Outline

- General Game Playing
- First-Order Logic
- Logic Programs
- The Game Description Language GDL
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 specialized game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.
Variety of Games
Noughts And Crosses
Chess
Bughouse Chess
Kriegspiel
Other Games
Single-Player "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)
- 2010 – Ary (France)
- upcoming World Cup: July 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 ← Application-Specific System → Environment
```

General Systems

```
Client ← General System → Environment
```

Rules

© Michael Thielscher, Michael Genesereth 2011
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”)"n
- Environment changes only in response to moves
Games as State Machines

[Diagram of a state machine with states labeled a, b, c, d, e, f, g, h, i, j, k.
Initial State, Terminal States, & Simultaneous Moves

Diagram: A graph with nodes labeled a, b, c, d, e, f, g, h, i, j, and k, connected by directed edges labeled with 'a/a', 'a/b', 'b/a', or 'b/b'. The initial state is indicated by an arrow pointing towards node a.
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
States versus Features

In many cases, worlds are best thought of in terms of atomic features that may change; e.g. “position-of-white-queen”, “black-can-castle”. Moves (a.k.a. actions) affect subsets of these features.

States represent all possible ways the world can be. As such, the number of states is exponential in the number of features of the world, and the transition tables are correspondingly large.

Solution: Represent features directly and describe how actions change individual features rather than entire states.
Example: Noughts And Crosses

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

- \texttt{role(xplayer)}
- \texttt{role(oplayer)}
- \texttt{init(cell(1,1,b))}
- \texttt{init(cell(1,2,b))}
- \texttt{...}
- \texttt{init(cell(3,3,b))}
- \texttt{init(control(xplayer))}

Some Rules

- \texttt{legal(P,mark(M,N))} <= \texttt{true(cell(M,N,b))} \land \texttt{true(control(P))}
- \texttt{next(cell(M,N,x))} <= \texttt{does(xplayer,mark(M,N))}
- \texttt{next(cell(M,N,o))} <= \texttt{does(oplayer,mark(M,N))}

All highlighted expressions are pre-defined keywords in GDL.
# No Built-In Assumptions

<table>
<thead>
<tr>
<th>What we see</th>
<th>What they see</th>
</tr>
</thead>
<tbody>
<tr>
<td>legal ((P, \text{mark}(M,N)) \leq)</td>
<td>legal ((P, \text{dukepse}(M,N)) \leq)</td>
</tr>
<tr>
<td>(\text{true}(\text{cell}(M,N,b)) \land \text{true}(\text{control}(P)))</td>
<td>(\text{true}(\text{welcoul}(M,N,kwq)) \land \text{true}(\text{himenoing}(P)))</td>
</tr>
<tr>
<td>next ((\text{cell}(M,N,x)) \leq)</td>
<td>next ((\text{welcoul}(M,N,ygg)) \leq)</td>
</tr>
<tr>
<td>(\text{does}(\text{xplayer}, \text{mark}(M,N)))</td>
<td>(\text{does}(\text{lorchi}, \text{dukepse}(M,N)))</td>
</tr>
<tr>
<td>next ((\text{cell}(M,N,o)) \leq)</td>
<td>next ((\text{welcoul}(M,N,pyr)) \leq)</td>
</tr>
<tr>
<td>(\text{does}(\text{oplayer}, \text{mark}(M,N)))</td>
<td>(\text{does}(\text{gniste}, \text{dukepse}(M,N)))</td>
</tr>
</tbody>
</table>
First-Order Logic: Vocabulary

Object Variables: \( X, Y, Z \)
Object Constants: \( a, b, c \)
Functions: \( f, g, h \)
Predicates: \( p, q, r \)
Connectives: \( \neg, \land, \lor, \leq \)
Quantifiers: \( \forall, \exists \)

The arity of a function or predicate is the number of arguments that can be supplied.
First-Order Logic: Syntax

Terms
- Variables: X, Y, Z
- Constants: a, b, c
- Functional terms: f(a), g(a,X), h(a,b,f(Y))

Sentences
- Atoms: p(X), q(a,g(a,b))
- Literals: p(X), ¬p(X) (i.e. atoms and negated atoms)
- Sentences: p(a) ∨ ¬p(a)
∀X ∀Y p(X,Y) <= ∀Y ∀X p(X,Y)
∀X p(f(X)) <= ∃Y q(X,f(Y)) ∧ ¬r(a)
First-Order Logic: Semantics

The **Herbrand universe** for a logic language is the set of all variable-free terms.

Example 1:
- **Object Constants:** \( a, b \)
- **Herbrand Universe:** \( \{a, b\} \)

Example 2:
- **Object Constant:** \( a \)
- **Unary function:** \( f \)
- **Herbrand Universe:** \( \{a, f(a), f(f(a)), \ldots\} \)
Semantics (Cont'd)

The **Herbrand base** is the set of all variable-free atoms. 
Example: \( \{p(a), p(b), q(a, a), q(a, b), q(b, a), q(b, b)\} \)

A **model** is an arbitrary subset of the Herbrand base. 

Examples: 
- \( M_1 = \{p(a), q(a, b), q(b, a)\} \)
- \( M_2 = \{p(a), p(b), q(a, a), q(a, b), q(b, a), q(b, b)\} \)
- \( M_3 = \{\} \)
Semantics (Finished)

\( \mathcal{M} \) is a model for a sentence \( \varphi \) under the following conditions.

- \( \mathcal{M} \) model for a variable-free atom \( \varphi \) iff \( \varphi \in \mathcal{M} \)
- \( \mathcal{M} \) model for \( \neg \varphi \) iff \( \mathcal{M} \) not a model for \( \varphi \)
- \( \mathcal{M} \) model for \( \varphi \land \psi \) iff \( \mathcal{M} \) model for \( \varphi \) and model for \( \psi \)
- \( \mathcal{M} \) model for \( \varphi \lor \psi \) iff \( \mathcal{M} \) model for \( \varphi \) or model for \( \psi \) (or both)
- \( \mathcal{M} \) model for \( \varphi \leq \psi \) iff \( \mathcal{M} \) model for \( \varphi \) whenever \( \mathcal{M} \) model for \( \psi \)
- \( \mathcal{M} \) model for \( \forall x \varphi \) iff \( \mathcal{M} \) model for \( \varphi\{x/t\} \) for all terms \( t \) in the Herbrand universe
- \( \mathcal{M} \) model for \( \exists x \varphi \) iff \( \mathcal{M} \) model for \( \varphi\{x/t\} \) for some \( t \) in the Herbrand universe

\( \varphi\{x/t\} \) means to replace each occurrence of \( x \) by \( t \) in \( \varphi \).
Examples

Recall the models

- $M_1 = \{ p(a), q(a,b), q(b,a) \}$
- $M_2 = \{ p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b) \}$
- $M_3 = \{ \}$

Some examples:

- $M_1$ is a model for $p(a) \land \neg p(b)$, whereas $M_2$ and $M_3$ are not.
- $M_2$ is a model for $p(b) \leq p(a)$. So is $M_3$ (!)
- All three are models for $\forall X \forall Y q(X,Y) \leq q(Y,X)$

If all models of sentences $\Phi$ also satisfy $\varphi$, then $\varphi$ is a logical consequence of $\Phi$. 
Logic Programs: A Subset of First-Order Logic

Clauses
- Facts: atoms
- Rules: Head $\leq$ Body

  Head: atom
  Body: sentence built from $\land$, $\lor$, literal

All variables in a clause are universally quantified (over the whole clause).

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

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  
  - x, o, b  
  - noop

- **Functions**
  - cell(number, number, mark)
  - control(player)
  - mark(number, number)

- **Predicates**
  - row(number, mark)
  - column(number, mark)
  - diagonal(mark)
  - line(mark)
  - open
  - draw

Players
Marks
Move
Feature
Feature
Move
Players and Initial State

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

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

\[
\text{legal}(\text{xplayer}, \text{noop}) \leq= \\
\text{true}(\text{control}(\text{oplayer}))
\]

\[
\text{legal}(\text{oplayer}, \text{noop}) \leq= \\
\text{true}(\text{control}(\text{xplayer}))
\]

Conclusions:

\[
\text{legal}(\text{xplayer}, \text{noop}) \\
\text{legal}(\text{oplayer}, \text{mark}(1,2)) \\
\ldots \\
\text{legal}(\text{oplayer}, \text{mark}(3,2))
\]

\[
\begin{array}{c|c|c|c}
& X & O & X \\
\hline
X & O & X & O \\
\end{array}
\]

- 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)

\text{control}(\text{oplayer})
Physics: Example

<table>
<thead>
<tr>
<th>cell(1,1,x)</th>
<th>cell(1,2,b)</th>
<th>cell(2,1,b)</th>
<th>cell(2,2,o)</th>
<th>cell(2,3,b)</th>
<th>cell(3,1,b)</th>
<th>cell(3,2,b)</th>
<th>cell(3,3,x)</th>
</tr>
</thead>
</table>

- control(oplayer)
- mark(1,3)

---

![Diagram of Tic-Tac-Toe game](image-url)
Physics

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

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

\[
\text{next}(\text{cell}(M,N,W)) \leq \text{true}(\text{cell}(M,N,W)) \land \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\]

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

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

\[
\begin{align*}
\text{next}(\text{cell}(M,N,W)) & \leq \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]

\[
\begin{align*}
& \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]

\[
\begin{align*}
& \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]

\[
\begin{align*}
& \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]

\[
\begin{align*}
& \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]

\[
\begin{align*}
& \text{true}(\text{cell}(M,N,W)) \land \\
& \quad \text{next}(\text{control}(\text{xplayer})) \leq \text{true}(\text{control}(\text{oplayer}))
\end{align*}
\]
Supporting Concepts

\[
\begin{align*}
\text{row}(M, W) & \leq \text{true}(\text{cell}(M,1,W)) \land \text{true}(\text{cell}(M,2,W)) \land \text{true}(\text{cell}(M,3,W)) \\
\text{column}(N, W) & \leq \text{true}(\text{cell}(1,N,W)) \land \text{true}(\text{cell}(2,N,W)) \land \text{true}(\text{cell}(3,N,W)) \\
\text{diagonal}(W) & \leq \text{true}(\text{cell}(1,1,W)) \land \text{true}(\text{cell}(2,2,W)) \land \text{true}(\text{cell}(3,3,W))
\end{align*}
\]
Termination and Goal Values

\[
\begin{align*}
\text{terminal} & \leq \text{line}(x) \lor \text{line}(o) \\
\text{terminal} & \leq \neg \text{open} \\
\text{line}(W) & \leq \text{row}(M,W) \lor \text{column}(N,W) \lor \text{diagonal}(W) \\
\text{open} & \leq \text{true}(\text{cell}(M,N,b)) \\
\text{goal}(xplayer,100) & \leq \text{line}(x) \\
\text{goal}(xplayer, 50) & \leq \text{draw} \\
\text{goal}(xplayer, 0) & \leq \text{line}(o) \\
\text{goal}(oplayer,100) & \leq \text{line}(o) \\
\text{goal}(oplayer, 50) & \leq \text{draw} \\
\text{goal}(oplayer, 0) & \leq \text{line}(x) \\
\text{draw} & \leq \neg \text{line}(x) \land \neg \text{line}(o) \land \neg \text{open}
\end{align*}
\]
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*, *goal values*, and *update* in terms of the relations *true* and *does*.

The upshot is that in every state every player can determine legality, termination, goal values, and, given a joint move, can update the state.
Knowledge Interchange Format

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

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

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

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

or, equivalently,

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

Semantics is the same.
Noughts And Crosses in KIF

```
(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))
    (does xplayer (mark ?m ?n))
    (true (cell ?m ?n b))
    (true (control ?p)))

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

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

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

<= open

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

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

<= (line ?w)
    (true (cell ?m ?n b))

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

<= (line x)

<= (line o)

<= terminal

<= open

<= terminal

<= (goal xplayer 100)
    (line x)

<= (goal xplayer 50)
    draw

<= (goal oplayer 100)
    (line o)

<= (goal oplayer 50)
    draw
```

#### Logic

**(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))**
**(does xplayer (mark ?m ?n))**
**(true (cell ?m ?n b))**
**(true (control ?p)))**

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

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

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

<= open

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

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

**(<= (line ?w)**
**(true (cell ?m ?n b)))**

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

**(<= (line x)**

**(<= (line o)**

<= terminal

<= open

<= terminal

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

**(<= (goal xplayer 50)**
**draw**

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

**(<= (goal oplayer 50)**
**draw**

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

**(<= (goal oplayer 0)**
**(line x))**
http://euklid.inf.tu-dresden.de:8180/ggpserver/index.jsp
Game Manager

- Graphics for Spectators
- Game Manager
- Temporary State Data
- Game Descriptions
- Match Records
- Player

TCP/IP
Communication Protocol

- Manager sends **START** message to players
  
  \[(\text{START} \ <\text{MATCH ID}> \ <\text{ROLE}> \ <\text{GAME DESCRIPTION}> \ <\text{STARTCLOCK}> \ <\text{PLAYCLOCK}>\)]
  
  - Match ID: the name of the game
  - 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 **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}>\)]
http://www.general-game-playing.de/downloads.html

<table>
<thead>
<tr>
<th>Downloads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Downloads</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>GameController</strong></td>
<td>GameController is a standalone game master close written entirely in Java and developed as part of the GGP{server} 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 Schiffer. Download the most recent version from the sourceforge project page.</td>
</tr>
<tr>
<td><strong>System requirements:</strong></td>
<td></td>
</tr>
<tr>
<td>• Java 1.5 runtime environment</td>
<td></td>
</tr>
<tr>
<td><strong>Usage:</strong></td>
<td></td>
</tr>
<tr>
<td>Java -Dgc.gamecontroller.renter=<a href="http://j2ee.de">http://j2ee.de</a></td>
<td></td>
</tr>
<tr>
<td><strong>Basic Prolog Player</strong></td>
<td>A basic player implemented in ECLiPSe Prolog based on code from FLUXPLAYER. Download current version (1.1)</td>
</tr>
<tr>
<td><strong>System requirements:</strong></td>
<td></td>
</tr>
<tr>
<td>• ECLiPSe Prolog version 9.10 or higher</td>
<td></td>
</tr>
<tr>
<td><strong>Changes since version 1.0:</strong></td>
<td></td>
</tr>
<tr>
<td>• the port should be free new after stopping the player</td>
<td></td>
</tr>
<tr>
<td>(last update: 12 March 2009)</td>
<td></td>
</tr>
<tr>
<td><strong>Basic Java Player</strong></td>
<td>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 Paternoster-JDE website.</td>
</tr>
<tr>
<td><strong>Basic C++ Player</strong></td>
<td>A basic player implemented in C++ with the measurer of the prolog player above. Download current version (1.6)</td>
</tr>
<tr>
<td><strong>System requirements:</strong></td>
<td></td>
</tr>
<tr>
<td>• Linux/Unix (or any system which provides sockets)</td>
<td></td>
</tr>
</tbody>
</table>
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 ...