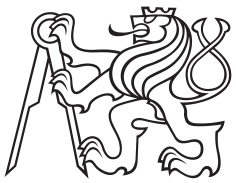
Bachelor's thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Playing General Imperfect-Information Games Using Game-Theoretic Algorithms

Jakub Černý

Program: Open Informatics
Field: Computer and Information Science

May 2014
Supervisor: Mgr. Branislav Bošanský

Acknowledgement / Declaration

I would like to thank my supervisor, Mgr. Branislav Bošanský, for giving me the opportunity to work on this project, for his patient guidance, much appreciated enthusiastic encouragement and frequent helpful critiques, as he introduced me to the art of technical writing.

I would especially wish to thank my amazing family for the love, support, and constant encouragement I have gotten throughout my study. In particular, I would like to thank my parents and my sister.

I also wish to acknowledge the help provided by Hayk Divotyan, Tomáš Flek, Jan Kleňha and Jan Zdeněk, my friends who I continuously asked for opinions on individual chapters of the thesis and they contributed by many valuable insights.

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

.....
Jakub Černý

In Prague, May 22, 2014

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

.....
Jakub Černý

V Praze, 22. května, 2014

Abstrakt / Abstract

Většina herních algoritmů je specificky navrhována tak, aby byly schopny hrát velmi dobře jednu hru. Tyto programy není možné použít pro hraní her jiných. Naproti tomu je hraní obecných her (tzv. GGP – general game playing) konceptem, který předpokládá hraní velkého množství her v jednotném herním prostředí. Tento vznikající obor všeobecné umělé inteligence dobře slouží jako vhodná testovací platforma pro výzkum doménově nezávislých algoritmů výpočetní teorie her. Pokud bychom dokázali navrhnout algoritmus schopný hrát úspěšně více než jednu hru, pak by takový systém mohl být použit v případech, které si žádají obecné nezávislé rozhodování v reálném čase, např. během řízení pátracích a záchranných misí nebo v navigaci dálkových autonomních robotických jednotek po neznámém území.

Hlavním záměrem této práce je zaměřit se na výběr dedukčního algoritmu schopného rigorózně interpretovat různé hry a použít herní algoritmus, který poskutuje dostatečné teoretické záruky pro řešení her s neúplnou informací. Hlavním výstupem této práce je Shodan, jeden z mála GGP hráčů schopných hrát hry s neúplnou informací. Shodan je postaven na algoritmu prohledávání herních stromů Monte Carlo (Monte Carlo tree search) s výběrovým kritériem EXP3, a pracuje s hrami představovanými výrokovými sítěmi.

Klíčová slova: Hraní obecných her; extenzivní hry; hry s neúplnou informací; Monte Carlo prohledávání; EXP3.

Překlad titulu: Hraní obecných her s neúplnou informací pomocí algoritmů výpočetní teorie her

Most game playing algorithms are specifically designed to play one single game. Therefore, transferring such programs into another context is not possible. On the other hand, general game playing (GGP) is a concept of playing vast number of games within a one concrete game environment. This emerging field of general artificial intelligence proficiently serves as a challenging testbed for research in domain-independent algorithms of computational game theory. If one can design a general game playing system capable to play more than one game successfully, such algorithm can be used in other areas which require a real-time domain-independent deciding, such as in providing intelligence for search and rescue missions or navigating remote autonomous robotic units in unknown territory.

Uppermost intention of this work is to focus on selecting the reasoner capable to properly and rigorously interpret the game; and employing the algorithm which meets the theoretical guarantees to work proficiently in the imperfect-information games. The most prominent outcome of this thesis is Shodan, one of the few GGP agents for playing general games with imperfect information. Shodan is based on Monte Carlo tree search with EXP3 selection criterion, working with the games represented as propositional networks.

Keywords: General game playing; Extensive-form Games; Imperfect-information Games; Monte Carlo tree search; EXP3.

Contents /

Tables / Figures

Chapter 1
Introduction

Since the origin of computer science, the conception of a machine playing a game against a human has become one of the primary goals of AI research. In those days the foundations of game theory have already been laid, thus providing sufficient methods for computing rational responses in elementary games.

The first computer capable to play the game of Nim was created in 1941 at MIT [1] and can be considered as an early AI agent. In the following years, several players in games like chess or checkers were programmed, but they lacked the intellect to be a worthy opponent even for a non-professional player. They didn't actually become successful in small games against human amateurs until 1960'. As the computational power of computers arose, agents became intelligent enough to solve even more complex games with large state space, including backgammon [2] or ghost[1]). However, such ability was granted not only by 'brute force' algorithms like backward induction, but also due to numerous optimizations in the way the players searched the game tree, reinforcement learning and insightful heuristics based on deep knowledge about the game itself. In conclusion, the improvement of these expert systems in late 80' finally led to designing the famous computer Deep Blue [3], challenging world best chess player between years 1996 and 1998.

Nevertheless, progress in this field demonstrated that better performance in game playing requires focusing on the narrow task. These so-called weak AI systems are being designed to solve one specific problem only, hence the transformation into another context is not possible. In other words, an agent written to play chess is simply not able to participate in the game of Go.

Unlike in applied AI systems, artificial general intelligence (AGI) aims at performing broad range of intelligent tasks, instead of excelling at a single one. General game playing (GGP) is a concept of playing vast number of games within a concrete game environment with just one agent, which makes it a part of AGI in the field of games [4]. This concept was originally introduced by J. Pitrat back in 1968, who pointed out that the intelligence of AI researchers is actually the reason why the AI systems are that dumb [5]. The main principle of playing general games is that the rules of each game are not known before the match starts. It means, that the player cannot rely on the preprogrammed domain-dependent heuristics closely related to the game itself. The goal is to create an agent capable of recognizing specific aspects of individual games and their incorporation into a general solving algorithm [6].

Many processes in the human society can be efficiently formalized as games. For example politics, economics or social interactions, among other things, form the partial games in the immense environment of the human world. Building a reasonable general deciding algorithm can help solve some of the important problems in these areas and find several interesting common elements. However, while e.g. in the game of Shogi the game board is fully observable for each player, a tremendously important aspect of many genuine real-world problems is the fact that numerous features of the game

[1]) `http://en.wikipedia.org/wiki/Ghost_(game)`

environment may not be known. Such problems can include, but are not restricted to, the examples like [7–8]:

- *Financial Markets.* In financial markets the borrower possesses much better information about his finances than the potential lender. From the lender's point of view, the likelihood the borrower will bring back the money is uncertain. The lender might try to solve this situation by looking at past credit history and evidence of salary, but this can provide only partial information. Consequently, the lenders will charge higher rates to compensate for the risk.
- *Monetary Policy.* The individuals react according to the monetary policy of a certain country, which influence the inflation and growth. In response to a change in the exchange rates, a central bank has to decide about the money supply and thus alter its monetary policy.
- *Insurance.* When insuring a certain property the insurer has only limited information about how well the customer will look after a piece of good. To overcome this uncertainty in insurance, insurers provide considerable discounts for 'no claims bonuses'. This is the best way of gaining better information about 'careful' and 'unlucky' clients. The fact is, insurance may even alter a person's behavior.
- *Entry into Markets.* The companies make decisions whether to enter a particular product market, which products to promote and how to set the prices. They know their own cost, but not the costs of their rivals. Such decisions are most likely to influence a future development of the market.
- *Employment.* Workers are aware of their skills, industriousness, and productivity. Conversely, employers face a lack of information about the qualities of prospective workers. Usually the employers first sign the workers to a contract, the bonuses or the promotions are then decided after the employer is more acknowledged about the worker's skill.
- *Research and Development.* The companies have to decide, how much will they invest into the research and the technologies. These choices are made according to the current knowledge about the market and similar projects of other competitors, so as the forecast about the future trends.

Agents in these game examples can possess only limited amount of information, making their situation even more challenging. Their decision are made under uncertainty, trying to maximize an expected potential profit.

Games of imperfect information form one part of a broad range of games with the incomplete information. In these games the player is required to be aware of the whole game description, but as the game progresses, he is not capable of identifying the actions performed by his opponents. The main goal of this thesis is to create a general deciding system which is able to make reasonable decisions in games of imperfect information and under a strict time limit.

1.1 Related work

General game playing is an emerging AI field in which the research is more deeply conducted for the last 8 or 9 years. The research groups are concentrated on several universities, primarily on Stanford, University of New South Wales, University of Alberta and University of Reykjavik. So far more than 150[1]) publications were published and the community regularly gathers on a workshop at IJCAI conference since 2009.

[1]) According to the list at `http://general-game-playing.de/literature.html`

There have also been annual General Game Playing competitions held at the AAAI Conference. Although main stream of research is focused in the field of games with complete information, the incomplete-information games are becoming the center of interest more recently.

Even though the community is relatively small and one cannot expect it to produce a large amount of textbooks for the newcomers to this field, a great introduction into the theory of general game playing was written by Michael Genesereth and Michal Thielscher [9]. The book covers the vast background of GGP, including game design and specification, reasoning about game rules, GGP server protocols, and even the basics of efficient game playing. Although it focuses on the perfect-information games, it is an excellent start and I am grateful that I could use it.

Competitions between an imperfect-information GGP agents have been organized since 2011. The first one took place in Berlin, Germany, in conjunction with the KI 2011 conference. The German Open in General Game playing[1]) was organized by Peter Kissmann and Tim Federholzner of the University of Bremen and embraced the tournament in playing general imperfect-information games as well. Three agents participated in the contest.

First player (and the winner) is called Fluxii, programmed by Stephan Schiffel from Reykjavík University. The player was derived [10] from the original perfect-information game-playing agent FluxPlayer [11]. For reasoning about possible actions the player uses Fluent calculus and its Prolog-based implementation FLUX [12]. The search is then performed using non-uniform depth-first search with iterative deepening and general pruning techniques. The player is able to recognize structures in game description for automatic construction of evaluation functions. Non-terminal states are then evaluated by the rules of fuzzy logic by the constructed heuristic. Second player named StarPlayer (UNSW) is a project by Timothy Cerexhe, Michael Schofield and Michael Thielscher from University of New South Wales. Although it was not stated anywhere, I suppose the player was based on HyperPlay technique adapted for use with a Monte Carlo decision making process, introduced by the same authors [13]. Hyper-Play technique is able to translate the description of an imperfect-information game into the format which is acceptable by the players of perfect-information games. The state space of the imperfect-information game is sampled to resemble a game with perfect information. Individual rounds are played in each of these games and then the separated results are assembled together again.

Finally, the last player was TIIGR, an agent designed by Tomáš Motal from Czech Technical University in Prague as a part of his Master's thesis [14]. The player is based on Monte Carlo tree search using the UCT selection criterion and overconfident approach. Palamedes[2]) JavaProver is used for reasoning about the game specification.

Second competition among several players capable to play games with imperfect information was carried out during the AI'12 General Game Playing Competition[3]), organized by Marc Chee and Michael Thielscher of the University of New South Wales. The tournament was held in conjunction with the AI'12 conference and was attended by 3 players.

The winning player was CadiaPlayer by Stephan Schiffel, Hilmar Finnsson, Stefán Freyr Gudmundsson and Yngvi Björnsson from Reykjavík University [15]. The deciding core of the CadiaPlayer is based on modified Monte Carlo tree search with the UCT

[1]) `http://fai.cs.uni-saarland.de/kissmann/ggp/go-ggp/`
[2]) `http://palamedes-ide.sourceforge.net/`
[3]) `http://ai2012.web.cse.unsw.edu.au/ggp.html`

criterion and Gibbs sampling [15]. The player benefits from several extensions of MCTS, such as its ability to prematurely terminate the simulations when the result seems unreliable; or to apply a speculative knowledge. CadiaPlayer managed to win the competition mainly thanks to his ability to process even a more complicated games and continue playing in complex situations.

Second, according to the score, was Nexusbaum by Tim Federholzner and Stefan Edelkamp from TZI University of Bremen; and Peter Kissmann from Saarland University [10]. Their player operates on partial belief states, which correspond to a subset of the set of states building a full belief state. To search for a partial belief state the agent uses the UCT-based Monte-Carlo method.

Third player was LeJoueur by Jean-Noël Vittaut, but it is stated here just for completeness – I couldn't find anything more specific about this particular one.

These are the only successful attempts to build and examine a general imperfect-information player I am aware of and which have served as an inspiration to me. However, all these approaches have not proved to converge to a solution, which is theoretically guaranteed to be the contemplated one.

1.2 Approach of this thesis

This thesis aims at designing a game-playing algorithm based on one of the state-of-the-art algorithms, which have been introduced over the last years. In the following chapters, the thesis first presents the backgrounds of general game playing, focusing mostly on theoretical basics of game theory. Then it discusses various possible approaches in building the whole GGP agent playing the games of imperfect information from scratch, its advantages and disadvantages and technical aspects of their implementation.

Uppermost intention of this work is to focus on selecting the reasoner capable to properly and rigorously represent the game and apply the algorithm which meets the theoretical guarantees to work proficiently in the imperfect-information games.

The design of the player is finally examined over the real games of imperfect-information and against another player, specifically TIIGR and a simple random player.

1.2.1 Overview

The thesis is organized in the following structure:

- In Chapter 1 is stated the motivation for writing this thesis, the work related to its topic and the chosen approach.
- The game theory provides the necessary theoretic background for playing general games. The basics of its mathematical formalism are formulated in Chapter 2.
- Chapter 3 presents various state-of-the-art algorithms for finding sensible responses in games; and their properties.
- The introduction to GGP is given in Chapter 4. This chapter also includes the methods for reasoning about game descriptions and learning in general games.
- The process of building a working GGP player and the description of player Shodan are presented in Chapter 5.
- Chapter 6 contains the experimental results achieved with the player.
- Finally, the general information about the achievements of this thesis is announced in Chapter 7, which also includes the possibilities in the future development of this player.

Chapter 2
Game Theory

In this chapter are laid out the foundations of game theory. At the beginning it formalizes the basic definitions, which are necessary to be able to correctly speak about games and game-plays. Consecutively it presents the standard representations and data structures. Finally, this chapter summarizes relevant aspects of games. The background in game theory is essential for finding rational responses and also for general reasoning about games.

A mathematical formalization of game theory in this chapter is inspired by [16].

2.1 Introduction

Game theory is a part of applied mathematics that studies a strategic decision making. It uses mathematical models to formulate interactions between intelligent rational decision-makers [17]. These interactions are called games.

2.1.1 Game

Games are played within a game environment[1]) (also called world) and are composed of system of rules, which defines *the players*, *the actions* and postulates the dynamics of *the game*. The game is called a puzzle, if there is no more than one agent involved. Otherwise it is a conflict [18].

Definition 2.1. Player *A player (or an agent) is an entity able to act. His activities alter the world in which he exists.*

The concept of game consists of active and passive elements. Passive elements represent the information, i.e. which actions are feasible for a particular agent in a given state, or how the game will evolve under certain conditions and actions taken. Active elements in the game form the players. Without the players, the game remains static. Only their actions can manipulate the game.

Definition 2.2. Action *An action (or a move) is a change in the game caused by a player in a particular situation.*

A valid game environment enables all agents to act and be immediately aware of their actions. Their activity can lead to changing current situation as a consequence of their decision making. Different situations which can occur before the game terminates are called states of the game.

[1]) The difference between games and game environments is sometimes omitted. Although, it is useful to distinguish them, especially in the context of general game playing. This problematics is further explained in chapter 4.

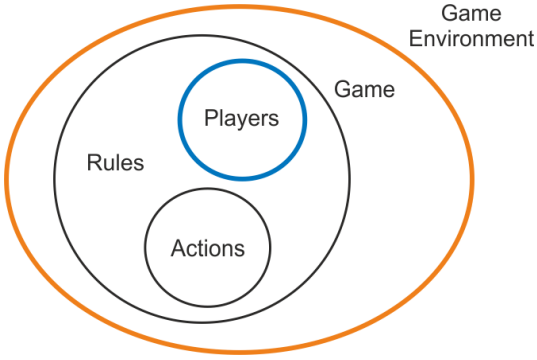


Figure 2.1. Games are played within a game environment.

Every game begins in a root state and then progresses according to the game dynamics, as participating agents make their decisions. All rational players select their actions to achieve their goals. Theory of utility was established to recognize the effects of their behavior and evaluate the situations in which the agents are located. Utility is a value which measures the usefulness of the current state of the game for each player.

Definition 2.3. Utility *Let S be a set with weak ordering preference relation $\preceq$. Utility (or outcome) is a cardinal element $e \in S$, representing the motivation of players. The function u: $S \to \Re$ is said to be utility function IFF $\forall x,y \in S$: $u(x) \leq u(y) \Leftrightarrow x \preceq y$.*

All together, a mathematical game is a structure, which conclusively defines the whole game and its development.

Definition 2.4. Game *Game is a tuple $G = (P, A, u)$, where:*

- *$P = \{p_1, p_2, \ldots, p_m\}$[1]) is a set of players;*
- *$A = \{A_1, A_2, \ldots, A_m\}$ is a set of sets of available actions for each player; and*
- *u is a utility function, u: $(a_1 \in A_1 \times a_2 \in A_2 \times \ldots \times a_m \in A_m) \to \mathbb{R}^m$.*

This general definition of game expects all players to act simultaneously in just one round and then it ends. Nevertheless, the end of a game in finite time is guaranteed only in the so-called finite games. It signifies that at some point they will terminate and the utilities are assigned. All finite games have starting and terminal states. In these games the number of players is finite, as well as the number of permitted actions for each player. An agent can face only finitely many situations in finite game, and the game-play cannot go on indefinitely [19].

2.1.2 Agents' strategies

When there is more than a single agent in the environment, the whole game changes in accordance to the activity of all players. In this setting the outcome depends not only on actions of one particular agent, but on the behavior of all of them. Strategies can be seen as plans contingency or policy for playing the game. In every situation, agent's reaction is defined by his strategy.

A strategy of an agent in the game is said to be pure, if in every possible attainable situation an agent can face, his strategy determines the one move the agent will make. For example in Figure 2.2 the set of pure strategies of agent Kevin is a set $\{(grab, feed); (grab, bask); (hold)\}$.

[1]) A set P with respect to player p_i is sometimes denoted as P = $\{p_i, p_{-i}\}$, where $p{-}i$ denotes a set P without player p_i.

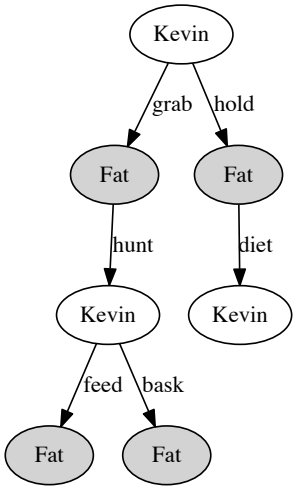


Figure 2.2. A possible development of a simple game.

This approach is certainly rational enough in puzzles, where there is only one agent to set the course of the world. In contrast, in the environments with greater number of other players it is preferable to rather randomize over the set of pure strategies, following selected probability distribution.[1]) This kind of strategy is called mixed. Playing a mixed strategy ensures that every agent can only guess what will happen; and compared to the pure strategies, the outcome is now less predictable.

Definition 2.5. Mixed strategy *Given a game $G = (P, A, u)$, for agent p_i a set of all pure strategies $\Pi(p_i)$ and denote $\Delta(Z)$ a set of all possible probability distributions over arbitrary set Z. Then $\Sigma(p_i) = \Delta(\Pi(p_i))$ is a set of mixed strategies for agent p_i.*

It is obvious, that even if the player can follow finitely many pure strategies, because of the continuity of probability distributions the set of mixed strategies is always infinite. Furthermore, a set of agent's pure strategies is a subset of his mixed strategies.

Playing the mixed strategy, the player can gain a various range of outcomes. To evaluate his strategy, he can use the expected payoff.

Definition 2.6. Expected utility of mixed strategy *Let $G = (P, A, u)$ be a game and σ be a list of mixed strategies for all players. Then expected utility for player p_i is $u_i(\sigma) = \sum_{a \in A} u_i(a) \prod_{j=1}^{|P|} \sigma(p_j, a_j)$.*

By $\sigma(\mathrm{p}_i,\mathrm{a}_j)$ is denoted a probability of player p_i taking action a_j.

2.1.3 Optimal strategy

The whole game theory was originally established to solve a simple question. What is an optimal reaction? How should an agent react to be the most likely to win the game? The answer is that the fundamental advantage for a player can be an information about the strategies of his opponents. In other words, once an agent is able to guess the next

[1]) This concept is being disputed since its origin in 1980s, because it lacks a reasonable behavioral support [20]. Sometimes rather than a strategy, randomizing the decisions can be seen as a belief of an agent, that he can profit from playing such action [21].

action of any other agent, he can deliberately follow a strategy which maximizes his terminal utility. In conclusion, the set of all optimal strategies (meaning the strategies with the highest equal expected utility) $\Sigma^*(\mathrm{p}_i) \subseteq \Sigma(\mathrm{p}_i)$ of a rational well-informed agent p_i is then absolutely decided by the strategies $\sigma(\mathrm{p}_{-i}) \in \Sigma(\mathrm{p}_{-i})$ of the others.

Definition 2.7. Best response *Agent's strategy $\sigma^*(p_i)$ in game $G = (P, A, u)$ is a best response to strategies $\sigma(p_{-i})$ IFF $\forall \sigma(p_i) \in \Sigma(p_i)$: $u_i(\sigma(p_i),\sigma(p_{-i})) \leq u_i(\sigma^*(p_i),\sigma(p_{-i}))$.*

Unfortunately, in most cases the information about the opponents' strategies is out of reach, or obtaining is impossible in sense of computational complexity. Another possibility would be to estimate the strategies, e. g. from the previous actions of other players, and consecutively adjust his own one.

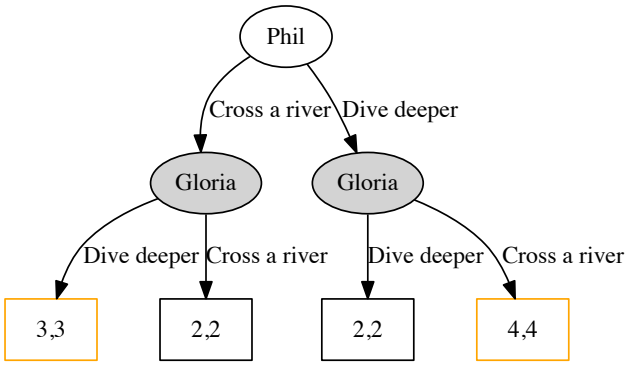


Figure 2.3. A possible development of a simple game with highlighted Nash equilibrium.

As every player intends to do his best to achieve victory and considers the decision-making of his opponents, behavior of all agents playing the same game over and over again is evolving. At a certain point of finding their best responses, players realize that changing their strategy would not lead to earn more than with their current decision plan. This concept of balance is called an equilibrium.

Definition 2.8. Nash equilibrium (NE) *Given a game $G = (P, A, u)$ and strategies $\sigma \in \Sigma$, players P are in Nash equilibrium IFF $\forall i \in [1,\ldots,|P|]$: $\sigma(p_i)$ is a best response to $\sigma(p_{-i})$.*

If the stage of the world allows no one to benefit from changing his strategy, the situation remains stable. It has been proved, that in every game with finitely many players and with finite set of pure strategies, there is at least one Nash equilibrium profile, although it might consist of mixed strategies [22].

An example of two Nash equilibria is depicted in Figure 2.3. It can be seen, that there are two sets of strategies, which produce the balance. Playing these strategies, neither Phil nor Gloria shall intentionally change his opinion. It would only decrease their final utility.

2.2 Game representations

There is a number of various representations of games. The most simple one was presented at the beginning of this chapter. Although the general definition is sufficient

enough for the mathematical apparatus, for concrete game examples it is more convenient to establish standard forms and structures for working with the game data. Different representations extend the general definition, thus allowing various games to express their specific aspects in more suitable form. Algorithms for finding Nash equilibria can be adapted to a particular representation to reduce computational complexity.

There exist several representations of games, taking into account stochasticity, number of players and decision points, possibility of cooperation and other important characteristics of the game. Presented forms are two fundamental non-cooperative representations, where every player aims to maximize his own utility.

2.2.1 Normal form

Normal (or strategic) form is a basic type of game representation, G = (P, A, u). Each player moves once and actions are chosen simultaneously. This makes the model simpler than other forms and easier to solve for Nash equilibrium, but lacks any temporal locality.

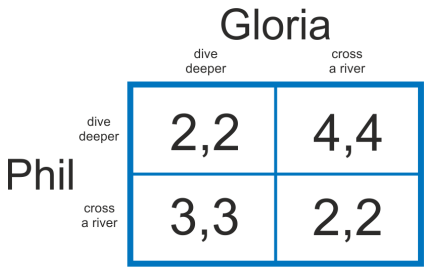


Figure 2.4. An example of a game in normal form.

Utility function in normal-form games is usually visualized as a payoff matrix. For example, consider a game presented in Figure 2.3. It is an example of a 2-player game in which every player has 2 possible actions. Its representation in normal form is depicted in Figure 2.4.

2.2.2 Extensive form

Extensive form models a multi-agent sequential decision making. Convenient representation of an extensive-form game is a game tree. Such structure allows to express even complicated branching of the game, restricting actions in different game states to the feasible ones only.

Definition 2.9. Game tree *Every game tree is a tuple $T = (S, Z, A, e, f, r)$, where:*

- *S is a set of game states;*
- *Z is a subset of S of terminal states;*
- *A is a set of game actions;*
- *e is an expander function, $e : s \in S \to \{a \in A \mid a \text{ is executable in } s\}$;*
- *f is a successor function, $f: (s \in S \times a \in e(s)) \to t \in S$; and*
- *$r \in S$ is a root state.*

In a sense of graph theory, S are the vertices of rooted directed tree T, Z are the leaves of T and elements of A form its set of edges.

Using the notion of a game tree, now it is possible to define an extensive-form game. This representation consists of a game tree with a set of players, who are assigned to the states of the tree; and a utility function, which determines the utility in every terminal state, i.e. in every leaf of the game tree.

Definition 2.10. Extensive-form game *Game in extensive form is a tuple $G = (P, T, b, u)$, where:*

- *$P = p_1, p_2, \ldots, p_m$ is a set of players;*
- *T is a game tree $T = (S, Z, A, e, f, r)$;*
- *b is a player belonging function[1]) $b: s \in S \rightarrow p \in P$; and*
- *u is a utility function $u: z \in Z \rightarrow \mathbb{R}^m$.*

Extensive form was originally designed for sequential games, where players take their actions one by one. Game trees in these games provide a suitable way to visualize the game-play. This representation is also more complex than normal form. It is able to describe games with chance nodes or imperfect information more conveniently.

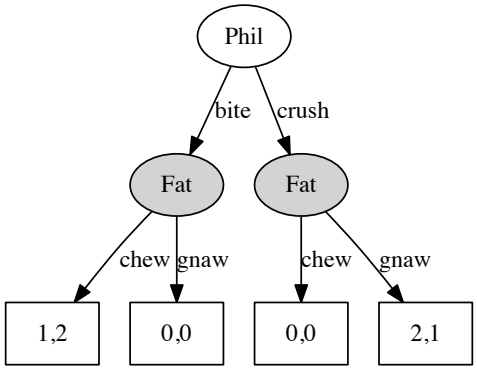


Figure 2.5. An example of a game in extensive form.

Although normal and extensive form are suitable for defining different kinds of games, it is possible to convert one form into another [23]. It is guaranteed that for every extensive-form game exists an equal corresponding normal-form game, however the procedure of transformation can be computationally impractical[2]). The mapping is not injective [24], meaning that multiple extensive form games can correspond to the same normal form.

2.3 Decisive game aspects

Games can be divided into several categories according to some of theirs prominent properties. Each class in some way specifies the game and its properties significantly influence the game-play. If an agent is aware of it and is able to recognize and utilize all factors of the game, he can use it for his own advantage.

2.3.1 Utility

The utility of terminal states of the game is a dominant aspect, which affects the way agents behave in the environment. The goals of the agents can be independent, i.e. by achieving his goal, a player p_i won't affect the possibility of other players to achieve theirs. Conversely, in some games to gain more, opponents are required to lose

[1]) Usually $b(r) = p_1$, unless stated otherwise.
[2]) More precisely, it can lead to an exponential blowup.

more. From the economical point of view, it is a clear example of preserving amount of resources, which is at one time always constant.

Definition 2.11. Zero-sum game *The game $G = (P, A, u)$ is said to be zero-sum IFF* $\forall a_1 \in A_1, \ldots, a_{|P|} \in A_{|P|} : \sum_{i=1}^{|P|} u_i(a_1, \ldots, a_{|P|}) = 0$.

When playing a zero-sum non-cooperative game, each player knows that in every terminal state, his utility is dependent on the utilities of his opponents. As a result, the optimal strategy of the player necessarily profits from exploiting the strategies of his opponents. By using their weaknesses the player can gain more.

A variant of a zero-sum game is a constant-sum game. In constant-sum games the sum of utilities for respective players is equal to a constant. Since the outcomes of the player can always be normalized, the constant-sum game can be represented by an equal corresponding zero-sum game[1]).

However, there exists a class of games in which there is no correlation between the utilities of the players like in the zero-sum games. For example, the trade is by definition a positive-sum game, because when two players (in this context companies or traders) agree to realize an exchange, each subject must consider the received merchandise to be more valuable than the merchandise traded off. The profit of both parties will differ, when they use a different amount of goods. This is a clear example of a general-sum game.

Definition 2.12. General-sum game *The game $G = (P, A, u)$ is said to be general-sum IFF* $\exists a_1, a'_1 \in A_1, \ldots, a_{|P|}, a'_{|P|} \in A_{|P|} : \sum_{i=1}^{|P|} u_i(a_1, \ldots, a_{|P|}) \neq \sum_{i=1}^{|P|} u_i(a'_1, \ldots, a'_{|P|})$.

It means the game is general-sum, if there exist 2 action profiles leading to terminal states with non-equal sum of utilities per each player.

2.3.2 Move sequencing

The way the players are making their moves is also an important aspect. In simultaneous-move games the players effectively take their actions at the same time. Actually, they don't have to make the moves at the exactly same time as long as each of them is not aware of the others' choices when he chooses his own strategy. In these games players select their moves only by evaluating the utility of possible terminal states. Slightly different are repeated simultaneous-move games with multiple turns, in which agents can assess opponents' patterns of behavior from their previous activity. Simultaneous-move games are more conveniently represented in the normal form.

An example of the simultaneous-move game is the Prisoner's Dilemma [25]. In this game two imprisoned criminals decide, whether they should betray the other prisoner or try to cooperate and stay quiet, without the prior agreement between the two of them. Although the prisoners might not be interviewed at the precisely same moment, they both do not know what is the strategy of the second prisoner. In brief, their final decision should depend on what the police offers them, i.e. what sanctions they face.

Unlike in games with simultaneous moves, in common model of games like chess, Go or checkers; players take their actions one by one, taking into account sequence of moves made by agents acting before them. When they are making their moves, they can use that information to alter their strategy. These so-called sequential games are

[1]) One can argue, that zero-sum game is a variant of constant-sum game and not vice-versa, but because zero-sum and constant-sum games are equal, it doesn't really matter.

time-dependent and thus often represented in extensive form. In sequential games it is important to know which player is going to move first, as it can be either an advantage or a disadvantage.

Definitions of sequential games vary from relatively easy games with small game trees like in Tic-Tac-Toe, up to those particularly difficult games with exceedingly complex trees and sophisticated rules, like some variations of chess. Such games are extremely demanding for a human player and some of them unsolvable even for high performance computers.

2.3.3 Information provided

A category of games in which the player possesses all the information about the game description is called perfect-information games. The agents are able to perceive every activity and alteration of the game states. In every situation, each player is completely informed of what every other player has done up to now. The consequence is that the players are capable to estimate all possible developments of the game and its termination. This concept of game is called perfect-information game.

Perfect-information games is a class of games in which everything is known and any relevant information can be acquired. From the point of view of every agent, any situation in the game environment is unique and distinguishable. Such idealism provides some indispensable advantages. For example, in games with perfect information the existence of pure-strategy Nash equilibrium is guaranteed [26]. Intuitively speaking, since players are able to predict all potential future courses of the game, following a mixed strategy is rather redundant.

Theorem 2.13. *Let $G = (P, T, b, u)$ be a finite perfect-information game in extensive form. Then in G exists a Nash equilibrium in pure strategies.*

However, as it has been said in the introduction, the complete knowledge of the game is not always accessible. In some games, players are given only partial information about their opponents, game rules or activity in the environment. The games in which the player observe only a limited amount of all possible information are called incomplete-information games.

In these games the player builds his own game tree, where he deals with the uncertainty by creating information sets – sets of states which he cannot distinguish with his current knowledge. An example of such game is depicted in Figure 2.6.

Definition 2.14. Information set *Given an extensive-form game $G = (P, T, b, u)$ and game tree $T = (S, Z, A, e, f, r)$, an information set $I_{p_i,j} \subseteq S$ is a set of states, $\forall h, h' \in I_{p_i,j}$:*

- *$e(h) = e(h')$; and*
- *$b(h) = b(h')$.*

There may exist a large number of information sets for each player in the game. The games where the incompleteness of information manifests itself through the player's hidden knowledge about his opponents' moves; are the games of imperfect information. In these games the rules are clearly defined and information about players is complete.

Definition 2.15. Imperfect-information game in extensive form *An imperfect-information game in extensive form is a tuple $G = (P, T, I, b, u)$, where:*

- *(P, T, b, u) is a perfect-information game in extensive form; and*
- *$I = I_{p_1}, I_{p_2}, \ldots, I_{p_m}$, where $I_{p_i} = I_{p_i,1}, I_{p_i,2}, \ldots, I_{p_i,n}$ is an appropriate set of information sets for player p_i satisfying $\bigcup_{j=1}^{n} I_{p_i,j} = \{s \in S \mid b(s) = p_i\}$.*

Note that simultaneous-move games can be seen as imperfect-information games, such as within one round, player cannot truly differentiate between his opponent's actions. Hence these actions are included into a single information set.

In Figure 2.6 is depicted an example of imperfect-information game. In this game the game states which belong to one information set are within a rectangle. There is also one chance node on the top of the tree, represented as a diamond.

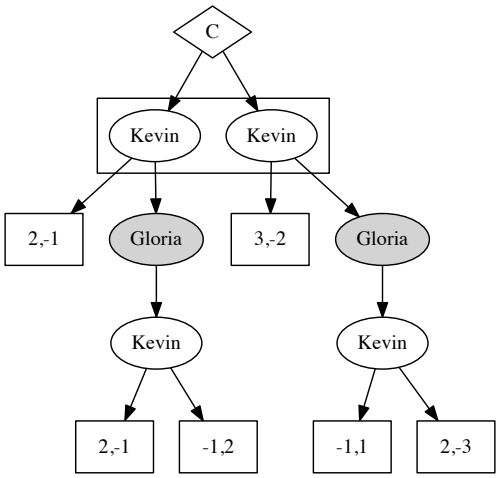


Figure 2.6. An example of a game with imperfect information.

For more precise classification of the states into the information sets, a player can use the information obtained during his previous actions. If he is aware of the whole history of his moves and observations he made up to now, such memory enables him to evaluate every possible situation in which he might find himself more accurately, stratifying the states into greater number of different information sets.

Definition 2.16. Perfect Recall *Let $G = (P, T, I, b, u)$ be a game with imperfect information. Player p_i is said to have perfect recall in G, if $\forall h, h' \in I_{p_i,l}$; and path $(r, a_0, h_1, a_1, h_2, \ldots, h_m, a_m, h)$ and $(r, a'_0, h'_1, a'_1, h'_2, \ldots, h'_{m'}, a'_{m'}, h')$:*

- *$m = m'$;*
- *$\forall j \in (0,1, \ldots, m)$, $b(h_j) = p_i : h_j \in I_{p_i,k}$ and $h'_j \in I_{p_i,k}$; and*
- *$a_0 = a'_0$ and $\forall j \in (0,1, \ldots, m)$, $b(h_j) = p_i : a_j = a'_j$.*

G is a game of perfect recall if every player has perfect recall in it.

Every perfect-information game is thus a game of perfect recall. This game feature is especially important in techniques for computing Nash equilibria, because games with imperfect recall are generally much harder to analyze.

Chapter 3
Computing Equilibria

Nash equilibrium (NE) is a fundamental concept in the game theory. There are two reasons why. First, it is widely used to predict and describe what will happen in strategic interactions between greater number of rational agents, like in wars or arms races [27], prediction of the traffic flow [28], auction theory [29] or regulatory legislation [30].

Second reason is stability. When looking for an optimal reaction, Nash equilibrium provides a strategy to which (by definition) the best response is the equilibrium. That ensures that an expected outcome won't be worse than an expected utility of the equilibrium. An opponent who changes his strategy from NE is now playing a worse strategy. The NE strategy cannot be exploited any further than by the opponent's Nash equilibrium strategy.

This chapter describes algorithms for finding Nash equilibria in both perfect-information and imperfect-information games. It shows their advantages and disadvantages and proposes some approaches to overcome some of their practical drawbacks. From now on, games are being described exclusively in the extensive form.

3.1 Perfect-information games

Under certain circumstances the Nash concept encounters several flaws, closely related to the behavior of the agents. Although every player is assumed to act rationally, the definition of NE enables the existence of non-credible threats. Non-credible threat is a term describing the situation which would actually any intelligent player never go through with. In the area of perfect-information games, such equilibria can be eliminated, when considering partial subgames from which the whole game consist.

Definition 3.1. Subgame *Given an extensive-form game G = (P, T, b, u) with game tree T = (S, Z, A, e, f, r), the subgame G' rooted at state t ∈ S is the restriction of G to the descendants of t.*

Using the notion of subgame, subgame-perfect equilibrium can be defined as a Nash equilibrium, which is also equilibrium in each possible subgame.

Definition 3.2. Subgame-perfect equilibrium (SPE) *Agents playing strategies $s_m \in S_m$ in game G = (P, T, b, u) are in subgame-perfect equilibrium IFF ∀subgames G' ∈ G, the restriction of s_m to G' is a Nash equilibrium of G'.*

In Figure 3.1 is depicted a simple sequential game of two players, as an example of a crucial difference between Nash and subgame-perfect equilibria. The game contains two NE, specifically $(crush, gnaw)$ and $(bite, chew)$. The first equilibrium is a subgame-perfect equilibrium too, since *gnaw* is a best action for Fat in right state and *crush* is an optimal action for Phil. Conversely, $(bite, chew)$ is an equilibrium, because if Fat knows that Phil will play *bite*, he would never *gnaw* for 0, since he can get 2 for *chew*.

And respectively, Phil will always go for *bite* if he knows that Fat will *chew*. But the truth is, that Phil would never go *bite* from the whole beginning – it is a non-credible threat.The reason is that for going *crush* he can obtain a utility 2, which is more than 1 for *bite*; and thus $(bite, chew)$ is not a subgame-perfect equilibrium.

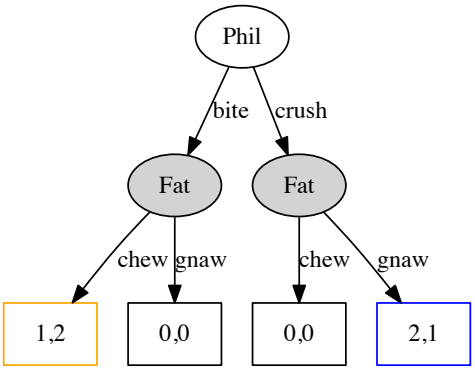


Figure 3.1. The difference between subgame-perfect (blue) and Nash (orange) equilibrium.

The fact is that the existence of SPE is always guaranteed in every finite extensive-form game [31].

Theorem 3.3. *Every finite extensive-form game G = (P, T, b, u) contains a subgame-perfect equilibrium.*

Even more advantageous property of SPE is that it can be found using backward induction algorithm. Backward induction algorithm is based on depth-first search and it recursive form is shown in Figure 3.2. This algorithm goes from the bottom layer of the game tree (line 8), considers at first all pre-terminal states and computes best strategy (line 10) in each of them to obtain the best utility (line 9). Using the acquired knowledge, it goes upwards through the game tree and evaluates each state separately, ignoring non credible moves. At the end when the root state is assigned an optimal action, the equilibrium is found.

```
1:  function BACKWARDINDUCTION(node h)
2:      if ISTERMINAL(h) then
3:          return u(h)
4:      end if
5:      best_util ← −∞
6:      best_action ← nil
7:      for all a ∈ e(h) do
8:          util_at_child ← BACKWARDINDUCTION(f(h, a))
9:          if util_at_child_b(h) > best_util_b(h) then
10:             best_util ← util_at_child
11:             best_action ← a
12:         end if
13:     end for
14:     return best_util
15: end function
```

Figure 3.2. Backward induction algorithm.

Backward induction finds only one pure-strategy SPE in linear time in the size of the game tree. In zero-sum games of 2 players, the concrete form of backward induction algorithm is well known since the beginning of game theory and its name is minimax [32].

The negative of backward inductions is its need to traverse the whole game tree in order to find the equilibrium. Fortunately, there exist a heuristic called α-β pruning [33]. This technique benefits from from the fact, that several branches of the game tree evaluated by backward induction can be eliminated. The evaluation of a branch δ of subtree generated from a particular state terminates when at least one other branch has been found that proves δ won't contribute to the equilibrium. There is no need to evaluate such branch any further. α-β pruning applied to a standard backward induction tree yield the same solution, but prunes all the unnecessary subtrees that are certain to not influence the equilibrium. In the optimal case, α-β pruning decreases the number of leaf nodes which are required to be evaluated in the game tree with branching factor b in depth d from b^d to $\sqrt{b^d}$. In the worst case the number is still b^d. The algorithm is not essential for this thesis, so it won't be explained in more detail.

3.2 Imperfect-information games

In games of imperfect information, using backward induction is not possible[1]), because of the non-singleton information sets, so another approach has to be chosen. It turned out [16], that besides pure and mixed strategies, games can be also played with a third type of strategy, not discussed yet. This new kind of strategy is called behavioral and its unique features enable to construct another game representation, which is based on behavioral strategies.

Behavioral strategy is similar to a mixed strategy in a sense of repeated one-turn games. Rather then randomizing over the set of pure strategies, behavioral strategy randomizes independently over actions in each information set with preset probability distribution.

Definition 3.4. Behavioral strategy *Let $G = (P, T, I, b, u)$ be a finite imperfect-information game, function $\gamma_j : I_{p_i,j} \to A$, set Γ_j a set of all distinct functions γ_j and denote $\Delta(Z)$ a set of all possible probability distributions over arbitrary set Z. Then $B(p_i) = \Delta(\Gamma_1)\times\Delta(\Gamma_2)\times\ldots\times\Delta(\Gamma_n)$ is a set of behavioral strategies for agent p_i.*

The interesting part is, that for games of perfect recall, the set of Nash equilibria doesn't change if we restrict ourselves to behavioral strategies [34].

Theorem 3.5. *In a game of perfect recall, any mixed strategy can be replaced by an equivalent behavioral strategy, and vice versa.*

When the existence of NE in this new strategy type is guaranteed, creating a new game representation has proved reasonable. Because this framework is directly dependent on the perfect recall, it is essential to involve a fundamental aspect of it, which seems to be a history. History of an agent is then maintained by a sequence of actions which player has to play to get into the current state.

[1]) It could be possible if there were single-stated information sets only, but this wouldn't be an imperfect-information game.

Definition 3.6. Sequence *Given an imperfect-information game G = (P, T, I, b, u) with a game tree T = (S, Z, A, e, f, r), a sequence of actions for player p_i is a list of actions $\sigma_i \in \Sigma_i$ that lie on the path from root state r to any state $s \in S$. All sequences leading to an information set I are denoted $seq_i(I)$.*

Sequences can be extended by finding feasible actions in the information set to which the particular sequence leads. Formally, for every sequence $\sigma_i \in \mathrm{seq_i(I)}$, a set of its extensions is a set $\mathrm{Ext}(\sigma_i) = \{\sigma_i a_j |\ \mathrm{a}_j \text{ is feasible in I}\}$. The game tree is constructed inductively in the way that for every information set the sequences which lead to it are taken and by their extensions is found a subsequent information set. Every state in every information set is clearly characterized by a combination of sequences of all players, which lead to this state.

Given a behavioral strategy, it is obvious that some sequences will be preferred over others in sense of their likelihood to be played.

Definition 3.7. Realization plan *The realization plan of $\beta_i \in B(p_i)$ for player p_i is a function $r_i : \sigma_i \in \Sigma_i \rightarrow$ [0, 1] defined as $r_i(\sigma_i) = \prod_{a \in \sigma_i} \beta_i(a)$.*

Realization plan computes the conditional probability of playing a sequence σ_i when considering a behavioral strategy β_i. Nevertheless, this realization probability cannot be truly arbitrary. The linear constraints C_i have to be met:

- $\mathrm{r}_i(\emptyset)$=1;
- $\forall \mathrm{I} \in \mathrm{I}_i : \sum_{\sigma'_i \in Ext_i(I)} \mathrm{r}_i(\sigma'_i) = \mathrm{r}_i(seq_i(\mathrm{I}))$; and
- $\mathrm{r}_i(\sigma_i) \geq 0$.

The first constraint says that the conditional probability of playing an empty sequence when considering any behavioral strategy is always 1. The second constraint ensures that the realization plans of sequences leading to the states reachable by one action from information set I sum up to the realization plan of reaching set I. This also allows the original behavioral strategy to be possibly recovered afterwards, just from these equations. Finally, the third constraint demands the realizations of all sequences to be nonnegative. It is quite natural, since the realization plans are the probabilities, which are by definition at worst zero.

With all this knowledge, now it is attainable to define a sequence-form representation of a game.

Definition 3.8. Sequence-form representation *Imperfect-information game of perfect recall in sequence form is a tuple G = (P, Σ, u, C), where:*

- *P is a set of players;*
- *$\Sigma = (\Sigma_1, \Sigma_2, ..., \Sigma_n)$, where Σ_i is a set of sequences for player p_i;*
- *u if a utility function, $u : \sigma \in \Sigma \rightarrow \mathbb{R}$; and*
- *$C = (C_1, C_2, ..., C_n)$, where C_i is a set of linear constraints on the realization probabilities $r_i(\sigma_i)$ of player p_i.*

With the sequence-form representation, the computation of NE can be done in time polynomial in the size of the game tree. This is far more efficient than with the induced normal form, for which even the within-form transformation causes an exponential increase and after that the equilibrium is found in polynomial time [16].

3.2.1 Full sequence method

The exact Nash equilibrium in zero-sum two-player games can be computed by solving the following linear program [16].

$$
\begin{aligned}
&minimize\ v_0 \\
&subject\ to\ v_{I_1(\sigma_1)} - \sum_{neco} v_{I'} \geq \sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2) \\
&\qquad r_2(\emptyset) = 1 \\
&\qquad \sum_{\sigma_2' \in Ext_2(I)} r_2(\sigma_2') = r_2(seq_2(I)) \\
&\qquad r_2(\sigma_2) \geq 0
\end{aligned}
\tag{3.1}
$$

The algorithm is deterministic and using a simplex method [35] for solving linear programs guarantees finding an equilibrium in a finite time. However, the calculation can be time-consuming. Due to this fact, using full sequence method as a real-time solving algorithm is without a good evaluation function computationally impossible.

3.2.2 Double-oracle method

Double-oracle method is a refinement of the full sequence method for finding exact Nash equilibrium in two-player zero-sum games. The players are allowed to play only some of the sequences of feasible actions, which restricts the original game and therefore allows faster convergence to the solution. The algorithm works iteratively and after each iteration adds the additional best-response sequences to the restricted game.

More precisely, in one iteration the algorithm [36]:

- creates a restricted game by limiting the set of sequences that each player is allowed to play
- computes a pair of Nash equilibrium strategies in this restricted game
- for each player, computes a best response strategy against the equilibrium strategy of the opponent, which may be any sequence in the complete game

The iterations are repeated until the final strategy profile is found. These strategies are proved to be a Nash equilibrium [36].

Double-oracle method is a remarkable improvement over the full sequence method, but its execution suffers from the same practical issues as full sequence. Termination of the algorithm within a few minutes is almost impossible even in the simplest games. For example, to solve a game of Phantom Tic-Tac-Toe requires more than 17,000 seconds [36].

3.2.3 Information set search

Information set search is a technique which uses opponent modeling when searching the game trees of zero-sum 2-player games [37]. Although the algorithm was originally derived from minimax, it does not suppose the players to intentionally exploit the strategy of their opponents. Instead it simulates their behavior to calculate an optimal strategy.

Definition 3.9. History *Given a two-player game $G = (P, \Sigma, u, C)$, a history h is a list of joint moves $h = (a_1, a_2, ..., a_m)$, such as $\sigma_1 = (a_1, a_3, ..., a_{m-1}), \sigma_2 = (a_2, a_4, ..., a_m)$ WLOG[1]). The set of all histories leading to information set I is denoted $seq(I)$.*

[1]) The sequences can also be defined as $\sigma_1 = (a_1, a_3, ..., a_m), \sigma_2 = (a_2, a_4, ..., a_{m-1})$, it doesn't matter.

Given a two-player zero-sum game with imperfect information G = (P, Σ, u, C) and behavioral strategies of its players β_1 and β_2, a realization plan of playing a history h ∈ $seq(I)$ is:

$$r(h, \beta_1, \beta_2) = \frac{r_1(\sigma_1)\ r_2(\sigma_2)}{\sum_{h' \in seq(I)} r(h', \beta_1, \beta_2)} \tag{3.2}$$

An expected utility of any non-terminal history can be calculated using following formula:

$$u(h, \beta_1, \beta_2) = \sum_{i=1}^{2} \sum_{a_j \in \sigma_i} \beta_i(a_j)\ u(ha_j, \beta_1, \beta_2) \tag{3.3}$$

As it can be seen, the formula is recursive and requires traversing through the whole game tree generated from the terminal node of history h, which can be computationally impractical. An expected utility can be reformulated for a given information set I:

$$u(I, \beta_1, \beta_2) = \sum_{h \in seq(I)} r(h, \beta_1, \beta_2)\ u(h, \beta_1, \beta_2) \tag{3.4}$$

Denoting A(I) a set of possible actions in information set I, a set A* of optimal actions for player who is on move in I is according to the theory of information set search computed as:

$$A^*(I, \beta_1, \beta_2) = argmax_{a \in A(I)} u(Ia, \beta_1, \beta_2) \tag{3.5}$$

From the optimal actions can be deduced an optimal strategy β_1^*(I) for player p_1 (WLOG) in information set I.

Theorem 3.10. *Given a two-player game G = (P, Σ, u, C) and a behavioral strategy β_2 of player p_2, an optimal strategy of player p_1 in information set I for playing action a ∈A(I) is*

$$\beta_1^*(a) \ = \ \begin{cases} \frac{1}{|A^*(I, \beta_1^*, \beta_2)|} & \text{if } a \in A^*(I, \beta_1^*, \beta_2); \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

The strategy β_1^* is claimed to be a best response to strategy β_2 [37]. The problem is that a precise model of opponent's strategy is in practice usually not accessible. Consequently, the solving algorithm has to rely on approximate model of behavior of player p_2. Among two most common [37] opponent modelings are:

- paranoid model
- overconfident model

Paranoid model assumes, that the opponent will always make his best possible move. A paranoid player expects his opponent to have the same information about the paranoid player's information sets as he does and that the opponent will take actions which are the worst for a paranoid player. This might cause significant errors in the model, when playing against a non-perfect player. The reason is, that the opponent considers different information sets than the paranoid player, which may not allows him to make such perfect moves.

On the other hand, overconfident model represents a player, who expects his opponent to be totally unable to consider available information and that he makes his moves purely randomly. Oddly enough, this approach can sometimes outperform the paranoid model, mainly in situations where the paranoid player's opponent modeling is completely wrong.

To conclude, information set search is a powerful technique for solving games, but its dependence on modeling can lead to incorrect conclusions about optimal actions. In fact, in situations where the model's strategy does not represent a best response to strategy β_1^* (and these are the most common [37]), the strategy profile found is not an equilibrium. Due to these problems, information set search is not a desired algorithm of this thesis.

3.2.4 Regret minimization

Regret minimization [16] is an alternative solution concept to Nash equilibrium originally introduced in the field of perfect-information games. The technique does not maximize the worst achievable outcome, but instead it minimizes the worst-case difference between the utility obtained when playing the current strategy and the utility that would have been gained if different decisions had been chosen.

Definition 3.11. Regret *Given a game $G = (P, A, u)$ and an action profile $a = (a_1, ..., a_i, ..., a_m)$, player p_i's regret regret for playing an action a_i is*

$$R_i(a_i) = \max_{a' \in A_i} u_i(a'_i, a_{-i}) - u_i(a_i, a_{-i}) \tag{3.7}$$

Regret minimization strategy ensures, that the player plays reasonably well when compared to the overall best possible outcome, no matter how do the other players behave. His best action is then defined as:

$$a^* = argmin_{a_i \in A_i} \max_{a_{-i} \in A_{-i}} (R_i(a_i)) \tag{3.8}$$

Although this solution concept was meant to solve the games of perfect information, regret minimization method was later extended to imperfect-information games too [38–39]. The concept is called counterfactual regret (CFR) and the method is based on iterative minimization of the immediate counterfactual regret at each information set. This is achieved by many repetitions of a single game.

Definition 3.12. Immediate counterfactual regret *Given a game $G = (P, T, I, b, u)$, immediate counterfactual regret is a player's p_i average regret for playing actions at information set I, if he had tried to reach it:*

$$\begin{aligned} R^T_{i,imm}(I, a) &= \frac{1}{T} \sum_{t=1}^{T} \pi^{\beta^t}_{-i}(I)(u_i(\beta^t|_{I \to a}, I) - u_i(\beta^t, I)) \\ R^T_{i,imm}(I) &= \max_{a \in A(I)} R^T_{i,imm}(I, a) \end{aligned} \tag{3.9}$$

where:

- *$A(I)$ is a set of feasible actions in information set I;*
- *$\pi^\beta(I)$ is a probability of reaching information set I if players behave according to strategy profile β. Therefore $\pi^\beta_{-i}(I)$ is;*
- *$\beta|_{I \to a}$ is a strategy profile identical to β, except that player p_i takes an action a in information set I;*
- *$u_i(\sigma, I)$ is counterfactual utility – the expected utility given that information set I is reached and all players play using strategy except that player p_i plays to reach I; and*
- *T is the number of repetitions of the game.*

Immediate conterfactual regret is a value defined in every information set. Conversely, the average overall regret specifies the regret of player p_i in the whole game:

$$R^T_{avg,i} = \frac{1}{T} \max_{\beta^* \in B_t} \sum_{t=1}^{T} (u_i(\beta^*_i, \beta^t_{-i}) - u_i(\beta^t)). \tag{3.10}$$

The relationship between these two definitions of regret turned out to be really important. It has been proved [39], that the average overall regret is bounded by the positive portion of immediate conterfactual regret:

$$R^T_{avg,i} \leq \sum R^{T,+}_{i,imm}(I), \tag{3.11}$$

where:

$$R^{T,+}_{i,imm}(I) = \max\left(R^T_{i,imm}(I), 0\right). \tag{3.12}$$

Furthermore, in two-player zero-sum games with perfect recall, minimizing the average overall regret of all players leads to an approximate Nash equilibrium [39].

Since the average overall regret is bounded this way, the algorithm for finding an appropriate strategy can take advantage of it and rather than focusing on the overall regret, it can try to decrease the immediate counterfactual regret in every information set. If the strategy is modified after each iteration according to this formula:

$$\beta_i^{T+1}(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a' \in A(I)} R_i^{T,+}(I,a')} & \text{if } R_i^{T,+}(I,a) > 0; \\ \frac{1}{|A(I)|} & \text{otherwise,} \end{cases} \tag{3.13}$$

the convergence to an approximate Nash equilibrium is guaranteed [38].

The method of finding equilibria based on conterfactual regret minimization became a significant improvement in solving imperfect-information games, especially in a sense of memory complexity. The reason is that the algorithm depends on the number of information sets, instead of the game states like in the full sequence method. This enables to compute the equilibria even in the much larger game spaces. Another advantage of CFR is its ability to compute best responses against one or more static opponents [38].

However, there are no theoretical guarantees for CFR to work in games with more loose conditions, e.g. in games with multiple opponents, non-zero-sum or imperfect-recall games; although it performs well also in some of these games [40]. In addition, the algorithm works so well especially due to the domain-dependent abstractions, which might be the real reason behind the success of CFR in card games like poker. In these games, the game descriptions are known in advance and the algorithm can use specifically designed abstractions. In GGP no good way of doing the abstraction is known [41], which is probably the most significant weakness of using CFR in GGP.

3.2.5 Monte Carlo methods

Monte Carlo (MC) methods are a statistic-based class of computational algorithms that use great number of repeated random sampling to calculate numerical values. These values are than used to estimate an unknown probability distribution. Monte Carlo techniques are usually applied when it is inconvenient to perform a deterministic calculation, either because it is impossible in a sense of time or memory, or the deterministic formula doesn't even exist.

A variant of MC invented in the game theory[1]) is called Monte Carlo tree search (MCTS) [42]. MCTS is a heuristic search algorithm, which is immensely successful

[1]) In fact, this variant is used in the whole decision theory, but it is not essential for this thesis.

especially during the search of huge decision trees. Programs based on MCTS are getting better every year, as the computational power increases. For example in the game of Go, which is still considered to be one of the most difficult games, the first algorithm has already reached a 6th[1]) out of 9 dan (level) on 19x19 board [43].

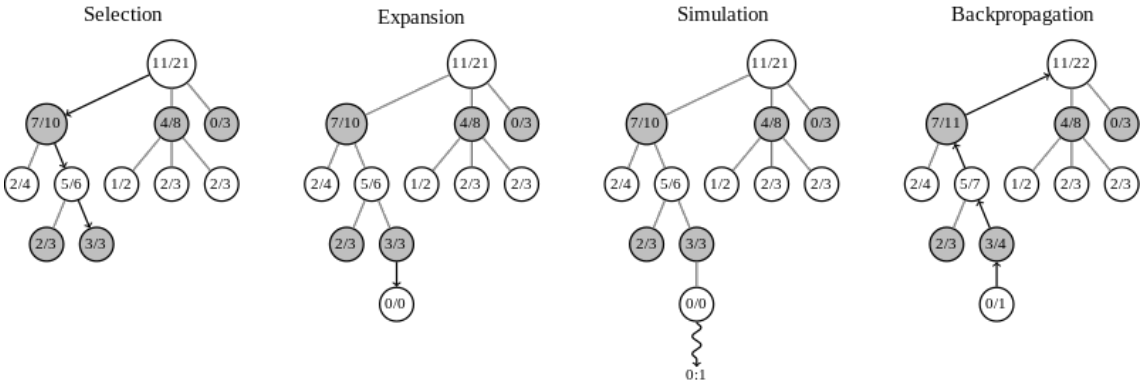


Figure 3.3. Four phases of each iteration of MCTS algorithm [44].

MCTS is based on analyzing the most promising nodes in the search space. The algorithm is iteratively constructing a partial game tree rooted at the initial state of the game.

Definition 3.13. Monte Carlo partial tree *Given a game G = (P, T, b, u) with a tree T = (S, Z, A, e, f, r), MCTS algorithm starts to construct a game tree T' = (S', Z', A', e, f, r, n, w), where:*

- *S' is a subset of S;*
- *Z' is a subset of S ;*
- *A' is a subset of A;*
- *n is a function which counts number of performed simulations in any state from S, n: S* $\rightarrow \mathbb{N}$*; and*
- *w is a function which stores a cumulative utility of play-outs performed, w: S* $\rightarrow \mathbb{R}$.

Every iteration of Monte Carlo tree search includes 4 phases, shown in Figure 3.3:

- Selection – the algorithm starts in a root-node r and selects one of the leafs l $\in$ Z' of the already expanded game tree.
- Expansion – then it expands none, one or more children of l, adds them to S', respectively modifies Z' and A', and picks one of them or l for simulation.
- Simulation – now a single random play-out from the node selected in expansion down to z $\in$ Z is performed.
- Backpropagation – the result of the play-out is then propagated back to the root, updating the values of n and cu functions in appropriate game states.

During the run, the algorithm expands the tree in accordance to random sampling. Every sampling is executed as a play-out, within which the moves are chosen randomly. The final utility of a particular play-out is used to modify statistics on the way from the root of the play-out up to the root of the MC partial tree. The value is utilized to evaluate nodes in the tree, so that when the next root of a play-out is being chosen, the nodes with better outcomes are preferred.

[1]) Using the methods for solving games based on informed search, the algorithms accomplished at most 2nd kyu, which can be compared to -2nd dan [43].

Critical is a balance of exploitation of states with high value estimates and exploration of states with uncertain value estimates. In the selection phase MCTS decides, in which way should be the tree expanded and evaluated during the current iteration. A popular method for the selection is based on maximization of a UCT (Upper Confidence Bound 1 applied to trees) formula (3.14) [45], which was derived from a UCB1 (Upper Confidence Bound 1) [46]. In every decision node on the way from the root to some of the leafs, the UCT value decides in which direction the descend should continue. Since the estimate of a current value remains uncertain until a large amount of simulations is executed, a balance between exploitation of nodes with high values and exploration of nodes with a low number of passes is critical.

$$UCT: \ \frac{u_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}} \tag{3.14}$$

Where:

- w_i is the cumulative utility after i moves;
- n_i is the number of simulations performed after i moves;
- c is the exploration parameter; and
- t is the number of simulations in a given node, t $= \sum_i n_i$.

The formula is balancing the exploration and exploitation in the way, that the first component is responsible for the exploitation, because it approaches 1 for a node with a high number of wins. The second component corresponds to exploration, since the fewer simulations were completed, the greater value the node acquires. The exploration constant defines how much stress is put on the exploration phase. A greater c signifies more emphasis over exploration. Some theoretical analyses [45] expect c to be equal $\sqrt{2}$, but the value is in practice chosen empirically.

Although the UCT formula works well in the perfect-information game, it has been proved the convergence to Nash equilibrium in games of imperfect information is not guaranteed [47]. The solution brings an EXP3 formula [48].

$$EXP3: \frac{(1-\gamma)\exp(w_h\eta)}{\sum_{h' \in A(p(h))} \exp(w_{h'}\eta)} + \frac{\gamma}{|C(p(h))|} \tag{3.15}$$

Where:

- h is a leaf node of MC partial tree;
- w_h is the cumulative utility over all simulations through h;
- p(h) is a parent of h;
- A(h) is a set of actions applicable in h;
- γ is the exploration parameter, $\gamma \in [0, 1]$; and
- $\eta = \frac{\gamma}{|A(p(h))|}$.

The Exploration-exploitation with exponential weights (EXP3) computes a probability of selection a node h. The formula is also balancing the exploration and exploitations, similarly to UCT. The greater the γ parameter will get, the more will the formula focus on exploration, since the first component will be almost equal zero.

The MCTS algorithm which uses the EXP3 criterion as its selection policy is proved to converge to an approximate Nash equilibrium in simultaneous-move MCTS (SM-MCTS) variant [49]. It is believed to converge also in a general imperfect-information MCTS algorithm [50], however, the proof is still missing. An encompassing survey over the numerous variants of MCTS is done in [51].

The main advantage of Monte Carlo methods (especially in the context of general game playing) is their domain-independence and ability to find more precise approximate Nash equilibrium with more iterations executed. At last but not least, the algorithm is elegant and easy to implement. A practical disadvantage is that for useful approximation of Nash equilibrium is required a significant amount of time and iterations.

Chapter 4
General Game Playing

This chapter presents both theoretical and practical fundamentals of general game playing. As it has been said in the introduction, GGP arose as an attempt to bring more general and domain-independent techniques into playing games. As an abstract concept, general game playing consists of 4 parts:

- *Knowledge Representation.* All game descriptions are obligated to follow a standard computer representation. This representation captures information about the game in a manner, that it can be later used to solve more complex problems like reasoning and finding optimal moves.
- *Reasoning.* The player has to reason about the game description to be able to construct the game tree. The reasoning in GGP includes finding legal moves, identifying players, determining the properties of following states, etc.
- *Planning and Search.* To play the game successfully, it is essential to recognize sensible actions among all possible ones. This purpose serve well the techniques of computational game theory.
- *Learning.* To support the planning the players can profit from previously gained or currently extracted knowledge.

In this chapter are presented 3 out of these 4 fundamental parts of playing general games. At the first place the chapter specifies the programming paradigm which is being used to describe both the games and the game environment. Consequently, it proceeds with defining the GDL language for knowledge representation; and GCL language for communicating with the game servers. Next the chapter presents two basic approaches into reasoning and finally, it identifies various methods of learning in GGP. The fourth part – planning and search – is then discussed in the following chapter. All descriptions in this chapter are loosely inspired by [9].

4.1 Game specification

Defining various games within a one concrete environment requires a robust mathematical background. Every agent has to be aware of all the possibilities the environment offers. This includes the difference between legal and illegal game descriptions. These boundaries clearly determines the kinds of games players can encounter and how they will be represented.

Logic programming is a programming paradigm, which is based on formal logic and thus provides suitable methods for precise definitions of both games and game environment.

4.1.1 Logic programming

Compared with imperative programming, logic programming defines the problem, rather then how to solve it. The problem is incorporated into logic environment and executing the program is to find a solution [52].

First, it is necessary to define several basic notions. Logic programming languages consist of four sets of symbols:

- object constant
- function constant
- relational constant
- variable

Object constants denote simple objects occurring in the environment, function constants are the names of functions, similar to calculus and relation constants represent various associations. Variables are used to substitute any object constant.

As an example, let's take a crocodile farm named Toothy Hill [53]. There are 4 crocodiles living on Toothy Hill, concretely Phil, Fat, Gloria and Kevin. All these names are object constants. Fat is a hatchling of Phil and Gloria, which makes `hatchling` a function constant. Then `green` can be a relational constant describing the property of some of the inhabitants of Toothy Hill. If X is a variable, we can then say that some X is green by `green(X)`.

Definition 4.1. Term *A term is either an object constant, a variable, or a functional term, i.e. an expression consisting of a function constant and n simpler terms.*

Terms portray an implementation of the basic symbols. So for example, if `hatchling` is a function constant, `X` is a variable and `Phil` is an object constant, then `hatchling(X,Phil)` is a functional term of two arguments.

Definition 4.2. Atom *An atom is an expression formed from a relation constant and n terms.*

Relation between objects signifies some mutual dependency or property of one or more objects. So if we take out little farm and consider another function constant `tail` and relational constant `longer`, then `longer(tail(Kevin),Gloria)` is an atom representing binomial relation.

Definition 4.3. Rule *A rule is an expression consisting of a distinguished atom, called the head, and a conjunction of zero or more atoms or negations of atoms, called the body.*

Now consider this logic program. Symbol ∼ denotes a negation.

```
crocodile(Phil)
crocodile(Fat)
toothy(Phil)
long(Phil)
longer(X,Y) :- long(X), ~long(Y)
threaten(X,Y) :- crocodile(X), crocodile(Y), toothy(X), longer(X, Y)
```

Logic programs consist of facts and rules. In the program above, the facts are instantiated relations. They represents the reality, that both `Phil` and `Fat` are crocodiles. The program also admits, that `Phil` is `long` and `toothy`. Two rules are included. First says that somebody is `longer` then someone else, if the first entity is long and the second one is not at the same time. Second rule presumes that someone will `threaten` someone else, if they are both `crocodiles`, but the first one is `toothy` and `longer` than the second one.

Definition 4.4. Proposition *For a given logic program P, a proposition is a structure consisting of an relation of n arguments and n objects from P. Every proposition is possible to evaluate as true or false.*

In logic program presented above, it is possible to ask, if Fat can be threatened by Phil. It is a proposition. The environment then takes the rule and instantiates it to fit the query.

```
threaten(X,Y)
X = Phil
Y = Fat
```

Following the definition of the rule, the instantiation is then pushed further into the body.

```
crocodile(Phil)
crocodile(Fat)
toothy(Phil)
longer(Phil,Fat)
```

The `longer` rule is nested into the `threaten` rule, so it means that the instantiation has to be pushed once again.

```
long(Phil)
~long(Fat)
```

And these are the facts of the program. There is no other long entity than `Phil`. Consequently, the proposition is true.

4.1.2 GGP environment

Game Description Language (GDL) is a logic programming language derived from Datalog, but static and purely declarative. The set of all relational, functional and object constants is always finite in every game (although in some cases it can be very, very large).

Language specification of GDL consists of two parts:

- game-independent vocabulary; and
- game-specific vocabulary.

Game-independent vocabulary provides the names and specification of every obligatory constant to define the game. Such terms can occur in any game and a player has to recognize their meaning. GDL uses default object constants substituting the integers 0,1,..,100, used as utilities or indexes. Relational constants are dependent on a GDL version and no function constants are predefined. It uses & symbol to define a conjunction and a $\sim$ symbol for negation. GDL exists in several forms[1]), describing numerous types of games. Here presented will be two standard versions – GDL-I and GDL-II.

Games defined within GDL environment have always finitely many states, including one initial state and one or more terminal states. Consequently, every game tree of a GDL-described game has a fixed maximal depth. The number of players is finite and

[1]) For example there is a simplistic variant of GDL called mGDL [54]. A modification called SGDL (Strategy Game Description Language) is intended for use in various strategy games [55]. There also exists an idea of an extension of GGP called GGPGPGPU (for representing a game on graphical processing units), which I believe would require a modified GDL. The two most common forms of GDL did come through an evolution too.

constant in every game and every player has at least one goal state. The players choose their moves simultaneously (i.e. the games belong to the class of simultaneous-move games) and the environment changes in response to their actions only.

Each game specified in GDL has to be well-formed [9], which means it is obligated to satisfy 3 conditions :

- *Termination.* A game description in GDL terminates if all infinite sequences of legal moves from the initial state of the game reach a terminal state after a finite number of steps.
- *Playability.* A game description in GDL is playable if and only if every role has at least one legal move in every non-terminal state reachable from the initial state.
- *Weak-winnability.* A game description in GDL is weakly winnable if and only if, for every role, there is a sequence of joint actions of all roles that leads to a terminal state where that role's goal value is maximal.

Moreover, everything which cannot be proved is in general game environment by default false.

4.1.3 GDL-I

GDL-I is a form of GDL [56], which describes extensive-form games with simultaneous moves. Games specified in GDL-I are obligated to provide at least one legal action for each player in every non-terminal state. Sequential perfect-information games can be implemented in GDL-I by enabling only one player to play a non-noop action in every state of the game.

constant name	meaning
`role(a)`	`a` is an agent in the game
`init(p)`	the proposition `p` is true in the initial state
`true(p)`	the proposition `p` is true in the current state
`distinct(o,p)`	the proposition `p` is not equal to the proposition `o`
`does(r,a)`	agent `r` performs action `a` in the current state
`next(p)`	the proposition `p` is true in the next state
`legal(r,a)`	action `a` is feasible for agent `r` in the current state
`goal(r,n)`	the current state has utility `n` for player `r`
`terminal`	the current state is a terminal state

Table 4.1. Relational constants in perfect-information GDL.

Relational constants of game-independent vocabulary for GDL-I are specified in table 4.1. The players of the game are specified by relational constant `role`. The initial state of the game (and also of the game tree) is described by all propositions, which are true in the root state. These are those, which satisfy the relation `init`. Legal actions for each player in the current state are restricted by `legal` and their number is always finite. As the game tree branches, some propositions become true. Constant `true` means that its argument is true in the current state and `next` remarks the same thing for a next state. `Distinct` requires its two parameters to be different propositions. The transfer in the game tree is caused by `does`, which says that a player executed a particular action in the current state. Finally, terminal state is identified by `terminal` and the utilities provides a binomial relation `goal`.

Now consider this example of Prisoner's Dilemma in GDL.

```
 1 role(Kevin)
 2 role(Fat)
 3 init(p)
 4 legal(Kevin, cooperate)
 5 legal(Kevin, defect)
 6 legal(Fat, cooperate)
 7 legal(Fat, defect)
 8 next(t) :- true(p)
 9 goal(Kevin, 100) :- does(Kevin, defect) & does(Fat, cooperate)
10 goal(Kevin, 66) :- does(Kevin, cooperate) & does(Fat, cooperate)
11 goal(Kevin, 33) :- does(Kevin, defect) & does(Fat, defect)
12 goal(Kevin, 0) :- does(Kevin, cooperate) & does(Fat, defect)
13 goal(Fat, 0) :- does(Kevin, defect) & does(Fat, cooperate)
14 goal(Fat, 33) :- does(Kevin, defect) & does(Fat, defect)
15 goal(Fat, 66) :- does(Kevin, cooperate) & does(Fat, cooperate)
16 goal(Fat, 100) :- does(Kevin, cooperate) & does(Fat, defect)
17 terminal :- true(t)
```

The game is a variation of the classic Prisoner's Dilemma, which was already described in section 2.3.2. There are two prisoners, here named Kevin and Fat, who are separated and try to decide, whether they should stay quiet and cooperate or betray the other prisoner. Both their actions are legal in every state of the game.

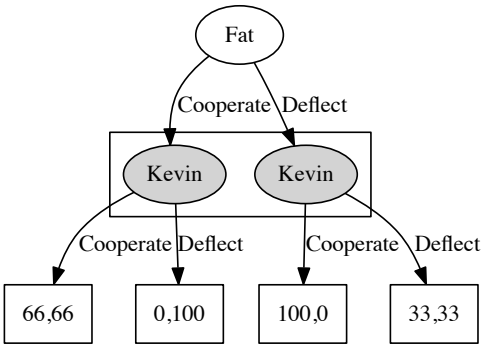


Figure 4.1. The game tree of Prisoner's Dilemma formulated in GDL-I.

To demonstrate the whole game progress, the game tree of Prisoner's Dilemma is depicted in Figure 4.1. So for example, if Kevin decides to betray Fat, but Fat is foolish enough to stay quiet, game terminates and Kevin gets his 100 and Fat quits the game with 0.

4.1.4 GDL-II

Because GDL-I expects all players to have complete information about the game and the moves of all players in the previous turns, it is not able to describe games with imperfect information. GDL-II was created as an extension of GDL-I, using its game-independent vocabulary and adding some new relational and object constants to be able to define incomplete games [57].

constant name	meaning
`random`	pre-defined role that moves purely randomly
`sees(r,p)`	agent `r` perceives `p` in the next state

Table 4.2. Extended constants in imperfect-information GDL.

`Random` is a new kind of player, representing the nature in the game, e.g. the dice roll. The players do not have complete information about the game. Instead, they may percept only selected parts of the state of the game. In every situation, the details they are informed of is defined by `sees`. The number of possible percepts is finite in every GDL-II game.

An example of a coin-flipping game formulated in GDL-II is presented below.

```
 1 role(random)
 2 role(Phil)
 3 init(t1)
 4 succ(t1,t2)
 5 succ(t2,t3)
 6 next(X) :- true(Y) & succ(X,Y)
 7 terminal :- true(t3)
 8 coin(head)
 9 coin(tail)
10 legal(random, tossCoin(X)) :- coin(X)
11 legal(Phil, guessFlip(X)) :- coin(X)
12 next(guessRight) :- true(guessRight)
13 next(guessRight) :- does(random, tossCoin(X)) &
                       does(Phil, guessFlip(X))
14 goal(random,100)
15 goal(Phil,100) :- true(guessRight)
16 goal(Phil, 0) :- true(t3) & ~true(guessRight)
17 sees(Phil, guessRight) :- true(guessRight) & true(t2)
```

This coin-flipping game has only one rational agent, called Phil. The game is played in two rounds and at the beginning of each round a `random` player toss a coin. Phil then tries to choose between two alternatives, `head` and `tail`. If he is successful, he will gain 100 for sure, but since the game provides no percept in these states, he would have never know. Anyway, the game continues to the second round and Phil faces the same choice. Whether he is victorious not, the game ends and he gets 100 if he guessed at least once the value right, or 0 otherwise.

The `sees` predicate guarantees, that Phil will become aware of the outcome of the game, once he makes his two choices. This is granted due to the `guessRight` constant, which is propagated through the game from the moment it became true. This can occur in two situations – after the player takes an action in the first round (more precisely, when he gets into the second round), eventually after the second round in case Phil is not declared a winner in the first round.

The game tree of the coin-flipping game is in Figure 4.2. Until the game end, Phil is not able to determine the state in which he is localed. The only thing he percepts during the game-play is the request to select another action. This is why there are two information sets in the tree. For example, if the actions taken by Phil are {`tail`, `head`} and the actions of a `random` player are {`head`, `head`}, Phil wins and will receive utility 100. This particular game-play corresponds to the leftmost branch of the game tree.

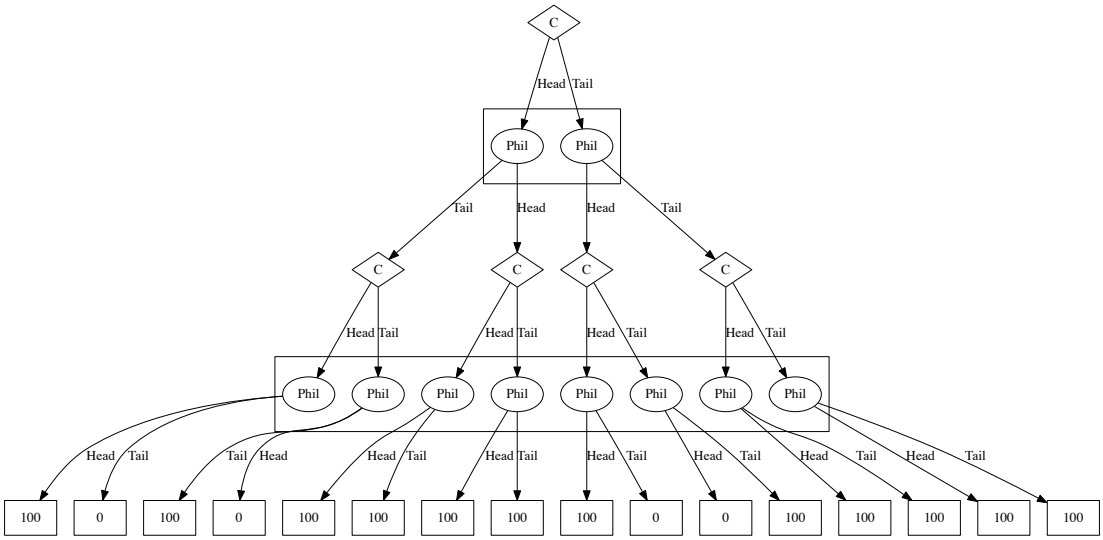


Figure 4.2. The game tree of a coin-flipping game formulated in GDL-II.

4.2 Game management

GGP is played through the network using HTTP protocol. GGP server registers the players and their IPs, schedules tournaments, or just a regular game playing, and provides results and analysis. Game manager is a program hosted at a GGP server, which manages the game-play of a particular game. When the match is about to begin, the manager receives the details about it from the game server. This includes the game specification in GDL[1]), two kinds of time limits and the info about the players. The GGP matches always incorporate two clocks – a time limit before the game starts (start-clock), and a time limit for each move (play-clock). Players are given a starting time limit, so that they can analyze the game before the game-play actually begins. Once the manager processes the settings of the match, the connection with each player is established.

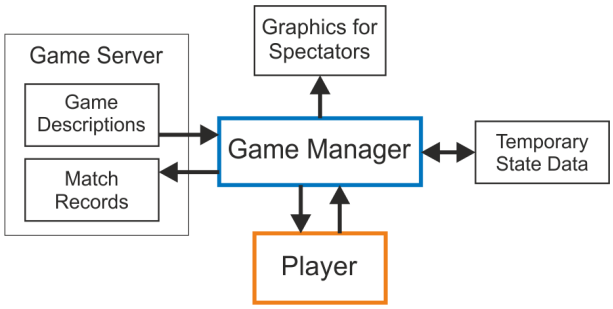


Figure 4.3. A schema of game manager as a link between a player and a game server, inspired by [9].

Manager also provides output for spectators. When the game terminates it sends the results back to the server's game scheduler.

[1]) For better readability, the presented form of GDL was in the infix form. But in fact, not all GDL systems enable to use such representation. Conversely, each one supports GDL in prefix form, so it became a standard notation. In Appendix C can be found the description of this form and an example.

4.2.1 GCL-I

For communication with players the manager uses a simple game communication language (GCL). Game communication language has two variants depending on which GDL version is used. As a first version is described GCL-I, a form of GCL designed for communication in GDL-I matches.

The language is quite plain. It consists of only five defined keywords. They are:

- `(info)`
- `(start id role description start-clock play-clock)`
- `(play id moves)`
- `(stop id moves)`
- `(abort id)`

`info` message is used to ping the players. It is a common manner to return a player's name in response to an `info` message.

The match starts when a player receives a `start` message. This message contains a match id – a unique identifier, so that the player can distinguish between different contests; a role assigned to a player; a whole game description in GDL; a start-clock, which determines the remaining time before the match starts; and finally a play-clock for setting a fixed time limit the players have for choosing the next move. Within the start-clock limit, the players have to respond `ready`.

The `play` message manages the progress of the single match. It informs each player about the moves done by all participants in the game and requests the next action at the same time.

When the game ends and the winner is determined, all players are notified by a `stop` message. It holds the same information as a `play` message, but it does not demand the agents to provide a next move. Players respond `done`.

The match can also terminate in an unforeseen manner, usually when some of the participants disconnects or if someone is continuously attempting to perform an illegal action. Such situation will result in sending the `abort` message to all players.

4.2.2 GCL-II

The communication language for matches in imperfect information games alter two keywords of the original GCL-I. They are:

- `(play id turn move percept)`
- `(stop id turn move percept)`

The meaning remains the same, but the messages require new arguments, due to the need to inform the players about their percepts. Both the `play` and the `start` message now contain three new arguments. First, a `turn` is an integer, a number increasing in every round by one and used to ensure the player about the maintenance of a communication, a `move` only confirms his previous action or the action selected by the manager, if his choice was somehow incorrect [1]); and a `percept` informs him about his perceptions.

[1]) The policy of a manager in cases when one of the players attempts to play an illegal action differs. Sometimes it just chooses a legal action itself and informs the respective player about it in the next `play` message. The manager's reaction depends on rules of a particular server. The same thing applies for GCL-I.

4.2.3 Game flow

Since this thesis focuses on imperfect information games, an example of a game-play will be given on the Blind Tic-Tac-Toe game in GDL-II variant with the communication in GCL-II. The whole description of the game can be found in [9].

The game starts when the manager sends the start message to all players involved in the match.

Game Manager to Player x: `( start blindttt30 x ( (role x ) ... ) 10 10 )`
Game Manager to Player y: `( start blindttt30 o ( (role x ) ... ) 10 10 )`
Player x to Game Manager: `ready`
Player y to Game Manager: `ready`

The id of the match is `blindttt30`. It enables the players to check, whether the incoming message truly belongs to the match they are playing right now.

As the game progresses, the players might all try to mark field (2 2). Since the simultaneous marking are not allowed, player `y` is informed about his fail to do so.

Game Manager to Player x: `( play blindttt30 2 ( mark 2 2 ) ( ok ) )`
Game Manager to Player y: `( play blindttt30 2 ( mark 2 2 ) nil )`
Player x to Game Manager: `( mark 3 3 )`
Player y to Game Manager: `( mark 1 3 )`

It is evident, that they won't interfere now, thus the denying of their actions will depend on a current state of the game.

When the game terminates, all players receive the `stop` message.

Game Manager to Player x: `( stop blindttt30 3 ( mark 3 3 ) (ok) )`
Game Manager to Player y: `( stop blindttt30 3 ( mark 1 3 ) (ok) )`
Player x to Game Manager: `done`
Player y to Game Manager: `done`

4.3 Game description reasoning

To successfully participate in an imperfect-information game, a player has to correctly figure out the way the game is played. For finding legal moves in every state of the game, the player reasons about the game descriptions in GDL-II. Since the reasoning can be quite time-consuming, efficient reasoner can provide an opportunity to focus on more advanced calculations, which is finding better moves. There are two possible basic approaches to this problematics.

The first option is based on fact, that dynamics of the particular game can be captured by a state machine. At every moment, the game is situated in a single state of the state automaton and as the actions are performed, the state of the game changes and the game moves along the corresponding edge on the state graph to reach the next situation.

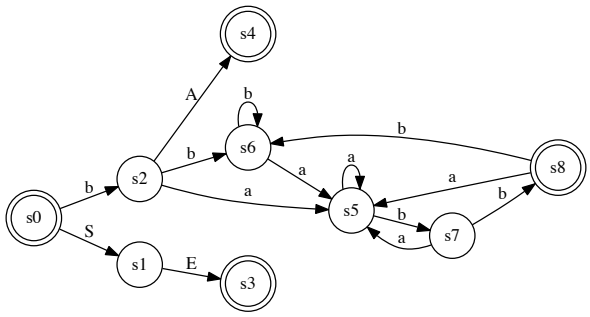


Figure 4.4. A simple game represented as a state machine, inspired by [58].

When the game is represented as a state machine, the simulation of a transit along the edge is done by an automated theorem prover working with the game description in GDL. The prover takes the facts which determine the state and computes all edges which originate in this particular state. As the game proceeds along the edge, the prover involves the information provided by the edge, i.e. which actions were executed. Using this knowledge, the prover is able to reconstruct the facts that become true in the next state.

The automated theorem provers works well due to the fact, that in the GGP environment the set of all relational, functional and object constants is always finite. This ensures that the prover will terminate in finite time whenever considering a well-formed GDL-II game. But although the provers has proved to be consistent and correct, they are quite slow when searching a large game tree.

The state machines conveniently capture the dynamics of the game, but ignore the dynamics of states themselves. For a state machine representation, a state is a solid data structure, which only stores the facts about that state, specifically which propositions are true. However, the flow of the game is directly dependent on the way the propositions change their values according to which actions were executed by the players. This suggests the game to be characterized as a propositional net, rather than as a state machine.

In terms of graph theory, a propnet is a directed bipartite hyper-graph. It is similar to a logic circuit, as it consists of 5 components:

- propositions
- negations
- conjunctions
- disjunctions
- transitions

Every game description can be transformed into a unique schema of corresponding propositional network. This schema determines the way the components of the particular network are interconnected. For example, in Figure 4.6 is depicted a propositional net of GDL-I version of Prisoner's Dilemma, as described in section 4.1.3. The components of every propositional net can be partitioned into 2 sets – connectives and propositions. The propositions are the components of the network, which can be marked with a boolean value. The connectives create relations between individual propositions a determine the way the values are assigned to each proposition.

A propositional network represents the states of simultaneous-move game by a set of boolean values of propositions in the net. Consequently, every state of the game in

extensive form, in which the players effectively take their moves at the same time share the same set of proposition's values. In every propnet, there are three basic types of propositions:

- base propositions
- input propositions
- view propositions

Base propositions are the propositions with incoming edges from transitions. They represent a property of a current state which emerged from a previous state after taking a particular action. In propositional net 4.6, an example of base proposition is `( true P )`. Input propositions are the propositions with no inputs. They correspond to an action taken by a player. Every action for every player has an equivalent proposition included in a propnet and in every state of the game, only one input proposition for each player is true (excluding a root state). In propnet 4.6, there is an input proposition called `( does Fat DEFECT )`. Every propnet contains one special input proposition, which is true in a root state only – `init`. Other extraordinary input propositions are constants, the propositions with fixed value. Naturally, they are true or false. The last kind of propositions are the view propositions. These are incorporated into inner architecture of the network and their value can be calculated from values of input and base propositions. For instance, the propnet 4.6 contains a view proposition `( legal Fat COOPERATE )`. In other words, to uniquely distinguish one state of the simultaneous game among other states, the values of input and base propositions are sufficient.

An algorithm which computes a value of a particular view proposition takes its input and performs a depth first search through the network's architecture until it reaches all bases and inputs on which the view proposition depends. Then it propagates the values back and sets the value of the proposition in accordance to the connectives which the algorithm meets on the way. As an example, consider the mapping 4.5 of bases and inputs of propnet 4.6.

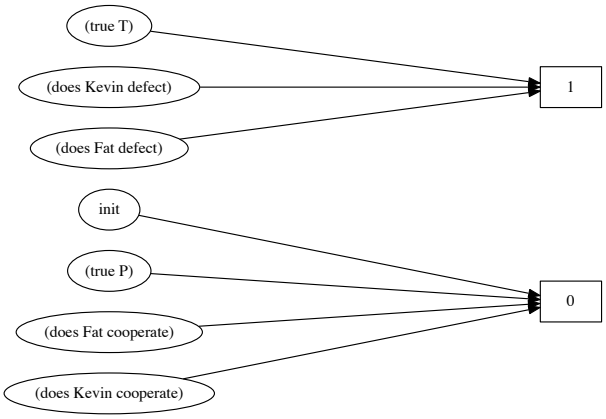


Figure 4.5. A sample mapping of the propositions to their boolean values in the propositional network of Prisoner's Dilemma.

To determine the value of `( goal Kevin 0 )` the algorithm goes backwards, starting in this proposition. It meets a conjunctive, which value is determined by its 2 inputs. These are `( does Fat defect )` and `( does Kevin cooperate )`. They are input propositions, so at this moment the algorithm checks the input markings.

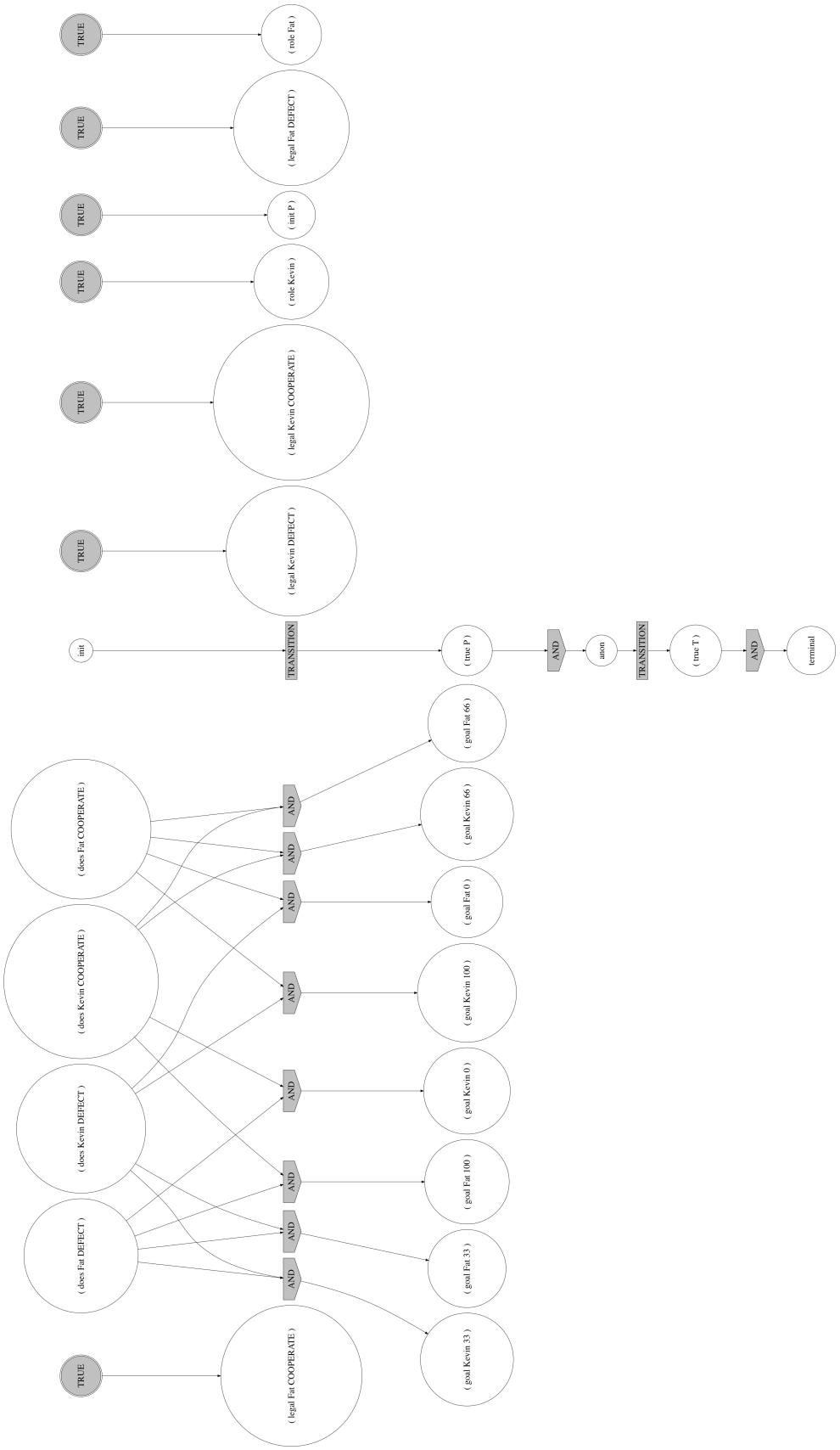


Figure 4.6. The propositional network representing the Prisoner's Dilemma.

According to the mapping of these propositions to their boolean values, the proposition `( does Fat defect )` is true, but `( does Kevin cooperate )` is false. From this follows that the conjunctive is false and the value of `( goal Kevin 0 )` is therefore false too.

To compute an initial state of a game, the propnet first marks all the input propositions with variable value false, as no player has taken any action yet. Then it assigns a true value to the `init` proposition. Thereafter, it takes all the base propositions and evaluates them. Only those, which depend on `init` proposition might be true, but this depends on the architecture of a particular network. For example, the only base proposition true in the initial state of the propositional network 4.6 is `( true P )`.

To determine the legal actions, the propnet performs a similar procedure as when computing the initial state. It marks all the input and base propositions in accordance to the current state. Then it takes every legal proposition for every player and evaluates them. Those, which are true, are the possible actions.

General games are simultaneous games. The next state is thus computed once all the players made their moves. The propnet then marks the input propositions according to these moves and evaluates every base proposition. This new set of values of input and base propositions identifies the following state.

Naturally, a particular state is terminal, if `terminal` proposition is true in this state. The utilities are then assigned to the players based on goal propositions. The propnet finds all these propositions and calculates their values. The goals which are true, define the utility for respective players.

4.4 Learning in games

Learning in games can help a player to extend his solving algorithm to perform more domain-specific informed search. The knowledge can be based on identifying distinct structures within a game description, such as factoring games; predicting how the game will evolve using some kind of statistic learning; or automatic generation of evaluation function.

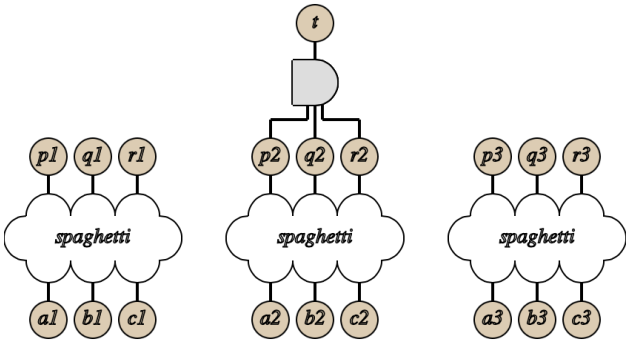


Figure 4.7. An example of identifying sub-games in propositional net of Multiple Buttons and Lights [9].

Factoring of games is a technique which enables the player to recognize independent components in compound games [59]. For instance, consider a game consisting of two Tick-Tac-Toe matches playing simultaneously. If a single Tic-Tac-Toe game has branching factor b, then the branching factor of the joint game is b^2 and the fringe of the game tree at depth d is therefore $(b^2)^d$. This can be significantly improved if the player realizes that there are actually two trees - both with branching factor b. The total size

of the fringe of these trees at depth d is now only $2b^d$. Another example is when the termination depends on one sub-game solely. Then playing a move which leads to one of other sub-games is literally a wasted action, which can be seen in Figure 4.7. Although factoring is a general technique, the most convenient form of identifying independent sub-games within one game is through propositional nets. The reason is that propnets depict the game through the properties of each game state, thus enabling to locate the sub-games more distinctly. The structure of the game projects itself on the structure of the propositional net. Factoring can be performed as a search in the propnet, which is clear when looking at Figure 4.7.

Learning can be utilized to predict the evolution of the game and push it towards more promising directions. Notably statistical learning is used to improve biasing in Monte Carlo simulation control, prioritizing the states and actions with higher statistical correlation in winning [60].

Perhaps the supreme way to involve the learning techniques in the general game playing is through their ability to automatically generate game-specific evaluation functions. Almost every kind of heuristic incorporates one or more of these universal methods:

- *Mobility.* The heuristic suppose the states with higher number of possible actions to be more beneficial.
- *Focus.* It is the opposite of mobility. It measures the narrowness of the game tree.
- *Goal.* The goal functions can indicate how close to the optimal terminal state the current state is.

A famous method of producing effective evaluation functions is based on detecting features from syntactic structure of the game. The learning algorithm first tries to recognize and exploit relational patterns such as successor relations and board-like grids. The features can be maximized or minimized, depending on what better serves the purpose of winning. The search is then performed, demanding multiple single-heuristic processes to pick up a best action. The suggested actions of each process are then compared, eventually choosing the best move [61].

For extending the feature-optimization heuristics, one can also utilize fuzzy logic and techniques for reasoning about actions to estimate a value to which a particular state satisfies the logical description of winning [62]. The state's distance to termination is critical, search algorithms profits from seeking terminal states with higher goal values and avoiding terminal states when goals are not yet reached or are too low.

Some approaches examine a simplified model of the game, which is much easier to evaluate. Then examine the relevant features of their respective heuristics by their performance on the model. The key aspect is the stability to work across various game scenarios emerging during the game-play [63]. Another way is to employ neural nets to construct improved evaluation functions [64], or even temporal difference learning [65].

Even with the ultimate evaluation function, the player is not safe from struggling with a horizon problem. A short-term's best action can generally prove to be unfortunate from the long-term point of view and searching just a little deeper would have had completely changed the strategy. The problem can be partially overcame by executing a variable depth search, examining the promising parts of the game tree more intensely.

The great challenge of general game playing still remains a transfer learning, as a method of applying knowledge already gained in previously played games. Algorithms capable to analyze analogies between various games can considerably profit from such ability [66].

To conclude, learning is an important part of general game playing for one reason – the player plays a broad range of diverse games and learning can significantly boost his

ability to play well, as there are no other options to create a domain-specific algorithm. On the other hand, it is widely known that more learning requires more data and time, which especially can be a great disadvantage for most general game players.

Chapter 5
Player Shodan

This chapter describes the process and practical difficulties which have to be overcame when creating a functional player capable to play a general imperfect-information games in GDL-II. It presents "Shodan", a player designed as a part of this thesis. The construction of Shodan is based on theoretical background presented in Chapters 2, 3 and 4.

5.1 Player construction

The player is implemented in Java programming language and he is dependent on two libraries. GGP Base[1]), a project of Sam Schreiber; and GT Library, a project of Computational Game Theory group from Agent Technology Center located at Czech Technical University in Prague.

GT Library contains domain-independent implementations of algorithms for solving extensive-form games and defines a general domain framework for describing the games. However, its formalization is different from the formalization of GGP. A domain[2]) is an implementation of an extensive-form game with imperfect information G = (P, T, I, b, u). Each domain has to define the rules of the game by describing the states of the game, available actions in each state, partitioning the states into the information sets, and modification of the states by actions. Finally, utility represents the outcome for both players when the game ends.

Every well-formed domain is obligated to implement 4 classes – GameInfo, GameState, Expander and Action. Each class corresponds to the respective part of the definition of imperfect-information game in extensive form. GameInfo defines the players and stores a domain-specific data. GameState is an implementation of a game state with an assigned player and information set. If the state is a terminal state, GameState determines its utility. GameState also defines the way the states change by applying actions. In every state of the game, Expander specifies the possible actions. Finally, Action class is an implementation of game's actions. As a part of this thesis was in the domain framework of GT Library created a distinct GGP domain, as a wrapper between these two formalisms.

GGP Base is a set of Java libraries and applications for building and testing perfect-information GGP players. It provides a common unified structure, which is obligated to run a player. This includes communication with the game servers, processing rules of games in GDL, representing games as state machines and simple automated theorem prover implemented in Java.GGP Base also contains several other tools, e.g. a utility for creating game descriptions, which is able to verify that the game presents a valid GDL; a tool for matches visualizations for both participants and spectators or automated tournament schedulers and result analyzers. The serious drawback of GGP Base is that it was primarily intended for use with GDL-I. Consequently, numerous classes of this library had to be modified to suit the framework of GDL-II.

[1]) http://www.ggp.org/developers/players.html
[2]) The whole description of a domain and a tutorial to build one is included in Appendix B.

5.1.1 Layers

Creating a GDL-II player is a complex process, which requires focusing on several important tasks mentioned in previous chapter. The relevant parts of the player's design must aim at solving such problems. With this in mind was suggested a structure of Shodan player, which is shown in Figure 5.1.

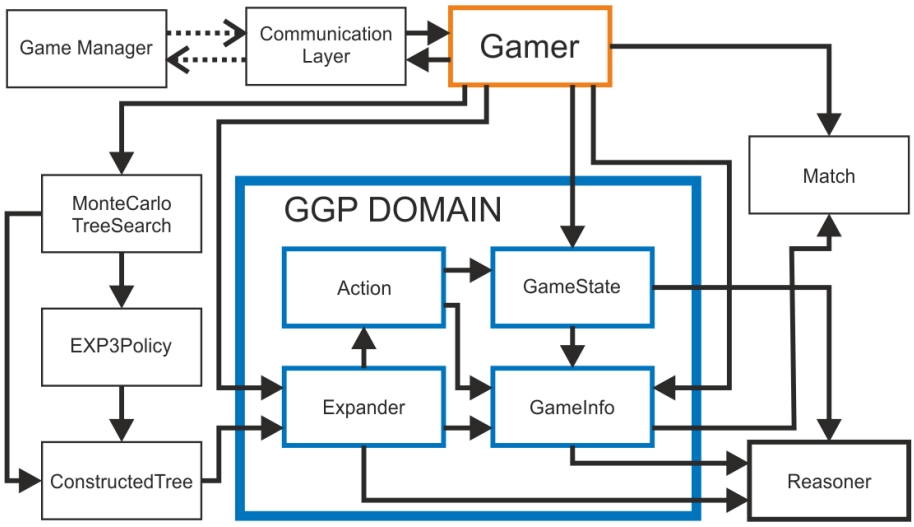


Figure 5.1. The inner structure of Shodan player.

The player's structure is composed of several interdependent layers, which together enable the player to participate in the matches. The central part of the player is based on the top of the Gamer class from the GGP Base, which was altered for use with domains of GT Library, instead of the state machines, for which it was originally intended. The player's layers surrounding the Gamer class are:

- Communication
- Representation and reasoning
- Game data structures
- Game solving algorithm

First, the player is obligated to understand the communication protocols to be able to play the tournaments on the game servers. The communication is handled by the communication packages from GGP Base, which were rewritten to suit the needs of GCL-II. In the structure of the player, the communication is incorporated into a communication layer.

The knowledge representation is done in GDL-II and player has to correctly figure out the way the game is played. The game is represented by a state machine or by a propositional network. Both of them have their own specific practical advantages and disadvantages and a respective reasoner (an automated theorem prover from GGP Base or a propositional net search algorithm) has to be appropriately connected with the GGP domain of GT Library.

The game data structures are distributed between several parts of the player's structure. The GDL description, the time limits and the histories of percepts and actions are stored in the modified Match class of GGP Base. The processed description is integrated into respective classes of GGP domain. The main and the greatest structure – the game tree – is handled by the Monte Carlo partial tree (on the figure called Constructed tree). The player builds and updates the game tree, as the match progresses.

The data are constantly accessed by both the communication handlers and the solver. The handlers identify the current information set in accordance with the data received from the server and dispose the tree above it. This includes all lateral branches, which do no pass the current information set. It means, that they are not longer reachable and can be pruned. The solver continues with construction of the game tree below.

For solving the game and planning the next moves Shodan uses Monte Carlo tree search method with EXP3 selection policy. These algorithms are a part of GT Library and had to be just a slightly modified because of the opponent modeling described later in this chapter.

5.2 Communication

In the moment the player invokes, the communication layer starts to listen at port 4001. It waits for an `info` or a `start` message. When the message is received, the player reacts accordingly to the protocol. It parses the massage, identifies its type and appropriate components of the message. If the `info` message is identified, it responds with the information about the player, thus confirming that the player is ready for a match. If a `start` message occurs, the layer acknowledges the main Gamer class, that a match is about to begin. The central class then has to perform all the necessary preparations, i.e. invoking the reasoner with the GDL description, setting the player's role in the match and registering the time limits. The start clock defines the remaining time before the game actually starts, which can be used for launching the computations and learning. Within this limit the player replies `ready`.

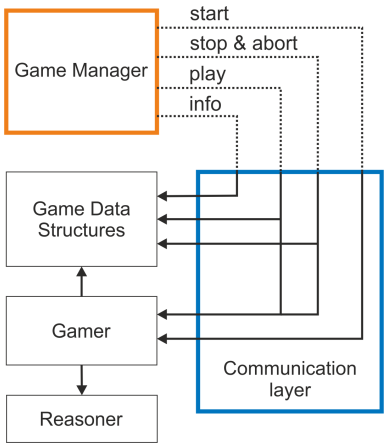


Figure 5.2. The structure of a communication layer.

The game starts and progresses with a `play` message. Once detected, the layer initiates the update of the current information set, which results in pruning the game tree; and notifies the Gamer class that the selection of the next move can begin. After receiving a `stop` or an `abort` message, the communication layer informs about it and causes a deletion of the redundant game data structures. Then Shodan returns to the manager `done`.

The layer is also able to detect irrelevant messages in the current time, for example the calls for another match.

5.3 Representation and reasoning

The player is able to use both kinds of game representations as described in previous chapter – a state machine and a propositional network. The GGP domain in the GT

Library is connected to one of the representations, which are carried out by the GGP Base and uses it to reason about the game description. The representations themselves need no cache, because the tree is cached by Monte Carlo partial tree.

5.3.1 Propositional network

Reasoning about game descriptions with propositional networks is a little more complex than reasoning with theorem provers. First, in the moment the communication layer recognizes a match-starting request, the propositional net has to be created. For this purpose Shodan uses PropNetFlattener from the GGP Base. The flattener takes the GDL description and instantiates the game domain. Once the instantiation is done, the individual propositions which have been found are snapped together by various gates (namely conjunctions, disjunctions, negations and transitions) and the propositional net is created. The network then locates the propositions corresponding to important methods necessary for the game-play, e.g. the `legal` propositions, `sees` propositions, `does` proposition, etc.

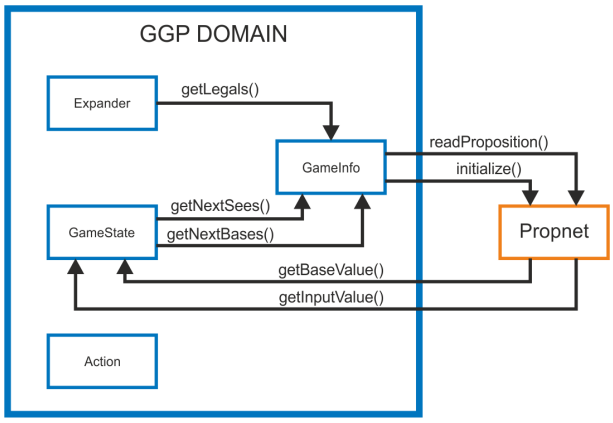


Figure 5.3. The structure of GGP domain designed to reason with propositional network.

For reasoning about the game states Shodan uses cached backward propnet traverse through the propnet architecture. This algorithm has been already described in section 4.3 of the previous chapter. To evaluate a desired proposition value, the domain performs a depth first search from the proposition until it reaches the bases or inputs. Then it propagates the value upwards. The search is cached in a sense that in every state of the game, Shodan registers value of every proposition, once calculated. So anytime the search reaches this proposition, it returns its value and does not have to continue deeper into the network architecture. It is reasonable, since a lot of propnet's inner propositions don't have to be evaluated numerous times. The disadvantage is that this approach is quite memory-consuming, even though the cache is implemented as a boolean array.

The GGP domain is modified for use with propositional networks. The GameInfo class has a backward traverse implemented and this method is called from other classes of propnet-based domain, how can be seen in Figure 5.3. So for example, if expander wants to determine the legal actions in a particular state, it asks the GameInfo class and this class performs the search through the network and returns the true propositions representing the legal actions. Another example are the modifications of game states by actions. The GameState class provides the GameInfo class the already calculated values of propositions (necessarily including the values of bases) and the actions performed by players. Based on this data, the GameInfo class then evaluates the values of base propositions in the next state.

The principal benefit of propositional networks is their swiftness. Propositional nets are faster than most theorem provers, if implemented efficiently. The speed of reasoning is essential for GGP, since it is a key to quick exploration of the game tree.

5.3.2 Propnet issues

Unfortunately, propnet flattening as implemented in the GGP Base experiences problems when encountering more complex game descriptions. Sometimes it creates more or less components then it should. For example in the Monty Hall problem, as specified on Dresden GGP server[1]), the flattener successfully finds:

```
( SEES CANDIDATE ( OPEN_DOOR 1 ) )
( SEES CANDIDATE ( OPEN_DOOR 2 ) )
( SEES CANDIDATE ( OPEN_DOOR 3 ) )
( SEES CANDIDATE ( CAR 1 ) )
( SEES CANDIDATE ( CAR 2 ) )
( SEES CANDIDATE ( CAR 2 ) )
( SEES CANDIDATE ( does CANDIDATE SWITCH ) )
( SEES CANDIDATE ( does CANDIDATE NOOP ) )
```

But it is unable to generate:

```
( SEES CANDIDATE ( does CANDIDATE ( CHOOSE 1 ) ) )
( SEES CANDIDATE ( does CANDIDATE ( CHOOSE 2 ) ) )
( SEES CANDIDATE ( does CANDIDATE ( CHOOSE 3 ) ) )
```

It seems that at a certain step the flattening algorithm does not consider the nested instantiations. This is not only the problem of `sees` predicate, since e.g. in Kuhn Poker (as specified in Motal's thesis [14]) the flattener finds several unprovable game-specific propositions.

Since the error could not be solved, the propnet was only used in experiments with games that are flattened correctly. The domain using the propositional net reasoning will be fully adapted as a default domain once the PropNetFlattener is fixed.

5.3.3 State machine

The theorem provers do not require an initialization similar to the propositional networks. For reasoning about games represented as state machines Shodan uses a theorem prover called Aima prover, which is contained in the GGP Base. Before the game starts, this prover performs a transformation of the GDL description into a functionally equivalent description so that it is able to reason about it. The reason for this behavior is that the Aima prover does not correctly apply `distinct` literals in rules in case they have not been bound yet. The same problem has the prover with `not` literals. The transformation works that way that it takes the `distinct` and `not` literals and move them later in the rule, so that they are listed after sentence literals which define those variables in the description. Aima prover works correctly and its implementation in GGP Base is stable, but its main disadvantage is its speed.

[1]) `http://ggpserver.general-game-playing.de/ggpserver/public/view_game.jsp?name=montyhall`

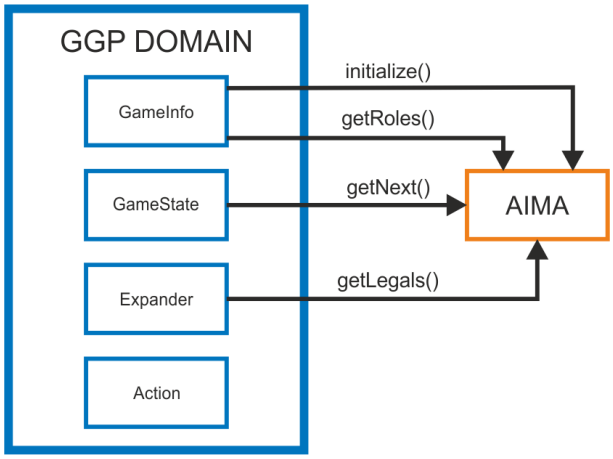


Figure 5.4. The structure of GGP domain designed to reason with Aima prover.

Just as with the propositional networks, also for AIMA prover was created a distinct GGP domain which integrates the methods for working with the prover. Its architecture can be seen in Figure 5.4. GameInfo class contains several queries which are later asked the prover to determine all important game-playing-related information. Specifically, this class contains following queries:

- DOES query – for selecting the desired actions;
- GOAL query – for computing goals for each player in current state;
- INIT query – for determining the properties of initial state;
- LEGAL query – for finding legal actions in the current state;
- NEXT query – for identifying true propositions of the next state;
- ROLE query – for extracting player roles from the game description;
- TERMINAL query – for evaluating the current state to be terminal;
- VARIABLE query – for allocating variables in compound queries; and
- SEES query – for revealing percepts in the current state.

For example, if the GameState constructor is called to crate a root state of a game, it selects an `init` query from the GameInfo class and asks the Aima prover about it. The prover then returns the propositions which are true in the initial state.

In the case the reasoning works properly, the player is able to generate the game tree and based on the percepts received from the GGP server he is capable to identify the appropriate information set in which he can be located.

5.4 Game solving algorithm

When a player successfully recognizes the information set of the game in which he can find himself, he can concentrate on choosing the move he considers the best with his current knowledge. This can be done by one of the methods described in Section 3.2 of Chapter 3. The selected algorithm is incorporated into the structure of a player as shown in Figure 5.5. It uses the game description reasoner to explore the game tree and the knowledge gained by learning to search the tree more efficiently.

Generally, a player can choose from two most significant approaches in solving the game. The first one can find only approximate Nash equilibrium and is based on some variant of Monte Carlo tree search. The second one can find an exact NE. However, the calculation is deeply affected by the game-play.

First, since the GGP runs on the tight time limit, solving the whole game for an exact NE remains impossible. Hence the players are forced to solve only a part of game and use various heuristics to evaluate the states after exploring the game tree to a particular depth. It is crucial to realize that the solvers do not calculate the Nash equilibrium of a game played, but an equilibrium of a subtree created as a restriction of an original game tree with the leaves values corresponding to the heuristic of that particular state. Using better heuristics produce more relevant evaluations of the states and therefore more sensible moves.

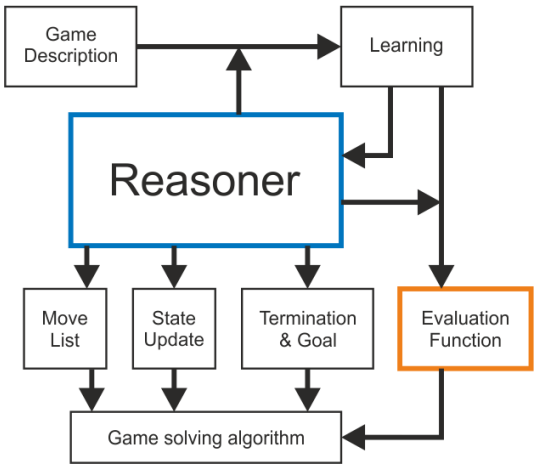


Figure 5.5. A general architecture of a planning player, inspired by [4].

Second, most algorithms computing NE are designed to solve the whole game or find an approximate solution, starting from the root state of the game. In GGP, time to calculate a move is given at every decision point. It seems rational to use already calculated results to get more precise predictions. Consequently, the algorithms has to be modified. The whole tree above the current state can be cut, because those parts of the tree will no longer be visited. The solver than runs on the subtree generated with the current state as a root.

Finally, even with an updated algorithm, the game-playing solvers struggle from another issue. As the game progresses, the player can find himself in a non-singleton information set. In this set, the player does not precisely know in which state he is. It means that the restricted game can have multiple roots. However, the solving algorithms can be executed from one state only. There are following options on which states from the information set the algorithm could be run:

- On every state of the information set.
- On the subset of all states.

The obvious negative of the first possibility is that the information sets can be huge[1]). In fact, to run the algorithm on the whole information set is in these cases simply not possible. When considering the second option, the question remains: how to select the subset. There are two possible approaches in doing this.

First option is to pick the states according to some rule or heuristic. The problem is the quality of the found solutions can be significantly affected by the selection criterion.

Second approach is to choose the states randomly. Random sampling can be done with paranoia or overconfidence [37]. Using paranoid sampling, the player expects

[1]) For example in the blind chess – kriegspiel, the cardinality of a single information set can exceed 500,000 [67].

his opponents to make rational decisions to maximize their final utility, based on the information provided in the current state of the game. Because the rational agent makes his moves according to the mixed or behavioral strategy, the probability distribution over the states in the information set corresponds with the likeliness of the game to be located in the particular state. This modeling works well when playing with a rationally reacting opponent, but can yield poor results against a random player.

Conversely, overconfidence suppose that the opponents do not consider the preference over different states in the information set and make them all same probable. This approach results in considering uniform probability distribution over the states in every information set. In contrast to the paranoid sampling, overconfidence works well against random players, but can fail when facing an experienced opponent.

However, this solution concept of applying a perfect-information algorithm on the information set can suffer from following theoretical problems – strategy fusion and non-locality [68]. Strategy fusion error is caused by the solving algorithm assuming that it can make the right decision in every state of the game. But this is not true, due to the fact that the information sets generally contain more than one state, and these states can have different perfect-information strategies. Playing a certain action in an information set can generate different expected outcomes even if the opponent 's strategy is pure. Consequently, fusing the optimal strategies from different states from one informations set does not guarantee the final strategy being optimal. An example of a strategy fusion error is given in Figure 5.6. In this game an agent Phil faces the decision of playing an action in his upper information set . First possibility is to take the left action, which results in winning in both states of the information set. But for the perfect-information algorithm, playing the right action also yields a victory, because it assumes that Phil is able to make a right decision in his lower information set. It is not true, because the outcome of playing the same action in different states of this set is opposite.

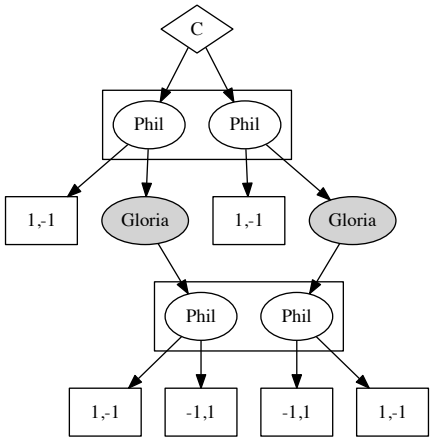


Figure 5.6. An example of a strategy fusion error in a simple game, inspired by [68].

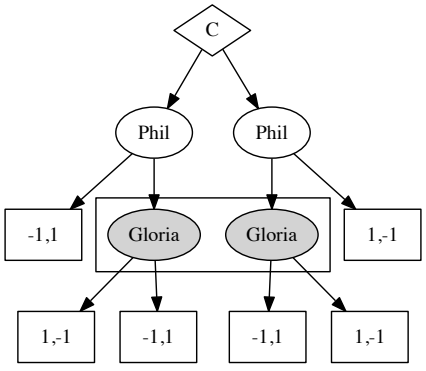


Figure 5.7. An example of a non-locality error in a simple game, inspired by [68].

Non-locality error is based on an assumption of a perfect-information algorithm, that any game state can be analyzed based on the values of terminals of its own subtree. But this is false too, since the outcome can also depend on other parts of the game tree,

covered in the information set. An example of a non-locality error is shown in Figure 5.7. In this simple game Gloria knows, that Phil is able to distinguish between his two states. Consequently, if he takes a right action and not the left one, she can be certain that the game is played in the left branch of the game tree. But the perfect-information solving algorithm analyzing her information set is not aware of it and would have made its move purely randomly. However, there exists numerous techniques for detecting how much can these errors influence a particular game, so that the player can draw a conclusion about the rationality of his behavior [68].

There is also a possibility to convert the perfect-information game-playing algorithms into the imperfect-information players using the technique called HyperPlay [13], which was noticed in section 1.1.

5.4.1 MCTS with EXP3

As a planning and solving technique was chosen Monte Carlo tree search method, which uses EXP3 selection policy during the selection phase of the algorithm. The main reason is that Monte Carlo methods are domain-independent, work satisfactorily under time pressure and can be adapted for sequential launching. The EXP3 selection criterion is most probable to converge to Nash equilibrium, as it has been said in Chapter 3. The solver of Shodan is adapted to these kinds of games:

- one-player stochastic
- two-player
- two-player stochastic

During the game-play, the player is aware of the whole information set in which he might find himself. Before MCTS can be run, the player has to decide, which state from the current information set should be chosen as a root. For this purpose the player uses belief distribution, an approach based on opponent modeling.

5.4.2 Calculation of belief distribution

Calculation of belief distribution is a method for determining a probability of being located in the particular state of the current information set. It is build on an assumption, that the opponent is likely to choose the best action that he can with his current knowledge.

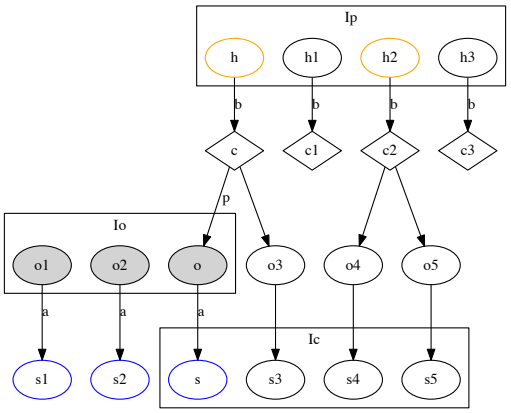


Figure 5.8. The process of calculation belief distribution over a new information set.

When the game starts, the distribution over the beliefs in the information set is uniform, as no one made his move. In fact, the information set can be singleton, if the player is the first player to move in the constructing game tree. The belief over an arbitrary state h in the initial information set I_i is thus calculated as follows:

$$b(h) = \frac{1}{|I_i|}. \tag{5.1}$$

From this point, Shodan proceeds inductively. As the game progresses and the beliefs over the previous information set are known, the player has to compute new decomposition of beliefs over a current information set I_c. To do this, he performs a depth first search from the previous information set I_p after playing the action b he chose in the previous round. In the two-player stochastic game to the new belief in the state $s \in \mathrm{I}_c$ contributes three components:

$$b'(s) = b(h)\ p(m)\ p(a). \tag{5.2}$$

First component corresponds to the normalized belief of a predecessor of s, $h \in \mathrm{I}_p$. The belief of h is normalized over all states $\mathrm{h}_i \in \mathrm{I}_p$ among their successors belong any state in I_c, denoted as $\mathrm{I'}_p \subseteq \mathrm{I}_p$:

$$b(h) = \frac{b(h)}{\sum_{h_i \in I'_p} b(h_i)}. \tag{5.3}$$

Second component is a probability of the Nature player playing an action $p \in \mathrm{A(c)}$ which leads to the subtree where is located state s. In GGP, the distribution over the feasible actions is always uniform, thus:

$$p(m) = \frac{1}{|A(c)|}. \tag{5.4}$$

Finally, third component signifies the willingness of the opponent to take an action a, which leads to the subtree containing state s. For calculation the probability, that the opponent permitted reaching state h is utilized a number of play-outs performed by Monte Carlo tree search. When denoting $C(h', a)$ a child of h' after playing action a and $n(h')$ the number of play-outs which were executed through h', the probability that the opponent will play action a in state $o \in \mathrm{I}_o$ is:

$$p(a) = \frac{\sum_{o_i \in I_o} n(C(o_i, a))}{\sum_{o_i \in I_o} n(o_i)}. \tag{5.5}$$

The values computed during the run of the depth first search might not yield a proper probability distribution. The final belief of a state s should be normalized, for the beliefs of the whole information set I_c to sum up to 1.

$$b(s) = \frac{b'(s)}{\sum_{s_i \in I_c} b'(s_i)} \tag{5.6}$$

The whole computation of new beliefs is depicted in Figure 5.8. In this example the states in $\mathrm{I'}_p$ which leads to I_c are $\{h, h_2\}$, the belief of h is thus divided by the sum of beliefs of these two states. There are two feasible actions in state c. Consequently, the probability of playing action m is $\frac{1}{2}$. Finally, the opponent is likely to take action a in state o with probability $p(a) = \frac{n(o)+n(o_1)+n(o_2)}{n(s)+n(s_1)+n(s_2)}$.

When the selection of a root is resolved, MCTS launches the repetition of iterations. These 2 phases are then repeated until the time allocated for a move expires. The action is chosen randomly with the distribution over the number of selections in each of possible actions in the current information set.

5.4.3 Calculation exceptions

However, there are some situations in which these precise formulas are not exactly applicable. These situations can occur especially in cases when Monte Carlo tree search is unable to perform enough iterations of tree expansion and evaluation. The phase of the belief computation, in which is calculated the chance that opponent is likely to take an action a, any state s $\in$ I_o can suffer from following problems:

- MCTS has not expanded the node C(s,a) yet;
- MCTS did not perform any iterations through s; or
- MCTS did not perform any iterations through C(s,a).

In these cases Shodan assumes the distribution over the states to be uniform.

5.5 Playing matches

The game starts, when the player receives a `start` message. Once that occurs, the communication layer informs the central Game class and this class instantiates all necessary classes for playing. This includes the classes from the GGP domain, which store the game characteristics: GameInfo and the rootstate as an instance of GameState. Simultaneously the reasoner is initialized (it means that either the propositional net is constructed or the prover performs the transformation of GDL). Monte Carlo tree search then starts the construction of the game tree and the player is ready to begin playing.

The game progresses as the player receives a `play` message from the game server. First, the player updates the history of percepts perceived so far. According to this information he passes through the next layer of the game tree and finds all states which belong to the current information set.

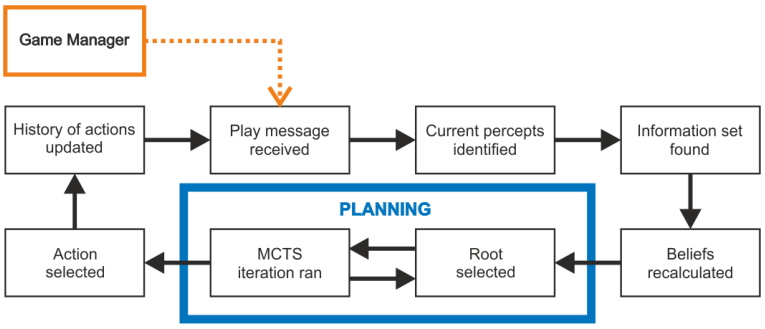


Figure 5.9. The process of choosing next action during the game-play.

During the search the player detects and cuts all branches of the game tree which are certain to be unreachable. At the same moment Shodan computes a belief over the selected states based on the algorithm described in the previous section. This helps him keep the track of the probability distribution determining how will the states be consequently picked as the roots of each MCTS iteration. Once the whole IS is collected, the solving algorithm runs in the loop, repeatedly selecting states from the current information set as roots and performing Monte Carlo iterations until the time remains. In the last second, the player calculates the strategy derived from the distribution of MCTS iterations over actions applicable in the current information set, chooses his next move, sends it to the manager and awaits the next `play` message.

When the game terminates, all game data structures built during the match are disposed.

Chapter 6
Experiments

This chapter describes the experiments performed with the player. First, it states the conditions under which the experiments were conducted. The chapter specifies the programs used for testing, the games played, the opponents, and also the time limits. Second, the results are given, together with a concise analysis.

6.1 Settings

Several experiments were carried out to evaluate the player presented in the previous chapter. Most games, which are being played in the imperfect-information GGP tournaments, come from the Dresden GGP server[1]). The Dresden GGP server is a tournament server for testing general game players on a broad range of different games and against other rational players. The matches are scheduled by the server or manually. The project also contains GameController[2]), a standalone Java program for running the matches locally. This program was used to perform all tests.

There are 25 GDL-II games currently present on the Dresden GGP server. The games vary from very simple ones like guessing the value of a rolled dice, up to relatively large games like Mastermind. Three of them were picked and the player was evaluated on each.

Three players were chosen as opponents for Shodan, specifically random, TIIGR [14] and Shodan itself. Shodan player comes in two version. First version uses propositional networks as the game representation and reasons by a backward search algorithm. The second version represents the games as state machines and uses automated theorem prover. Because the propositional network provides faster reasoning, the propnet-based Shodan is the primary player. If the GGP Base flattener is able to process the game description, the propnet-based Shodan plays the particular game. Which version of the player participated in the game is stated in the description of each game.

Four parameters were set in each experiment. The time limit, the game, the role of Shodan in the game and the opponent. The exploration/exploitation coefficient γ was set to 0.05, based on several initial experiments. Each game was played 240 times, players exchanged their roles and faced different time limits. The experiments were performed on Windows 7 64Bit running on processor i5 4670K, 4Ghz, using 16GB DDR3 1600Mhz RAM.

6.1.1 Time limits

Time limits are the most significant parameter which affects the performance of Shodan. The reason is, that with shorter limits the reasoner produces smaller amount of MCTS iterations. This can lead to imprecise decisions and unreasonable opponent modeling. As the time limits get larger and the number of iterations grows , more iterations enable

[1]) `http://ggpserver.general-game-playing.de/ggpserver/`
[2]) `http://sourceforge.net/projects/ggpserver/files/gamecontroller/`

more precise convergence to an approximate Nash equilibrium. Specifically, following play-clock time limits for each move were chosen:

- 30s;
- 60s;
- 120s; and
- 200s or 300s.

Because none of the players uses learning, the start-clock was fixed on 30 seconds. Usually, all players responded with `ready` within the first 10 seconds after the start of a game.

6.1.2 Games played

Following three games from the Dresden GGP server were chosen:

- Latent Tic-Tac-Toe
- Monty Hall problem
- Meier

Latent Tic-Tac-Toe is a blind variant of Tic-Tac-Toe. It is played by two players on 3x3 board and in every turn, only one player tries to mark a cell. The other one is only able to perform a noop action. If the player is successful in marking the selected field, the other one takes an action in the next round. Otherwise, the player has to decide again. The goal is the same as in perfect-information Tic-Tac-Toe, which is to place three follow-up marks in a horizontal, vertical, or diagonal row. The winner obtains utility 100, the loser gets 0. If no player is successful in creating a row of his marks, the game results in a draw, rewarding both players 50. The propnet-based version of Shodan plays the matches of Latent Tic-Tac-Toe, because the flattener is able to process this game's description.

The Monty Hall problem is a brain teaser named after Monty Hall, a host of American television game show Let's Make a Deal. It is a form of probability puzzle [69], which goes as follows. Suppose you are standing before three doors and you have to pick one. Only behind one of them is a hidden treasure, remaining two do not conceal anything. The chance of success $\frac{1}{3}$, that is clear. But once you make your choice, one of the other two doors opens and there is just empty space there. Then you are given a chance to reconsider your turn. Should you switch?

This teaser has become famous after its analysis in Parade magazine in 1990. One of the readers asked this question and the response [70] was that it is reasonable to change your decision and pick the other door. In this game the player scores 100 if he chooses the right door and 0 otherwise. Prover-powered Shodan competes in these matches because the impossible to create a propositional net for this game.

The third game is Meier, a two-player game also called Lier's Dice [71]. In this game, the first player rolls two dices and looks at the result. Then he can claim the true value of the dices or bluff. One way or another, the second player can accuse the first player of not telling the truth or roll the dices. Regardless of the result, he has to declare a strictly greater value, be it the true result or just a lie. The game ends when one of the players indicts the other one of bluffing. The dices are revealed and if the accusation is true, the claiming player obtains utility 100 and the other one's outcome is 0. Otherwise, the falsely accused player gets 100 and the other player quits with 0. Prover-based Shodan plays this game.

The exact definitions of all these games in GDL-II can be found on the Dresden GGP server.

6.2 One-player games

Monty Hall problem is an already solved game with one strictly best strategy. That optimal action is to change your mind and pick the other door. With such behavior, the player has 66% chance to win. The results prove that Shodan almost always changes, since this action is strictly better than to keep the original choice.

	30 s	60 s	120 s	300 s
candidate	63.33 ±17.24	66.66±16.87	60.00±17.53	63.33±17.24

Table 6.1. Ratio of wins and losses in Monty Hall problem with 95% confidence intervals [%].

Results correspond with the expectations also due to the fact that the game is small enough to perform a lot of MCTS iterations even within 10 seconds or less.

6.3 Two-player games

6.3.1 Against random

Matches against a random player are a convenient way to test a player against a basic opponent. The reason is that since the random player does not improve with longer time limits, the increase in performance of rational agents should be clearly visible. Another advantage is that the randomness of this player guarantees that after numerous matches, the outcome of any intelligent player has a sufficient information value.

Shodan performs quite well against a random player and it can be seen that the player improves his win ratio with longer computations. The assumption that the opponent will react rationally can be a disadvantage against random agents, but in this case it was not confirmed.

First set of tests against a random player was performed on Latent Tic-Tac-Toe. Latent Tic-Tac-Toe is a relatively large game with a great state space. This can affect the efficiency of deciding under shorter time limits. However, it seems that against random player, even the lower number of iterations yields almost satisfactory results.

	30 s	60 s	120 s	200 s
X player	80.00±12.07	86.66±9.32	90.00±8.66	86.66±10.43
O player	68.33±9.95	71.66±10.17	75.00±10.24	85.00±9.57

Table 6.2. Ratio of wins and losses in matches of Latent Tic-Tac-Toe against random with 95% confidence intervals [%].

In Meier, Shodan absolutely dominates over the random player in both roles. Whether Shodan plays as the first or the second player, it usually wins. But in fact, this is not so surprising, because the strategy calculated without a lot of iterations (i.e. without the significant affection on calculation of belief distributions) says to claim the truth when rolling the dices and accuse the opponent of lying when not on the round. With this simple strategy, Shodan was able to win most matches, independently on the role.

The precise number of iterations in Meier varied from tens of thousands in the beginning of the match to hundreds of thousands in the middle and millions in the ending.

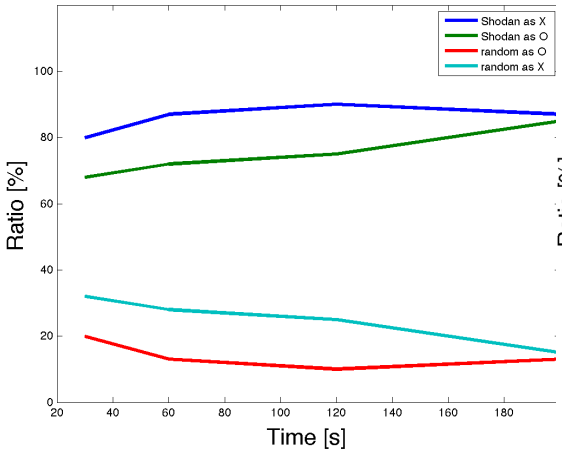


Figure 6.1. The ratio of wins and losses in matches of Latent Tic-Tac-Toe against random.

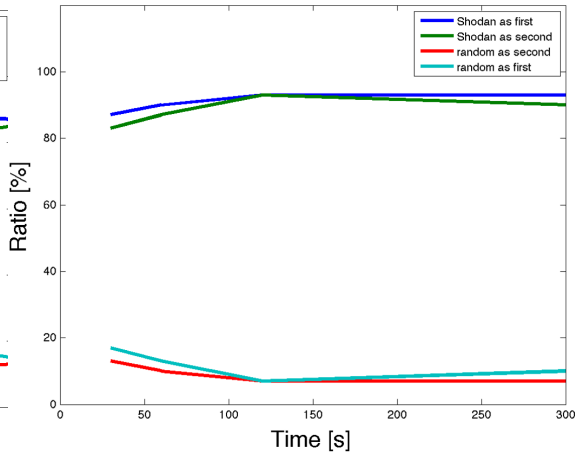


Figure 6.2. The ratio of wins and losses in matches of Meier against random.

	30s	60s	120s	300s
first player	86.66±12.17	90.00±10.74	93.33±8.93	93.33±8.93
second player	83.33±13.34	86.66±12.17	93.33±8.93	90.00±10.74

Table 6.3. Ratio of wins and losses in matches of Meier against random with 95% confidence intervals [%].

To sum up, the player performs well against a random player in both Latent Tic-Tac-Toe and Meier. The reason is that although the state space is large, Shodan is able to perform enough iterations to react rationally. Furthermore, the assumption that the opponent behaves rationally does not negatively affect the sampling of the state space of both games.

6.3.2 Against TIIGR

Matches against TIIGR were the first real challenge. TIIGR is also based on MCTS, but she[1]) involves UCT selection policy, while Shodan uses EXP3 criterion. The whole architecture of Shodan is more complex than the design of TIIGR player. As a result, Shodan is not able to perform the same amount of MCTS iterations as TIIGR. While Shodan executes an equal or greater number of repetitions at the beginning of both Latent Tic-Tac-Toe and Meier matches, as the game progresses, the player is unable to compete with TIIGR in this aspect. This leads to a greater number of losses in Latent Tic-Tac-Toe, especially in matches with longer time limits, where TIIGR is able to compare larger number of possible outcomes.

	30s	60s	120s	200s
X player	70.00±12.95	73.33±12.19	75.00±12.21	71.66±13.84
O player	21.66±11.2	28.33±13.84	23.33±11.25	16.66±9.78

Table 6.4. Ratio of wins and losses in matches of Latent Tic-Tac-Toe against TIIGR with 95% confidence intervals [%].

[1]) In his thesis, Motal considers his player to be female.

The first player possesses a considerable advantage in this game because of the opportunity of the first move in each round. This means that he does not have to think so much about blocking his opponent and he can focus on finding actions leading to victory. When playing as X player, Shodan wins. That fulfills expectations, since the first player has a great advantage over his opponent. As the O player, Shodan loses a lot. The main reason is that during 30-seconds-per-move match, the player is able to produce only tens of thousands of iterations. That is extremely insufficient. There is not enough data for accurate opponent modeling and it can result in considerable deviations in the model. This problem is not so noticeable during the first half of the match, when the cardinality of information sets does not exceed 50. However, when the game progresses to the middle and further, the number of states in each information set significantly increases and fluctuations in the middle can cause disproportional preference over some states. This only confuses the player even more. When the player is not able to perform enough MCTS iterations, calculations the belief distribution seems to be quite a drawback. It could be, for the most part, substituted by uniform probability distribution over the states.

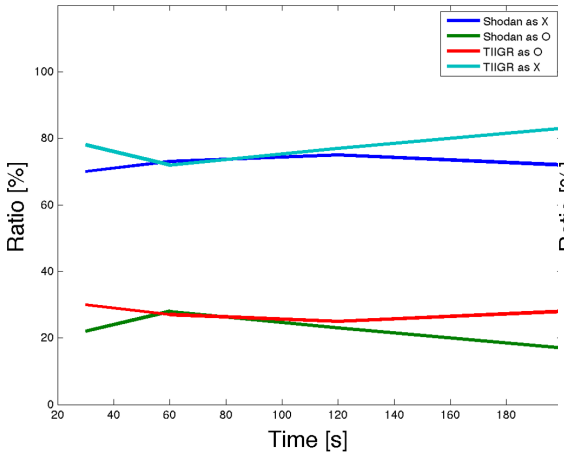


Figure 6.3. The ratio of wins and losses in matches of Latent Tic-Tac-Toe against TIIGR.

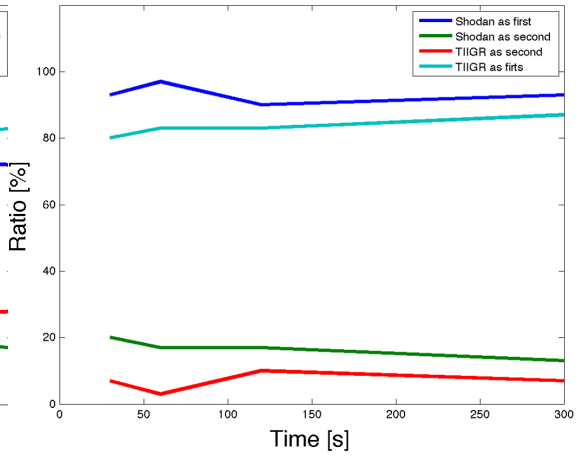


Figure 6.4. The ratio of wins and losses in matches of Meier against TIIGR.

On the other hand, the performance of Shodan in Meier is interesting. When claiming the true value as the first player, Shodan wins almost in every match. This is due to the fact that TIIGR uses perfect information sampling when choosing the root states of the Monte Carlo iterations in every information set of the game. As a result, TIIGR loses the track of behavior of rational opponents. She supposes that since there are 36 possible outcomes of rolling two dices, the opponent will most likely bluff, so she calls a bluff [14]. But that is not true, because every rational opponent will most probably tell the truth, since his opponent has a lower chance to be able to speak truthfully in the next round. Shodan won most games in which he was the first player this way, that is by claiming the real value of the dices and waiting for TIIGR's accusation.

	30s	60s	120s	300s
first player	93.33±8.93	96.66±6.43	90.00±10.74	93.33±8.93
second player	20.00±14.31	16.66±13.33	16.66±13.33	13.33±12.16

Table 6.5. Ratio of wins and losses in matches of Meier against TIIGR with 95% confidence intervals [%].

The position as the second player is less positive. The fact is, that in the beginning of the game, which is the most crucial part in this case, Shodan was not always able to perform enough iterations to evaluate all states in the closest deeper layer of the game tree and thus the belief distribution often degraded to uniform sampling. Nevertheless, instead of claiming a bluff, Shodan took the dices and inverted the game into a situation where he is the casting player in all games he won. The result is, then, the same as when playing the first role. To sum up, Meier is an interesting game with potential to become a popular game for testing the calculation of belief distributions.

6.3.3 Against itself

Shodan also performed several experiments against itself. Prover-based and propnet-based implementations competed in Latent Tic-Tac-Toe. Propositional networks are generally faster reasoners than automated theorem provers, so this comparison is interesting due to asymmetric roles of the players in this game.

The experiments show that even with more efficient reasoner, the first moving player still has a significant advantage over the second agent. However, with growing time limit, the differences between reasoners increase and in experiments with 200 second per each move a variant of Shodan reasoning with propositional networks strictly dominates over the prover-based implementation.

	30s	60s	120s	200s
`X player`	50.00±14.86	65.00±11.65	71.66±12.15	78.33±12.15
`O player`	28.33±11.2	36.66±10.43	45.00±10.87	51.66±12.85

Table 6.6. Ratio of wins and losses in matches of Latent Tic-Tac-Toe against Shodan with 95% confidence intervals [%].

The performance of playing general imperfect-information games with Monte Carlo tree search planning methods is highly dependent on the speed of reasoning. It can be seen from the experiments that fast exploration of the game tree is really fundamental for successful decisions. The example of Meier matches with TIIGR also confirmed the sensibility of calculations belief distributions based on opponent modeling.

Chapter 7
Conclusion

An approach to designing, testing and evaluating domain-independent playing algorithm capable to participate in matches of general imperfect-information games was proposed in this thesis. The main contribution of this work to the field of general game playing systems is the creation of Shodan. This player is the first GDL-II player which utilizes propositional networks as game representations, benefits from asymmetric preference distribution over states in information sets based on opponent modeling, and plans the next moves with methods most likely to converge to an approximate Nash equilibrium.

Game	Against random	Against TIIGR
Latent Tic-Tac-Toe	80.41	47.48
Meier	89.95	54.99

Table 7.1. An average utility of Shodan in the experiments.

The algorithm is based on three key features. The first one is *propositional network representation*, the ability of Shodan to represent a game as a gate array. However, this approach requires several optimizations in the game description and it has to subsequently transform the program into an array before it can be even used for reasoning. This considerable initial inconvenience is computationally expensive, but the resulting advantage of propositional nets is their speed and the natural ability to be implemented on hardware.

The second essential feature of the player is *calculation of belief distributions*. Belief is a value which represents the conviction of a player that he is located in a given state of the game. Set of beliefs in each information set represents the probability distribution over all states in this set. Sampling the information set using beliefs helps the player to draw more precise conclusions about his current location in the game space. The calculation is built on a simple assumption that the states belonging to a single information set are not necessarily equally probable. The reason for this is that the opponent acts rationally and his behavior causes some branches of the game tree to be preferred over others. As a result, the sampling of any information set should not be always uniformly random. Nevertheless, the method is directly dependent on opponent modeling. Shodan utilizes the data provided by Monte Carlo tree search for this purpose.

Finally, the most important part of Shodan is the planning core, i.e. the solving algorithm. The player uses *Monte Carlo tree search* method with *EXP3 selection policy* for picking the node for expansion during the run of each MCTS iteration. Monte Carlo techniques are a well-known class of stochastic computational algorithms that use great number of repeated random sampling. Selection policies are crucial for optimal run of Monte Carlo tree search iterations, because they balance out the exploitation of states with high value estimates and exploration of states with uncertain value estimates. The problem with imperfect-information games is that the UCT formula, widely used

for the selection phase of MCTS in perfect-information games, has been proved not to converge to Nash equilibrium. EXP3 selection criterion is currently the policy which is the most believed to converge; however, the proof is still missing.

The thesis also presents the results of several experiments performed with the player. Three games of different kinds – single-player game, two-player deterministic game and two-player stochastic game – were played. Shodan ran the games with several time limits and the opponents of Shodan were: random player, TIIGR and Shodan itself. The results show two important things. First, the calculation of belief distributions works well against a rational opponent, but it requires a lot of MCTS iterations to produce relevant predictions. Second, compared to a UCT-based MCTS player, Shodan struggles with the efficiency of planning under shorter time limits. This is mainly caused by the computational complexity of EXP3 selection criterion and the calculation of beliefs, which requires additional game tree traversals.

7.1 Future work

Judging from the presented performance of Shodan, there is certainly a vast space for future improvements. The already implemented general game playing algorithm can be extended in several ways.

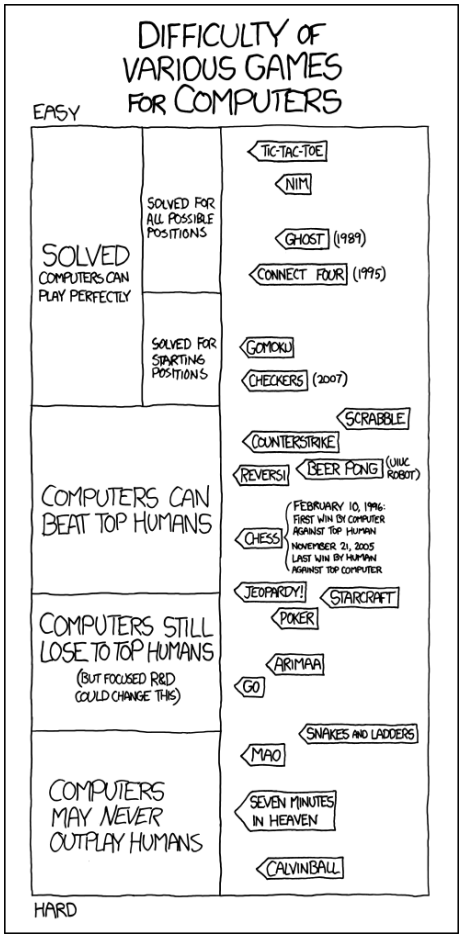


Figure 7.1. Ability of computers to play games. [72]

First, the most significant bottle-neck of Shodan is the performance of reasoning about GDL. One possibility is to increase the computing speed of construction of game trees by using field-programmable gate arrays (FPGA) to represent the propositional network. This will allow more states to be expanded within the time limit.

Another way is to focus on precise calculation of belief distributions over sets of states which the player cannot distinguish between. Using opponent modeling is essential for creating rational preferences over states in information sets. The player is then able to consider aspects which can describe the real state of the game with better precision.

Finally, an efficient generation of domain-specific evaluation functions based on detecting of features from the syntactic structure of a game would increase the performance. These learning algorithms try to recognize and exploit relational patterns such as successor relations and board-like grids. They are useful for adapting domain-independent algorithms to a specific game by exploiting its unique properties.

The performance of various game AIs in different games is presented in Figure 7.1. The ultimate goal is to create a domain-independent algorithm, which is capable to compete in most of these games even with the human top players.

References

[1] Raymond Redheffer. A Machine for Playing the Game Nim. *The American Mathematical Monthly*, 55(6):343–349, 1948.

[2] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, 1995.

[3] Murray Campbell, A. Joseph Hoane, Jr., and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1-2):57–83, January 2002.

[4] Michael Thielscher. General Game Playing in AI Research and Education. In *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 7006, pages 26–37. Springer, 2011.

[5] Jacques Pitrat. Realization of a general game-playing program. In *IFIP Congress (2)*, pages 1570–1574, 1968.

[6] Michael R Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26(2):62–72, 2005.

[7] Jonathan Levin. Extensive form games, 2002.

[8] Tejvan R Pettinger. Economics revision guide, 2008.

[9] Michael Genesereth and Michael Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2013.

[10] Stefan Edelkamp, Tim Federholzner, and Peter Kissmann. Searching with partial belief states in general games with incomplete information. In *KI*, pages 25–36, 2012.

[11] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. *AAAI*, 2007.

[12] M. Thielscher. *Reasoning Robots: The Art and Science of Programming Robotic Agents*. Applied Logic Series. Springer, 2006.

[13] Michael Schofield, Timothy Cerexhe, and Michael Thielscher. Hyperplay: A solution to general game playing with imperfect information. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[14] Tomáš Motal. General Game Playing in Imperfect Information Games. Master's thesis, Czech Technical University in Prague, 2011.

[15] Hilmar Finnsson. *Simulation-Based General Game Playing*. PhD thesis, Reykjavík University, 2012.

[16] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009.

[17] Roger B Mayerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.

[18] C. Crawford. *Chris Crawford on Game Design*. NRG Series. New Riders, 2003.

[19] Encyclopaedia Britannica Online. Finite game. http://www.britannica.com/EBchecked/topic/207387/finite-game, 2014.

[20] Robert J. Aumann. What is game theory trying to accomplish?, 1985.

[21] Robert Aumann and Adam Brandenburger. Epistemic conditions for nash equilibrium. *Econometrica*, 63(5):1161–1180, 1995.

[22] J Nash. Non-Cooperative Games. *Annals of Mathematics*, 54(2), 1951.

[23] Yoav Shoham and Kevin Leyton-Brown. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan and Claypool Publishers, 2008.

[24] George J. Mailath, Larry Samuelson, and Jeroen Swinkels. Extensive form reasoning in normal form games. Working papers, Wisconsin Madison - Social Systems, 1990.

[25] William Poundstone. *Prisoner's Dilemma: John Von Neumann, Game Theory and the Puzzle of the Bomb*. Doubleday, New York, NY, USA, 1st edition, 1992.

[26] Ernst Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth International Congress Mathematics*, pages 501–504, Cam
-bridge, 1913. Cambridge University Press.

[27] T.C. Schelling. *The Strategy of Conflict*. Harvard University Press, 1980.

[28] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

[29] Paul Milgrom and Robert Weber. A theory of auctions and competitive bidding. *Econometrica*, 50:1089–1122, 1982.

[30] Hugh Ward. Game theory and the politics of global warming: the state of play and beyond. *Political Studies*, 44(5):850–871, 1996.

[31] R. Selten. *An Oligopoly Model with Demand Inertia*. Center for Research in Management Science, University of California, 1968.

[32] J. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[33] Stuart Russell. *Artificial intelligence : a modern approach*. Prentice Hall, Upper Saddle River, NJ, 2010.

[34] H.W. Kuhn, K.J. Arrow, and A.W. Tucker. *Contributions to the Theory of Games*. Number v. 2 in Annals of mathematics studies. Princeton University Press, 1953.

[35] Katta Murty. *Linear programming*. Wiley, New York, 1983.

[36] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, Jiří Čermák, and Michal Pěchouček. Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. *In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 2013.

[37] Austin Parker, Dana S. Nau, and V. S. Subrahmanian. Paranoia versus Overconfidence in Imperfect-Information Games. In Rina Dechter, Hector Geffner, and Joseph Y. Halpern, editors, *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl*, pages 63–87. College Publications, 2010.

[38] Michael Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[39] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret Minimization in Games with Incomplete Information. *Advances in Neural Information Processing Systems 20 (NIPS)*, 2007.

[40] Nick Abou Risk and Duane Szafron. Using counterfactual regret minimization to create competitive multiplayer poker agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 159–166, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[41] Mohammad Shafiei Khadem. Simultaneous move games in general game playing. Master's thesis, University of Alberta, 2010.

[42] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *In: Proceedings Computers and Games 2006*. Springer-Verlag, 2006.

[43] Historical kgs ratings of a few bots. `http://senseis.xmp.net/?KGSBotRatings`. Accessed: 2014-04-12.

[44] Wikimedia Commons. Steps of monte-carlo tree search, 2013.

[45] Levente Kocsis and C Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, 2006.

[46] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

[47] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing uct versus cfr in simultaneous games. 2009.

[48] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The non-stochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003.

[49] Viliam Lisý, Vojtech Kovarík, Marc Lanctot, and Branislav Bosanský. Convergence of monte carlo tree search in simultaneous move games. *CoRR*, abs/1310.8613, 2013.

[50] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):120–143, 2012.

[51] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012.

[52] Peter A Flach. *Simply logical - intelligent reasoning by example.* Wiley professional computing. Wiley, 1994.

[53] P.K. Dick. *Valis.* Vintage. Knopf Doubleday Publishing Group, 2004.

[54] Adrian Lancucki. Ggp with advanced reasoning and board knowledge discovery. *CoRR*, abs/1401.5813, 2014.

[55] T. Mahlmann, J. Togelius, and G.N. Yannakakis. Modelling and evaluation of complex scenarios with the strategy game description language. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 174–181, Aug 2011.

[56] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical report, 2008.

[57] Stephan Schiffel and Michael Thielscher. Representing and Reasoning About the Rules of General Games With Imperfect Information. *Journal of Artificial Intelligence Research*, 49:171–206, 2014.

[58] GraphViz. Finite automaton.
`http://www.graphviz.org/Gallery.php`, 2014.

[59] Evan Cox and Eric Schkufza. Factoring general games using propositional automata. *Workshop on General Game Playing*, 2009.

[60] Hilmar Finnsson. Generalized Monte-Carlo Tree Search Extensions for General Game Playing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[61] Gregory Kuhlmann and Peter Stone. Automatic heuristic construction in a complete general game player. *AAAI*, (July), 2006.

[62] Stephan Schiffel and Michael Thielscher. Automatic construction of a heuristic search function for general game playing. *Department of Computer Science*, 2006.

[63] James Clune. Heuristic evaluation functions for general game playing. *AAAI*, pages 1134–1139, 2007.

[64] Daniel Michulke. Neural networks for high-resolution state evaluation in general game playing. In *Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11)*, pages 31–37, 2011.

[65] Martin Günther. Automatic feature construction for general game playing. Master's thesis, TU-Dresden, 2008.

[66] Gregory John Kuhlmann. *Automated Domain Analysis and Transfer Learning in General Game Playing*. PhD thesis, University of Texas at Austin, 2010.

[67] Mark Richards, Abhishek Gupta, Osman Sarood, and Laxmikant V. Kale. Parallelizing Information Set Generation for Game Tree Search Applications. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '12, pages 116–123, Washington, DC, USA, 2012. IEEE Computer Society.

[68] Jeffrey Richard Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[69] Stefan Krauss and X T Wang. The psychology of the monty hall problem: discovering psychological mechanisms for solving a tenacious brain teaser. *J Exp Psychol Gen*, 132(1):3–22, 2003.

[70] Marilyn Savant. *The power of logical thinking : easy lessons in the art of reasoning– and hard facts about its absence in our lives*. St. Martin's Griffin, New York, 1997.

[71] Albert Morehead. *Hoyle's rules of games : descriptions of indoor games of skill and chance, with advice on skillful play : based on the foundations laid down by Edmond Hoyle, 1672-1769*. Signet, New York, 2001.

[72] Randall Munroe. Game AIs.
`http://xkcd.com/1002/`, 2011.

Appendix A
Specification

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Jakub Č e r n ý

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Playing General Imperfect-Information Games Using Game-Theoretic Algorithms

Guidelines:

The extensive form is a well-established mathematical model for representing finite sequential games. Designing domain-independent algorithms for solving large-scale extensive-form games is a challenging task and several algorithms have been introduced over the last years. These include exact algorithms based on the sequence-form representation and mathematical programming, or approximative algorithms based on no-regret learning. On the other hand, general game-playing (GGP) focuses on comparing domain-independent algorithms on simple games described in a universal Game Description Language (GDL). Moreover, the new variant of GDL, GDL-II, is able to describe the rules of a game with imperfect information. The task of the student is to combine one of the state-of-the-art algorithms for solving imperfect-information games with the GGP domain and design a game-playing algorithm. Then, compare the new game-playing algorithm with the existing GGP players based on simple heuristic game-tree search.

Bibliography/Sources:

[1] Branislav Bosansky, and Christopher Kiekintveld, and Viliam Lisy, and Jiri Cermak, and Michal Pechoucek: Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013). 2013

[2] Michael Thielscher. A General Game Description Language for Incomplete Information Games. AAAI. Vol. 10, 2010.

[3] Yoav Shoham, and Kevin Leyton-Brown: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2009.

Bachelor Project Supervisor: Mgr. Branislav Bošanský

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jakub Č e r n ý

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Hraní obecných her s neúplnou informací pomocí algoritmů výpočetní teorie her

Pokyny pro vypracování:

Konečné sekvenční hry lze matematicky popsat pomocí tzv. extenzivní formy. Návrh doménově nezávislých algoritmů pro řešení velkých extenzivních her je předmětem dlouhodobého výzkumu. Mezi existující algoritmy patří řada exaktních algoritmů, které využívají tzv. sekvenční formu a metody matematického programování, a také aproximativních algoritmů, které využívají opakovaný průchod stromem a optimální strategii naleznou pomocí metod učení. Na druhou stranu, problematika obecného hraní her (GGP z angl. general game-playing) je zaměřena na návrh algoritmů pro skutečné hraní jednoduchých her, které jsou popsány pomocí obecného jazyka GDL (z angl. Game Description Language). V nové verzi, GDL-II, umožňuje tento jazyk popsat i hry, ve kterých hráči nemají plnou informaci o stavu hry. Cílem studenta je proto použít některý z exaktních nebo aproximativních algoritmů pro řešení extenzivních her pro vlastní hraní v doméně GGP a porovnat tento přístup s existujícími algoritmy založenými na heuristickém prohledávání herního stromu.

Seznam odborné literatury:

[1] Branislav Bosansky, and Christopher Kiekintveld, and Viliam Lisy, and Jiri Cermak, and Michal Pechoucek: Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013). 2013

[2] Michael Thielscher. A General Game Description Language for Incomplete Information Games. AAAI. Vol. 10, 2010.

[3] Yoav Shoham, and Kevin Leyton-Brown: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2009.

Vedoucí bakalářské práce: Mgr. Branislav Bošanský

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Appendix B
Domain Documentation

This appendix serves as a tutorial into creating a distinct game domain in GT Library. The concept of domain was originally designed to represent two-player zero-sum extensive-form games with imperfect information. However, this framework enables to describe also other types of games, e.g. simultaneous-move games. Every well-formed game is obligated to have following structure:

- Game Info
- Game State
- Game Expander
- Game Actions

In domain, Game State class is an implementation of a game state and its instances form a set of all game states. Naturally, Game Action class is an implementation of an action and its instances form a set of all possible actions. Game Expander is a class, which implements a function selecting from the set of all actions only those which are feasible in a given state. Finally, Game Info class provides a basic info about the game, e.g. a list of participating players or a maximal obtainable utility. All together, the domain creates a mechanism for a successful build of the entire game tree.

The crucial thing is to ensure, that all game states are accurately assigned to one of the information sets. Based on the observations, the information set key should add the game state to the set of appropriate indistinguishable states. A slight error can result in constructing a game tree, which is not the same as it was intended. The framework of GT Library also expects all players to have a perfect recall.

B.1 Game Info

Game Info is a class generally describing the game, defining players, maximal utility possible to obtain and maximal depth of the game. It serves as a static database accessed from any other class in the domain. The rules of the game are fixed and therefore the variables are defined as final.

- `getMaxUtility()` → `Double` Returns a non-negative double or infinity, representing a maximal utility in the game, which is a parameter necessary for the successful run of the solving algorithms, where it serves as an upper bound for tree pruning.
- `getFirstPlayerToMove()` → `Player` Specifies a player who takes an action in the root of the game tree.
- `getOpponent(Player player)` → `Player` Toggles between the players.
- `getInfo()` → `String` Describes the game and its setup - e.g. Goofspiel with fixed nature sequence.
- `getMaxDepth()` → `Integer` Returns a positive integer or infinity. Maximal depth is an optional parameter, used to define an upper bound to terminate the game - e.g. 3 rounds.

- `getAllPlayers()` → `Player[ ]` Returns all players, including also the Nature player (if present).

B.2 Game State

Game State class is the main and the most important class of the domain. It identifies the states and assigns them to one of the information sets, handles the currently playing player and if recognized as terminal, provides the utilities to determine the winner. Following the standard definition of extensive-form games, each game state belongs to one of the players, or it can be defined as a chance node to simulate a move by nature.

- `getProbabilityOfNatureFor(Action action)` → `Double` Returns a probability of performing the action. This value is assigned to all actions in every chance node during the construction of the game tree.
- `getISKeyForPlayerToMove()` → `Pair<Integer, Sequence>` Creates an information set key mapping the state to one of the information sets. ISKey consists of a pair observation history - actions history.
- `getPlayerToMove()` → `Player` Determines which player is on the move in this state.
- `getUtilities()` → `Double[ ]` Provides utilities for each player at the current state. If this state is not terminal, utilities are zero.
- `isGameEnd()` → `boolean` Checks whether the condition for a terminal state is satisfied - e.g. if the current depth is equal to max depth.
- `isPlayerToMoveNature()` → `boolean` Checks whether the condition for a chance node is satisfied, so if the Nature player is on move - e.g. at the end of each round.
- `hashCode()` → `Integer`
- `equals(Object obj)` → `boolean`
- `copy()` → `GameState`

B.3 Game Expander

Expander is a class responsible for branching the game tree. It defines all legal actions available in a given state, creates their instances and returns them at once.

- `getActions(GameState gameState)` → `List<Action>` Returns the feasible actions in gameState.

B.4 Game Action

Each instance of Action class corresponds to one possible action for a player on the move.

- `perform(GameState gameState)` → `void` Performs the action in gameState.
- `toString()` → `String` Describes the action - e.g. in Rock-Paper-Scissors, one action can represent The rock.
- `hashCode()` → `Integer`
- `equals(Object obj)` → `boolean`

B.5 Implementing new domain

Standard type of game domain describes *two-player zero-sum extensive-form game with imperfect information*. Any game of this kind can be implemented into the domain the following way:

1. step - *Define game info*
   Info about the game should contain all the necessary information needed to define the initial state – number of players, maximal utility, first player to move and if the end of the game depends on number of rounds, it should be defined in `getMaxDepth()` function.
2. step - *Create an expander*
   Expander takes a game state and creates a list of all possible actions in this state. Formulate a way to find feasible actions in every state of the game tree.
3. step - *Describe the actions*
   Every action should be unique. They are described by the information set in which they are executed and some other attributes which distinguish them within the same information set. For example in Tic-Tac-Toe the actions are distingushed by the cell which can be marked. Define these domain-dependent aspects of actions.
4. step - *Summarize all in a game state class*
   Create the `getISKeyForPlayerToMove()`, `hashCode()` and `equals(Object obj)` functions to differentiate one state from others. Implement the whole one-round game cycle. The cycle transforms one state into another, so it should consist of performing the action, changing the IS key and switching between the players. Remember, states could belong also to the Nature player to bring randomness into the game. By convention, `GameState()` constructor returns the root state of the game.

Appendix C
Infix Form GDL to Prefix Form

In this appendix is described the difference between prefix and infix form of GDL. Infix version is more comprehensible than the prefix form, however, prefix GDL is widely used by most GGP servers to store the game descriptions.

C.1 Transformation table

In table C.1 is shown the mapping from infix to prefix GDL. The syntax of prefix GDL does not differ from any other prefix notation, how used e.g. in Scheme[1]).

Infix GDL	Prefix GDL
`p(a,Y)`	`(p a ?y)`
`~p(a,Y)`	`(not (p a ?y))`
`p(a,Y) & p(Y,c)`	`(and (p a ?y) (p ?y c))`
`q(Y) :- p(a,Y) & p(Y,c)`	`(<= (q ?y) (and (p a ?y) (p ?y c)))`
`q(Y) :- p(a,Y) & p(Y,c)`	`(<= (q ?y) (p a ?y) (p ?y c))`

Table C.1. Mapping from infix GDL to its prefix variant [9].

C.2 An example

An example is given on a GDL form of Prisoner's Dilemma problem. This game is a canonical example with applications in social sciences, economics and politics; and was explained in detail in section 2.3.2 of Game Theory chapter.

The prefix form of this game looks this way:

```
 1 role(Kevin)
 2 role(Fat)
 3 init(p)
 4 legal(Kevin,cooperate)
 5 legal(Kevin,defect)
 6 legal(Fat,cooperate)
 7 legal(Fat,defect)
 8 next(t) :- p
 9 goal(Kevin,100) :- does(Kevin,defect) & does(Fat,cooperate)
10 goal(Kevin,66) :- does(Kevin,cooperate) & does(Fat,cooperate)
11 goal(Kevin,33) :- does(Kevin,defect) & does(Fat,defect)
12 goal(Kevin,0) :- does(Kevin,cooperate) & does(Fat,defect)
13 goal(Fat,0) :- does(Kevin,defect) & does(Fat,cooperate)
14 goal(Fat,33) :- does(Kevin,defect) & does(Fat,defect)
15 goal(Fat,66) :- does(Kevin,cooperate) & does(Fat,cooperate)
16 goal(Fat,100) :- does(Kevin,cooperate) & does(Fat,defect)
17 terminal :- true(t)
```

[1]) `http://www.r6rs.org/`

The infix form of Prisoner's Dilemma is described as follows. Appropriate lines of infix and prefix variant correspond.

```
 1 ( role Kevin )
 2 ( role Fat )
 3 ( init p )
 4 ( legal Kevin cooperate )
 5 ( legal Kevin defect )
 6 ( legal Fat cooperate )
 7 ( legal Fat defect )
 8 ( <= (next t)  p )
 9 ( <= ( goal Kevin 100 ) ( does Kevin defect )
        ( does Fat cooperate ))
10 ( <= ( goal Kevin 66 ) ( does Kevin cooperate )
        ( does Fat cooperate ))
11 ( <= ( goal Kevin 33 ) ( does Kevin defect )
        ( does Fat defect ))
12 ( <= ( goal Kevin 0 ) ( does Kevin cooperate )
        ( does Fat defect ))
13 ( <= ( goal Fat 0 ) ( does Kevin defect )
        ( does Fat cooperate ))
14 ( <= ( goal Fat 33 ) ( does Kevin defect )
        ( does Fat defect ))
15 ( <= ( goal Fat 66 ) ( does Kevin cooperate )
        ( does Fat cooperate ))
16 ( <= ( goal Fat 100 ) ( does Kevin cooperate )
        ( does Fat defect ))
17 ( <= terminal ( true t ) )
```

Appendix D
Abbreviations and Symbols

In this appendix are stated the complete lists of all abbreviations and symbols used in the text.

D.1 Abbreviations

AAAI	Association for the Advancement of Artificial Intelligence.
AGI	Artificial general intelligence.
AI	Artificial intelligence.
CFR	Counterfactual regret.
EXP3	Exploration-exploitation with exponential weights.
GCL	Game communication language.
GDL	Game description language.
GGP	General Game Playing.
GGPGPGPU	GGP through general-purpose computing on graphics processing units.
IJCAI	International Joint Conference on Artificial Intelligence.
IS	Information set.
ISS	Information set search.
IFF	If and only if.
MC	Monte Carlo methods.
MCTS	Monte Carlo tree search.
MIT	Massachusetts Institute of Technology.
NE	Nash equilibrium.
SM-MCTS	Simultaneous-move Monte Carlo tree search.
SPE	Subgame-perfect equilibrium.
UCB1	Upper confidence bound 1.
UCT	Upper confidence bound 1 applied to trees.
WLOG	Without loss of generality.

D.2 Symbols

$\alpha - \beta$	A pruning heuristic used in backward induction.
β	A behavioral strategy.
Δ	Any probability distribution.
π	A pure strategy.
Π	A set of pure strategies.
σ	A mixed strategy; or a sequence in sequence-form game.
Σ	A set of mixed strategies; or a set of sequences in sequence-form game.
$\mathbb{N}$	A set of natural numbers.
$\mathbb{R}$	A set of real numbers.
TEX	Typographical system Tex.

Appendix E
CD Content

At the CD are located several files which require a third party software to be executed. Specifically, this includes the source code of Shodan, the text of this thesis and numerous figures and algorithms schemata. All necessary programs can be downloaded for free from the websites in footnotes:

- **Java** – in version at least 1.7[1])
- **TEX** – both LaTeX and plainTeX[2])
- **GraphViz** – in version at least 2.36[3])

Although an early version of Shodan ran at Java 1.6, with the new update the GGP Base necessitates several external libraries compatible only with Java 1.7 and greater. This document was typeset in PlainTeX using $\mathcal{CS}$plain[4]) for a few Czech characters and the CTUstyle[5]) template by Petr Olsak, to whom I'm really grateful for that. The images were mostly programmed manually or exported using Java to the DOT language[6]), and later processed with GraphViz to PDF. Other figures were created in CorelDRAW Graphics Suite[7]).

For connecting the player to the Dresden GGP[8]) server or to the Tiltyard[9]) server is demanded a public IP address. Without it the server is not able to ping the player and schedule matches with his participation.

The enclosed CD contains following files and directories:

- **cernyj49.pdf** – the text of this thesis
- **doc** – directory with the TeXsource files of this document
  - **figs** – contains all figures
  - **graphs** – contains graph structures in DOT language
  - **specification** – contains the specification of this thesis
  - **text** – contains text source files
- **gdlgames** – directory with several games in GDL-II
- **players** – directory with .jar files of Shodan and TIIGR
- **source** – directory with implementation in Java
  - **GT library** – contains the main classes of Shodan
  - **GGP Base** – contains the supporting classes

[1]) http://www.oracle.com/technetwork/java/javase/downloads/index.html
[2]) https://www.tug.org/texlive/
[3]) http://www.graphviz.org/
[4]) http://petr.olsak.net/csplain.html
[5]) http://petr.olsak.net/ctustyle.html
[6]) http://www.graphviz.org/pdf/dotguide.pdf
[7]) http://www.coreldraw.com/us/product/graphic-design-software/
[8]) http://ggpserver.general-game-playing.de/
[9]) http://tiltyard.ggp.org/