Quantifying Hidden Information

by

Troillet Lucien
Student ID Number: 1246005

A dissertation submitted to the
Engineering Course, Department of Engineering,
Graduate School of Engineering,
Kochi University of Technology,
Kochi, Japan

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

August 21, 2023

Abstract

The study of games to develop increasingly complex and "intelligent" algorithms has been a subject of interest for years with good reason. Games provide a great "sterile" environment to study and develop intelligence. They are essentially the laboratory environments of Intelligent systems as they can provide many different levels of difficulty and varying types of challenges while still having well defined rules, precise goals and easily quantifiable rewards. Additionally, they do not have any real-world consequences so they do not pose any risk, negative effects, or responsibility problems when experimented with while still allowing to train on increasingly challenging problems.

Deep Blue's victory against Kasparov was probably the first instance of an "artificial intelligence" beating a human expert at a complex task in the eyes of the grand public and, since then "intelligent" programs have been able to surpass humans on increasingly difficult and complex tasks. Among others, AlphaGo, AlphaStar, Libratus and MuZero have been great landmarks of reinforcement learning.

At this point, AI having achieved superhuman performance in Perfect Information Games, research is increasingly concerned with Imperfect Information Games; a subset of all games that makes some information about the current game states obstructed from the players, creating a situation where, to be efficient, one has to take this uncertainty into account and be able to deal with it efficiently. Imperfect Information Games are probably most commonly card games where one does not receive information about cards possessed by an opponent. There are however many other types such as games with an inherent level of randomness like dice games or games with other types of hidden information like video games using fog of war or hidden enemy weaknesses, etc...

It is easy to see why there would be such interest in these more uncertain scenarios. In real world situations, one often does not have access to all relevant information about a problem while still needing to take action based on available information. In that sense, Imperfect Information Games are much closer to real life problems and being able to play such games efficiently with trained computer agents will be a requirement to being able to solve everyday life problems with reinforcement learning. In short, Imperfect Information Games are a key step towards using intelligent computer agents in less “sterile” environments.

However, while more research has focused on Imperfect Information Games, terminology has not yet followed the current progress. The current vocabulary has not changed from the one used to explain the challenges of Perfect Information Games despite the challenges of Imperfect Information Games being very different from the ones posed by their counterparts.

When searching for reasons why a game is studied, one oftentimes finds one of the three most common reasons listed: either the game is very popular, or it has already been studied in previous research, or it is difficult. These first two reasons are not inherently scientific and the third, while being more objective, is often explained using very basic measures of difficulty such as action space or state space size but these measures do not give any information about the specific challenges of a game, nor are they any different from the measures used to describe Perfect Information Games.

In this thesis we focus on defining new terminology and illustrating how it can be used to further classify Imperfect Information Games. First we introduce overarching themes of Imperfect Information Games, the current vocabulary used to describe them and its weaknesses. We then propose two new concepts specifically related to Imperfect Information: Imperfect Information Impact and Imperfect Information Visualization. Following this we conduct two case studies, one qualitative and the other quantitative, on how these concepts appear in Imperfect Information Games.

In the first case study, we show the main overarching techniques a player can use to deal with Imperfect Information in games and illustrate how these behaviors can emerge in different games. To do so, we create multiple increasingly challenging versions of an Imperfect Information Board Game called Geister and exploit its specific rules to show variations in emerging gameplay. We then link these behaviors to the two main concepts we put forward in this work.

In the second case study we focus on a series of versions of DouDizhu, a type of card games originating from China. Using these card games, we train multiple agents with variable access to imperfect information and compare their performance as well as their ability to achieve an inner representation of the information they are missing about the game world. Their performance is then used as a way to measure both the Imperfect Information Impact and Imperfect Information Visualization of the different versions of DouDizhu. With these measures we then compare the different version of DouDizhu to analyze how they compare to our expectations.

Contents

List of Figures

List of Tables

Acknowledgements

To consider the number of people who had an influence in bringing me, not only to the end of this thesis, but also simply to its beginning is an impossible task. The simple fact that the journey that brought me all the way to doing a PhD in Japan started 13 years ago from a friend promising me some beers if I joined an event shows how strong the butterfly effect can be. So thank you Julien Shaefer for starting my journey. But I should probably focus on people who had a more direct impact on this work or the acknowledgements will be longer than the thesis.

Foremost, Professor Kiminori Matsuzaki who has been a great doctoral supervisor all along this academic journey. He has been a very kind and helpful supervisor, always coming up with very interesting points and ways to consider problems. He also definitely noticed my love for trying new and random things and not only encouraged it but participated in it in some occasions; this led to me maintaining the university's gpu cluster and to an interesting little code optimization competition in which he absolutely destroyed me. Me being a person who is easily stressed by all bureaucratic procedures and deadlines, Professor Matsuzaki's relaxed and easy-going attitude has done wonders in keeping me calm throughout some of the more stressful parts of the PhD. He also was very supportive during the difficult starting period of the PhD, while I was stuck in Switzerland due to the Covid pandemic, and even took time on some occasions to help me with experiments and writing when I was struggling to deal with this strange situation.

Professor I-Chen Wu, Wei Thing-Han and Yoshida Shinichi also have my gratitude for some of their key insights on my work and great suggestions for ideas to expand it in the future.

Thank you as well to Wang Weikai and Li Chuanfa, my two co-doctoral students who had a strong participation in many aspects of my work. I only hope that I might have helped them as much as they helped me through long nights of work.

All of the Japanese students in my lab also deserve some gratitude for all the times they helped me deal with various problems arising from my lackluster Japanese skills. They have been a great source of support and fun to hang out with despite the difficulties in communication.

I would also like to express my gratitude to Kochi University of Technology and a special nod to the people working in IRC for giving me such a nice opportunity to do research, especially on the other side of the world.

For some of the more personal marks of gratitude, I would obviously like to thank my mother and father, Nathalie Troillet-Tièche and Nicolas Troillet, who have been supporting me and encouraging me in all my wild life choices.

There are so many more people who have helped me get here in their own small or big way and they have all my gratitude for their influence even if they don't realize it. I hope that I too can have a small positive influence in their lives.

One of the most important scientific influences in my life has always been my grandfather, Yvan Tièche, who I feel I inherited the spirit of a scientist from. I really wish he would have gotten to see his grandson achieve the same milestone he did so long ago and how much I feel he had an impact on it. I'm sure he an I would have had really interesting discussions about my work.

I hope his spirit will carry on through me.

Chapter 1

Introduction

1.1 Background

Imperfect Information Games (IIGs) are currently a focus of multiple research papers. Card and video games especially [5, 6, 10, 17, 21, 39]. After the successes of trained intelligent agents on Perfect Information Games (PIG) such as Chess and GO, both of which presenting challenges that require the ability to efficiently navigate through action spaces too big to fully explore, research has naturally started moving on to different challenges, one of the most prominent of which being Imperfect Information.

In games Imperfect Information is the presence of some level of uncertainty which can manifest in some elements of the game being hidden to a given player or in some elements of the game being dictated by a random distribution. Altogether the most basic requirement for a game to be an IIG is that when taking some of their actions, the player cannot know with certainty what world state it will lead to.

Dealing with Imperfect Information is a logical continuation for the field of AI research. Most real life situations include some unknowns, thus it is logical that the development of more advanced trained agents would require being able to deal with situations where not all variables are known.

While several studies on IIGs use multiple games as training and testing environments to improve and create better algorithms, they mostly do not analyze the games themselves. There is of course ample terminology and notation systems allowing to describe Imperfect Information Games which are indeed informative of what challenges the game can pose and/or help describe the game in a way that can be used for algorithmic purposes. This is, however, often more of something that is used to present the algorithms used to play these games rather than to characterize the game itself. Even taking this into account, such categories as Imperfect Information Extensive Form Games [26] and Partially Observable Stochastic Games [19], are high level descriptors that can apply to a wide range of games. After all, "Rock-Paper-Scissors" can fall into both of the two previously mentioned categories, that does not necessarily make it a game worthy of developing intelligent agents for.

Altogether, there isn't any terminology or measures specifically designed to be able to quantify the challenge given by an IIG. Measures previously used to characterize the complexity of PIGs, such as state space size, are also applicable to IIGs, however the challenge of IIGs does not reside in the same place as that of PIGs. Otherwise the same algorithms could be use for both and produce good results. Hence, IIGs should also have measures that specifically allow to quantify their challenge but, so far, such concepts seem to be lacking in research.

As far as the authors know, only a few studies did an attempt of analyzing games in a way that quantifies their characteristics. In the work by Wakatsuki et al. [40] multiple players were used to gather information about game complexity on different game variants. In the work by Goodman et al. [18] the authors attempt to characterize multi-player tabletop games using feature-spaces obtained through gameplay.

1.2 A Quick Walk-through

In this work we aim to observe how Imperfect Information affects games; to do so we proceed in two main phases both of which rely on a common principle.

Common to both phases, is the observation of trained agents. In this paper we propose a different approach to analyzing games through trained advanced players. Given the complexity and stochastic nature of IIGs, it is mostly impossible to solve any of them at any level. In the absence of the possibility to solve such games, a reasonable alternative consists in analyzing the behavior of advanced players in an attempt to deduce game characteristics. Thus, in order to observe the influence of Imperfect Information on games we need to rely on current best performances as a substitute. To do so, we use trained agents to observe their strategies and how Imperfect Information affects their play style.

The two phases are essentially case studies done on different games. In the first case study, we observed Geister, a board IIG from Germany[3]. The details and rules of Geister are described in Section 4.1. To do so we took inspiration from Chen and Kaneko [14] to train CFR based agents for Geister and then observed what behaviors they developed during training to infer how Imperfect Information affected the gameplay. Geister has the advantage of being easily modified by increasing or decreasing its board size without needing to change any core rules of the game. These modifications do however change the strategies a player must employ to be effective. To observe how strategies changed, we created multiple versions of Geister with varying sizes of the game's board and varying number of pieces. From this point, we analyzed how the agents changed their behaviors to better suit the different environments in which they were trained. We then attempted to interpret said behaviors to isolate key elements of importance when dealing with imperfect information. Geister lent itself well to this assessment as it contains both elements of perfect information and elements of imperfect information. As such, it allows to see which of these elements a player prioritizes in order to win the game. Through this analysis we mainly put forward two concepts to further research in the second case study, namely imperfect information visualizability and imperfect information impact.

In the second case study we relied on Deep Monte Carlo [44, 46] to play three versions of DouDizhu, a popular Chinese card game. DouDizhu is a three player game with both elements of cooperation and competition. It is one of these card games that is really popular in an area of the world but not to a point where there would be a specific governing body to establish solid rules for it. This makes it all the more interesting for our purposes as it presents multiple different versions (of which we selected three) with slight variations in rules while not having too much differences in core gameplay which made it easier for us to design training procedures for these different games based on previous best performing work [42, 44, 46]. With this algorithm we train agents for the three versions of the game throughout two experiments, one designed to measure visualizability by training a sub-network to estimate hidden information and one to measure impact by providing some players with cheated inputs that have access to more information about the state of the game then they would usually have at their disposal.

We then compare all the versions of DouDizhu as an example of how impact and visualizability might help describe the specific challenges of different imperfect information games.

1.3 Contributions

We consider the following to be the main contributions of the thesis.

The first contribution is the proposal of new concepts and terminology following the analysis of the vocabulary and measures used to discuss Imperfect Information Games and how they fail at expressing the challenge posed by these games

Our second contribution is the qualitative analysis of the concepts we put forward and how their influence can be seen in the strategies that players elect to use in games through the case study of Geister.

Our third contribution is the creation of a methodology to measure our main concepts and the selection of appropriate measures to use in said methodology illustrated through the case study of DouDizhu.

1.4 Outline

This thesis is organized as follows. In the current chapter we introduce the main components of the work and further detail the content to come.

Chapter 2

We introduce some of the main concepts of the current work in this chapter either already existing or developed by us. Said concepts are subdivided into general game terminology, more context on imperfect information specifically and finally an introduction to the main concepts we put forward in this thesis.

Chapter 3

In this chapter we present the mathematical notation used for this work and introduce the two main algorithms used. The mathematical concepts are explained using Partially Observable Stochastic Games terminology and then used to introduce our two main algorithms: *Deep Regret Minimization with Advantage Baselines and Model-free Learning* and *Deep Monte Carlo*.

Chapter 4

This chapter contains the first case study. Here we introduce the game of Geister and its rules, we also dive deeper into how we modify the game to create different versions of it. Following this more introductory part, we detail the experiment design we operated with, present the results of said experiment, further analyze them, and discuss our findings and their implications.

Chapter 5

This is the second case study done on DouDizhu and its variants. First we introduce all three variants and their specific rules. Afterwards we give more details on the algorithms used for both of the experiments and then present each experiment and their results.

Chapter 6

In this chapter we go over some of the questions that were asked or we feel might need clarification.

Chapter 7

We summarize this thesis and discuss the future work.

Chapter 2

Concepts

2.1 Game Terminology

As this work strongly relates to games, it is important to establish some terminology especially related to Units of gameplay length.

From here on, we will use the terminology established in "Characteristics of Games" [16] and presented hereafter:

- *Atom*: The smallest complete unit of play. Does not require a winner to have been determined yet, but should contain enough elements of the game and decision making that the players can feel they have truly played the game. This can be one full turn of play in a board game, a level in a video game, the period of gameplay between the last point(s) earned and the next one(s) earned, etc...
- *Game/Round*: The standard full unit of play. The length between a starting state and the determination of a winner (Not necessarily the final winner in games that have multiple rounds).
- *Session*: A single continuous period of play.
- *Campaign*: A series of game sessions that are linked somehow.
- *Match*: A series of games determined to be enough to decide a final victor. Best of three is a typical example. Volleyball and many E-sports use a Best of five system. Games were money is bet often play until certain earnings have been achieved by a player.

The exact range that these concepts encompass can vary greatly between games due to the huge variety of rules that can exist in the universe of all things that can be considered to be games. In some games, some or all of these concepts might not be applicable and in others two or more of these concepts might be the same things.

In this work we consider that an Atom is defined by all players taking one action, a game/round is defined by a reward having been obtained by all players (the game ending), and a Session is one training iteration when applicable. Finally a Campaign would be the full training time of our agents. This attribution of terminology would not necessarily be usable on games other than the ones considered in this work. It is not necessarily a common thing to apply these terms to reinforcement learning procedures but it helps for clarity's purpose.

2.2 Imperfect Information

The main purpose of this thesis is to discuss Imperfect Information and, how it is viewed and characterized in games. As with any complex concept, the first step of talking about it is defining it. To do so we start with games where it is absent as a contrast: Perfect Information Games.

2.2.1 Perfect Information Games (PIG)

Even starting from PIGs, definitions can be a bit confusing. Strictly speaking, PIGs are games where players have full knowledge of the current state and history when relevant. However, when papers discuss PIGs, they generally apply a more restrictive definition that also excludes stochastic events; hence non-stochastic PIGs (nsPIG). This further restricts the definition to: games where players have full knowledge of the current state, history and action outcome. NsPIGs are hence fully deterministic and thus solvable in the sense that, given perfect players, one can always guarantee a specific final game outcome.

When nsPIGs are discussed, it is commonly in relation to how they have been solved. A two player finite zero-sum nsPIG can result in three outcomes (or game-theoretic values):

- **First-player Win**

- **Draw**
- **Second-player Win**

A game is solved when the final game-theoretic value resulting from optimal play by both players is known. Generally three levels of solving are recognized [20]:

- **Ultra-weakly solved**: the theoretical end result from a game start position is known,
- **Weakly solved**: the theoretical end result and a strategy to force this result are known, and
- **Strongly solved**: a best move can be found from *any* state in the game.

While these distinctions are a good foundation for nsPIGs, they are not simply applicable to IIGs. Due to their stochastic nature, it is rare to have even an ultra-weakly solved IIG.

The second main way to talk about PIGs and nsPIGs is to quantify their difficulty. The most common way to do so is by describing the complexity of their state- and/or action-space, generally in terms of size. In short, the state-space complexity is the number of possible states one can encounter during a game and the action-space complexity is the number of possible actions existing across any state of the game. These measures are more readily applicable to IIGs.

In Table 2.1 we show some examples of nsPIGs and both of these measures. It might be worth pointing out here that these two measures have a fundamental difference: Space complexity is a measure that is more inherent to the game while the levels of solving are not. Essentially, levels of solving are a descriptor of our level of achievement in the game's research.

Nevertheless, one can see from this table why state-space complexity would be a good measure of difficulty as, with increasing order of state-space size, the solving status progressively decreases. Hence, the main challenge of such games is to find strategies that allow a player to navigate efficiently state-spaces that might be too big to fully explore.

2.2.2 Imperfect Information Games (IIG)

As discussed in the previous subsection, there is some confusion as to what exactly an IIG is. In a general sense the opposite of nsPIGs is any game with some level of uncertainty other

Table 2.1: Game state spaces and resolution status

Game	State space order	Status
Tic-tac-toe	10^3	Strongly solved
Dōbutsu shōgi	10^8	Strongly solved[37]
Awari	10^{11}	Strongly solved [28]
5x5 Quixo	10^{12}	Strongly solved [36]
Kyoto Shogi	10^{16}	Weakly solved [31]
Checkers	10^{20}	Weakly solved [29]
Qubic	10^{30}	Weakly solved [27]
Chinese checkers	10^{78}	Ultra Weakly solved[33]

than final outcome uncertainty.

Final outcome uncertainty is the one form of uncertainty that is somewhat inherent to all things that are classically considered games. “Characteristics of Games” [16] outlines how difficult it is to establish exactly what anything needs to be qualified as a game. However the one thing that they do give as a necessary requirement for something to be a game is **Final Outcome Uncertainty**. All games require that the final outcome is not known in advance to truly be a game. A given player cannot know for sure if they will win or lose at the start of a game otherwise there is no reason to play. If a game is fully solved, it stops being a game. As long as the players have a strong enough understanding of the game’s strategy to, without fail, reach the theoretical perfect outcome (weakly solved), any game collapses into a problem with a solution. In that sense a “game” can very generally be defined as a problem of which the solution is not known to the parties attempting to solve it. Altogether, this final outcome uncertainty is a required Imperfect Information element in any game.

However, while an important concept and technically a form of Imperfect Information, final outcome uncertainty is not generally what is described when talking about Imperfect Information games. Imperfect Information as a concept can be a bit hard to pin down as all complex concepts are. Starting from the more general concept of **Uncertainty**, we already established that all games at least have final outcome uncertainty. NsPIGs only have final outcome uncertainty and no other source whereas IIGs have more sources of uncertainty.

The two main ways uncertainty can be present in a game is through stochastic processes or obfuscated information which, to separate form final outcome uncertainty, we will group together under the term of **gameplay uncertainty**.

Stochastic elements can take many forms, the most typical example would be dice. This does not mean that all stochastic processes within a game cause it to inherently become an IIG. For example, a random distribution of cards at the beginning of a game does not create an Imperfect Information game if the cards are then revealed. Randomness creates an IIG if it makes some parts of the future states of the game indeterminable at the time a player takes an action. In a coin flip game, the only action is to choose heads or tails but randomness makes it impossible to determine in advance which action is the best.

Obfuscated information can originate from two main phenomena:

- Hidden and simultaneous actions
- Hidden Information

Simultaneous actions happen in games like Rock, Paper, Scissors where both player take their action at the same time. There is no randomness in the process both players use to choose their action but the outcome is still uncertain at the time the action is taken. Hidden actions, are practically identical to simultaneous actions. An example would be a game where players have to select a card to play from their hand and all players reveal their card simultaneously.

Hidden Information is when some of the current state of the game is already determined but hidden from one or multiple players. This has two main sub-categories: Private Information and Environmental secrets. Private Information is any information that "belongs" to a player. A typical case would be a player's hand in poker. Until the final phase of a poker game, no other player has access to that specific information and the best they can do is to use other player actions to attempt to deduce it. Environmental secrets are information that is part of the environment of the game that no player has access to. In Chinese dark Chess for example, pieces are flipped so that neither player knows who it belongs to and what type it is. Until a player chooses to flip the piece this information is hidden from both in the same way. These two categories are however not always well defined. Fog of war in strategy style video games, for example walks the line of these two categories as it can hide information of the current state of a map which might contain both details about a game environment and its neutral resources but also enemy positions and resources.

There can be overlap between stochastic processes, simultaneous actions and hidden information. To reuse the previous example of Chinese Dark Chess [11] , while in real games the state of each hidden piece is determined through random distribution at the start of the game, one can also consider that every reveal of a piece is in fact a stochastic event with equal probability for each remaining piece to be revealed. In such a case, the Environmental Secrets are equivalent to pure Randomness. This would however not hold up if there was a way to access the hidden state of pieces without revealing it to the other player in the game.

In this work, we focus on hidden information.

The practical consequence of gameplay uncertainty is that while a single round of any two-player game will still finish in one of them winning or a draw, there will not be a single possible outcome from perfect gameplay. Hence which player wins or loses becomes a continuous sliding scale of which there are five notable divisions:

- **First-player win** ($p_1 = 1$)
- **First-player dominated** ($0.5 < p_1 < 1$)
- **Draw** ($p_1 = 0.5$)
- **Second-player dominated** ($0 < p_1 < 0.5$)
- **Second-player win** ($p_1 = 0$)

In the end, while IIGs do also require the ability to navigate large state spaces, their main challenge resides in being able to manage the inherent uncertainty of these games. While managing stochastic processes mostly comes down to probabilistic calculations, managing uncertainty due to hidden information is a much more complex skill as there are many context clues a player can obtain by observing other player's behaviors. In some games, interpreting these clues and exploiting them is a key aspect of being able to perform well.

Following is a summary of some of these terms as we understand them in this work:

- **Perfect Information Games**: games that do not obfuscate any part of the game state or history

- **Imperfect Information Games**: games where part of the game state or history is obfuscated
- **Hidden Information Games**: sub-set of Imperfect Information Games where part of the game state or history is not accessible to all players.
- **Non-Stochastic Games**: games that have deterministic state transitions and do not include RNG mechanics
- **Stochastic Games**: games with RNG mechanics

Note that stochastic/non-stochastic and perfect/imperfect information games are not mutually exclusive.

The focus of this work is mainly on hidden information and hidden information games, stochastic or non-stochastic.

2.2.3 Imperfect Information in Research

One of the main frustrations of this Researcher when beginning their Thesis, has been the following question:

"Why is this a game worthy of being studied?" — K. Matsuzaki

This is somewhat surprisingly not an easy question to answer. The requirement to studying an Imperfect Information Game cannot simply be that it has Imperfect Information, otherwise coinflips would have entire papers dedicated to it (Although it might, even if not necessarily for its value as a game to be studied.)

Looking at other research done in the field, we have found three main reasons as to why a specific game was chosen to be studied:

- Popularity
- Precedence
- Difficulty

We will provide more context on these reasons in the following subsections

Popularity

"Poker is also arguably the most popular card game in the world, with more than 150 million players worldwide." — Bowling et al. [6]

The name is somewhat self-explanatory. Games that are more popular tend to be studied more. However, popularity by itself is not a very scientific measurement. The number of people who know the game doesn't give any information about why it is a good game to study. So why do reaserchers care so much about game popularity? There can be many different causes for this:

- Popularity beggets interest: If a study focuses on a popular game, there is automatically a bigger pool of people that might hear from it and take interest in it. Essentially the goal might be to piggy back on a game's popularity.
- Bigger pool of research interest: Statistically a game that has more players, or even just more people that have an interest in the game, will make it more likely that one of them has the means and interest in studying the game on a deeper level.
- Human Data: More players obviously creates more human data on how to play the game. This can greatly help study a game.
- Game Quality: While popularity is not necessarily a guarantee that a game is good or interesting to play, it is safe to assume that better games will generally be more popular. A game being good or interesting also will give it a better chance to be a target of interest for studies.
- Popularity bias: Conversely, it can also be true that people assume a game being popular must mean that it is good. Again, one would expect to find at least some level of correlation between the two but it is not necessarily true. In any case this is a possible false equivalency that people might make.
- Financial Incentive: The popularity of a game has an inherent financial incentive attached to it (at least in modern video games). Because of this it makes sense that companies attached to the game in some manner would promote more studies of the game as a way

to bring attention to it and/or to make it more interesting to play as a way to make more sales or retain players longer.

Precedence

"In fact, the foundational papers on game theory used poker to illustrate their concepts..." — Brown and Sandholm [10]

Another often cited reason is simply that the game has already been studied by others in the field. While it is a bit more understandable to build upon the research of others, it is not an assessment of the game itself. There are multiple reasons why Precedence might be considered important, a lot of them are at least partially similar to the ones given in for popularity:

- Established Research: Having established research can be useful in multiple aspects. It allows to build upon existing knowledge and can help establish research gaps among other things.
- Established Reputation: Having others studying the game before will have increased the interest for it in general.
- Guidance: Studying something entirely new comes with the challenge of being the only person or group one can refer to. Having other research existing allows to refer to other works and enables one to seek guidance from people who are already knowledgeable on the domain.
- Researcher Pool: Having research preciously done on a game increases the pool of researchers having experience with it. In turn this makes said researchers more likely to further work on said game.
- Precedence beggets interest: Reading other's research can be a great source of curiosity and inspiration which can motivate others to do their own research on the same subject.
- Precedence bias: this would essentially be the same as the Popularity bias explained earlier.

- Available resources: The great thing about research is that many teams share their results and tools they used to achieve said results. This makes it all the more easy to get into similar research for other researchers in the future by lowering hte barrier of entry.

Difficulty

The difficulty of a game is also a popular reason why a game is said to be worthy of study. Most of the time, this difficulty is given in the form of state space complexity or action space complexity estimations. This is arguably the most scientific out of the measures presented so far. These measures however, are not specific to Imperfect Information Games, they are the same measures commonly seen so far with Perfect Information Games. Altogether, it is, at the very least, surprising that IIGs, who are considered to pose different challenges than PIGs have not given rise to measures associated with these different challenges.

2.2.4 The issue

The problem with all reasons presented above, is that neither of them relates specifically to Imperfect Information.

The interest of studying IIGs is that they offer different challenges than PIGs. Simply calculating the state-space size is not representative of the challenge of a given IIGs. One would expect that ways of representing these challenges would have emerged with the research done on these specific games but this does not seem to have been the case so far.

In general, a lot of Machine Learning applied to games operates on a “Might makes right” basis. Any increase in performance on any game is considered an interesting result and whatever changes in algorithm and/or methodology are more likely to be cited and used in future research. Obviously this is not inherently a bad process however it does create a situation where research is focused on end results and somewhat neglects obtaining a full understanding of the problem being optimized for.

2.3 Analyzing Hidden Information

As explained in previous sections, the vocabulary to talk about and characterize the variety of Imperfect Information Games is somewhat lackluster. Here we introduce the core concepts we created and attempt to put forward with this work.

2.3.1 Hidden Information Visualization

In IIGs, a lot of information can be gleamed from good observation of the game states one has access to and the other player actions one witnesses. Being able to interpret these signs is often a key to playing well in IIGs. Many advanced players can deduce hidden information from clues in games. A professional Poker player might be able to get a good read on their opponent's cards by analyzing how they behave given a certain situation to the point that they might be able to outright guess what hand they have.

This kind of behavior is a form of informed estimation of hidden information or Imperfect Information Visualization. Some games can give context clues about hidden information from the gameplay. Blackjack, for example, allows a player to get some level of Imperfect Information Visualization by counting cards that have come out of the deck.

On the other hand, some games make it difficult to get any representation of hidden information. Going back to the hidden coin toss game from the above subsection, in such a case there is no way to get a better Visualization of the hidden coin and a player would thus be stuck at a 50/50 guess rate since the game offers no way to deduce what side the coin landed on.

2.3.2 Hidden Information Impact

The first measure we introduce is the Impact of Imperfect Information. This concept aims to create a representation of how much the Imperfect Information aspect of a game has an influence on a player's ability to play it to an optimal level. The level of complexity of a game does not inherently give any representation of how much of its complexity comes from having Imperfect Information present in it. With Impact we aim to create the ability to represent how much of a games complexity comes from the Imperfect Information aspect of it.

Practically a game where a player can play successfully while ignoring hidden information

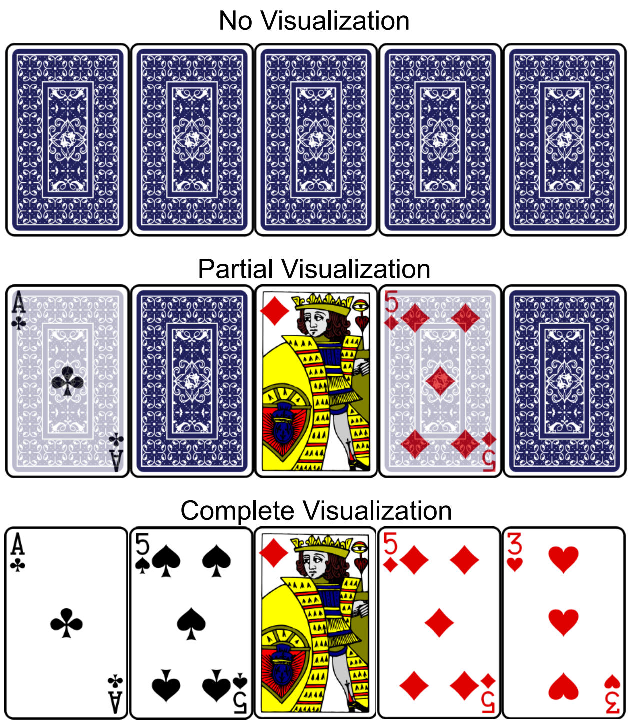


Figure 2.1: Different levels of visualization

has low Imperfect Information Impact. Inversely, a game where any effective strategy necessarily accounts for hidden information has high Imperfect Information Impact.

An extreme example of a low impact would be a chess game with added hidden colors to pieces but no other modification to game rules as illustrated in Figure 2.2. In this case, as the rules remain the same other than the colors, the colors don't modify the way a player needs to behave in order to perform optimally. Hence the hidden information would have no impact at all.

An extreme example of a high impact would be a coin toss game where the player has to guess what face a hidden coin landed on as illustrated in Figure 2.3. In this case the hidden information would have maximum impact as it is impossible to win without it. In other words, if a player knew what side the coin was going to be in advance, the entirety of the game's complexity disappears.

To put this in relation to concepts discussed earlier in Subsection 2.2.2, a game that has 100% imperfect information impact has all of its complexity residing in the imperfect informa-

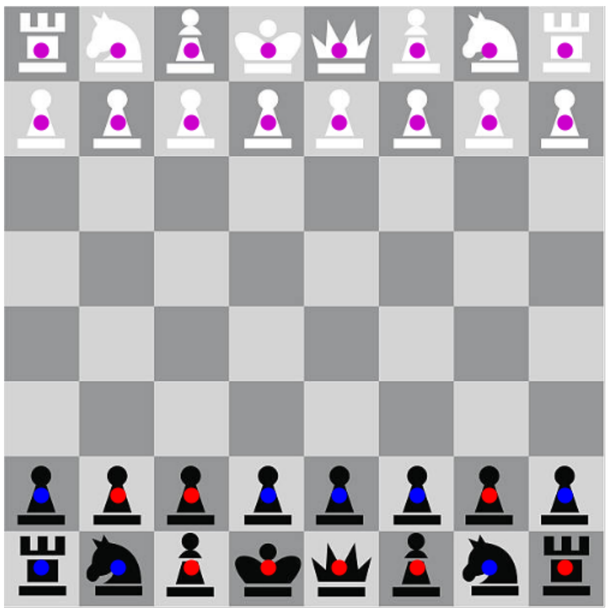


Figure 2.2: No impact game (purple means colors hidden to black player)

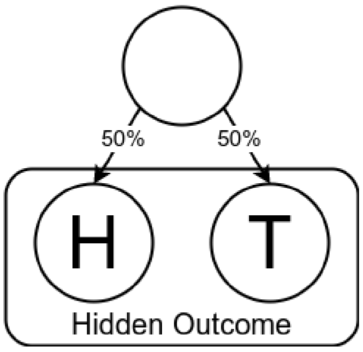


Figure 2.3: Full impact game

tion. If a player has complete knowledge of this imperfect information, the game's outcome becomes fully known and it ceases to be a game as the uncertainty of outcome disappears.

Of course, given a sufficiently complex game, even perfect information does not guarantee a known outcome. In this work we aim to get data about imperfect information impact by comparing player performance with and without giving them access to a "cheated" vision of the imperfect information. Dramatic increases in score with cheated information should point to high impact.

Chapter 3

Algorithms

3.1 Terminology

The terminology used in this work to describe games and algorithms is based on that of Partially Observable Stochastic Games [19] but follows a notation closer to the one presented in the work of Steinberg et al. [32].

A game is played by a group of $\mathcal{N} = \{1, 2, ..., N\}$ **agents**. Each of which can be expressed by an index i. Conventionally all **opponents** of agent i are designed by $-i$. All these agents interact with a **world state** $w \in \mathcal{W}$ using their available actions a from a **joint actions'** space $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N$. The **space of legal actions** for agent i in the world state w is $\mathcal{A}_i(w)$ and the joint actions of all agents is expressed as $a = (a_1, a_2, ..., a_N) \in \mathcal{A}$. For any world state w a player i receives a **reward** indicated by $\mathcal{R}_i(w, a)$. For any world state and action, there is a probability distribution $\mathcal{T}(w, a) = \{(w'_1, p_1), (w'_2, p_2), ..., (w'_x, p_x)\} \in \Delta\mathcal{W}$ In other words, for any state w potentially leading to state w' through the joint action a there is a Markovian state **transition function** that expresses $p(w'|w, a) = \tau(w', w, a)$. However, the transition function can oftentimes be fully ignored. This would by default be the case in any game where actions lead to deterministic transitions but can also be done in some non-deterministic transition games by simply replacing the transition function with a "chance" player. The games considered in this work only having deterministic transition functions, we will however not delve deeper into the subject. Suffice it to say, that the following equations include the transition function for the sake of being thorough, but in a deterministic transition game, the transition function can simply be removed.

Through a succession of states, actions and transitions, a **History/Trajectory** $h = (w^0, a^0, w^1, a^1, ..., w^{t-1}, a^{t-1}, w^t)$ is built. The reward associated to a history is the reward of the second to last world within that history and last action: $\mathcal{R}_i(h) = \mathcal{R}_i(w^t, a^{t-1})$, which is the latest reward obtained by i in this history. Similarly, the set of legal actions of agent i given history $h = (w^0, ..., w^t)$ is $\mathcal{A}_i(h) = \mathcal{A}_i(w^t)$. A history that can lead to another is written as $h' \sqsubset h$. In this situation, h' is a precursor to h. Moreover, if h' can also be equivalent to h we write $h' \sqsubseteq h$. **Terminal histories** are the subset of all histories $\mathcal{Z} \subset \mathcal{H}$ where the game has ended. Formally $\forall h \in \mathcal{H}, h \in \mathcal{Z}$ if $\forall i \in \mathcal{N}, \mathcal{A}_i(h) = \emptyset$. We denote a specific terminal history with z.

As we are dealing with Imperfect Information, a player generally does not have full knowledge of a world state. Thus, upon transitioning to a new world state, a player receives an **Observation** $O_i = \mathcal{O}_i(w, a, w')$. This observation generally includes, but is not restricted to, a partial view of the world, the reward achieved in the new world, and the opponents' actions. An **Infostate** (or **Action-observation history** (AOH)) is the perceived history of a world for player i and is expressed by the sequence of all Observations and actions they have witnessed: $s_i = (O_i^0, a_i^0, O_i^1, a_i^1, ..., O_i^t)$. An Infostate s_i contains all histories that are indistiguishable for i. The set of i's infostates is $\mathcal{I}_i$. Any history h belongs to a unique infostate of agent i and that infostate is expressed with $s_i(h)$. Conversely, the set of histories that are compatible with an infostate s_i are denoted by $\mathcal{H}(s_i)$. As an infostate regroups all indistiguishable histories, it follows that the agent cannot distinguish them based on its legal actions. However, as reward is achieved not only from a state but also from an action, there are cases where a player could get two different rewards from the same infostate and action combination. In games where the reward can be expressed as the value obtained from reaching an infostate $\mathcal{R}(s_i)$ rather than for taking an action from an infostate $\mathcal{R}(s_i, a_i)$, the rewards of an infostate and its histories will also be indistinguishable. Hence:

- Equivalent action space: $\mathcal{A}(s_i) = \mathcal{A}(h) \quad \forall h \in s_i$
- Equivalent rewards: $\mathcal{R}(s_i) = \mathcal{R}(h) \quad \forall h \in s_i$

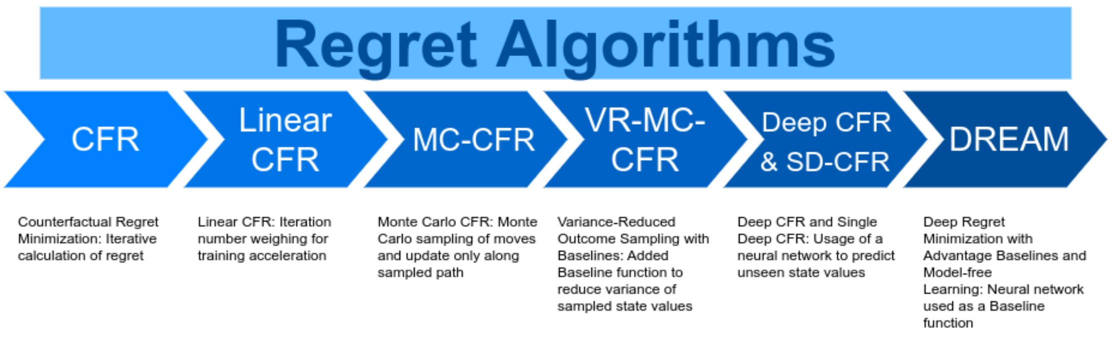
An agent i's strategy is expressed as a **policy** π_i. In essence, a policy is a function that maps a probability to each action a available to agent i from infostate s_i. This can be either expressed as the policy vector $\pi_i(s_i(h))$ of size $|\mathcal{A}_i(s_i(h))|$ or as the single value $\pi_i(s_i(h), a_i)$. The tuple of all agent policies is called the **policy profile** $\pi = (\pi_1, \pi_2, ..., \pi_N)$ and the policy profile of all agents excluding i is π_{-i}. The global policy given a history h then becomes $\pi(h) = (\pi_1(s_1(h)), \pi_2(s_2(h)), ..., \pi_N(s_N(h)))$. Once again, as an infostate should contain all histories indistiguishable for an agent, we can write $\pi_i(h) = \pi_i(s_i) = \pi_i(s_i(h))$. The global policy π for a given joint action a can be calculated as $\pi(h, a) = \prod_{i \in \mathcal{N}} \pi_i(s_i(h), a_i)$ The transition function can also be redefined using policy. In this case the probability that a history h' will be the outcome of an action a taken by player i with their opponents following strategy profile π_i is $\tau(h', h, a_i, \pi_{-i})$. This writing of the transition function is a shorthand for:

$$\tau(h', h, a_i, \pi_{-i}) = \sum_{\forall a_{-i} \in \mathcal{A}_{-i}(h)} \pi_{-i}(h, a_{-i}) \cdot \tau(h', h, a_i \cup a_{-i}) \tag{3.1}$$

Given the policy profile, one can calculate the expected sum of rewards that any agent might get. This is also called the **expected value** (EV). Given a policy profile π and a history h, the expected value of an agent i is $v_i^\pi(h)$ which, through undistinguishability, is equivalent to $v_i^\pi(s_i)$. The tuple of all player EVs for an entire game played under policy profile π is written $v(\pi) = (v_1(\pi), v_2(\pi), ..., v_N(\pi))$. Moreover, we also need to express the EV of an action a given a policy π and a history h (or infostate s_i) for a given agent: $v_i^\pi(h, a_i)$. This is useful to express the EV given a case where an agent follows the policy but chooses to deviate from it using the given action at the given history.

A **Nash equilibrium** represents a point where no agent can hope to achieve a higher EV. It is a policy profile where an agent changing their policy could never result in their EV becoming higher. A policy profile is a Nash equilibrium, π^* if $\forall i, v_i(\pi^*) = \max_{\pi_i} v_i(\pi_i, \pi^*_{-i})$. A Nash equilibrium minimizes the exploitability, $e(\pi) = \sum_{i \in \mathcal{N}} \max_{\pi'_i} v_i(\pi_i, \pi^*_{-i})$, of a strategy profile.

Finally, we also need to introduce the concept of **reach**, $x^\pi(h)$. Reach is the cumulation of probabilities associated to random events on the path to h. Given a history $h = (w^0, a^0, w^1, a^1, ..., w^t)$, its reach is calculated as: $x^\pi(h) = \prod_{h^n \sqsubset h, i \in \mathcal{N}} \pi_i(h^n, a_i)\tau(h^{n+1}, h^n, a_i)$. An agent i's reach is the probability for all its actions to have lead to a history h. Formally, $x_i^\pi(h) = \prod_{h^n \sqsubset h} \pi_i(h^n, a_i)$. In this case the reach of



Counterfactual Regret Minimization: Iterative calculation of regret

Linear CFR: Iteration number weighing for training acceleration

Monte Carlo CFR: Monte Carlo sampling of moves and update only along sampled path

Variance-Reduced Outcome Sampling with Baselines: Added Baseline function to reduce variance of sampled state values

Deep CFR and Single Deep CFR: Usage of a neural network to predict unseen state values

Deep Regret Minimization with Advantage Baselines and Model-free Learning: Neural network used as a Baseline function

Figure 3.1: Regret algorithms evolution

a specific infostate is not equivalent to the reach of a history for a given player. We define it as follows: $x_i^\pi(s_i) = \sum_{\forall h \in \mathcal{H}(s_i)} \prod_{h^n \sqsubset h} \pi_i(h^n, a_i)$. As multiple histories can be part of the same infostate, multiple probabilistic paths can lead to it. If an agent always plays to reach a specific infostate, consideration of his policy can be removed. This is the **external reach** $x_{-i}^\pi(s_i) = \sum_{h \in \mathcal{H}(s_i)} \prod_{h^n \sqsubset h, j \in \mathcal{N} \setminus \{i\}} \pi_j(h^n, a_j)\tau(h^{n+1}, h^n, a_j)$ If $x_i^\pi(s_i) > 0$ then external reach is also $\sum_{h \in \mathcal{H}(s_i)} \frac{x^\pi(h)}{x_i^\pi(s_i)}$

3.2 Deep Regret Minimization with Advantage Baselines and Model-free Learning

Deep Regret Minimization with Advantage Baselines and Model-free Learning (DREAM) [32] is an algorithm included in the family of regret-based algorithms. In the following subsections we will present the foundations of some of the most prominent regret-based algorithm (briefly explained in Figure 3.1) and finish by explaining DREAM itself, similarly to the way it was presented in its original work.

3.2.1 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) [7, 25] is the most basic form of the family of Regret based algorithms. CFR iteratively calculates how much value an agent would gain from taking a specific action rather than playing to its previous strategy. It uses these values to calculate a new strategy profile π^t each iteration t. Combining all strategies over all iterations

T generates a global policy.

The core concept of CFR is regret. First, the **instantaneous regret** is defined as follows:

$$r_i^t(s_i, a_i) = x_{-i}^{\pi^t}(s_i)(v_i^{\pi^t}(s_i, a_i) - v_i^{\pi^t}(s_i)) \tag{3.2}$$

This formulation of regret expresses the added value an agent i could have won at time t if they had deviated from the current policy profile π^t by taking action a_i at infostate s_i. This difference in potential value is pondered by the likelihood of reaching said infostate s_i. Intuitively, the instantaneous regret is a quantifiable value of how much a player would regret not having taken a different action at a key point in the game. Most likely, any reader will have at least once in their life played a game and later realized that, had they done something differently at a key point, they could have won the game; that is the essence of regret.

From instantaneous regret, we can define the **average regret**:

$$R_i^T(s_i, a_i) = \frac{\sum_{t=1}^T r_i^t(s_i, a_i)}{T} \tag{3.3}$$

Again, this value can make sense intuitively. It can happen that an action would happen to be very beneficial in a given game where the right circumstances line up. This does not necessarily mean that the same action would always be beneficial under any circumstances; a veteran player would take their experience in account and realize that this special circumstance could have been a fluke.

Considering that a negative regret value essentially means that an agent is “thankful” that they did not take a specific action, we might not want to consider changing the policy to use this action in further iterations. To do so, it is useful to define the positive regret: $R_{i+}^t(s_i, a_i) = \max\{0, R_i^t(s_i, a_i)\}$

Given these regrets, one can create a policy for a following iteration $t+1$ that tries to exploit the possible increase in value. This is called **regret matching**:

$$\pi_i^{t+1}(s_i, a_i) = \begin{cases} \frac{R_{i+}^t(s_i,a_i)}{\sum_{a' \epsilon \mathcal{A}_i(s_i)} R_{i+}^t(s_i,a_i)} & \text{if} \sum_{a' \epsilon \mathcal{A}_i(s_i)} R_{i+}^t(s_i, a_i) > 0 \\ \\ \text{zero-policy} & \text{otherwise} \end{cases} \tag{3.4}$$

The zero policy is the strategy that a player defaults to when the sum of positive regrets is zero: $\sum_{a_i \in \mathcal{A}(s_i)} R_{i+}^t(s_i, a_i) = 0$. In [8] two zero policies are given: random selection, equation 3.5, where an action is sampled randomly, and greedy selection, equation 3.6, where the action with the highest regret is sampled. The initial at $t = 0$ is always the random selection.

$$\pi_{random}(s_i, a_i) = \frac{1}{|\mathcal{A}_i(s_i)|} \tag{3.5}$$

$$\pi_{greed}(s_i, a_i) = \begin{cases} 1 \text{ if } R_i^t(s_i, a_i) = \max_{a_i \in \mathcal{A}_i(s_i)} R_i^t(s_i, a_i) \\ \\ 0 \text{ else} \end{cases} \tag{3.6}$$

However, in this work, we used an epsilon greedy zero policy when applicable, equation 3.7, which empirically led to more stable results in our experiments.

$$\pi_{epsilon}(s_i, a_i) = (1 - \epsilon) * \pi_{greed}(s_i, a_i) + \epsilon * \pi_{random}(s_i, a_i) \tag{3.7}$$

The final policy or **average policy** is:

$$\bar{\pi}_i^T(s_i, a_i) = \frac{\sum_{t=1}^T x_i^{\pi^t}(s_i) \pi_i^t(s_i, a_i)}{\sum_{t=1}^T x_i^{\pi^t}(s_i)} \tag{3.8}$$

For any infostate, each iteration's policy is pondered by the chance of that specific policy to lead to the given infostate. In other words, when reaching a given game state through a specific strategy, one should likely keep using the same strategy instead of using another one that might not be adapted to deal with the reached state.

The main disadvantage of CFR is that it requires traversing the entire game tree. Each iteration regrets are updated for every possible infostate in the game by using the current iteration's policy. These regrets are used to create the next iteration's policy. This process creates an average policy over all iterations that converges to a Nash equilibrium. Obviously exploring the entire game tree each iteration is infeasible for larger games. Some of the following regret algorithms deal with this issue.

3.2.2 Linear CFR

Linear CFR [9] weighs each iteration's regret update by the iteration number to accelerate the training process. Hence previous formulas presented in Eq. 3.3 and Eq. 3.8 are modified as follows:

$$R_i^T(s_i, a_i) = \sum_{t=1}^{T} (t r_i^t(s_i, a_i)) \tag{3.9}$$

$$\bar{\pi}_i^T(s_i, a_i) = \frac{\sum_{t=1}^{T} (t x_i^{\pi^t}(s_i) \pi_i^t(s_i, a_i))}{\sum_{t=1}^{T} (t x_i^{\pi^t}(s_i))} \tag{3.10}$$

3.2.3 Monte Carlo CFR

Monte Carlo CFR (MCCFR) [23] allows the regret updates to work without exploring the entire game tree. It uses sampling for a single agent at a time and only updates regrets along the explored agent's trajectories.

Two sampled methodologies are presented in the original MCCFR paper:

External-Sampling

A single action is sampled for each non-traverser agent and a single outcome is sampled from $\mathcal{T}$, but every traverser action is explored.

Outcome-Sampling

A single action is sampled for each agent at each decision point, and a single outcome is sampled from $\mathcal{T}$. The traverser policy ξ^t varies from the traverser agent i and others j.

$$\xi_i^t(s_i, a_i) = \epsilon \frac{1}{|\mathcal{A}_i(s_i)|} + (1-\epsilon)\pi_i^t(s_i, a_i) \qquad \xi_j^t(s_j, a_j) = \pi_j^t(s_j, a_j) \tag{3.11}$$

3.2.4 Variance-Reduced Outcome Sampling with Baselines

Variance-Reduced Outcome Sampling with Baselines (VR-MCCFR) [30][15] is a variant of CFR that adds a user given baseline function to reduce the variance of sampled values by estimating them automatically on a given state. The baseline function allows to have a prior estimate of how good a given history is even if said history has never been sampled before.

$$\begin{aligned}\tilde{v}_i^{\pi^t}(s_i(h), a_i|z) &= \tilde{v}_i^{\pi^t}(h, a_i|z) \\ &= \begin{cases} b_i(h, a_i) + \dfrac{\sum_{h'}\{\tau(h', h, a_i, \pi_{-i}^t)\tilde{v}_i^{\pi^t}(h'|z)\} - b_i(h, a_i)}{\xi_i^t(s_i, a_i)} & \text{if } (h, a_i) \sqsubseteq z \\ b_i(h, a_i) & \text{otherwise} \end{cases}\end{aligned} \tag{3.12}$$

$$\tilde{v}_i^{\pi^t}(s_i(h)|z) = \tilde{v}_i^{\pi^t}(h|z) = \begin{cases} \mathcal{R}_i(h) & \text{if } h = z \\ \sum\limits_{a_i \epsilon \mathcal{A}_i(h)} \pi_i^t(h, a_i)\tilde{v}_i^{\pi^t}(h, a_i) & \text{otherwise} \end{cases} \tag{3.13}$$

With b as the history-action baseline; it should be closely correlated with the expected value.

3.2.5 Deep CFR

Deep CFR [8] adds the use of a neural network to predict values of unseen states and then uses the network to sample further in a repeating loop.

During each iteration t a predetermined number of game traversals T are used to generate sampled which in turn are stored in a reservoir buffer B_i^d where i is the traverser agent. The buffered samples are then used to train an average regret prediction network $\hat{D}_i^t$ with approximate advantages as per Equation 3.14.

$$D_i^T(s_i, A_i) = \frac{R_i^T(s_i, a_i)}{\sum_{t=1}^{T}(x_{-i}^{\pi^t}(s_i))} \tag{3.14}$$

In Equation 3.14, the Regret $R_i^T(s_i, a_i)$ is divided by $\sum_{t=1}^{T}(x_{-i}^{\pi^t}(s_i))$ to compensate for different likelihoods to reach different infostates.

This network is then used to approximate the policy π_i^{t+1} as shown in Equation 3.15.

$$\pi_i^{t+1}(s_i, a_i) = \frac{\hat{D}_{i+}^t(s_i, a_i)}{\sum_{a' \epsilon \mathcal{A}_i(s_i)} \hat{D}_{i+}^t(s_i, a_i')} \tag{3.15}$$

Where $\hat{D}_{i+}^t(s_i, a_i) = \max\{0, \hat{D}_i^t(s_i, a_i)\}$ is the positive advantage, similarly to positive regret $R_{i+}^t(s_i, a_i)$.

Again, in this case we use an epsilon-greedy zero policy but the original DREAM used a simple greedy policy.

However, one of the main aspects of CFR being that the average policy converges to a Nash equilibrium, Deep CFR also requires training a second, average, neural network $\hat{S}$ that approximates the average strategy over all iterations. To train this network, sampled policy values from game traversals during training are saved to a second, reservoir, buffer B_i^s.

Single Deep CFR

Alternatively, Single Deep CFR (SD-CFR) stores all value networks from each CFR iteration. It then randomly selects one of these value networks at the beginning of each game and uses it to calculate the policy for the entire duration of that game. This allows to statistically replicate the average policy exactly at the cost of more storage.

3.2.6 DREAM

The main point of DREAM is to combine SD-CFR, outcome sampling and learned baselines. To do so, two networks are used, a network $\hat{D}_i^t(s_i, a_i|\theta_i^t)$ to estimate advantages using trained parameters θ_i^t and a baseline network $\hat{Q}_i^t(s^*(h), a_i|\phi_i^t)$ using trained parameters ϕ_i^t to estimate baselines. Each of these networks has a corresponding buffer B_i^d (reservoir) and B_i^q (circular) respectively from which they learn.

Samples are collected by traversing the game T times with Outcome sampling.

Starting with the baseline, note that it receives $s^* = (s_0(h), s_1(h), ..., s_N(h))$, a combination of all players' infostates, as input. In other words the baseline also has access to hidden information during training. This network learns by minimizing the mean squared error (MSE) to target values sampled from its previous iteration. Its target values are calculated as presented in equation 3.16 and stored in B_i^q.

$$\tilde{q}^t_{i,DREAM}(s_i(h), a_i) = \mathcal{R}_i(h) + \sum_{a'_i \epsilon \mathcal{A}_i} \pi^t(s_i(h'), a'_i)\hat{Q}^t_i(s^*(h'), a'_i) \tag{3.16}$$

Next, to sample advantages, we first need to calculate baseline-adjusted sampled expected values for each sampled action a executed at history h leading to h'. This is done with equations 3.17 and 3.18.

$$\tilde{v}^{\pi^t}_{i,DREAM}(h, a_i|z) = \begin{cases} \hat{Q}^t_i(s^*(h), a_i) - \frac{\tilde{v}^{\pi^t}_{i,DREAM}(h'|z) - \hat{Q}^t_i(s^*(h), a_i)}{\xi^t(s_i, a_i)} & \text{if } a_i = a'_i \\ \hat{Q}^t_i(s^*(h), a_i) & \text{otherwise} \end{cases} \tag{3.17}$$

$$\tilde{v}^{\pi^t}_{i,DREAM}(h|z) = \begin{cases} \mathrm{R}_i(h) & \text{if } h = z \\ \sum_{a_i \epsilon \mathrm{A}_i(h)} \pi^t_i(h, a_i)\tilde{v}^{\pi^t}_{i,DREAM}(h, a_i|z) & \text{otherwise} \end{cases} \tag{3.18}$$

In these equations, we once again have the history/infostate equivalency: $\tilde{v}^{\pi^t}_{i,DREAM}(s_i(h), a_i|z) = \tilde{v}^{\pi^t}_{i,DREAM}(h, a_i|z)$ and $\tilde{v}^{\pi^t}_{i,DREAM}(s_i(h)|z) = \tilde{v}^{\pi^t}_{i,DREAM}(h|z)$.

From these values, we can calculate the sampled immediate advantages

$$\tilde{d}^t_{i,DREAM}(s_i, a_i) = \tilde{v}^{\pi^t}_{i,DREAM}(s_i, a_i) - \tilde{v}^{\pi^t}_{i,DREAM}(s_i) \tag{3.19}$$

As we are using outcome sampling (eq. 3.11), data samples stored in B_i^d are weighted by $\frac{1}{x_i^{\xi^t}(s_i)}$ to compensate for the shift caused in sampling distribution by ϵ.

Finally, to also add Linear CFR to DREAM, one can weigh training losses of each sample $\tilde{d}_i^t$ by t.

Average Policy

As with Deep- and SD-CFR, the Nash equilibrium policy in DREAM is the average policy over all iterations. This can be done with either Deep or SD approaches.

Either, action probability vector samples from each iteration are stored in a buffer from which an average policy network is trained once training iterations have been completed. In this case, however, one should corret for sampling bias by weighing samples from iteration t by $\frac{1}{x_{-i}^{\xi}}(h)$.

Or, advantage networks form each iteration are stored on disk and a random one is selected at the beginning of each game. If one trained using Linear-CFR, the probability of selecting a specific network should be proportional to its iteration.

3.3 Monte Carlo

As presented in the book “Reinforcement Learning: An Introduction” [34], there are multiple different Monte Carlo (MC) methods. The one we focus on in this work is on-policy Monte Carlo Control.

3.3.1 On-Policy Monte Carlo Control

This basic version of Monte Carlo has the obvious problem of not adapting well to big state spaces. It requires a table of all seen state, action pairs with associated value and cannot extrapolate to unseen state, action pairs. To solve this, [44, 46] created a variation of MC methods called Deep Monte Carlo (DMC).

Algorithm 1 On-Policy Monte Carlo Control

```
 1: procedure INITIALIZE($Q, Returns$)
 2:     for $s \in \mathcal{S}, a \in \mathcal{A}$ do
 3:         $Q \leftarrow EmptyDictionary$
 4:         $\pi(s) \leftarrow arbitrary$
 5:         $Returns(s, a) \leftarrow$ empty list
 6:     end for
 7: end procedure

 8: procedure REPEAT FOREVER($\pi, Q, Returns, \epsilon$)
 9:     $e \leftarrow$ Generate an episode using $\pi$
10:     for $s, a$ in $e$ do
11:         $R \leftarrow$ return following the first occurrence of $s, a$
12:         Append $R$ to $Returns(s, a)$
13:         $Q(s, a) \leftarrow average(Returns(s, a))$
14:     end for
15:     for $s$ in $e$ do
16:         $a^* \leftarrow \arg\max_a Q(s, a)$
17:         for $a \in \mathcal{A}(s)$ do
18:             $\pi(s, a) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$
19:         end for
20:     end for
21: end procedure
```

3.3.2 Deep Monte Carlo

DMC enhances MC methods with the use of Neural Networks as estimators to replace Q(s,a) value tables. The main DMC training loop follows a basic reinforcement learning architecture where sample taken from a game environment are used to iteratively update a network which, in turn, is used to sample new data points. The sampling process can be subdivided into multiple actor processes who create games samples in parallel. It can be summarized as follows and is illustrated in Figure 3.2.

1. Actor processes receive their Infoset from the game node.

2. Actors evaluate expected rewards for each state, action pair.

3. The best action is selected Based on policy π and sent to the game environment.

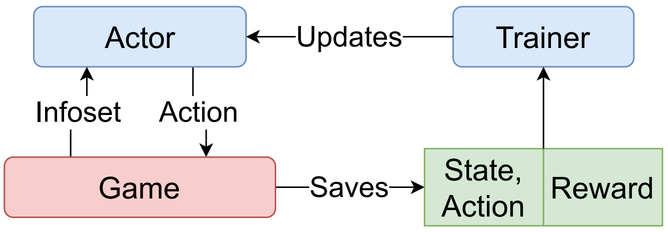


Figure 3.2: DMC training process

4. Sampled state, action, and reward are saved.

5. Samples are used by the Trainer to learn the rewards of each s, a pair.

6. After training, the Trainer updates the Actors. The policy for next iteration is: $\pi(s) \leftarrow \arg\max Q(s, a)$.

Using the on-policy Monte Carlo control algorithm as a reference, we can also describe DMC in pseudo-code format as shown in Algorithm 2. As indicated in the pseudo-code, this process can be parallelized.

DMC with opponent hand estimation

In further research [46], DMC was extended to include opponent modeling in the form of a second network trained to estimate the state of opponents' private information h.

Algorithm 2 Deep Monte Carlo

```
    $V$: Action value network
    $B$: Shared buffer of samples
    $\epsilon$: Exploration probability
    $\psi$: Learning rate
    $M$: Batch size
 1: procedure INITIALIZE
 2:     $B \leftarrow Buffer$
 3:     $V \leftarrow Network$
 4: end procedure

 5: procedure ACT FOREVER($V, B, \epsilon$)
 6:     $t \leftarrow 0$
 7:     $s_t \leftarrow$ Initial State
 8:     while $!s_t.terminal$ do
 9:         $a_t \leftarrow \begin{cases} \arg\max_a V(s_t, a) & p(1-\epsilon) \\ \text{random action} & p(\epsilon) \end{cases}$
10:         $s_{t+1}, r_t \leftarrow \mathcal{T}(s_t, a_t)$        ▷ Where $\mathcal{T}$ is the Transition function.
11:         $t = t + 1$
12:     end while
13:     for $turn = t-2, t-3, ..., 0$ do        ▷ Obtain cumulative reward
14:         $r_{turn} \leftarrow r_{turn} + \gamma r_{turn+1}$
15:         $B.append(\{s_{turn}, a_{turn}, r_{turn}\})$
16:     end for
17: end procedure

18: procedure LEARN FOREVER($V, B, \psi, M$)
19:     if $|B| \geq M$ then
20:         $batch \leftarrow B.pop(M)$
21:         $loss \leftarrow MSE(V(batch[s], batch[a]), batch[r])$        ▷ $s, a, r$: info states, actions, rewards
22:         $V.update(loss, \psi)$
23:     end if
24: end procedure
```

Algorithm 3 Deep Monte Carlo with Estimation

```
    $V$: Action value network
    $E$: Hidden information estimation network
    $B$: Shared buffer of samples
    $\epsilon$: Exploration probability
    $\psi$: Learning rate
    $M$: Batch size
 1: procedure INITIALIZE
 2:     $B \leftarrow EmptyBuffer$
 3:     $V \leftarrow ActionNetwork$
 4:     $E \leftarrow EstimationNetwork$
 5: end procedure

 6: procedure ACT FOREVER($V, E, B, \epsilon$)
 7:     $t \leftarrow 0$
 8:     $s_t, h_t \leftarrow$ Initial State                    ▷ Where $h_t$ is hidden information
 9:     while $!s_t.terminal$ do
10:         $e_t \leftarrow E(s_t)$                ▷ Where $e$ is the hidden information estimation
11:         $a_t \leftarrow \begin{cases} \arg\max_a V(s_t, a, e_t) & p(1-\epsilon) \\ \text{random action} & p(\epsilon) \end{cases}$
12:         $s_{t+1}, r_t, h_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$        ▷ Where $\mathcal{T}$ is the Transition function.
13:         $t = t + 1$
14:     end while
15:     for $turn = t-2, t-3, ..., 0$ do                    ▷ Obtain cumulative reward
16:         $r_{turn} \leftarrow r_{turn} + \gamma r_{turn+1}$
17:         $B.append(\{s_{turn}, a_{turn}, r_{turn}, h_{turn}\})$
18:     end for
19: end procedure

20: procedure LEARN FOREVER($V, P, B, \psi, M$)
21:     if $|B| \geq M$ then
22:         $batch \leftarrow B.pop(M)$
23:         $estimations \leftarrow E(batch[s])$
24:         $eloss \leftarrow CrossEntropy(estimations, batch[h])$
25:         $vloss \leftarrow MSE(V(batch[s], batch[a], estimations), batch[r])$
26:         $E.update(eloss, \psi)$
27:         $V.update(vloss, \psi)$
28:     end if
29: end procedure
```

Chapter 4

Geister

Geister is a German board game which is somewhat reminiscent of a simplified version of Chess. However, contrary to chess, Geister is an Imperfect Information board game as some information about pieces are hidden from players.

Notably, it won the 1982 “Game of the Year” [1] and 1986 “Årets Spel Best Family Game” [2] awards. It is has also been a popular Research game in Japan [12, 13, 14, 35] and even has a competition dedicated to it in the form of the Geister AI Tournament [3].

In the game, each player controls a series of ghosts, some of which are good and some of which are evil. The key hidden information is that the opponent does not know which of a player’s ghosts are good and evil. Each player has to use this hidden information to their advantage in order to win the game by escaping the board, capturing the opponent’s good ghosts or losing their own evil ghosts.

4.1 Rules

Geister is played on a square board with six rows and six columns as shown in Fig 4.1. Both players control eight pawns/ghosts. Four of them are red/evil and four are blue/good. Hereafter we refer to them as red ghosts and blue ghosts. For simplicity, if a ghost’s color is unknown we use purple to represent it. A player cannot see the color layout of the opponent’s ghosts.

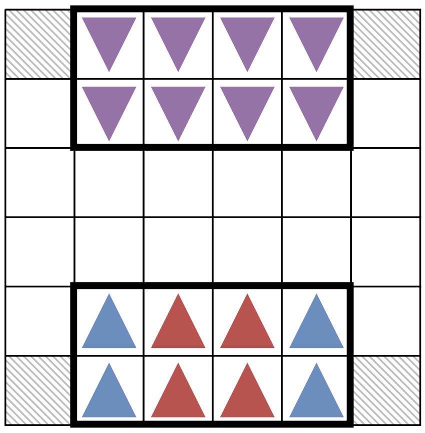


Figure 4.1: Example Geister starting board

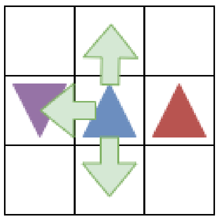


Figure 4.2: Ghost movement

4.1.1 Setup

Sente, the starting player, is chosen randomly. Each player sets their eight ghosts however they wish in a centered 2x4 starting box on their side of the board, as illustrated by the darker rectanbles in Fig 4.1.

4.1.2 Gameplay

Each turn a player moves one of their ghosts by one square in any orthogonal direction as long as the destination square is not occupied by one of their ghosts, as shown in Figure 4.2. If a ghost moves to an opponent's ghost's square, it captures the opponent's ghost. The captured ghost is taken by the capturing player and its color revealed. Sente starts and the two players alternate turns until the end of the game.

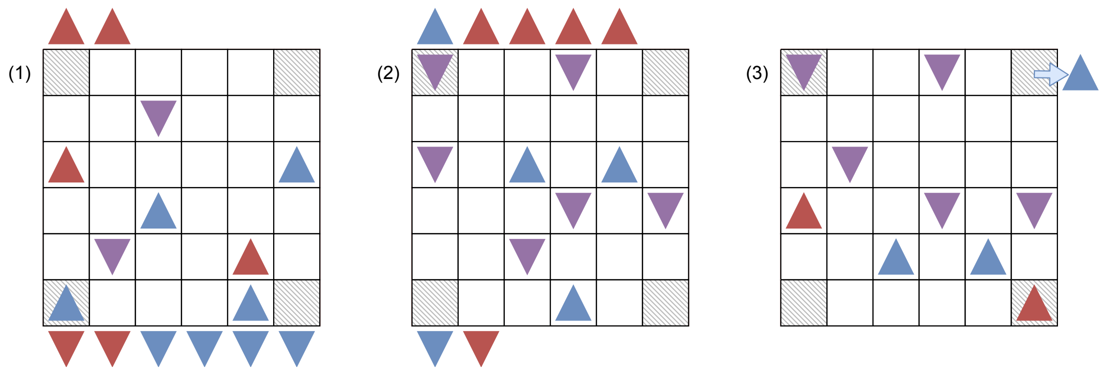


Figure 4.3: Geister win conditions

4.1.3 Win Conditions

The game ends when a player fulfills one of the following win conditions also illustrated in Figure 4.3.

1. Blue capture: A player captures all opposing blue ghosts.
2. Red capture: A player's red ghosts are all captured by the opponent.
3. Escape: A player's blue ghost moves out of the board from an opposing corner.

4.2 Points of Interest

Overall, Geister has multiple interesting aspects to it:

- II board game
- Big action space
- Infinite game duration
- Absence of randomness
- II dependent and independent strategies

On a scientific level, Geister is interesting, firstly because it is an Imperfect Information board game which is one of the rarer sub-types of II Games.

Second, as a board game, its action space is not only fairly big but also somewhat harder to create abstractions for as there are generally no obvious ways to group states together.

Third, Geister is a game that has the potential to continue indefinitely. To study it, one has to manage the possibility that the game can continue forever and that an optimal strategy might consists in infinitely delaying a game.

Fourth, given its rules, there is no actual randomness in Geister. There is no event dictated by some stochastic phenomenon in the game. From the placement of the pieces, to their movement, and what happens when a piece is captured, everything is dictated by the rules or controlled by a player. However there is perceived randomness in the game from the point of view of a given player as they can not see the opposing ghost colors.

Finally, because of its three different victory conditions, Geister is a game that has an win conditions inherently based on the three main ways to deal with hidden information. The Escape win condition, allows a player to ignore the hidden information in the game. As long as they have a good enough strategy, they can ignore opponent ghost colors and focus on making one of their blue ghosts escape from the board. The Red Capture win condition is associated with the weaponization of hidden information. In order to win in this way, a player is required to manipulate the opponent's understanding of hidden information in a way that makes them make unfavorable decisions. In that case, a player does not need to take hidden information into account as long as it can camouflage its own information well enough. The Blue Capture win condition, corresponds to the exploitation of hidden information. To have a good chance to win this way, a player has to achieve a good representation of the hidden state of the game.

4.3 Modified Versions of Geister

Geister lends itself well to the modification of the original game as aspects of it can easily be changed while maintaining its core rules.

Some of the main characteristics of the game are within its board an number of ghosts of each type and all of them can be changed without impacting the gameplay.

These modifiable characteristics are listed bellow and illustrated in Figure 4.4:

- Width

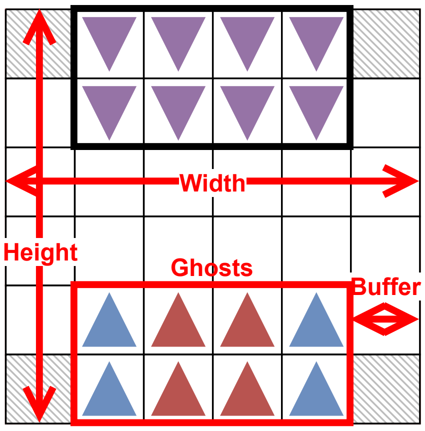


Figure 4.4: Geister Board Characteristics

- Height
- Ghosts
- Buffer

Width and Height are self-evident, the number of ghosts as well as long as one assumes that blue and red ghosts are evenly distributed. Buffer, refers to the number of empty columns left on each side of the starting box to put one's ghosts.

Given these four characteristics, the size of the starting box is automatically determined by simple rules when placing ghosts during setup. First, the starting box is centered horizontally and its width is equal to the width of the board minus twice the required buffer space. Second, a player has to fill the row of the starting box closest to them when placing their ghosts. Once a starting box row is full they move on to the next one. Third, if there are not enough ghosts left to fill the last row, a player can place them freely on any of the valid spaces of that rows.

Table. 4.1 presents all versions of Geister used in this work.

4.4 Experiment

In this work, Geister served as the first case study for the analysis of Imperfect Information in games.

Table 4.1: Geister variants

Board Identifier	**Board width**	**Board height**	**Ghosts**	**Buffer columns**
x2y3g2b0	2	3	2	0
x2y4g2b0	2	4	2	0
x4y2g2b0	4	2	2	0
x3y3g2b0	3	3	2[a]	0
x3y3g3Rb0	3	3	3[b]	0
x3y3g3Bb0	3	3	3[c]	0
x4y3g2b1	4	3	2	1
x4y3g4b0	4	3	4	0
x4y4g2b1	4	4	2	1
x4y4g4b0	4	4	4	0
x4y4g4b1	4	4	4	1
x6y5p8b1	6	5	8	1
x6y6p2b2	6	6	2	2
x6y6p4b1	6	6	4	1
x6y6p6b0	6	6	6	0
x6y6g8b1[d]	6	6	8	1

[a]Middle column left empty
[b]Two Red Ghosts
[c]Two Blue Ghosts
[d]Default Geister

Due to the stochastic nature and complexity of most IIGs it is not possible to get exact models for optimal strategies. To compensate for this, we focused on observing trained agent behaviors to gather data about the attributes introduced earlier in Chapter 2.

We performed a simple experiment to illustrate different emerging behaviors in players and how they influence the gameplay.

Our agents are trained using the DREAM algorithm detailed in Chapter 3.

4.4.1 Infoset History Length

CFR algorithms normally have access to the full infoset of a game. However, using only the latest turn's game state has been shown to work and results in players deemed acceptable [12].

Table 4.2: Board encoding

Encoding layer	Cell information
1	Player’s blue ghosts
2	Player’s red ghosts
3	Opponent’s ghosts

Table 4.3: Meta encoding

Encoding dimension	Meta Information
1	Captured blue pieces
2	Captured red pieces
3	Current turn
4	Is next player

4.4.2 Encoding

A simple encoding of any board with three layers of matrices allows to describe the board state perceived by each player. One layer accounts for the position of a player’s blue ghosts, another accounts for the same player’s red ghosts, and the last one tracks all opponent ghost positions. This is illustrated in Table 4.2.

However, Geister has other attributes that do not reside on the board. These meta-attributes are the number of blue ghosts a player has captured, the number of red ghosts they captured, the turn number and a boolean value indicating if it is that player’s turn to move. We encode this information in a separate, four dimensional, vector detailed in Table 4.3.

4.4.3 Networks

As a board game, Geister lends itself well to the use of convolutional networks. However the meta-attributes are not well suited to convolution without some adaption. In [12], meta-information is inserted into the board encoding allowing for a pure convolutional network. In this experiment, we keep the board and meta information separate as it allows to keep the “meta network head” the same across all Geister variants. The board head is a series of convolution layers and the meta head uses fully connected layers as illustrated in Fig 4.5. We then flatten the convolutional layer outputs, concatenate them with the fully connected layer ones and feed them to a new set of fully connected layers, which we call *combined layers* for ease of distinction with

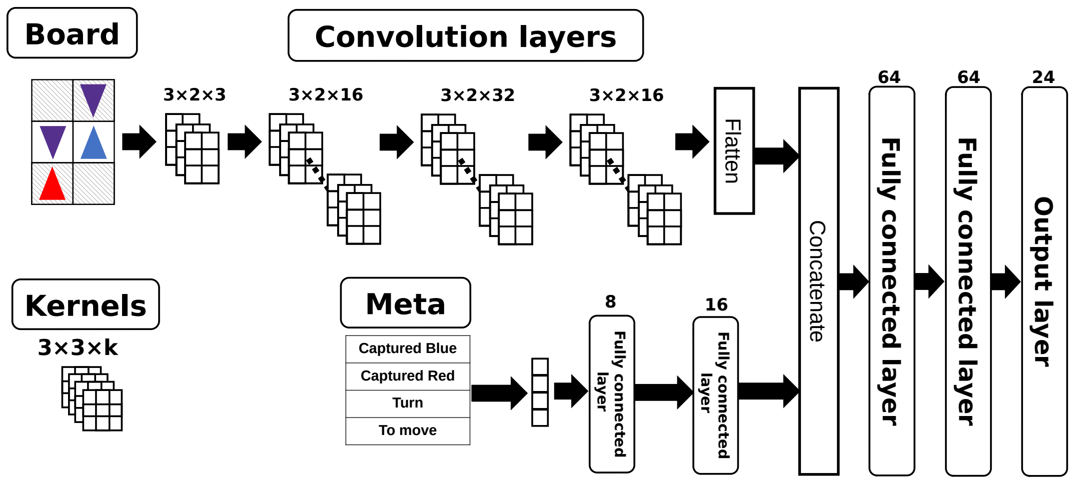


Figure 4.5: Advantage Network with an x2y3g2b0 variant board[38]

the first set of fully connected layers.

All layers in our networks use ReLU. The three convolutional layers use 16, 32 and 16 channels, the fully connected layers have 8 and 16 neurons, and the combined layers have 64 and 64 neurons. The convolution process is always performed using 3x3 kernels and a padding value of one. Figure 4.5 illustrates the network structure using a board example from the x2y3g2b0 game variant.

During the learning process, a total of four networks are trained each iteration. One baseline and one advantage network per player.

4.4.4 Advantage and Baseline

Both the Baseline and Advantage Network follow Fig 4.5's structure. However, the Baseline network has duplicated board and meta heads as it receives both the first- and second-player's game views as inputs. Both flattened convolutional outputs and both fully connected outputs are concatenated and forwarded to the combined layers.

4.4.5 Geister variant scaling

The network structure we use is designed to scale seamlessly with the size of the board. This is done to simplify training our agent on different board sizes.

The parameters we chose allow any board to be used as input for the network and have a constant size through all convolution layers thanks to the padding of one.

The fully connected layers and combined layers of the network mostly do not scale with board size excepting the output layer. The final network output contains $4 * boardsquares$ neurons for each possible cardinal direction in each position. Border squares are also given four corresponding neurons even though some of the corresponding directions are not possible to move into. Impossible moves are filtered out from the output afterwards.

4.4.6 Experimental Parameters

Our training was run on a series of machines equipped with an Intel(R) Core(TM) i7-9800X CPU and two GeForce RTX 2080 Ti GPUs. We used python 3.9.7, pytorch 1.10.0 Cuda driver version 510.73.05 and cuda toolkit version 11.6.

Through prior trials, we tested a variety of different hyperparameters on the simplest Geister variant used in our experiments, x2y3g2b0 as it allowed for the fastest tuning. We used this to scale hyper parameters up until losses were more stable while still having a reasonable experiment duration.

The batch size was selected to match the GPU cuda core count which is 4352 for the RTX 2080 Ti. The capacity for both Reservoir and Circular buffers is set to 1280000 which empirically provided a reasonable buffer size while not making experiments last too long.

The training is done for 1000 iterations. For each iteration the game was traversed 4096 times using 4 samplers. Alternating iterations were used to train alternating players.

Both Baseline and Advantage networks were tuned using the Adam optimizer with a learning rate of 0.001 and a mean square error Loss function. To have the sampled losses of later iterations be more relevant, they were weighted by the iteration they were sampled at.

Every ten training iterations, our agent is confronted to a random player on each Geister variant. They play 100 games against each other with randomly generated starting color disposals. To compensate for any overly favorable or unfavorable starting colors, every starting board is played a second time after switching positions.

We repeated everything 5 times using different seeds.

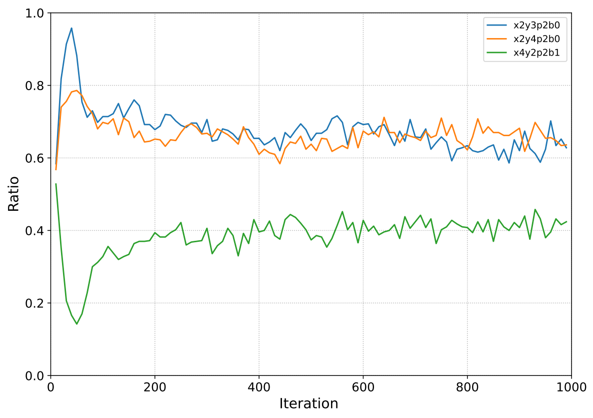


Figure 4.6: Ratio of red ghost moved on first turn by the agent in smaller boards

4.5 Results

Table 4.4 is composed of basic DREAM player statistics we tracked for each session of 100 battles between it and the Random player over all iterations.

4.5.1 First moves

We observe the agent's propensity to move a red or blue ghost with its first action when playing first.

In smaller boards, there is a clear preference for opening with a red move across all training iterations as shown in Figure 4.6. This observation holds true for boards with uneven number of pieces, x3y3g3B/Rb0, Figure 4.7, where the proportion of starting red moves is above the ratio of red ghosts. With increasing board sizes, this color inequality progressively disappears once each player's starting spaces are separated by at least two lines and/or the number of ghosts per player is greater than two. The one case where the preference shifts to moving a blue ghost with the first move is when there is no row between the starting positions of each player such as in boards x4y2p2b1 and x4y4p4b1.

Inspecting the evolution of the "red first" trend through the training, it appears that it is strongest in early iterations but subsides at least partially in later ones. This coincides with higher winning rates for the agent when playing as first-player in these early iterations as visible

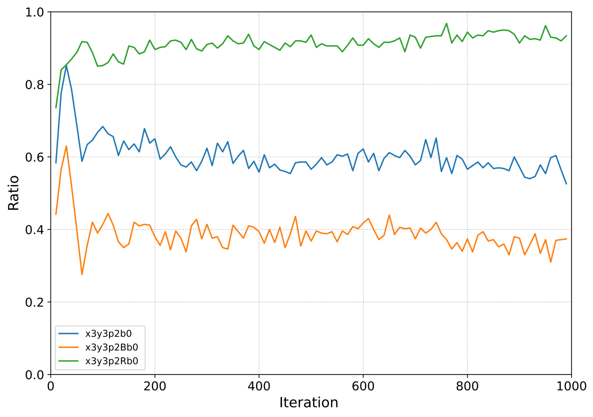


Figure 4.7: Ratio of red ghost moved on first turn by the agent in x3y3 boards

in Figure 4.8.

4.5.2 Win rate

The agent's win rates are presented in Table 4.4. It is clear that there is a discrepancy in win rates when playing as first- or second-player in smaller boards but it disappears in bigger board sizes. In the smallest version of the game, x2y3g2b0, the agent's win rate when second-player is statistically random.

Generally first-/second-player win rate discrepancies occur either in boards with direct contact between both players' pieces at the start of the game or in boards with an uneven number of rows. A higher number of pieces also seems to mitigate this inequality as is can be seen in versions x4y3g2b1 and x4y3g4b0.

The agent achieved it's best absolute performance on the x4y3g4b0 and x4y4g4b0 game versions. The second best performances are in the x3y3g3Bb0 and x3y3g3Rb0 versions.

Originally, we also calculated median values for all items in Table 4.4. However, we later removed it as there was no significant differences between Average and Median values for all cases but one. The x6y6g8b1 variant's average agent wins, first-player agent wins and second-player agent wins were 32.76, 16.58 and 16.17 respectively while the corresponding median values were 22, 11 and 10. This difference is also visible in Figure 4.12 where the maximum and minimum behaviors show much more difference from the average than in other variants.

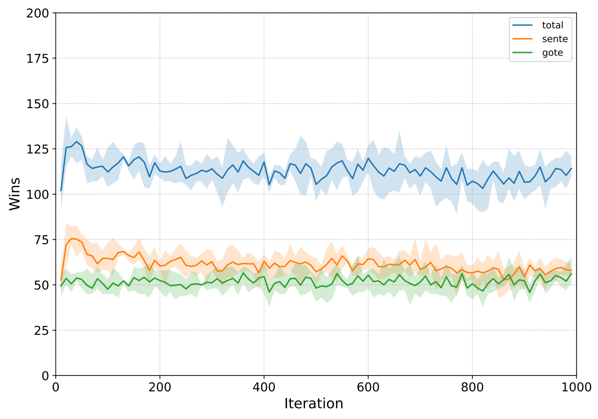


Figure 4.8: Agent wins in x2y3g2b0 variant

This is due to one of the seeded experiment runs for this Geister variant having developed a different agent behavior than the others which shifted the results over all seeds.

4.5.3 Win type

Our variations of Geister all have three possible win conditions per player and the possibility to draw as explained in Section 4.1. Tracking which of these termination conditions was met at the end of a game allows us to have insight into what strategies the agent prefers in each Geister version. Figure. 4.11 shows the agent's win types in full colors. Draws and the random agent's wins are shown in lighter colors. In this sense, labels such as "agent red capture" means the agent won a game by using the "red capture" objective; hence their opponent (random) captured all the agent's red ghosts.

In the x2y3g2b0 version of the game, almost all games finish with a player capturing a ghost as illustrated in Figure 4.11. However, there are visibly more games ends caused by the random player capturing a piece than by the agent capturing one. When the agent plays as first-player, there are a similar number of games ending with the agent capturing it's opponent's blue ghost (agent blue capture) or capturing it's opponent's red ghost(random red capture). However there are more cases where the random player captures the agent's red ghost(Agent red capture) than instances where it captures the agent's blue ghost(Random blue capture). When the agent is second-player, these two pairs of win types are roughly equal.

Table 4.4: Geister variants statistics[a]

Board Identifier	First move Red	Agent wins	Random player wins	Agent wins First-player	Agent wins Second-player
x2y3p2b0	0.67 (0.08)	112.92 (8.36)	87.07 (8.36)	61.28 (6.43)	51.64 (5.21)
x2y4p2b0	0.66 (0.09)	121.58 (7.70)	78.41 (7.70)	60.44 (5.42)	61.14 (5.72)
x3y3p2Bb	0.38 (0.07)	129.00 (8.33)	70.99 (8.33)	65.56 (5.47)	63.43 (5.38)
x3y3p2Rb	0.91 (0.04)	133.40 (8.17)	66.59 (8.17)	68.12 (5.64)	65.28 (5.10)
x3y3p2b0	0.60 (0.08)	119.73 (7.34)	80.26 (7.34)	63.01 (5.21)	56.72 (5.31)
x4y2p2b1	0.38 (0.08)	117.34 (7.15)	82.65 (7.15)	64.51 (5.61)	52.83 (4.89)
x4y3p2b1	0.49 (0.05)	124.76 (7.30)	75.23 (7.30)	66.15 (5.11)	58.61 (5.24)
x4y3p4b0	0.59 (0.07)	146.01 (9.82)	53.98 (9.82)	74.93 (5.80)	71.08 (5.99)
x4y4p2b1	0.59 (0.14)	126.29 (8.21)	73.69 (8.21)	64.37 (5.60)	61.92 (5.41)
x4y4p4b0	0.47 (0.07)	135.84 (13.65)	64.14 (13.65)	68.80 (7.73)	67.04 (7.75)
x4y4p4b1	0.41 (0.06)	119.95 (11.76)	80.02 (11.75)	61.95 (7.41)	58.00 (6.49)
x6y5p8b1	0.50 (0.05)	65.02 (13.68)	52.11 (6.98)	37.96 (10.39)	27.06 (6.33)
x6y6p2b2	0.53 (0.06)	101.20 (8.77)	86.42 (7.47)	50.78 (6.02)	50.41 (5.36)
x6y6p4b1	0.50 (0.04)	54.07 (8.03)	52.28 (6.77)	27.39 (6.15)	26.67 (4.91)
x6y6p6b0	0.50 (0.05)	26.84 (6.57)	23.93 (5.23)	14.22 (6.11)	12.62 (3.96)
x6y6p8b1	0.50 (0.05)	32.76 (30.30)	17.44 (4.27)	16.58 (14.82)	16.17 (16.37)

[a] Average (Standard deviation)

In the slightly bigger x2y4g2b0, the win type distribution is similar to that of the previously discussed board when the agent is the first-player. There is however no significant difference in win type behavior in this instance regardless of if the agent is first- or second-player.

The x4y2g2b1 version of the game is the smallest to show a significant percentage of wins by escape. Escapes are mostly used by the agent when it is the first-player but also have a steady occurrence rate when the agent is the second-player. Regarding the games ending by capture moves, the overall trend is similar to that outlined in the x2y3g2b0 version.

In variant x4y3g2b1, the agent has a similar behavior when first-player than in the previous variant but achieves more escapes overall and has a higher inclination towards performing capture moves.

The x4y3g4b0 variant is the one where the agent achieves its best performance against the random one. Looking at the win types, the main way the agent achieves this performance is by putting a lot of emphasis on escaping the board no matter which position it is playing as.

Variants x4y4g2b1 and x4y4g4b0 have similar behaviors to x4y3g2b1 and x4y3g4b0 respectively even though the number of escapes in both cases is slightly lowered.

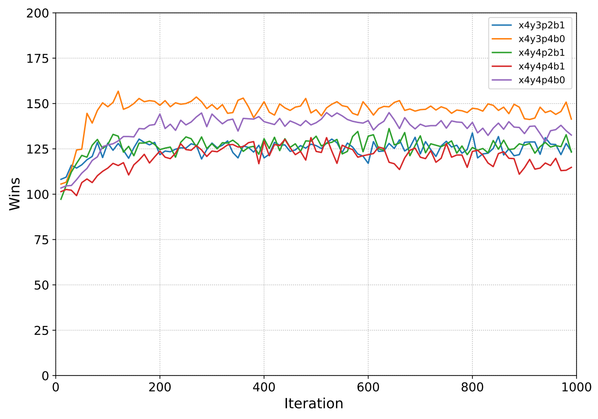


Figure 4.9: Agent wins against random player on Medium boards

Overall, variant x4y4g4b1 also has a varied usage of win conditions. It does have a slightly lower overall win rate than the other two x4y4 board variants but is also one of the variants with the most evenly distributed win types.

The two x3y3 variants of the game with uneven number of ghosts show contrasting behaviors. In Figure 4.11, the x3y3g3Bb0 variant has a high fraction of wins achieved by capturing a blue ghost. The x3y3g3Rb0 variant shows the opposite behavior emerging. The majority of the agent's wins are the result of the random player capturing its red ghost. Variant x3y3g2b0 does not have such an unbalanced favoritism for one strategy even though it does rely most on wins by red captures. The x3y3 variants are some of the smallest boards to have a steady albeit small amount of wins by escaping regardless of the agent playing as first- or second-player.

All variants from x6y5g8b1 and bigger have most games terminating on a draw by reaching the turn limit except when each player only has two ghosts. Excluding that version, the agent never manages more than seventy wins over 200 games on the bigger boards. Most games end in Draws.

With the original version, x6y6g8b1, the vast majority of games end in draws.

In one of our seeds for the game version x6y6g8b1, the agent progressively achieved more wins by escape. This event only happened in one of the five seeds and is visible in the large maximum and minimum performance differences of Figure 4.12.

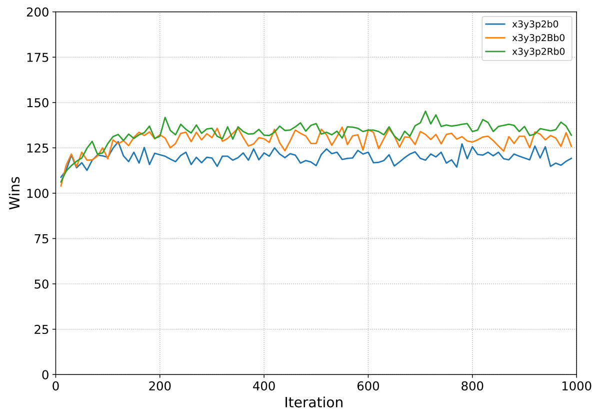


Figure 4.10: Agent wins against random player on x3y3 boards

4.6 Analysis

4.6.1 Red baiting

The observed tendency of the agents to move a red ghost first, especially in smaller games, points to it attempting to bait the other player into capturing their red ghost by making it more easily accessible.

The two cases (x4y2g2b1 and x4y4g4b1) with a blue first move preference also reinforce that the agent is trying to leave its red ghosts in reach. In both of these game versions, players start with their ghosts in direct reach of their opponent. It is logical that the agent player would develop a tendency to move blue ghosts first to get them out of reach of their opponent. This also would leave the red ghosts in range and brings its blue ghosts one step closer to escaping.

The correlation between red first move and win percentage in early iterations in the x2y3g2b0 board reinforces the idea that the agent is using its red ghosts as a bait. It is likely a sign that, early in its training process, the agent finds out that it can get a lot of wins by increasing the likelihood of its red ghost being captured by placing it in front of enemy pieces. This is a good strategy against the random player which has a 50% chance to capture the ghost on such a small board. However the decline of this tendency seemingly indicates the agent learns not to overexploit this strategy, when training against itself, as it can be punished.

The last strong evidence for the red baiting strategy in smaller boards is the high proportion

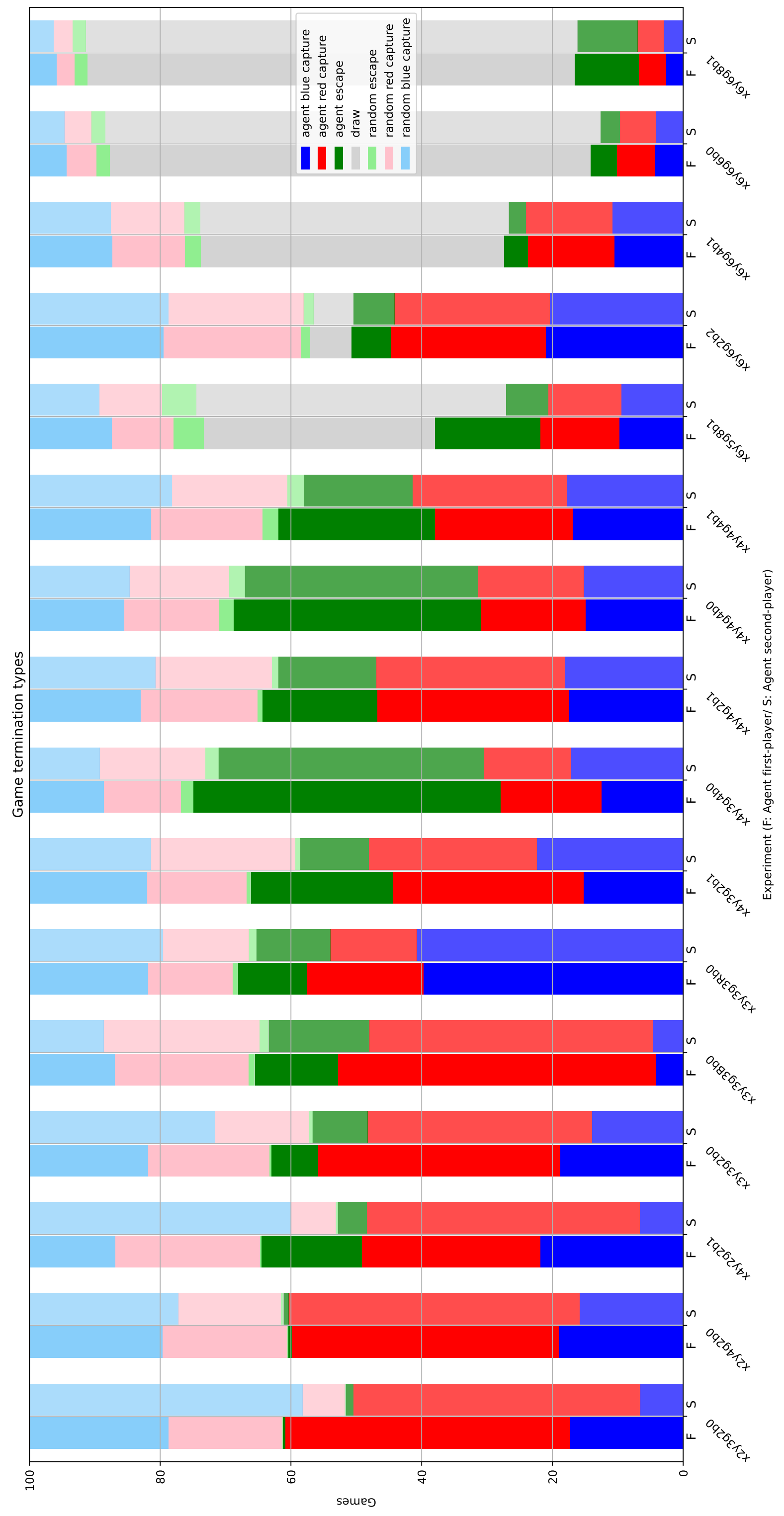


Figure 4.11: Geister win types per experiment

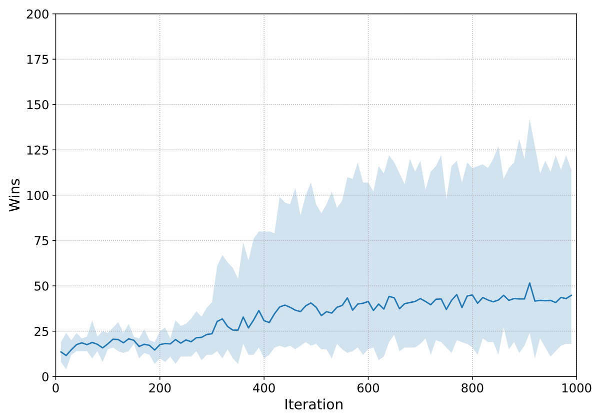


Figure 4.12: Agent wins against random player on x6y6g8b1

of games ending with the random player capturing a ghost when the agent is the second player, proving that the game almost always ends with a capture move from the random player. Thus, *the agent actively avoids capturing its opponent's ghosts* at least during its first turn and *attempts to end the game by baiting with its red ghosts*. This is visible when the agent is the first player too but to a lesser degree.

4.6.2 Asymmetrical gameplay

As seen in subsection 4.5.2, the agent achieves different win rates depending on its position on smaller boards. Moreover, win types show that, while the agent tends to mostly avoid capturing ghosts and favors red baiting, its strategy works best when it is first player. When playing as the second-player, its success rate with red baiting becomes random. In both positions, most games end with the random player capturing one of the agent's ghosts but when in the second-player position there is no advantage to do so. The win rate is essentially random. This is most visible on the x2y3g2b0 board.

These elements point to the gameplay as second-player being more reactive than proactive on smaller boards. I.e. *smaller Geister variants favor the first-player*. The first-player can behave as they wish and the second one must adapt. However, the win rate of the agent not being much higher when playing as first-player also indicates that, *despite the first player advantage, the smallest game variant remains fairly random and hard to strategize efficiently for*.

On bigger boards, indications to the agent favoring red baiting progressively disappear. Moreover, the difference in number of wins when playing as first- or second-player also becomes smaller as the game size increases.

4.6.3 Number of Ghosts

The highest win rates being achieved in variants of Geister that use more than two ghosts per player could indicate that the added number of pieces is an important element in order to allow a player to benefit from a strategy. It should be pointed out that going from 2 to 4 ghosts removes the "sudden death" component of a capture. Multiple ghosts of the same color allow to capture and then adapt rather than risking it all on a single capture move.

Alltogether, it seems that *the number of ghosts has a bigger impact on how efficient a strategy can be than the size of the board.*

Analyzing the most common types of wins in different variants of the game also reinforces how *having a higher number of pieces influences the behavior of a player.* This is especially visible in the x3y3g3Rb0 and x3y3g3Bb0 variants of the game. In the x3y3g3Rb0 variant, the agent visibly becomes much more aggressive with regards to capturing other ghosts. The reason for this is probably that the agent tries to capture a piece as soon as possible knowing it will always survive this first capture and can still adapt its strategy if it does not result in a win. It probably also allows it to be less restricted in its movements by the enemy ghosts. This is reversed in the x3y3g3Bb0 variant: the agent seems to be attempting to bait the other player into capturing its only red ghost.

Going back to the 4 ghosts variants of the game, they show the highest percentage of wins by escaping. In these variants, the agent seems to prefer the strategy least reliant on randomness as escaping is the only win type not reliant on hidden information.

4.6.4 Board rows

As pointed out in subsection 4.5.2, versions of the game that are only different by one row, such as x2y3g2b0 and x2y4g2b0 or x4y3g4b0 and x4y4g4b0, seem to have a lower difference in winrate between the first- and second-player when the number of rows is even. Being able to pass the center of the board first, apparently gives the first-player an unfair advantage.

Chapter 5

DouDizhu

5.1 Base Doudizhu

DouDizhu, a.k.a. “Fighting the Landlord”, is a popular card game in China that requires three participants, with a massive following of hundreds of millions of players. It is a shedding type game where one wins points by being the first in a round to throw away all of one’s cards.

Multiple different versions of DouDizhu exist with rules that can vary depending on what variant you are playing or even what region of China you play in. The base rule set we describe here, is one of the more generally accepted ones and is also the one used in previous works such as [44] and [42].

5.1.1 Rules

The game is played with a standard deck of 52 cards and 2 jokers. Card suits are ignored and card values are ranked from weakest to strongest: **3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A, 2, Black Joker, and Red Joker**.

The players are divided in two teams: the landlord (one player) and the peasants (two players). The peasants win a game/round by having either of them empty their hand before the landlord does. The landlord wins a game/round by emptying their hand before either peasant does.

A game/round of DouDizhu is divided in two main phases, the bidding and the cardplay. In the initial stage of a round, the dealer shuffles and deals 17 cards to each player, leaving three

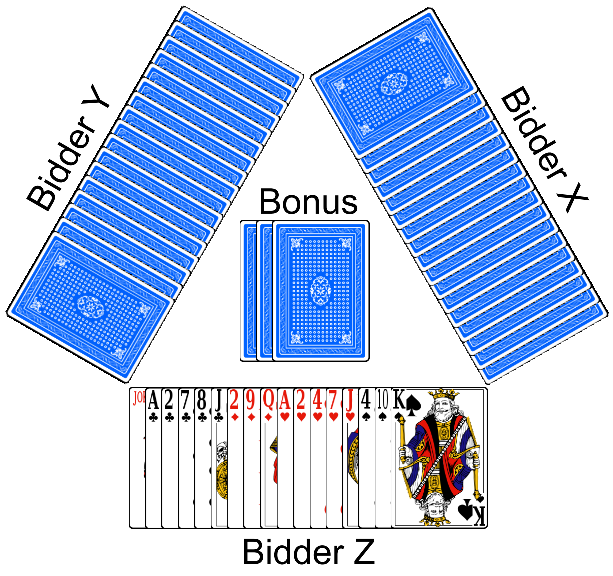


Figure 5.1: Player view during bidding phase

cards face-down which we call bonus cards. This is the start of the bidding phase

Bidding

The bidding phase is a crucial aspect of DouDizhu, as it determines the roles of the players (landlord and peasants) and sets the stage for the remainder of the round.

Starting with the player to the left of the dealer and continuing in a clockwise order, each player has the opportunity to bid for the landlord position once. Each player can bid a value of 1, 2, or 3, provided that their bid value is higher than any previous bid. They can also always choose to not bid and pass. For simplicity, we consider a pass move to be a bid with a value of 0 that is always available.

After all players bid once, the bidding phase finishes with two possible outcomes:

- **All zero bids**: All players get zero points. The Cardplay phase is skipped. Cards are redistributed. A new round starts.
- **At least one non-zero bid**: The player with the highest bid becomes the Landlord, the three bonus cards are revealed to all, the Landlord adds them to their hand. The other players become Peasants. The round moves on to the Cardplay phase

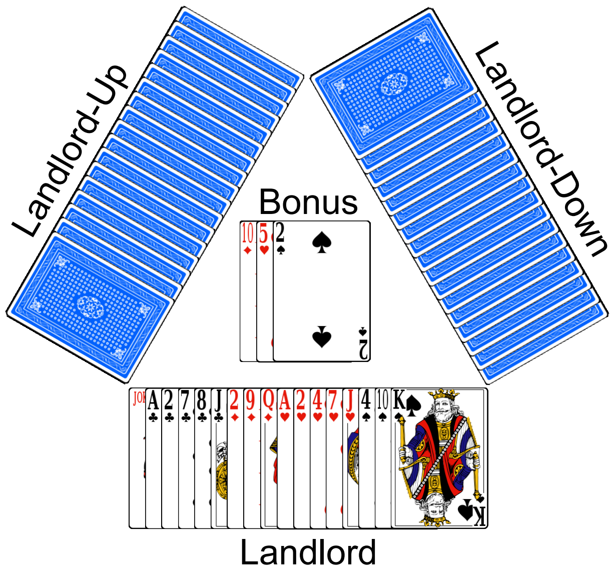


Figure 5.2: Player view at end of bidding phase

Cardplay

After bidding, the cardplay phase starts. Players, starting with the Landlord, take turns playing valid combinations of cards with the goal of being the first to empty their hand. The first combination of cards played initiates a series of combinations during which players are only allowed to play that same combination but with a higher rank than that of the preceding player (with the exception of bombs). The series of combinations ends once two consecutive players pass, at which point the last player to have played a valid combination can choose any combination to initiate a new series.

In DouDizhu an atom of play could arguably be two things. Either an atom consists of one action per player (a.k.a. a full turn) or alternately an atom can be considered to be the point from which a player initiates a new series of combinations until that series finishes with two players passing in succession. For simplicity, we consider the atom to be the first.

The cardplay phase continues until one player has played all of their hand cards. If the Landlord empties their hand, they have won, otherwise the Peasants win.

The total reward for the round is twice the Landlord's bid value doubled for each bomb that has been played. If the Landlord won the round, each peasant has to pay half of the reward to

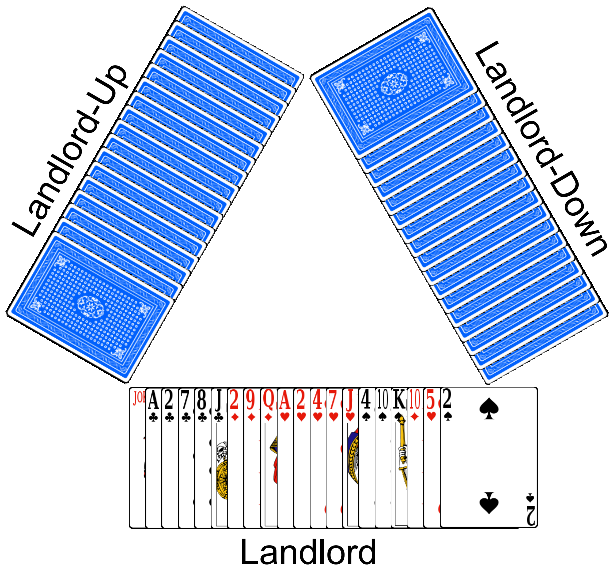


Figure 5.3: Player view at start of cardplay phase

the landlord. Conversely, the landlord has to pay half of the reward to each peasant.

Card Combinations

DouDizhu has multiple possible card combinations that can be used, as described bellow. This variety of possible actions is clearly important to consider during cardplay but also during the bidding phase as they define a hand's strength.

We previously mentioned how all cards are ranked but feel a reminder is warranted here: **Card ranks** (weakest to strongest): 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A, 2, Black Joker, and Red Joker.

Combinations:

- **Solo:** A single card of any rank
- **Pair:** Two cards of same rank
- **Trio:** Three cards of same rank
- **Trio with Solo:** Three cards of same rank with a distinct kicker
- **Trio with Pair:** Three cards of same rank with two cards of same rank

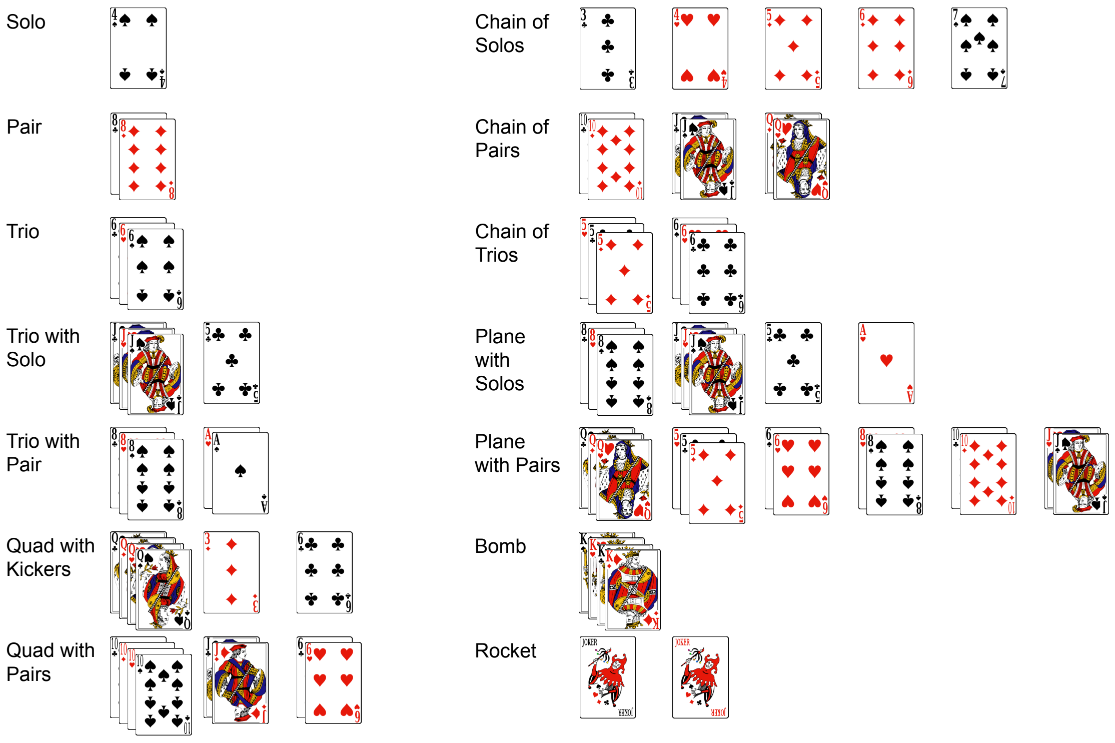


Figure 5.4: Combination examples in DouDizhu

- **Quad with Kickers:** Four cards of same rank with two kickers
- **Quad with Pairs:** Four cards of same rank with two pairs
- **Chain of Solos:** Five or more cards of consecutive rank
- **Chain of Pairs:** Three or more consecutive pairs
- **Chain of Trios:** Two or more consecutive trios
- **Plane with Solos:** Two or more consecutive “Trio with Solo”
- **Plane with Pairs:** Two or more consecutive “Trio with Pair”
- **Bomb:** Four cards of same rank (doubles the round’s reward)
- **Rocket:** Red and Black Joker (doubles the round’s reward)

Please note that 2s and Jockers are not considered consecutive to Aces. Thus they can’t be part of any Chain or Plane unless they are kicker cards.

As discussed in the previous subsection, the bomb and the rocket are a special case. Both of them double the current pot when they are played. Moreover, they can be played over any previous combination regardless what type it is. However, when a bomb is played, another player can still play a bomb of their own over the previous one as long as that bomb's rank is higher than the previous one's.

For simplicity, whenever we refer to bombs, we generally mean bombs and rockets as they essentially act in the same way.

5.1.2 The Bidding Game

For training purposes, bidding is seen as a game that incorporates cardplay as a stochastic process.

The Bidding game starts with two chance nodes. The first chance node corresponds to the random distribution of cards from the deck to all three players and to the hidden three bonus cards. The second chance node determines which of the three players will act first. These chance nodes are then followed by three nodes, one for each player in order of positions around the table, in which each player will have to select a single legal bidding action.

If all players pass they get a reward of 0 and the game ends. Otherwise, a cardplay chance node returns the reward gained by each player. The reward distributions are determined by the set of fixed player strategies.

5.1.3 The Cardplay Game

Similarly to the Bidding phase, the Cardplay phase of Doudizhu can be interpreted as a separate subgame which incorporates the bidding game as a stochastic shuffling process.

This subgame starts with a chance node whose distribution is based on the bidding game. This first chance node distributes cards to each of the three players, 17 for both peasants and 20 for the landlord. Following this, groups of three decision nodes, one for each player, happen in sequence until one decision node leads to a state where a player does not have cards in hand anymore.

We present these two subgames in this way to help introduce the possible separation of both phases of the game. Once each subgame has been expressed as presented above, it becomes clear that the main link between both of them is the chance node whose probability distribution is dictated by the other subgame. Replacing this node with another distribution allows for a complete separation of both subgames. This is the way that most research to this point [24, 42, 44, 46] has used to study Doudizhu by replacing the bidding chance node with a fully random card distribution in the cardplay subgame in order to focus solely on that subgame. This is a pattern we also follow in this work.

5.2 Variants

In the context of this work, we used two other variations of DouDizhu. As there don't seem to be any official English names for them, we will refer to them by approximate translations from their Chinese names: "RunFast" and "NoShuffle".

5.2.1 No Shuffle

No shuffle is a variation of doudizhu that essentially uses entirely the same rules as described above. The only difference with the base game is that the cards are distributed in a way that makes it much more likely to receive multiple cards of the same rank together. As this is a manipulation of the inherent randomness in the distribution of cards at the start of a game round, it is somewhat difficult to execute in a real situation and is more commonly found in phone apps. This makes it hard to find exact rules as to how the randomness of the card distribution is influenced in no shuffle, as this specific point is hidden behind proprietary software. To the best of our efforts, we were not able to find exact No shuffle rules online. The following subsection is our best approximation of the way cards are distributed in No Shuffle.

Distribution Algorithm

Instead of shuffling the deck normally, in No Shuffle, we create a "partial shuffle". The goal of this shuffling method is two fold:

- Make it more likely for a player to have four cards of a kind.
- Distribute ranks fully randomly among players.

Algorithm 4 Swap Shuffling

```
procedure SWAPS_SHUFFLE($swaps$)
    $ranks = [3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A, 2, Joker]$
    $ranks.shuffle()$
    $deck \leftarrow List()$
    for $card$ in $ranks$ do
        if $card == Joker$ then
            $deck.extend([BlackJoker, RedJoker])$
        else
            $deck.extend([card] * 4)$
        end if
    end for
    for $i$ in $range(swaps)$ do
        $first, second \leftarrow randint(len(deck), size = 2)$
        $deck[first], deck[second] \leftarrow deck[second], deck[first]$
    end for
    $position \leftarrow [0, 1, 2].shuffle()$
    $hands \leftarrow Dictionary()$
    $hands[landlord] \leftarrow deck[index[0] * 17 : (index[0] + 1) * 17] + deck[51 : 54]$
    $hands[landlord\_up] \leftarrow deck[index[1] * 17 : (index[1] + 1) * 17]$
    $hands[landlord\_down] \leftarrow deck[index[2] * 17 : (index[2] + 1) * 17]$
    $hands[three\_cards] \leftarrow deck[51 : 54]$
    for $hand$ in $hands$ do
        $hand.sort()$
    end for
    return $hands$
end procedure
```

We achieve this with a simple rank shuffling and swapping algorithm shown in Alg. 4. This algorithm has the advantage of making it possible to have a sliding scale of randomness in the distribution of cards. If the number of swaps is zero, each player will have almost only quads but passed a certain number of swaps, the deck essentially becomes fully randomized.

With some brief testing, we saw that having 5 swaps most closely replicated no shuffle behaviors seen in Tencent's "Happy DouDizhu". Of course this is what we consider a reasonable guess as to how the shuffling in this variant works as we are not able to access the true shuffling algorithm.

Moreover, in our implementation the three bonus cards were not distributed with NoShuffle. This was due to a bug in our implementation and noticed too late to fix it. As the main interest of this version of DouDizhu is that the distribution of cards is skewed towards having multiple of one rank it should not be overly detrimental to our ability to analyze imperfect information characteristics. It does however probably remove the advantage of being the Landlord as they do not get bonus cards but at the same time could be to their advantage as they have to play less cards to the end of the game. We still include and analyze the results of this bugged game as they can be interesting and will add results of the corrected version if time allows.

5.2.2 Runfast

Runfast changes the dynamic of the game in a few ways, of which the most important is that all players now play individually. They get their own score and are competing with the two others; there is no explicit element of cooperation anymore.

Our implementation of the rules of Runfast is based on the Tencent application "Happy DouDizhu".

Deck

Runfast is played with a modified deck. It contains all normal cards from "3" to "Kings" but only three "Aces" and one "2". This is a total of 48 cards. The ranking of cards is the same as DouDizhu with the Jokers removed: 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A, 2.

Dealing

Each player gets 16 cards dealt randomly at the beginning of the game by the dealer. We haven't found a precise rule for which player is the dealer on a given round. Given that the application manages the card dealing automatically, we don't have an exact rule for which player should deal the cards on a given round. Our choice to proceed with this in a real life game, would be to choose a random dealer on the first round and have all three players take turns being the dealer in a clockwise manner on following rounds.

As there is no bidding phase in Runfast, players move on directly to the cardplay phase.

Cardplay

The cardplay phase of Runfast behaves similarly to Doudizhu with a few differences.

First, as positions are not determined by the bidding phase, the starting player is the one that start with the 3 of spades (3) and players then play in a clockwise manner. This is the only instance where a card suit matters in this version of the game.

As in DouDizhu, the first player is free to choose whichever valid combination of cards they want to play. In Runfast, passing is not an action one can take freely. A player is only allowed to pass when they have no other available action. Otherwise, as long as at least one action is available, the player must use it.

Bombs do not double the round's pot. Moreover, when a player plays a bomb, as explained above, the following players have to continue playing bombs until two consecutive players have to pass. When this occurs, the last player to have played a bomb gets paid a value of 20 (10 by each opponent). We refer to this as the "Bomb chain" rule.

Reward

The reward for the winner of a round is equal to the sum of remaining cards in the other players' hands. Each player pays a value equal to the amount of cards remaining in their hand at the end of the round to the winner.

However, if player X takes their action and the following player (Y) has only one remaining card, player X should use their highest possible card in the event they play a single. If X fails to do so and Y wins with the next move, X is considered to have unfairly assisted Y in winning and consequently must pay not only their share of the reward owed to Y but also player Z's share. This happens even if X didn't have a higher card than Y in their hand; as long as they haven't used their highest ranked card in this situation they will have to pay the full value owed to Y for the round.

Combinations

Actions in Runfast are similar to the ones in DouDizhu with some key differences.

1. **Pass:** Only available when no other actions are possible
2. **Solo:** A single card of any rank
3. **Pair:** Two cards of same rank
4. **Trio:** Three cards of same rank (only available if it allows to finish the round)
5. **Trio with Solo:** Three cards of same rank with a distinct kicker (only available if it allows to finish the round)
6. **Trio with Two kickers:** Three cards of same rank with any two cards
7. **Chain of Solos:** Five or more cards of consecutive rank (2 excluded)
8. **Chain of Pairs:** Two or more consecutive pairs
9. **Plane with Two kickers:** Two or more consecutive "Trio with Two kickers"
10. **Quad with Three kickers:** Four cards of same rank with any three cards
11. **Bomb:** Four cards of same rank (Starts a bomb chain)

Please note that Kickers can be of the same rank as the Trios they are played with.

5.3 Related work for DouDizhu

In this section we introduce other research done on DouDizhu and more specifically the research on which our experimentation was based on. As previously discussed in Section. 5.1, DouDizhu is divided in two main parts; the bidding phase and the cardplay phase.

5.3.1 Bidding agents

Many agents tackle problems similar to the bidding phase of DouDizhu. Poker's betting system, for example has a lot of similarities with DouDizhu's bidding. We could even consider them to be the same process if we regard the following cardplay phase of DouDizhu and Poker's card reveal/bet resolution phases as a subsequent process with some stochastic outcome from a given player's point of view.

However, according to the author's knowledge, there are only two agents that somewhat tackle the bidding phase of DouDizhu: DouZero [44] and DouZero+ [46]. The reason we say "somewhat" is that, even though these works do include Bidding networks, they are used to call the landlord position instead of having to bid for it. In this "calling" system, the first player to choose to call the landlord automatically gets that position, which is not the same as having to bid for it. Essentially, it is equivalent to a bidding system that only allows 0 or 3 as actions.

5.3.2 Cardplay agents

Numerous studies have aimed to master the cardplay stage of DouDizhu, resulting in several types of agents. Jiang et al. [Jiang et al.] used MCTS in conjunction with predictions of other players' handcards and a trained action-abstracting network. Zhang et al. [45] used inspiration from AlphaZero and developed an Action-Prediction MCTS algorithm, which achieved better performance than its predecessors by avoiding the challenging task of predicting other players' handcards.

Zha et al. [44] also proposed the DouZero agent, which used Deep Monte Carlo. In subsequent developments of DouZero, Zhao et al. [46] improved performance by adding a network that predicts the opponent's hand and a network that evaluates the quality of the players' hands. Luo et al. [24] further improved this agent by combining the two networks and using a Resnet architecture while employing a heuristic function to enhance the efficiency of the training process during the self-play phase. Wang et al. [41] added probability and value factorization to DouZero, resulting in faster and more stable training of the original agent.

There are also agents based on Q-learning and actor-critic methods. For example, the Combinational Q-Network (CQN) reduces the action space by decoupling heuristics actions. Yu et al. [43] created an agent with a neural fictitious self-play and actor-critic training framework, which outperformed previously well-known open agents within a few hours of training. Guan et al. [42] trained an agent with perfect information and used an actor-critic framework to handle imperfect information in the actual gameplay stage.

5.3.3 DouZero+ landlord-calling agent

In douzero+ [46], the authors used supervised learning to train a landlord-calling agent.

The implementation involves fixing the hand to be evaluated, randomly sampling 5000 possible arrangements of the other players' hands and the three bonus cards. These 5000 starting states are then played through with the Douzero cardplay agent (remaining cards are added to the hand to be evaluated). The supervised agent is then trained to replicate the Winning Rate over these 5000 simulations given the evaluated hand. The author generated a sample of 10,000 starting hands, of which 80% were used for training and 20% were used for evaluation.

Once training is done, an evaluated winning rate of at least 50% implies it is worth it to call for the landlord.

5.3.4 PerfectDou

PerfectDou [42] is currently the best Dou Di Zhu intelligent agent, and its actor-critic based imperfect-execution-perfect-training framework can be adapted for other games that involve imperfect information. The implementation involves creating two separate neural networks, the actor and the critic, which are fed imperfect and perfect information inputs respectively. Upon completion of training, the actor network alone is utilized to produce game strategies, removing the need for knowledge of the opponent's hand.

In PerfectDou, the author also employs a heuristic function to calculate the minimum number of actions required to empty one's hand; this information is added to the inputs of both neural networks. To address the issue of sparse rewards, the author has designed a reward system based on the minimum required actions to empty hand, which outputs the immediate reward for each action.

5.4 Algorithm

The agents we trained in this case study are based on DouZero [44] and DouZero+ [46], for which we have presented the algorithm (DMC) in Section 3.3. At the time this work was starting they were the best DouDizhu agents. Since then, Perfectdou [42] has achieved better scores but, while their paper explains their methodology, there is no available repository of their full implementation as of yet. Hence, Douzero and Douzero+ are the highest performing DouDizhu agents freely available

Both of these agents use the DMC algorithm with some specific adaptations for the game of DouDizhu. Firstly, both of these agents train three separate networks, one for each of the three possible positions in the game (Landlord, Landlord Down, and Landlord Up).

In the context of Douzero, the game node is the cardplay phase of Doudizhu. States contain a given players hand cards and played card combination history. Actions are card combinations. The Reward is the base reward multiplied by the number of bombs played (but not multiplied by the highest bid).

5.4.1 Encodings

We use the same card encoding method as [44] shown in Figure 5.5

As suits do not matter in any version of DouDizhu, the only important thing is to understand how many cards of a specific rank are in a given player's hand.

For each rank, there are five possible states in a player's hand; a player can have zero, one, two, three or four cards of a given rank (except both Jokers who only have 2 possible states but are also considered as two separate ranks) which we further call **rank states**.

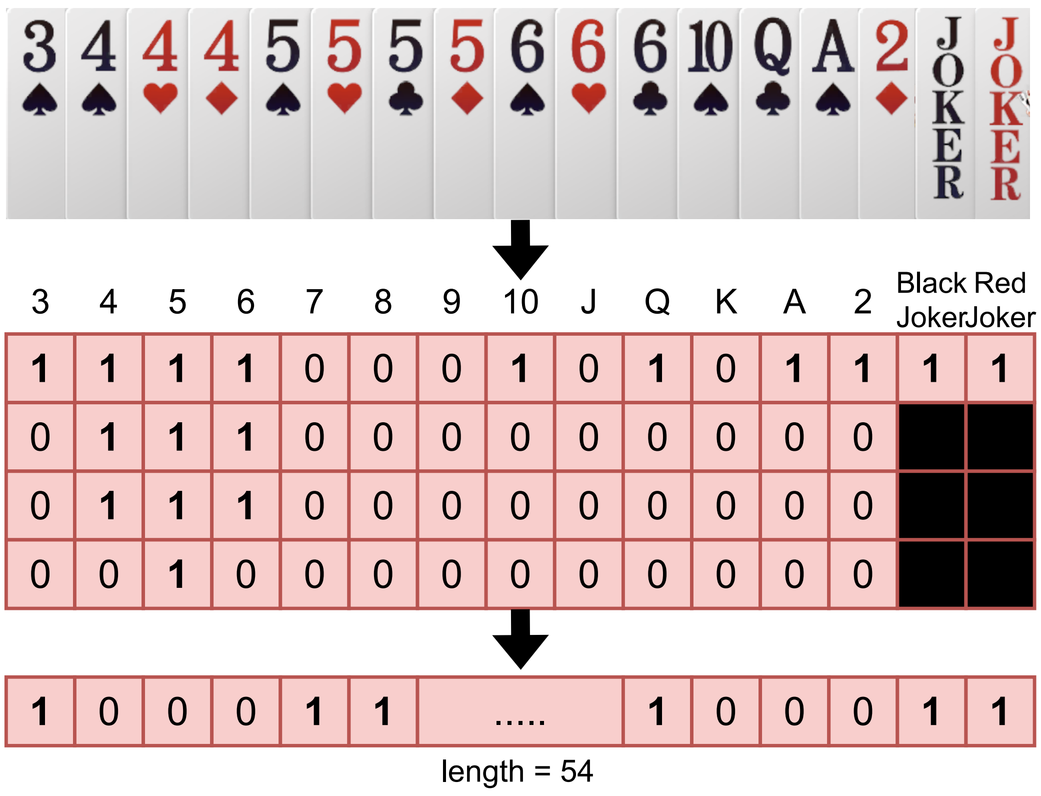


Figure 5.5: Card Encoding

Card Encoding

In first encoding, we consider that the 0 card state is redundant and forgo representing it. As shown in Figure 5.5, each column represents one rank. Moreover, for the other states, they are set to True in an additive manner rather than an excluding manner (if a player has 3 cards of a rank, then they have 1, 2, and 3 cards available). Hence, each line correspond to an additional card of said rank.

Once flattened, we are left with a 54 dimensional vector encoding a set of cards. This encoding can represent played cards, cards used in an action, a player's current hand, etc...

Hidden Information Encoding

This encoding is used for hidden information specifically. It does not forgo the 0 card state representation and does consider only one state to be true at a time. This makes it so that

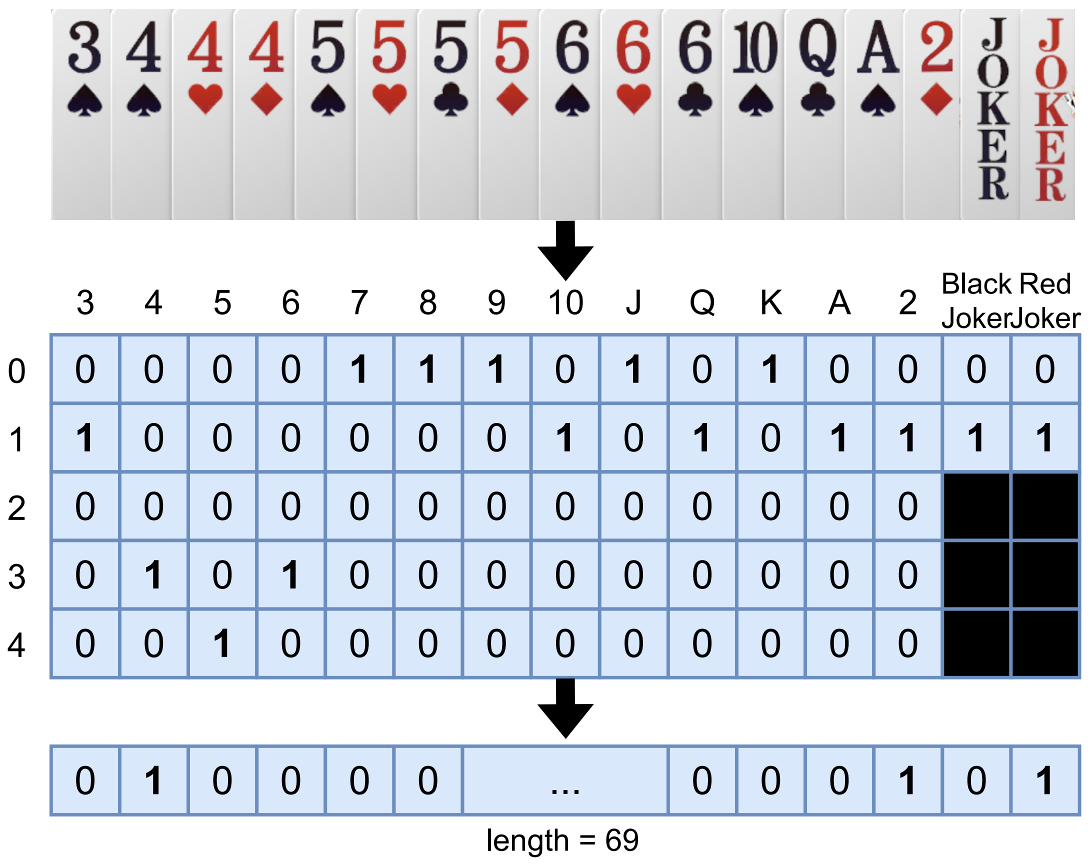


Figure 5.6: Hidden Information Encoding

each column sums up to 1, which allows for an easy representation of probabilities for each rank. It allows to encode a player's attempt at estimating the state of the hidden information by converting the binary values to probabilities of a player possessing a corresponding amount of cards of a given rank.

Flattening this encoding results in a 69 dimensional "Visualization Vector" which is our basic representation of hidden information. In DouDizhu a visualization vector is used to represent the estimation of what the probabilities of a given opponent having a given number of cards of a certain rank in their hand are.

5.4.2 Networks

All networks used here are based on DouZero [44] and DouZero+ [46].

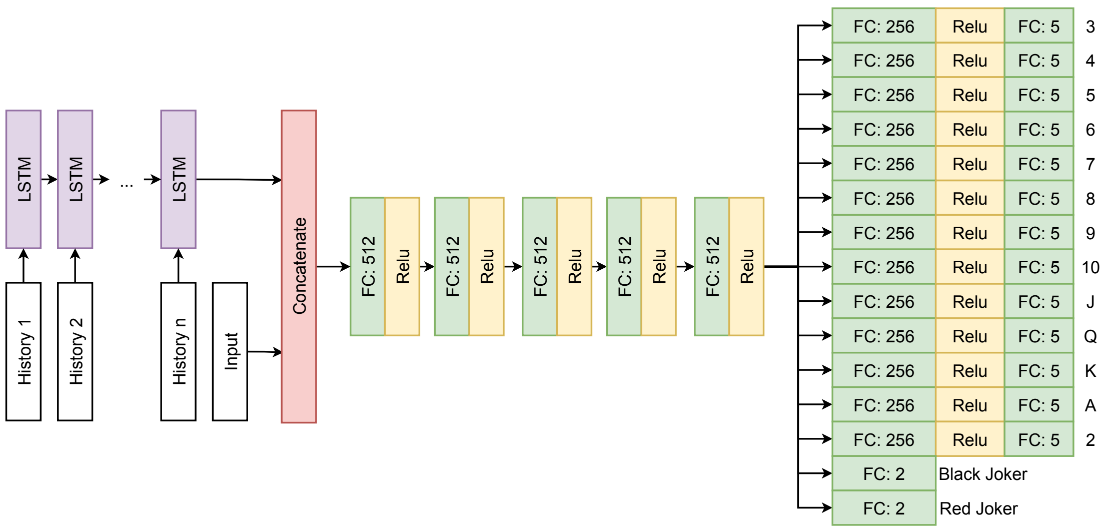


Figure 5.7: Estimation network

Visualization

In the Visualization experiment, we use Deep Monte Carlo with Estimation as presented in Algorithm 3. The hidden information estimation network receives the current player's infostate consisting of the following elements:

- Hand cards
- Cards played by previous player
- Cards played by next player
- Two last actions
- Number of cards left in each other player's hand
- Number of bombs played
- History of all played actions

All inputs are represented using card encodings or one hot vectors for integer values and then concatenated. The History of all actions is however kept separate to be passed through an LSTM block of the network (Figure 5.7).

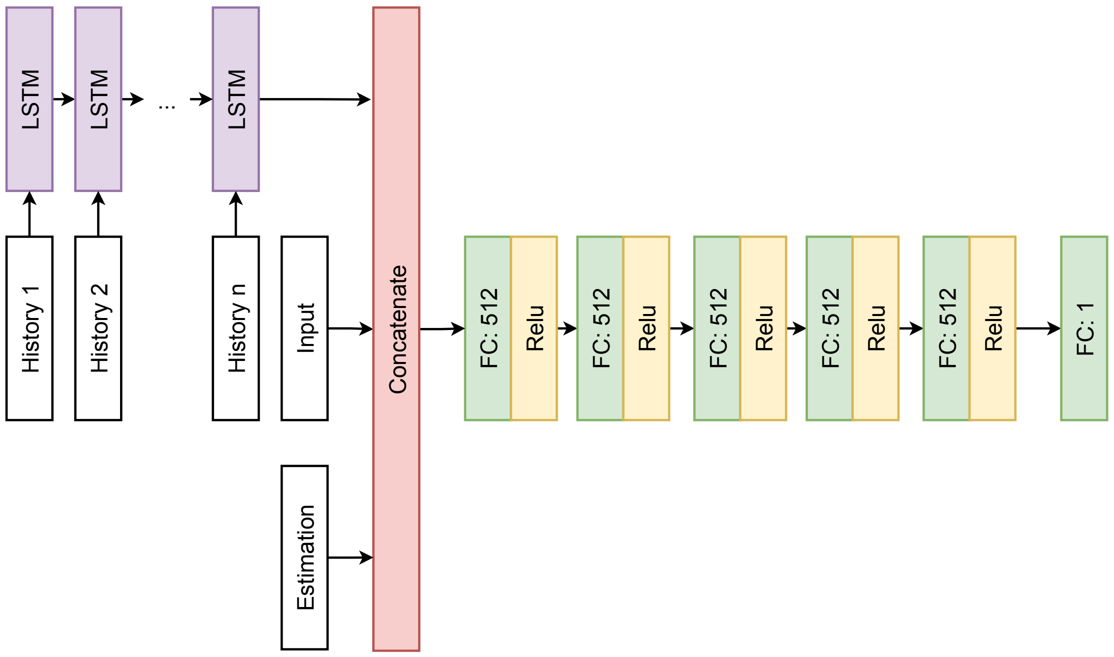


Figure 5.8: Action value network for Visualization

The output of the estimation network is then filtered to remove all logits corresponding to non available cards, after which each output block corresponding to a rank is passed through a softmax function and flattened. The resulting visualization vector is then added to the inputs of the action value network (Figure 5.8).

Impact

For the impact experiment, we use only the second network. We do however slightly modify the input of said network to not receive the visualization vector from the estimation network and replace it with a full card encoding of the hidden information for the cheated player. In the case of the non-cheated player, the full card encoding is replaced with a vector of zeros.

5.5 Visualization Experiment

This experiment is designed to measure the Imperfect Information Visualizability concept presented in subsection 2.3.1. The way we quantify this concept is based on finding a way to measure the distance between attempted estimations and true states of hidden information. To get these estimations, we trained agents using DMC with Estimation (Alg. 3) to see how well they could perform at estimating other player hand cards.

5.5.1 Hidden Information Size

The first step of determining the visualizability of a given game is to establish the size of the hidden information in said game. In the case of DouDizhu and its variants, the hidden information solely lays in which of the two other players has which of the yet unseen cards in their hand.

Given the Visualization Vector encoding (Subsection 5.4.1), we can give a simple state space complexity of hidden information. Altogether there are 69 $(13 \text{ ranks} * 5 \text{ states} + 2 \text{ jokers} * 2 \text{ states})$ binary components to a visualization Vector. Each of the two opponents has a hand that requires visualizing, this implies the hidden state space size would be $|S| = (2^{69})^2 = 2^{138}$.

However, this is certainly an overestimation. All cards are in a player's hand or have already been played. Any card not played or not in the current player's hand is bound to be in another player's hand and any card not in one of the visualized player's hand is by default in the other player's hand. Considering this last point, we can already reduce the hidden state size by estimating only the hidden state of one other player (in our case always the next player to play) instead of both: $|S| = 2^{69}$

Moreover, when taking in account that all players have access to their 17 hand cards and are allowed to see three bonus cards at the start of the game, there are 20 cards that can immediately be removed from the realm of possibility or added with certainty to an estimation when estimating opponent hand cards. In effect, this guarantees that at least 20 values of the encoding are 0, thus we have: $|S| = 2^{69-20} = 2^{49}$

Lastly, the hidden information state space size will progressively go down as players use cards during the game. The given state space complexity is the biggest possible one and occurs at the beginning of the game.

5.5.2 Visualizability Measures

To measure the visualizability of a game, we need to either develop our own measure or adapt an existing metric.

The ideal visualization vector is equal to the vector encoding of the hidden state of the following player's hands. We can consider the true state of the information is a given point inside the state space of this vector and any visualization is a point elsewhere in this state space. The more precise the visualization vector, the closer its point should lie to the true state's point. Hence, a simple way to measure the accuracy of a visualization is by using a measure of distance between it and the true state.

Distance measures

The distance measures we consider in this work are the following:

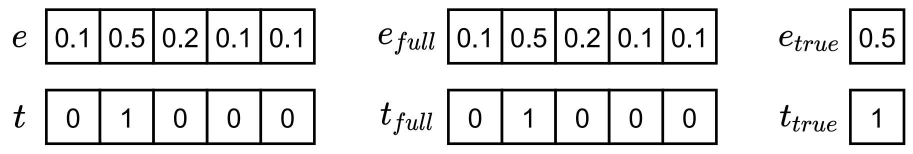
- Euclidean Distance: $d(e,t) = \sqrt{\sum_{i=1}^{n} (t_i - e_i)^2}$
- Mean Squared Error (MSE): $d(e,t) = \frac{1}{n}\sum_{i=1}^{n} (t_i - e_i)^2$
- Mean Absolute Error (MAE): $d(e,t) = \frac{1}{n}\sum_{i=1}^{n} |t_i - e_i|$
- Binary Cross Entropy (BCE): $d(e,t) = -\frac{1}{n}\sum_{i=1}^{n} t_i * log(e_i) + (1 - t_i) \cdot log(1 - e_i)$

Where e is the estimation vector, t is the true value of the hidden state vector, and n it the length of both of these vectors.

Some of these measures are not classically considered distance measurements, however, insofar as they are establishing a form of proximity relationship between data points they do enter a broad understanding of distance where they tend to zero as data points become more similar. Simply put, we are using the same loss functions that can be used during an artificial neural network's training to compare a network's output and it's target. This is inherently what the estimation and true value vectors were at the time of training. Euclidean Distance is not commonly used as a loss function but it might be the most instinctively understandable which is why we use it as well.

Finally, there are two main ways to consider how exact a prediction might be. The first is to consider that estimation over all possible states are important, the second is to consider that only the prediction over the true state is important. The argument for the second being: when estimating hidden information what is important is how close the estimation is to the truth rather and how exactly an estimation is wrong does not matter. Moreover, calculating distances based

Figure 5.9: Full and True Space distance vectors



on all components of the visualization vector gives an increased disadvantage to estimations based on more possible states; each added state adding another possible dimension of error. To mitigate this we introduce two secondary concepts: that of **full space distance** and **true space distance**. Quite simply, full space estimation is the distance we have discussed so far using the entire visualization vector and the complete corresponding true state of the following player's hand. True space estimation, however, only calculates distances based on the components of the visualization vector where the corresponding true value is one. Mathematically the full space distance is $d_{full}(e, t) = d(e, t)$ and the true space distance is $d_{true}(e, t) = d(e_{true}, t_{true})$ where $e_{true} = \{e_i | t_i = 1\}$ and $t_{true} = \{t_i | t_i = 1\}$, which is a simple vector of ones. Graphically, as illustrated in Figure 5.9, if e and t are the estimated state and the true state of another player's cards over a single rank, the full state distance simply takes the same vectors while true state distance removes all zero values from t and corresponding values from e.

Maximum distance

In the interest of creating measures that can be used with any possible game and still allow for comparison of visualizability among very different games, there is a need to constrain said measures within a certain range. Constraining values in a given range becomes feasible given knowledge of the maximum possible error of an estimation. Using this maximum possible error, we can scale any estimation to a value between 0 and 1 by dividing the estimation's distance measure with the maximum possible error's distance.

Given our encoding, the maximum possible error occurs when the visualization vector is a bitwise inverse of the true hidden information vector. Calculating the distance between two bitwise inverse vectors of dimension 69 gives us the maximum distance.

However, this maximum distance is not a good scaling factor as it ignores the fact that different probabilities in the estimation are not independent. As detailed in Subsection 5.4.1,

Table 5.1: Measure Characteristics

Measure	Range	Full Max Distance		True Max Distance	
		DouDizhu	Runfast	DouDizhu	Runfast
Euclidean	$[0,\infty)$	$\sqrt{30} = 5.4772$	$\sqrt{26} = 5.099$	$\sqrt{15} = 3.873$	$\sqrt{13} = 3.6055$
MSE	$[0,\infty)$	$30/69 = 0.4348$	$26/61 = 0.4262$	$15/15 = 1$	$13/13 = 1$
MAE	$[0,\infty)$	$30/69 = 0.4348$	$26/61 = 0.4262$	$15/15 = 1$	$13/13 = 1$
BCE	$[0,\infty)$[a]	∞	∞	∞	∞

[a] Capped at 100 in pytorch

the sum of probabilities corresponding to one of the encoding's columns (card rank) should always sum up to one. Hence a realistic maximum error evaluation for a rank would be any vector belonging to a set such that:

$$X = \{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1 \text{ and } \forall t_i \in t, t_i = 1 \Rightarrow x_i = 0\} \tag{5.1}$$

Where t is the true state vector for a given rank and n is 4 or 2 for Jokers. As no visualization vector's distance should be able to exceed the maximum error's distance, we can further restrict this set X to a subset that maximizes all distance measures presented above.

$$X_s = \{x \in \mathbb{R}^n : \exists x_i = 1 \text{ and } x_{j \neq i} = 0 \text{ and } t_i = 0\} \tag{5.2}$$

Combining $13(n = 5)$ and $2(n = 2)$ vectors from X_s we get a maximum error visualization vector for which we can calculate the maximum distance with each distance measure used. Maximum possible distances are given in Table 5.1.

Scaling these measures by this maximum possible distance allows us to obtain a universal measurement scaled between zero and one applicable to any game. The requirement for this to work however, is to be able to express the hidden state of a game as boolean values and to express the maximum possible error in as tight a subset as possible.

Note that we do still use 69 to calculate the maximum distance despite the space size of the hidden encoding being reduced from 2^{69} to 2^{49} (see subsection 5.5.1). We consider the elimination of cards from the range of possibilities can be seen as a learned behavior performed by whatever agent is doing the estimations. Hence, if an agent is sufficiently bad at estimating hidden information, it would be possible for it to have a estimated distance larger than one which would go against our goal in introducing these measures. This will have the effect that good agents probably will never reach the maximum possible error in most cases.

5.5.3 Experimental setting

Our training was run on a series of machines equipped with an Intel(R) Core(TM) i7-9800X CPU and two GeForce RTX 2080 Ti GPUs. We used python 3.7.0, pytorch 1.6.0 Cuda driver version 510.73.05 and cuda toolkit version 11.6. When training our agents, we did so with 10 actors divided over the two GPUs main process updated the network from the actors' samples. Three machines were used, one for each version of DouDizhu, on which each training ran for 200 hours. On each of these, the algorithm remains the same with the only differences being the game environments and some of the input information dimensions in the case of Runfast. During training we saved images of our networks every thirty minutes as was done in the original DouZero paper [44].

To show the progress of each agent, as a sanity check and to assess convergence, we plot the performance of each of three networks trained per game (one for each player position) at all points of training against the final versions of the other two networks.

Once the training was done, we generated 10000 starting hands for each of the three Doudizhu versions. We had our trained agents play each of these games and recorded their estimations from each position and for each action. We then measured the distance between each estimation and true state of hidden information using the measures introduced in subsection 5.5.2, the results of which are shown in Table 5.2.

As a comparison for these scores, we also created heuristic hand estimations using three methods.

- **Random Method**
- **Masked Random Method**
- **Weighted Mask Random Method**

While it is not strictly necessary for us to compare the estimations generated based on DMC with estimations, it is still useful to have them as a minimal quality check and also as a basic reference comparison.

Random Method

Each possible state of a rank column in the hidden encoding (subsection 5.4.1) is given equal probability such that the probability over the rank column sums to one.

Masked Random Method

All values in the hidden encoding corresponding to cards that have been played, cards that are in the estimator's hand or three bonus cards when applicable are set to 0. In the case of the Landlord, the three bonus cards are already in their hand, hence are disregarded. For the Landlord Down player, each occurrence of a rank in the three bonus cards removes one of the possible states for that rank from highest to lowest (until a card of that rank is played by the landlord at which point it is only filtered out because it has been played). For the Landlord Up player, each occurrence of a rank in the three bonus cards removes one of the possible states for that rank from lowest to highest (until a card of that rank is played by the landlord). Remaining values of each column get equal probabilities, summing up to one over the column.

Weighted Masked Random Method

Impossible values are masked in the same way as with the masked random method. Subsequently, we hypothesize that each card held by the next player could be any of the remaining unplayed cards, each with an equal probability. Thus, remaining values are calculated according to eq. 5.3.

$$p(j_r|m_r, h, n) = \frac{\binom{m_r}{j_r} \times \binom{n-m_r}{h-j_r}}{\binom{n}{h}} \tag{5.3}$$

This represents the estimated chance for the next player to possess j_r cards of rank r given that m_r is the number of unaccounted cards of that rank left in the game, h is the amount of cards left in the next player's hand, and n it the total unaccounted cards left in the game.

This formula works fine as is in Runfast but needs some adjustments in DouDizhu and NoShuffle due to the three bonus cards. To adjust we need to remove all the combinations made impossible by the current amount of guaranteed present or guaranteed absent cards in the next player's hand. For the Landlord-Down position, we update m_r to $m_{r,LD} = m_r - tc_r$, and n

to $n_{LD} = n - tc$, where tc is the number of unplayed bonus cards and tc_r is the number of unplayed bonus cards of rank r. Similarly for Landlord-Up, the update is $m_{r,LU} = m - tc_r$, $j_{r,LU} = j - tc_r$, $n_{LU} = n - tc$, and $h_{LU} = h - tc$.

5.5.4 PerfectDou

Additionally, despite not being able to access a full implementation of PerfectDou, a trained agent is available online [4].

We trained an additional Doudizhu estimation agent by replacing the action-value network during training with the Perfectdou's action-value network. While the trained estimation network had no impact on Perfectdou's network, it is still able to learn to estimate hidden information based on Perfectdou's strategy. As DouDizhu and No shuffle only differ in how starting hands are distributed, PerfectDou can be used for both of these games.

5.5.5 Results

Global distances

Measuring all the distances detailed in subsection 5.5.2 for the trained agent and for the heuristic estimation methods (subsection 5.5.3) yields the results presented in Table 5.2. In this table all distances are scaled by the maximum distance detailed in subsection 5.5.2 with the exception of BCE which we did not scale as the maximum distance is infinite and practical distances where too small to be scaled even by the caped distance of 100 used in pytorch.

First, from the table results, we notice that all truth minus estimation, $t_i - e_i$, based distances (Euclid, MSE and MAE) follow the same trends even if the numbers are not the same. Hence from here we will mostly discuss based on Euclid distance which will also port to the other two distances unless specified.

Table 5.2 shows that the heuristic estimations behave as expected with Random $>$ Masked Random $>$ Weighted Masked Random. This, however, stops being true for NoShuffle where Masked Random performs better than Weighted Masked Random for all measures (both in Full and True space estimations). NoShuffle has a higher likelihood to have multiple cards of a same rank in a player's hand so a good estimator for it should give either a high likelihood to have

Table 5.2: Global scaled distance scores (lower is better)

		Full Space Estimation				True Space Estimation			
Game	Method	Euclid	MSE	MAE	BCE [a]	Euclid	MSE	MAE	BCE [a]
DouDizhu	Agent	0.42	0.17	0.30	0.38	0.45	0.20	0.29	0.82
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.49	0.22	0.44	0.27	0.55	0.28	0.43	0.71
	Weight. mask.	0.46	0.20	0.38	0.25	0.50	0.23	0.37	0.63
RunFast	Agent	0.35	0.16	0.27	0.41	0.37	0.19	0.28	0.86
	Random	0.58	0.38	0.76	0.51	0.72	0.60	0.77	1.52
	Masked random	0.42	0.21	0.41	0.26	0.47	0.27	0.42	0.69
	Weight. mask.	0.39	0.18	0.36	0.23	0.42	0.22	0.36	0.61
NoShuffle	Agent	0.38	0.15	0.27	0.30	0.42	0.18	0.27	0.70
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.50	0.22	0.44	0.28	0.59	0.31	0.43	0.78
	Weight. mask.	0.56	0.28	0.47	0.86	0.66	0.39	0.49	2.53

[a] Not scaled

zero cards of a rank or three or more cards of that rank. In that case, the Weighted Masked Random method, which is assuming same likelihood for any card to end up in a player's hand, should almost always have a minimal likelihood for a rank to be at four cards in a player's hand whereas in NoShuffle the minimal likelihood should be to have one or two cards of a rank with higher likelihoods for having zero, three or four cards of a rank in a hand.. Altogether this is expected behavior so it is good to see it confirmed here.

Looking at BCE reveals information on how exactly estimators make mistakes. BCE does not increase the same way as the other measures due to its logarithmic component. In our encoding, the maximum difference on a given component between an estimated value and a true value is 1. For Euclid distance, this results in an increase of the error by $1^2 = 1$ which is then passed through square root but for BCE this would result in an added error of $-1 * \log(0) = \infty$ (capped at 100 as we use pytorch for calculations). Hence BCE can theoretically return an infinite distance value as long as one of the estimations gives a probability of one to a component of the encoding where the true state is actually zero or vice-versa. Practically this means that BCE distance punishes estimations with high “confidence” that are wrong whereas other distance measures have a more “general” increase with bad estimations. This explains why our agent performs worse than the Masked Random and Weighted Masked Random methods on

DouDizhu and RunFast with BCE. It means that the agent is giving high probability estimations at one state rather than giving all possible states equal probability like the Masked random method would do. In other words, it is likely that our agent is not giving conservative statistical estimations but actually attempting to guess a given state based on game observations. This can however cause it to have costly misses with regards to BCE. It also highlights a weakness in distance measures used here as all possible estimated states are considered completely separate entities whereas there is a strong argument to be made that estimating another player has 4 of a rank when they actually have 3 is better than estimating they have 0. We have since then been made aware of other distance measures such as the Wasserstein metric that would compensate for this problem however this is something that would need to be tested in future work.

Looking at all agent results, the trends do not change between Full and True Space Estimations, however they do change between BCE and other distances. For other distances, the order is as follows: DouDizhu $>$ NoShuffle $>$ Runfast, while for BCE it is RunFast $>$ DouDizhu $>$ NoShuffle. This does show us that whatever distance measure we use can greatly affect the perceived quality of the estimation of a given game and that it is thus important to select an appropriate measure to classify the level of visualization of a game. On the other hand, this can also be interpreted as different measures highlighting different qualities of a specific estimation. From what we highlighted so far, BCE seems like a good way of measuring Visualization; it favors good estimations, harshly punishes cases when a state is wrongfully given high probability and does not punish so harshly when multiple states are given about even probabilities. In other words, it rewards a player when they are very certain of their estimation and are correct about it and encourages players to take a more measured approach on estimations that could be wrong rather than giving high probability to an uncertain state. The ordering of distances of all three games also makes somewhat sense with BCE, one would expect the visualizability of NoShuffle to be the highest as most of the time a player has more than one card or a given rank. DouDizhu coming in second with BCE also makes intuitive sense as some of the cards are shown to all players at the start of a round, which allows to have an easier elimination and deduction process when estimating another player's hand cards. However this can be seen another way; one could expect it should be lowest as generally a player with many cards of a given rank will play them together for a strong combination which might make it hard to predict whatever cards they have until said cards are played. More testing would be necessary to establish which of these scenarios is true.

The main problem with BCE is that it cannot be scaled as any estimation on any component of the encoding could yield a theoretically infinite distance if wrong enough. It is also worth pointing out that the training was done using Crossentropy Loss, which works similarly to BCE and could thus have an influence on the score distribution we see here.

The other three measures do have the advantage of having a more stable progression. In short, supposing that estimations over all possible components of the encoding would improve except one which would become fully wrong, BCE would still show a high distance value whereas the three others would all show an improvement.

Overall, we do see that the three games have different distance measures hinting to ease of Visualization being different for all of them. These measures are however fairly close between games for the three first measures at least. It is somewhat expected that similar games would have similar levels of possible Visualization. More games tested in the future would probably have more differences in global Visualization scores.

Removed Extremes

Given our suspicions on the effect of a few elements of an estimation having a disproportionate effect on the BCE distance, we decided to measure global distances with the 10% highest and lowest distances removed for all four tested distance measures. Results with high/low removed are given in table. 5.3

With the extremes removed, we can see that the three first distances do not change much, at least compared to the BCE measures which do change more significantly as per example with the agent's distance changing from 0.38 in table. 5.2 to 0.31 in table. 5.3 with Full Space Estimation measures. In the case of NoShuffle, this even makes our agent become the best performer, now overtaking the Masked Random estimation method.

Altogether, this confirms that BCE is more affected by some few high errors which does make it somewhat unreliable as a distance measure.

Table 5.3: Global scaled distance scores (lower is better, 10-90% samples)

		Full Space Estimation				True Space Estimation			
Game	Method	Euclid	MSE	MAE	BCE [a]	Euclid	MSE	MAE	BCE [a]
DouDizhu	Agent	0.43	0.17	0.30	0.31	0.46	0.20	0.29	0.77
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.50	0.23	0.45	0.28	0.56	0.28	0.44	0.72
	Weight. mask.	0.47	0.20	0.39	0.25	0.51	0.23	0.38	0.64
RunFast	Agent	0.36	0.16	0.26	0.31	0.38	0.18	0.27	0.77
	Random	0.58	0.38	0.76	0.51	0.72	0.60	0.77	1.52
	Masked random	0.43	0.21	0.42	0.26	0.48	0.27	0.43	0.69
	Weight. mask.	0.39	0.18	0.36	0.23	0.43	0.22	0.37	0.61
NoShuffle	Agent	0.40	0.15	0.27	0.24	0.43	0.17	0.27	0.63
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.50	0.23	0.45	0.29	0.60	0.31	0.44	0.80
	Weight. mask.	0.57	0.29	0.48	0.47	0.67	0.40	0.50	1.56

[a] Not scalled

Position Based Visualization

Table 5.4 shows scores on a playing position basis.

First, unsurprisingly, all positions in Runfast have very close distance measures. Given that the starting position is selected randomly, no player should get a big advantage or disadvantage with regards to estimations. As the starting player is chosen randomly based on who possesses a given card there shouldn't be any positional advantages Having the expected same values for most measures serves as a nice sanity check but, in future studies, we should attribute the landlord position to the randomly selected starting player rather than attributing positions and then selecting the first player.

Second, the three first distance measures show that Landlord-Up and Landlord-Down are within 0.01 of each other, both of which being lower than the Landlord's prediction distances. This is probably due to the Peasants having more solid data to estimate the hidden state in the form of the three bonus cards. On the other hand, BCE distance shows that the worst prediction comes from the Landlord-Up position. This seems to also carry over to NoShuffle where Landlord-Up is the worse estimator not only for BCE distance but for all distances. In all cases, Landlord-Down seems to have the best estimations.

Finally, again there is no major difference in behavior between Full and True Space Estima-

Table 5.4: Agent scaled distance scores by Position (lower is better)

		Full Space Estimation				True Space Estimation			
Game	Position	Euclid	MSE	MAE	BCE [a]	Euclid	MSE	MAE	BCE [a]
DouDizhu	All	0.42	0.17	0.30	0.38	0.45	0.20	0.29	0.82
	L	0.45	0.19	0.34	0.36	0.48	0.22	0.33	0.82
	D	0.40	0.16	0.28	0.34	0.43	0.18	0.28	0.76
	U	0.41	0.17	0.28	0.44	0.44	0.19	0.27	0.89
RunFast	All	0.35	0.16	0.27	0.41	0.37	0.19	0.28	0.86
	L	0.35	0.16	0.27	0.43	0.38	0.19	0.28	0.88
	D	0.35	0.16	0.27	0.41	0.37	0.19	0.27	0.86
	U	0.35	0.16	0.27	0.40	0.37	0.18	0.27	0.85
NoShuffle	All	0.38	0.15	0.27	0.30	0.42	0.18	0.27	0.70
	L	0.38	0.15	0.30	0.22	0.43	0.19	0.29	0.56
	D	0.35	0.12	0.23	0.24	0.38	0.14	0.23	0.55
	U	0.42	0.17	0.29	0.43	0.45	0.20	0.28	1.02

[a] Not scaled

tions.

Turn Based Visualization

In the following graphs (Fig 5.10, Fig 5.11 and Fig 5.12), we show the average distance measures over the duration of a game. These measures are taken over 10'000 games so there are at maximum 10'000 different hands being estimated at each turn. The right axis and grey line show how many games are still unfinished at any given turns. This does mean that later average distances are based on less samples and thus less reliable in general.

When looking at the progression of estimations over games we see that the three first estimation distances consistently go down during the game. Our guess as to why this occurs is that, the addition of cards to the action history allows to eliminate possibilities from consideration and thus naturally make the distances become smaller as the game progresses.

Once again, BCE is the outlier as in some cases, it increases as the game goes on. It might however display an interesting behavior. Removing impossible cards from consideration still allows a player to give too much probability to certain states, hence the increase in BCE is probably due to cases where the agent gives high probability to certain states that are wrong. There is a sense that, as the game progresses, our agent moves on from more conservative sta-

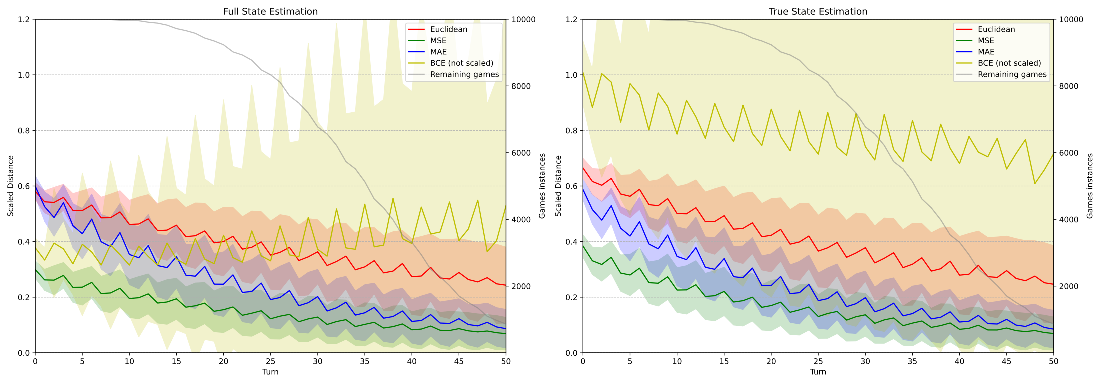


Figure 5.10: DouDizhu turn based estimations (with standard deviation)

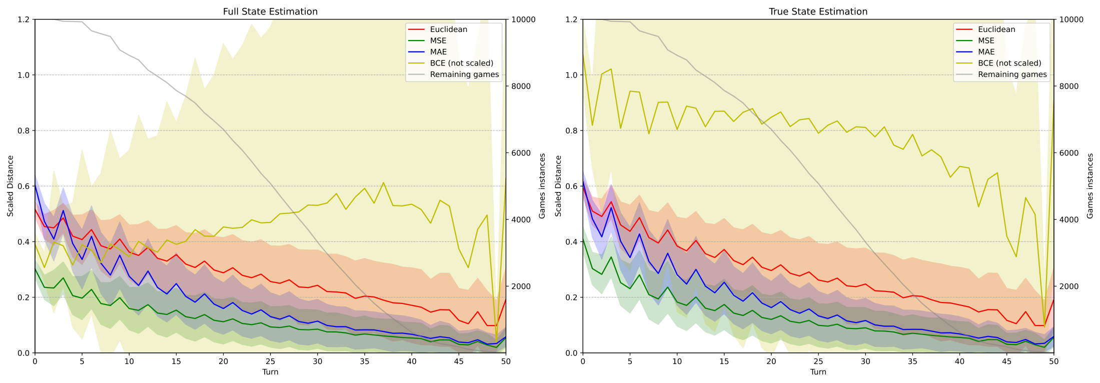


Figure 5.11: RunFast turn based estimations (with standard deviation)

tistical estimations to actual guesses of what cards an opponent has based on what they played. However, this also means that there is a higher chance to make estimations that are further from the truth than simple statistical estimations and thus would create a higher BCE error. Essentially the agent tries to give more accurate estimations which causes more costly mistakes until enough cards are removed from consideration that the possibility to make a mistake is reduced enough that the error starts going down. The issue with this theory is that, while the standard deviation of the other measures remains fairly stable across turns, even in later ones, the BCE's standard deviation is large and thus any conclusions made on the basis of these graphs in that regard would need more testing for significance.

Looking at the saw-tooth like pattern of distances cycling every three turns on all three graphs, these are due to only one of the three positions playing each turn. This correlates

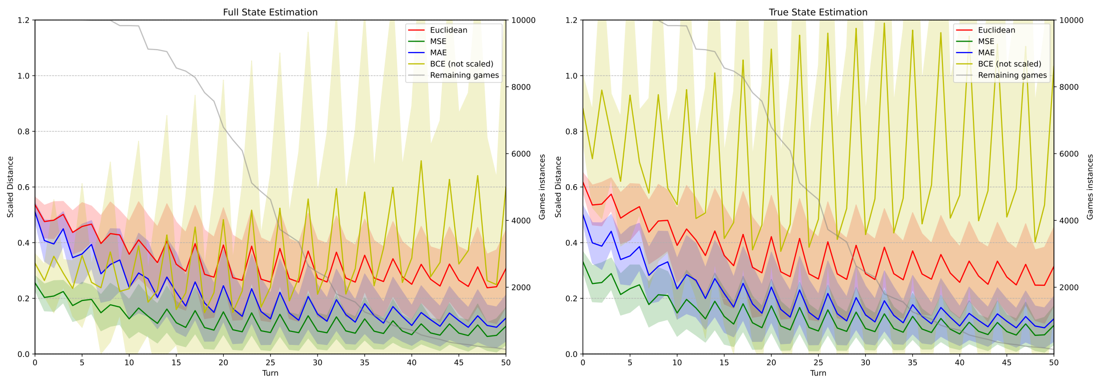


Figure 5.12: NoShuffle turn based estimations (with standard deviation)

with previously made observations that the landlord player does not manage to give as good estimations as the two other players in DouDizhu as the highest distance in the cycle is always on the turn where the Landlord is estimating. The reason why this cycle happens is probably due to the landlord player being able to, on average, dictate the flow of the game. In DouDizhu, as the game starts with the landlord playing cards, there is an immediate advantage for the other players to estimate another player's hand as they automatically will be able to remove the cards played by the landlord form consideration. Hence, they will on average have better estimations than the landlord. The reason why this pattern persists over the entire game is probably due to two things: first the landlord being the first to play and always being able to play on the first turn will make it so that they statistically will have less cards in hand than the two other players which would make estimations easier for others. Second, the landlord has a lot of control over the flow of the game as a common strategy is to play powerful combinations to prevent others from playing as long as possible, which will cause the landlord to play many cards while the other two players are not able to play as much cards at least in the first turns.

Interestingly this provides some insight into NoShuffle as we saw earlier that the landlord-up was the one with the highest distances in that version. When looking at Figure 5.12, we can see that landlord starts having the highest distances in the cycle but this behavior switches to landlord-up having higher distances between turns 9-11 depending on the distance measure, both in full- and true-state estimations. This might indicate that, on average, landlord-up is the one with less cards in hand by that time in the game which would point to them having taken

Table 5.5: Gobal scaled distance scores (PerfectDou)

Game	Method	Full Space Estimation Euclid	MSE	MAE	BCE [a]	True Space Estimation Euclid	MSE	MAE	BCE [a]
DouDizhu	Agent	0.43	0.19	0.33	0.34	0.47	0.22	0.33	0.81
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.51	0.24	0.48	0.30	0.58	0.31	0.47	0.78
	Weight. mask.	0.48	0.21	0.42	0.27	0.53	0.26	0.41	0.71
NoShuffle	Agent	0.34	0.12	0.24	0.17	0.37	0.15	0.23	0.44
	Random	0.66	0.39	0.78	0.51	0.82	0.59	0.76	1.49
	Masked random	0.48	0.21	0.42	0.27	0.57	0.29	0.41	0.74
	Weight. mask.	0.55	0.27	0.45	1.03	0.65	0.38	0.46	2.89

[a] Not scaled

control of the flow of the game by that point.

PerfectDou

Training a network to estimate hands on games played by PerfectDou yields the distances shown in Table 5.5.

Overall, the behaviors are the same as observed in Table 5.2. However, all distances are better when the estimator network is trained on PerfectDou games than when trained in conjunction with the DMC method.

When looking at distances based on positions (Table 5.6), the trend seen previously of the Landlord-Up having the worst estimations. In NoShuffle, Landlord-Up has the best estimations all over, which is somewhat surprising as it is the reverse of previous behaviors. Considering this and the somewhat smoother turn based average estimation behaviors, shown in Figure 5.13 and fig. 5.14, there is a chance that the gameplay as performed by perfectdou differs significantly in some ways from that of the DMC trained agent. As we were not able to train our own version of PerfectDou for NoShuffle and had to used an available agent trained on base DouDizhu for this experiment, it would make sense that the observed behaviors result from the PerfectDou agent not having a playstyle tailored to NoShuffle.

Estimation distances (Figure 5.13 and Fig. 5.14) also show similar trends to the ones presented above. However, games played with PerfectDou seem to finish much faster than when

Table 5.6: Agent scaled distance scores (by Position)

Game	Position	Full Space Estimation Euclid	MSE	MAE	BCE [a]	True Space Estimation Euclid	MSE	MAE	BCE [a]
DouDizhu	All	0.43	0.19	0.33	0.34	0.47	0.22	0.33	0.81
	L	0.45	0.20	0.36	0.32	0.50	0.24	0.36	0.79
	D	0.42	0.17	0.32	0.29	0.46	0.21	0.31	0.71
	U	0.43	0.19	0.32	0.42	0.47	0.21	0.31	0.92
NoShuffle	All	0.34	0.12	0.24	0.17	0.37	0.15	0.23	0.44
	L	0.37	0.14	0.28	0.20	0.41	0.17	0.27	0.52
	D	0.34	0.12	0.23	0.16	0.37	0.14	0.23	0.42
	U	0.31	0.10	0.20	0.15	0.33	0.12	0.19	0.38

[a] Not scalled

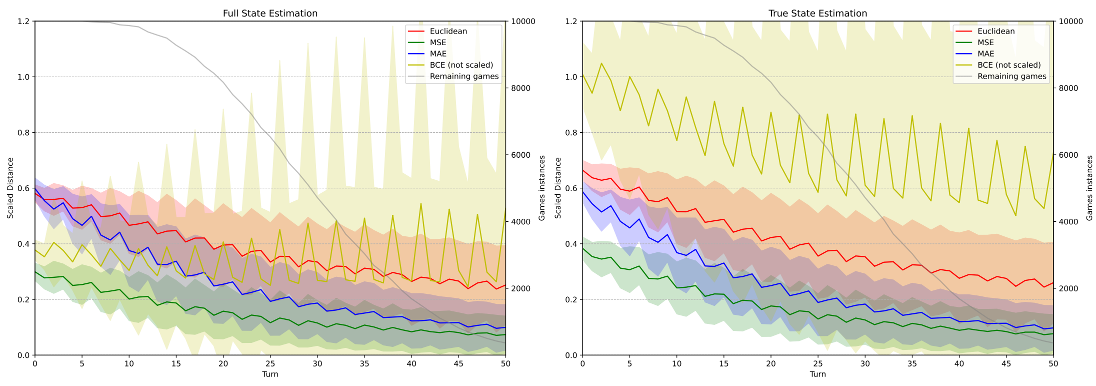


Figure 5.13: DouDizhu turn based estimations (PerfectDou)

using DMC; so much so that out of the 10000 testing hands we used for evaluation, none of the ensuing rounds went past turn 45. Hence behaviors observed on the latest turns can show a lot of variability as they are not based on a significant number of games. In any case we still observe the downward trend of the three first distance measures and the rising and subsequent falling of the BCE distances.

5.6 Impact Experiment

5.6.1 Information Configurations

To measure Imperfect Information Impact, we need to compare the performance of a player when they have cheated/perfect information to their performance with normal/imperfect infor-

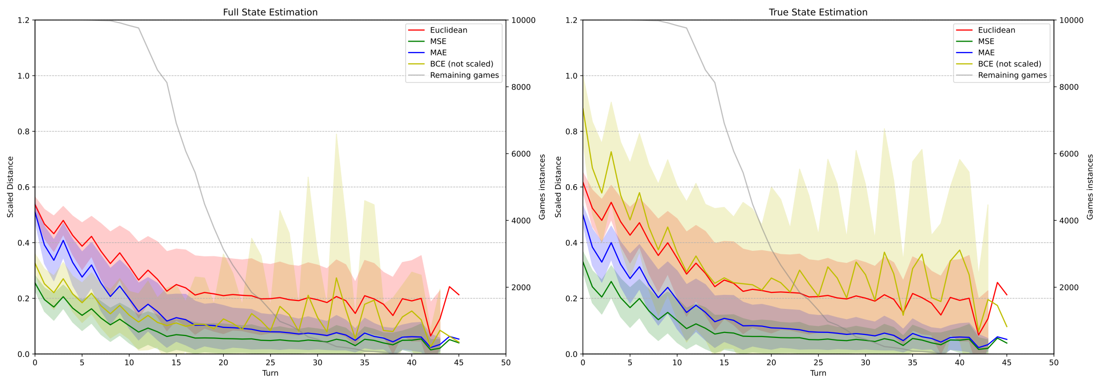


Figure 5.14: NoShuffle turn based estimations (PerfectDou)

Table 5.7: Information Configurations

Player	Ctrl	PL	PD	PU
L	I	P	I	I
D	I	I	P	I
U	I	I	I	P

mation. In order to do so, we consider four player configurations for each version of DouDizhu.

Mathematically, the Perfect Information agent receives a history instead of an infostate to judge the value of an action a_t

$$a_t \leftarrow \begin{cases} \arg\max_a V(h_t, a, e_t) & p(1-\epsilon) \\ \text{random action} & p(\epsilon) \end{cases} \tag{5.4}$$

Practically, we extend the basic input of the value network detailed in subsection 5.4.2 by adding the information of both other players' hands to it in the form of two flattened card encodings. To keep networks with same input dimensions, we simply replace the cheated input data with all zero inputs for the Imperfect Information agents during training.

Given the score distributions of these four agent configurations, we can then compare how much an agent's performance changes for each possible position.

5.6.2 Impact Measures

As defined in subsection 2.3.2, the way we measure imperfect information impact on a game is by comparing the difference in performance of a player when said player plays normally or has access to a cheated perfect view of the hidden information.

To do so, we need to get a sense of how much of a significant improvement a player achieves with this cheated information. Simply tracking average scores over multiple games would not be enough to do this properly. Naively, the simplest method would be to track the percentage or performance improvement for a normal and cheating version of the same player. The naive approach is obviously not ideal but we use it here to illustrate some of the issues of measuring imperfect information impact.

- **Close to zero scores:** In any game played by trained agents, one can expect the average scores of each player would tend to zero. Consequently any score increase gained by having perfect information could generate extreme improvement percentages that would not have much meaning.
- **Different game score systems:** Different games might have very different scoring systems. An average score difference of 1000 in one game might be less significant than an average score difference of 1 in another game.
- **Negative scores:** In some games, given position of play might be unfavored and even in ideal playing conditions would only yield a negative average score. Score improvements through cheated information would then yield either decreasing or negative percentages.

Overall these problems show that a measure based on pure score comparisons would not be ideal. Instead a better approach is to use statistical effect size measures to compare the score distributions of the normal and cheated information players.

Bellow are some of the ones we considered in this work.

- Standardized Mean Difference (SMD)
  - Cohen's D
  - Glass's Delta

- Pearson's R
- Eta Squared and Partial Eta Squared

Additionally we can confirm how significant these effect sizes are using p-value and f-value.

Assuming two random variables B and C for which we both sampled n times under the base player's score distribution and the cheating player's score distribution respectively the effect size measures are expressed as follows:

Standardized Mean Difference

SMD gives a representation of how far from one another two distributions are in terms of standard deviations. A value of 0.2 is considered a small effect size, 0.5 is medium and 0.8 is large.

$$\theta = \frac{\bar{c} - \bar{b}}{s} \tag{5.5}$$

Where $\bar{c}$ is the sampled mean and s is a standard deviation.

For Cohen's D, s is the pooled standardized deviation defined as follows:

$$s_{pooled} = \sqrt{\frac{(n-1)(s_c^2 + s_b^2)}{2n-2}} \tag{5.6}$$

And a group's variance is defined by:

$$s_c^2 = \frac{1}{n-1} \sum_{i=1}^{n} (c_i - \bar{c})^2 \tag{5.7}$$

For Glass's Delta, s is the standard deviation of the control group, in this case that of the normal player.

Pearson's R

This measures the linear relationship between two datasets. It ranges from 1 (completely correlated) to -1 (completely negatively correlated) where 0 indicates no correlation

$$r_{cb} = \frac{\sum_{i=1}^{n}(c_i - \bar{c})(b_i - \bar{b})}{\sqrt{\sum_{i=1}^{n}(c_i - \bar{c})^2}\sqrt{\sum_{i=1}^{n}(b_i - \bar{b})^2}} \tag{5.8}$$

ETA squared and Partial ETA squared

These two measure the portion of the variability explained by one or multiple factors. In our case, the only "independent variable" being the player having access to basic or cheated information, ETA squared and Partial ETA squared are mathematically the same. This measure can go from 0 (no effect) to 1 (full effect)

$$\eta^2 = \frac{n((\bar{c} - \bar{cb})^2 + (\bar{c} - \bar{cb})^2)}{\sum_{i=1}^{n}((c_i - \bar{cb})^2 + (b_i - \bar{cb})^2)} \tag{5.9}$$

Where $\bar{cb}$ is the average over both cheated score samples and base score samples.

5.6.3 Experiment

As with Visualization experiments, we trained our agents on a series of machines equipped with an Intel(R) Core(TM) i7-9800X CPU and two GeForce RTX 2080 Ti GPUs. We used python 3.7.0, pytorch 1.6.0 Cuda driver version 510.73.05 and cuda toolkit version 11.6. When training our agents, we did so with 10 actors divided over the two GPUs main process updated the network from the actors' samples. Twelve machines were used, one per cheated/normal information configuration and per game version, on which each training ran for 200 hours. On each of these, the algorithm remains the same with the only differences being the game environments and some of the input information dimensions in the case of Runfast. During training we saved images of our networks every thirty minutes as was done in the original DouZero papers [44, 46].

Once the training was done, we generated two sets (A and B) of 10000 starting hands for each of the three Doudizhu versions. We had each of the perfect/imperfect agent configurations play all of these starting hands and registered the scores obtained in each game. Note that we use the same starting hands as for the Visualization experiment but the ensuing gameplay can vary given that the agents are different.

Table 5.8: Average scores and Standard Deviations[a]

Game	Data	Config	L	D	U
DouDizhu	A	Ctrl	-0.05 (3.44)	0.03 (1.72)	0.03 (1.72)
	A	PL	0.30 (3.31)	-0.15 (1.66)	-0.15 (1.66)
	A	PD	-0.55 (3.37)	0.27 (1.69)	0.27 (1.69)
	A	PU	-0.72 (3.36)	0.36 (1.68)	0.36 (1.68)
	B	Ctrl	-0.09 (3.34)	0.05 (1.67)	0.05 (1.67)
RunFast	A	Ctrl	-1.14 (34.63)	1.27 (36.66)	-0.13 (35.33)
	A	PL	1.50 (36.07)	-0.82 (36.48)	-0.67 (34.95)
	A	PD	-1.90 (34.16)	3.59 (38.10)	-1.69 (34.84)
	A	PU	-2.32 (34.28)	0.29 (36.26)	2.03 (36.19)
	B	Ctrl	0.29 (36.58)	0.53 (36.37)	-0.82 (36.61)
NoShuffle	A	Ctrl	-30.91 (95.75)	15.46 (47.88)	15.46 (47.88)
	A	PL	-29.76 (95.49)	14.88 (47.74)	14.88 (47.74)
	A	PD	-45.83 (118.80)	22.92 (59.40)	22.92 (59.40)
	A	PU	-43.21 (91.54)	21.61 (45.77)	21.61 (45.77)
	B	Ctrl	-28.68 (107.11)	14.34 (53.56)	14.34 (53.56)

[a] Average (Standard deviation)

5.6.4 Results

Base Statistics

Table 5.8 shows some basic statistics for each of our configurations on each game. We have two control configurations for each game where all players are playing normally without cheated information, one tested on set A and one on set B.

For all of these configurations, it is important to note that the average scores for each player position, no matter the configuration, is much smaller than the standard deviation. Under such conditions, it immediately becomes clear that simply comparing averages it not enough to establish how much a player is impacted by a specific configuration which we will do in further subsections.

Nevertheless we can point out that there at least are differences in behaviors. In DouDizhu, the landlord's score is, on average, close to zero if slightly negative (-0.05 on set A and -0.09 on set B) under control conditions. This average does however seem to change significantly under configurations where one of the positions has perfect information. Each of the players seem to have a slight increase in score when a player on their team has perfect information and

Table 5.9: Impact measures (same starting hands)

Game	Config	Pearson's R	Cohen's D	Glass's δ	Partial ETA	P-value	F-value
DouDizhu	L	0.51	0.11	0.16	0.003	1.5e-13	54.69
	D	0.51	0.15	0.15	0.005	1.2e-24	105.33
	U	0.49	0.20	0.20	0.009	5.1e-43	189.97
RunFast	L	0.86	0.07	0.08	0.001	1.6e-07	27.81
	D	0.87	0.06	0.06	0.001	1.1e-05	19.32
	U	0.83	0.06	0.06	0.001	2.1e-05	18.15
NoShuffle	L	0.53	0.01	0.01	0.000	4.0e-01	0.72
	D	0.31	0.13	0.14	0.005	1.6e-22	95.62
	U	0.46	0.13	0.13	0.004	1.8e-20	86.20

a slight decrease in score when someone on the other team has perfect information no matter the version of DouDizhu. The main exception to this are the scores under the PL configuration in NoShuffle which sit between the average scores of both Control configurations.

Impact Measures

To be able to establish how significant of an advantage it is for a player to have cheated information, we use the measures discussed in subsection 5.6.2.

Comparing the score distribution achieved by each cheating configuration against the score distribution of the control we obtain the results shown in Table 5.9. In this table, all score distributions are obtained based on games played with the A set of starting hands. For each of the measures, we are comparing the score distribution of the player who has access to perfect information in a configuration against their score distribution under the control configuration. That is to say, in this table 5.9 and table 5.10, “L” configuration implies that scores of the landlord under PL information configuration are compared to their corresponding scores under Ctrl information configuration.

In table 5.9, Persons's R shows a high level of correlation between the scores of each cheater configuration with the control configuration. This makes sense as the starting hands can have a high impact on a player's chance of winning no matter which version of DouDizhu is being played, hence scores obtained from different configurations on the same starting hand a likely to be strongly correlated. While it is important to notice how much starting hands impacts

Table 5.10: Impact measures (different starting hands)

Game	Config	Pearson's r	Cohen's D	Glass's δ	Partial ETA	P-value	F-value
DouDizhu	L	-0.02	0.12	0.12	0.003	9.7e-17	69.16
	D	-0.01	0.16	0.13	0.005	7.6e-22	92.49
	U	-0.01	0.19	0.19	0.009	1.5e-39	173.94
RunFast	L	0.04	0.03	0.03	0.000	1.9e-02	5.51
	D	0.03	0.08	0.08	0.002	6.1e-09	33.82
	U	-0.01	0.08	0.08	0.002	3.3e-08	30.57
NoShuffle	L	-0.03	-0.01	-0.01	0.000	4.5e-01	0.57
	D	-0.03	0.14	0.15	0.006	9.2e-27	115.01
	U	-0.03	0.16	0.15	0.005	7.0e-25	106.40

scores, we also ran the experiment with a second set of starting hands for the control Configuration (Control scores on set B compared to cheating configurations on set A) in order to see if observed trends persist despite independent starting hands (shown in Table 5.10).

When looking at P-values in both tables, it is clear that having cheated information is significant ($p < 0.05$) for all configurations in all versions of the games except for the Landlord in NoShuffle. As pointed out earlier in subsection 5.6.4, there is little difference between the average score of the landlord under PL and Control configurations which shows in the resulting p-values. Seemingly, the NoShuffle game is so dominated by the two peasants that even having Perfect Information does not help them improve their score.

In no case do either of Cohen's D and Glass's Delta show values that classically would qualify as anything but a small effect. Both of these do however show consistent numbers across both Table 5.9 and 5.10 with DouDizhu having both SMD measures between 0.1 and 0.2 depending on positions, RunFast having all values below 0.1, and NoShuffle having close to 0.0 values for the Landlord and values above 0.1 for the Peasants.

Altogether this points to cheating having a impact but said impact also being mitigated by any score being highly reliant on the quality of a starting hand. Hence, while there is a true advantage to cheating (with the exception of Landlord in NoShuffle), that advantage is probably only exploitable in rare instances. If one looks at P-Values and the thresholds for Cohen's D as a proxy for impact measurement, it is arguable that most situations show significant impact but that it only plays a role in a small subset of games. Essentially, most of the time, a player has an optimal way to play their hand that would not change even if they knew what cards their opponents were holding. In some rare cases, they might benefit from knowing what cards their opponents held and could modify their optimal actions accordingly but this is overshadowed by the impact of the quality of one's hand in most cases. This could be considered a sub-quality of impact: **impact sparsity**.

Partial ETA as well shows small values for all cases (under 1%), which naively can be interpreted as a player being able to benefit from cheating in under 1 round in every 100. Again, while these measures are small, p-values show that they are still significant.

It is worth reminding that both DouDizhu and NoShuffle would normally both be preceded by a bidding phase which allows players to bid for the position as the Landlord depending on the cards they get at the start of the game. Giving players the ability to control what cards they play a position with could significantly impact their score distributions.

Seeing both SMD and partial ETA values together we can conclude that, while there is a clear impact of hidden information in DouDizhu type games, said impact is only a small one.

Chapter 6

Discussion

In this section, we would like to discuss the aspects of IIGs that we outlined in this work and come back on certain aspects of the results we presented.

6.1 Geister

Are different Geister versions fair or dominated?

From our results, we saw at least some level of indication that different versions of Geister are either dominated by the first-player or Draws

The main indications for Geister variants being dominated by the first-player are the asymmetrical gameplay and the difference in win rates seen in Table 4.4 and in Figure 4.3.

How large is Hidden Information Impact in Geister?

In smaller versions, hidden information has a high impact. This is caused by the restricted movement options due to the lack of space on the board. Because of this, it is most likely that a game will end in a capture move, thus making the hidden information impactful for most victories.

On the other end, we can surmise that the hidden information has a low impact in Geister variants that have a high escape rate. This is most apparent in variants x4y3g4b0 and x4y4g4b0 where approximately 50% of the agent's victories come by escaping. Escape is a win condition that inherently ignores hidden information, a high reliance on it implies that it is not necessary to use Imperfect Information to successfully play the game. Hence there is low Imperfect Information Impact in those variants.

Is there an indication that player have a representation of hidden information?
Variants of the game that only have two ghosts have low Imperfect Information visualization. Having only one ghost of each color on board makes any capture roughly a 50/50 win chance. While certain movements might give a slight hint to the color of a ghost, we can expect that, with two pieces, any such move could just as likely be a bluff. We do have to point out that our agent would not be able to pick up on such hints as it does only receives the most recent game state as input.

Games versions in which the agent shows a high propensity to let its opponent capture ghosts rather than attempting captures are also an indication that the agent has difficulties estimating which piece to capture and judges it safer to use red baiting. The agent essentially tries to avoid dealing with Imperfect Information by putting the onus on the opponent to manage it.

With the results of our experiments, there is no clear indication that any of the versions of Geister allow a player to get a good Visualization of Imperfect Information.

Is the high amount of draws in bigger games strategy or failure to learn?
There are two main reasons why we might be seeing such an amount of games ending in draws in bigger variants of Geister. Either the best outcome is to play for a draw in general unless a promising opportunity presents itself or the agent's network is not adapted to bigger versions of the game.

Given what we saw from some gameplay, the agent does not seem to behave optimally in bigger boards. Hence it seems our implementation did not adapt well to the bigger variants.

How can these concepts be quantified on other IIGs?
Geister has the distinct advantage of having multiple win conditions. Some of them depend on Hidden Information and some don't which makes it a good testing ground for hidden information related behaviors.

However, this is not something that can easily be done on any IIG. Even in this case, the analysis we provided is qualitative rather than quantitative.

To measure Impact and Visualization in other games would probably require different techniques. A way to measure Imperfect Information Impact would be to compare the performance of a trained agent when playing normally or a cheated version of a game where all hidden information is revealed. A similar idea has been applied by Shogo et al. [35] to measure the ability to visualize information on a given game. A prediction agent could be trained to give an estimation of the current hidden state of the game.

With the now qualitative ways to assess hidden information what would the results on Geister be?
It is too early to know with certainty at this point. Measuring visualization and impact on Geister could be a future work project. We can however expect that Geister played by efficient players would have low visualization measures as a good player would probably be good at obfuscating their hidden information. In Geister the only context cue to estimate a piece's color would be the piece's movement and where they are set at the start of the game but discovering the color of a specific piece does not give any hint as to what the color of another piece is. A good player could probably efficiently obfuscate these hints and thus make it difficult to estimate any part of the hidden information.

Given this hypothesis, one would expect the players to start putting more emphasis on escaping as it allows to win while mostly ignoring hidden information which would, in turn start giving some hints as to piece colors again which would not be good for the player trying to escape. From this we would expect to see more defensive strategies emerging, possibly making draws the most likely outcome in general.

In this context, impact would probably be high, as having access to cheated information could allow to break the hypothesized stalemate status quo.

6.2 DouDizhu

Are different DouDizhu versions fair or dominated? Given the score distributions we saw in our control conditions and prior research done in other papers, it seems that the base version of Doudizhu is dominated by the peasant team. This extends to NoShuffle whereas RunFast will automatically be a draw given that there is no positional advantage for each player given the random first player position.

It does however remain to be confirmed how including the bidding phase at the start of the game would impact this.

What would occur given progressive increase in cheated information rather than a binary one?

In DouDizhu and DouDizhu variants it is likely that results would not change significantly given that they all have low impact. It might however be interesting to witness if the impact measures increase gradually or in other ways.

Another version of this question asked was "What would occur if a player was given perfect information only after a certain turn in the game?" The answer to that question is probably that it would be almost equivalent to playing normally with hidden information. We saw from our results that the impact in these games is probably sparse. Hence, only in some key hands is there a strong advantage from getting perfect information when one shouldn't have access to it. Most likely, in these cases, whatever change to strategy from seeing other player cards there is, the change in strategy probably occurs near the start of the game and consists in changes in combinations that are played in order to not play to combinations where an opponent has a strong hand. Since, as the game progresses, the options in playing different combinations are reduced; there is less chance that having cheated perfect information would lead to strategy changes.

In any case, observing how the impact measures progress as the level of cheated information that is available to a player changes would probably yield interesting insights and is something we would like to experiment with in future research.

6.3 General

What are the limitations of these concepts?

The concepts we introduce in this work are very general ones to define IIGs. However IIGs encompasses a broad spectrum of terms and many sub-categories could be defined. Goodman et al.[18] attempts to classify a series of IIGs according to multiple variables and use a similar idea to ours in the sense of using trained agents to play the game and observe their results.

This approach makes sense as it is difficult to have logical models for optimal player behaviors in IIGs but it is dependent on agent performance and thus has a potential for bias.

How applicable are these concepts to other games?

In theory, impact measurements should be applicable to any game where a performance score can be given. As having a score function to optimize for is a basic requirement of any kind of machine learning it ensues that any game can be trained with can be used to measure impact. Some more abstract games like Dungeons and Dragons or other tabletop RPGs where the concept of reward is uncertain would probably create some issues.

Visualization measurements have somewhat stricter requirements. To be able to measure them, the hard condition is to be able to encode the hidden information into a binary format as was done here. The soft condition is to be able to calculate a maximum distance so as to be able to scale estimations.

Can these measures be compared between games?

Yes, the main idea behind many of the decisions taken in this work was to develop measures that can be used on as many game as possible and would represent the same concepts on each of these games.

Out of the different measures tested, which are the best to use?

Concerning the Visualization measures we tested, there seems to be little difference between Euclid, MSE and MAE. BCE having shown unstable behaviors it is automatically disqualified. In the end the best distance to use is probably the Euclid distance as it is the most instinctively understood one.

Regarding Impact, partial ETA is probably the best impact measure to use as it essentially gives a percentage of explained variability. Applied as we did in this work, partial ETA gives an easy way to understand impact as a percent based value. We should however keep using p-value as a quality check measure.

Both of these however are not final in any way. It is very likely that, new research into this topic coming, new measures would bring better representations of these two main concepts. Even in the context of this work, we found new possible measures to use that we did not yet have the time to test.

Chapter 7

Conclusion and Future Work

This thesis has investigated the influence of Hidden Information in Imperfect Information Games, first in a qualitative way and then in a quantitative way. We have attempted to introduce new concepts to help fix gaps in terminology used to discuss such games and proposed ways to measure and compare said concepts on as much games as possible. We conclude the thesis in this chapter.

7.1 Conclusion

Imperfect Information Games have been a focus of Machine Learning in recent years but most works have been focused on increased performance of algorithms on these games rather than anything else to the point that there aren't any generally accepted measures to quantify the challenge of these games yet.

In an attempt to fix this issue, we first went over much of the terminology and concepts used to discuss games and pointed to what the main differences of Imperfect Information Games, as opposed to Perfect Information Games, were and what their main challenge was. In doing this, we underlined that Imperfect Information Games is too unclear a term and further detailed our understanding of them through the concepts of uncertainty and what the possible sources of uncertainty in a game could be. We then further focused the reach of this work to Hidden Information. With all of this context established, we introduced and explained the two main concepts we propose in this work: Hidden Information Visualization and Hidden Information

Impact. The first of these two concepts focuses on the ability of a player to estimate the state of hidden information and the second deals with how much hidden information influences the score a player is able to get in the game.

Following the groundwork of establishing and discussing relevant vocabulary, and introducing our main concepts, we presented the two main algorithms, DREAM and DMC, used in this work to conduct case studies on Geister and DouDizhu.

We then detailed Geister as a game and discussed its interesting attributes, one of the main ones being the three possible win conditions reflecting the three main ways to manage hidden information. These three ways were either to ignore it altogether, or to weaponize it against an opponent, or to exploit your good ability to estimate it. We created multiple different versions of the game based on changing board sizes and number of pieces on which we then trained multiple agents using the DREAM algorithm. From this point, we analyzed player behaviors by observing tendencies in moving pieces and win conditions used on each version of the game. Overall, smaller versions of Geister presented clear tendencies to try to exploit hidden information against opponents, medium versions showed an increased tendency to ignore hidden information, versions with unbalanced number of pieces of each color had clear differences in capture behaviors showing at least some level of exploitation, and bigger versions tended to draw most of the time. From these observed behaviors, we developed the two concepts of impact and visualization which we then measured more scientifically in the following case study.

DouDizhu was used for the second case study. In it we established a more scientific approach to measuring impact and visualization and tested this approach on DouDizhu and two of its variants, NoShuffle and RunFast. We first explained the rules of all three versions and proceeded to explaining more details of how the DMC algorithm was used to create agents for each of these versions with a specific focus on the encoding of hidden information as it would be relevant for the measure of visualization.

After establishing the common ground for both measures, we focused on Visualization by presenting the difficulties associated with the task, how we dealt with them and the different distance measurements we used to compare between a player's estimation of a hidden state and the truth of that state. In total, we used four distance measurements (Euclid, MSE, MAE and BCE) and established that there was no strong difference in behavior between the first three while BCE, despite giving us some possibly interesting insights, proved too unstable to use as a main Visualization measure. We did see that there were indeed differences in Visualization for the three versions of the game and also depending on the player position in the game as well as clear progress in Visualizaiton as the game continues.

For the Measure of Impact, we first established how we measured this concept by comparing score distributions under different configurations of cheated (perfect) or non-cheated access to hidden information for each player position. We then selected different measurement methods used in meta statistical analysis to compare the score distributions obtained by a given player position in different information configurations. From this we were able to establish that, while there is a significant Impact of hidden information in DouDizhu and its variants, it is a small Impact.

From our results we surmised that DouDizhu is a game where hidden information mostly does not influence the strategy of a player as the policy a player uses is mainly determined by what cards they are dealt at the beginning of a game. There do however exist cases where knowing hidden information does have an influence on a player's decision making and can thus impact their score but this might only occur in less than 1% of situations.

Finally, we proposed that the best way to measure Visualization and Impact at this point is to use Euclid distance and Partial ETA respectively. We also reiterated that our methodology should be applicable to most games under some hidden encoding conditions and would also allow to compare any of these games on the basis of the influence of hidden information on them.

7.2 Future Work

Based on the results we have obtained in this work, we are confident that the study of how hidden information influences a game is a promising field; it is however a relatively ignored endeavor so far and has a lot of potential for improvement.

First, even though we have established a first methodology to measure the challenge posed by hidden information, we expect there is still much that can be improved with regards to the measures used. With regards to the use of distance measurements for Visualization, the specific measures we tested were very influenced by our background in Deep Learning and its loss functions. We do know that other fields use measures for similar problems such as CIELUV in colorimetry or Wasserstein distance which we were suggested could be adapted to our task. This ties in to one of the flaws of our current measures that don't consider that some failures to estimate can be considered less wrong than others (eg: estimating a player has four of a card in hand when they have three is less wrong than estimating they have none). Concerning Impact, our method seems very close to cheat detection which is a field that could also be looked into to find other ways to establish how much one's score is influenced by hidden information. Altogether there are many fields that have historically already dealt with similar tasks and from which inspiration could be taken to improve this work.

Second, so far our methodology has only been used on DouDizhu and its variants, it remains to be tested on other games. A first step to this would be to go back and measure visualization and impact on Geister and its variants to confirm that the behaviors we observed qualitatively are also possible to measure quantitatively with our methodology. Other games such as Poker would also be a good test bed. Probably some of the most interesting games to test our measurement methods on would be games where the hidden information does not lend itself to our required encoding as easily as it would probably yield insights into ways to improve the procedure.

Lastly, the results in this work are a stepping stone in the work of better understanding the challenges of Imperfect Information Games. We gave our interpretation of the results but more detailed experiments would be needed to further reinforce the validity of these interpretations. There are also multiple alternate ways to perform such analysis that we have not tested yet, such as analyzing how progressively providing more cheated information to a player would impact their score distributions.

Altogether, we expect that, if interest in this research catches on, we would rapidly be seeing new and better ways to quantify the influence of Imperfect Information on games.

Bibliography

[1] 1982 Spiel des Jahres Recommended | Board Game Honor | BoardGameGeek. https://boardgamegeek.com/boardgamehonor/8789/1982-spiel-des-jahres-recommended

[2] 1986 Årets Spel Best Family Game Winner | Board Game Honor | BoardGameGeek. https://boardgamegeek.com/boardgamehonor/9096/1986-arets-spel-best-family-game-winner

[3] Geister Online. https://geister.work/

[4] (2023). [NeurIPS 2022] PerfectDou: Dominating DouDizhu with Perfect Information Distillation. original-date: 2022-10-01T16:30:35Z. https://github.com/Netease-Games-AI-Lab-Guangzhou/PerfectDou

[5] Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., and Bowling, M. (2020). The Hanabi Challenge: A New Frontier for AI Research. *Artificial Intelligence*, 280:103216. arXiv: 1902.00506.

[6] Bowling, M., Burch, N., Johanson, M., and Tammelin, O. (2015). Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149. Publisher: American Association for the Advancement of Science Section: Research Article.

[7] Bowling, M., Johanson, M., Zinkevich, M., and Piccione, C. (2007). Regret Minimization in Games with Incomplete Information.

[8] Brown, N., Lerer, A., Gross, S., and Sandholm, T. (2018). Deep Counterfactual Regret Minimization. *arXiv:1811.00164 [cs]*. arXiv: 1811.00164.

[9] Brown, N. and Sandholm, T. (2018). Solving Imperfect-Information Games via Discounted Regret Minimization. *arXiv:1809.04040 [cs]*. arXiv: 1809.04040 version: 1.

[10] Brown, N. and Sandholm, T. (2019). Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890. Publisher: American Association for the Advancement of Science Section: Research Article.

[11] Chen, B.-N., Shen, B.-J., and Hsu, T.-s. (2010). Chinese Dark Chess. *ICGA Journal*, 33(2):93–106. Publisher: IOS Press.

[12] Chen, C. and Kaneko, T. (2019). Acquiring Strategies for the Board Game Geister by Regret Minimization. In *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 1–6. ISSN: 2376-6824.

[13] Chen, C. and Kaneko, Tomoyuki (2019). Utilizing History Information in Acquiring Strategies for Board Game Geister by Deep Counterfactual Regret Minimization. 2019:20–27.

[14] Chen, Chen and Kaneko, Tomoyuki (2020). Application of DREAM to the Board Game Geister. 2020:111–117.

[15] Davis, T., Schmid, M., and Bowling, M. (2019). Low-Variance and Zero-Variance Baselines for Extensive-Form Games. *arXiv:1907.09633 [cs]*. arXiv: 1907.09633.

[16] Elias, G. S., Garfield, R., and Gutschera, K. R. (2012). *Characteristics of Games*. MIT Press. Google-Books-ID: QVP8AQAAQBAJ.

[17] Gibson, R. G. (2014). Regret Minimization in Games and the Development of Champion Multiplayer Computer Poker-Playing Agents.

[18] Goodman, J., Perez-Liebana, D., and Lucas, S. (2021). Visualising Multiplayer Game Spaces. *IEEE Transactions on Games*, pages 1–1. arXiv:2202.05773 [cs].

[19] Hansen, E., Bernstein, D., and Zilberstein, S. (2004). Dynamic Programming for Partially Observable Stochastic Games. *AAAI Workshop - Technical Report*.

[20] Heule, M. J. H. and Rothkrantz, L. J. M. (2007). Solving games: Dependence of applicable solving procedures. *Science of Computer Programming*, 67(1):105–124.

[21] int8 (2018). Counterfactual Regret Minimization - the core of Poker AI beating professional players. Section: Poker AI.

[Jiang et al.] Jiang, Q., Li, K., Du, B., Chen, H., and Fang, H. DeltaDou: Expert-level Doudizhu AI through Self-play. *IJCAI*, (2019):1265–1271.

[23] Lanctot, M., Waugh, K., Zinkevich, M., and Bowling, M. (2009). Monte Carlo Sampling for Regret Minimization in Extensive Games. *Advances in Neural Information Processing Systems*, 22:1078–1086.

[24] Luo, Q., Tan, T.-P., Su, Y., and Jin, Z. (2022). MDou: Accelerating DouDiZhu Self-Play Learning Using Monte-Carlo Method With Minimum Split Pruning and a Single Q-Network. *IEEE Transactions on Games*, pages 1–12. Conference Name: IEEE Transactions on Games.

[25] Neller, T. W. and Lanctot, M. (2013). An introduction to counterfactual regret minimization. In *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, volume 11.

[26] Neumann, J. V. and Morgenstern, O. (1953). *Theory of games and economic behavior*. Princeton University Press, Princeton, [3d ed.] edition. Open Library ID: OL6132467M.

[27] Patashnik, O. (1980). Qubic: $4 \times 4 \times 4$ Tic-Tac-Toe. *Mathematics Magazine*, 53(4):202–216. Publisher: Mathematical Association of America.

[28] Romein, J. and Bal, H. (2003). Solving awari with parallel retrograde analysis. *Computer*, 36(10):26–33. Conference Name: Computer.

[29] Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers Is Solved. *Science*, 317(5844):1518–1522. Publisher: American Association for the Advancement of Science.

[30] Schmid, M., Burch, N., Lanctot, M., Moravcik, M., Kadlec, R., and Bowling, M. (2018). Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines. *arXiv:1809.03057 [cs].* arXiv: 1809.03057.

[31] Shioda, M. and Ito, T. (2021). Kyoto Shogi Is Weakly Solved. *The 26th Game Programming Workshop 2021*, 2021:42–45.

[32] Steinberger, E., Lerer, A., and Brown, N. (2020). DREAM: Deep Regret minimization with Advantage baselines and Model-free learning. *arXiv:2006.10410 [cs, stat].* arXiv: 2006.10410.

[33] Sturtevant, N. R. (2020). On Strongly Solving Chinese Checkers. In Cazenave, T., van den Herik, J., Saffidine, A., and Wu, I.-C., editors, *Advances in Computer Games*, Lecture Notes in Computer Science, pages 155–166, Cham. Springer International Publishing.

[34] Sutton, R. and Barto, A. (1998). Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

[35] Takeuchi, S., Tochikawa, J., and Matsuzaki, K. (2022). Impact of Imperfect Information Estimation in the Game of Geister. *Information Processing Society of Japan Proceedings*, 63(3):787–795.

[36] Tanaka, S., Bonnet, F., Tixeuil, S., and Tamura, Y. (2020). Quixo Is Solved.

[37] Tanaka, T. (2009). An Analysis of a Board Game "Doubutsu Shogi". *IPSJ SIG Technical Reports*, 2009-GI-22(3):1–8.

[38] Troillet, L. and Matsuzaki, K. (2021). Analyzing simplified Geister using DREAM. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, Copenhagen, Denmark. IEEE.

[39] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Wu, Y., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C.,

Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.*

[40] Wakatsuki, M., Kado, Y., Takeuchi, Y., Okubo, S., and Nishino, T. (2019). What are the Characteristics of the Card Game Daihinmin? In *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 587–592.

[41] Wang, H., Wu, H., and Lai, G. (2022). WagerWin: An Efficient Reinforcement Learning Framework for Gambling Games. *IEEE Transactions on Games*, pages 1–11. Conference Name: IEEE Transactions on Games.

[42] Yang, G., Liu, M., Hong, W., Zhang, W., Fang, F., Zeng, G., and Lin, Y. (2022). PerfectDou: Dominating DouDizhu with Perfect Information Distillation. arXiv:2203.16406 [cs].

[43] Yu, X., Wang, Y., Qin, J., and Chen, P. (2023). A Q-based policy gradient optimization approach for Doudizhu. *Applied Intelligence*, 53(12):15372–15389.

[44] Zha, D., Xie, J., Ma, W., Zhang, S., Lian, X., Hu, X., and Liu, J. (2021). DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning.

[45] Zhang, Y., Yan, D., Shi, B., Fu, H., Fu, Q., Su, H., Zhu, J., and Chen, N. (2021). Combining Tree Search and Action Prediction for State-of-the-Art Performance in DouDiZhu. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 3413–3419, Montreal, Canada. International Joint Conferences on Artificial Intelligence Organization.

[46] Zhao, Y., Zhao, J., Hu, X., Zhou, W., and Li, H. (2022). DouZero+: Improving DouDizhu AI by Opponent Modeling and Coach-guided Learning.