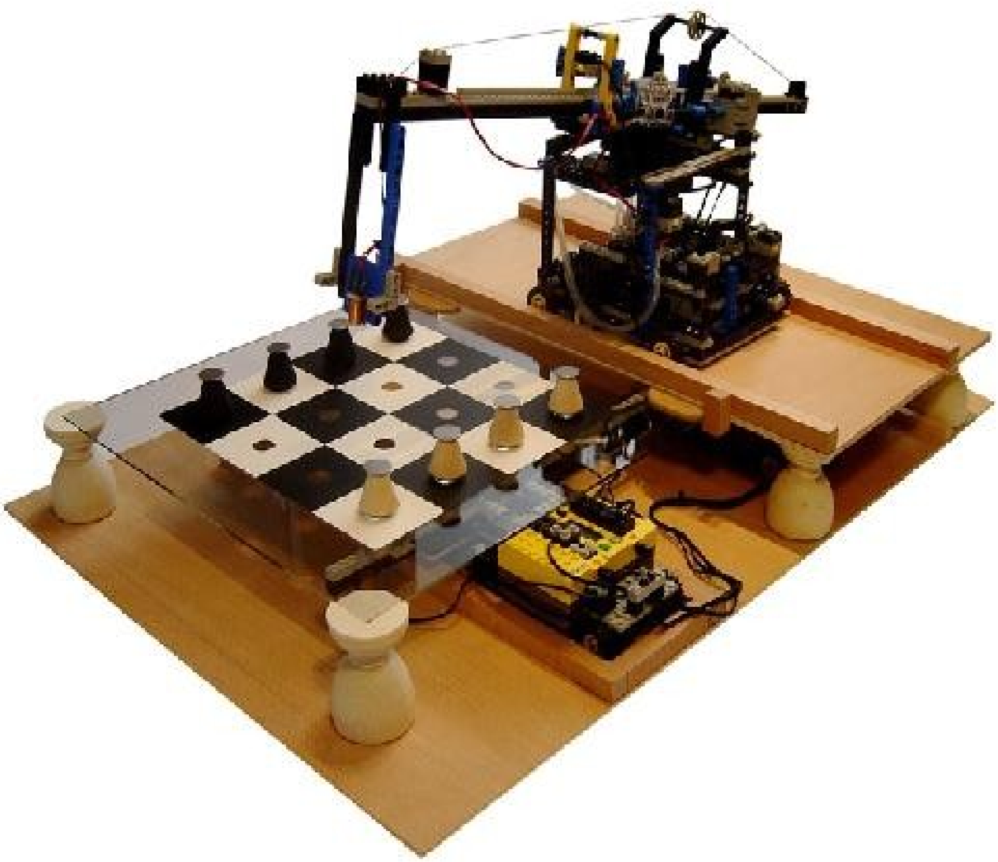
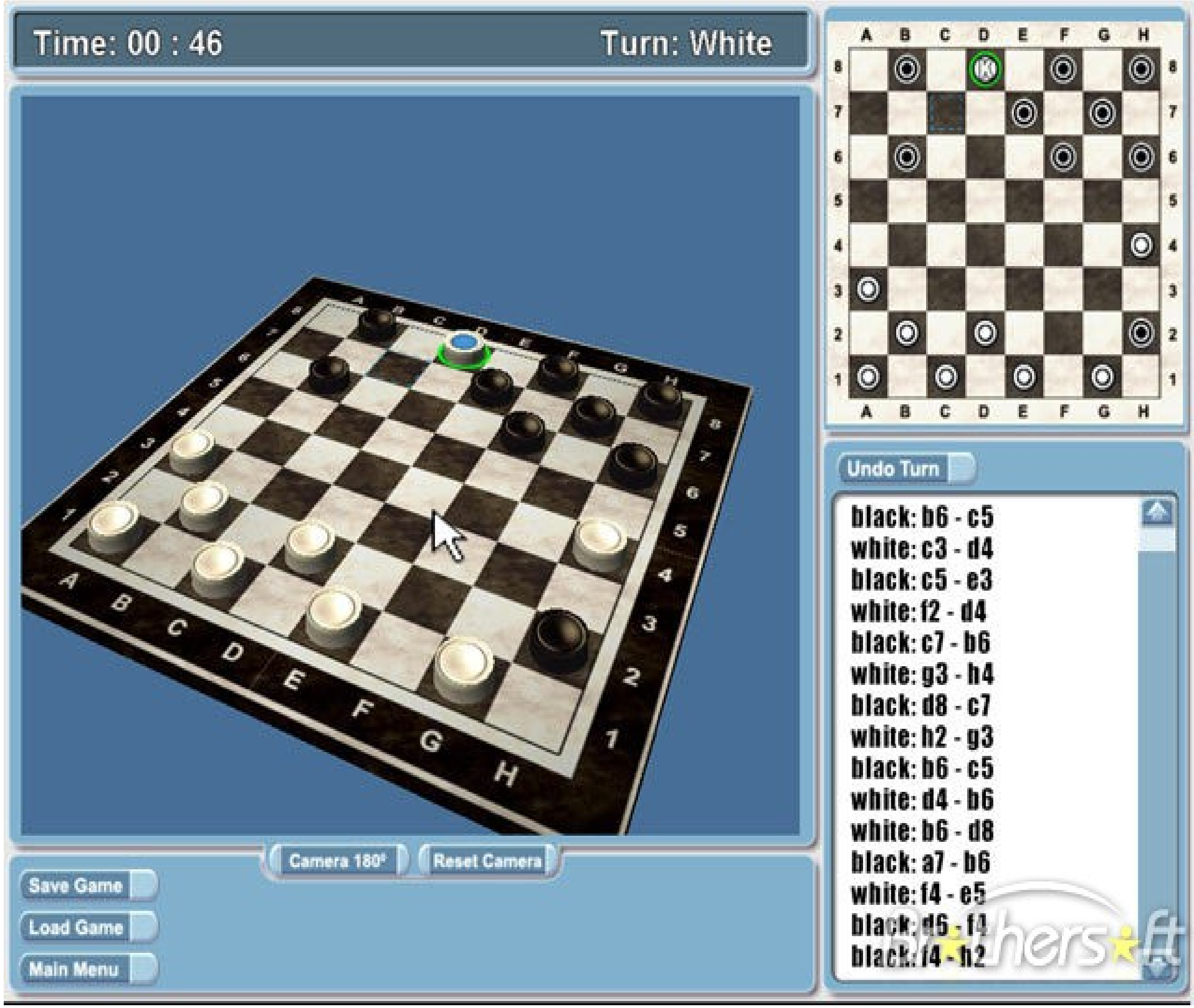
Computer Game Playing

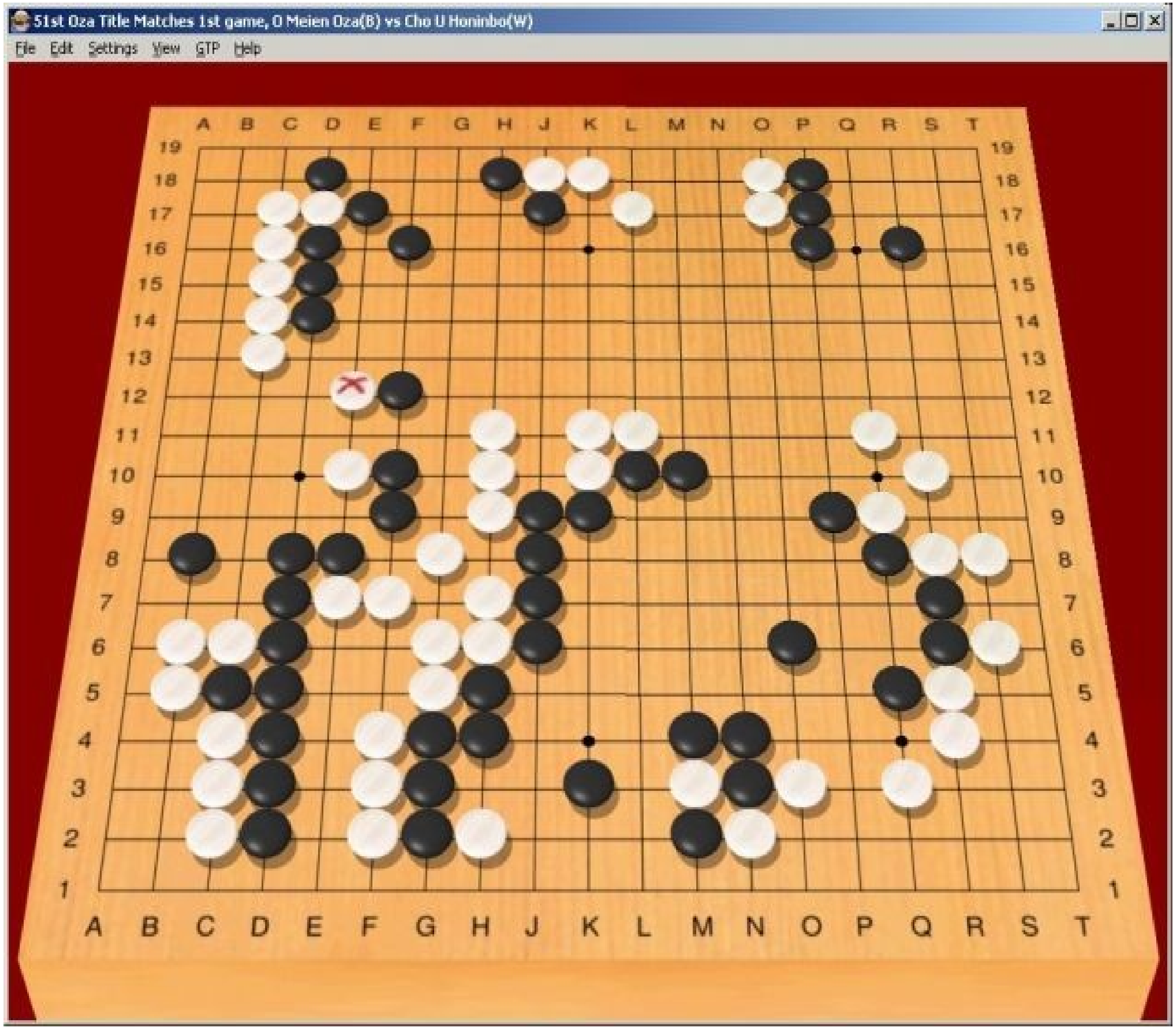


Deep Blue Beats World Champion (1997)

"Game Over" —Checkers Solved in 2007



Computer Go


51st Oza Title Matches 1st game, O Meien Oza(B) vs Cho U Honinbo(W)
File Edit Settings View GTP Help
A B C D E F G H J K L M N O P Q R S T
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

General Game Playing

General Game Players are systems

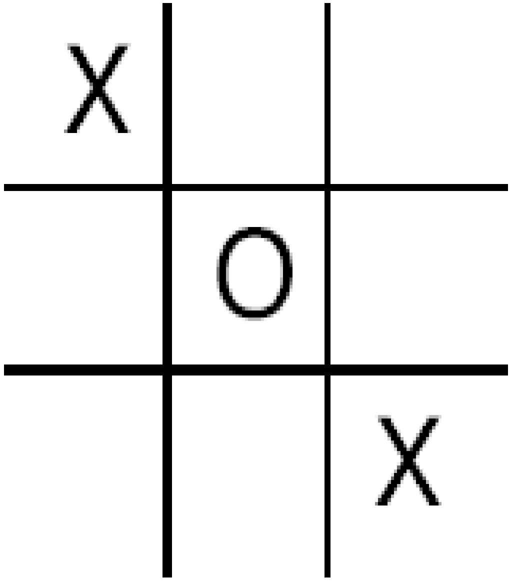
- able to accept a formal description of arbitrary games
- able to use such descriptions to play the games effectively

Cognitive Information Processing Technologies for GGP systems:

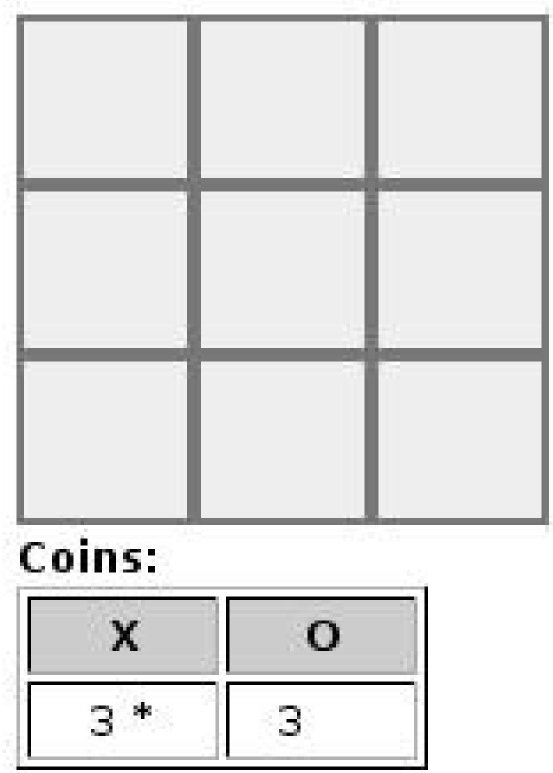
- Knowledge representation
- Reasoning
- Learning
- Rational behaviour

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

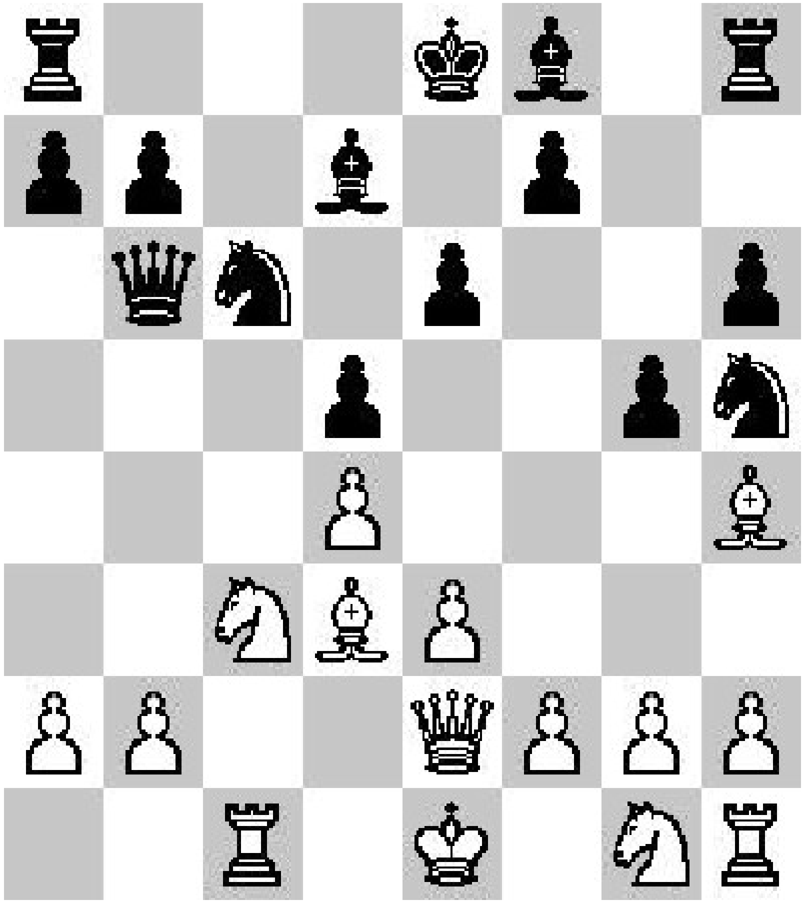
Tic-Tac-Toe



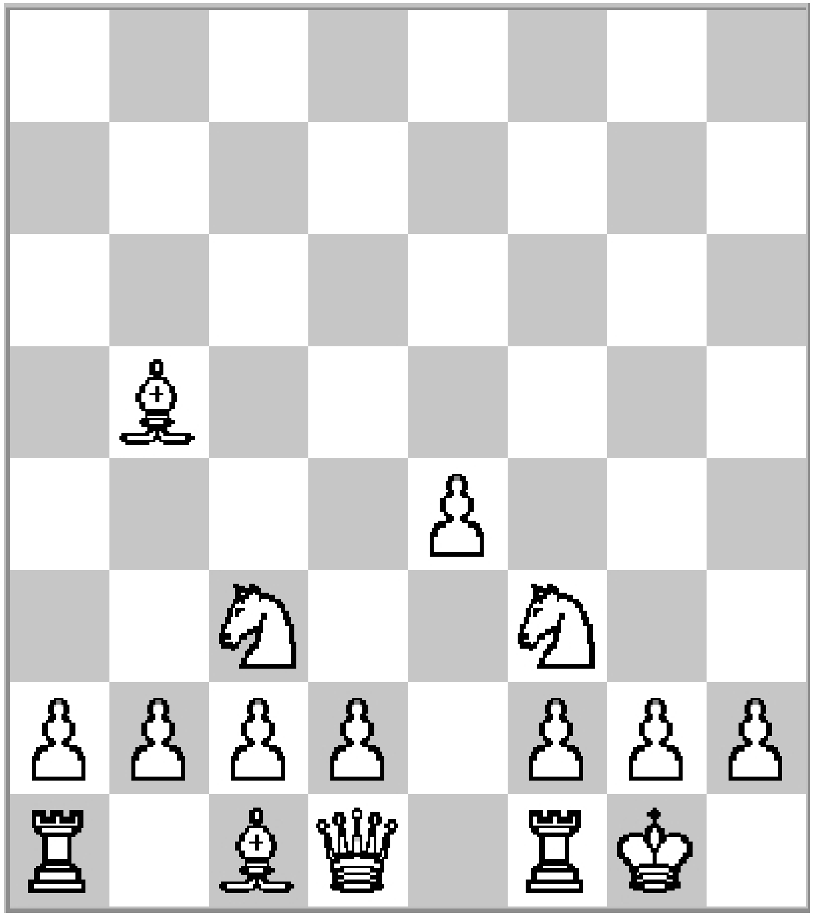
Bidding Tic-Tac-Toe

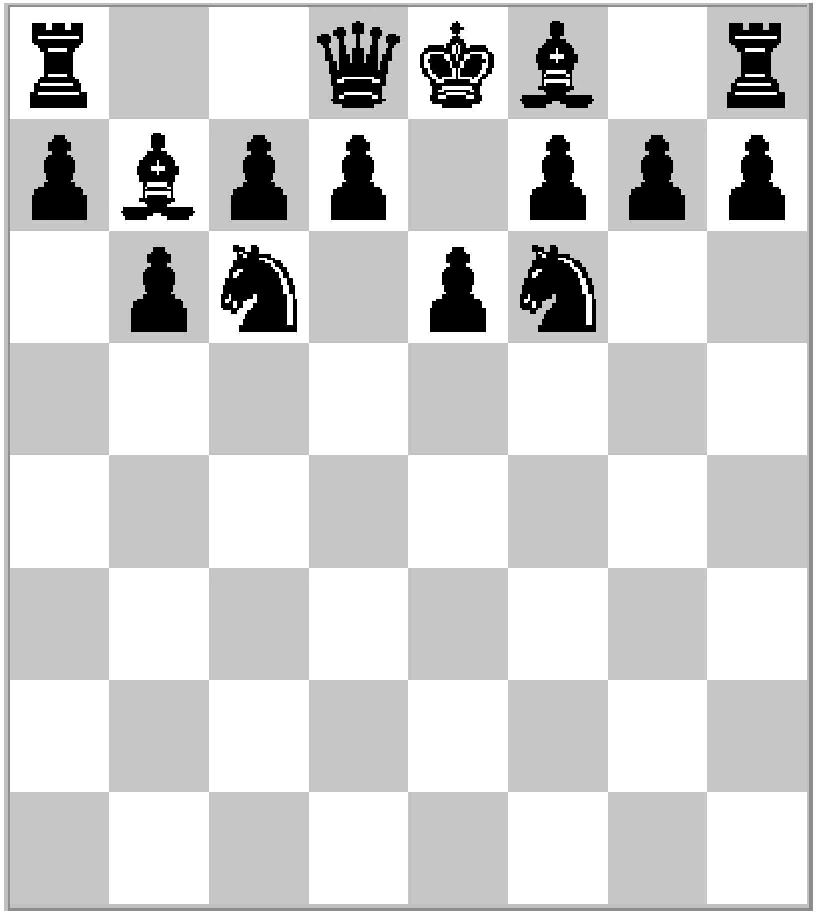


Chess

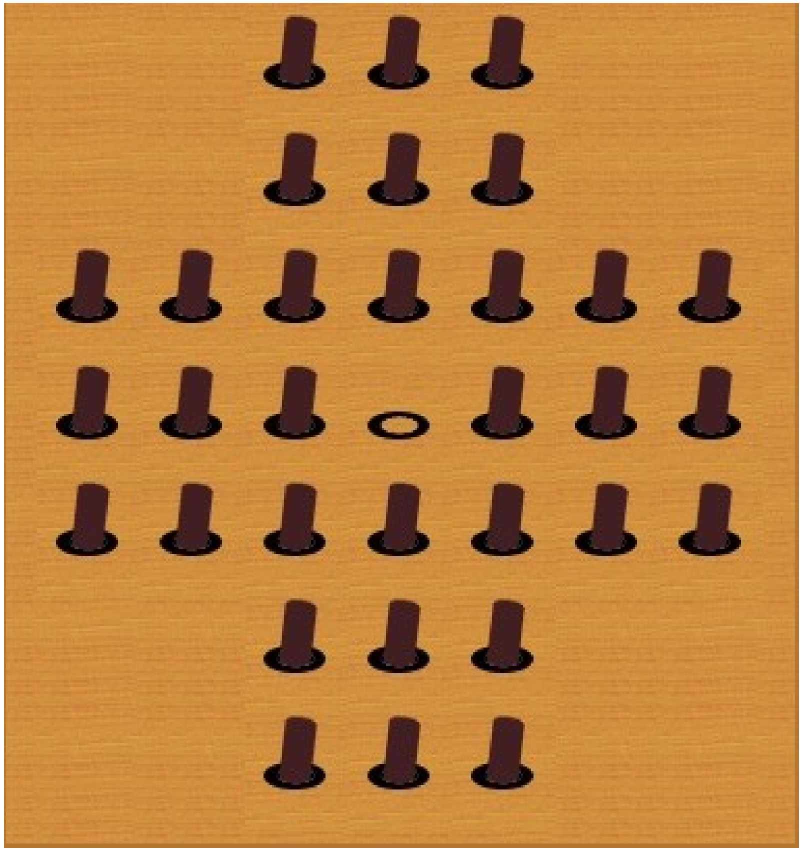


Kriegspiel





Single-Player Games



Monopoly


My
MONOPOLY
TSC STEREOPOLY
PROPERTY TRADING BOARD GAME
COMMUNITY CHEST
CHANCE
FREE PARKING
GO TO JAIL
JUST VISITING
IN JAIL
CHANCE
INCOME TAX
PAY £100
COMMUNITY CHEST
COLLECT £200 SALARY AS YOU PASS
GO
SUPER TAX
PAY £100
ELECTRIC COMPANY
WATER WORKS

Poker


raise $200
call $204
pass
turn
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

Websites – http://games.stanford.edu

www.general-game-playing.de

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

Annual World Cup

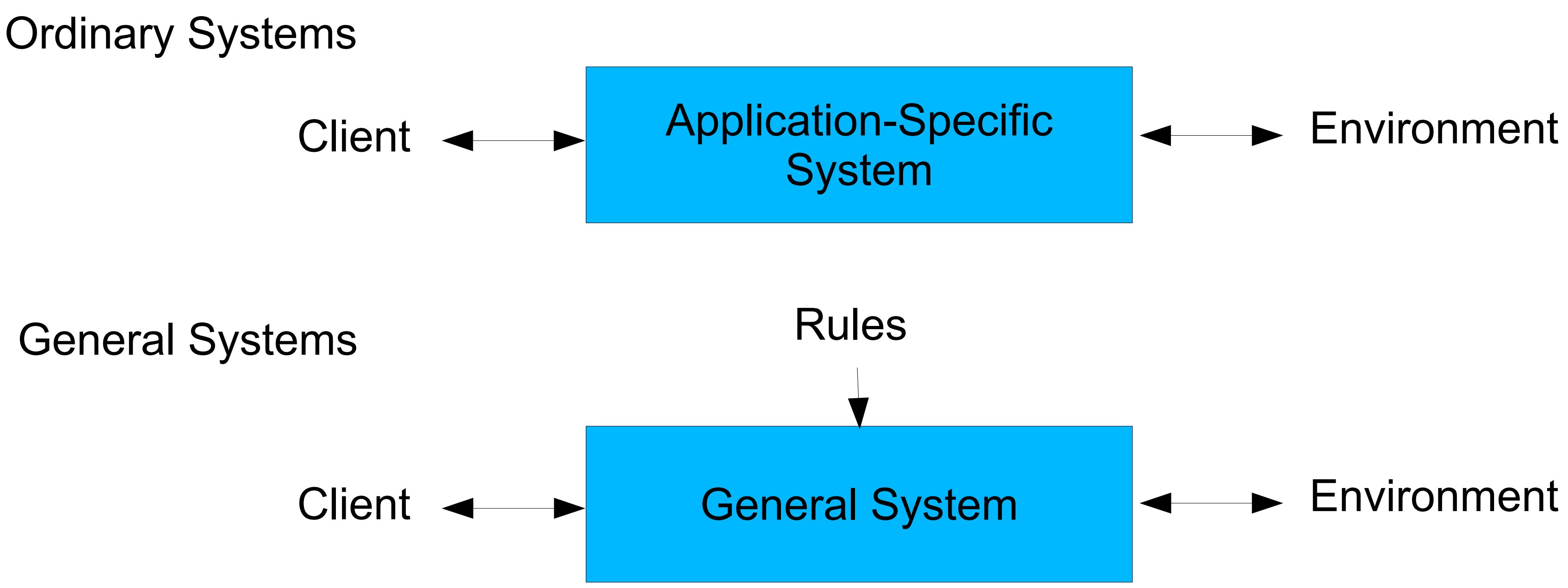
- First: Pittsburgh 2005
- Most recent: Barcelona 2011
- Next: Toronto 2012

German Open, Berlin 2011

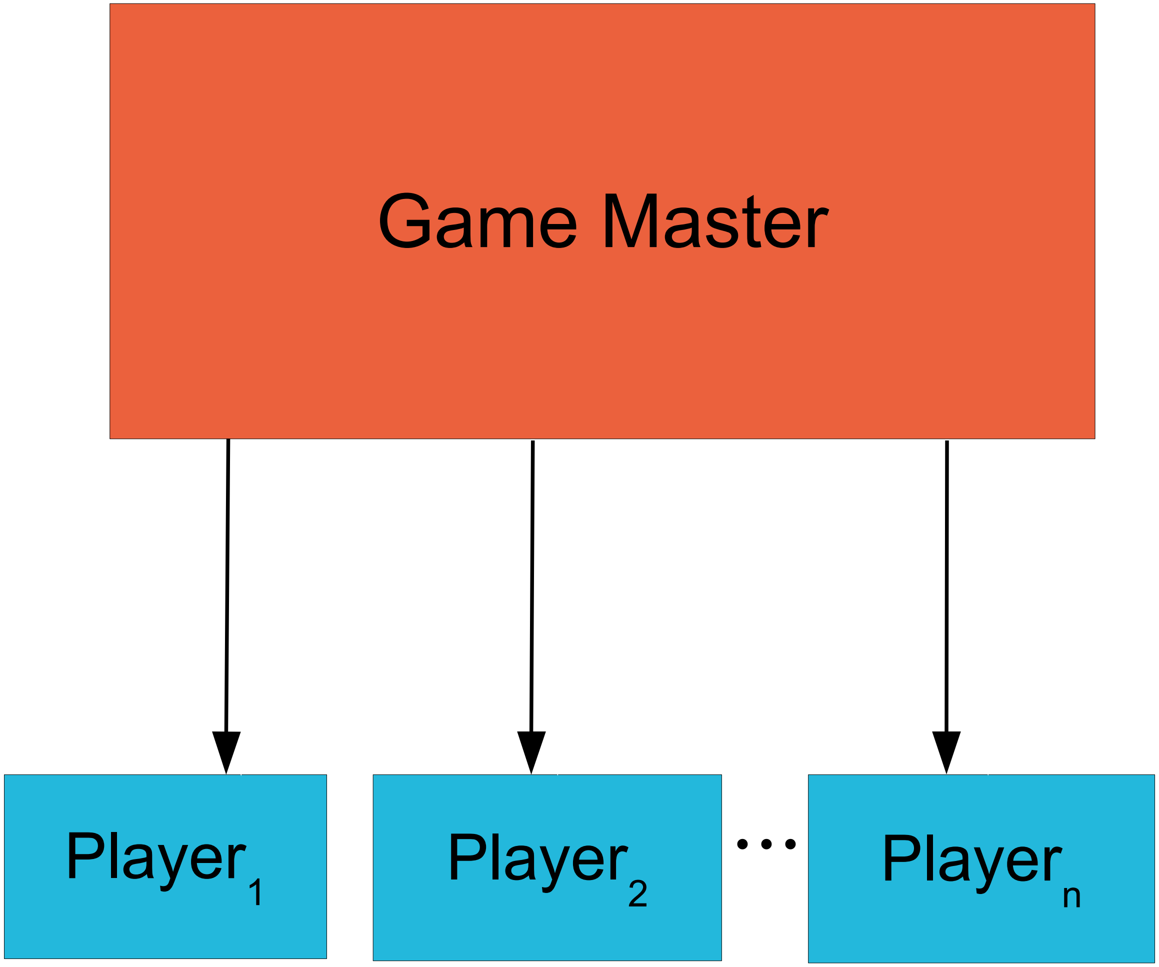
General Game Playing and AI

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.

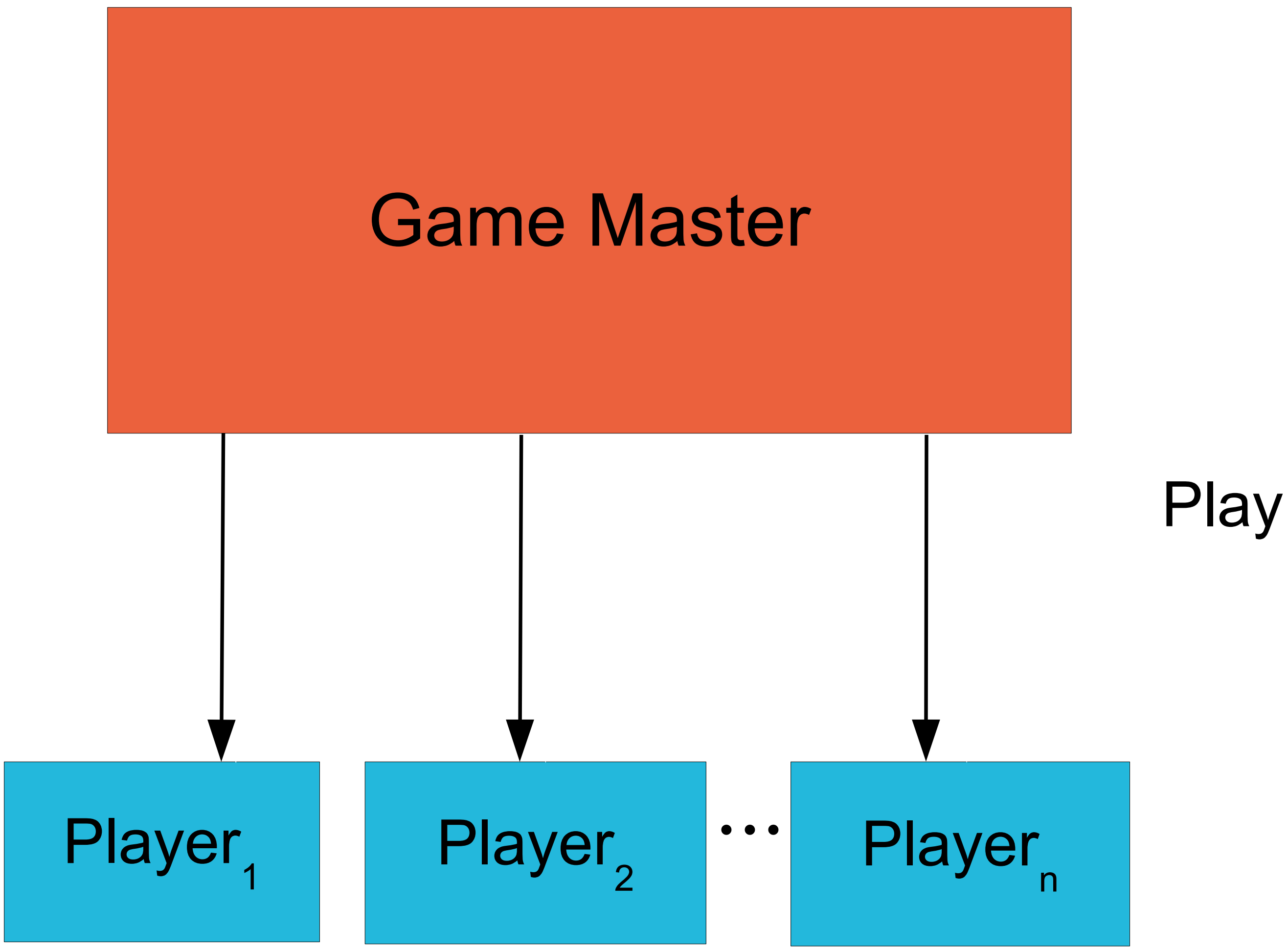


How it Works

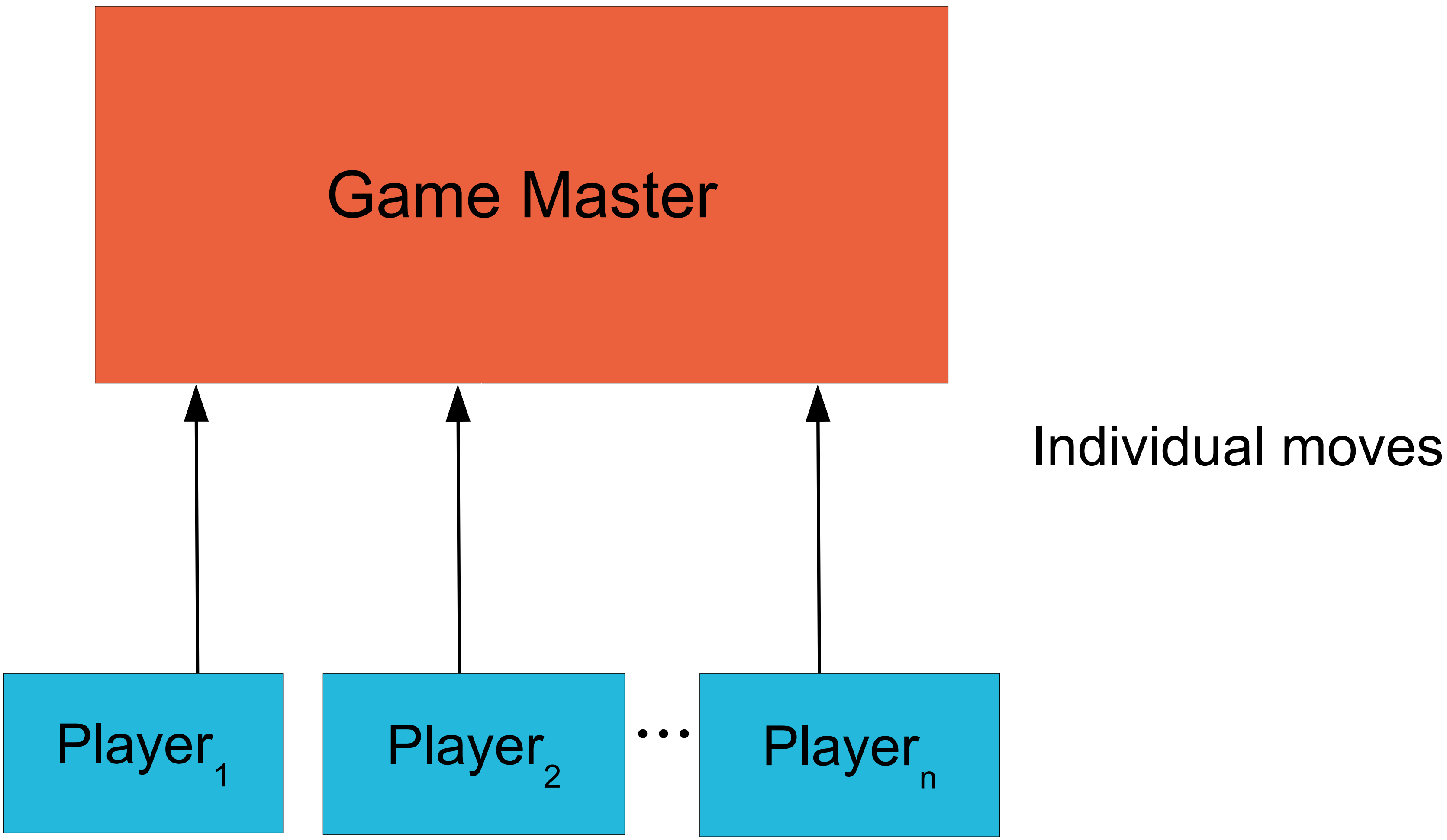


Game description
Time to think: 1,800 sec
Time per move: 45 sec
Your role

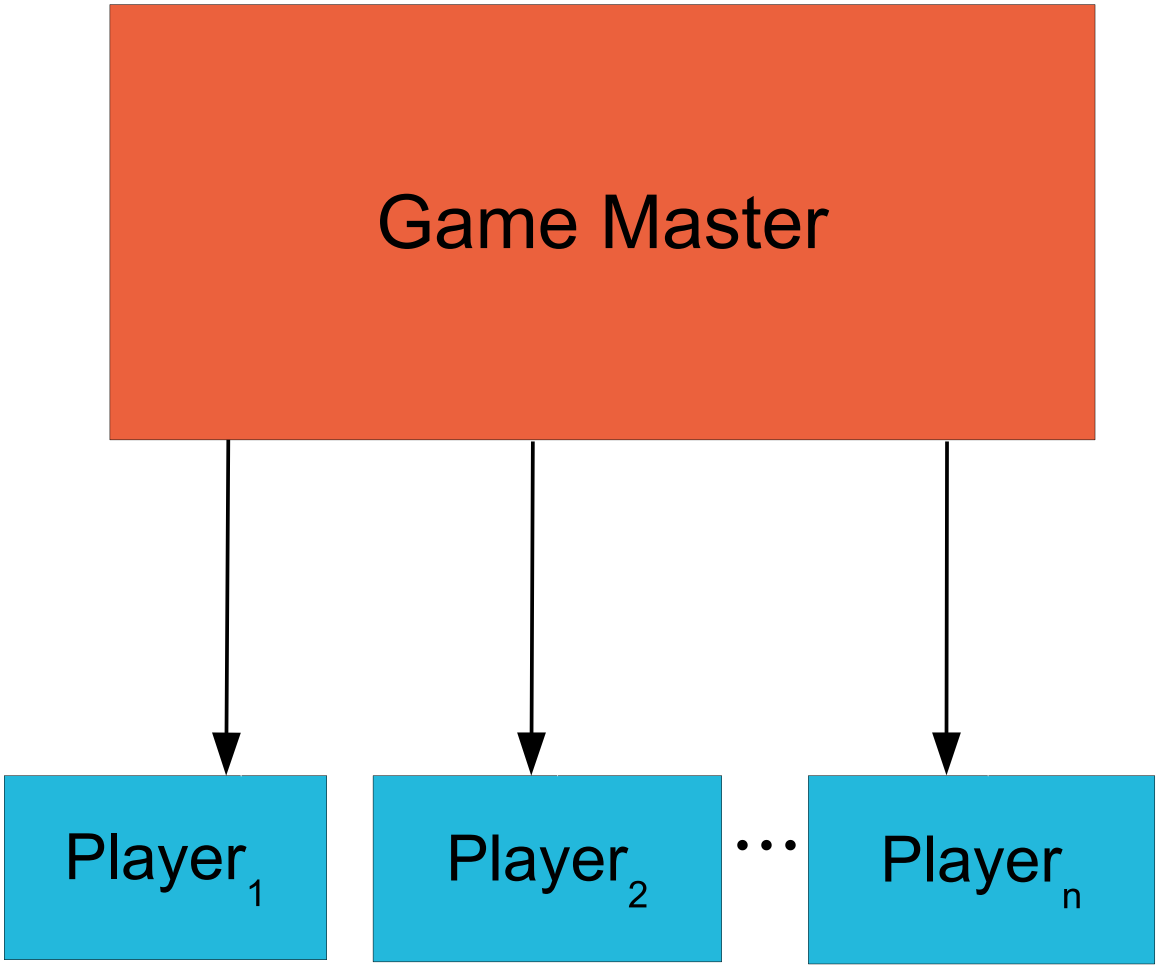
How it Works



How it Works

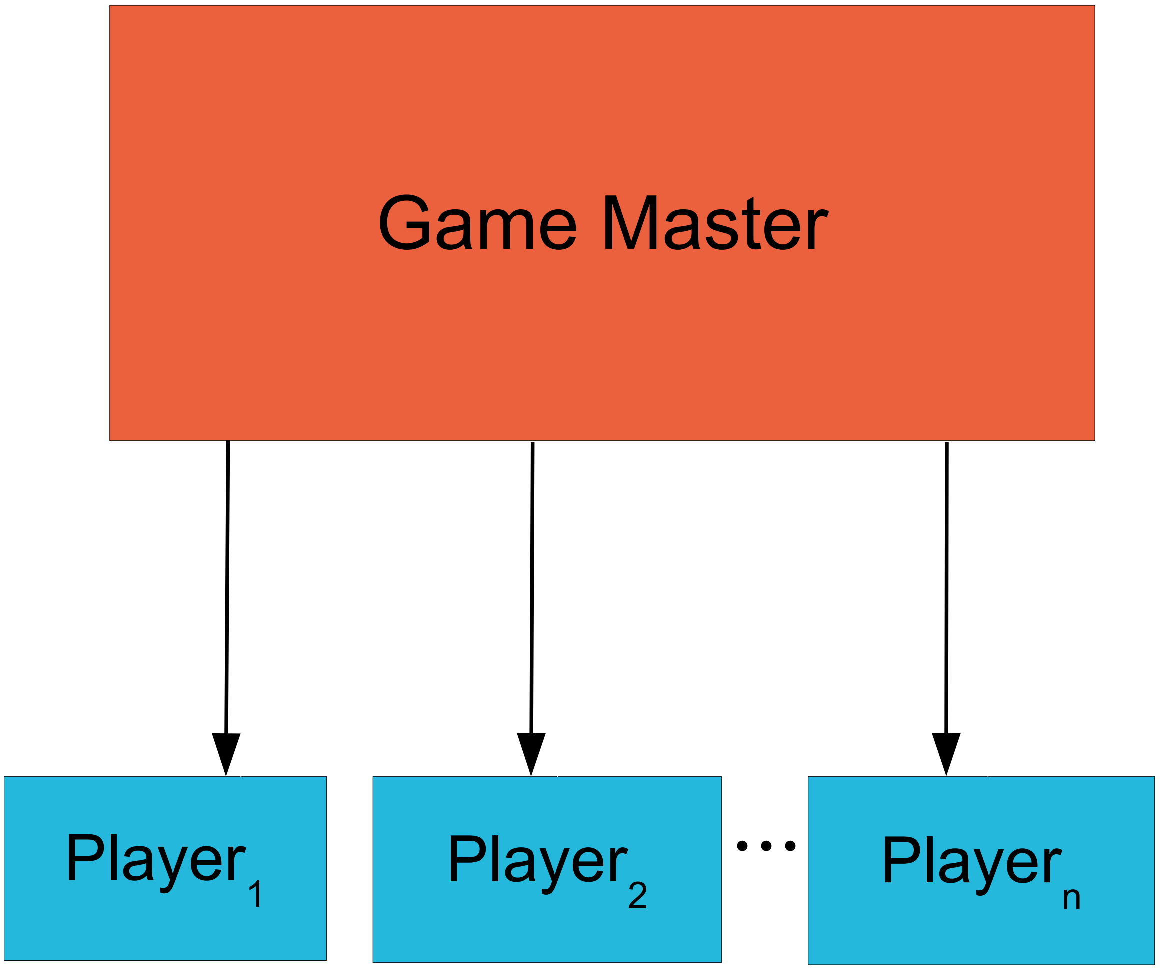

Game Master
Individual moves
Player_1
Player_2
...
Player_n

How it Works



Individual information about state/moves

How it Works

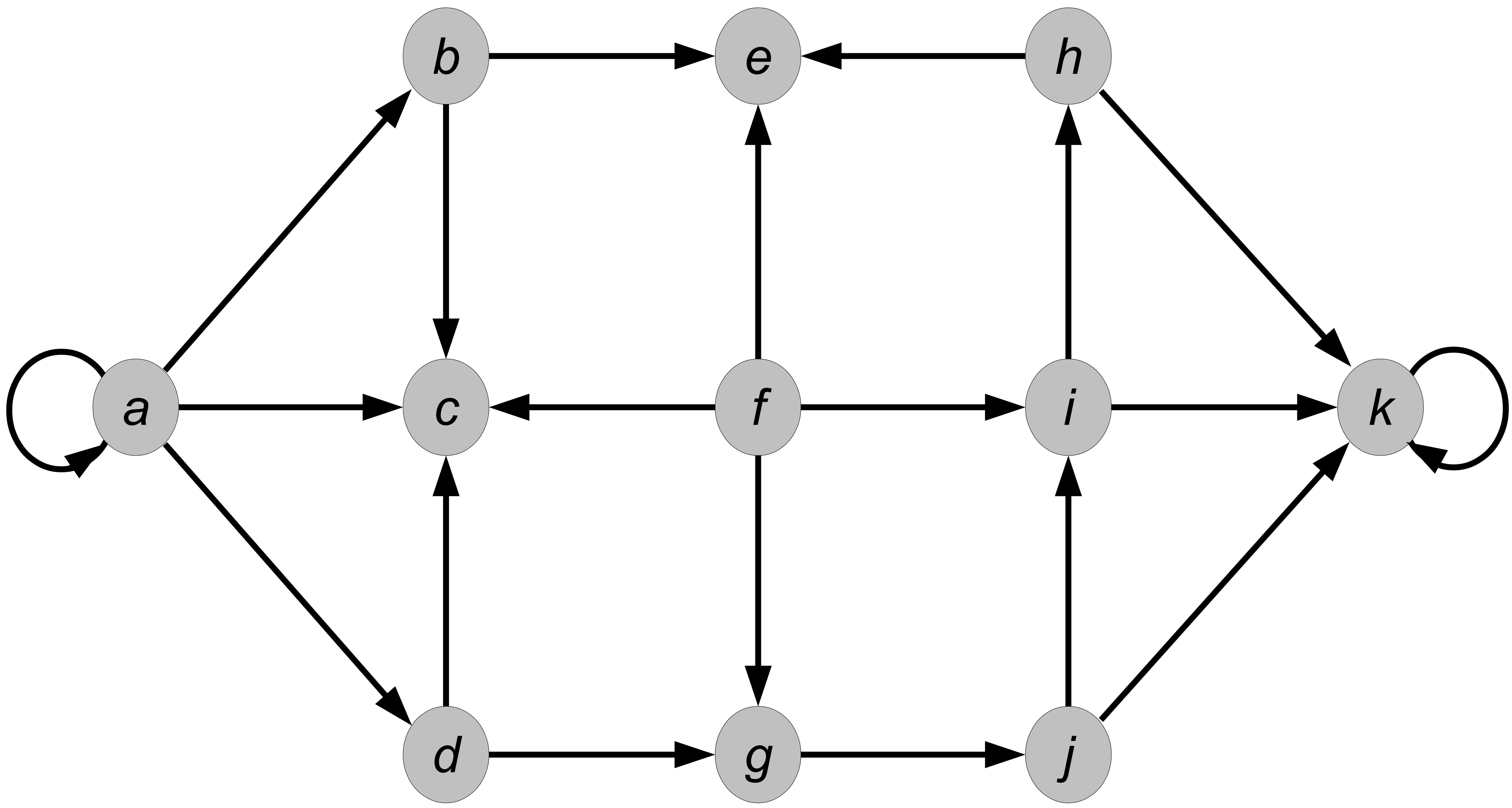


End of game

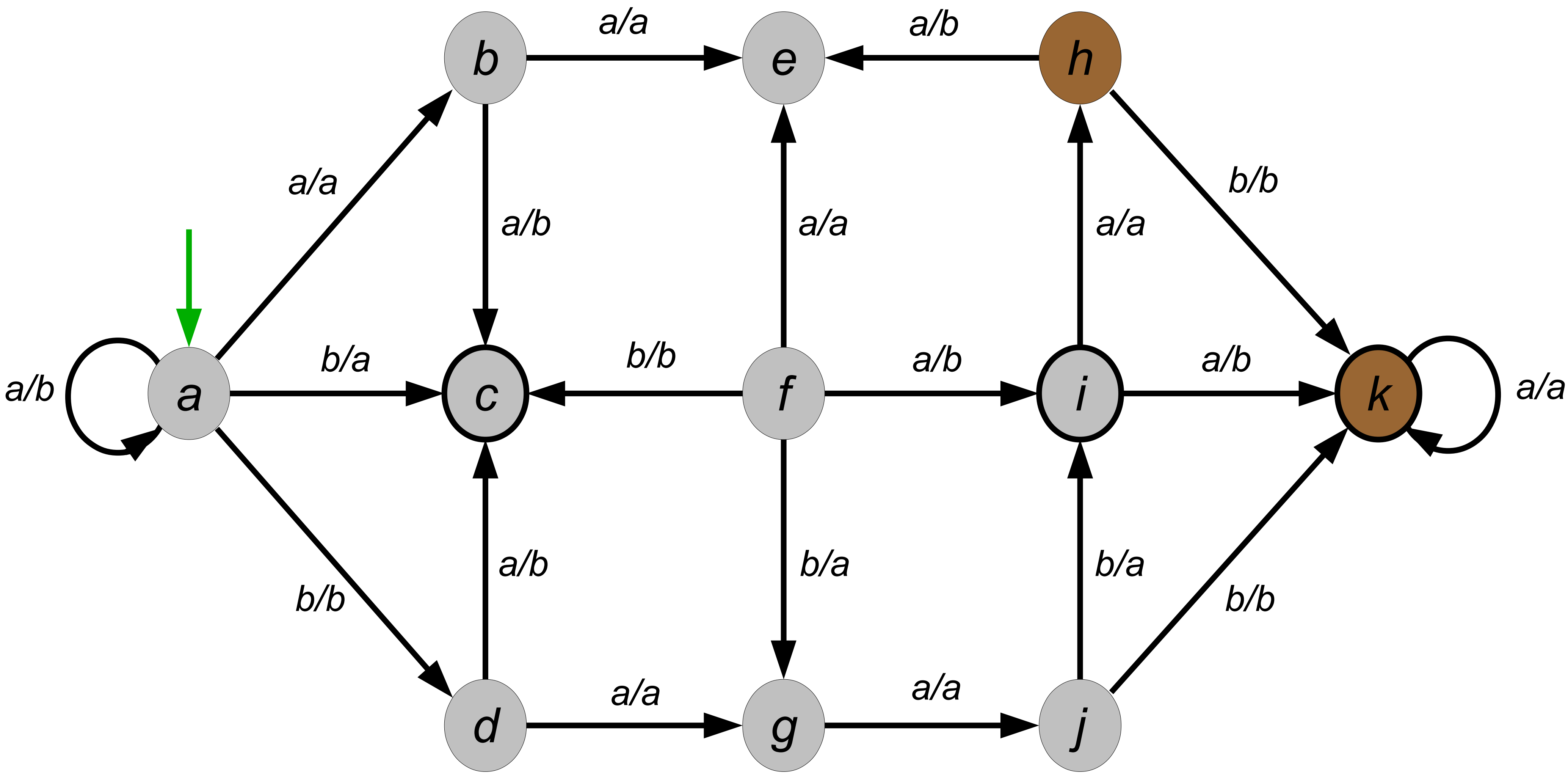
Knowledge Representation for GGP:

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

$u: S \times A_1 \times \ldots \times A_n \rightarrow S$ – 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. ~ 5000 states in Tic-Tac-Toe, ~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.)

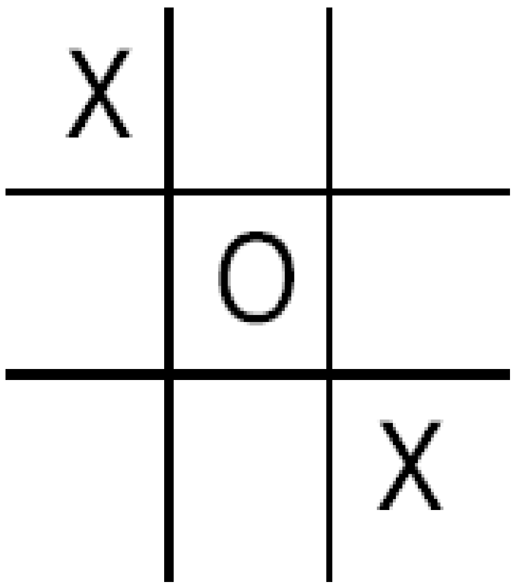
Game Description Language

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.

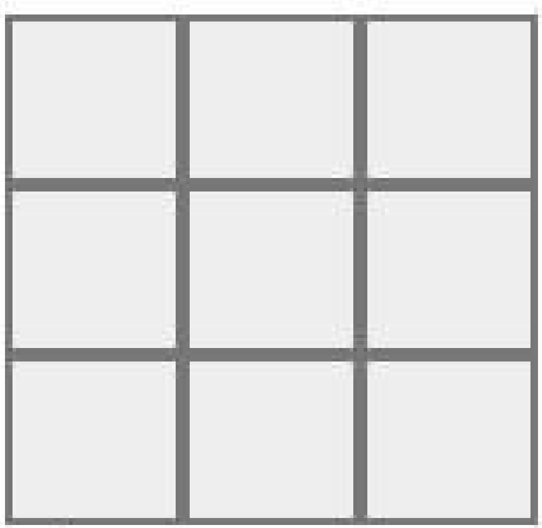
- `role(r)` means that `r` is a role (i.e. a player) in the game
- `init(f)` means that `f` is true in the initial position (state)
- `true(f)` means that `f` is true in the current state
- `does(r,a)` means that role `r` does action `a` in the current state
- `next(f)` means that `f` is true in the 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



Coins:

X	O
3 *	3

```
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

```
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))
```

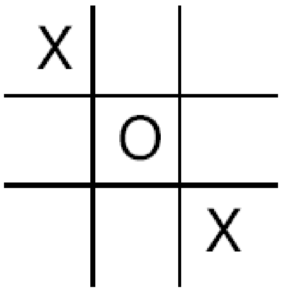
```
init(coins(P,3)) <= role(P)
init(tiebreaker(xplayer))
init(bidding_stage)
```

Move Generator (1)

```
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))
```




```
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)
```

Conclusions:

```
legal(xplayer,noop)

legal(oplayer,mark(1,2))

...

legal(oplayer,mark(3,2))
```

Move Generator (2)

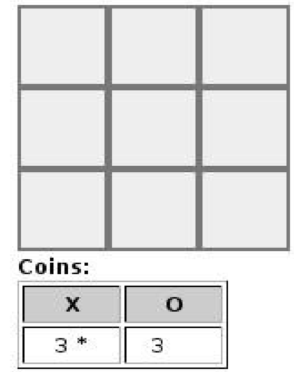
```
legal(P,bid(B,TB)) <=
    true(bidding_stage) ∧
    true(coins(P,C)) ∧
    less_or_eq(B,C) ∧
    tiebreak_bid(P,TB)

tiebreak_bid(P,with_tiebreaker) <=
    true(tiebreaker(P))

tiebreak_bid(P,no_tiebreaker)   <=
    role(P)
```

Conclusions:

```
legal(xplayer,bid(1,with_tiebreaker))
legal(xplayer,bid(1,no_tiebreaker))
...
legal(oplayer,bid(1,no_tiebreaker)
...
```



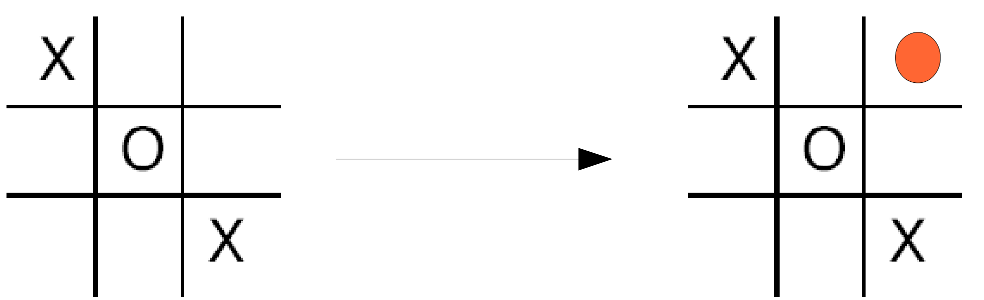

```
cell(1,1,b)
cell(1,2,b)
cell(1,3,b)
cell(2,1,b)
cell(2,2,b)
cell(2,3,b)
cell(3,1,b)
cell(3,2,b)
cell(3,3,b)
coins(xplayer,3)
coins(oplayer,3)
tiebreaker(xplayer)
bidding_stage
```

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)
```

```
oplayer
mark(1,3)
```

```
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)
coins(xplayer,4)
coins(oplayer,2)
tiebreaker(oplayer)
bidding_stage
```



Physics (1)

```
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(coins(P,C)) <= true(control(Q)) ∧ true(coins(P,C))

next(tiebreaker(P)) <= true(control(Q)) ∧ true(tiebreaker(P))
```

```
next(bidding_stage) <= true(control(P))
```

Physics (2)

```
next(coins(P,C)) <= true(bidding_stage) ∧
                    winner(P) ∧
                    does(P,bid(B,TB)) ∧
                    true(coins(P,C1)) ∧ add(C,B,C1)

next(coins(xplayer,C)) <= true(bidding_stage) ∧
                          winner(oplayer) ∧
                          does(oplayer,bid(B,TB)) ∧
                          true(coins(P,C1)) ∧ add(C1,B,C)

next(coins(oplayer,C)) <= true(bidding_stage) ∧
                          winner(xplayer) ∧
                          does(xplayer,bid(B,TB)) ∧
                          true(coins(P,C1)) ∧ add(C1,B,C)
```

Physics (3)

```
next(tiebreaker(P)) <= true(bidding_stage) ∧ role(P) ∧
                       winner(Q) ∧ distinct(P,Q) ∧
                       does(Q,bid(B,with_tiebreaker))

next(tiebreaker(P)) <= true(bidding_stage) ∧
                       true(tiebreaker(P)) ∧
                       winner(Q) ∧
                       does(Q,bid(B,no_tiebreaker))
```

```
next(cell(M,N,W)) <= true(cell(M,N,W)) ∧ true(bidding_stage)

next(control(P)) <= winner(P) ∧ true(bidding_stage)
```

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
```

Supporting Concepts

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

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

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

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

More Supporting Concepts

```
winner(P) <= does(P,bid(B1,TB1)) ∧ does(Q,bid(B2,TB2)) ∧
             distinct(P,Q) ∧ greater(B1,B2)

winner(P) <= does(P,bid(B,with_tiebreaker)) ∧
             does(Q,bid(B,no_tiebreaker))

winner(P) <= does(P,bid(B,no_tiebreaker)) ∧
             does(Q,bid(B,no_tiebreaker)) ∧
             distinct(P,Q) ∧
             true(tiebreaker(Q))


succ(0,1)  succ(1,2)  succ(2,3)  ...

greater(X,Y) <= ...

less_or_equal(X,Y) <= ...

add(X,Y,Z) <= ...
```

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 <u>safe</u> if and only if every variable in the clause appears in some positive subgoal in the body.

Safe Rule:

```
r(X,Y) <= p(X,Y) ∧ ¬q(X,Y)
```

Unsafe Rule:

```
r(X,Z) <= p(X,Y) ∧ ¬q(Y,Z)
```

In GDL, all rules are required to be safe.

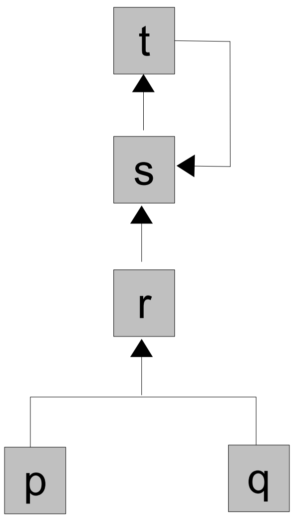
(Note that this implies all facts to be variable-free.)

Dependency Graph

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.

```
r(X,Y) <= p(X,Y) ∧ q(X,Y)
s(X,Y) <= r(X,Y)
s(X,Z) <= r(X,Y) ∧ t(Y,Z)
t(X,Z) <= s(X,Y) ∧ s(Y,X)
```



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:

```
t(X,Y) <= q(X,Y) ∧ ¬r(X,Y)
r(X,Z) <= p(X,Y)
r(X,Z) <= r(X,Y) ∧ r(Y,Z)
```

Negation that is not stratified:

```
r(X,Z) <= p(X,Y)
r(X,Z) <= p(X,Y) ∧ ¬r(Y,Z)
```

In GDL, all sets of rules are required to be stratified.

Guaranteeing Decidability (3)

If a set of rules contains a clause

$$p(s_1, ..., s_m) <= b_1(\vec{t_1}) \wedge ... \wedge q(v_1, ..., v_k) \wedge ... \wedge b_n(\vec{t_n})$$

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 $\vec{t_j}$ 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 `?`.

$$r(X,Y) \Leftarrow p(X,Y) \wedge \neg q(Y)$$

```
(<= (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.

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

Playing Games

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.

Download GameController

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)

Download Basic Players

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

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

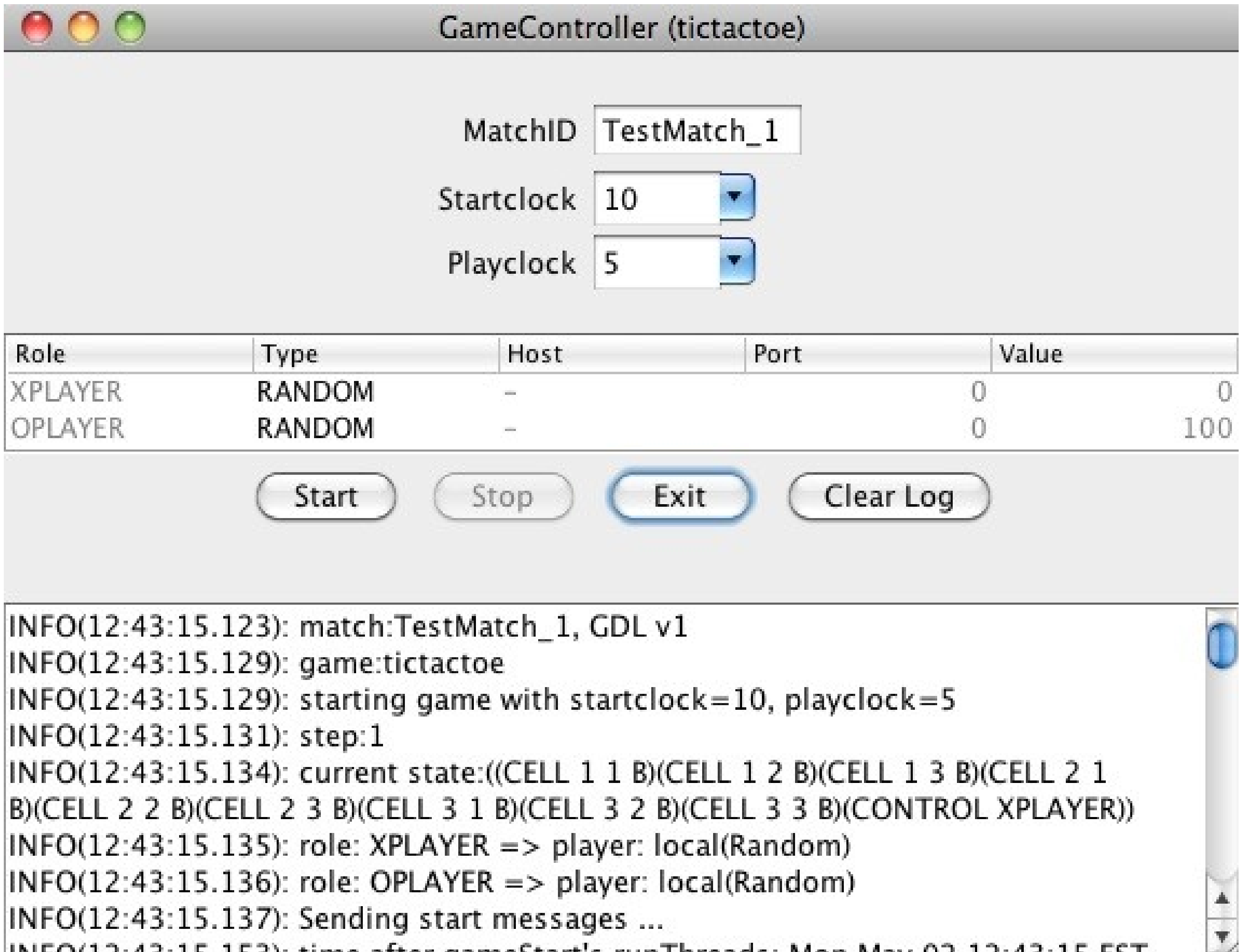
- Manager sends PLAY message to players
  `(PLAY <MATCH ID> <PRIOR MOVES>)`

  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>)`

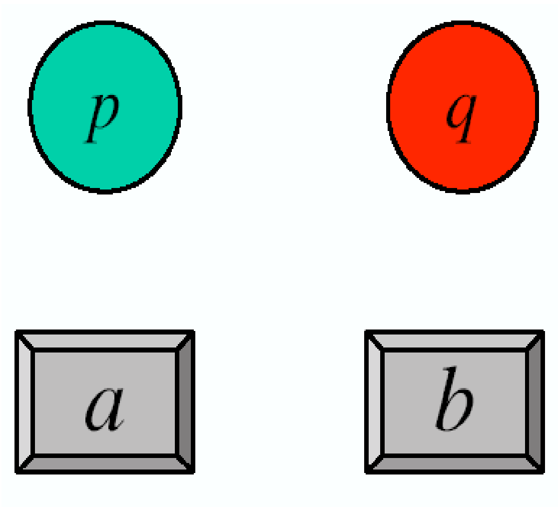
GameControllerApp



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

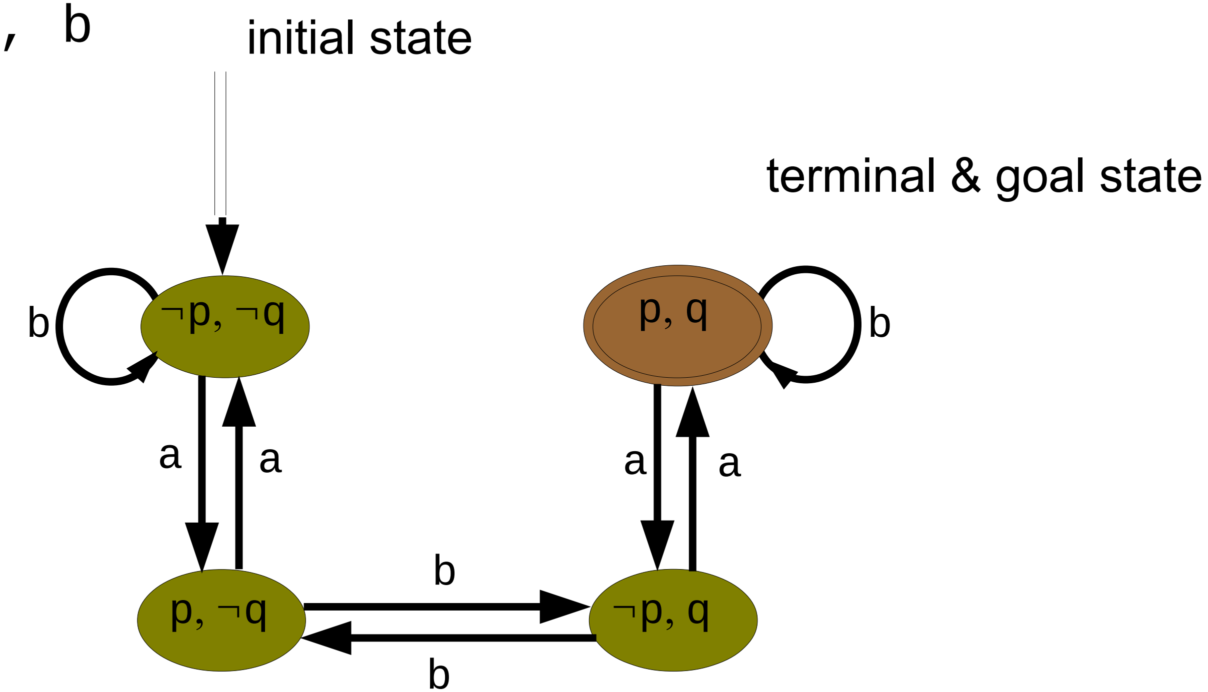
- Goal

  $p \wedge q$

State Transition System

State features: p, q

Actions: a, b



Solution (= Plan): a, b, a

Single-Player, Perfect-Information Games

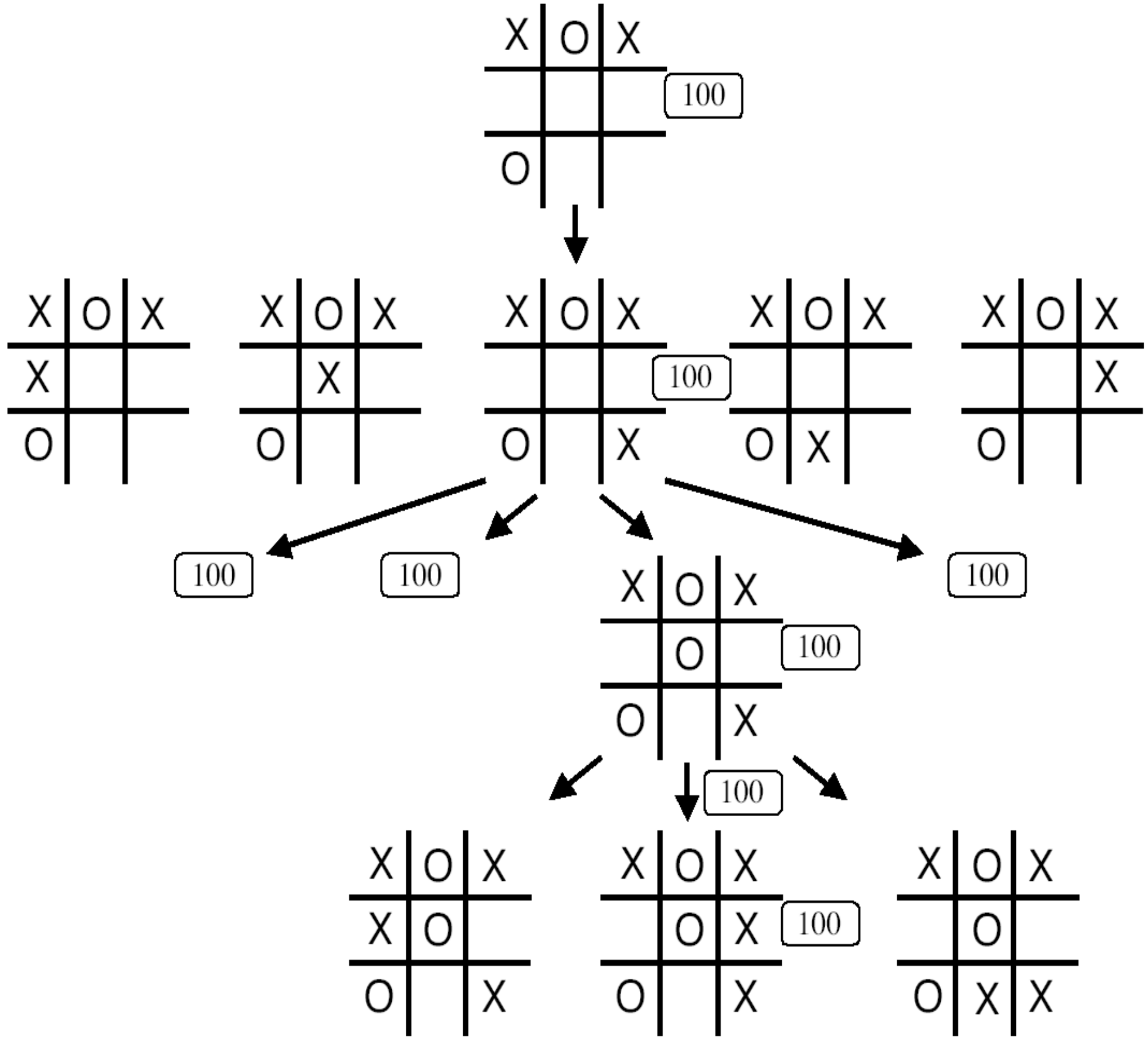
Many single-player games can be solved using standard search techniques

- Iterative deepening
- Bidirectional search

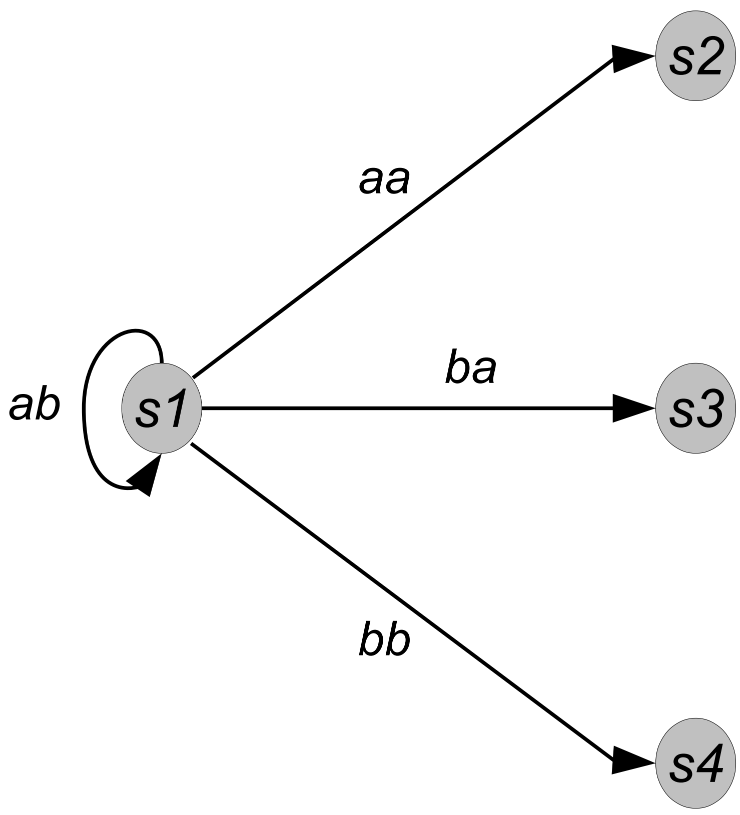
Special techniques

- Constraint solving (suitable for Sudoku, Gene Sequencing and the like)

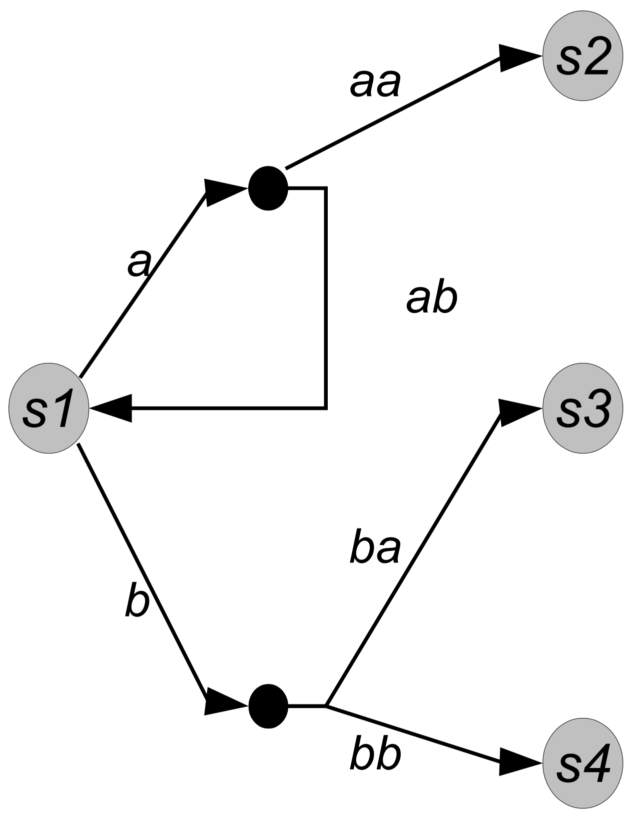
Multi-Player Games: Game Tree Search



How to Deal With Simultaneous Moves

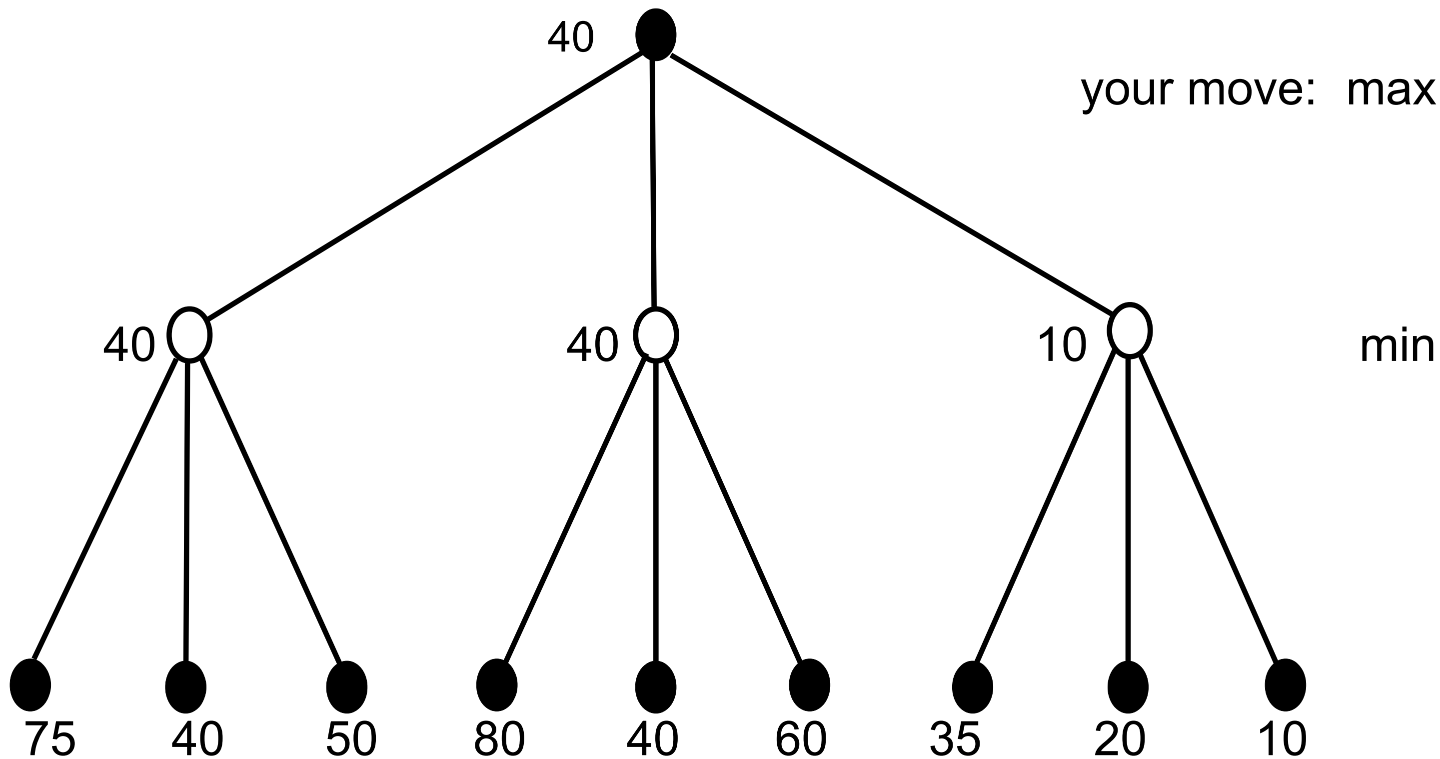


State transition graph

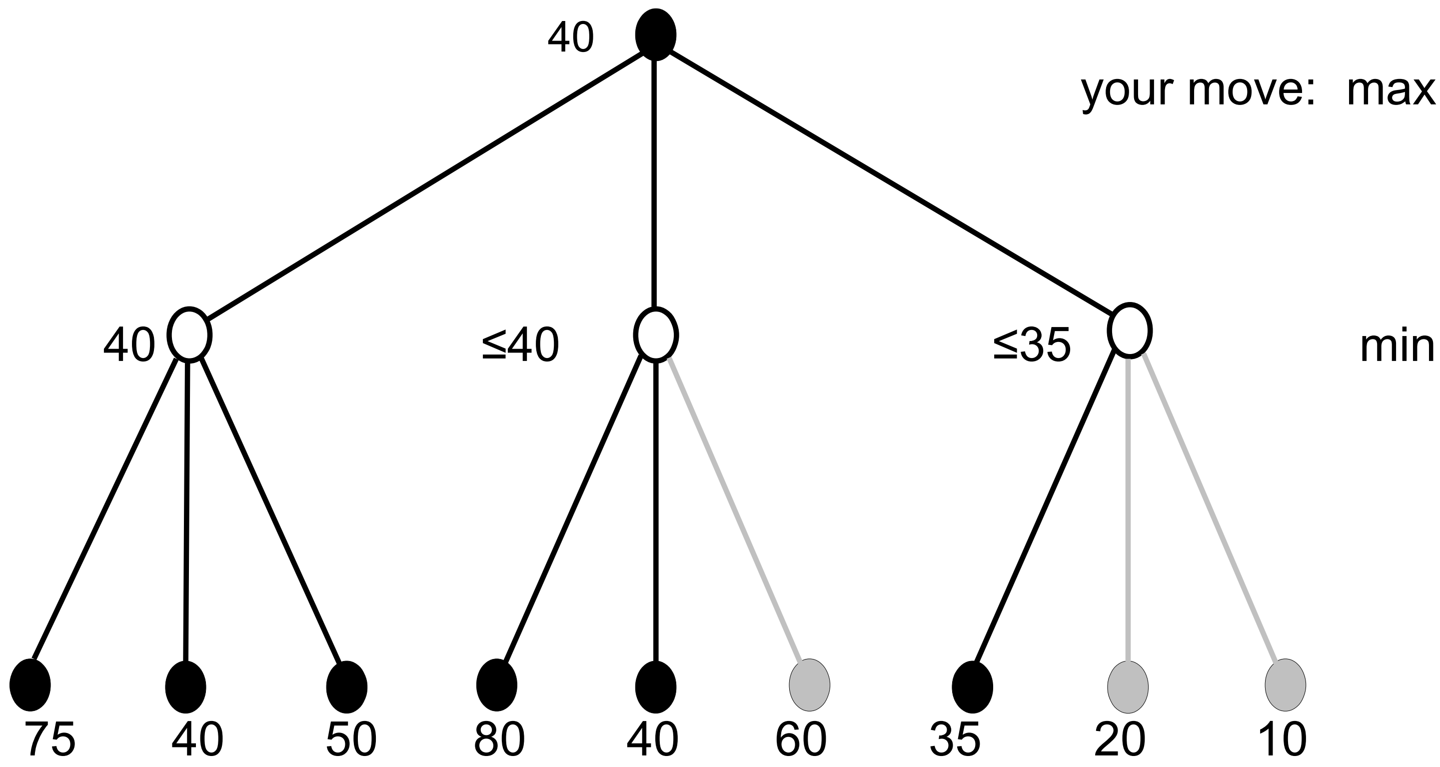


Bi-partite graph

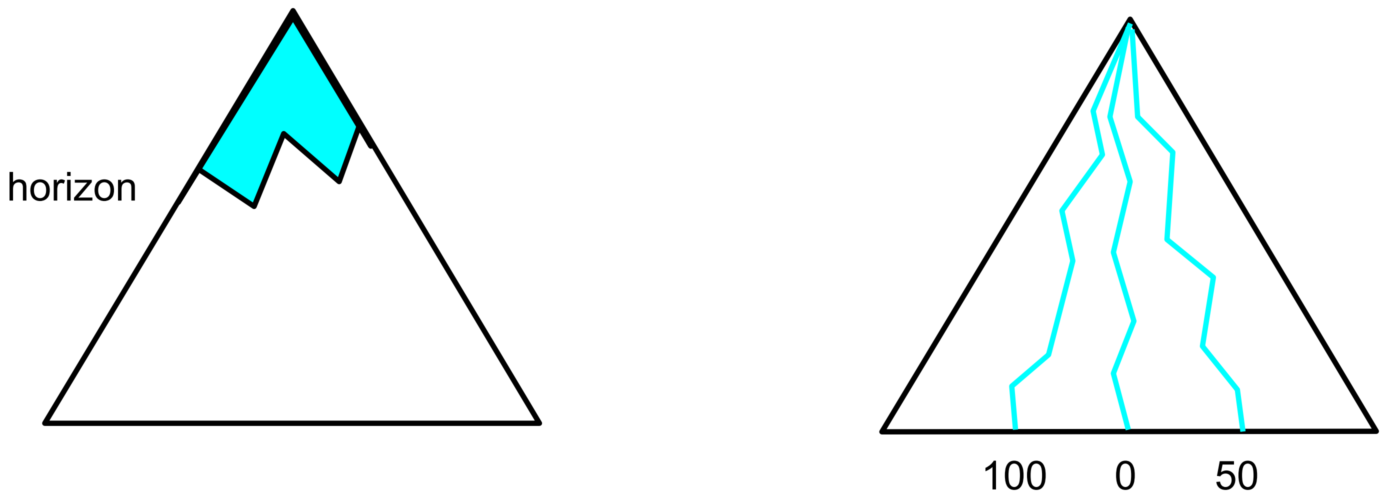
Minimax



Minimax With α-β-Heuristics



Stochastic Search (1)

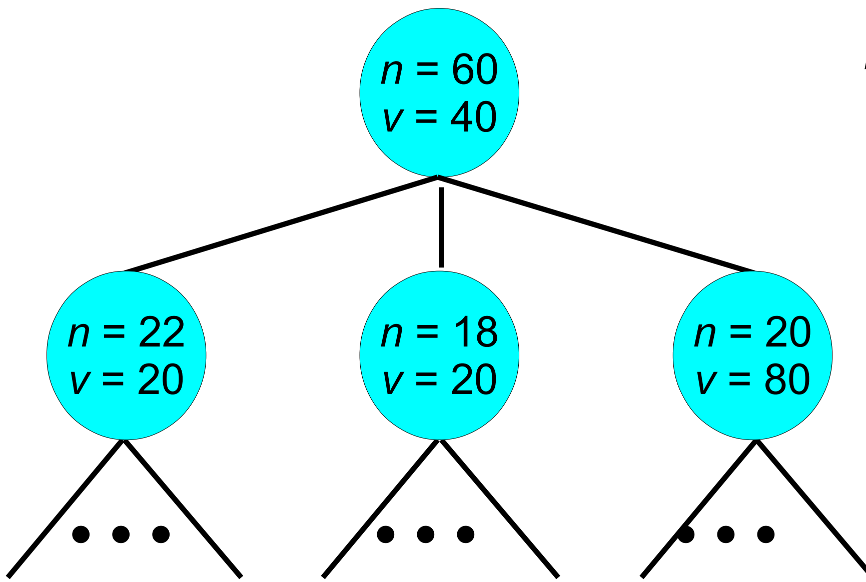


Minimax Search

Monte Carlo Tree Search
(random simulations)

Stochastic Search (2)

Value of move = Average score returned by simulation



n = # *of sample runs*
v = *average score*

Stochastic Search (3): Confidence Bounds

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

$$argmax_i \left(v_i + C * \boxed{\sqrt{\frac{\log n}{n_i}}} \right)$$

confidence bound