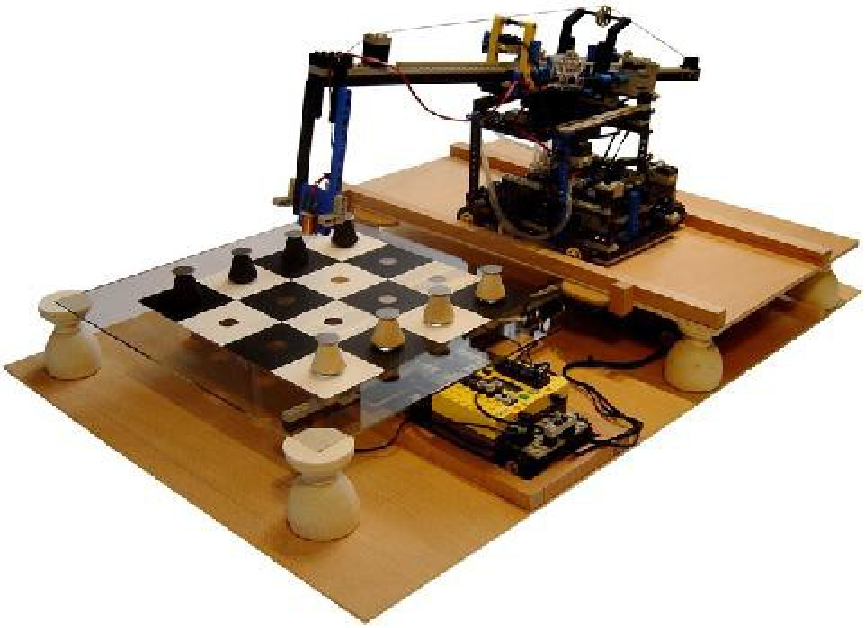
Outline

- General Game Playing
- Propositional Logic
- First-Order Logic
- The Game Description Language GDL

Introduction:
General Game Playing

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.

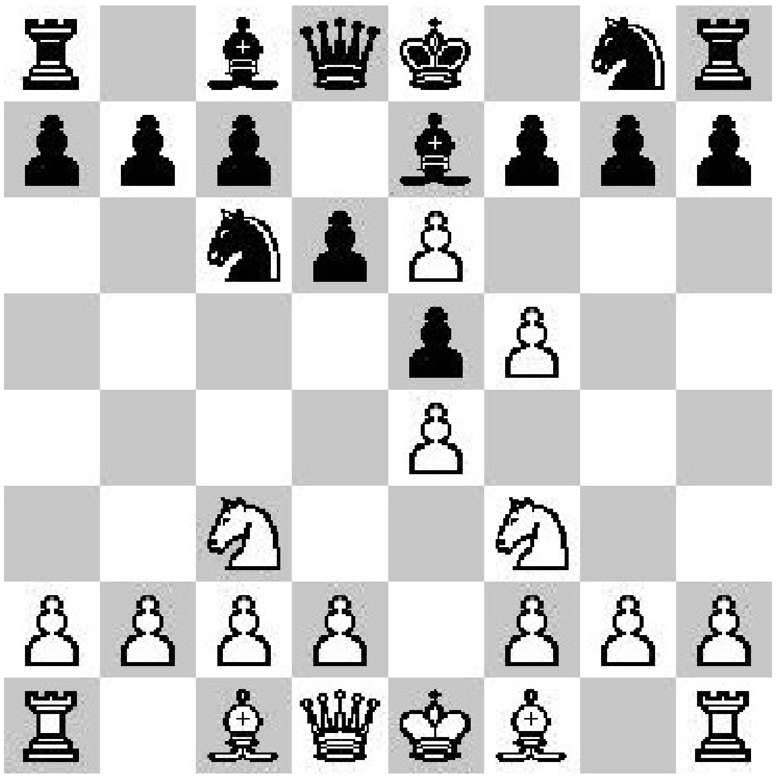
Unlike specialized game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.

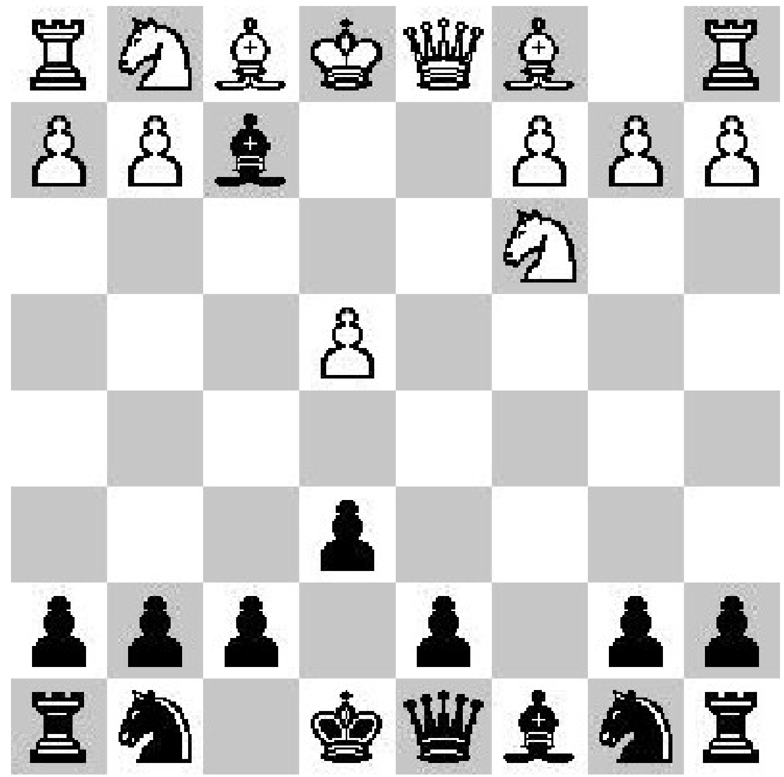
Variety of Games

Noughts And Crosses

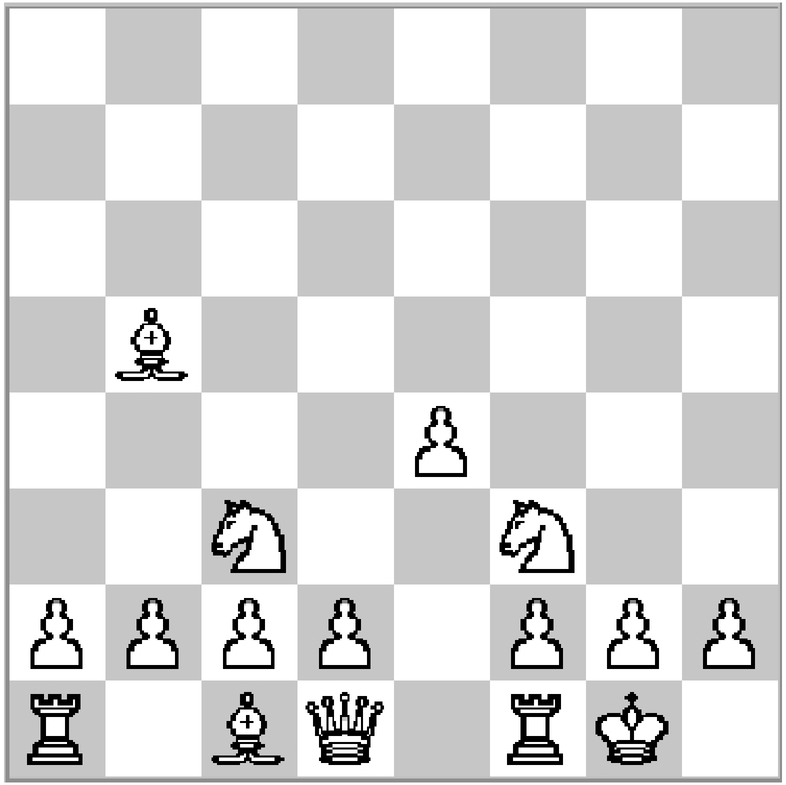
Chess

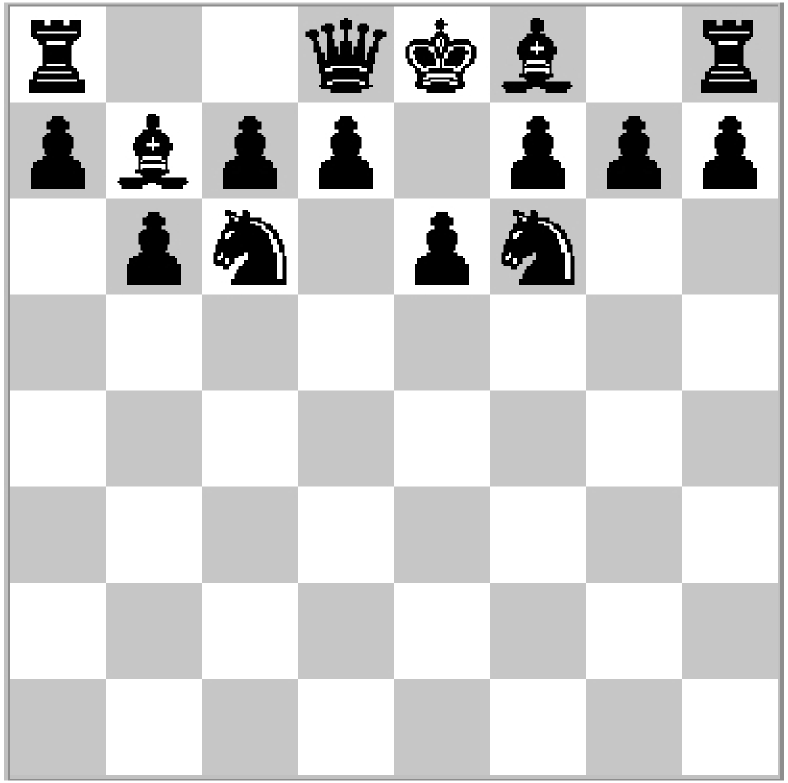
Bughouse Chess



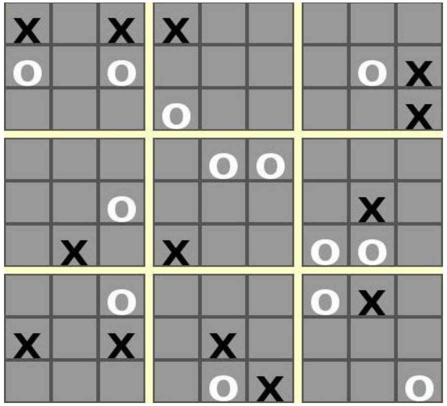


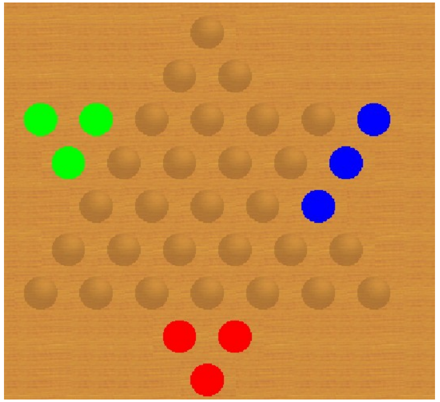
Kriegspiel



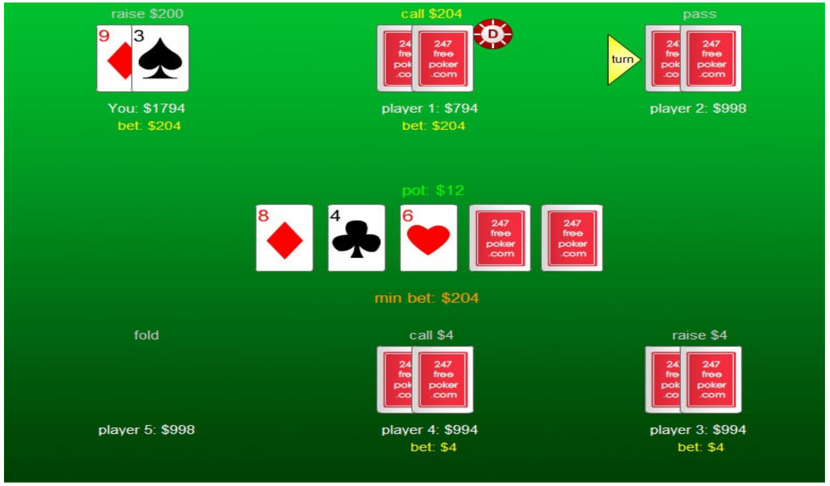


Other Games



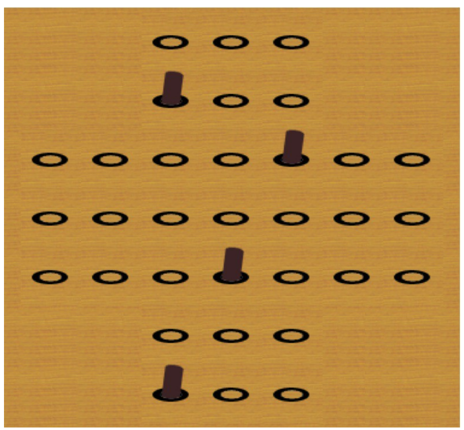


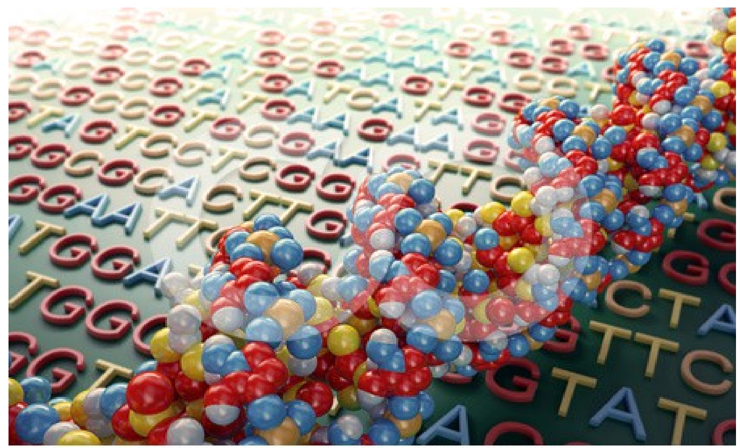

Select the neighboring nation to invade
Human


raise $200
You: $1794
bet: $204
call $204
player 1: $794
bet: $204
pass
turn
player 2: $998
pot: $12
min bet: $204
fold
player 5: $998
call $4
player 4: $994
bet: $4
raise $4
player 3: $994
bet: $4

Single-Player Games (aka. Planning)

7				1			4	
	2				9		5	6
		4		6		2		
		8	6		1		2	
		7				1		
	9		3		8	6		
		5		2		4		
8	4		1				6	
	1			8				2





International Activities

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

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

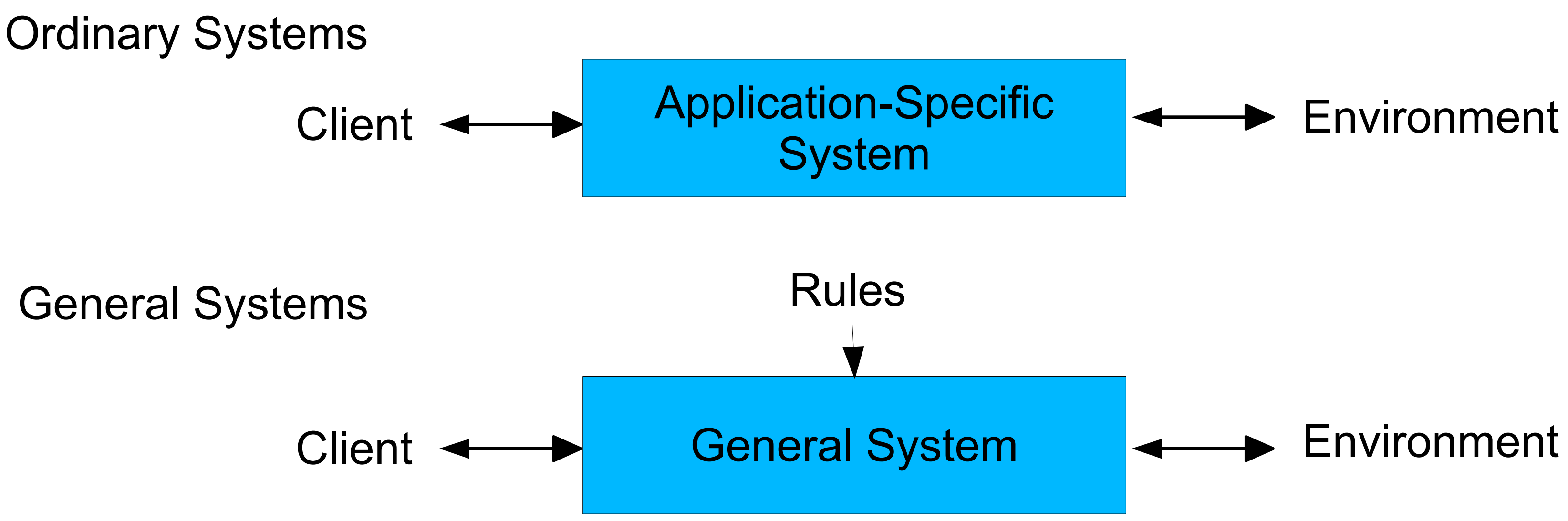
World Cup, administered by Stanford

- 2005 – Cluneplayer (USA)
- 2006 – Fluxplayer (Germany)
- 2007, 2008 – Cadiaplayer (Iceland)
- 2009, 2010 – Ary (France)
- 2011 – TurboTurtle (USA)
- 2012 – Cadiaplayer (Iceland)

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 examples of systems that can adapt to radically different environments without human intervention.



Finite Synchronous Games

Finite environment

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

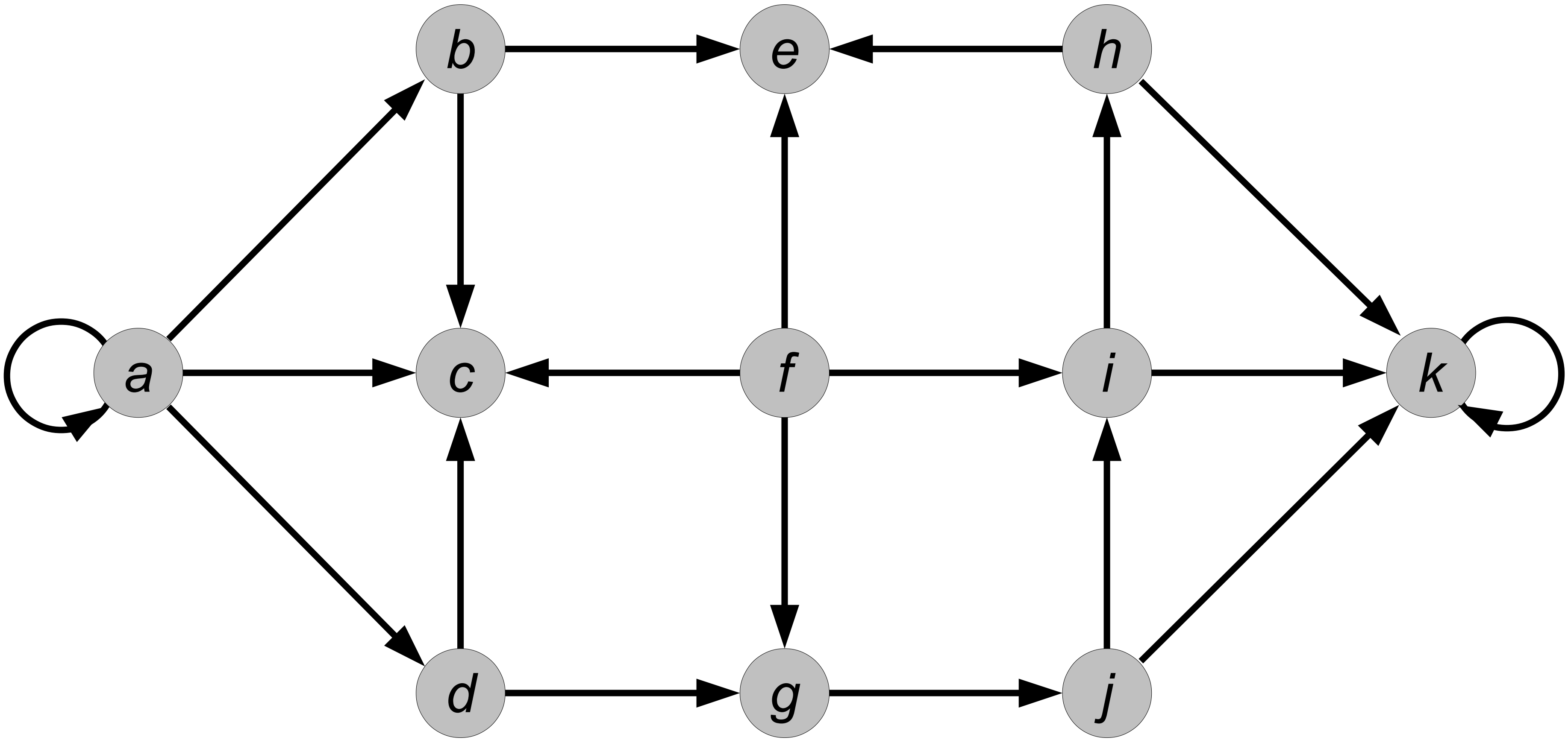
Finite Players

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

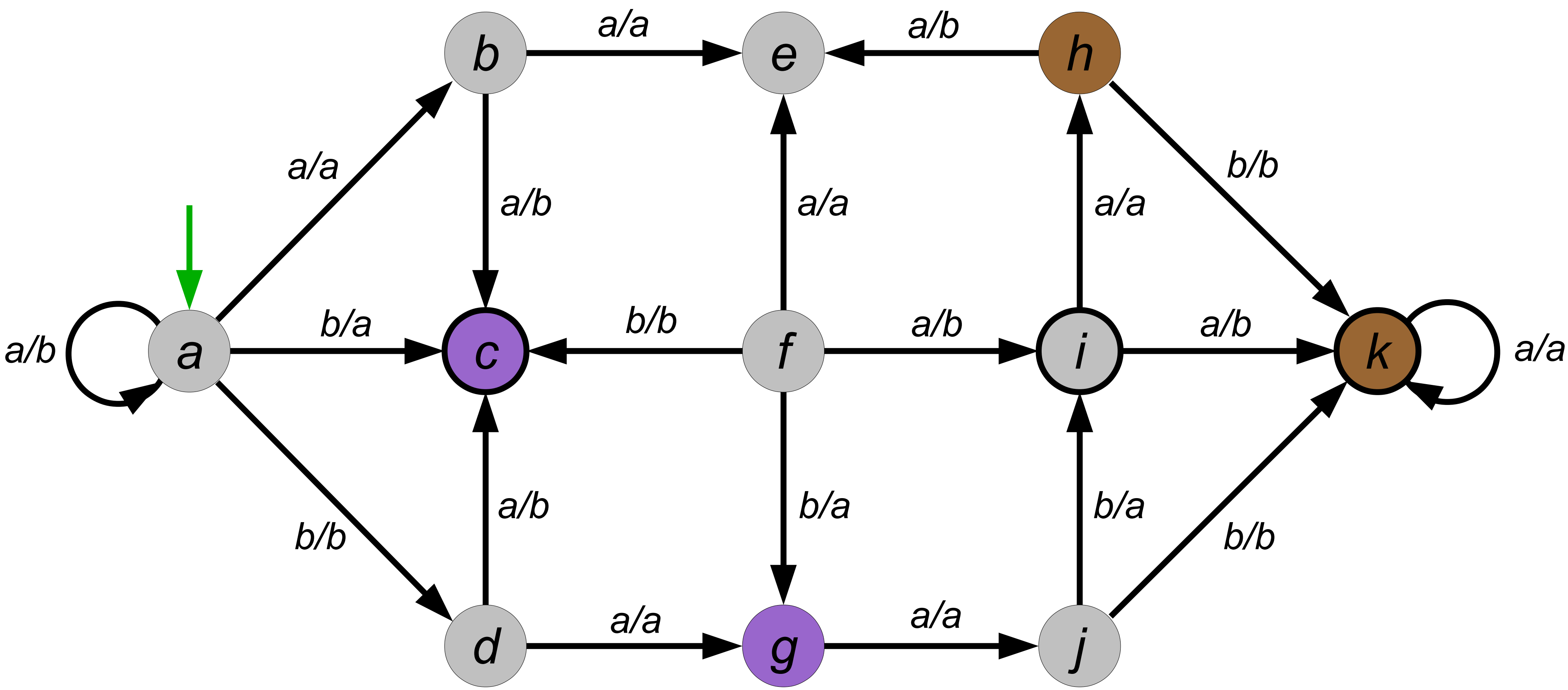
Synchronous Update

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

Games as State Machines



Initial State, Terminal States, & Simultaneous 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

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”.

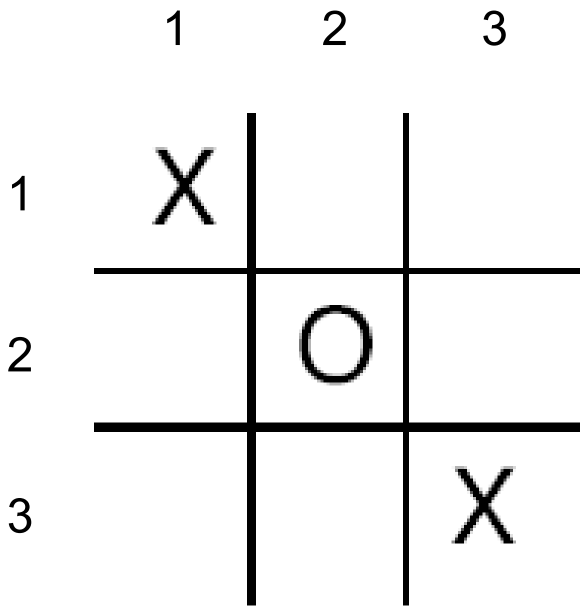
Moves (a.k.a. actions) affect subsets of these features.

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

```
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.

Pure Logic: No Built-In Assumptions

What we see

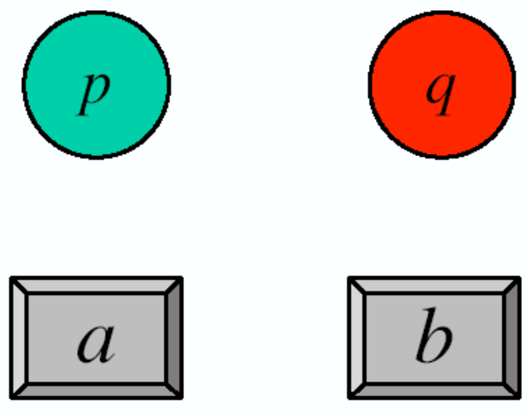
```
legal(P,mark(M,N)) <=
   true(cell(M,N,b)) ∧
   true(control(P))

next(cell(M,N,x)) <=
   does(xplayer,mark(M,N))

next(cell(M,N,o)) <=
   does(oplayer,mark(M,N))
```

What they see

```
legal(P,dukepse(M,N)) <=
   true(welcoul(M,N,kwq)) ∧
   true(himenoing(P))

next(welcoul(M,N,ygg)) <=
   does(lorchi,dukepse(M,N))

next(welcoul(M,N,pyr)) <=
   does(gniste,dukepse(M,N))
```

Propositional Logic

A Simpler Example First: the Buttons-And-Light Game



Pressing button *a* toggles *p*.

Pressing button *b* interchanges *p* and *q*.

Initially, *p* and *q* are off. Goal: *p* and *q* are on.

Propositional Logic: Vocabulary

Proposition Symbols: `p, q, r`

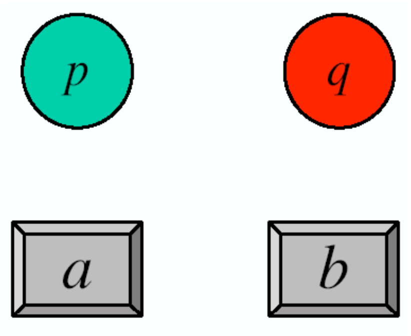
Logical Connectives:

- $\neg$p (negation, read: "not p")
- p$\wedge$q (conjunction, read: "p and q")
- p$\vee$q (disjunction, read: "p or q")
- p<=q (implication, read: "p if q")
- p<=>q (equivalence, read: "p if and only if q")

Sentences: built from proposition symbols and connectives

e.g. p<=>(q$\wedge\neg$r)$\vee\neg$q

Example: Buttons and Lights



Proposition Symbols: `initp, initq, truep, trueq, nextp, nextq, doesa, doesb, goal`

The logic of this game:

```
nextp <=> (¬truep ∧ doesa) ∨ (trueq ∧ doesb)
nextq <=> (trueq ∧ doesa) ∨ (truep ∧ doesb)
```

```
¬initp
¬initq
goal <=> (truep ∧ trueq)
```

Propositional Logic: Semantics

Truth table

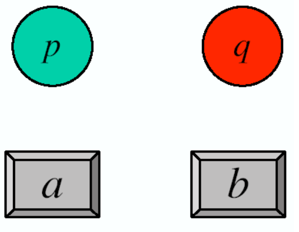
p	q	¬p	p∧q	p∨q	p<=q	p<=>q
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*false*	*false*
true	*false*	*false*	*false*	*true*	*true*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Example

					φ	ψ	nextp
truep	trueq	doesa	doesb	¬truep	¬truep ∧ doesa	trueq ∧ doesb	φ ∨ ψ
false	*false*	*true*	*false*	*true*	*true*	*false*	*true*
true	*false*	*false*	*true*	*false*	*false*	*false*	*false*

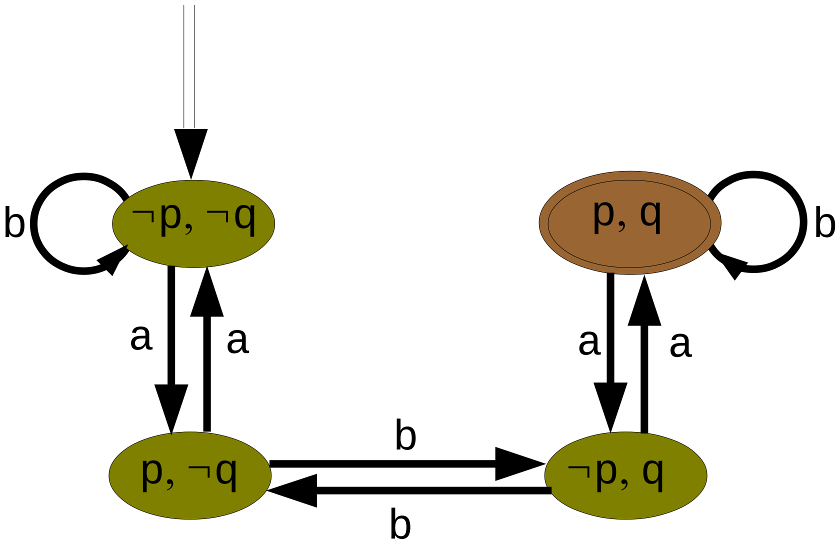
Knowing What Happens in Buttons-and-Lights

```
nextp <=> (¬truep ∧ doesa) ∨ (trueq ∧ doesb)
nextq <=> (trueq ∧ doesa) ∨ (truep ∧ doesb)
```



truep	trueq	doesa	doesb	nextp	nextq
false	*false*	*true*	*false*	*true*	*false*
false	*false*	*false*	*true*	*false*	*false*
true	*false*	*true*	*false*	*false*	*false*
true	*false*	*false*	*true*	*false*	*true*
false	*true*	*true*	*false*	*true*	*true*
false	*true*	*false*	*true*	*true*	*false*
true	*true*	*true*	*false*	*false*	*true*
true	*true*	*false*	*true*	*true*	*true*

Summary: State Transitions for Buttons-and-Lights



More on Semantics: Models

A **model** (in propositional logic) is an arbitrary subset of the proposition symbols.

$\mathcal{M}$ *is a* **model for a sentence** φ under the following conditions:

- $\mathcal{M}$ model for a proposition φ iff φ $\quad \mathcal{M}$
- $\mathcal{M}$ model for $\neg\varphi$ iff $\mathcal{M}$ not a model for φ
- $\mathcal{M}$ model for $\varphi \wedge \psi$ iff $\mathcal{M}$ model for φ and model for ψ
- $\mathcal{M}$ model for $\varphi \vee \psi$ iff $\mathcal{M}$ model for φ or model for ψ (or both)
- $\mathcal{M}$ model for $\varphi <= \psi$ iff $\mathcal{M}$ model for φ whenever $\mathcal{M}$ model for ψ
- $\mathcal{M}$ model for $\varphi <=> \psi$ iff $\mathcal{M}$ model for φ just in case $\mathcal{M}$ model for ψ

If all models of sentences Φ also satisfy φ, then φ is a **logical consequence** of Φ.

First-Order Logic

First-Order Logic: Vocabulary

Object Variables:	`X, Y, Z`
Object Constants:	`a, b, c`
Functions:	`f, g, h`
Predicates:	`p, q, r`
Connectives:	¬, ∧, ∨, <=, <=>
Quantifiers:	∀, ∃

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)`

Example

constant

variable

```
role(xplayer)
role(oplayer)

init(cell(1,1,b))
init(cell(1,2,b))
...
init(cell(3,3,b))
init(control(xplayer))
```

```
P,M,N legal(P,mark(M,N)) <=
          true(cell(M,N,b)) ∧
          true(control(P))

M,N next(cell(M,N,x)) <=
          does(xplayer,mark(M,N))

M,N next(cell(M,N,o)) <=
          does(oplayer,mark(M,N))
```

predicate

function

First-Order Logic: Semantics

The **Herbrand universe** for a logic language is the set of all terms without variables.

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)),...}`

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:

- $\mathcal{M}_1$ = `{p(a), q(a,b), q(b,a)}`
- $\mathcal{M}_2$ = `{p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)}`
- $\mathcal{M}_3$ = { }

Semantics (Finished)

$\mathcal{M}$ *is a* **model for a sentence** φ (in which all variables are bound by a quantifier) under the following conditions:

- $\mathcal{M}$ model for a variable-free atom φ iff φ $\mathcal{M}$
- $\mathcal{M}$ model for ¬φ iff $\mathcal{M}$ not a model for φ
- $\mathcal{M}$ model for φ∧ψ iff $\mathcal{M}$ model for φ and model for ψ
- $\mathcal{M}$ model for φ∨ψ iff $\mathcal{M}$ model for φ or model for ψ (or both)
- $\mathcal{M}$ model for φ <= ψ iff $\mathcal{M}$ model for φ whenever $\mathcal{M}$ model for ψ
- $\mathcal{M}$ model for φ <=> ψ iff $\mathcal{M}$ model for φ just in case $\mathcal{M}$ model for ψ
- $\mathcal{M}$ model for ∀X φ iff $\mathcal{M}$ model for φ{X/t} for all terms t in the Herbrand universe
- $\mathcal{M}$ model for ∃X φ iff $\mathcal{M}$ model for φ{X/t} for some t in the Herbrand universe

φ{X/t} means to replace each occurrence of X by t in φ.

Examples

Recall the models

- $\mathcal{M}_1 = \{$`p(a), q(a,b), q(b,a)`$\}$
- $\mathcal{M}_2 = \{$`p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)`$\}$
- $\mathcal{M}_3 = \{\,\}$

Some examples:

- $\mathcal{M}_1$ is a model for `p(a)`$\land\neg$`p(b)`, whereas $\mathcal{M}_2$ and $\mathcal{M}_3$ are not.
- $\mathcal{M}_2$ is a model for `p(b) <= p(a)`. So is $\mathcal{M}_3$ (!)
- All three are models for $\forall$X$\forall$Y `q(X,Y) <= q(Y,X)`

If all models of sentences Φ also satisfy φ, then φ is a **logical consequence** of Φ.

Logic Programs: A Subset of First-Order Logic

Clauses

- Facts: atoms
- Rules: `Head <= Body`

  `Head`: atom

  `Body`: sentence built from $\wedge$, $\vee$, literal

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

A **logic program** is a finite collection of clauses.

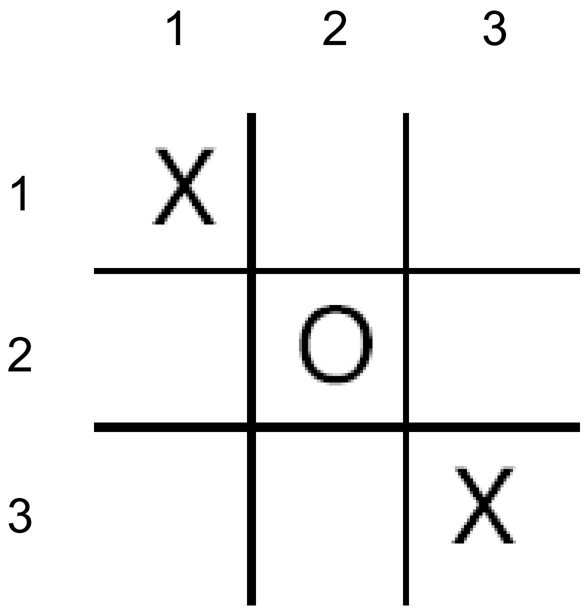
General Game Description Language
GDL

Back to General Game Playing

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: Elements of the Vocabulary

- Object constants

`xplayer, oplayer`	Players
`x, o ,b`	Marks
`noop`	Move

- Functions

`cell(number,number,mark)`	Feature
`control(player)`	Feature
`mark(number,number)`	Move

- Predicates

```
row(number,mark)
column(number,mark)
diagonal(mark)
line(mark)
open
draw
```

Players and Initial State

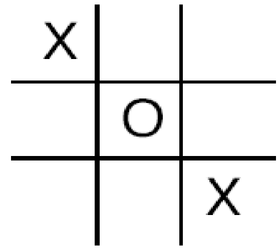
```
role(xplayer)
role(oplayer)

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

```
legal(P,mark(M,N)) <=
    true(cell(M,N,b)) ∧
    true(control(P))

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

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

Conclusions:

```
legal(xplayer,noop)
legal(oplayer,mark(1,2))
...
legal(oplayer,mark(3,2))
```



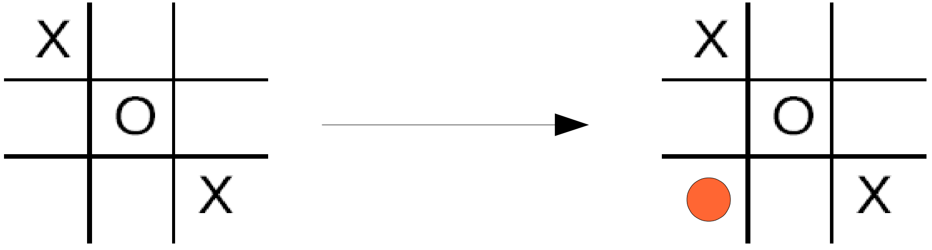

```
true(cell(1,1,x))
true(cell(1,2,b))
true(cell(1,3,b))
true(cell(2,1,b))
true(cell(2,2,o))
true(cell(2,3,b))
true(cell(3,1,b))
true(cell(3,2,b))
true(cell(3,3,x))
true(control(oplayer))
```

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)
control(oplayer)
```

$\xrightarrow[\texttt{mark(1,3)}]{\texttt{oplayer}}$

```
cell(1,1,x)
cell(1,2,b)
cell(1,3,o)
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(xplayer)
```



Physics

```
next(cell(M,N,x)) <= does(xplayer,mark(M,N))

next(cell(M,N,o)) <= does(oplayer,mark(M,N))
```

```
next(cell(M,N,W)) <= true(cell(M,N,W)) ∧
                     does(P,mark(J,K)) ∧
                     (distinct(M,J) ∨ distinct(N,K))
```

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

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

Supporting Concepts

```
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(W) <=
    true(cell(1,3,W)) ∧
    true(cell(2,2,W)) ∧
    true(cell(3,1,W))
```

Termination and Goal Values

```
terminal <=
    line(x) ∨ line(o)
terminal <=
    ←open

line(W) <=
    row(M,W) ∨
    column(N,W) ∨
    diagonal(W)

open <=
    true(cell(M,N,b))
```

```
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)

draw <=
    ¬line(x) ∧ ¬line(o) ∧ ¬open
```

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: `not`, `and`, `or`.
Case-independent. Variables are prefixed with ?.

```
r(X,Y) <= p(X,Y) ∧ ¬q(Y)

    (<= (r ?x ?y) (and (p ?x ?y) (not (q ?y))))
```

or, equivalently,

```
    (<= (r ?x ?y) (p ?x ?y) (not (q ?y)))
```

Semantics is the same.

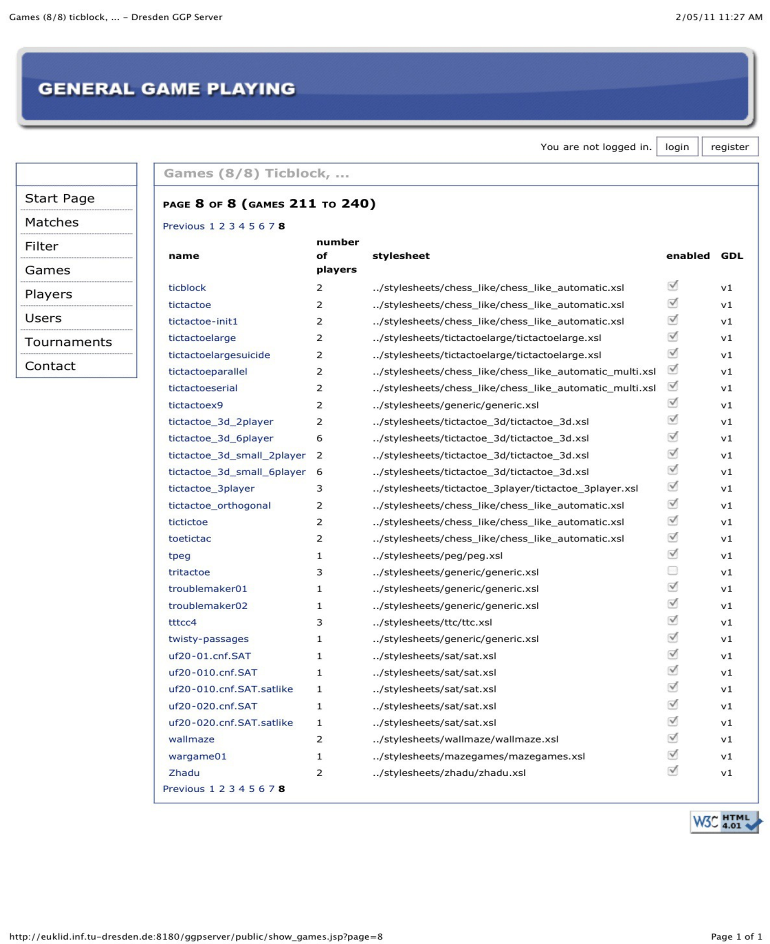
Noughts And Crosses in KIF

```
(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))
(<= (next (cell ?m ?n o))
    (does oplayer (mark ?m ?n))
(<= (next (cell ?m ?n ?w))
    (true (cell ?m ?n ?w))
    (does ?p (mark ?j ?k))
    (or (distinct ?m ?j)
        (distinct ?n ?k)))
(<= (next (control xplayer))
    (true (control oplayer)))
(<= (next (control oplayer))
    (true (control xplayer)))
```

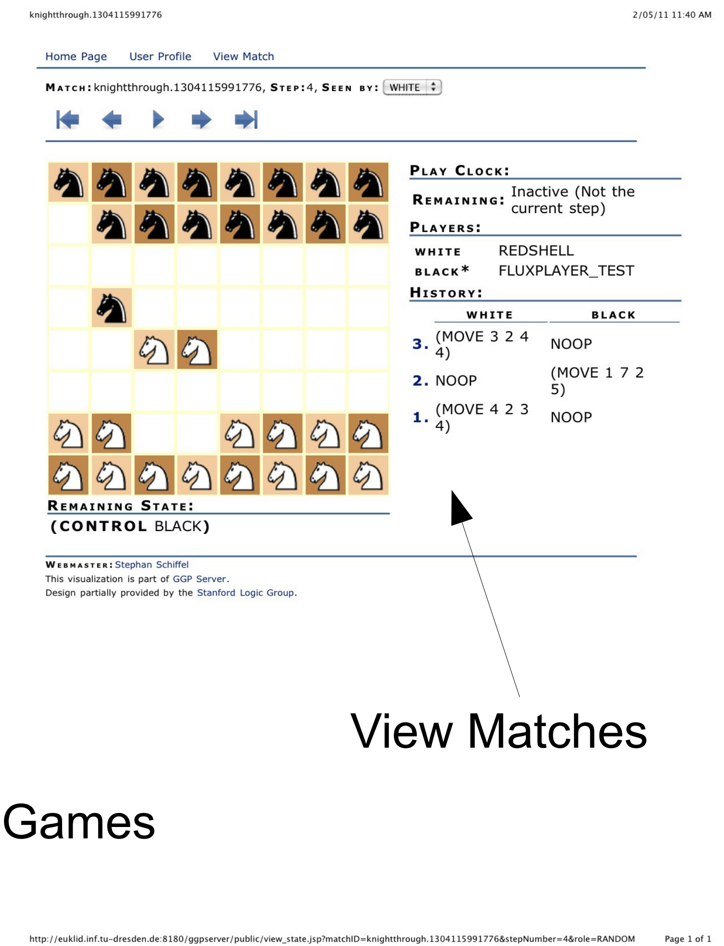
```
(<= (legal ?p (mark ?m ?n))
    (true (cell ?m ?n b))
    (true (control ?p)))
(<= (legal xplayer noop)
    (true (control oplayer)))
(<= (legal oplayer noop)
    (true (control xplayer)))

(<= (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)))
```

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

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

(<= terminal
    (or (line x) (line o)))
(<= terminal
    (not open))

(<= (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))
```

http://130.208.241.192/ggpserver/index.jsp



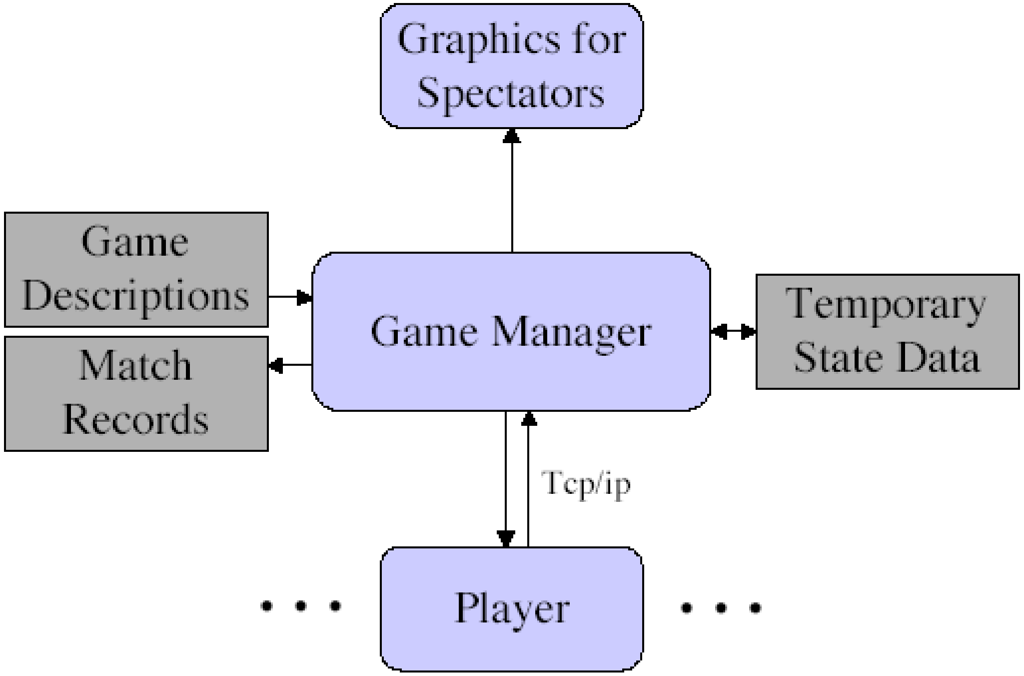
name	number of players	stylesheet	enabled	GDL
ticblock	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
tictactoe	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
tictactoe-init1	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
tictactoelarge	2	../stylesheets/tictactoelarge/tictactoelarge.xsl	✓	v1
tictactoelargesuicide	2	../stylesheets/tictactoelarge/tictactoelarge.xsl	✓	v1
tictactoeparallel	2	../stylesheets/chess_like/chess_like_automatic_multi.xsl	✓	v1
tictactoeserial	2	../stylesheets/chess_like/chess_like_automatic_multi.xsl	✓	v1
tictactoex9	2	../stylesheets/generic/generic.xsl	✓	v1
tictactoe_3d_2player	2	../stylesheets/tictactoe_3d/tictactoe_3d.xsl	✓	v1
tictactoe_3d_6player	6	../stylesheets/tictactoe_3d/tictactoe_3d.xsl	✓	v1
tictactoe_3d_small_2player	2	../stylesheets/tictactoe_3d/tictactoe_3d.xsl	✓	v1
tictactoe_3d_small_6player	6	../stylesheets/tictactoe_3d/tictactoe_3d.xsl	✓	v1
tictactoe_3player	3	../stylesheets/tictactoe_3player/tictactoe_3player.xsl	✓	v1
tictactoe_orthogonal	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
tictictoe	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
toetictac	2	../stylesheets/chess_like/chess_like_automatic.xsl	✓	v1
tpeg	1	../stylesheets/peg/peg.xsl	✓	v1
tritactoe	3	../stylesheets/generic/generic.xsl		v1
troublemaker01	1	../stylesheets/generic/generic.xsl	✓	v1
troublemaker02	1	../stylesheets/generic/generic.xsl	✓	v1
tttcc4	3	../stylesheets/ttc/ttc.xsl	✓	v1
twisty-passages	1	../stylesheets/generic/generic.xsl	✓	v1
uf20-01.cnf.SAT	1	../stylesheets/sat/sat.xsl	✓	v1
uf20-010.cnf.SAT	1	../stylesheets/sat/sat.xsl	✓	v1
uf20-010.cnf.SAT.satlike	1	../stylesheets/sat/sat.xsl	✓	v1
uf20-020.cnf.SAT	1	../stylesheets/sat/sat.xsl	✓	v1
uf20-020.cnf.SAT.satlike	1	../stylesheets/sat/sat.xsl	✓	v1
wallmaze	2	../stylesheets/wallmaze/wallmaze.xsl	✓	v1
wargame01	1	../stylesheets/mazegames/mazegames.xsl	✓	v1
Zhadu	2	../stylesheets/zhadu/zhadu.xsl	✓	v1



View Matches

Games

Game Manager



Communication Protocol

- Manager sends START message to players
  (START <MATCH ID> <ROLE> <GAME DESCRIPTION> <STARTCLOCK> <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

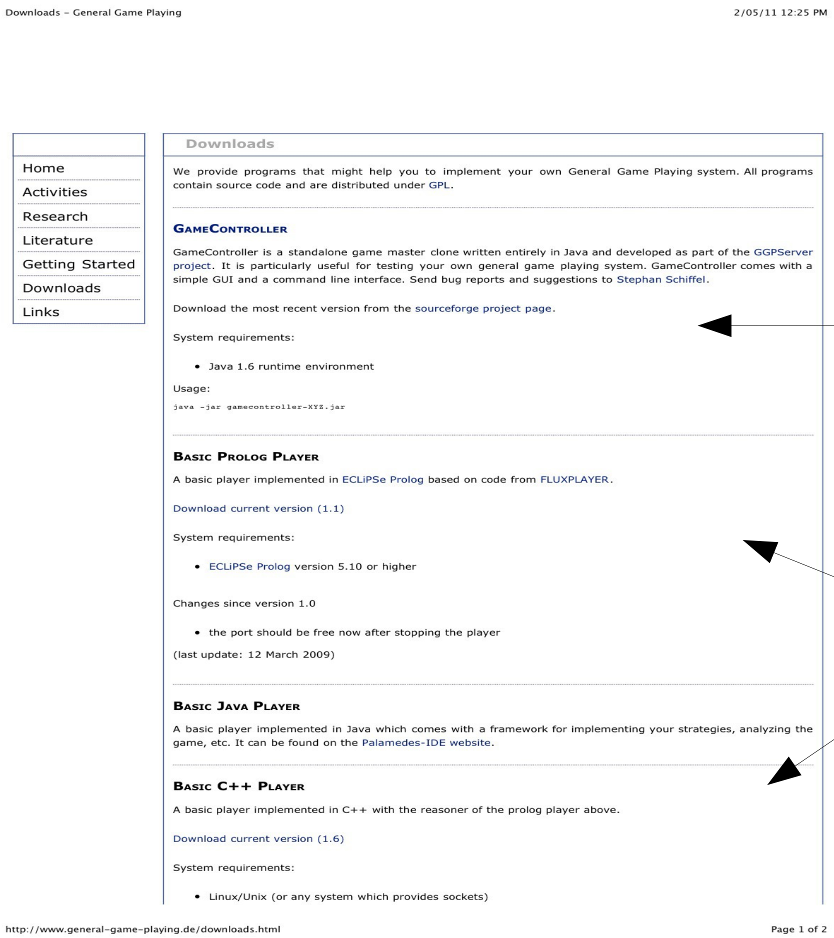
- Manager sends PLAY message to players
  (PLAY <MATCH ID> <PRIOR MOVES>)

  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>)

http://www.general-game-playing.de/downloads.html



Downloads – General Game Playing 2/05/11 12:25 PM

- Home
- Activities
- Research
- Literature
- Getting Started
- Downloads
- Links

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 Schiffel.

Download the most recent version from the sourceforge project page.

System requirements:

- Java 1.6 runtime environment

Usage:

```
java -jar gamecontroller-XYZ.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:

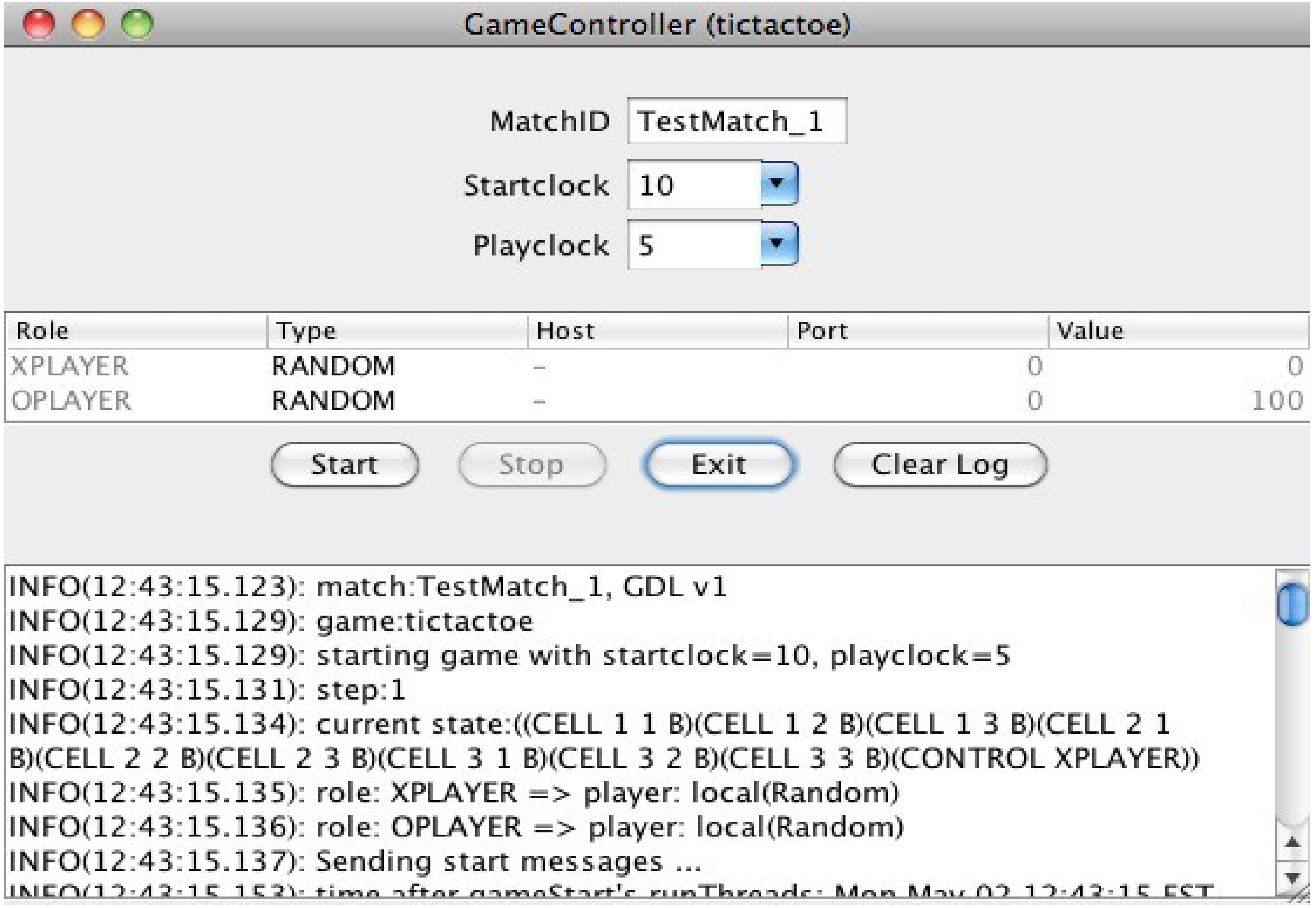
- Linux/Unix (or any system which provides sockets)

http://www.general-game-playing.de/downloads.html Page 1 of 2

Download Manager

Download Basic Players

GameControllerApp



Background Reading

Logic

- Russell & Norvig AIMA: Chapter 7 – Section 7.4
  Chapter 8 – Sections 8.1 and 8.2

General Game Playing

- `games.stanford.edu/competition/misc/aaai.pdf`

New Online Course on General Game Playing

- www.coursera.org/courses/ggp

starts 1 Apr 2013