Computer Game Playing
Deep Blue Beats World Champion (1997)
"Game Over" —Checkers Solved in 2007
Computer Go
General Game Playing

General Game Players are systems
- able to accept a formal description of arbitrary games
- able to use such descriptions to play the games effectively

Cognitive Information Processing Technologies for GGP systems:
- Knowledge representation
- Reasoning
- Learning
- Rational behaviour

Unlike specialised game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.
Tic-Tac-Toe

\[
\begin{array}{ccc}
X & & X \\
\hline
& O & \\
\hline
X & & \\
\end{array}
\]
Bidding Tic-Tac-Toe

Coins:

<table>
<thead>
<tr>
<th>X</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 *</td>
<td>3</td>
</tr>
</tbody>
</table>
Kriegspiel
Single-Player Games
Monopoly
Poker

You: $1794
bet: $204

player 1: $794
bet: $204

player 2: $998

pot: $12

min bet: $204

fold
call $4
raise $4

player 5: $998
player 4: $994
bet: $4
player 3: $994
bet: $4
International Activities

[www.general-game-playing.de](http://www.general-game-playing.de)

- Games
- Game Manager
- Reference Players
- Development Tools
- Literature

Annual World Cup
- First: Pittsburgh 2005
- Most recent: Barcelona 2011
- Next: Toronto 2012

German Open, Berlin 2011
General Game Playing and AI

Why games?

- Many social, biological, political, and economic processes can be formalised as (multi-agent) games.
- General game-players are rational agents that can adapt to radically different environments without human intervention.

Ordinary Systems

Client \rightarrow Application-Specific System \rightarrow Environment

General Systems

Client \rightarrow General System \rightarrow Environment
How it Works

Game description
Time to think: 1,800 sec
Time per move: 45 sec
Your role

Game Master

Player_1
Player_2
⋯
Player_n
How it Works

Game Master

Player_1 \rightarrow \text{Play} \rightarrow \text{Player}_2 \rightarrow \cdots \rightarrow \text{Player}_n
How it Works

Game Master

Player \(_1\)  Player \(_2\) ... \(\text{Player}_n\)

Individual moves
How it Works

Game Master

Player\textsubscript{1} \rightarrow \text{Individual information about state/moves} \rightarrow \text{Player\textsubscript{2}} \rightarrow \cdots \rightarrow \text{Player\textsubscript{n}}
How it Works

Game Master

Player_1
Player_2
\cdots
Player_n

End of game
Knowledge Representation for GGP: Describing Games
Games as State Machines
Initial State, Terminal States, Simultaneous Moves
Game Model (Perfect-Information Games)

An $n$-player game with perfect information is a structure with components:

$\{r_1, \ldots, r_n\}$ – players

$S$ – set of states

$A_1, \ldots, A_n$ – $n$ sets of actions, one for each player

$l_1, \ldots, l_n$ – where $l_i \subseteq A_i \times S$, the legality relations

$u : S \times A_1 \times \cdots \times A_n \rightarrow S$ – update function

$s_1 \in S$ – initial game state

$t \subseteq S$ – the terminal states

$g_1, \ldots, g_n$ – where $g_i \subseteq S \times \mathbb{N}$, the goal relations
Encoding Alternatives

State Machines. Astronomically large state spaces, e.g. \( \sim 5000 \) states in Tic-Tac-Toe, \( \sim 10^{44} \) states in Chess.

Lists and Tables. Still the same size. Just switching to database states does not decrease the size of direct representation.

Programs. One possibility is to write a program to generate legal moves and successor states and to evaluate goals and termination. However, which language? Java, C? What if a player wants to reason about the structure of a game in general? This is difficult if the game is encoded in procedural form.

Logic. There are existing interpreters / compilers. Logic is easier to use for analysis than procedural encodings.
Formal Game Descriptions

Whatever form is used, the description must give all information necessary to determine legality of moves, state transition, termination, and goals.

Nothing is assumed except for logic.

- No arithmetics
- No physics
- No common sense

(To emphasise this, game descriptions can be written in terms of nonsense symbols.)
Game Description Language

In the Game Description Language (GDL), a game is a logic program. GDL uses the constants 0, 1, ..., 100 and the following predicates as keywords.

- role(r) means that r is a role (i.e. a player) in the game
- init(f) means that f is true in the initial position (state)
- true(f) means that f is true in the current state
- does(r,a) means that role r does action a in the current state
- next(f) means that f is true in the next state
- legal(r,a) means that it is legal for r to play a in the current state
- goal(r,v) means that r gets goal value v in the current state
- terminal means that the current state is a terminal state
- distinct(s,t) means that terms s and t are syntactically different
Tic Tac Toe

cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
control(oplayer)
Bidding Tic Tac Toe

cell(1,1,b)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,b)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,b)
coins(xplayer,3)
coins(oplayer,3)
tiebreaker(xplayer)
bidding_stage
Bidding Tic Tac Toe: Vocabulary

- **Object constants**
  - `xplayer, oplayer` (Players)
  - `x, o, b` (Marks)
  - `bidding_stage` (Fluent)
  - `with_tiebreaker, no_tiebreaker` (Flags)
  - `noop` (Move)

- **Functions**
  - `cell(number, number, mark), control(player), coins(player, number), tiebreaker(player)` (Fluents)
  - `mark(number, number), bid(number, flag)` (Moves)

- **Domain predicates**
Players and Initial State

\texttt{role}(xplayer)
\texttt{role}(oplayer)

\texttt{init}(cell(1,1,b))
\texttt{init}(cell(1,2,b))
\texttt{init}(cell(1,3,b))
\texttt{init}(cell(2,1,b))
\texttt{init}(cell(2,2,b))
\texttt{init}(cell(2,3,b))
\texttt{init}(cell(3,1,b))
\texttt{init}(cell(3,2,b))
\texttt{init}(cell(3,3,b))

\texttt{init}(coins(P,3)) \leq \texttt{role}(P)
\texttt{init}(tiebreaker(xplayer))
\texttt{init}(bidding\_stage)
Move Generator (1)

\[
\text{legal}(P, \text{mark}(M,N)) \iff \\
\quad \text{true}(\text{cell}(M,N,b)) \land \\
\quad \text{true}(\text{control}(P))
\]

\[
\text{legal}(xplayer, \text{noop}) \iff \\
\quad \text{true}(\text{control}(oplayer))
\]

\[
\text{legal}(oplayer, \text{noop}) \iff \\
\quad \text{true}(\text{control}(xplayer))
\]

Conclusions:

\[
\text{legal}(xplayer, \text{noop})
\]

\[
\text{legal}(oplayer, \text{mark}(1,2))
\]

... 

\[
\text{legal}(oplayer, \text{mark}(3,2))
\]
Move Generator (2)

\[ \text{legal}(P, \text{bid}(B, TB)) \leq \]
\[ \text{true}(\text{bidding\_stage}) \land \]
\[ \text{true}(\text{coins}(P, C)) \land \]
\[ \text{less\_or\_eq}(B, C) \land \]
\[ \text{tiebreak\_bid}(P, TB) \]

\[ \text{tiebreak\_bid}(P, \text{with\_tiebreaker}) \leq \]
\[ \text{true}(\text{tiebreaker}(P)) \]

\[ \text{tiebreak\_bid}(P, \text{no\_tiebreaker}) \leq \]
\[ \text{role}(P) \]

Conclusions:

\[ \text{legal}(xplayer, \text{bid}(1, \text{with\_tiebreaker})) \]
\[ \text{legal}(xplayer, \text{bid}(1, \text{no\_tiebreaker})) \]

...\n
\[ \text{legal}(oplayer, \text{bid}(1, \text{no\_tiebreaker})) \]

...
Physics: Example

cell(1,1,x)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,o)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,x)
coins(xplayer,4)
coins(oplayer,2)
tiebreaker(oplayer)
control(oplayer)

mark(1,3) → oplayer

cell(1,1,x) → cell(1,1,x)
cell(1,2,b) → cell(1,2,b)
\textbf{cell(1,3,o)} → cell(1,3,o)
cell(2,1,b) → cell(2,1,b)
cell(2,2,o) → cell(2,2,o)
cell(2,3,b) → cell(2,3,b)
cell(3,1,b) → cell(3,1,b)
cell(3,2,b) → cell(3,2,b)
cell(3,3,x) → cell(3,3,x)
coins(xplayer,4) → coins(xplayer,4)
coins(oplayer,2) → coins(oplayer,2)
tiebreaker(oplayer) → tiebreaker(oplayer)
control(oplayer) → \textbf{bidding\_stage}

X | O | X
---|---|---
O |   | O
X |   | X
Physics (1)

$\text{next} (\text{cell}(M,N,x)) \leq \text{does}(\text{xplayer}, \text{mark}(M,N))$

$\text{next} (\text{cell}(M,N,o)) \leq \text{does}(\text{oplayer}, \text{mark}(M,N))$

$\text{next} (\text{cell}(M,N,W)) \leq \text{true} (\text{cell}(M,N,W)) \land$

$\text{does}(P, \text{mark}(J,K)) \land$

$(\text{distinct}(M,J) \lor \text{distinct}(N,K))$

$\text{next} (\text{coins}(P,C)) \leq \text{true}(\text{control}(Q)) \land \text{true}(\text{coins}(P,C))$

$\text{next} (\text{tiebreaker}(P)) \leq \text{true}(\text{control}(Q)) \land \text{true}(\text{tiebreaker}(P))$

$\text{next} (\text{bidding\_stage}) \leq \text{true}(\text{control}(P))$
Physics (2)

\[
\begin{align*}
\text{next}(\text{coins}(P,C)) & \leq \textbf{true}(\text{bidding}\_\text{stage}) \land \\
& \quad \text{winner}(P) \land \\
& \quad \text{does}(P,\text{bid}(B,TB)) \land \\
& \quad \textbf{true}(\text{coins}(P,C1)) \land \text{add}(C,B,C1)
\end{align*}
\]

\[
\begin{align*}
\text{next}(\text{coins}(\text{xplayer},C)) & \leq \textbf{true}(\text{bidding}\_\text{stage}) \land \\
& \quad \text{winner}(\text{oplayer}) \land \\
& \quad \text{does}(\text{oplayer},\text{bid}(B,TB)) \land \\
& \quad \textbf{true}(\text{coins}(P,C1)) \land \text{add}(C1,B,C)
\end{align*}
\]

\[
\begin{align*}
\text{next}(\text{coins}(\text{oplayer},C)) & \leq \textbf{true}(\text{bidding}\_\text{stage}) \land \\
& \quad \text{winner}(\text{xplayer}) \land \\
& \quad \text{does}(\text{xplayer},\text{bid}(B,TB)) \land \\
& \quad \textbf{true}(\text{coins}(P,C1)) \land \text{add}(C1,B,C)
\end{align*}
\]
Physics (3)

\[
\text{next} (\text{tiebreaker}(P)) \iff \text{true} (\text{bidding\_stage}) \land \text{role}(P) \land \\
\text{winner}(Q) \land \text{distinct}(P, Q) \land \\
\text{does}(Q, \text{bid}(B, \text{with\_tiebreaker}))
\]

\[
\text{next} (\text{tiebreaker}(P)) \iff \text{true} (\text{bidding\_stage}) \land \\
\text{true} (\text{tiebreaker}(P)) \land \\
\text{winner}(Q) \land \\
\text{does}(Q, \text{bid}(B, \text{no\_tiebreaker}))
\]

\[
\text{next} (\text{cell}(M, N, W)) \iff \text{true} (\text{cell}(M, N, W)) \land \text{true} (\text{bidding\_stage})
\]

\[
\text{next} (\text{control}(P)) \iff \text{winner}(P) \land \text{true} (\text{bidding\_stage})
\]
## Termination and Goal Values

**terminal** <=

- \( \text{line}(x) \lor \text{line}(o) \)

**terminal** <=

- \( \neg \text{open} \)

**line**(W) <=

- \( \text{row}(M,W) \lor \text{column}(N,W) \lor \text{diagonal}(W) \)

**open** <=

- \( \text{true}(\text{cell}(M,N,b)) \)

**goal**(xplayer, 100) <= \( \text{line}(x) \)

**goal**(xplayer, 50) <= \( \text{draw} \)

**goal**(xplayer, 0) <= \( \text{line}(o) \)

**goal**(oplayer, 100) <= \( \text{line}(o) \)

**goal**(oplayer, 50) <= \( \text{draw} \)

**goal**(oplayer, 0) <= \( \text{line}(x) \)

**draw** <=

- \( \neg \text{line}(x) \land \neg \text{line}(o) \land \neg \text{open} \)
Supporting Concepts

\[
\text{row}(M,W) \leq \begin{cases} 
\text{true}(\text{cell}(M,1,W)) \land \\
\text{true}(\text{cell}(M,2,W)) \land \\
\text{true}(\text{cell}(M,3,W)) 
\end{cases} \\
\text{column}(N,W) \leq \begin{cases} 
\text{true}(\text{cell}(1,N,W)) \land \\
\text{true}(\text{cell}(2,N,W)) \land \\
\text{true}(\text{cell}(3,N,W)) 
\end{cases} \\
\text{diagonal}(W) \leq \begin{cases} 
\text{true}(\text{cell}(1,1,W)) \land \\
\text{true}(\text{cell}(2,2,W)) \land \\
\text{true}(\text{cell}(3,3,W)) 
\end{cases} \\
\text{diagonal}(W) \leq \begin{cases} 
\text{true}(\text{cell}(1,3,W)) \land \\
\text{true}(\text{cell}(2,2,W)) \land \\
\text{true}(\text{cell}(3,1,W)) 
\end{cases}
\]
More Supporting Concepts

\begin{align*}
\text{winner}(P) & \leq \text{does}(P, \text{bid}(B_1, TB_1)) \land \text{does}(Q, \text{bid}(B_2, TB_2)) \land \\
& \text{distinct}(P, Q) \land \text{greater}(B_1, B_2) \\
\text{winner}(P) & \leq \text{does}(P, \text{bid}(B, \text{with_tiebreaker})) \land \\
& \text{does}(Q, \text{bid}(B, \text{no_tiebreaker})) \\
\text{winner}(P) & \leq \text{does}(P, \text{bid}(B, \text{no_tiebreaker})) \land \\
& \text{does}(Q, \text{bid}(B, \text{no_tiebreaker})) \land \\
& \text{distinct}(P, Q) \land \\
& \text{true}(	ext{tiebreaker}(Q))
\end{align*}

\begin{align*}
\text{succ}(0, 1) & \quad \text{succ}(1, 2) \quad \text{succ}(2, 3) \quad \ldots \\
\text{greater}(X, Y) & \leq \ldots \\
\text{less_or_equal}(X, Y) & \leq \ldots \\
\text{add}(X, Y, Z) & \leq \ldots
\end{align*}
Completeness

Of necessity, game descriptions are logically incomplete in that they do not uniquely specify the moves of the players.

Every game description contains complete definitions for legality, termination, goalhood, and update in terms of the relations true and does.

The upshot is that in every state every player can determine legality, termination, goalhood, and—given a joint move—can update the state.
Syntactic Restrictions

1. **role** as head of clause only appears in facts (i.e., clauses with empty body)
2. **init** only appears as head of clauses and does not depend on any of **true**, **legal**, **does**, **next**, **terminal**, **goal**
3. **true** only appears in bodies of clauses
4. **does** only appears in clause bodies, and none of **legal**, **terminal**, **goal** depends on **does**
5. **next** only appears as head of clauses
Guaranteeing Decidability (1): Safety

A clause is safe if and only if every variable in the clause appears in some positive subgoal in the body.

Safe Rule:

\[ r(X, Y) \leq p(X, Y) \land \neg q(X, Y) \]

Unsafe Rule:

\[ r(X, Z) \leq p(X, Y) \land \neg q(Y, Z) \]

In GDL, all rules are required to be safe.

(Note that this implies all facts to be variable-free.)
The dependency graph for a set of clauses is a directed graph in which

- the nodes are the relations mentioned in the head and bodies of the clauses
- there is an arc from a node $p$ to a node $q$ whenever $p$ occurs in the body of a clause in which $q$ is in the head.

\[
\begin{align*}
  r(X,Y) & \leq p(X,Y) \land q(X,Y) \\
  s(X,Y) & \leq r(X,Y) \\
  s(X,Z) & \leq r(X,Y) \land t(Y,Z) \\
  t(X,Z) & \leq s(X,Y) \land s(Y,X)
\end{align*}
\]

A set of clauses is recursive if its dependency graph contains a cycle. Otherwise, it is non-recursive.
Guaranteeing Decidability (2): Stratification

A set of rules is said to be stratified if there is no recursive cycle in the dependency graph involving a negation.

Stratified Negation:
\[
\begin{align*}
t(X, Y) & \leq q(X, Y) \land \neg r(X, Y) \\
r(X, Z) & \leq p(X, Y) \\
r(X, Z) & \leq r(X, Y) \land r(Y, Z)
\end{align*}
\]

Negation that is not stratified:
\[
\begin{align*}
r(X, Z) & \leq p(X, Y) \\
r(X, Z) & \leq p(X, Y) \land \neg r(Y, Z)
\end{align*}
\]

In GDL, all sets of rules are required to be stratified.
Guaranteeing Decidability (3)

If a set of rules contains a clause

\[ p(s_1, ..., s_m) \leq b_1(t_1) \land ... \land q(v_1, ..., v_k) \land ... \land b_n(t_n) \]

where \( p \) and \( q \) occur in a cycle in the dependency graph, then for every \( i \in \{1, ..., k\} \)
- \( v_i \) is variable-free, or
- \( v_i \) is one of \( s_1, ..., s_m \), or
- \( v_i \) occurs in some \( t_j \) such that \( b_j \) does not occur in a cycle with \( p \) in the dependency graph \( (1 \leq j \leq n) \).

This ensures that arguments cannot grow arbitrarily through the application of recursive rules.
Playability / Winnability

A game is **playable** if and only if every player has at least one legal move in every non-terminal state.

(Note that in chess, if a player cannot move, it is a stalemate. Fortunately, this is a terminal state.)

In GGP, every game should be playable.

A game is **strongly winnable** if and only if, for some player, there is a sequence of individual moves of that player that leads to a terminating goal state for that player.

A game is **weakly winnable** if and only if, for every player, there is a sequence of joint moves of the players that leads to a terminating goal state for that player.

In GGP, every game should be weakly winnable, and all single player games should be strongly winnable.
Knowledge Interchange Format

Knowledge Interchange Format is a standard for programmatic exchange of knowledge represented in relational logic.

Syntax is prefix version of standard syntax.
Some operators are renamed: not, and, or.
Case-independent. Variables are prefixed with ?.

\[ r(X, Y) \leq p(X, Y) \land \neg q(Y) \]

\[ (\leq (r \ ?x \ ?y) \ (and \ (p \ ?x \ ?y) \ (not \ (q \ ?y)))) \]

or, equivalently,

\[ (\leq (r \ ?x \ ?y) \ (p \ ?x \ ?y) \ (not \ (q \ ?y))) \]

Semantics is the same.
Tic Tac Toe in KIF Notation

(role xplayer)
(role oplayer)
(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))

(<= (next (cell ?m ?n x))
  (true (cell ?m ?n b))
  (true (control ?p)))

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

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

(<= (legal ?p (mark ?m ?n))
  (true (cell ?m ?n b))
  (true (control ?p)))

(<= (line ?w) (row ?m ?w))
(<= (line ?w) (column ?n ?w))
(<= (line ?w) (diagonal ?w))

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

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

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

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

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

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

(<= goal xplayer 100)
  (line x))
(<= goal xplayer 50)
  (draw)
(<= goal xplayer 0)
  (line o))
(<= goal oplayer 100)
  (line o))
(<= goal oplayer 50)
  (draw)
(<= goal oplayer 0)
  (line x))
Playing Games
Downloads

We provide programs that might help you to implement your own General Game Playing system. All programs contain source code and are distributed under GPL.

**GameController**

GameController is a standalone game master clone written entirely in Java and developed as part of the GGPServer project. It is particularly useful for testing your own general game playing system. GameController comes with a simple GUI and a command line interface. Send bug reports and suggestions to Stephan Schiffner.

Download the most recent version from the sourceforge project page.

System requirements:
- Java 1.6 runtime environment

Usage:
java -jar GameController-0.7.3.jar

**Basic Prolog Player**

A basic player implemented in ECLiPSe Prolog based on code from FLUXPLAYER.

Download current version (1.1)

System requirements:
- ECLiPSe Prolog version 5.10 or higher

Changes since version 1.0
- the port should be free now after stopping the player
(last update: 12 March 2009)

**Basic Java Player**

A basic player implemented in Java which comes with a framework for implementing your strategies, analyzing the game, etc. It can be found on the Palamedes IDE website.

**Basic C++ Player**

A basic player implemented in C++ with the reasoner of the prolog player above.

Download current version (1.6)

System requirements:
- Linux/Unix (or any system which provides sockets)
Communication Protocol

- Manager sends **START** message to players
  
  `(START <MATCH ID> <ROLE> <GAME DESCRIPTION> <STARTCLOCK> <PLAYCLOCK>)`
  
  - Role: the name of the role you are playing (e.g. `xplayer` or `oplayer`)
  - Game description: the axioms describing the game
  - Start/play clock: how much time you have before the game begins/per turn

- Manager sends **PLAY** message to players
  
  `(PLAY <MATCH ID> <PRIOR MOVES>)`
  
  Prior moves is a list of moves, one per player
  - The order is the same as the order of roles in the game description
  - e.g. `((mark 1 1) noop)`
  - Special case: for the first turn, prior moves is `nil`

- Players send back a message of the form **MOVE**, e.g. `(mark 3 2)`

- When the previous turn ended the game, Manager sends a **STOP** message
  
  `(STOP <MATCH ID> <PRIOR MOVES>)`
GameControllerApp

MatchID: TestMatch_1
Startclock: 10
Playclock: 5

<table>
<thead>
<tr>
<th>Role</th>
<th>Type</th>
<th>Host</th>
<th>Port</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPLAYER</td>
<td>RANDOM</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OPLAYER</td>
<td>RANDOM</td>
<td>-</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

INFO(12:43:15.123): match:TestMatch_1, GDL v1
INFO(12:43:15.129): game:tictactoe
INFO(12:43:15.129): starting game with startclock=10, playclock=5
INFO(12:43:15.131): step:1
INFO(12:43:15.135): role: XPLAYER => player: local(Random)
INFO(12:43:15.136): role: OPLAYER => player: local(Random)
INFO(12:43:15.137): Sending start messages ...
INFO(12:43:15.153): time after gameStart's runThreads: Mon May 03 12:43:15 EST
Implementing a Basic General Game Player:

Blind Search
Single-Player Games: A Simple Example

Pressing button $a$ toggles $p$.
Pressing button $b$ interchanges $p$ and $q$.
Initially, $p$ and $q$ are off. Goal: $p$ and $q$ are on.
Game Description

(role robot)

Legality

(legal robot a)
(legal robot b)

Update

(<= (next p) (does robot a) (not (true p)))
(<= (next q) (does robot a) (true q))
(<= (next p) (does robot b) (true q))
(<= (next q) (does robot b) (true p))

Termination and Goal

(<= terminal (true p) (true q))
(<= (goal robot 100) (true p) (true q))
Solving Single-Player Games = Planning

- **Initial state**
  
  `{ }  
  (since there is no rule for *init* in this game)

- **Actions**
  
  a  
  Preconditions: none  
  Effects: toggles truth-value of \( p \)

  b  
  Preconditions: none  
  Effects: interchanges truth-values of \( p \) and \( q \)

- **Goal**
  
  \( p \land q \)
State Transition System

State features: \( p, q \)

Actions: \( a, b \)

Solution (= Plan): \( a, b, a \)
Many single-player games can be solved using standard search techniques

- Iterative deepening
- Bidirectional search

Special techniques

- Constraint solving (suitable for Sudoku, Gene Sequencing and the like)
Multi-Player Games: Game Tree Search
How to Deal With Simultaneous Moves

State transition graph

Bi-partite graph
Minimax

Your move: max
Minimax With \( \alpha-\beta \)-Heuristics

![Minimax Tree Diagram]

- Your move: max
- Nodes: 75, 40, 50, 80, 40, 60, 35, 20, 10
Stochastic Search (1)

Minimax Search

Monte Carlo Tree Search
(random simulations)
Stochastic Search (2)

Value of move = Average score returned by simulation

- $n = 60$
- $v = 40$
- $n = 22$, $v = 20$
- $n = 18$, $v = 20$
- $n = 20$, $v = 80$

$n = \# \text{ of sample runs}$
$v = \text{average score}$
Stochastic Search (3): Confidence Bounds

- Play one random game for each move
- For next simulation choose move

\[
\text{argmax}_i \left( v_i + C \sqrt{\frac{\log n}{n_i}} \right)
\]

Confidence bound

\[n = 60\]
\[v = 70\]

\[n_1 = 4\]
\[v_1 = 20\]

\[n_2 = 24\]
\[v_2 = 65\]

\[n_3 = 32\]
\[v_3 = 80\]

...