Prof Michael Thielscher
Adjunct at School of Computing & Mathematics
University of Western Sydney

School of Computer Science and Engineering
The University of New South Wales

mit@cse.unsw.edu.au
Computer Game Playing

Kasparov vs. Deep Blue (1997)
General Game Playing

General Game Players are systems
- able to understand formal descriptions of arbitrary games
- able to learn to play these games effectively.

Translation: They don't know the rules until the game starts.

Unlike specialised game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.
Variety of Games
Example: Noughts And Crosses
Single-Player Games
Chess
Bughouse Chess (a 4-Player Variant of Chess)
Other Games
International Activities

Websites – www.general-game-playing.de
games.stanford.edu

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

World Cup, administered by Stanford
- 2005 – Cluneplayer (USA)
- 2006 – Fluxplayer (Germany)
- 2007, 2008 – Cadiaplayer (Iceland)
- 2009, 2010 – Ary (France)
- 2011 – TurboTurtle (USA)
General Game Playing and Artificial Intelligence

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

![Ordinary Systems Diagram]

General Systems

![General Systems Diagram]
Describing the Rules of a Game to a General Game Player
Finite Synchronous Games

Finite environment
- Environment with finitely many positions (= states)
- One initial state and one or more terminal states

Finite Players
- Fixed finite number of players
- Each with finitely many “actions”
- Each with one or more goal states

Synchronous Update
- All players move on all steps (possibly some “no-ops”)
- Environment changes only in response to moves
Direct Description

Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of tables (for legal moves, update, etc.)

Problem: Size of description. Even though everything is finite, the necessary tables can be large (e.g. $\sim 10^{44}$ states in Chess)

Solutions:
- Reformulate in modular fashion
- Use compact encoding
Example: Noughts And Crosses

```
X
O
X
```

```
(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)
```
**Game Description Language (GDL): Facts and Rules**

### Some Facts

- `(role xplayer)`
- `(role oplayer)`
- `(init (cell 1 1 b))`
- `(init (cell 1 2 b))`
- ...
- `(init (cell 3 3 b))`
- `(init (control xplayer))`

### Some Rules

- `(<= (legal ?p (mark ?m ?n)) (true (cell ?m ?n b)) (true (control ?p)))`
- `(<= (next (cell ?m ?n x)) (does xplayer (mark ?m ?n)))`
- `(<= (next (cell ?m ?n o)) (does oplayer (mark ?m ?n)))`

All highlighted expressions are pre-defined keywords in GDL.
What we see

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{legal} \ ?p \ (\text{mark} \ ?m \ ?n)\text{))} \\
&\text{ (true (cell ?m ?n b))} \\
&\text{ (true (control ?p)))}
\end{align*}
\]

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{next} \ (\text{cell} \ ?m \ ?n \ x)\text{))} \\
&\text{ (does xplayer (mark ?m ?n)))}
\end{align*}
\]

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{next} \ (\text{cell} \ ?m \ ?n \ o)\text{))} \\
&\text{ (does oplayer (mark ?m ?n)))}
\end{align*}
\]

What they see

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{legal} \ ?p \ (\text{dukep} \ ?m \ ?n)\text{))} \\
&\text{ (true (welcoul ?m ?n kwq))} \\
&\text{ (true (himenoing ?p)))}
\end{align*}
\]

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{next} \ (\text{welcoul} \ ?m \ ?n \ ygg)\text{))} \\
&\text{ (does lorchi (dukep ?m ?n)))}
\end{align*}
\]

\[
\begin{align*}
\text{(}\leq& \text{ (} \text{next} \ (\text{welcoul} \ ?m \ ?n \ pyr)\text{))} \\
&\text{ (does gniste (dukep ?m ?n)))}
\end{align*}
\]
Logic Programs: A Subset of First-Order Logic

Clauses

- **Facts**: atoms
- **Rules**: Head \( \leq \) Body

**Head**: atomic formula (i.e., predicate with arguments)

**Body**: formula using conjunction, disjunction, negation

A **logic program** is a finite collection of clauses.
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
Back to Noughts And Crosses

X
O
X

(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)
Noughts and Crosses: Vocabulary

- **Object constants**
  - `xplayer, oplayer` (Players)
  - `x, o, b` (Marks)
  - `noop` (Move)

- **Functions**
  - `(cell number number mark)` (State feature)
  - `(control player)` (State feature)
  - `(mark number number)` (Move)

- **Predicates**
  - `(row number mark)` (Move)
  - `(column number mark)` (Move)
  - `(diagonal mark)` (Move)
  - `(line mark)`
  - `open`
  - `draw`
Players and Initial State

```scheme
(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))
```
Move Generator

(\leq \text{legal} \quad ?p \quad (\text{mark} \quad ?m \quad ?n))

(\text{true} \quad (\text{cell} \quad ?m \quad ?n \quad \text{b}))

(\text{true} \quad (\text{control} \quad ?p))

(\leq \text{legal} \quad \text{xplayer} \quad \text{noop})

(\text{true} \quad (\text{control} \quad \text{oplayer}))

(\leq \text{legal} \quad \text{oplayer} \quad \text{noop})

(\text{true} \quad (\text{control} \quad \text{xplayer}))

Conclusions:

(\text{legal} \quad \text{xplayer} \quad \text{noop})

(\text{legal} \quad \text{oplayer} \quad (\text{mark} \quad 1 \quad 2))

...  

(\text{legal} \quad \text{oplayer} \quad (\text{mark} \quad 3 \quad 2))

\begin{align*}
\text{X} & \quad & \text{O} & \quad & \text{X} \\
\text{X} & \quad & \text{O} & \quad & \text{X} \\
\text{cell} & & \text{1} & & \text{1} & & \text{x} \\
\text{cell} & & \text{1} & & \text{2} & & \text{b} \\
\text{cell} & & \text{1} & & \text{3} & & \text{b} \\
\text{cell} & & \text{2} & & \text{1} & & \text{b} \\
\text{cell} & & \text{2} & & \text{2} & & \text{o} \\
\text{cell} & & \text{2} & & \text{3} & & \text{b} \\
\text{cell} & & \text{3} & & \text{1} & & \text{b} \\
\text{cell} & & \text{3} & & \text{2} & & \text{b} \\
\text{cell} & & \text{3} & & \text{3} & & \text{x} \\
\end{align*}
Physics: Example

\[(\text{cell 1 1 x})\]  \rightarrow  \text{oplayer}  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 1 1 x})

\[(\text{cell 1 2 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 1 2 b})

\[(\text{cell 1 3 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 1 3 o})

\[(\text{cell 2 1 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 2 1 b})

\[(\text{cell 2 2 o})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 2 2 o})

\[(\text{cell 2 3 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 2 3 b})

\[(\text{cell 3 1 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 3 1 b})

\[(\text{cell 3 2 b})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 3 2 b})

\[(\text{cell 3 3 x})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{cell 3 3 x})

\[(\text{control oplayer})\]  \rightarrow  \text{mark(1,3)}  \rightarrow  (\text{control xplayer})
Physics

\[
(\leq (\text{next } (\text{cell } ?m ?n x)) (\text{does } \text{xplayer} (\text{mark } ?m ?n)))
\]

\[
(\leq (\text{next } (\text{cell } ?m ?n o)) (\text{does } \text{oplayer} (\text{mark } ?m ?n)))
\]

\[
(\leq (\text{next } (\text{cell } ?m ?n ?w))
  (\text{true } (\text{cell } ?m ?n ?w))
  (\text{does } ?p (\text{mark } ?j ?k))
  (\text{or } (\text{distinct } ?m ?j) (\text{distinct } ?n ?k)))
\]

\[
(\leq (\text{next } (\text{control } \text{xplayer})) (\text{true } (\text{control } \text{oplayer})))
\]

\[
(\leq (\text{next } (\text{control } \text{oplayer})) (\text{true } (\text{control } \text{xplayer})))
\]
### Supporting Concepts

<table>
<thead>
<tr>
<th>Formula</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\leq \text{(row } ?m \text{ } ?w))</td>
<td>(\text{true (cell } ?m \text{ } 1 \text{ } ?w)), (\text{true (cell } ?m \text{ } 2 \text{ } ?w)), (\text{true (cell } ?m \text{ } 3 \text{ } ?w))</td>
</tr>
<tr>
<td>(\leq \text{(diagonal } ?w))</td>
<td>(\text{true (cell } 1\text{ } 1 \text{ } ?w)), (\text{true (cell } 2\text{ } 2 \text{ } ?w)), (\text{true (cell } 3\text{ } 3 \text{ } ?w))</td>
</tr>
<tr>
<td>(\leq \text{(column } ?n \text{ } ?w))</td>
<td>(\text{true (cell } 1\text{ } ?n \text{ } ?w)), (\text{true (cell } 2\text{ } ?n \text{ } ?w)), (\text{true (cell } 3\text{ } ?n \text{ } ?w))</td>
</tr>
<tr>
<td>(\leq \text{(diagonal } ?w))</td>
<td>(\text{true (cell } 1\text{ } 3 \text{ } ?w)), (\text{true (cell } 2\text{ } 2 \text{ } ?w)), (\text{true (cell } 3\text{ } 1 \text{ } ?w))</td>
</tr>
</tbody>
</table>
Termination and Goal Values

\[
\begin{align*}
\text{terminal} & \quad (\text{or} \ (\text{line} \ x) \\
& \quad \quad (\text{line} \ o)) \\
\text{terminal} & \quad \text{(not open)} \\
\text{line} \ ?w & \quad (\text{row} \ ?m \ ?w) \\
\text{line} \ ?w & \quad (\text{column} \ ?n \ ?w) \\
\text{line} \ ?w & \quad (\text{diagonal} \ ?w) \\
\text{open} & \quad \text{(true} \ (\text{cell} \ ?m \ ?n \ b)) \\
\text{goal} \ xplayer \ 100 & \quad (\text{line} \ x) \\
\text{goal} \ xplayer \ 50 & \quad \text{draw} \\
\text{goal} \ xplayer \ 0 & \quad (\text{line} \ o) \\
\text{goal} \ oplayer \ 100 & \quad (\text{line} \ o) \\
\text{goal} \ oplayer \ 50 & \quad \text{draw} \\
\text{goal} \ oplayer \ 0 & \quad (\text{line} \ x) \\
\text{draw} & \quad \text{(not} \ (\text{line} \ x)) \\
& \quad \text{(not} \ (\text{line} \ o)) \\
& \quad \text{(not} \ \text{open})
\end{align*}
\]
Summary: Noughts And Crosses

(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))

(legal ?p (mark ?m ?n))
(legal xplayer noop)
(legal oplayer noop)
(legal ?p (mark ?m ?n))

(next (cell ?m ?n x))
(next (cell ?m ?n o))

(row ?m ?w)
(column ?n ?w)
(diagonal ?w)

(goal xplayer 100)
(goal xplayer 50)
(goal xplayer 0)
(goal oplayer 100)
(goal oplayer 50)
(goal oplayer 0)

(line ?w)
(row ?m ?w)
(column ?n ?w)
(diagonal ?w)
(open)
(terminal)
(terminal)

(goal xplayer 100)
(goal xplayer 50)
(goal xplayer 0)
(goal oplayer 100)
(goal oplayer 50)
(goal oplayer 0)

(line x)
(draw)
(not open)
Playing Games
http://euklid.inf.tu-dresden.de:8180/ggpserver/index.jsp
Game Manager
Communication Protocol

Manager sends **START** message to players

\[(\text{START } \text{<MATCH ID> } \text{<ROLE> } \text{<GAME DESCRIPTION> } \text{<STARTCLOCK> } \text{<PLAYCLOCK>})\]

- 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

\[(\text{PLAY } \text{<MATCH ID> } \text{<PRIOR MOVES>})\]

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. \(((\text{mark } 1 \ 1) \ \text{noop})\)
- Special case: for the first turn, prior moves is \text{nil}

Players send back a message of the form **MOVE**, e.g. \((\text{mark } 3 \ 2)\)

When the previous turn ended the game, Manager sends a **STOP** message

\[(\text{STOP } \text{<MATCH ID> } \text{<PRIOR MOVES>})\]
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.

**GameManager**

GameManager is a standalone game master close written entirely in Java and developed as part of the GGPPlayground project. It is particularly useful for testing your own general game playing systems. GameManager comes with a simple GUI and a command line interface. Send bug reports and suggestions to Stefan Schmitt.

Download the most recent version from the sourceforge project page.

System requirements:
- Java 1.5 runtime environment

Usage:
```
java -jar gamemaster.jar [-FF:30]
```

**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 9.10 or higher

Changes since version 1.0:
- the port should be free new 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 Parameida-IDE website.

**Basic C++ Player**

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

Download current version (1.5)

System requirements:
- Linux/Unix (or any system which provides gmake)
GameControllerApp

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 ...
Implementing a General Game Player
Single-Player Games: A (Very) 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 \textit{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): \( a, b, a \)
Single-Player Games with Complete Information

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 (Example)
How to Deal With Simultaneous Moves

State transition graph

Bi-partite graph
Minimax

your move: max

40

75 40 50

40

80 40 60

10

35 20 10
Minimax With $\alpha$-$\beta$-Heuristics
Stochastic Search (1)

Game Tree Search

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

Value of move = Average score returned by simulation

\( n = \# \text{ of sample runs} \)
\( v = \text{average score} \)

\[ n = 60 \]
\[ v = 40 \]

\[ n = 22 \]
\[ v = 20 \]

\[ n = 18 \]
\[ v = 20 \]

\[ n = 20 \]
\[ v = 80 \]
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

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

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

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

\[ n_3 = 32 \]
\[ v_3 = 80 \]
Advanced Techniques: Metagaming

- Blind search requires no intelligence but is limited in its ability to play well
- Solution: assign intermediate scores to nodes based on an evaluation function
- Metagaming means to reason about properties of games in order to automatically learn evaluation functions
- This is the intelligence built into a general game player!
Further Reading

If you're interested in doing a project/thesis/... on General Game Playing:
Contact me at
mit@cse.unsw.edu.au

Further Reading

www.general-game-playing.de
games.stanford.edu/competition/misc/aaai.pdf
www.ru.is/faculty/hif/papers/cadiaplayer_aaai08.pdf
cgi.cse.unsw.edu.au/~mit/Papers/AAAI07a.pdf