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

1.1 Motivation

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.

1.2 Multi-game playing

1.2.1 Early attempts at multi-game playing

Write some more here...

1.2.1.1 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 to be useful for any game. Hoyle was tested in 18 two-player board games, its potential in complex games was never verified.

1.2.2 General Game Playing

Fill this in

1.3 Objectives

Main Objectives:

- Research and compare current techniques and solutions for GGP players
- Improve or develop new techniques for game playing
- Use the improvements to make a new GGP player
- Benchmark the player against other existing GGP players

Secondary Objectives:

- Publish the results
- Enter the annual GGP competition
- Enter other relevant competitions

1.4 Planning

Fill this in

Chapter 2

Background

2.1 The basics of General Game Playing

General Game Playing is a project of the Stanford Logic Group of Stanford University, California, which aims to create a platform for GGP. Since 2005, there have been annual GGP competitions at the AAAI Conference.

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.

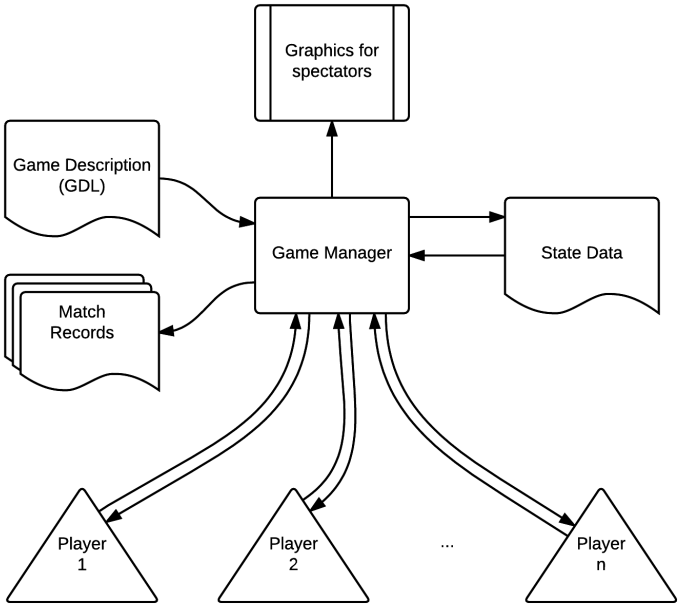

Figure 2.1: Match Components

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.

2.1.1 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.

An example of tic tac toe described in GDL with some syntax explanation can be seen in A

2.1.2 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.

2.1.3 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. 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.

2.2 Markov Decision Problem

The Markov Decision Problem (MDP) provides a way of mathematically modeling decision making and can help to formalize the task of GGP Players. In short, a Markov Decision Process is composed of 5 components:

- States - S : The set of all possible states of the problem.
- Actions - $A(s)$: The set of possible actions in state s.

- Model - $T(s, a, s') \sim Pr(s'|s, a)$: The laws of the universe in which the problem is contained. In other words, what is the outcome (state s') of taking an action a in state s. The model can be deterministic or stochastic ($Pr(s'|s, a)$), where multiple outcomes are possible.
- Reward - $R(s, a, s')$: What is the reward of taking action a in state s and transitioning to state s'.
- Discount Factor - γ : Reduces the importance of distant rewards. It's important in the common case of a stochastic problem, since distant rewards may become unreachable due to external influence.

A policy, π, defines what action $\pi(s)$ the agent will take when in state s. Policies are solutions to MDP's.

In the domain of General Game Playing each different match has it's own MDP, since the Model is defined not only by the game rules but also by the behavior of other players. Rewards and discount factors are also not solely dependent on the game rules (is it always a good move to take a Knight in chess?). Most GGP matches have, therefore, unknown rewards and probabilities from the point of view of the player, making them *reinforcement learning* problems. In these problems it is useful to define the function $Q(s, a)$, given by 2.1, and is an assessment of the quality of taking action a in state s.

$$Q(s, a) = \sum_{s'} Pr(s'|s, a)(R(s, a, s') + \gamma V(s')) \tag{2.1}$$

$V(s')$ is the value of being in state s', the discounted sum of the expected rewards to be earned (on average) by following the policy from state s' on-wards. There are several ways to implement this calculation.

If all the probabilities and rewards are known the policy can be solved recursively according to 2.2 and 2.3.

$$\pi(s) \equiv argmax_a \left\{ \sum_{s'} Pr(s'|s, a)(R(s, a, s') + \gamma V(s')) \right\} \tag{2.2}$$

$$V(s) \equiv \sum_{s'} Pr(s'|s, \pi(s))(R(s, \pi(s), s') + \gamma V(s')) \tag{2.3}$$

2.3 Basics of a GDL player

A simplified example of a GGP Player architecture can be seen in figure 2.2. As seen in the figure, the player must have some form of these components:

- HTTP Server: Interfaces with the Game Master
- GDL Interpreter: Analyses the game rules, in some cases changing their representation into a more efficient data structure
- Game State: Holds any relevant information about the game, like the current state or past moves by opponents (to maybe model their strategy)

- Game Planner: Attempts to choose the best action for the current state of the game. If it uses simulations as part of its operation (very common) it may interface with the GDL Interpreter to discover what actions are available in each of the simulated states.

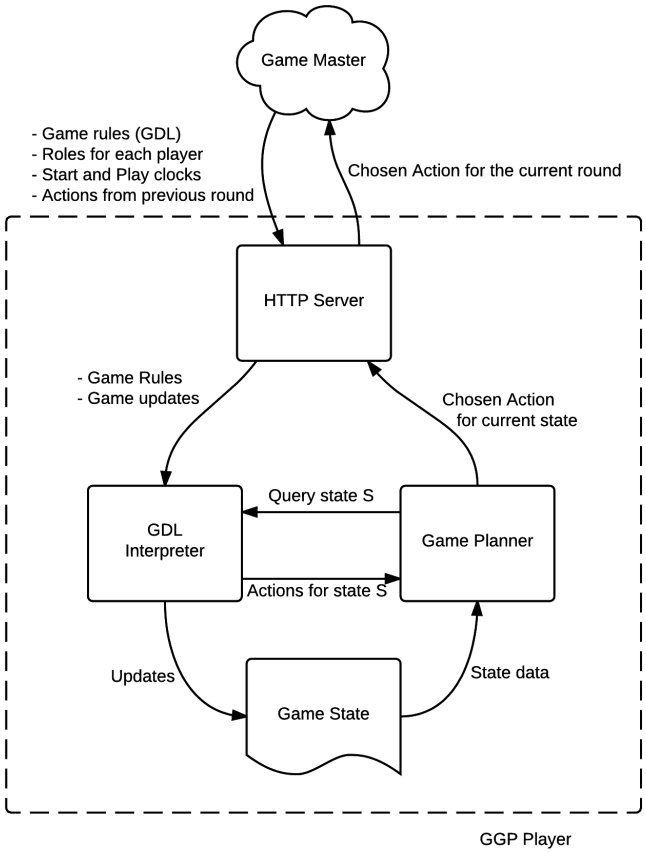

Figure 2.2: Simplified example of a GGP Player architecture

The Game Planner is the biggest factor for system performance, as it is both the component that chooses the strategy and the one that requires the most computing power. The GDL Interpreter can also affect performance in a smaller but relevant way by using a significant portion of the computational power of the system. It becomes more important as more simulations are needed.

The currently most relevant solutions for both Game Planning and GDL interpretation are discussed in chapter 3.

2.3.1 Player performance measurement

2.4 GDL Interpreter Techniques

2.4.1 Propositional networks

Fill this in

2.5 Game Planner Techniques

2.5.1 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 2.4:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln |N(s)|}{N(s,a)}} \right\} \tag{2.4}$$

   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. This is how the rewards of non-terminal states is computed.

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.

2.5.2 Heuristics

Heuristics are methods that simplify or accelerate problem solving by not guaranteeing optimal solutions but satisfying approximations. Heuristics can be very useful whenever finding an optimal solution is impossible or impractical.

In the case of GGP these techniques try to learn or identify features of the game, so the player can more efficiently evaluate the quality of the available actions. One of the biggest drawbacks of this type of technique is that heuristics become less useful the more general they become, meaning that, ideally, specialized heuristics should be created at run time, which is a very complex problem.

Although currently not competitive with Monte Carlo Tree Simulation (MCTS) on their own, heuristic techniques are used to try to improve MCTS.

There have been, however, some Heuristic based techniques to GGP that don't use MCTS.

2.5.3 Fill this in

Chapter 3

State of the Art

3.1 Competition as a benchmark

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).

There are other, smaller, GGP competitions that usually have the same results in terms of what techniques tend to perform best.

3.1.1 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.

3.1.2 Cadia Player

Winner in 2007, started the reign of MCTS players in GGP.

3.1.3 Sancho

Champion in 2014, info here, need to research it: http://sanchoggp.blogspot.pt/2014/05/what-is-sancho.html

MCTS player with a few notable changes: - propositional-network-based state machine (based on Petri Networks) - general graph instead of MCTS tree - some heuristics determined at game setup time

3.1.4 Galvanize

This years champion, MCTS based, research it: https://bitbucket.org/rxe/galvanise_v2

3.1.5 comparison

research the top players from here: http://www.ggp.org/view/tiltyard/players/ and here: https://docs.google.com/spreadsheets/d/12SWEXYAmCCGCm5jI-e0t1HGlMFXXeKSgyuU1boTioJ4/edit#gid=0&vpid=A1 and make a comparison table. If possible, run some of them and do my own comparisons.

Chapter 4

Preliminary Results

Bibliography

[1] J. Mandziuk, Y. Ong, and K. Waledzik, “Multi-game playing - A challenge for computational intelligence,” *Computational Intelligence . . .*, pp. 17–24, 2013.

[2] M. Swiechowski, “Specialized vs . Multi-Game Approaches to AI in Games,” pp. 1–12, 2014.

[3] M. Swiechowski, H. Park, J. Mandziuk, and K.-j. Kim, “Recent Advances in General Game Playing,” *The Scientific World Journal*, vol. 2015, 2015.

[4] J. Mandziuk and M. Swiechowski, “Generic Heuristic Approach to General Game Playing,” *Sofsem 2012, Lncs 7147*, pp. 649–660, 2012.

[5] H. Finnsson, *CADIA-Player: A General Game Playing Agent*. M.sc. thesis, Reykjavík University - School of Computer Science, 2007.

[6] Y. Björnsson, “Learning Rules of Simplified Boardgames by Observing,”

[7] M. Genesereth, N. Love, and B. Pell, “General game playing: Overview of the AAAI competition,” *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.

Appendix A

GDL example: tic tac toe

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

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Roles - Each player is assigned one of these roles at the beginning of the match
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (role xplayer)
  (role oplayer)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Initial State - With the init keyword one can define initial states  of the game.
;; the control keyword defines which player or players have the first move
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (init (cell 1 1 b)) ;; Initialization of the cells with the value '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)) ;; xplayer has the first move

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Dynamic Components - Components that change throughout the game, like the cells
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Cell

;; variables are written with '?' in front of them

  (<= (next (cell ?m ?n x)) ;; the cell with coordinates m,n changes to 'x' if...
      (does xplayer (mark ?m ?n)) ;; xplayer chooses to mark it...
      (true (cell ?m ?n b))) ;; and it's still in state 'b' (blank)

  (<= (next (cell ?m ?n o)) ;; same thing for oplayer
      (does oplayer (mark ?m ?n))
      (true (cell ?m ?n b)))
```

```
(<= (next (cell ?m ?n ?w)) ;; the cell changes to state ?w if...
    (true (cell ?m ?n ?w)) ;; it's already in state ?w
    (distinct ?w b)) ;; and ?w isn't blank

(<= (next (cell ?m ?n b)) ;; the cell ?m ?n changes to 'b' if...
    (does ?w (mark ?j ?k)) ;; player ?w marks ?j ?k...
    (true (cell ?m ?n b)) ;; the cell is already state 'b' or...
    (or (distinct ?m ?j) (distinct ?n ?k))) ;; ?w marked another cell (?m,?n != ?j,?k)

(<= (next (control xplayer)) ;; if oplayer is in control, xplayer takes control
    (true (control oplayer)))

(<= (next (control oplayer)) ;; and vice-versa
    (true (control xplayer)))


(<= (row ?m ?x) ;; a row means 3 cells in the same line with the same state ?x
    (true (cell ?m 1 ?x))
    (true (cell ?m 2 ?x))
    (true (cell ?m 3 ?x)))

(<= (column ?n ?x) ;; a column means 3 cells in the same column with the same state ?x
    (true (cell 1 ?n ?x))
    (true (cell 2 ?n ?x))
    (true (cell 3 ?n ?x)))

(<= (diagonal ?x) ;; same idea for each diagonal
    (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)) ;; a line of ?x exists if there is a row of ?x
(<= (line ?x) (column ?m ?x)) ;; ...or a column of ?x
(<= (line ?x) (diagonal ?x)) ;; ...or a diagonal of ?x


(<= open ;; the game is open if there's at least one cell with the state 'b'
    (true (cell ?m ?n b)))

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

(<= (legal ?w (mark ?x ?y)) ;; player ?w can mark ?x ?y if...
    (true (cell ?x ?y b)) ;; the cell is blank
    (true (control ?w))) ;; player ?w is in control

(<= (legal xplayer noop) ;; player xplayer can do a 'noop' move (do nothing) if...
    (true (control oplayer))) ;; the oplayer is in control

(<= (legal oplayer noop) ;; and vice-versa
    (true (control xplayer)))

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

```
  (<= (goal xplayer 100) ;; xplayes gets 100 points if there is a line of 'x'
      (line x))

  ;; xplayer gets 50 points if there are no open cells and no lines of 'x' or 'o'
  (<= (goal xplayer 50)
      (not (line x))
      (not (line o))
      (not open))

  (<= (goal xplayer 0) ;; xplayer gets 0 points if oplayes gets a line
      (line o))

  (<= (goal oplayer 100) ;; same thing for oplayer
      (line o))

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

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

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

  (<= terminal ;; the game ends if there is a line of 'x'
      (line x))

  (<= terminal ;; ...or a line of 'o'
      (line o))

  (<= terminal ;; ...or if the game is no longer open
      (not open))

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

Listing A.1: tic tac toe described in GDL