Politecnico di Torino

Master degree course in Ingegneria Informatica

POLITECNICO DI TORINO

Master Degree Thesis

Computational Intelligence Techniques for Games with Incomplete Information

Supervisors
Giovanni Squillero
Alberto Paolo Tonda

Candidates
Stefano Griva
matricola: 287729

Anno accademico 2022-2023

Summary

We developed a new artificial intelligence that takes into consideration hidden and external information when playing games. Our aim is to show how AIs can improve their performances by using hidden information, that would normally require complex human deduction to normally exploit.

Modern game AIs often rely on clear and curated data, deterministic information and overall accurate numbers to make their calculations, however there are a lot of games that involve pieces of information that are incomplete or hidden. Incomplete information can be extremely helpful to an AI, but it requires additional care when taken into consideration because it's usually based on statistical analysis and heuristics. Our focus is set on a few innovative computational intelligence techniques that aim at improving the efficiency of hidden information-based AIs, by allowing them to explore non-deterministic scenarios:

1. The Double Inverted Index is an algorithm that can be used in hidden information games, such as card games, to narrow the great possibilities and scenarios to calculate down to a reasonable number. This approach is based on how humans would think in similar situation.

2. The Blunder Threshold is a technique that helps the AI navigating probabilistic scenarios balancing the pros and cons of deeper analysis and uncertain information.

We'll explore different parameters and options for the previously mentioned techniques and show their efficacy in practice with a focus on the chosen test game Hearthstone.

Acknowledgements

I would like to express my sincere gratitude to professor Squillero, who provided me the opportunity to work on such a unique topic and shared with me invaluable knowledge and resources that guided me trough out the whole process.
I wish to extend my special thanks to professor Tonda for providing the starting agents, assisting me with the brainstorming and problem solving process as well as for providing constant and expeditious feedback on my work, without which I would have never reached such results.

I would like to thank my girlfriend Linh her support during this long endeavour and for always being there to cheer me when I needed it. I would like to thank my parents and my sister for their unending support, praising me for my successes and helping me up when I stumbled. Lastly I would like to thank all my friends and colleagues, with whom I spent my whole academic career, for making this wonderful time fly by in a breeze.

Thank you all for making my whole experience unique and unforgettable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial intelligence is an ever growing field in computer science, with new techniques and algorithms getting developed every day. With such an influx of innovation it's important to consider all the possibility available when trying to push the boundaries of what AIs can perform.
Our aim is to show how AIs can improve their performances by exploiting hidden information in a way that would normally require complex human deduction.
Additionally we want to explore how these hidden information can be exploited, focusing on the trade off between effectiveness and computational complexity, ultimately looking for the best improvement at the lowest cost possible.

1.1 Artificial intelligence and games

In the field of artificial intelligence observing and understating the behaviour displayed by the machine can be daunting and complicated. In order to simplify the problem researchers have in the past adopted simpler systems that are easier to represent and understand; included in this group there are games. Games have many peculiarities that make them an ideal starting step from which one can construct and analyze an AI:

- Games have a simple rule set that can be easily represented in a data structure, making it understandable by the computer.

- Games have a clear and easily identifiable objective, making it simple to evaluate the result gathered by an AI.

- Games are a traditional hallmark for intelligence, meaning they historically are an interesting target for study.
- Games can be a good model for real world competitive or cooperative activities, which makes the results applicable to other fields.

Chapter 2

Background

2.1 Game categories

Games can be categorized in many classes depending on their rules. The main features to be aware of are:

1. Deterministic vs probabilistic.
2. Perfect information vs imperfect information.
3. Turn-based vs real-time.
4. Zero sum vs non-zero sum.
5. Competitive vs cooperative.
6. Single player vs multiplayer.

The first feature describes whether a game involves chance or is completely deterministic from start to finish.
The second feature is related to the information included in the game: in perfect information games all the player have access to the same pieces of information, while the term imperfect, or hidden, information is used for games where the information isn't completely available to all the players.
The third feature differentiates games based on the timing the players are allowed to play, whether they alternate in turns or the play freely.
The fourth feature is about the outcome of the game and whether the sum of all players' result adds to zero or not.
Competitive vs cooperative is a spectrum that describes the objective of the game and whether the players are put up against one other or if it's in their

best interest to cooperate.
The sixth and final aspect of a game is the number of player.

Here are a few examples of famous games described using their features:
Chess: deterministic perfect information turn-based zero sum competitive game with 2 players.

Figure 2.1. Chess board

Poker: probabilistic imperfect information turn based zero-sum competitive multiplayer.
Hanabi: probabilistic imperfect information turn-based non-zero sum cooperative multiplayer.

Figure 2.2. Board game Hanabi

Solitaire: probabilistic imperfect information turn-based zero sum single player.
Hide and seek: probabilistic imperfect information real time zero sum multiplayer.

2.2 Computational intelligence techniques

Despite the multitude of games and categories they fill, the most common type of games are competitive 2 players turn-based games. In theese kind of games AIs can use a particular kind of search called *Adversarial Search*. When deciding on a move to make in a game, adversarial search is the process of, not only consider all available options given to the player, but also extend the horizon of observation to the possible moves available to the opponent. Ideally this process can be repeated over and over evaluating the consequences of a single move many turns in the future.
Adversarial search can be a very powerful tool, however it works best in very

particular situations:

- Adversarial search gives the best results when applied to deterministic games, where the AI doesn't have to keep track of probabilistic results or base its calculations on chance.

- Adversarial search is limited to simple use cases with a limited and ideally small number of options. This is because the higher is the number of possibilities, the quicker the decision tree constructed using theese choices will grow, making the problem computationally unfeasible.

While the complexity of the problem is a limiting factor, there are advanced techniques that can help reducing its impact on the efficiency of the adversarial search: theese mainly boil down to pruning the decision tree by discarding obvious bad decisions or not exploring uninteresting branches using heuristics based on the game rules.

2.2.1 MinMax

The simplest example of an adversarial search algorithm is MinMax, first invented by John von Neumann. MinMax creates a complete decision tree aiming to minimizing the loss, looking for the best worst scenario. Here are the steps used by MinMax:

1. Create a decision tree starting from the starting position and exploring all possible moves from the current player.

2. Repeat the process from each new node of the tree until all nodes left are terminal node (end game positions in which no more moves can be played)

3. Evaluate each terminal position by giving it a positive score if the player who won was the starting player, or a negative score if the other player won.

4. Work your way backwards from the terminal nodes giving each node you find a score depending on who is the player who's about to make a move in the position described by the node: select the highest score achieved by the children of the node if the starting player is about to move, otherwise select the lowest score.

5. At the end of the process the root of the decision tree, that describes the current position, should have an evaluation that is represents the best possible result if both players play their respective best moves from now until the end of the game.

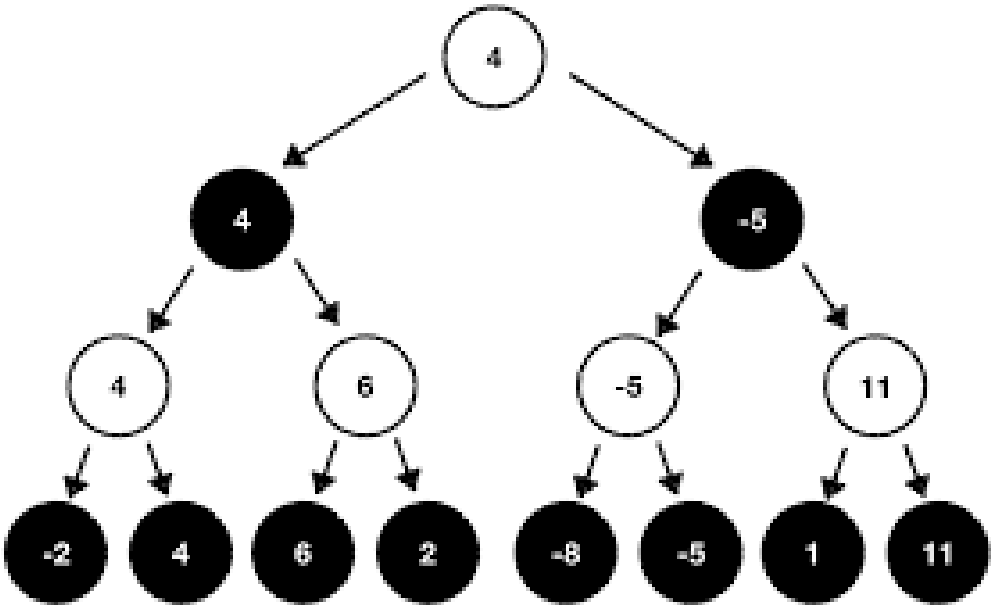

Figure 2.3. MinMax tree

2.2.2 Stochastic search and Montecarlo Tree Search

Sometimes it's impossible to use deterministic techniques to study and analyze all possible moves, be it because the number of possibilities and the depth of the game are too high to completely study them, or because the games is probabilistic and therefore involves chance. To help in theese scenarios, it's possible to use Stochastic search techniques, which can't guarantee to find the best absolute option, but can lead to very good results within time and computational constraints. An example of a stochastic search algoritm is the Montecarlo Tree Search(MTCS): The focus of MCTS is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space. The application of Monte Carlo tree search in games is based on many playouts. In each playout, the game is played out to the very end by selecting moves at random. The final game result of each playout is

then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.

The most basic way to use playouts is to apply the same number of playouts after each legal move of the current player, then choose the move which led to the most victories. The efficiency of this method often increases with time as more playouts are assigned to the moves that have frequently resulted in the current player's victory according to previous playouts. Each round of Monte Carlo tree search consists of four steps:

- Selection: Start from root R and select successive child nodes until a leaf node L is reached. The root is the current game state and a leaf is any node that has a potential child from which no simulation (playout) has yet been initiated. The section below says more about a way of biasing choice of child nodes that lets the game tree expand towards the most promising moves, which is the essence of Monte Carlo tree search.

- Expansion: Unless L ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node C from one of them. Child nodes are any valid moves from the game position defined by L.

- Simulation: Complete one random playout from node C. This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).

- Back-propagation: Use the result of the playout to update information in the nodes on the path from C to R.

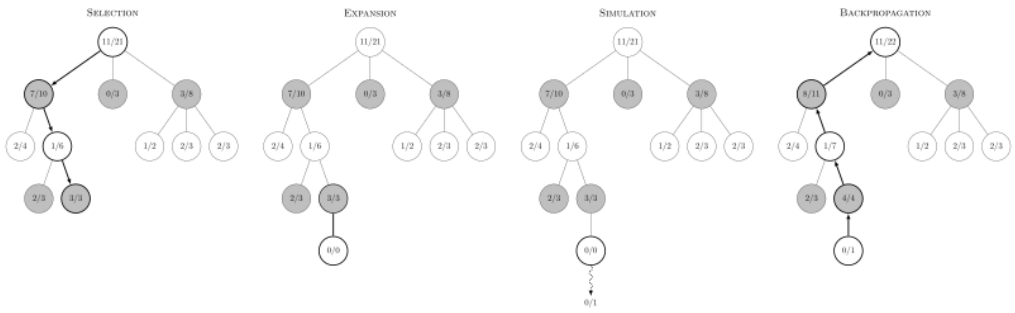

Figure 2.4. Montecarlo tree search tree

2.3 Obstacles of probabilistic and hidden information games

The space of AIs for probabilistic and hidden information games is large and not very well explored; this is because it's more complicated to code agents when working in non-deterministic environments.

2.3.1 Probabilistic games

The first and most obvious problem of probabilistic games is the complexity of the task and the time and computational resources required to tackle it. Most algorithms suited to deal with deterministic problems tend to analyze all possible options, in order to locate the best outcome reachable, in probabilistic games, however, every time an event is determined by chance (rolling a dice or like drawing a card from a deck) the algorithm has 2 options:

1. Cover all possible outcomes.

2. Simulate the event.

The first option requires the algorithm to take all possible outcomes into consideration and expand the decision tree using all the new states and children of the one previously under study. Depending on how often probabilistic events occur, the decision tree can grow extremely fast and large, making reaching a solution computationally unfeasible. The second option is a trade off between results and complexity. The algorithm simulates a random result of the probabilistic event and takes that as the deterministic outcome of the event, continuing the computation from there. This method is bias towards the simulated outcome, regardless of whether that's positive or negative for the AI; in order to correct this some algorithms simulate the event multiple times averaging the results. In this way the result obtained isn't perfect, but it can be very close to one and it can be obtained with limited resources.

2.3.2 Hidden information games

When developing AIs for hidden information games, some of the challenges are similar to probabilistic games: for example every time a player could make a move based on hidden information, that decision can look random from outside and therefor can be handled as such. A good example of a

hidden information game that can be tackled this way is poker: During a hand of poker no player is aware of the cards in other players hands, but this lack of information can be substituted with simulation. It's fairly trivial for an AI to calculate the range of possible hands that it can beat or the ones that would win against it, making it possible to stipulate a strategy based on probability calculation and decide if and how much to bet on any given hand. Stronger poker AIs also incorporate their opponents behaviour in the evaluation of the hand, based on optimal play patterns, to extrapolate even more information hidden to them. Sometimes however, it's impossible, or at least impractical, to represent hidden information as probabilistic events, be it because the chance of each outcome is too complicated to calculate or there are simply too many possibilities to keep track of. Probably the most famous genre of games that uses hidden information to the point of making it impossible to calculate all possible situation are collectible card games.

2.4 Hearthstone as a case study

Hearthstone is a digital collectible card game developed by Activision Blizzard in 2014. Compared to other similar games, Hearthstone is considered to be simpler thanks to a few key features, the main one being that the players alternate playing in turns, but are not allowed to play cards outside of their turns.
Because of this simpler nature and the focus on progressive strategical play rather than timely responses, as well as the availability of resources to simulate the game, Hearthstone has been the go-to game for many studies about AI in card games.

2.4.1 Rules

The aim of a game of Hearthstone is to reduce the opponent's life total to 0, starting from a total of 30. At the start of a Hearthstone the game each player has 3 or 4 cards in their hand, depending on whether they are the starting player or not, the the players alternate in turns. During each of their turn a player:

- Draws a card.
- Gains a mana crystal.
- Can play one or more cards.

- Can attack one or more minions.
- Can activate their hero power.

Mana crystals are used to play cards: each card has a cost and in order to play it the player has to exhaust as many crystals as the cost; theese crystals will be refilled at the start of the next turn, while at the same time the player will gain a new one. Each player starts their first turn with one mana crystal and can't have more than ten at the same time. In Hearthstone cards can be one of three type:

- Minion
- Spell
- Weapon

Minions have a mana cost, an attack and a defense. From the turn after the one when it was summoned a minion can be used to attack, when attacking a minion can attack a minion controlled by the opponent or attack the opponent life total directly. If a minion attacks the opponent, their life total is reduce by an amount equal to the minion attack, otherwise when attacking a minion both minion get their defense reduced by the other minion's attack, if a minion's defense ever goes at or below 0 the minion is destroyed. A player can control at most 7 minions at the same time.

Figure 2.5. An example of Hearthstone minions

Some times minions have effects and keyword that modify their behaviour. A few examples are:

- Battlecry: the minion has an effects that activates when played, this effect changes from minion to minion.

- Deathrattle: The minion has an effect when destroyed.
- Charge: the minion can attack even on the turn it was summoned.
- Taunt: minions controlled by the opponent can only attack minions with taunt as long as at least one of them is on the board.
- Lifesteal: when attacking this minion heals its controller for the same amount of damage it deals.

Spells have a cost and effect. Spells can have many different effects, ranging from drawing cards to dealing damage, from destroying minions to reviving dead ones.

Figure 2.6. Examples of Hearthstone spells

Weapons have a cost, an attack and a durability stat. When a player plays a weapon it gets equipped to the player, each player can hold a single weapon at the time, if a player is already holding a weapon when playing a new one the first one is destroyed. While holding a weapon a player can attack as if they were a minion, using the attack of the weapon as attack and their health as defense; for each attack performed this way the durability of the weapon gets reduced by one, if it ever reaches 0 or below, the weapon is destroyed.

Figure 2.7. An example of Hearthstone weapons

When playing a game of Hearthstone, each player plays as one of over 9 heroes and each hero has their own hero power. Hero powers cost 2 mana and can be activated once each turn. Hero powers usually have less impact than minions and spells, but are meant to be used often. Some effects of hero powers are:

Figure 2.8. Examples of Hearthstone hero powers

2.4.2 Strategies

Hearthstone is a complex game and as such it's possible to choose between many strategies, depending on the deck and the hero that the player decides to use. While strategy can be a very complicated topic, the main axis on which each deck is constructed is how aggressive and proactive its strategy

is versus how calculated and reactive. The more aggressive deck, also known as "aggro", tend to play cheaply costed minion with a lot of attack, but not that much defense, they aim at winning the game as quickly as possible by attacking the opponent and ignoring their minions before the opponent can catch up with more expensive and powerful cards. On the other hand of the spectrum there are "control" deck, that aim to slow down the pace of the game with defensive minions and spells, while the keep their health high with healing effects, if they're successful they then try and close the game with very expensive cards that are hard to contest. In between aggro and control it's possible to find many different decks that are generally grouped together and the name of "mid-range"; in this kind of decks the player is looking to gradually get in a more and more advantageous position by being flexible and adapt during the game. When deciding on what strategy to use, it's important to consider that some are more complicated to play, with the complexity being vaguely correlated with the length of the game that each strategy aims to reach (this is because in longer games it's easier to commit a mistake), as well as what strategy one is expecting to play against, this is because each strategy tends to perform better against some and worse against others: aggro is traditionally favored against control, since it can overpower them in the early turns and close the game out before the control deck can strike back; on the other hand mid-range performs better against aggro than it does against control, since it's better equipped to deal with aggression, but struggles to keep up with control in the later parts of the game.

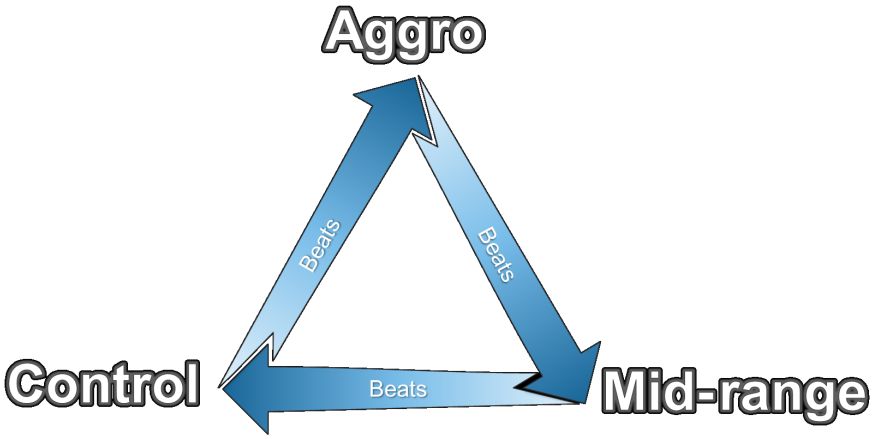

Figure 2.9. Strategic match-up chart

2.4.3 Sabberstone

With Hearthstone being a **digital** card game, it is easy to implement simulators that allow AIs to play at the game by interfacing with APIs. An additional feature of such simulators is that two AIs facing each other can play as fast as they can and don't have to be restrained by UI or other parts of the game that are essential for humans, but only slow down digital players; this is very important when testing the efficiency of an agent, as it's necessary to reach a high enough number of games meet statistical relevancy. Sabberstone is an open source Hearthstone simulator developed in C# and intended to be used as a research tool for artificial intelligence for Hearthstone.

Chapter 3

Proposed approach

When attempting to develop an AI for a game as complex as Hearthstone, it's necessary to implement creative solutions on top of structured and well documented algorithms. The first step is to start from an AI that ignores hidden information all together and build the new logic on top of it step by step.

3.1 Algorithms and agents

MCTS is perfectly suitable as a search algorithm for games like Hearthstone, but the granularity of the search, the depth of the tree and the evaluation of each move can be adjusted to obtain different results. During each turn the player is tasked to make multiple decisions, considering which move to make and the order they're made in. With this in mind, the player should consider all of their options at the start of their turn, when they've just drawn a new card and when the opponent is not allowed to play until their next turn. From here the agent should consider what its next move will be, with an eye to what the rest of his option would be after the best one is picked and how would these affect each other in the evaluation.

3.1.1 Montecarlo tree search

At the beginning of the turn the agent will perform a MCTS and analyze the position, considering all available options as the children of the root node, pick the best move it finds, considering the next followups as well as the immediate gain. Each move after the first will follow a similar paradigm, except they'll start from the newly found game state, allowing for a more

in-depth analysis of the branch taken. This means that instead of blindly following the hypothetical line of move that the first MCTS found when choosing the first move, each move will be studied in details an perhaps improved in quality.

3.1.2 Depth of the tree

The depth of the tree is also a key parameter to be considered. Due to time and computations constraints, the tree has to have a limited depth, however depending on the scope of the search, this limitation can be of very little impact. Logically the tree should stop when finding a terminal situation and the ideal position to label as terminal is when the current turn ends. This is the ideal situation, because, after ending the turn, the player looses control of the choices made from now until their next turn, making additional calculations much less valuable; coincidentally ending the turn is always an option in Hearthstone, because it's up to the player to declare the end of their turn by pressing the "end turn" button when done playing. In order to limit the tree's depth and therefore confine the analyzable space of possibilities by the algorithm, without impairing on it's ability to select the best options, one only needs to label all nodes where the agent selected "end turn" as its option.

3.1.3 Evaluation methods

Once the structure of the tree is decided, each node needs a value that estimates how good or bad a certain position is and to reach this value the AI needs an evaluation method. Evaluating a position in a game of Hearthstone is very complicated and deceptive, due to the high amount of hidden information, however it's possible to get a practical evaluation of the board state with some simple heuristics. An evaluation based on heuristics will be biased towards one or more aspects of the game, mainly because it's based on the comprehension of the analysis of the game by the programmer who codes the evaluator. The most straight forward method of evaluating a position using heuristics is to add up all the various factors that compose a game-state and weight them depending on their importance; for instance one could sum all of the minions attack and defense and subtract the one controlled by the opponent, add the current player's health, but divide this value by 2, to signal that it's not as important as minions' statistics, and so on. A lot of parameters can be factor in during this calculations:

- Minion defense.
- Minion attack.
- Number of minions.
- Health of players.
- Mana crystals available.
- Cards in hand of players.
- Abilities of minions in play.

Lastly all of theese can be added linearly or they can be standardised, depending on the desired result. The only way to reach the best result with this kind of approach is to test multiple sets of parameters and compare the result of both the evaluator and the relative MCTS.

3.1.4 Hidden information

The last step in developing a good AI for Hearthstone is to consider hidden information and to do this it's possible to take inspiration in how humans play the game. While hidden information is as hidden to humans as it is for an AI, human players can gather information and recognize patterns from game to game and use this knowledge to make educated guess about the hidden pieces of information. The knowledge players accumulate between games is called meta-game knowledge, from the Greek root meta- which means beyond. Meta-game knowledge can take many forms, depending on the skill of the player, from understating the opponent's strategy from the very first few turns and adjusting accordingly, to expect a specific card in the opponent's deck and playing around it by never leaving a good opening for that. This last case is an incredibly important skill for expert level play and it can drastically influence a game by limiting a player's options without expending too many resources. In order to acquire this skill, one has to understand the connections between cards and and figure out what hidden cards go with the one the opponent already played.

3.2 Inverted index

An inverted index is a database index used in computer science that stores a mapping from content, such as words or numbers, to their places in a table,

a document, or a series of documents. It is also known as a postings list, postings file, or inverted file. An inverted index's goal is to speed up full-text searches at the expense of more processing time whenever a new document is added to the database. It is the most widely used data format in document retrieval systems and is widely used in search engines.
An inverted index has a few qualities that make it desirable for a Hearthstone AI. first is its speed: an inverted index is generally quick but it's slow to update; luckily the construction and update of such an index happens only using meta-knowledge and therefore in between games, which means that the downside is negligible. The second reason an inverted index works perfectly for this kind of game is that it perfectly describes the kind of information the AI is looking for, which is the relations between the cards contained in a deck and the deck itself, intended as the collection of cards as well as the general strategy of said deck. In order to reach the best performance the AI should not only understand what deck it's facing, but also consider the cards that compose it, this way it can play around them and offer the lowest amount of opportunities to the opponent.

3.2.1 Double inverted index

Starting from a list of popular decks, constructed by either collecting data from games or observing the meta-game from outside, the first step to construct the ideal data structure is to create an inverted index that uses the cards as entries to look for possible decks in which it is played. At this point, during a game, the algorithm could look up every file that contains one of the possible decks that the opponent could be playing, restricting the list with every card played, however this process is slow by itself, as the IO of opening files can be slow. The solution to this process is to modify the previous inverted index by translating each deck as a list of the card it contains.

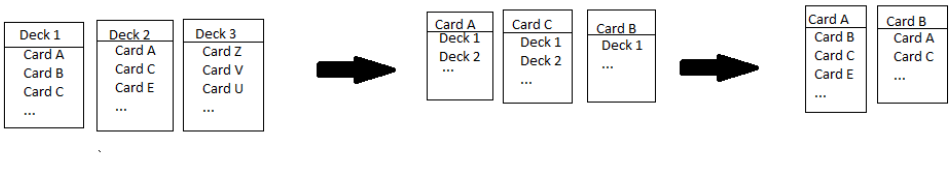

Figure 3.1. Double inverted index building process

The end product of this last step is an inverted index that uses cards as key and returns a list of cards that are played with that card in one or more deck and for each card played by the opponent it's possible to intersect each list obtained with the index, slowly getting closer and closer to the opponent's deck list.

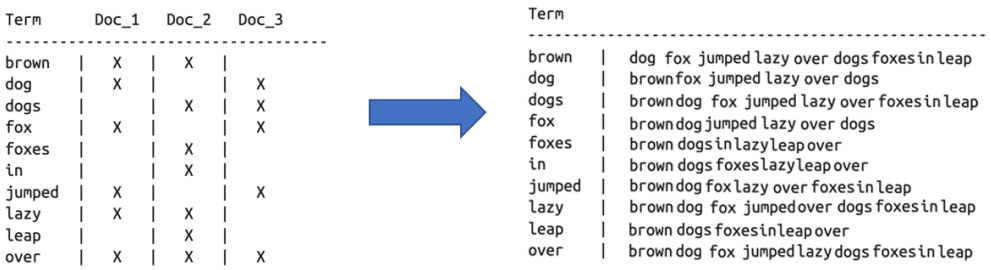

Figure 3.2. Double inverted index end result

Sometimes the meta-game information gathered isn't perfect or the opponent could be playing an unknown or sub optimal deck, which could contain cards that aren't in the inverted index; this could throw off the algorithm. There are many ways to solve this issue, each one with up and downsides:

- Ignore all impossible information: when a new cards would make the set of possible cards that the opponent could have in the deck empty, the AI discards the last card. This is the simplest method and it can return an optimal behaviour, however it performs the best only when the unpredictable card that the opponent played is the one getting discarded.

- Ignore the least likely information: when the nth card would create an empty set, the algorithm tests all possible n-1 sets that all the cards played thus far would make and keeps only the biggest one. This approach has a better chance than the one before to exclude the least popular card, meaning the inverted index can still perform well, however the calculation can be slow, the result isn't guaranteed and it might be necessary to repeat the process if the first choice wasn't correct.

- Produce multiple indexes: when a card would create an empty set, the algorithm splits the index in multiple n-1 indexes and keeps going on all non-empty one. This approach is guaranteed to exclude the least popular card, however the process is extremely slow and using the information

obtained by this set of indexes is complicated and computationally exhaustive.

Often times the **Ignore all impossible information** gives acceptable results while requiring by far the lowest overhead, however the other approaches are valuable when working without constraint.

3.3 Incorporating hidden information in MCTS

Using an inverted index to extract meta-game information isn't enough for the AI to understand it or use it in any significant way, the next step is to connect the possible cards the opponent could be playing to the MCTS.
Once the AI has access to the cards the opponent could play, it could use them to simulate what the opponent could do, if their hand contained a random set of cards obtained from the inverted index. This isn't perfect in therms of result, as the information is still speculative and the amount of cards is usually going to be quite high, but with this new simulation it's now possible to move the end of the tree from the end of the turn to any arbitrary point, whether that's during the opponent next turn or during any other turn after. Simulating the opponent turn has a few issues that how and when the information gathered from this simulation can be used:

1. Due to how the mana crystal system work, each node in the tree tends to have less options as the previous one; when a turn starts, however, all mana crystals are refilled and the number of options grows back up. This means that for each turn considered in a single MCTS calculation, the tree grows much larger and quickly becomes unusable.

2. Each time the AI simulates a new game-state based on a card in the opponent hand, the new state is only hypothetical and has a good chance of not actually happen. The more information is drawn from hypothetical positions, the higher is the chance of reaching unrealistic situation that have a very low chance of happening and tend to just increase the error of the algorithm without actually providing useful insight.

Both previous points show that the meta-game information isn't suited to drastically increase the depth of analysis of a position, and that not all new results are to be considered useful.

3.3.1 Blunder Threshold

A blunder is a mistake make out of carelessness and, in games, it can be the difference between winning and loosing. In Hearthstone blunders can take many forms, from playing a spell on the wrong card, to attacking a minion instead of the opponent when there's a chance of killing them, however the most common kind of blunder is when playing a card when the opponent has a clear answer that makes the first card nearly useless. This last kind of blunders is directly linked to the card from the opponent hand and it can be detect by just simulating the card being played; in other words this kind of mistake can be avoided by considering the opponent options without analysing all of their possibilities in detail.
A **blunder threshold** is a value used to consider if a move can be a blunder, given the correct cards in the opponent hand. When using MCTS, a blunder can be detected when the evaluation of a position drastically drops. While a drop is inevitable when simulating a move for the opponent, since they're trying to improve their position, sharp and sudden drops are usually connected with important and avoidable mistakes. Deciding if and when taking the information gathered from meta-game knowledge depends on many factors, for example game loosing mistakes should be avoided at all costs, but the chance of the opponent having the correct counter-move is also important, sometimes it's better to commit to a risky play, knowing the chance of blundering isn't zero, but still reasonably low, instead of making a safer play, that will often result in a weaker state most of the time. Lastly not all value are suitable blunder thresholds, that's because the opponent might have a good move, regardless of what the AI does, meaning it should play its best move and ignore the blunder.

```
float score = evaluate(gameState)
float opponentScore = evaluateBestOpponentFirstMove(gameState);
if ((score - opponentScore) > blunderTreshold)
        {
            return opponentScore;
        }
return score;
```

Figure 3.3. Snippet of the code implementation of a blunder threshold

To properly use the Blunder Threshold, the AI should evaluate each terminal position it reaches as normal, however, instead of using this evaluation

as the value for the node, it should continue the search, until it reaches the new terminal position (set by constraints instead of logical end-of-turn). After reaching the new terminal state it should calculate the new evaluation and use it if it's lower than the original by at a value equal or greater than the threshold, identifying a possible blunder and signaling to the rest of the tree.

3.3.2 Strategy-based BT

It would be impossible to determine the most optimal value for a blunder threshold, because the gravity of a blunder, compared to the risk of the opponent having a countermove, is constantly changing during a game, increasing and decreasing depending on many factors, such as duration of the game, what cards have been played and how the game has generally been going, but also parameters that are impossible to quantify, such as the opponent play style and skill level; on the other hand a static, supposed to catch all the important blunders, while discarding possible risk that are worth taking, can be determined with sufficient testing, assuming a tolerable error.
The static approach, however simple, can be improved by gathering absolute information from the game state. The most obvious factor that affects what constitutes a blunder, and the blunder threshold as a consequence, is the deck the AI is playing. Since the deck content, before getting shuffled before starting the game, is a known information for the player, the AI can adjust the blunder threshold to better suit the deck and its supposed strategy, for example:

- An aggressive deck is supposed to take high-risk high-reward bets, giving the opponent an advantage on the board, while reducing their health to 0 as quickly as possible. Aggro decks should use a blunder threshold that reflects this kind of pattern and detect blunder only when their very serious.

- A control deck has to play a slow, but calculated, game, where the opponent will inevitably gain some advantage and it's important to not over commit, while keeping the phase of the game slow. Control decks perform better with a blunder threshold near the middle, where they don't panic over every great play the opponent can make, but can still realize when the game is slipping away and react accordingly.

- Midrange decks play the best with a very low blunder threshold, that's

because their strategy involves getting small advantages at every possibility and any slip up could lead to a defeat. Since adaptability is midrange's best weapon, a higher caution around making mistakes is very helpful.

By basing the blunder threshold on the strategy of the deck, it possible to alter the value before starting the game, working with a static value, that is easier to test and nudge towards the optimal value, with the added benefit of using a more tuned threshold.

3.4 Single step simulation

After highlighting how to use meta-game knowledge to improve the AI results, it's necessary to show how theese data structures actually take form inside the MCTS. When reaching a "end of turn" action in the tree, the agent should start to simulate moves from the opponent, using the inverted index to determine the cards in hand; in practice, however, the index returns a list of possible cards and it's up to the AI to exhaust all the interesting combination that could compose the opponent hand and start simulating from there. In addition to the increasing complexity that multiple combination of cards can create, the number of possible options is very large, due to the newly refilled mana crystals, meaning that expanding the tree from this point on is very expensive. In order to work within time and computational complexity constrains the best node where to stop the expansion is close to the "end of turn" one, ideally the one right after that. The proposed approach is to run the MCTS as usual, but instead of stopping at the end of turn, the AI should create a random hand of cards, taken from the ones returned by the reverse index, and simulate all of the options once; at the end of the process the AI should consider the best option found for the opponent and check if the gap between the evaluation of that state and the one of the "end of turn" state is higher than the blunder threshold, then adjust the evaluation of the previous state when necessary, before continuing with the normal MCTS algorithm.

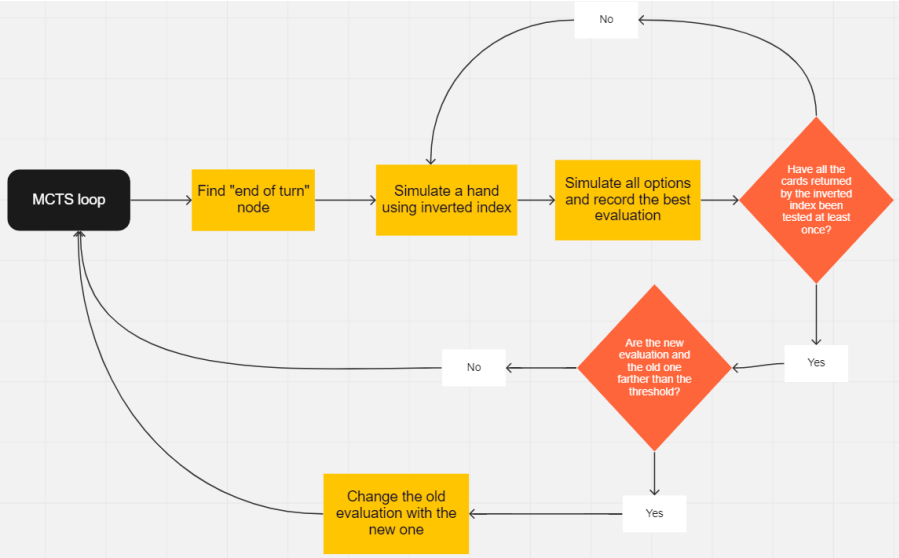

Figure 3.4. Single step simulation flowchart

Because the AI is only simulating one node after the end of the turn, it can skip checking all the combination possible, since it will never check what happens when playing more than a card, this drastically reduces the complexity of the tree. With a single step, the AI can detect very obvious mistakes that it would normally be blind to, such as playing a minion that can be easily destroyed with an attack or filling the board with minions when the opponent has access to spells that can kill multiple at the same time. A downside of single step simulation is that the AI is blind to possible two or more cards combination that can turn the tide of the game, luckily this kind of combination aren't very common and, sometimes, the first card of a combination can be enough to point at a possible blunder and suggest a different sequence of actions.

3.4.1 Testing the AI

In order to find the best parameters for the AI, as well as to best understand the behaviour of it, it's necessary to have an adequate testing environment, with conditions that become gradually more complicated and highlight what works and what doesn't.

Complete information

The first step in testing any strategy based on hidden information is to check if this kind of information is even helpful. To do so we decided to do is to have the AI play against other AIs while having perfect information of the opponent hand, this way, if the AI that uses the cards in the opponent hand performs better than an identical counterpart that doesn't use this kind of information, we can say that the strategy has the potential to work, since, if every guess it makes on the opponent hand is correct, than it will perform like in this test. The results obtained in this phase aren't strict proof that the strategy and the parameters being tested will return a strong AI, but this step is necessary to understand which strategy don't have chance of working, for any reason whatsoever.

Deck knowledge

After a strategy has proven that it has potential, the next step is to test whether the inverted index, and consequentially the uncertainty of what cards the opponent is holding in hand, is a limiting factor or not. The best way to test the performance of the AI when having to guess the cards in the opponent is to fill the inverted index database with a single deck, that is exactly the one the opponent is playing, this way the AI will never get confused about possible other decks or cards that don't have a chance of actually appearing in the game. During this test the AI is bound to get worse, or at best keep the same results as before, because every time it starts analysing the opponent's options it has to try all possible cards in their deck, devolving more time than before at this activity, therefore spending less time expanding the MCTS and exploiting the results. This is the first phase where the AI actually uses hidden information to make decisions, however, despite the cards in hand being unknown, the AI should always take into consideration all the possible next move by the opponent, since it will try all of them.

Meta-game knowledge

If a set of parameters still gives acceptable results after the previous phase, then it becomes the subject of the most complex set of tests, that closely resemble an actual competition: the database of the inverted index is filled with all the popular decks found using meta-game knowledge, and then the AI is put up against a multitude of opponents with different parameters. The database should contains enough decks that the inverted index can't be used

to find the opponent exactly deck with just a few cards, but if the deck is contained in the database, the AI will eventually know what cards it's up against, slowly returning to a case similar to the one in the previous phase.

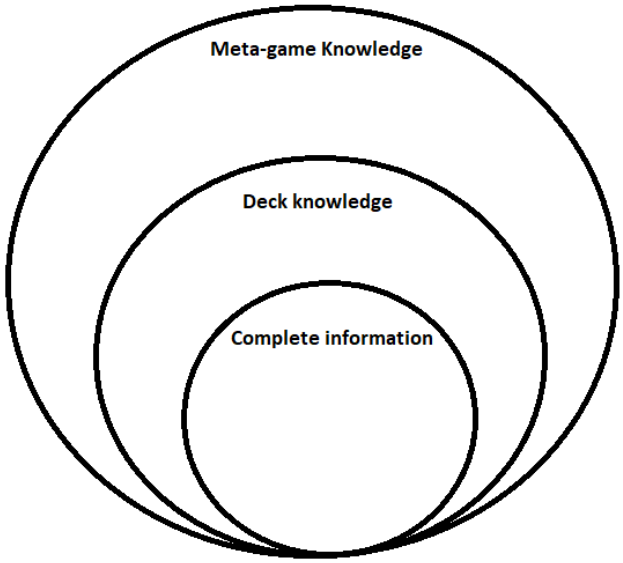

Figure 3.5. Games in each testing phase include all possible games in the previous phases

This last phase should show if the AI is performing accordingly to expectation in an environment that is very complex, but is also realistic.

3.4.2 Multiple steps simulation

A different approach from single step simulation can be taken: instead of trying all possible one-move options, the AI can simulated a random hand and test it for a higher amount of step before evaluating the new position. This new approach, called Multiple Steps simulation (MSS), needs to be repeated a predetermined amount of time and then return the maximum evaluation found; this way the algorithm can scout ahead for the opponent's possible counter-plays, gaining insight and depth in the analysis, in exchange for the certainty of having tested all of the possibilities. In alternative the evaluation can be based off of an average calculated from all the simulations; this type of evaluation is more indicative of the average quality of options the opponent has rather than specifically looking fro blunders, therefore the blunder threshold might need to be adjusted to better respond to the different kind

of input. Multiple steps simulation has a chance of uncovering information that single step simulation could never find, however the number of iteration that the algorithm has to go through to find sufficient information, without it being misleading, is very high. Because of this multiple steps simulation is better used in situation where time and memory aren't a factor, or where the limits aren't very restricting.
MSS comes with a few risks that need to be taken into consideration, the main one being that some options might get overlooked. When the algorithm composes a random hand some cards, or combination of cards, will inevitably be tested more than others and sometimes when testing multiple moves at the time, the end result might not be as valuable as one found in between the steps taken to get there, meaning that taking random options could result in a worse evaluation for the opponent than one found earlier, hiding possible blunders behind random simulation. In addition to options being overlooked, the blunder threshold value needs to be reconsidered, because the change in evaluation can be more drastic than the one experienced with single step simulation and also because the likelihood of a simulated event actually happening is quite low and gets lower and lower the deeper the analysis goes.

Chapter 4

Experimental results

In order to prove the theory behind the efficiency of the inverted index and the effectiveness of the blunder threshold, we decided to test them by simulating games between different AIs and record their results.

4.1 Agents

The starting process involved selecting an agent that doesn't use meta-game knowledge upon which building all the data structures and algorithms that we're interested in. We started the testing process with 5 different agents:

- The first 3 agents were all different MCTS agents, that varied for a few parameters: tree policy, selection policy and estimator policy. Some were choosing the path to follow based on the best results, others based on the average of the visits, some where exploring new nodes at random, others were focusing on the ones that yield the better results, some were calculating estimation linearly, others by attenuating each member of it by performing a square root.

- The forth agent was a greedy agent: a greedy agent is more focused on exploiting the best solution it found thus far, rather than exploring multiple promising lines. Greedy agents are usually quicker, but often times get stuck in sub-optimal solutions.

- The last agent was an experimental agent, that could change its parameters during the game based off of the classes of the two players, as well as how the game was going.

All five agents were based on the same MCTS algorithm, however the different approaches made them excel at different things. We made all the agents play against each other testing the performance of everyone keeping track of all the different match-ups.
Here are the results:

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	65%	16.2%	76.2%	52.6%
As Mid-range	83.7%	51.3%	75%	70%
As Aggro	22.5%	27.5%	62.5%	37.5%
Overall	57%	31.7%	71.2%	53.4%

Table 4.1. Results for the MCTS1 agent

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	55%	12.5%	77.5%	48.3%
As Mid-range	85%	47.5%	78.8%	70,4%
As Aggro	25%	25%	51.3%	33.8%
Overall	55%	38.3%	63.4%	52.2%

Table 4.2. Results for the MCTS2 agent

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	46.3%	20%	88.8%	51.7%
As Mid-range	83.8%	56.3%	81.3%	73.8%
As Aggro	27.5%	23.8%	57.5%	36.3%
Overall	52.5%	33.3%	75.9%	53.9%

Table 4.3. Results for the MCTS3 agent

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	61.3%	12.5%	73.8%	49.2%
As Mid-range	83.8%	47.5%	76.3%	69.2%
As Aggro	21.3%	20%	30%	23.7%
Overall	55.5%	26.7%	60%	47.4%

Table 4.4. Results for the Greedy agent

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	22.5%	16.3%	78.8%	39.2%
As Mid-range	86.3%	47.5%	75%	69.6%
As Aggro	8.8%	17.5%	48.8%	25%
Overall	39.2%	27.1%	57.9%	41.4%

Table 4.5. Results for the experimental agent

If we take a closer look at the results, it's easy to spot 2 agents that perfromed better then the rest while still being quite comparable between each other: MCTS1 and MCTS3. Comparing the results of these two best performers shows that MCTS3 has the overall best winrate, but it comes mainly from its outstanding pefrormance when piloting the mid-range deck, while MCTS1 has is just .5% behind in terms of overall winrate, but has a more balanced winrate distribution. In the end we decided to use MCTS1 as the base for the new AI since it's the agent that shows a better understanding of the game overall and therefore provided the highest potential for improvement.
MCTS1 has the following characteristics:

- Calculates its best move based on the average of the results of the visit of each node.ù
- Preferred expanding nodes based on promising results rather than choosing at random.
- Calculated the evaluation of each position using a linear algorithm.

4.2 Testing process

To test the AIs in the best way possible, we provided them 3 different decks, each with a different strategy; this way we could see how each agent would perform in the whole spectrum and we had them play each other on a long series of games.

4.2.1 The decks

The first deck is an aggressive shaman deck that aims at flooding the board with cheap and aggressive creatures and lower the opponent hp to 0 as quickly as possible.

Figure 4.1. Aggro Shaman decklist

Figure 4.2. Control Warrior decklist

The second deck is a control deck that uses the warrior as its class. Thanks to he's hero power, that grants him more health, and the efficient removal spell at he's disposal, the warrior is well suited at withstanding the opponent aggression and reach the later part of the game, where he can play strong and potentially game ending minions, of which the deck is filled.

The third and last deck is a mid-range hunter deck. This deck plays many efficient minions, most of which have powerful effects upon death, such as summoning other smaller minions or dealing damage, this way the hunter can gain small value by attacking the enemy minion even when he looses some. The deck also plays some expensive spells and strong minions that can help closing a game out.

Figure 4.3. Mid-range Hunter decklist

4.2.2 The tournament

To see how each agent would perform, we put them against each other in a long series of game called **tournament**. In order to witness the proficiency of the agent in piloting a specific deck we decided to divide each tournament in two part:

- The first part is dedicated to see the ability of the AI in playing a given deck. During this phase we provided the same deck to both players and let them play in a double round Robin[1], with each player playing both as the first player and the second one. This phase was repeated 3 times, one for each deck provided, and for each deck we ran ten double round Robin.

- The second part of the tournament is spent investigating how the AI performs when its strategy is different from the one of the opponent. During this phase we made each AI play all other AIs with all possible match-ups that haven't been tested yet, meaning excluding warrior vs warrior, shaman vs shaman and hunter vs hunter, bot has the player going first and the player going second. We repeated this process a total of ten times.

4.3 Complete information results

The first step in testing a set of parameters is check whether on not they are promising. To do so we took the best performing AI, it being MCTS1, and implemented the part of the code responsible for the single step simulation. After that we tested its performance against MCTS1 itself while having both play with complete knowledge of what cards each player held in hand. We decided to use static blunder thresholds based on the strategy the AI was playing. After a few test, where the AI played against itself, we landed on:

- 0.5 when playing control: a fairly high value, still sensible to blunders, but it allowing small hypothetical mistakes.

[1]A round Robin involves each player playing against each other player, so that everyone plays everyone. A double round Robin is composed of 2 round Robin back to back and it's usually preferred to a single round Robin in games that have 2 sides with different advantages

- 0.0 when playing mid-range: the test showed that, when playing mid-range, always listening to the opponent simulation, rather than stopping at the end of the turn, lead to the best results.

- 0.8 when playing aggro: the threshold used for aggro is by far the highest in value, this is because when playing aggro the AI often had to take risks in order to win the game.

4.3.1 One vs one

Here are the result of the 2 player tournament between MCTS1 and **Hidden Information**, where the latter is the new AI: The results are really

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	60%	30%	70%	53.3%
As Mid-range	80%	55%	70%	68.3%
As Aggro	25%	25%	70%	40%
Overall	55%	36.6%	70%	54%

Table 4.6. Results for the one vs one tournament with complete information

interesting and show important patters:

- In all the mirror matches, control vs control, mid-range vs mid-range and aggro vs aggro, Hidden Information (HI) has a winrate above 50%, meaning it improved from MCTS, which was the version we used to decide HI's parameters.

- Mid-range is performing above all the other decks. This could be because of many factors: the deck might be stronger than the other two, it might be easier to play or it might be more resilient to mistakes.

- Aggro is the worst deck between the 3. Regardless of who play the deck, it usually comes out as the looser, the only difference is when Hi plays aggro vs aggro, where the new agent seems to play better on average.

Overall HI has a winrate higher than 50%, with high peaks in specific match-ups, therefore we decided to move forward and test the AI in a more varied field.

4.3.2 Complete Tournament

The next test was checking weather HI playing against different opponents, opponents that didn't share the same decision making method, would show worse results.
All of the following data has been extracted from a tournament against all 5 of the agent tested, using all three decks and giving the AI full information about the cards in hand. We only included the games played by HI. Com-

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	64%	22%	76%	54%
As Mid-range	83%	60%	80%	74.3%
As Aggro	33%	24%	49%	35.3%
Overall	60%	35.3%	68.3%	54.6%

Table 4.7. Results for the complete tournament with complete information

paring theese results with the previous ones we can see that the AI performs similarly regardless of the opponent, with a few exceptions:

- The AI's winrate when playing aggro vs aggro plummeted from a high 70% to slightly less than 50%, this could be because the other agents are more proficient at playing aggro in general, as it also shows when looking at the overall winrate vs Aggro.

- HI's worst match-up is now control vs mid-range, in which control is usually favourite. This result however is more indicative of how well all the AIs play the mid-range deck and how strong the deck itself is, rather than HI's ability to pilot a control deck; we arrived at this conclusion by observing the overall winrate as control, that went up rather than down despite the bad match-up.

- Despite the few places where the winrate went down, most winrate went up. This fact reinforces the fact MCTS1 was already a strong candidate and that HI's parameters usually perform better than the rest of the field.

With a consistent winrate, although with some shaky match-ups, HI shows it can be a promising AI.

4.4 Deck knowledge results

The next step in testing is observing HI's performance when it has to actually use hidden information. We implemented the inverted index algorithm and we loaded it with just the 3 decks in the testing pool. Thanks to the class system in Hearthstone the algorithm can narrow the decks down to just one at the start of the game.

4.4.1 One vs one

As we did in the step before, we started by testing Hi's performance against the base agent from which its parameters were taken.
Here are the results: As it was to be expected, the results in this phase show

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	60%	20%	80%	53.3%
As Mid-range	75%	45%	80%	66.6%
As Aggro	20%	45%	45%	36.3%
Overall	51.6%	36.6%	68.3%	52.6%

Table 4.8. Results for the one vs one tournament with deck information

a worse performance, compared to the one where the AI had perfect complete information.
A few takeaways from this test are:

- The main winrate drop is registered when playing against Control, this is due to two different reasons: the first one is that when playing against Control it can be very beneficial to know exactly when the opponent has access to powerful spells that can easily clear the board and reset any advantage gained, now with incomplete information the AI has to play around theese scary spell the whole game, resulting in a worse performance. The second one is related to the length of the game: Control decks tend to hold many cards in hand and prolong the game as much as possible, this means that it usually reach a high number of mana crystals and can therefore play a larger pool of cards. This high number of possible cards tends to require longer simulations, slowing down the main MCTS algorithm and reducing its efficiency due to time constraints.

- HI's Aggro vs Mid-range match-up significantly improved, despite still not edging a winrate higher than 50%. This shows how volatile and unpredictable the match-up can be but also how well the AI can play it, especially when considering it has an 80% when playing from the other side.

- Mid-range's winrate lowered a little. This was to be expected, as this deck is the one that most benefits from having complete information and is allowed to squeeze as much advantage from each move.

With a winrate barely over 50%, we still consider Hi as a small improvement over MCTS1, regardless of the performance getting worse, as this result was expected.

4.4.2 Complete Tournament

Not unlike the complete information phase, we put HI's newest iteration in a tournament against all the other agents. We expected the results to show a slight performance loss, compared to the complete tournament ran in the previous phase. Here are the results: As predicted, the overall winrate

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	57%	16%	69%	47.3%
As Mid-range	84%	54%	82%	73.3%
As Aggro	27%	28%	51%	35.3%
Overall	56%	32.6%	67.3%	52%

Table 4.9. Results for the complete tournament with deck information

lowered a little, with some match-ups being particularly notable:

- Control vs Mid-range is steadily in decline with each test. While the rest of the control match-up are still well above 50%, mid-range still proves to be in favor of the mid-range deck. It's important also to note that HI performs very well when playing the opposite side, which means it is still ate least on par with the rest of the agents.

- Piggybacking off of the previous point, control has a winrate below 50%. While some variance is still in effect, it's important to note that HI isn't always the favorite when playing the control deck.

- The Mid-range mirror match is back in favour of the new AI, even tho it's lower than the previous complete tournament. This result once again enforces the fact that hidden information are less reliable than complete knowledge, but being able to use them gives the new agent an advantage.

HI's performance at the end of this last tournament show marginal improvements over its original MCTS1 form, however its sligtly positive winrate, as well as its proficiency in playing the mid-range deck mean it's still an interesting test subject.

4.5 Meta-game Knowledge results

The last step in testing the new AI is to simulate an almost random environment, where the only information it has is a list of popular decks. This last test is the most important, as it resembles how actual competition would work. Before starting we predicted that the performance recorded would lower once again, however the difference between this and the previous phase should be lower than the one between the previous two, mainly because the inverted index should be able in most cases to narrow down the possible decks very quickly, falling back to the previous test case.

4.5.1 One vs one

We loaded over 100 decks in the inverted index and asked HI and MCTS1 to play a full tournament against each other.
Here are the results: The results are unsurprisingly similar to the ones from

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	55%	20%	75%	50%
As Mid-range	80%	60%	85%	75%
As Aggro	20%	35%	40%	31.6%
Overall	51.6%	38.3%	66.6%	52.2%

Table 4.10. Results for the one vs one tournament with meta-game information

the previous phase:

- Mid-range keeps performing the best, with a very high winrate, but HI keeps showing a solid winrate even when playing against it. This shows

that the approach has improved HI performance when playing Mid-range as well as a small edge when playing against it.

- Control sits at a flat 50% winrate, meaning the new hidden information didn't change much in the way the AI plays control decks. HI wins slightly more than MCTS1 does when playing Control, however the difference is so slim that we can probably safely say that this approach didn't change much.
- Similarly to Control, the Aggro result don't show major upgrades, while overall the AI preforms better than the opponent, the advantage is small even on a large number of games.

With the one vs one test going more or less as expected, we expected HI to settle on similar performances against the rest of the field.

4.5.2 Complete Tournament

The very last part of the test was to take HI's latest set up and put it against all the agents we started from.
Here are the results: As predicted, the result are very similar to the one from

Winrate	vs Control	vs Mid-range	vs Aggro	Overall
As Control	53%	13%	83%	49.6%
As Mid-range	81%	55%	76%	70.6%
As Aggro	32%	29%	60%	40.3%
Overall	55.3%	32.3%	73%	53.6%

Table 4.11. Results for the complete tournament with meta-game information

the last phase:

- HI's winrate is overall higher than 50%.
- The average winrate combining playing as and against a certain deck are all above 50%.
- The only few outliers are the exceptionally low winrate as Control vs Mid-range and the surprising rise in Aggro winrate. The first is symptom of how much the Control deck rely on the hidden information to evaluate when to commit to a play and when not to, the second shows again how the AI is just playing better than most other agents.

The results gathered with this last tournament show that HI has in fact improved, being able to outperform the same AI from which it was constructed as well as maintaining an edge over the rest of the AIs.

4.6 Comparing results

After gathering the final results for the new agent, it's worth comparing them with the ones gathered in table 4.2 and assess the newest data. By looking at HI's overall winrate, when playing a specific deck, we can see that new agent improved when playing aggro, became slightly better at playing mid-range, and became a little worse at playing control. Overall the HI's winrate is higher than its predecessor, signaling that the approach taken has had a positive impact on the agent, even if it could still use some fine tuning to counteract the negative impact it has when playing control. It's also worth noting that the starting point from which we gathered the results for HI differ slightly from the one of MCTS1: the new agent had to face all 5 other AIs, while MCTS1 only played against 4, as it did not play against itself, and, with MCTS1 being one of the best performers, therefore HI played against stronger opponents on average, naturally pushing its results lower. With this new consideration in mind, as well as the fact that MCTS1 was the agent with the best results, when playing against control, HI's results are even more impressive.

4.7 Multiple steps simulation results

During the development of HI using single step simulation we decided to study the results of a multiple step simulation agent to see if we could exploit the theoretically more promising algorithm. The first test we ran involved the new agent (MSS) playing with complete information against MCTS1. MSS used the same blunder threshold as HI, with the main difference being that instead of testing all possible options from the opponent once, it would simulate 1000 random sequences of options that all ended with the "end turn" action and use the best result as the opponent simulation. The results showed by MSS were very poor, the AI couldn't surpass 5% winrate when playing against its own carbon copy with perfect information, even when given ten times as much time to process information, to circumvent the high computational cost that the algorithm has. Multiple steps simulation remains an interesting algorithm that has a high chance of outperforming

single step simulation, however finding the right parameters and calibrating all variables to extract such a result will require a lot of testing. Because of theese considerations we decided to scrap the AI and continue on the more concrete approach.

Chapter 5

Conclusions

- When speaking about AI in games, hidden information is a wildly unexplored space, that can yield very interesting result, but one requires an unorthodox approach to it.
- We showed that textbook AI algorithms can improve their performances when coupled with well thought heuristics as well as sufficient testing, balanced between exploring new options and exploiting the most promising ones.
- The results gathered from the performance of the newly developed Hidden Information agent clearly prove that the approach we took has potential and can increase an agent's result.
- The Blunder Threshold is an effective method of detecting possible mistakes and works well with the way AIs approach a game. Even a simple approach like the static strategy-based threshold is enough to guide an agent towards a better option.
- The inverted index is a powerful algorithm that can help sort information quickly. It's use has been essential to reach our results and similarly powerful algorithm will need to be found when applying to other kind of hidden information in different games.

5.1 Possible improvements

While the AI we developed shows promising results, it doesn't mean it can't improve with additional work, tests and new idea. There are two main points that we can spot where the AI could be improved:

1. Even more precise Blunder Threshold.

2. Multiple steps simulation.

The first area of improvement is to look for more accurate methods of determining the blunder threshold's value. We settled for the simple, yet effective, static strategy-based threshold, however this approach voluntarily ignores a lot of possibly useful information, such as the opponent's supposed strategy or the current state of the game. With a more accurate threshold, that can span between sensibly different values, the AI could more accurately detect blunders and determine weather or not to take risk. Not unlike the first point, multiple steps simulation has the potential to return more intricate and complete information about a certain game-state, which in terms leads to better play. We decided to focus on single step simulation and find useful results there, however it's surely possible to find better results with a correctly calibrated multiple steps simulation. The challenge with MSS is to determine the correct simulation policy when simulating the opponent's turn, as well as find the optimal ending point, weather that's a specific action, the end of the game or a predetermined depth. There is also the possibility that both of theese characteristics could be mutable during the game, meaning that an agent capable of understating when to analyse deeper vs when to stop early could prove an interesting research topic.

5.2 Alternative routes

The Montecarlo tree search is just one way to perform adversarial search and while it is a very powerful one, it doesn't mean it's the best way to do it. In the past few decades many games have been conquered by AIs, some examples include the complete information board games chess and go, but span all the way to strategy based real time videogames like Dota2 and Starcraft2. The common factor of many of theese AIs is that they are all piloted by neural network, that are data structures that aim to imitate the behaviour of the human brain. Many of theese neural networks are trained by letting the agent play against itself an enormous number of times and adjust its weights to improve in a process called supervised learning. While neural networks tend to be very cryptic and hide their internal behaviour to human observer, due to the sheer mathematical complexity they work with, it would be possible to develop AIs for Hearthstone using this same technique and even provide meta-game information to the neural network.

Using meta-game information to train a neural network is a very unexplored topic, but it could lead to very important result.

5.3 Other interesting study case

Recently a new card game has been developed with the aim of providing a set of rules that make it easy to develop AI to play, the name of this game is *Legends of Code and Magic*. This new game is similar to Hearthstone, but is more deterministic and has an overall simpler set of rules. The one particular that makes Legends of Code and Magic interesting when talking about hidden information is that the AIs don't play with preconstructed decks, but are tasked to build one from a set of procedurally generated cards. This means that players can't use meta-game information, as each game is completely different form the previous the others, however, since the pool of cards given to the AI to construct the deck is the same for both players, it's possible to save theese cards and use them the same way we did, incorporating human-researched meta-game knowledge directly into the decision process of the AI. Legends of Code and Magic already has a prominent community of amateur coders, as well as academic researchers, interested in it and it could very well become the next topic of future research.

Bibliography

[1] Qi-Yue Yin, Jun Yang, Kai-Qi Huang, Mei-Jing Zhao, Wan-Cheng Ni, Bin Liang, Yan Huang, Shu Wu & Liang Wang, *AI in Human-computer Gaming: Techniques, Challenges and Opportunities*

[2] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki & Jacek Mańdziuk, *Monte Carlo Tree Search: a review of recent modifications and applications*

[3] Jiangong Zhang, Xiaohui Long, Torsten Suel*Performance of compressed inverted list caching in search engines*

[4] Chiara Federica Sironi *Monte-Carlo Tree Search for Artificial General Intelligence in Games*

[5] Noam Brown, Tuomas Sandholm, Brandon Amos *Depth-Limited Solving for Imperfect-Information Games*

[6] Jonathan Rubin, Ian Watson *Computer poker: A review*

[7] John C. Harsanyi,*Games with Incomplete Information Played by "Bayesian" Players, I–III Part I. The Basic Model*

[8] Dirk Bergemann, Stephen Morris,*Robust Predictions in Games With Incomplete Information*

[9] Elemar Júnior,*A First Take at Building an Inverted Index and Querying Using C#*

[10] Algoscale, *Why you need to know about Adversarial Search in AI?*

[11] André Santos; Pedro A. Santos; Francisco S. Melo,*Monte Carlo tree search experiments in hearthstone*

[12] Amy K. Hoover, Julian Togelius, Scott Lee & Fernando de Mesentier Silva, *The Many AI Challenges of Hearthstone*

[13] Pablo García-Sánchez; Alberto Tonda; Giovanni Squillero; Antonio Mora; Juan J. Merelo, *Evolutionary deckbuilding in hearthstone*

[14] Chaslot, G., Bakkes, S., Szita, I., Spronck, P.,*Monte-Carlo Tree Search: A New Framework for Game A*

[15] Rémi Munos, *From Bandits to Monte-Carlo Tree Search: The Optimistic*

Principle Applied to Optimization and Planning

[16] Yan Xia, Kaiming He, Fang Wen, Jian Sun, *Joint Inverted Indexing*

[17] Markus Eger, Pablo Sauma Chacón, *Deck Archetype Prediction in Hearthstone*

[18] Fernando de Mesentier Silva; Rodrigo Canaan; Scott Lee; Matthew C. Fontaine; Julian Togelius; Amy K. Hoover, *Evolving the Hearthstone Meta*