General Game Playing

Adaptable, Multi-Game Playing Programs

Ricardo Filipe Amendoeira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Nuno Cavaco Gomes Horta
Prof. João Paulo Carvalho

December 2015

Abstract

Your abstract goes here.

Keywords: my keywords

Resumo

O teu resumo aqui...

Palavras-Chave: as tuas palavras chave

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Game playing has always been a fundamental part of Artificial Intelligence (AI) research, as it can be used to test strategy in a straightforward way. Different games can be created to test specific features or properties that are of research interest, such as opponent modeling.

Throughout the history of AI research there was always a big focus on the ability of playing specific games, like chess, well. This focus led to systems like Deep Blue (the computer system that defeated Garry Kasparov at chess in the 90's) that, while very advanced on the games they are designed for, delegate all the interesting analysis to the system designers. These systems are also completely useless for all games other than the one they were designed for (even if the difference is very small).

This over-specialization limits the usefulness of these systems. It is then of our interest to also be able to design more general systems, which could be closer to one of the most powerful features of human intelligence, adaptability:

> *"A human being should be able to (...) design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a problem, pitch manure, program a computer, fight efficiently, die gallantly.*
> *Specialization is for insects."*
>
> - Robert A. Heinlein, *Time Enough for Love*

General Game Playing (GGP) aims to develop general game playing systems, systems that can play any game when given the rules, acting as a stepping stone for General Intelligence research. Artificial General Intelligence has been a topic for science-fiction stories and, if possible, can be the biggest revolution in human history, allowing for an unprecedented ability of problem-solving.p

Chapter 2

Background

2.1 Early attempts at multi-game playing

2.2 The basics of a GGP match

A GGP match consists of 3 major components:

- Game Description: The game rules, in Game Description Language (GDL).
- Game Manager: This system acts as a referee and manages communication with the players and other systems like graphics for the spectators. *State Data* is usually part of the Game Manager.
- Players: Players are the most interesting component of a GGP game, they need.

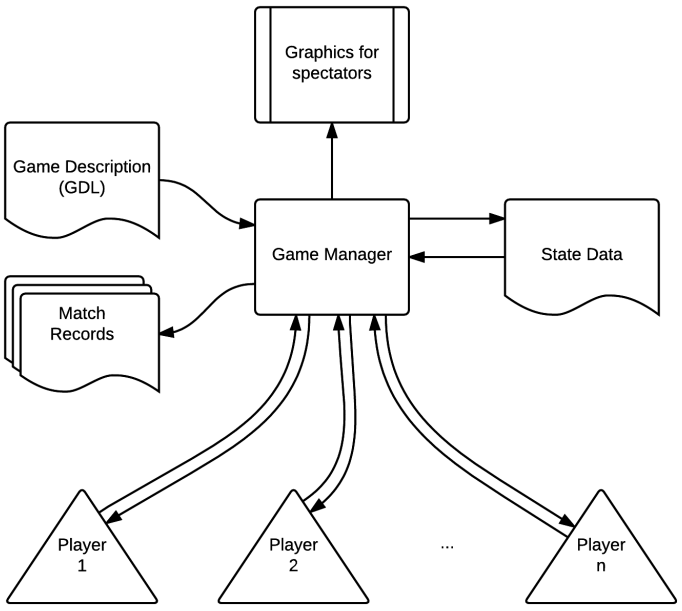

Figure 2.1: Match Components

At the beginning of a match, the Game Manager sends all Players a match identifier, the game description, the role of each player and the time limits (for preparation (*startclock*) and for each round (*playclock*)).

The match commences, after all players respond, with the Game Manager requesting a move from all players. Each round ends when all players send their moves or the time limit runs out (a random legal move is chosen for players that don't respond in time), after which the Game Manager will send each player another request along with all moves taken in the previous round.

2.2.1 Game Description Language

The GDL is the standard way of describing games in the GGP community. GGP players interpret the language using something called a *Reasoner*. Choosing a good way of interpreting the game rules is one of the keys to performance and so many players develop their own custom *reasoners*.

It can describe any finite deterministic move-based strategy game with an arbitrary number of players (most board games). GDL-II is an extension that has been made to allow for probabilistic games and incomplete information, like most card games.

Both GDL and GDL-II are variants of Datalog (query and rule language similar to prolog) and use first order logic. Since GDL is a very conceptual description of the rules their interpretation is very computationally expensive. Choosing a good way of doing this interpretation (components that do this are called reasoners) is therefore very important to player performance, even in the recent years.

2.2.2 Game Manager

The purpose of the Game Manager is to be a single source of truth about what's happening in a match, and verify all moves taken by players. It must be able to interpret GDL, to verify these moves

Players communicate their moves to the Game Manager (via HTTP), who checks the validity of the moves. A random legal move is chosen if a player chooses an illegal move or doesn't respond in time.

It should also provide a way of archiving the match history (all game states and moves taken) and other useful features like an interface for spectating.

2.2.3 Game Player

Game Players are systems that can interpret a GDL game description, communicate with the Game Manager and devise strategies, to maximize their result in a certain game.

Game Players are, of course, the most interesting part of any GGP match. Their aim is to be as general as possible while also having reasonably good performance in any game, which is a surprisingly difficult feat. Suffice it to say, sophisticated AI techniques like heuristics are very hard to successfully apply in a general, domain independent, way. So far, The most relevant techniques are discussed in chapter 3.

Chapter 3

State of the Art

3.1 Competition

The General Game Playing annual competition has been, since its creation in 2005 in Stanford University, the way to know which methods are the state of the art. While in the early years of the competition there was a bigger focus on intelligent heuristics, Monte Carlo Tree Search (MCTS) has dominated the competition ever since, since it's domain independent (general), inherently parallel and has shown better performance in most of the tested games. One of the most important features of MCTS is that the process of building the tree can be paused when necessary (for example when a turn ends) and continued at a later time. This allows a player to continue the simulations throughout the whole game, without restarting after each turn. The variant of MCTS commonly used in GGP is called Upper Confidence Bounds Applied for Trees (UCT), which provides a simple method to balance tree exploration (search new branches) and exploitation (search deeper in the known branches).

3.2 Techniques

3.2.1 Monte Carlo Tree Search

Introduced to the GGP competition by CADIA player in 2007, It's currently the most used and successful method in GGP. Starting from the current state, the algorithm traverses the tree until the move timer ends, doing as many iterations as possible.

Each iteration has four steps: selection, expansion, simulation and back propagation:

1. Selection: Some technique is used to select which already traversed node to start from for a search. The most common technique is Upper Confidence Bounds applied for Trees (UCT), which includes a constant that can be tweaked to favor more or less exploration of non visited branches. The UCT algorithm is described in 3.1:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln |N(s)|}{N(s,a)}} \right\} \tag{3.1}$$

Where a^* is the selected node, $a \in A(s)$ means an action that contained in the set of possible actions in the current state s, $Q(s,a)$ is an assessment of performing a in state s, C is the exploration

ratio constant, $N(s)$ is the number of previous visits to state s and $N(s, a)$ is the number of times a has been sampled in state s.

2. Expansion: Adding a node with the first unvisited state yet to the tree, meaning a state that wasn't already in the tree.

3. Simulation: Perform a random simulation until a terminal game state is reached.

4. Back-Propagation: The scores obtained by all players at the end of the simulation are back-propagated to all nodes traveled in the selection and expansion stages.

The success of MCTS can be mostly attributed to it not requiring any game-specific knowledge, although this can become a problem if other techniques like heuristics become advanced enough at learning important features of games, as heuristic search can be much faster than simulation. MCTS also has the advantage of parallelizing well. The biggest problems for MCTS are games that can have infinite moves without ending and tree size.

There have been several suggested improvements to the basic MCTS, although most aren't very thoroughly tested yet. One of the most interesting ones is Simulation Heuristics, proposed by MINI-Player, which aims to add some sort of learning to the standard MCTS algorithm. The heuristics proposed are very light-weight and are the following:

- Random: The standard MCTS
- History Heuristic: Tries to identify globally good actions (generally good regardless of state)
- Mobility: Favors actions that lead to states with more move options relatively to other players.
- Approximate Goal Evaluation: Tries to calculate the degree of satisfaction of a GDL goal rule.
- Exploration: Measures the difference between states as a way to do a diverse exploration.
- Statistical Symbol Counting: Before the start clock simulations are done to calculate the correlation between game score and certain game symbols (moves, pieces, board locations, etc). Symbols that do not change much are then ignored to allow more computation to be made on the more relevant ones.

3.3 AI-Based Techniques

Although currently not competitive with MCTS, there are several different approaches more similar to classical AI. Some of these are attempts at multi-game playing that predate GGP. These are techniques that try to learn or identify features of the game. One of the biggest drawbacks of this type of technique is that no general heuristic exists, meaning heuristics have to be discovered at run time, which is a very complex problem.

3.3.1 Hoyle

A system developed in the 90's, using a training scheme called lesson and practice, where lessons are games played against an expert and practice is self-playing. Predates GDP and was developed for 2-player games. The system used a set of game independent Advisers, each specialized in a game aspect such as position. These Advisers could recommend moves that could then be chosen by higher tier advisers. There were 3-tiers, depending on specialization:

1st. These Advisers specialized in immediate consequences: they performed very shallow searches to avoid things like instant loss moves. These decisions were final.

2nd. Advisers in this tier chose moves according to certain goals. These decisions were also final.

3rd. Advisers in the last tier differed from the first tiers in an important way: The decision of each Adviser wasn't final, the final decision was decided by a process similar to taking a vote between the Advisers in this 3rd tier. Advisers votes were weighted in accordance to the lesson and practice results: Advisers that were more often correct during the training stage received bigger weights.

This process of weighting the Advisers was crucial to the performance of the system and could even be worse than a random player if done incorrectly. If none of the 3rd tier advisers were even remotely related to the game being played the results would also be disappointing. Having a varied pool of Advisers was for this reason vital but they were never, by definition, general enough. Hoyle was tested in 18 two-player board games, its potential in complex games was never verified.

3.3.2 FluxPlayer

The winner of the second GGP competition, in 2006, this player used fuzzy logic to determine how close to terminal a certain state is.

Fuzzy logic is a form of logic that allows multiple different values beyond True or False, and also allows overlap between these values. For example: water can be considered cold, warm or hot, but also warm and hot at the same time, since for some temperatures it is not clear which one is the correct one. Fuzzy logic also allows each of these states to have varying degrees of certainty: the water can be 80% warm and 10% cold, allowing conditions like if water is very warm: add some cold water, where the very keyword is also part of the fuzzy logic implementation.

The system also used a novel heuristic search that could be computed from the specifications of the game.

Appendix A

GDL example: tic tac toe

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Tictactoe - GDL Description
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Roles
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (role xplayer)
  (role oplayer)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Initial State
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (init (cell 1 1 b))
  (init (cell 1 2 b))
  (init (cell 1 3 b))
  (init (cell 2 1 b))
  (init (cell 2 2 b))
  (init (cell 2 3 b))
  (init (cell 3 1 b))
  (init (cell 3 2 b))
  (init (cell 3 3 b))
  (init (control xplayer))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Dynamic Components
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Cell

  (<= (next (cell ?m ?n x))
      (does xplayer (mark ?m ?n))
      (true (cell ?m ?n b)))

  (<= (next (cell ?m ?n o))
      (does oplayer (mark ?m ?n))
      (true (cell ?m ?n b)))

  (<= (next (cell ?m ?n ?w))
      (true (cell ?m ?n ?w))
```

```
      (distinct ?w b))

  (<= (next (cell ?m ?n b))
      (does ?w (mark ?j ?k))
      (true (cell ?m ?n b))
      (or (distinct ?m ?j) (distinct ?n ?k)))

  (<= (next (control xplayer))
      (true (control oplayer)))

  (<= (next (control oplayer))
      (true (control xplayer)))


  (<= (row ?m ?x)
      (true (cell ?m 1 ?x))
      (true (cell ?m 2 ?x))
      (true (cell ?m 3 ?x)))

  (<= (column ?n ?x)
      (true (cell 1 ?n ?x))
      (true (cell 2 ?n ?x))
      (true (cell 3 ?n ?x)))

  (<= (diagonal ?x)
      (true (cell 1 1 ?x))
      (true (cell 2 2 ?x))
      (true (cell 3 3 ?x)))

  (<= (diagonal ?x)
      (true (cell 1 3 ?x))
      (true (cell 2 2 ?x))
      (true (cell 3 1 ?x)))


  (<= (line ?x) (row ?m ?x))
  (<= (line ?x) (column ?m ?x))
  (<= (line ?x) (diagonal ?x))


  (<= open
      (true (cell ?m ?n b)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (<= (legal ?w (mark ?x ?y))
      (true (cell ?x ?y b))
      (true (control ?w)))

  (<= (legal xplayer noop)
      (true (control oplayer)))

  (<= (legal oplayer noop)
      (true (control xplayer)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (<= (goal xplayer 100)
      (line x))
```

```

  (<= (goal xplayer 50)
      (not (line x))
      (not (line o))
      (not open))

  (<= (goal xplayer 0)
      (line o))

  (<= (goal oplayer 100)
      (line o))

  (<= (goal oplayer 50)
      (not (line x))
      (not (line o))
      (not open))

  (<= (goal oplayer 0)
      (line x))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (<= terminal
      (line x))

  (<= terminal
      (line o))

  (<= terminal
      (not open))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```