round. After that, the frozen state will be released. Freezing only affects combat and does no damage. Note that the hero is frozen here, not the player. Player can take all other actions as usual, including using hero skills, playing cards, equipping weapons, and commanding minions. Frozen characters can

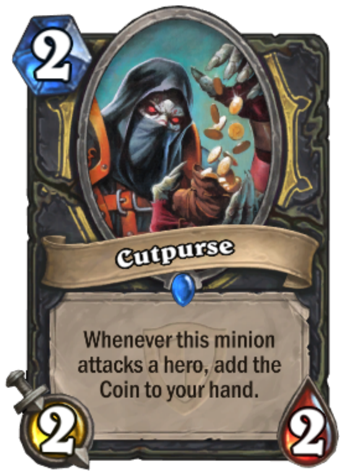

2
Cutpurse
Whenever this minion attacks a hero, add the Coin to your hand.
2
2

Figure 2.6: A minion card with lasting effect

still defend themselves when attacked. Silencing the Frozen character will remove the Frozen state.

2.2.3 Deck

We mentioned earlier that cards are the core component of Hearthstone, and the decks built by them are of course extremely important. Cards are divided into many card sets in the game. The complete list of card sets includes: Basic, Classic, Expansion, Adventure and Hall of Fame sets. The card sets of cards reflects how to get cards, which means collecting. Most basic cards are automatically included in the player's collection, and half of all class-specific basic cards are obtained by level heroes up to level 10. So we can choose 30 cards from the card set we want to use to build a deck. So how to choose these 30 cards is a very tricky problem.

2.2.3.1 Mana Curve

In most CCGs, each card has a cost, indicating the amount of resources required to use the card, which is the mana crystal in Hearthstone. The cost of cards is used to balance different cards, because mana crystals increase over time during the game process, and cheaper cards tend to be weaker, but can be played in early game stage, while expensive cards can be powerful enough to change the game rule which makes them only can be used when players have enough mana crystal at late stage of game. The magic curve is a histogram that represents the characteristics of the deck by digitizing the cost of each card. With this mana curve, it is easy to understand the deck prototype. In any case, the mana curve should generally maintain some kind of balance in order to give players ability to cope with any stage of game whether early, mid or late stage.

2.2.3.2 Deck archetypes

Because CCG's card pool is large, players can theoretically build an infinite variety of decks. However, some of these decks with specific patterns are particularly powerful. This particular model is called a deck archetypes. The deck archetype is a deck category formed by a specific subset of decks, and is usually used with a unique strategy to form a specific game style. Although each CCG has its own unique archetype, Hearthstone can roughly group all decks into three archetypes. Corresponding to these three archetypes, the appropriate strategy is to win by playing the maximum value of the card at the right time. For example, Aggro Hunter's strategy is to deal as much damage as quickly as possible, and Cube Warlock will defend until he has enough mana to effectively play a strong card. Each strategy has a counter strategy that may be more or less available to the opponent, which is also for game balance. Because the player chose definite 30 cards to form this deck, the player's strategy has been determined to some extent. This is also the main research direction of previous researchers, calculating fixed strategies for each different decks. These three prototypes are introduced below in general.

- *Aggro* which means "Aggression" and it is a type of deck driven by a relatively simple strategy: players try to finish the game in the early stages of the game and try to make the most damage to the opponent via maxmize resources. In general, if players with an Aggro deck cannot end the game fast enough, they will eventually fail in mid or late stage of the game. This deck usually has a large number of low-cost cards and the mana curve shifts to the left.
- $Mid - range$ type of decks means that players' main goal is to survive until they manage to draw all the necessary cards for combinations. The combination usually includes two or more core cards, allowing players to release considerable damage (ideally fatal) in one round, thus ensuring the winning of the game. Players with Archetypes of these decks may lose the game if the opponent is able to finish the game before all parts of all combinations are collected. Aggro decks are one of these opponents.
- *Control* is a type of deck of keeping opponents in control, and eliminates the threat of the early stage, and extends the game to the late stage, where they can use high cost, high value cards to end the game. Players with control decks may be at risk of failure if they cannot find a good solution to the cheap and effective threats to Aggro decks, or they cannot resist the lethal combination of Combo decks. This deck usually has a mana curve that moves to the right.

2.2.4 Challenge

According to the definition, the differences between CCGs and ordinary card games are very obvious. Although Hearthstone and poker are both games of imperfect information, we are familiar with the card features of poker. Each card of poker has a different number which represents the strength of this card. And Poker has a total of 54 cards. In certain poker games, we follow certain rules to restrict the order and the way of playing cards, thereby creating a competitive environment. In this environment, players perform one action in one round in most modes of poker games. Similarly, in Hearthstone, players also play cards in hands which draw from a deck containing 30 cards. But these cards are very different, they have a variety of functions, as we have mentioned before. Such a variety of cards can make the use of cards complicated. On the other hand, the opponent's deck is unpredictable, and unless he finishes all the cards, we cannot fully know his deck. This deck is selected from a larger card pool which contains more than two thousands cards. So we can't predict the strategies and actions of our opponents. In addition, poker players can perform only one action to play cards or pass, but Hearthstone players can perform many different sequences of actions in one round. In most CCGs, players have several action options, and their combinations make up the real sequence of actions. The order in which the actions are performed in every unique action sequence is crucial and determines interactions which we call 'combo'. That is the diversity of game play of CCGs and these differences make the game tree of CCGs too deep and large to handle.

2.3 Game Framework

Developing intelligent agent is a main research spot in the field of artificial intelligence, especially in the solving games. It focuses on the research of agents' cognition, learning, rational decision-making, problem solving and so on, so as to support the construction and implementation of intelligent application systems. An agent is a behavioral entity that resides in an environment and is capable of performing actions autonomously and flexibly to meet design goals. With the rapid development of computer networks, many application systems have become more and more complex. Especially the field of multi-agent reinforcement learning (MARL)[**?**] has expanded rapidly, and various methods have begun to appear and been discussed. These methods integrate the development of single agent RL, game theory and other fields.

2.3.1 Single-Agent

In single-agent reinforcement learning (RL), the environment of the agent is described by a Markov Decision Process (MDP).[**?**] A finite Markov Decision Process can be defined as a tuple $\langle S, A, P, R \rangle$:

- S is the collection of all states that the agent can observe from the environment. where s_t is used to indicate the state observed by the agent at time t. We can easily understand $s_t \in S$.
- A is the collection of all actions that the agent can take, where a_t is the action it takes at time t, and similarly $a_t \in A$.
- P is the probability distribution of state transitions, which is common in the Markov Process. We use $P(s_{t+1}|s_t, a_t)$ to describe the state transition that occurs at time t, which continues until the termination state.
- $R(s, a)$ is a reward function, according to which the agent will get a reward r_t from the environment, and the final reward is $R_t = \sum_{k=0}^{inf} \gamma^k r_{t+k}$.

In fact, there is also a parameter π, which is strategy, describes how the agent chooses its behavior in a particular environment or, as game theory definition, state. The strategy may be stochastic, in which $\pi \in [0, 1]$ or deterministic. If the strategy does not change over time, then we call it stationary.

The goal of reinforcement learning is to maximize this expected discounted benifit, which is called value function, at each step k:

$$v_\pi(s) = E_\pi\left\{\sum_{j=0}^{inf}(\gamma^j r_{k+j+1}|S = s_t)\right\} \tag{2.1}$$

in which $\gamma \in (0, 1]$ is the discount factor. R_k represents a very long-term benefit, and γ controls the future impact on the present, and the farther the future has less impact on the present. So the agent can try to maximize its long-term reward based on the benefits of this step.

Naturally, the action-value function is proposed. The action-value function (Q-function) is the expected discounted benifit of current state and chosen action based on the policy mentioned above :

$$Q_\pi(s, a) = E_\pi\left\{\sum_{j=0}^{inf}(\gamma^j r_{k+j+1}|S = s_t, A = a_t, \pi)\right\} \tag{2.2}$$

Then after a series of derivations, we can get the famous Bellman equation[**?**]:

$$v_\pi(s) = E_\pi\left\{r(s_{t+1}|s_t, a_t) + \gamma V_\pi(s_{t+1})|S = s_t\right\} \tag{2.3}$$

$$Q_\pi(s, a) = E_\pi\left\{r(s_{t+1}|s_t, a_t) + \gamma V_\pi(s_{t+1})|S = s_t, A = a_t\right\} \tag{2.4}$$

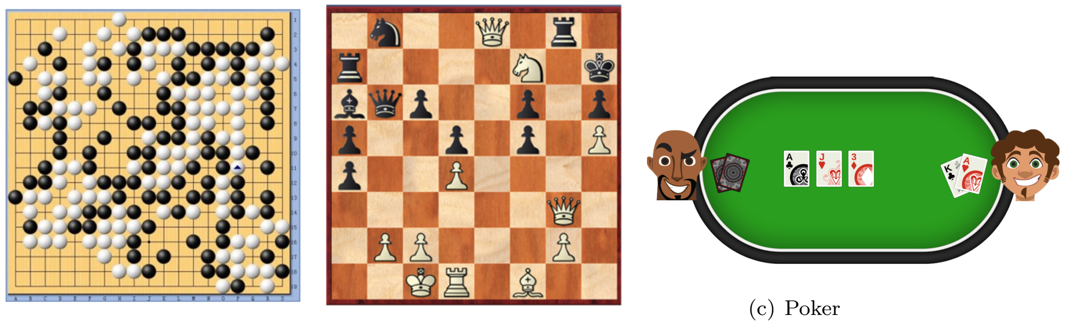
With these definitions and basic knowledge, a majority single-agent RL algorithms exist. There are model-based methods and model-free methods based on different approaches.

2.3.2 Multi-Agent

The multi-agent environment is based on the same concept of a single-agent model, but all agents decide the actions of the environment, not just any one of them. The biggest difference is that all of the agents can have totally different impact on the environment, so depending on what other agents are doing, the operation may have different results.

This is the challenge in the problem of applying single-agent techniques to the multi-agent field. Actually single-agent techniques designs are used to solve stationary environment, but now from the perspective of each agent, the environment is no longer stationary. In order to simulate such a situation, we have to move our attention to game theory [**?**], which aims to solve multi-agent situations. In particular, the most commonly used classification are perfect information game and imperfect information game.

In order to solve such a complicated imperfect information game, we use a mature algorithm Counterfactual Regret Minimization (CFR) [8] which is one of the most efficient methods for computing Nash equilibria in large, zero-sum, in imperfect information games. In the domain of poker, CFR has proven effective, and produced a lot of variants, such as MCCFR [9], CFR+ [10] and DisCFR [11]. In order to introduce CFR algorithm, some basic knowledge of game theory is discussed below.



(a) Go (b) Chess (c) Poker

Figure 2.7: Perfect and imperfect information game

2.3.3 Perfect Information

The perfection of information is an important notion in game theory when considering sequential and simultaneous games. It is a key concept when analysing the possibility of strategies in games. The concept of perfect information game is usually described with several characteristics:

- Each player knows the structure and operation mode of the game.
- Each player has the same information about the entire environment that would be available at any time of the game.

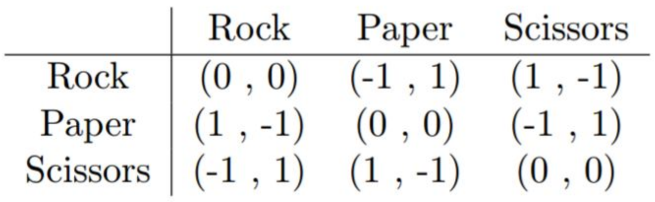
Good examples of perfect information would be Go, chess and other battle game where each player observes whole game situation and is informed by other player ’ s actions any time on the board.

2.3.4 Imperfect Information

Imperfect information is the opposite concept with perfect information in notion of perfection of information. Obviously they are complementary. So the concept of imperfect information game is described with characteristics below:

- Each player also knows the structure and operation mode of the game.
- Each player has the different information about the entire environment which means players only aware part of the whole.

Intuitively, the example that comes up immediately is the card game. In a card game, players can only see the cards in own hand and the cards on the table, but cannot know which cards are on the opponent’s hand. Examples of perfect information and imperfect information game is shown in Fig 2.7



	Rock	Paper	Scissors
Rock	(0 , 0)	(-1 , 1)	(1 , -1)
Paper	(1 , -1)	(0 , 0)	(-1 , 1)
Scissors	(-1 , 1)	(1 , -1)	(0 , 0)

Figure 2.8: Rock-Paper-Scissors

2.3.4.1 Matrix Game

The concept of a matrix game (MG) or strategy game is a simple framework for dealing with a single-shot game, which is a game with multiple players but only one state with an associated reward structure. Usually a matrix game can be described as a tuple $\langle n, A_{1...n}, R_{1...n}\rangle$:

- n is the agents number.
- A_i is the set of action that the agent i can take, where $A = A_1 \times A_2 \times ... \times A_n$ is the whole joint action set.
- R_i is the reward function of player i.

An important characteristic of MG is that the reward function of each agent depends on the behavior of all agents, not just its own behavior. The concept of matrix game based on each player's reward structure can be represented as an n-dimensional matrix. Like MDP, the game types which are considered have limited state and action space. That is to say, the state and action space are not infinite. The concept of strategy is similar to the policy in MDPs in the matrix games. Generally, the name pure strategy is a deterministic strategiy, and the way agent act like a random choice which are often referred as mix strategy. There is an associated reward for each of the players $R_i(\sigma)$ for every joint strategy, which can be defined as the rewards for individual actions:

$$R_i(\sigma) = \sum_{a \in A} R_i(a) \prod_{j=1}^{n} \sigma_i(a_i) \tag{2.5}$$

Nash equilibrium refers to such a situation in the game. For each player, as long as others do not change the strategy, he cannot improve his situation. Nash proved that Nash equilibrium exists in the premise that each player has only a limited number of strategic choices and allows for a mix strategy. That is an important property of Matrix Games. More details of Nash equilibrium will be discussed in following section. There is a way always be used to classifying Matrix Games is:

- Zero-sum games are two-player games (n = 2) in which one players' reward is always symmetric to the reward of the other player. An easily understand example is Rock-Paper-Scissors, which shown in Fig 2.8.

	Wait	Go
Wait	(0 , 0)	(1 , 1)
Go	(4 , 4)	(-2 , -2)

Figure 2.9: Shared Resource Channel

	Tell	Not Tell
Tell	(-2 , -2)	(0 , -4)
Not Tell	(-4 , 0)	(-1 , -1)

Figure 2.10: Prisoner ’ s Dilemma

- Team games have a fair number of players, but their rewards are the same for each joint action. There is an example of this kind of Matrix Game which is shown in Fig 2.9. The example simulates a shared resource channel where agents are working together, and one of the best choices is to make a concession.
- General-sum games are actually all types of matrix games. However, we always use this classification when a game is not belonging to zero-sum games. There is an example of general-sum games which is famous, named Prisoner's Dilemma shown in Fig 2.10.

2.3.4.2 Extensive Form Game

The extensive-form game is a natural model for sequential decision-making enviroments with multiple agents from game theory [2]. It uses a game tree to represent the entire game process. Tic-Tac-Toe is one of the simplest perfect information games and part of its game tree is shown in Fig 2.11. On each non-terminal game state players have several available actions and every terminal state contains rewards for each of players.

Players of Tic-Tac-Toe know exactly which game state they are on currently, and also understand which game state they would move to via observing the opponent's actions.

However, in imperfect information games, the situation is different because of the unobservability. The key difference is the information sets, which are sets of game states, in which players cannot distinguish which current state exactly they are in and so must choose actions for all such indistinguishable states with the same distribution. Thus the extensive-form of imperfect information games is denoted

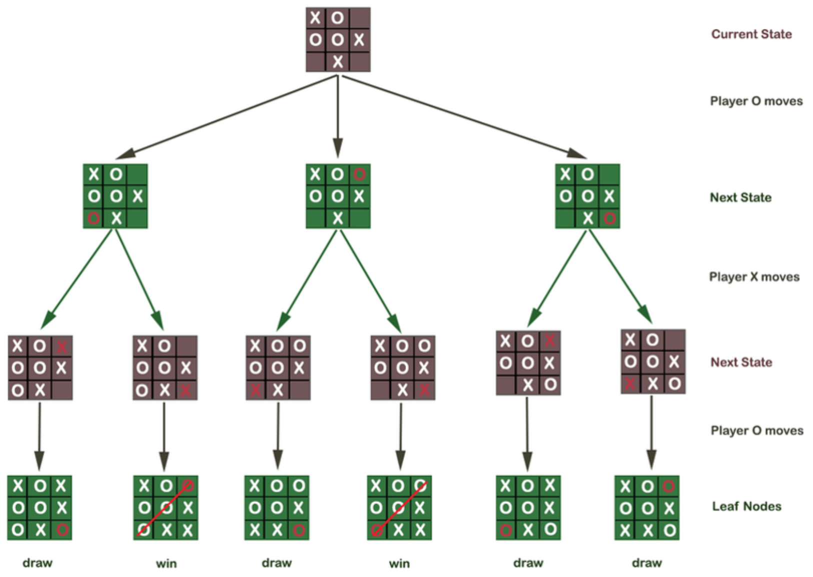


Figure 2.11: Part of game tree of Tic Tac Toe

as:

- N is a finite set of players.
- H is a finite set of sequences which is the histories of actions. $Z \in H$ are the terminal histories.
- $A(h)$ are the actions available after a non-terminal history $h \in H$.
- P is the player function, $P(h)$ is the player who takes an action under that history.
- I_i is a information set of player i belongs to $\mathcal{I}_i$ which is a information partition of player i.
- u_i is a utility function for each player i from the terminal states Z to the regrets R. If $N = (1, 2)$ and $u_1 =$ -u_2, it is a zero-sum extensive game. Define $\Delta_{u,i} = \max_z u_i(z) - \min_z u_i(z)$ to be the range of utilities to player i.

2.4 Strategies and Equilibria

2.4.1 Strategies

A strategy of player i, σ_i in an extensive game, is a function which represents a distribution over $A(I_i)$ to each $I_i \in \mathcal{I}_i$, and $\sum_i$ is the set of strategies of player i. A strategy profile σ consists of strategies for each player, σ_1, σ_2,..., with σ_{-i} referring to all the strategies in σ except σ_i. Let $\pi^\sigma(h)$ be the probability of history h if players choose actions according to σ. It naturaly decomposes $\pi^\sigma(h) = \Pi_{i\in N}\pi_i^\sigma(h)$ into each player's contribution to this probability. According to these notions, the overall value to player i of a strategy profile is the expected payoff $u_i(\sigma) = \sum_{h\in Z} u_i(h)\pi^\sigma(h)$, combining with each player would obtain a value at each final state of the game.

2.4.2 Nash Equilibrium

A tranditional solution concept of a two-player extensive game is the Nash equilibrium and it refers to such a situation in the game: for each player, as long as the other players do not change their strategies, the player cannot improve his own performance. The Nash equilibrium has been proved to exist in the premise that each player has only a limited number of strategic choices and allows for a mix strategy. This is an important property of extensive games. A Nash equilibrium is a strategy profile σ where

$$\begin{aligned} u_1(\sigma) &\geq \max_{\sigma_1^{'}\in\sum_1} u_1(\sigma_1^{'}, \sigma_2), \\ u_2(\sigma) &\geq \max_{\sigma_2^{'}\in\sum_2} u_1(\sigma_1, \sigma_2^{'}). \end{aligned} \tag{2.6}$$

An approximation of a Nash equilibrium or ϵ-Nash equilibrium is a strategy profile σ where

$$\begin{aligned} u_1(\sigma) + \epsilon &\geq \max_{\sigma_1^{'}\in\sum_1} u_1(\sigma_1^{'}, \sigma_2), \\ u_2(\sigma) + \epsilon &\geq \max_{\sigma_2^{'}\in\sum_2} u_1(\sigma_1, \sigma_2^{'}). \end{aligned} \tag{2.7}$$

2.5 Regret Minimization

Regret is an online learning concept, and a family of efficient learning algorithms has been developed. To define this concept, playing an extensive game repeatedly should be considered. Let σ_{-i}^t become the strategy used by player i on round t. Then we could get the average overall regret of player i at time T is:

$$R_i^T = \frac{1}{T} \max_{\sigma_i^{'}} \sum_{t=1}^{T} (v_i(\sigma_i^{'}, \sigma_{-i}^t) - v_i(\sigma^t)). \tag{2.8}$$

If we define $\overline{\sigma}_i^t$ to be the average strategy for player i from time 1 to T, then we could obtain:

$$\overline{\sigma}_i^t(I)(a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}. \tag{2.9}$$

for each $a \in A(I_i)$ and to each $I \in \mathcal{I}_i$. Between regret and the Nash equilibrium solution concept, previous mathematicians have proven there is a well-known connection.

Theorem *In a zero-sum game at time T, if both player's average overall regret is less than ϵ, then $\overline{\sigma}^T$ is a 2ϵ equilibrium.*
This algorithm is applying to select strategy for play i is minimizing the regret if player i's average overall regret goes to zero as $T \to \infty$. So, The regret minimization algorithm can calculate the approximate Nash equilibrium in the game through self-playing. In addition, the bounds of the algorithm on the average overall regret limit the convergence rate of the approximation.

Actually regret minimization is always applied to bandit problems comparing to extensive form games. Although it is possible to convert any finite extensive game into an equivalent normal form game, but the size of the representation increasesexponentially which makes regret algorithms are impractical for solving games. Although there is researchers has introduced the another methods which can minimize regret in extensive games by working with the realization plan representation. That is algorithms of Lagrangian Hedging (LH) family. But we still propose a regret minimization procedure that exploits the compactness of the extensive game.

2.6 Counterfactual Regret

The basic idea of our method is to break down the overall regret into a set of additional regret clauses that can be independently minimized. In particular, we introduced a new concept of regret for a extensive games, called counterfactual regret, which is defined based on some individual sets of information. We show that overall regret is limited by the sum of counterfactual regrets, and we also show how to minimize counterfactual regrets on each independent set of information.

Firstly, we take one arbitrary information set $I \in \mathcal{I}_i$ and player i's action as an object. Then we define $u_i(\sigma, h)$ is the expected utility when corresponding history h is reached and at that time all other players is playing undering a determinate strategy σ. Following this concepts, we could define $u_i(\sigma, I)$ is the counterfactual utility which equals to the expected utility when information set I is reached and all players act under strategy σ except that player. Then if we call $\pi^\sigma(h, h^{'})$ is the probability of coming from history h to history $h^{'}$:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi_{-i}^{\sigma}(h)\pi^{\sigma}(h)(h^{'})u_i(h^{'})}{\pi_{-i}^{\sigma}(I)}. \tag{2.10}$$

After that, we continue to define $\sigma|_{I\to a}$ is the strategy profile identical to σ except that player i always chooses action a at information set I. Then we could obtain the immediate counterfactual regret:

$$R_{i,imm}^T(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \pi_i^{\sigma^t}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I)). \tag{2.11}$$

Theoretically, this is the player's regret at information set I with counterfactual utility of his decisions, with an additional weighting term which is the counterfactual probability that inforamtion set I would be reached if that player tries to. But we always take the positive regret as consideration, then $R_{i,imm}^{T,+}(I) = \max(R_{i,imm}^{T}(I), 0)$ is defined as the positive portion of immediate counterfactual regret. According to the following statement :

Theorem $R_i^T \leq \sum_{I \in \mathcal{I}_i} R_{i,imm}^{T,+}(I)$
We could get the conclusion that minimize immediate counterfactual regret equals to minimize the overall regret. So it is possibe to find an approximate Nash equilibrium when minimizing the immediate counterfactual regret. Immediate counterfactual regret can be minimized by controlling $\sigma_i(I)$ is the point. In order to achieve that, Blackwell ' s algorithm for minimizing regret independently on each information set should be considered. Exactly for all $a \in A(I_i)$ and all $I \in \mathcal{I}_i$:

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I)). \tag{2.12}$$

If we define $R_i^{T,+} = \max(R_i^T(I, a), 0)$, then can obtain the strategy file for time $T+1$ is:

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)} & if \sum_{a \in A(I)} R_i^{T,+}(I, a) > 0 \\ \frac{1}{|A(I)|} & otherwise. \end{cases}$$

That is, the action is selected based on the number of positive counterfactual regrets for not playing the action. If there is no positive counterfactual regrets action, then action would be randomly chosen. As a result, we come to another result.

Theorem *If player i selects actions according to above equation then* $R_{i,imm}^T(I) \leq \Delta_{u,i}\sqrt{|A_i|}/\sqrt{T}$ *and consequently* $R_i^T \leq \Delta_{u,i}|\mathcal{I}_i|/\sqrt{|A_i|}/\sqrt{T}$ *where* $|A_i| = \max_{h:P(h)=i}|A(h)|$.
The results show that the strategy obtaining from above equation can be used for self-play to calculate Nash equilibrium. And the average overall regret limit is linear in the number of information sets. These limits are similar to what Gordon's Lagrangian hedging algorithm can achieve. At the same time, minimizing counterfactual regrets does not require expensive secondary program projections for each iteration.

Chapter 3

Related Research

In recent years, Hearthstone has gradually become a testbed for AI research. A group of enthusiastic players and developers have been involved in some projects and created multiple applications. And there are websites that are acquiring and summarizing user data, such as game results, deck composition, card statistics, and so on, and provide this information to relevant communities. So that researchers from the field of machine learning and AI can choose Hearthstone for research because it has accumulated a considerable amount of training data.

In previous research, researchers have full confidence in this new field and tried different methods to solve new problems. For example, some researchers have tried to use evolutionary algorithms to solve the problem of building better decks. Although this research is described as a preliminary study, these results they had still constructed a reasonable deck from a basic set of cards. This is very exciting, considering the uniqueness and variety of Hearthstone cards. Other researchers are also considering creating auto-agents that can play Hearthstone like human players. In particular, some researchers have used the Monte Carlo Tree Search (MCTS) algorithm to find the best action strategy in the game. In addition, other researchers use deep neural networks to improve the performance of MCTS-based agents. The combination of MCTS and predictive models makes these methods similar to earlier versions of DeepMind's AlphaGo program. However, it is quite different with Go, players in Hearthstone do not have perfect information about the state of the game, and many actions have inconclusive results. These two attributes shows the challenge of search ing the game tree. Of course, there are other perspectives here, and some studies have also tried to build models to predict opponents' cards in the game. Some studies can correctly predict the opponent's card by calculating the probability of the relevant card appearing. In more than 50 percentage of the cases, the card is successfully predicted to appear in the next game round. Such high predictability can be explained by the fact that, even with the huge number of Hearthstone possible decks, players still tend to build their own decks based on certain archetypes, and their composition is often influenced by other mature decks which are created by professional players. This is why we mentioned the three archetypes before. Speaking of using data to calculate correlations, Hearthstone is also the subject of international data mining competitions. AAIA's Data Mining Challenge: Help AI play Hearthstone, which is dedicated to developing a model to predict players' chances of winning. The successful model uses the integration of a one-dimensional convolutional neural network to extract features from each combination of two player cards on the board, thereby effectively predicting the player's win rate. On the other hand, some researchers have already made great progress on algorithm of solving No-Limit Texas Hold'em which is the game with hidden information. They described algorithm designed to handle information asymmetry between players by combining recursive reasoning and learning from self-playing games. These games and Hearthstone have many similar challenges. So it is very reasonable to use some of their thinking angles to solve the hearthstone.

3.1 Existing Methods

If we classify these algorithms roughly, we will get some descriptions like the following.

- *Random*

  The agent always chooses one action randomly from all possible actions without any logic.

- $Noaggression$

  The agent never actively attacks the opponent, it just randomly chooses between $PLAY\,CARD$ and $PASS$ which means end-turn.

- *Greedy*

  The agent always chooses the most valuable action, and does not consider too much. Its actions are driven by an algorithm built on several game metrics using an evolutionary approach.

- $GameStateValue$

  The agent makes an action selection based on the value of the current state. It is a recursive Alpha-Beta pruning algorithm controlled by a heuristic algorithm.

- $MonteCarloTreeSearch$

  Researchers have tried to apply the Monte Carlo tree search algorithm, which is very mature in the perfect information game, to Hearthstone, and made some adaptive improvements recently [4]. It makes use of a real player-based deck database to make Monte Carlo tree search possible on imperfect information situations.

Besides those game playing methods, researchers are also trying to solve it in several different aspects. Such as deck building [5], card generation and data mining [7].

3.2 Simulator

Because Hearthstone is actually a closed-source commercial game, it cannot be used as an experimental platform for research. To implement these algorithms and construct these agents, researchers and developers have developed and provided multiple simulators. Access to the game simulator allows game agents to perform dynamic reasoning about the game. The simulator calculates the legal movements in each state of the game, chooses to update the state after the movement, tests whether the game reaches the final state, and calculates the score in the completed game. States and actions are comparable and hashable. Of course, these reactions and feedbacks of the simulator are to imitate the behavior of the Hearthstone client and are strictly controlled within the rules of Hearthstone. There are already several mature simulators in the Hearthstone community $Hearthsim$. Some are open source simulator, and some are not. We would like to take advantage of these open source simulators which implements the main game mechanism instead of those which are calling the Hearthstone client server for a real battle to complete the play. Also there are simulators already include several artificial intelligence that allows players to play against, or run simulations, and provide statistics after the game is over.

- $Metastone$ is an open source simulator with several AI implemented. The biggest characteristic between it and other simulators is that it includes a sandbox mode that can be used to make your own cards and play them under formal game model.
- $Fireplace$ is a Python 3 simulator. It utilizes Hearthstone CardDefs XML files to store most game cards and has a default implementation for all simple minions. It also has an extremely extensive test suite and is used to create the Kettle protocol for playing simulation games online and on official clients.
- $Sabberstone$ is an simulator written in C# Net Core. At present, it has implemented more than 95 percentage of standard cards. Sabberstone was developed to provide Hearthstone addicts and AI developers with the possibility to implement AI, and a game tree search example has been implemented. On the other hand, state can be cloned in any game node. Sabberstone uses a simple declarative task system to implement the card mechanism. Sabberstone comes with a built-in Kettle server that can communicate with a real Hearthstone client via Stove or play games using the Joust web interface.
- $HearthSim$ is a generic Hearthstone game simulator with AI implemented. It is designed to run large numbers of AI vs AI games in order to test and understand the values of various game mechanics and cards. This simulator is focused on two components: the game mechanics and the AI.
- $Hearthbreaker$ is an open source Hearthstone simulator for the purposes of machine learning and data mining of Blizzard's Hearthstone: Heroes of WarCraft. Most cards in the game have been implemented. Hearthbreaker is not like platform where players can play Hearthstone against each other but is designed to be used as a library for analysis. The results of simulations can be used to analysis the relevance of different cards and can also be used to compose training data.

Besides these simulators we can take advantege of, there are also a variety of powerful tools developed by community members to help us with various statistics, process tracking and game behavior definition. $HSReplay$ is a replay specification, including reference implementations. $HearthstoneDeckTracker$ is a deck replay tracker and collection manager for Hearthstone. And $HearthstoneJSON$ exports all the card data in Hearthstone to JSON to work with more easily than the raw files.

Chapter 4

Proposed Approach and Experiment

4.1 Experiment of existing methods

In order to totally understand how simulator works and the performance of algorithms we listed before. We consider implementing and running these algorithms on a simulator as a previous experiment, especially the Monte Carlo tree search algorithm, which is very mature in the perfect information game, to Hearthstone, and made some adaptive improvements.[4] It makes use of a real player-based deck database to make Monte Carlo tree search possible on imperfect information trees. It is a very groundbreaking approach. We tested these techniques on different decks.

4.1.1 Experiment

In order to show the characteristics of different algorithms on different decks, we first tried several algorithms against the random opponents, and then used these algorithms to compete against each other for several decks belonging to different arthetype.

4.1.2 Versus Random

First let several algorithms compete against opponents using random choice on a warrior deck called $PirateWarriorDeck$, and get the results in Table 2.1.

4.1.3 Versus Each Other

After that we tested these algorithms on the $PirateWarriorDeck$, $MidrangeShamanDeck$, $FreezeMageDeck$, $MiracleRogueDeck$, where $PirateWarriorDeck$ belongs to $Aggro$, $MidrangeShamanDeck$ belongs to $Mid{-}range$, and $FreezeMageDeck$ and $MiracleRogueDeck$ belongs to $Control$. The $MiracleRogueDeck$

Algorithm	Win rate	Turns taken per game
Greedy	96.83%	7.2154
MCTS	98.47%	8.2083
GSV	99.94%	5.7709

Table 4.1: Win Rate Against Random

is an opportunistic deck that needs to generate high explosive power under certain conditions to end the game. It can be regarded as a special type of *Control*.Some results are shown in Table 2.2-2.5.

Algorithm		Lose rate		
		Greedy	MCTS	GSV
Win rate	Greedy		72.60%	4.97%
	MCTS	27.00%		0.23%
	GSV	95.03%	99.77%	

Table 4.2: Aggro deck: WARRIOR

Algorithm		Lose rate		
		Greedy	MCTS	GSV
Win rate	Greedy		65.30%	14.08%
	MCTS	34.7%		9.50%
	GSV	85.91%	90.50%	

Table 4.3: Mid-range deck: SHAMAN

The final target of solving Hearthstone is too complicated to ensure convergence of algorithms of CFR family. So we follow the previous research route which is used to solve Texas hold'em. That is we simplify the original large game into a small model and increases its complexity step by step. In this chapter, we firstly conduct our experiments on several simplest enviroment which created by a hard coded engine written by us manually for this research. After that, as we mentioned, we would make enviroment complexer stey by step, but the hard coded engine could no longer meet our needs. We tried and selected the most suitable simulator in the aforementioned simulator to conduct subsequent experiments, but even the most suitable simulator still could not fully meet our needs, so we rewrite the game engine of the simulator. Then created a more complex environment to train the agent.

4.2 Apply Counterfactual Regret Minimization on simple environment

In order to verify the convergence of the algorithm, real games have to be extremely simple through some simplified means. The goal of these simplification is to reduce the number of information sets

Algorithm		Lose rate		
		Greedy	MCTS	GSV
Win rate	Greedy		82.00%	54.00%
	MCTS	18.00%		19.00%
	GSV	46.00%	81.00%	

Table 4.4: Control deck: MAGE

Algorithm		Lose rate		
		Greedy	MCTS	GSV
Win rate	Greedy		94.60%	22.00%
	MCTS	5.40%		0.50%
	GSV	78.00%	99.50%	

Table 4.5: Control deck: ROGUE

for each player so that the small model can be solved. These simplifications can be to set the game mode very simple, for example, without any special game mechanics, or set the card very simple, such as without any special effects, or the hero's health is setting to very low, so it is helpful for the game to end as soon as possible to reduce the depth of the game tree.

4.2.1 Model environment

This game model does not have any game mechanics, only simple turn-based function. In other words, the two sides exchange for a round, and in their own round, they can make any legal action. We first program three cards for this simplified model. They are:

- *Saber* ATTAK 1, HEALTH 1, COST 1
- *Lancer* ATTAK 1, HEALTH 2, COST 1
- *Rider* ATTAK 2, HEALTH 1, COST 1

The original intention of these three creations is to first create the simplest card with 1 attack power, 1 health, and 1 mana cost. Then create another card that is slightly different from this basic card, but simple enough, so the Lancer and Rider are born. They don't have any special effects. They can only be placed on the battlefield as a minion, and then act on the player's next turn. They can attack or do nothing. They are shown in Fig 4.2.

Firstly we create one-card game models in which each player has only one card, which is one minion in the deck and player can play this card after drawing it. When there is only one card, this game is extremely like an perfect information game, because the opponent's actions are so few that the agent can fully observe the information on the board. However, in this mode, because players can choose to do nothing in a single round in CCGs, the agent's observation of the game situation is also limited until the opponent plays the only card. Therefore, this is still a game of imperfect information.

Besides this simplest model, we also conduct experiments on two-card game models in which each player has two minion cards. One is in hand at the beginning of the battle and draw another one from

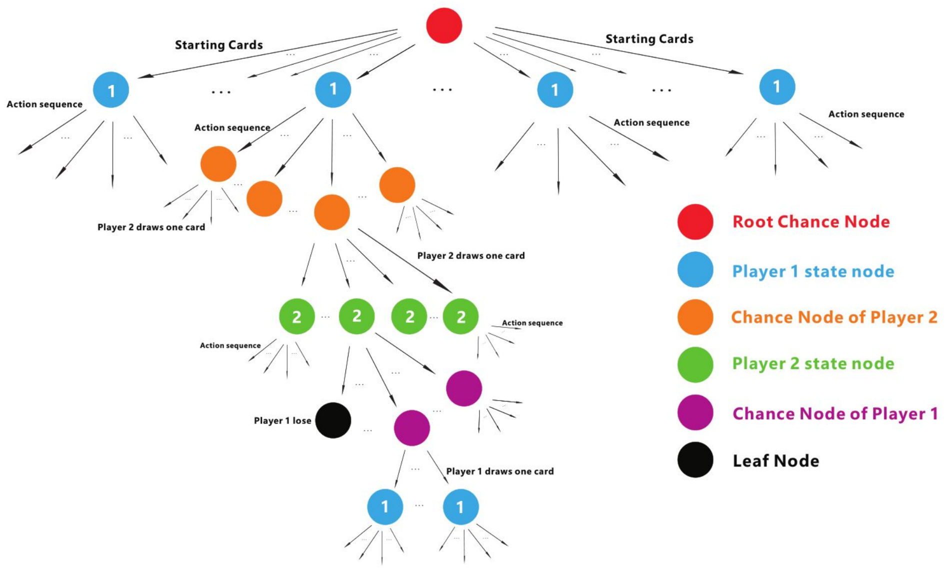


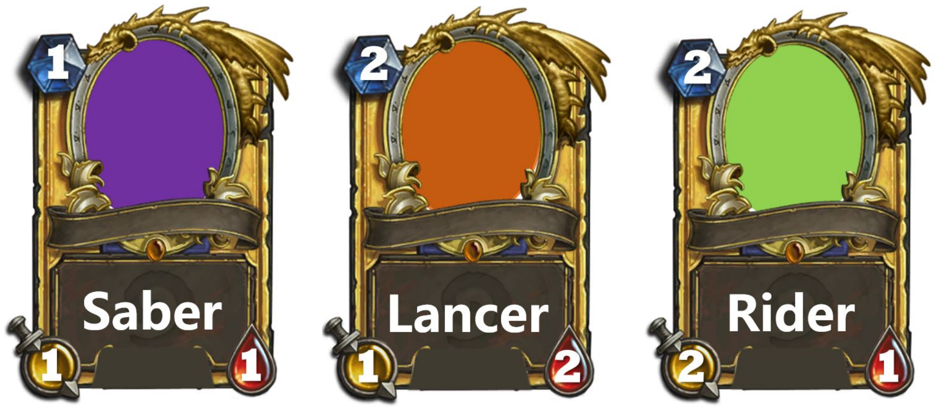
Figure 4.1: Game Tree

the deck later. When the player has two cards, this is obviously a game with imperfect information. Because the agent can't know what kind of card the opponent has at the beginning of the game, and which card the opponent drawn after the second round. Such an environment is more capable of verifying the applicability of the CFR algorithm in the case of imperfect information such as CCGs.

In one-card game and two-card game, we tested whether both players have the same card or different cards. After that, we adjust these models to an unbalance situation in which players may have different numbers of cards or Hero have different health points. In our simple model, players only have minions but no spells or weapons. So they can only use the minions to attack enemy heroes from the second round after placing the minions. We also remove hero power and effect of minion cards. There is another problem which also the significant difference between poker and CCGs. It is judgment on winning and losing. In normal poker games, after the players play cards according to certain rules, the players win or lose within a certain number of rounds. But there is an infinite round of games in Hearthstone when players choose 'pass' all the time. To prevent this unexpected situation, we limit the maximum number of rounds of game. The game tree is shown in Fig 4.1.

4.2.2 Build information set tree

In poker games, researchers usually use the information that can be observed on current situation as an information set. That's because in a poker game, one player per round can only make one action, and the agent only needs to observe this action then understand the state transition. Thereby, an infoset tree is formed via observing information and private information. As Hearthstone has more than one actions in one round such as 'attack', 'play cards', 'cast spells' or 'pass' and has a few types



(a) Saber (b) Lancer (c) Rider

Figure 4.2: Perfect and imperfect information game

of cards such as summon 'minions', 'spells' and 'weapons', we consider action sequences as variable that determines state transition in the game tree instead of actions in normal poker games. Summon minions could be considered as servants. Spells affect the battlefield which are discarded after being played. Weapons which give heroes strength to attack. Creating a node for every action would cause the game tree to be too deep. The information set for each indistinguishable group of nodes not only contains the action sequences token before, but also includes hand information which means what cards are in hand.

4.2.3 Vanilla CFR and Chance Sampling CFR

In order to verify the applicability of the algorithm, we chose the most basic CFR algorithm, vanilla CFR. In order to have a reference or comparison, we also choose Chance sampling CFR as another algorithm. We apply the two different CFR algorithms from the CFR family to every model. The vanilla algorithm would traverse the whole game tree while the chance sampling algorithm just samples one action sequence from the available action sequences. We plot two results of the same model in one table in different colors.

4.2.4 Result

In order to let the agent learn to avoid draw in limited rounds which cause Nash equilibrium to some extent, we make the agent regret reaching a draw and become much more positive when it reduces opponents hero's health point below zero. We conduct experiments on several one-card games which have different cards. Part of game tree of one model is shown in Fig 4.3. Agents learned probability distributions for every action available and we show the final probabilities which also are strategies in the tree. As we mentioned before, in order to obtain the strategies, we compute counterfactual value

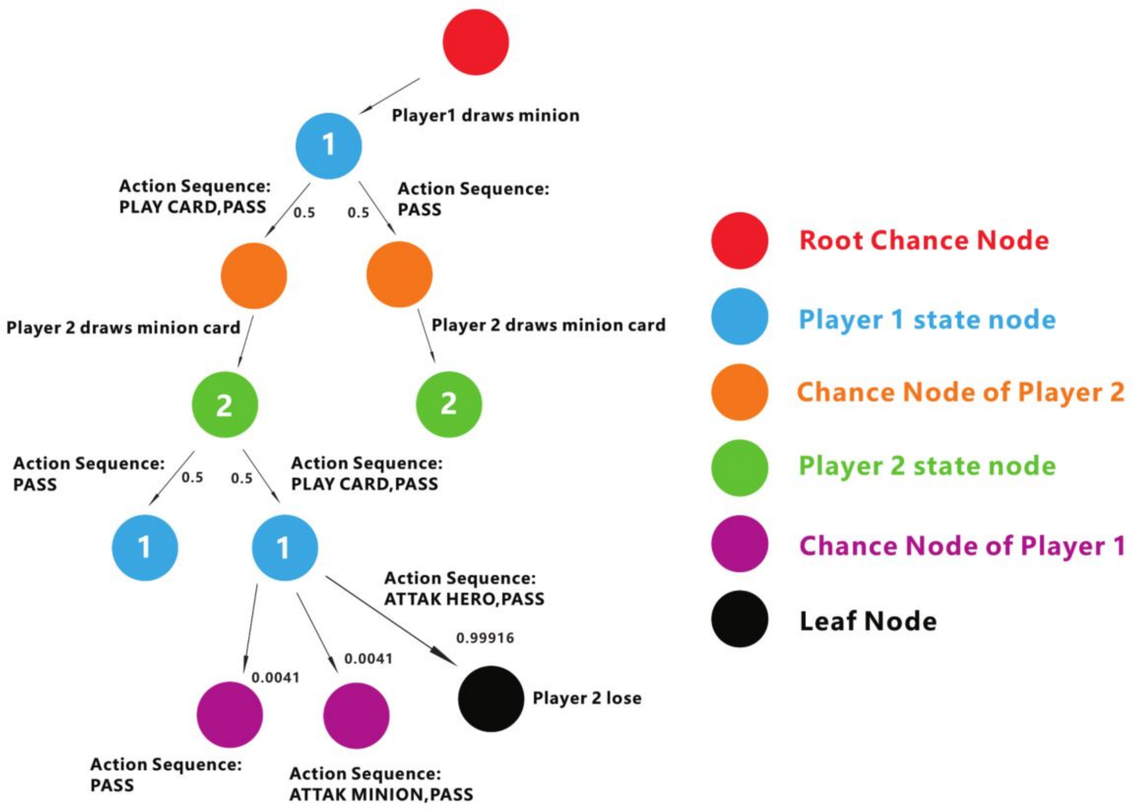


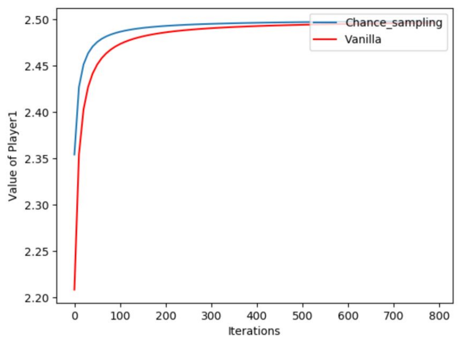
Figure 4.3: Game Tree of One-card Game

at first. So there is a value for every node and in order to verify the convergence and observe it, we calculate the value of Player 1 at the root node and plot these values over iterations.

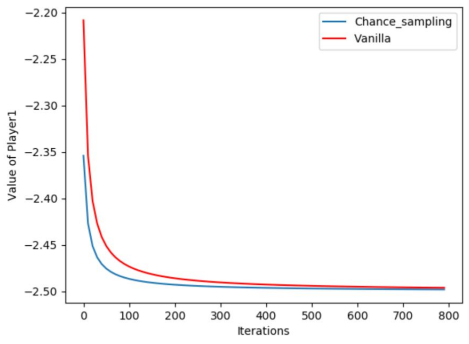
The results of two one-card game models are shown in Fig 4.4. In Fig 4.4(a), two players have the same card so Player 1 wins in most branches. In Fig 4.4(b), Player 1 has a minion card that costs two crystals and Player 2's card costs one crystal. That makes Player 1 lose in most branches.

Another two results of two-card game models are shown in Fig 4.5. Both players have two cards that cost one and two crystals for Fig 4.5(a). Player 1 has two cards that cost two and three crystals while Player 2 has two cards that cost one and two crystals for Fig 4.5(b). In Fig 4.5(c), Player 1 has two cards that cost one and three crystals while Player 2 has two cards that cost two and three crystals.

Fig 4.6 shows the results of two players in the unbalanced situation in which players has different number of cards or different health points. In Fig 4.6(a), Player 1 has two cards that cost one and two crystals while Player 2 has one card that costs one crystal. In Fig 4.6(b), Player 1 has three cards that cost one, two and three crystals respectively while Player 2 has two cards that cost one and two



(a) Players have same card



(b) Players have different card

Figure 4.4: One-card game

crystals. In Fig 4.6(c), two players have same card that costs one crystal but Player 1 only has one health point and Player 2 has two. In Fig 4.6(d), Player 1 has three cards that cost one, two and three crystals while Player 2 has two cards that cost one and two crystals, and Player 1 has one health point but Player 2 has two health points.

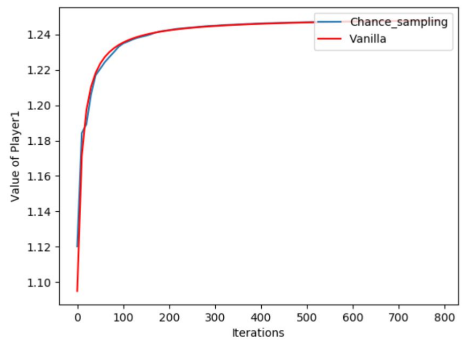
4.3 Experiment on simulator environment

After fully verifying the convergence of the CFR algorithm in CCGs games, we proceeded to further experiments. For the next stage of experiment, we first consider expanding the game engine we created to introduce more game features into it to meet the needs of subsequent experiments. But because when we first created this engine, we were hard coded, so if we want to add some game mechanics or card effects, we need to modify numerous code to coordinate the variable exchange between the various subjects. And it will also cause the calculation of the model to be particularly large. Finally, we decide to choose the simulator that is most suitable for implementing this algorithm among the existing simulators.

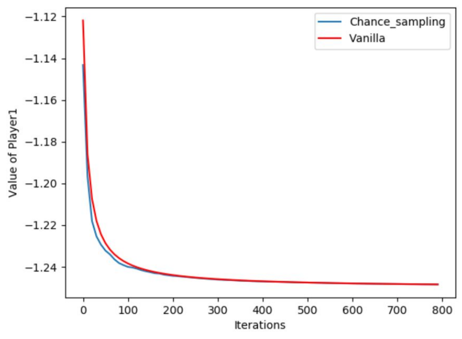
4.3.1 Modified Simulator

As mentioned earlier, in the relevant community of Hearthstone, there are multiple mature simulators. These simulators can almost completely imitate the functions of real games, and have assembled huge card pools, and some have even added simple game AI. These simulators do exactly what we want, a complete set of game feedback logic. But the first problem arises, these simulators are not all developed by the same developer, so they do not have a unified format and interface. That is to say, the model trained in one simulator cannot be used normally in another simulator, because the feedback information of each step of the simulator is different.

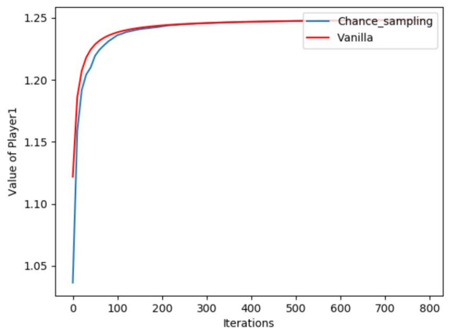
Another thorny issue is that most simulators do not have the ability to adjust the game model,



(a) Players have same cards



(b) Players have different cards



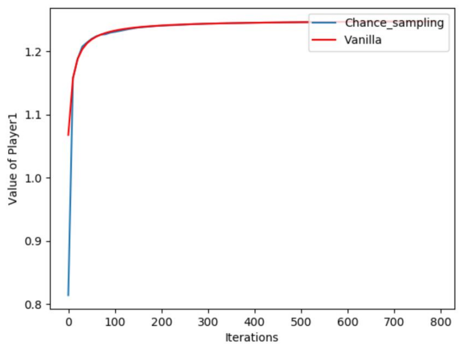
(c) Players have different cards

Figure 4.5: Two-card game

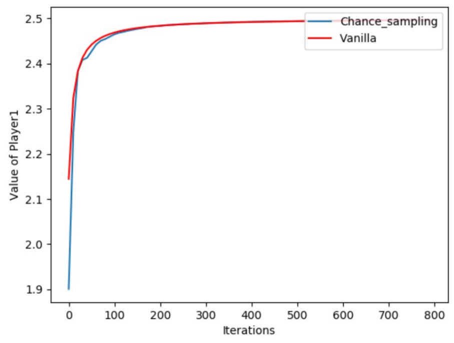
they are designed to execute a complete real game. In other words, each start requires 30 cards, and all game mechanics are preset. This makes simulator games as complicated as real games.

The third issue is even more difficult. Following the description of the second point, because it is completely imitating a real game, most simulators do not implement the function of returning to the previous state. That is the undo function often mentioned in the exploration of machine learning. These simulators act like servers to the end of a game and never look back. The advantage is that you can reach the final game and record the entire game. Such a piece of game data is provided as a replay to a model such as a neural network for training. But this is devastating for us, because the CFR algorithm needs to calculate the regrets of the nodes during the exploration and return them to the parent node, which makes us have the necessary requirements for the undo function.

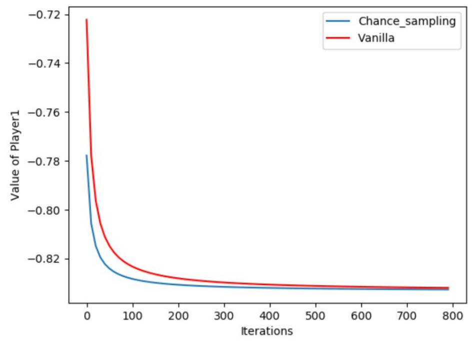
Combining the above problems, after comparison and practical operation, we chose a python-based simulator that can be called as a package, Heartbreaker. A screen-shot of GUI of Hearthbreaker is shown in Fig 4.7. The biggest advantage of this simulator is that each function of each object of it can be called independently. It is really like a package, which allows users to freely build it. This is very



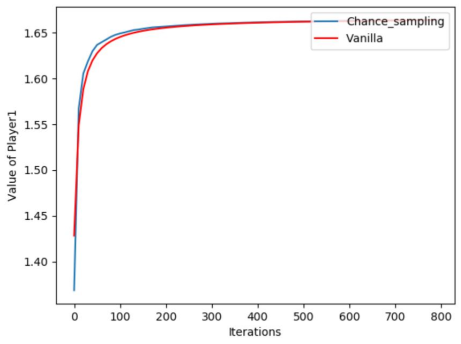
(a) Players have different number of cards



(b) Players have different number of cards



(c) Players have different health points



(d) Players have different number of cards and health points

Figure 4.6: Unbalanced game

convenient. Although Hearthbreaker is already the most suitable simulator, it still has some problems. So we rewrite the engine of the simulator without using many methods in the engine. To make it easier for us to build a tree of information sets, we rewrite how the simulator works. For example, in real game and simulator giving hand or draw phase, the action of drawing is random. And we want to be able to traverse every possibility in order to calculate what action should be taken in each case. So we rewritten the entire pre-game phase.

4.3.2 Model environment

Compared to the environment created to verify the convergence of the algorithm, we hope to calculate the strategy in a more complex action space. So we added a lot of game mechanics and card effects to the model. We have added fatigue mechanism, hero skills, spell cards and weapon cards. We also uses minions containing taunt, battlecry, and aura. Fig shows some of the cards we use with these effects.

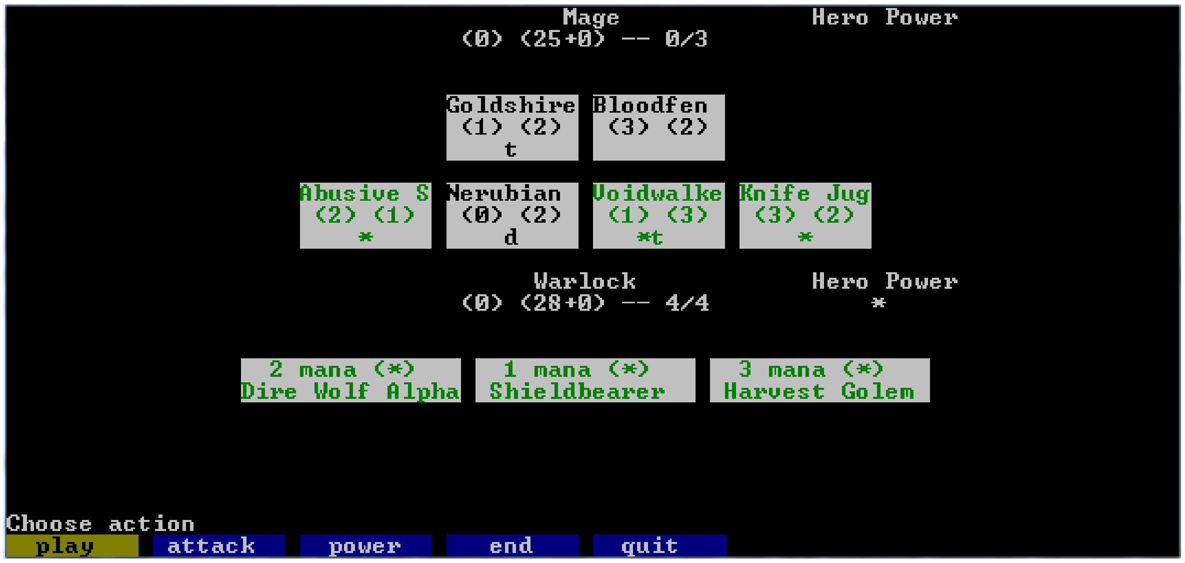


Figure 4.7: GUI of Hearthbreaker

In the process of adjusting the game model, the following factors have a great impact on training time and hardware usage:

- $NumberofCards$

  The number of cards determines how many possible starting hands the agent can have. It also limits the size of the legal action space. This determines the width of the information set tree.

- $HeroHealthPoints$

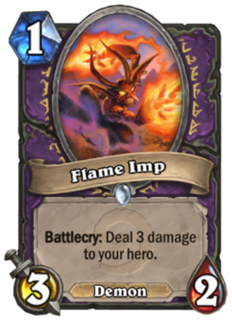
  The hero health points are sign of whether the game is over. The game ends only when one player defeat the hero of another player. In other words, the opponent's health points of hero needs to be reduced to 0 or below. It determines the depth of the tree of the information set.

- $RoundLimitation$

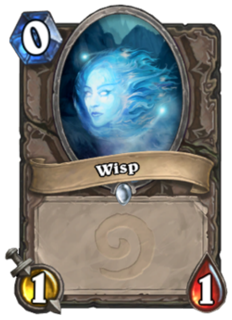
  Just like the simple model, in CCGs, players can choose to do nothing in this round and directly end the round. If both sides "tacitly" choose to end the round during the whole game, there will be numerous "do nothing" nodes before accumulating fatigue to kill the hero, which is a waste of memory and computing time. To some extent, it also determines the depth of the tree of the information set.

So we set the environment based on these three parameters that have great influence on computation. Several different experiments were performed. Through the idea of controlling variables, we set up four cards with different rounds and different hero health experiments, and the same rounds, different cards, and different hero health experiments, and finally the same hero health and different rounds

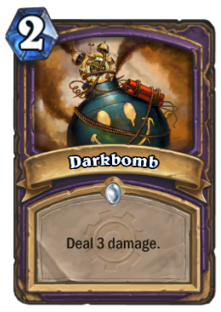
2
Armorsmith
Whenever a friendly minion takes damage, gain 1 Armor.
1
4
(a) Armorsmith



(b) Flameimp



(c) Wisp



(d) Darkbomb

Figure 4.8: Cards

And experiments with different cards. Example of Heroes and cards are shown in Fig 4.8 and Fig 4.9.

4.3.3 Build information set tree

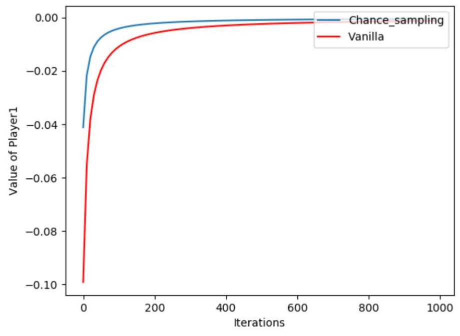
These settings make the game model more complicated, and the state of the game jump more frequently. When it comes to the change of game state, one thing that has to be explained is that in ordinary poker games, any player making an action will cause the game state to change. But in this complex model, changes in the state of the game can be caused by multiple reasons. For example, if a player has no cards to draw in the deck, he will take a cumulative damage, which is the fatigue mechanism we just mentioned. In other words, the state of the game changes between the end of the previous round of player actions and the start of the next round of player actions. Another example is that some cards have a mechanism such as "trigger". The chance of its effect depends on the progress of a certain game, and it will also change the game state at the same time. In this complicated case, in order not to construct a new information set with each state transition, we further optimized the information of every set. In the tree of the information set, as a branch connecting two nodes, the information set integrates a large number of game nodes by recording the action sequences of the two players until reaching the current node and the information of current game situation. This makes the game model computable even in the case of vanilla CFR.

4.3.4 Result

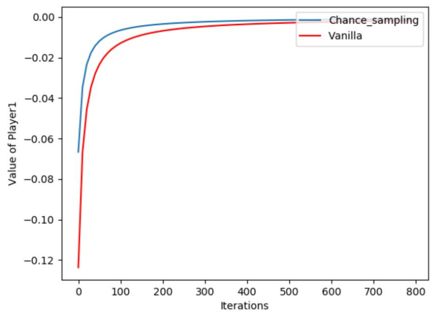
As in the previous experiments, by observing the value of Player 1 on the root node of the tree of the entire information set, we can know whether the agent falls on a Nash equilibrium point after numerous iterations. If the game does not have a Nash equilibrium, or there are multiple Nash equilibriums, then the value of Player 1 at the root node will be very unstable, or even this value will not be obtained. After calculate the value of Player 1 at the root node and plot these values over iterations, we could observe that agent always find a Nash equilibrium in different game settings.

Figure 4.9: hero

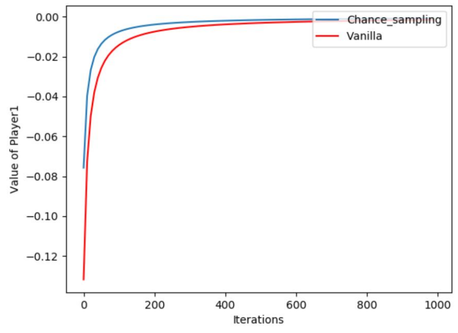
So far we have carried out experiments with three cards, four cards, five cards with different hero health, different rounds of constraints, these experimental results are shown in Fig. In an acceptable computing time, although the simple CFR algorithm occupies a lot of computing resources, it successfully trains the agent that can find the equilibrium strategy.



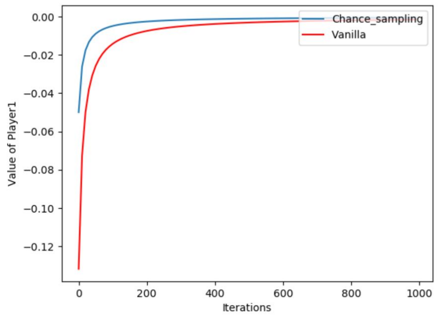
(a) Warlock with 3 minions 1 spell and Warrior with 4 minions



(b) Warlock with 4 minions and Warrior with 4 minions

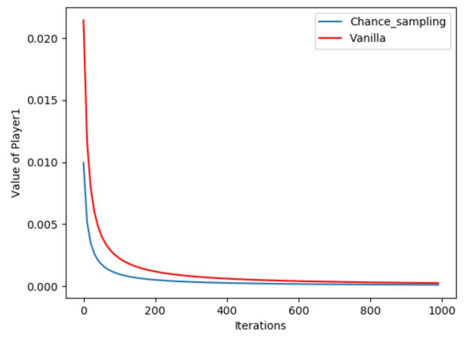


(c) Both Warlock with 3 minions 1 spell

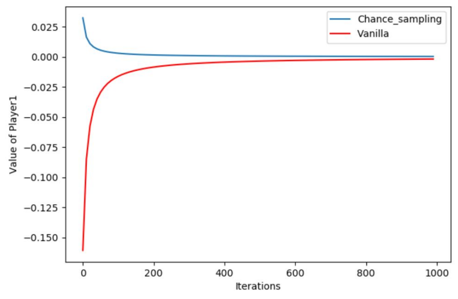


(d) Both Warlock with 4 minions 1 spell

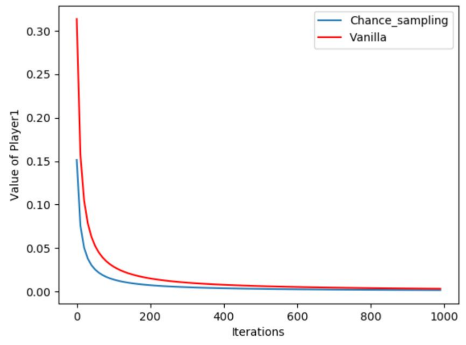
Figure 4.10: Same HP with different cards



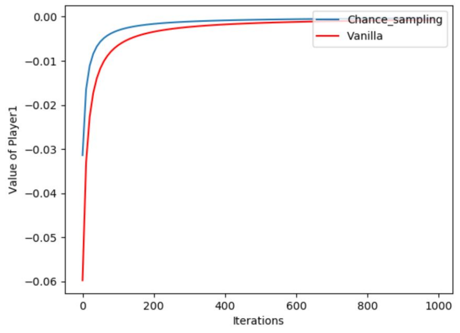
(a) Warlock with 3 minions 1 spell and Warrior with 3 minions 1 spell



(b) Warlock with 3 minions 1 spells and Warrior with 3 minions 1 spell



(c) Warlock with 3 minions 1 spells and Warrior with 3 minions 1 spell



(d) Warlock with 2 minions 1 spells and Warrior with 2 minions 1 spell

Figure 4.11: Same cards with different HP

Chapter 5

Conlusion

5.1 Conlusion

In this work, we propose a novel approach to solve Collectible Card Games (CCGs) from a traditional imperfect information perspective. In order to solve the complicated card games like Hearthstone, we create a game enging with simple game mechanics, and creating several battle environments based on Hearthstone rules and apply Counterfactual Regret Minimization (CFR) algorithms on those models. In each model, we calculate the value of player that makes the action first. Although the model tested is small, we confirm the convergence of CFR algorithms.

After that, we explored the feasibility of the algorithm on more complex models. We first analyzed source code of several simulators. After selecting the most appropriate simulator, in order to obtain a game model that is more convenient to implement the algorithm, we modified the game engine of the simulator, and successfully used the mature game engine to introduce more complex game mechanics and card special effects. Based on this, we use the proven algorithm to calculate the equilibrium strategy for complex game models. Finally, under the condition of acceptable computing time and computing resource consumption, successfully trained an agent that can find a balanced strategy in a game with a complex game mode such as Collectible Card Games.

5.2 Future work

Although in the whole work, we verified the convergence of Counterfactual Regret Minimization in Collectible Card Games, and explored the equilibrium strategy under the small model. But there is still a long way to go to solve a game of actual size. The number of cards and game mechanics of real games will become larger and more complicated, which makes the traditional machine learning method limited by the amount of calculation. Fortunately, in the continuous exploration of poker games in recent years, researchers have successfully solved No-Limited Texas Hold ’ em. If we want to continue to try to solve the intelligent agent problem under the larger game model, we need to learn and refer to the method of solving poker games. Reducing the amount of calculation and calculation time is a very important task. Secondly, trying other machine learning algorithms in Collectible Card Games is also very worth looking forward to. Research on supervised learning using replay has already been done. Trying reinforcement learning in situations where full observation is not possible may provide some good ideas.

Bibliography

[1] Mastering the game of Go with deep neural networks and tree search, David Silver and Huang, 2016.

[2] *GameTheory*, Drew Fudenberg and Jean Tirole, 1991.

[3] Monte Carlo Search Applied to Card Selection In Magic: The Gathering, C.D. Ward and Peter Cowling, 2009.

[4] Monte Carlo Tree Search Experiments In Hearthstone, A. Santos, P. A. Santos, and F. S. Melo, 2017.

[5] Predicting winrate of Hearthstone decks using their archetypes, Jan Betley, Anna Sztyber, and Adam Witkowski, 2018.

[6] Investigating Similarity between Hearthstone Cards: Text Embeddings and Interchangeability Approaches, Andrzej Janusz and Dominik Slezak, 2018.

[7] Helping AI to Play Hearthstone: AAIA'17 Data Mining Challenge, Andrzej Janusz, Tomasz Tajmajer and Maciej Swiechowski, 2017.

[8] Regret Minimization In Games With Incomplete Information, Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione, 2008.

[9] Monte Carlo Sampling for Regret Minimization in Extensive Games, Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling, 2009.

[10] Solving Large Imperfect Information Games Using CFR+, Oskari Tammelin, 2014.

[11] Solving Imperfect Information Games via Discounted Regret Minimization, Noam Brown and Tuomas Sandholm, 2018.

[12] Superhuman AI for multiplayer poker, Noam Brown and Tuomas Sandholm, 2019.

[13] Deep Counterfactual Regret Minimization, Noam Brown, Adam Lerer, Sam Grossand and Tuomas Sandholm, 2019.

[14] Improving Hearthstone AI by Learning High-Level Rollout Policies and Bucketing Chance Node Events, Shuyi Zhang, Michael Buro, 2018.

[15] Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms, Maciej ´ Swiechowski, Tomasz Tajmajer, Andrzej Janusz, 2018

[16] A Neural Network Approach to Hearthstone Win Rate Prediction, Jan Jakubik, 2018

[17] Monte Carlo Tree Search Experiments in Hearthstone, André Santos, Pedro A.Santos, Francisco S.Melo, 2017

[18] Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions, Richard Gibson, 2012

[19] Superhuman AI for head-up no-limit poker Libratus, Noam Brown, Tuomas Sandholm, 2018

[20] DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker, Matej Moravcık, Martin Schmid, Neil Burch, 2017

Publications

Publications related to this work are as follows:

- Haoyu Zhang and Yoshimasa Tsuruoka. Towards Finding a Solution for Collectible Card Games with Counterfactual Regret Minimization. ゲームプログラミングワークショップ 2019 論文集, pp. 235 - 241, 2019.