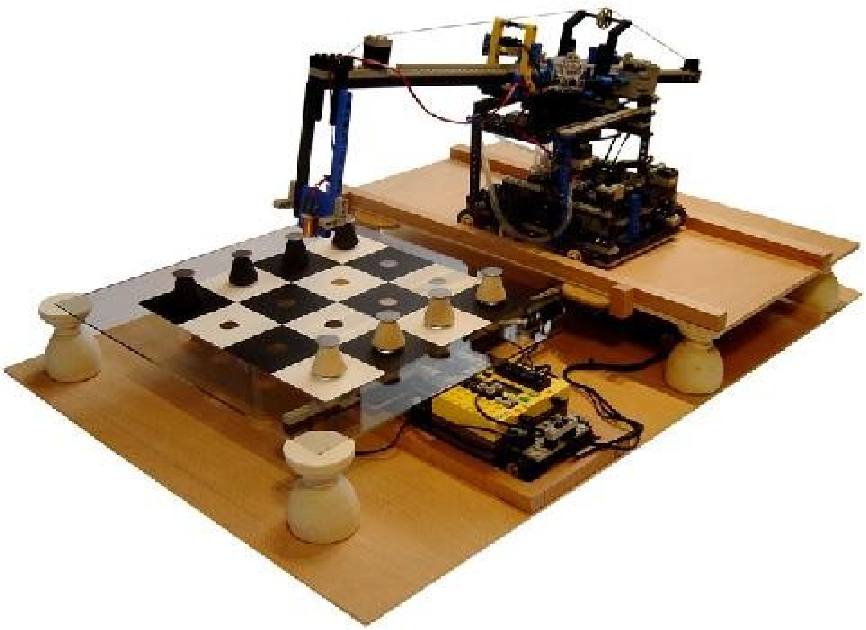
Prof Michael Thielscher
Adjunct at School of Computing & Mathematics
University of Western Sydney

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.

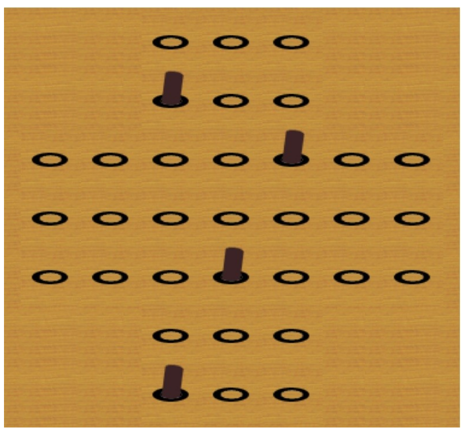
Unlike specialised game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.

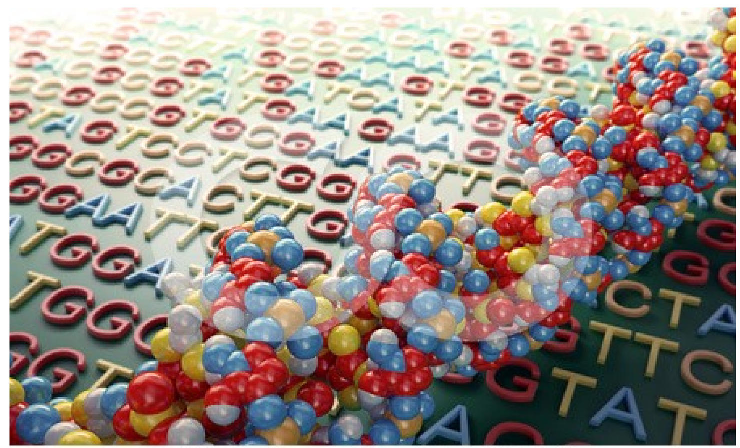
Variety of Games

Example: Noughts And Crosses

Single-Player Games

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

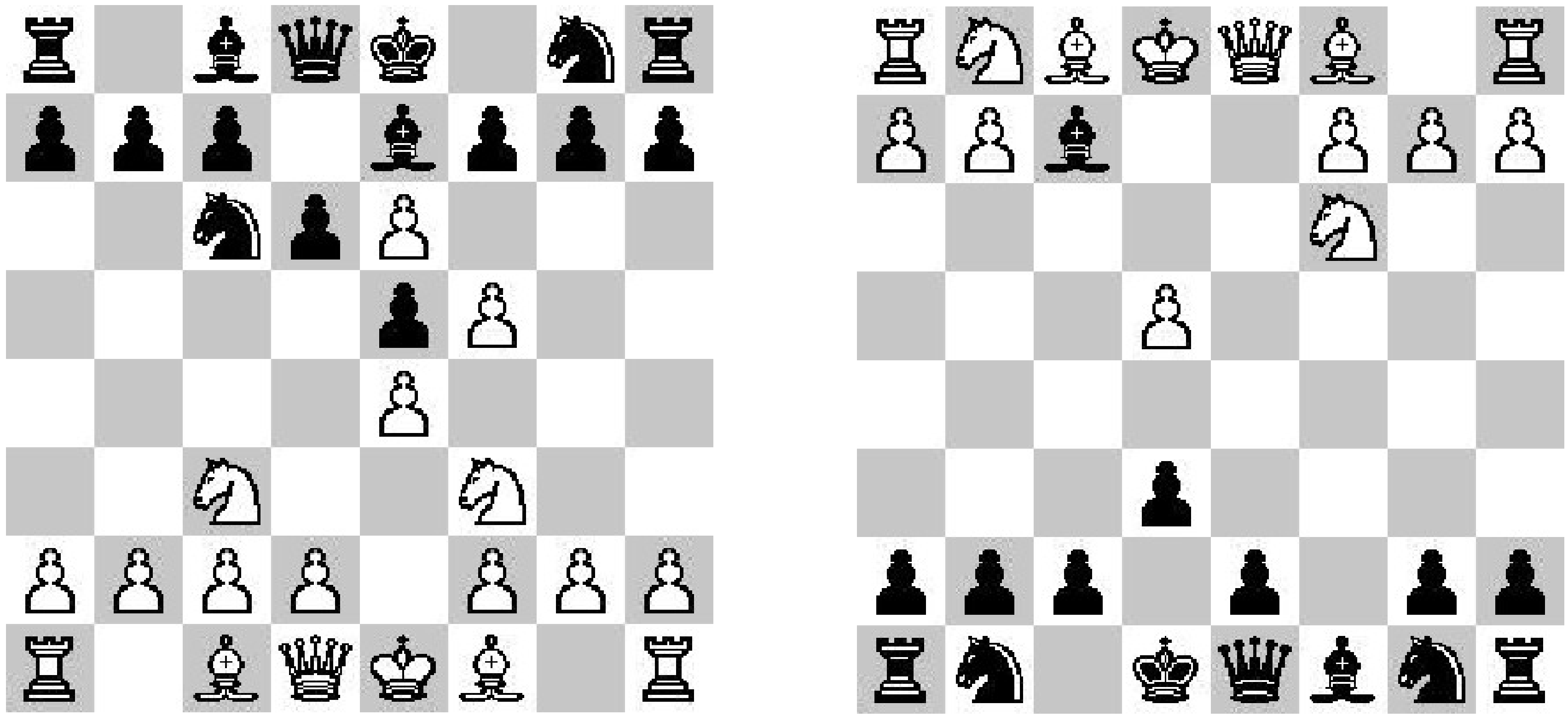




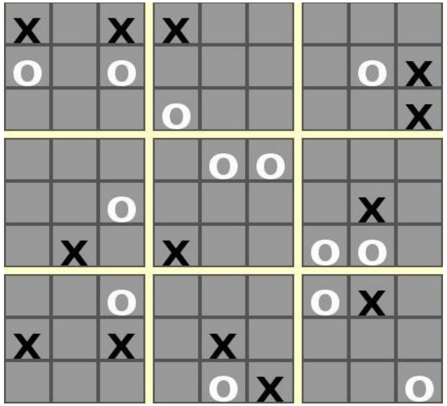
© Mopic * www.ClipartOf.com/433340

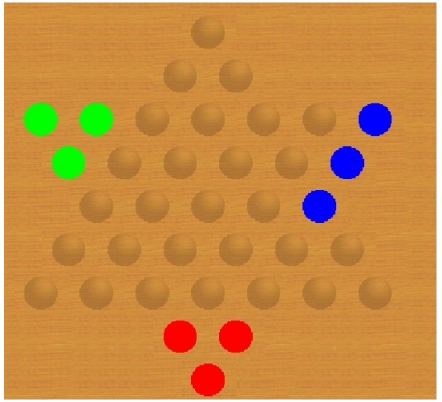
Chess

Bughouse Chess (a 4-Player Variant of Chess)

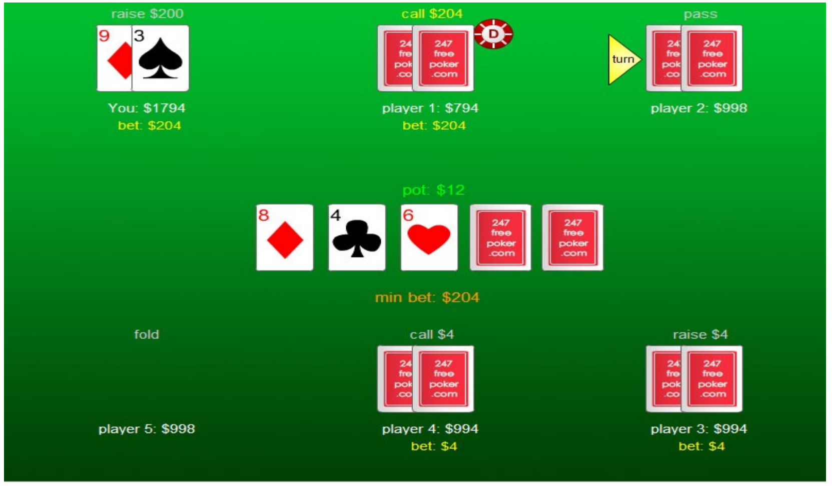


Other Games






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

International Activities

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

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

World Cup, administered by Stanford

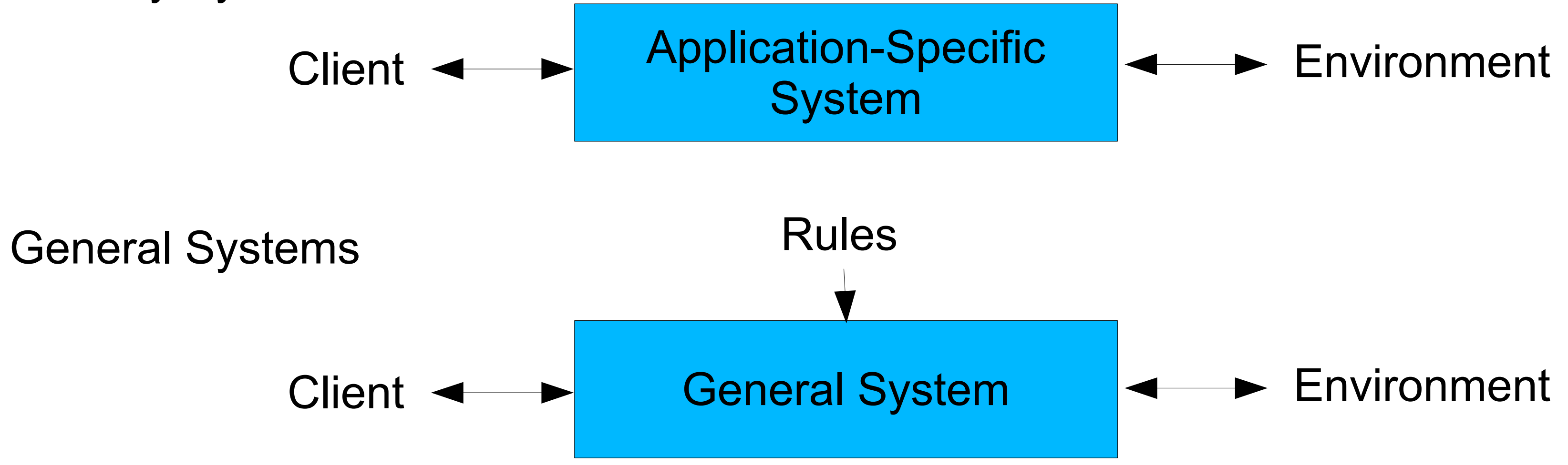
- 2005 – Cluneplayer (USA)
- 2006 – Fluxplayer (Germany)
- 2007, 2008 – Cadiaplayer (Iceland)
- 2009, 2010 – Ary (France)
- 2011 – TurboTurtle (USA)

General Game Playing and Artificial Intelligence

Why games?

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

Ordinary Systems



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

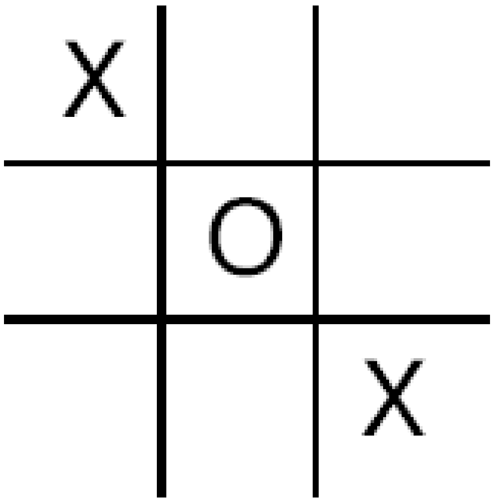
Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of tables (for legal moves, update, etc.)

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

Solutions:

- Reformulate in modular fashion
- Use compact encoding

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.

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 (dukep ?m ?n))
    (true (welcoul ?m ?n kwq))
    (true (himenoing ?p)))

(<= (next (welcoul ?m ?n ygg))
    (does lorchi (dukep ?m ?n)))

(<= (next (welcoul ?m ?n pyr))
    (does gniste (dukep ?m ?n)))
```

Logic Programs: A Subset of First-Order Logic

Clauses

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

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

`Body`: formula using conjunction, disjunction, negation

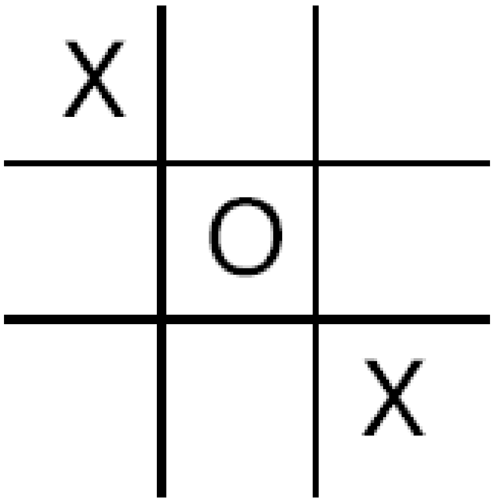
A <u>logic program</u> is a finite collection of clauses.

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: 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

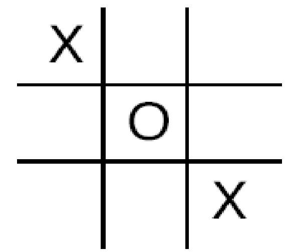
```
(row number mark)
(column number mark)
(diagonal mark)
(line mark)
open
draw
```

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)))
```




```
(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)
```

Conclusions:

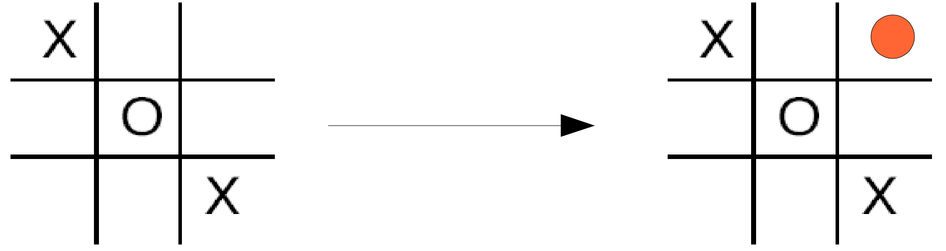
```
(legal xplayer noop)
(legal oplayer (mark 1 2))
...
(legal oplayer (mark 3 2))
```

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


(<= (line ?w) (row ?m ?w))
(<= (line ?w) (column ?n ?w))
(<= (line ?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 (not (line x))
         (not (line o))
         (not open))
```

Summary: Noughts And Crosses

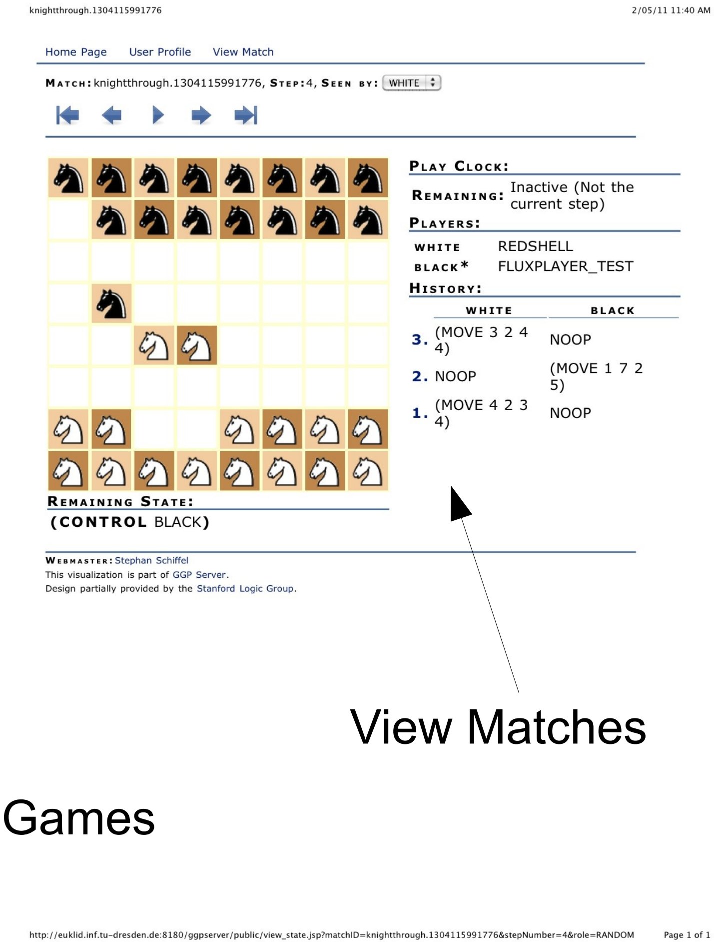
```
(role xplayer)
(role(oplayer)

(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))

(<= (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

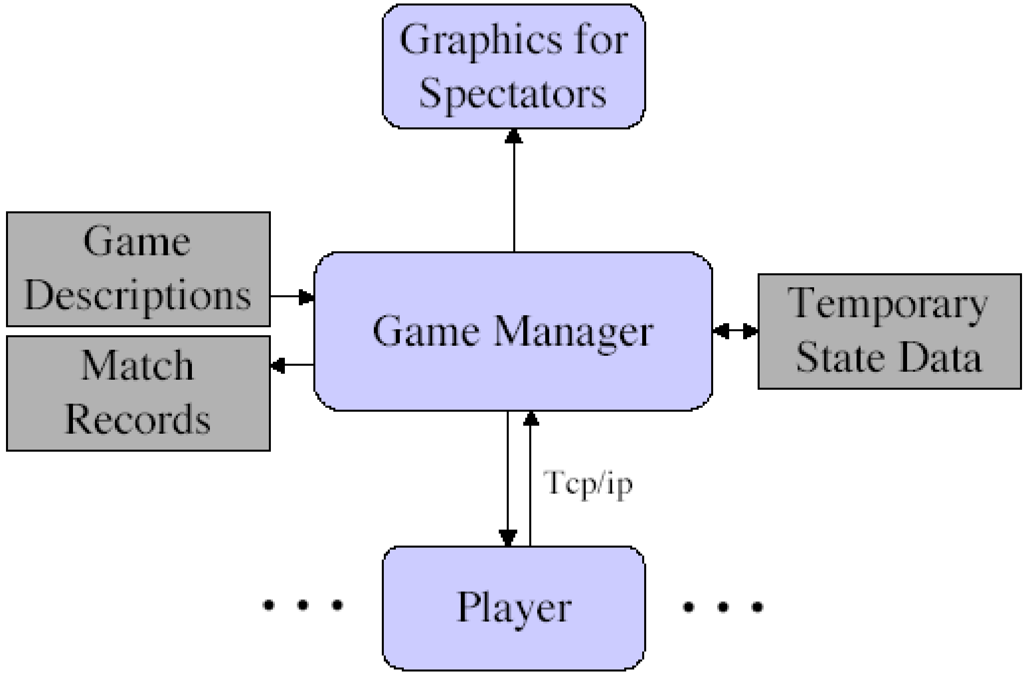
http://euklid.inf.tu-dresden.de:8180/ggpserver/index.jsp



View Matches

Games

Game Manager



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>)`

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:

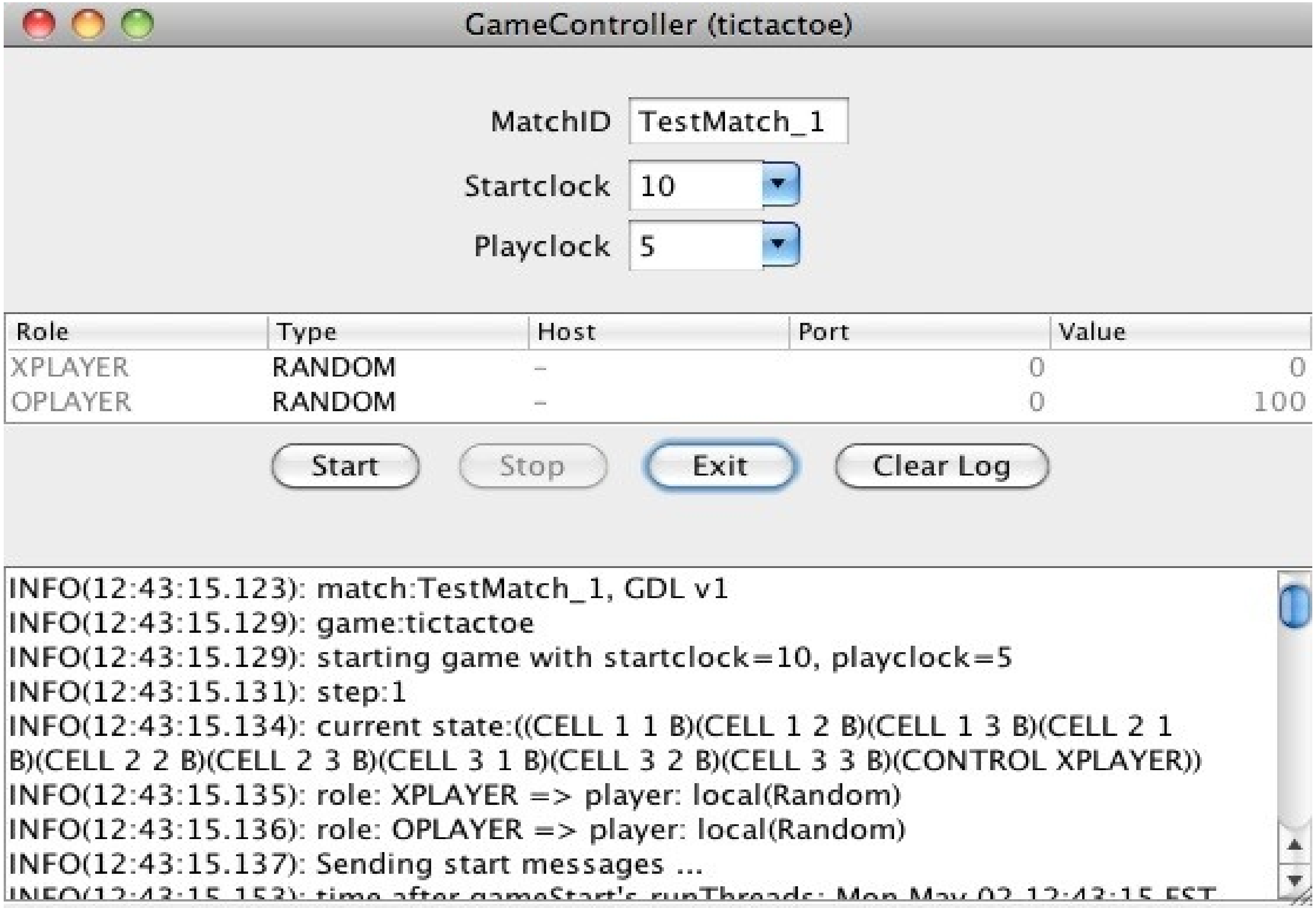
- Linux/Unix (or any system which provides sockets)

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

Download Manager

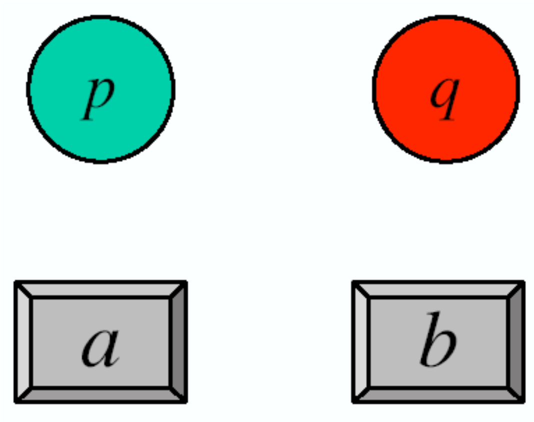
Download Basic Players

GameControllerApp



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 `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

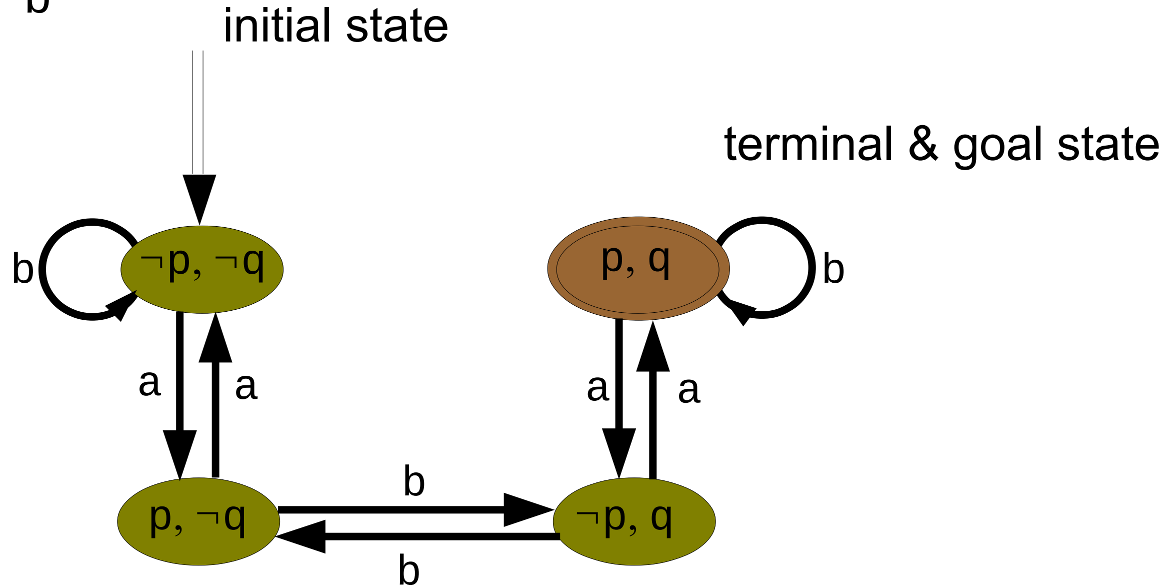
- Goal

  $p \wedge q$

State Transition System

State features: p, q

Actions: a, b



Solution (= Plan): a, b, a

Single-Player Games with Complete Information

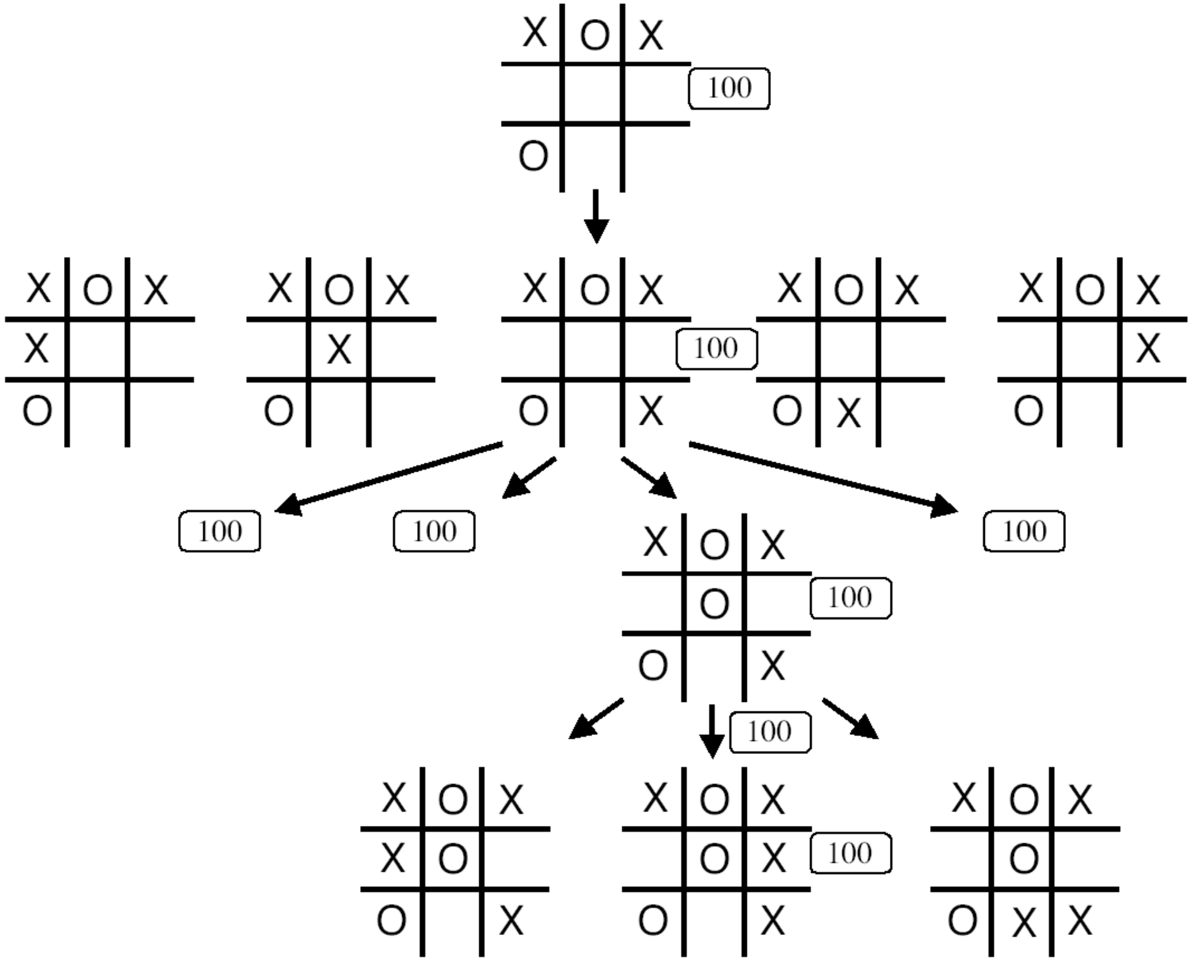
Many single-player games can be solved using standard search techniques

- Iterative deepening
- Bidirectional search

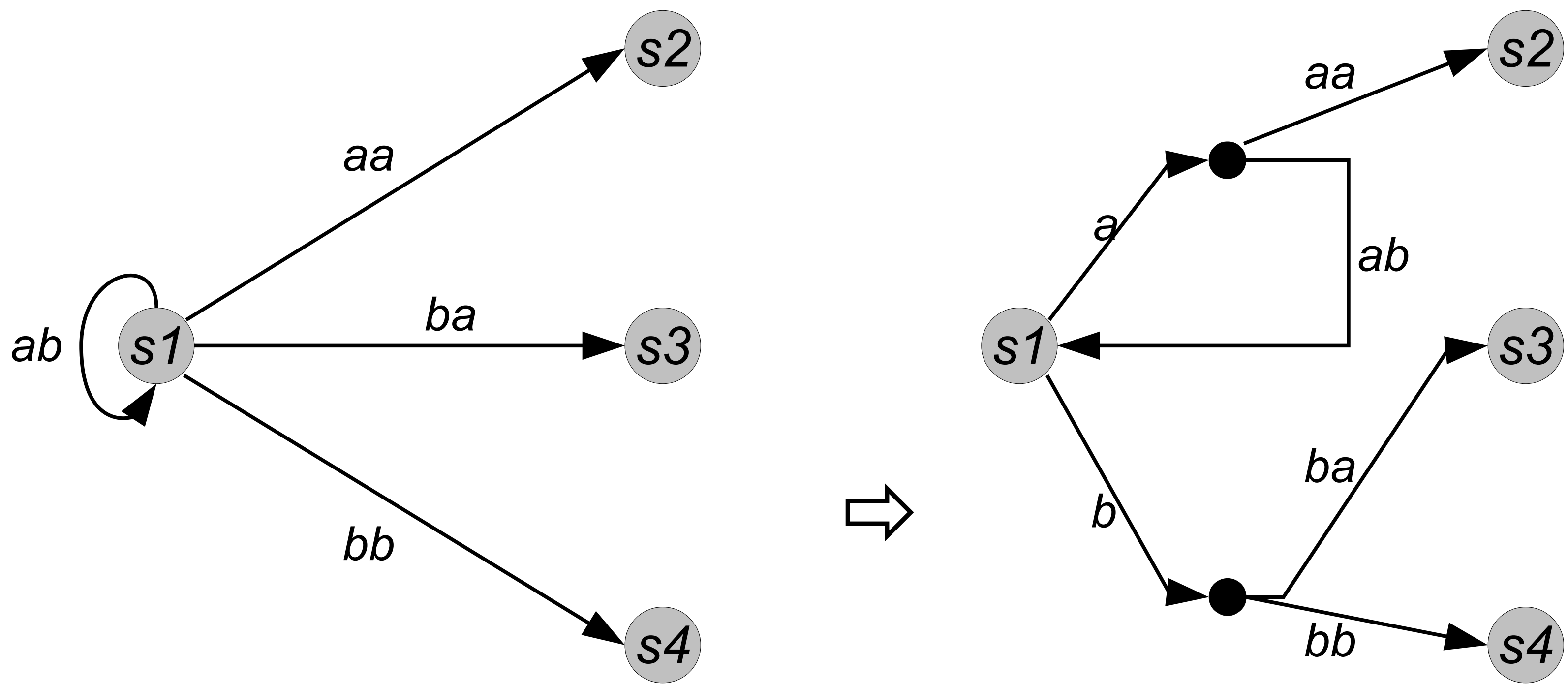
Special techniques

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

Multi-Player Games: Game Tree Search (Example)



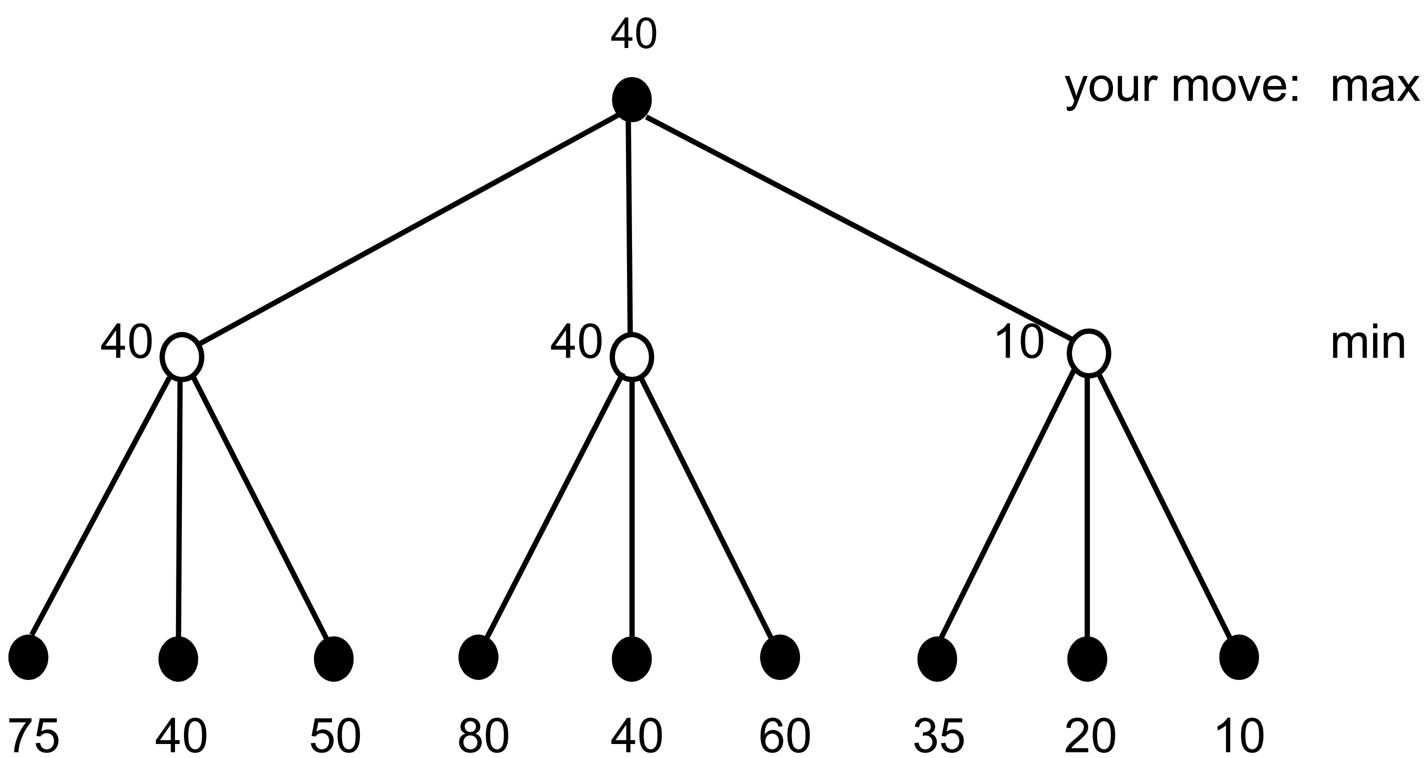
How to Deal With Simultaneous Moves



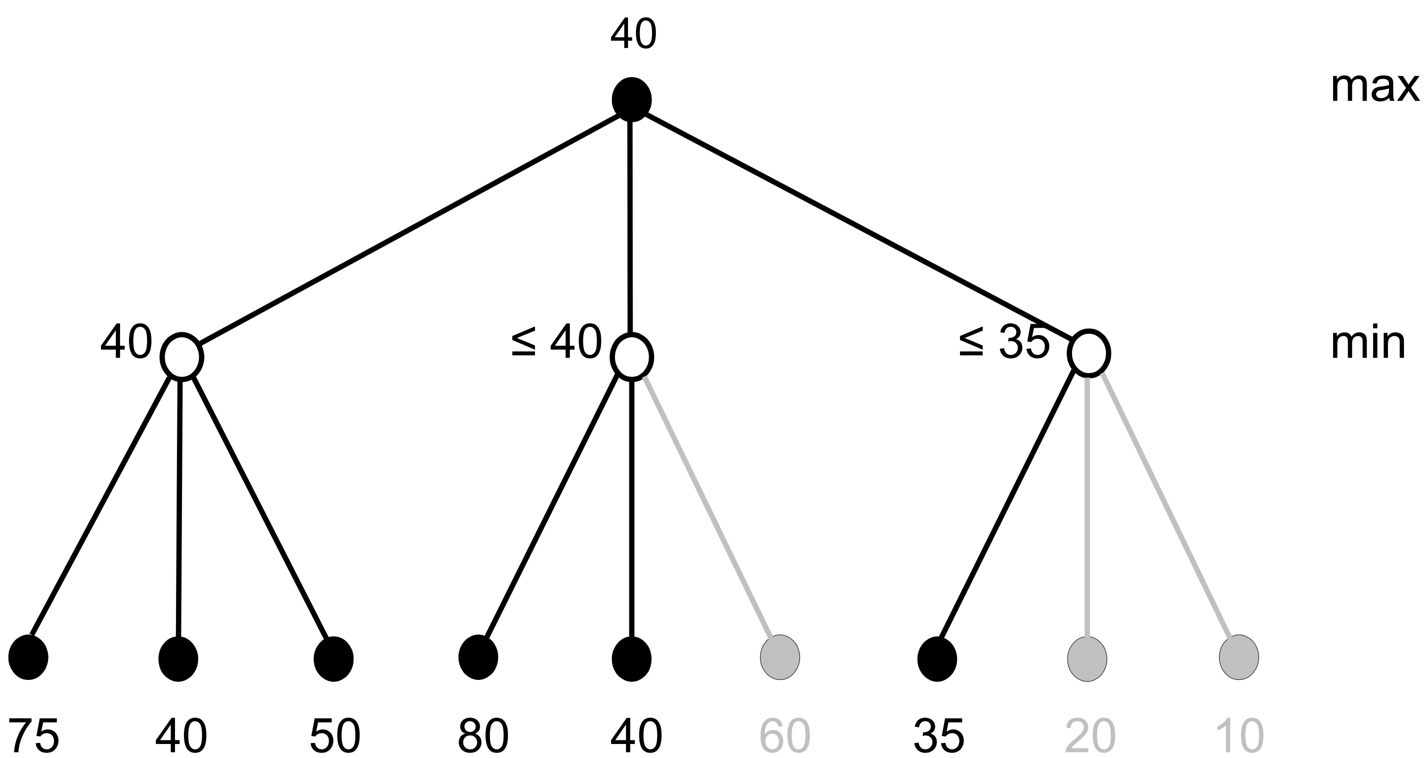
State transition graph

Bi-partite graph

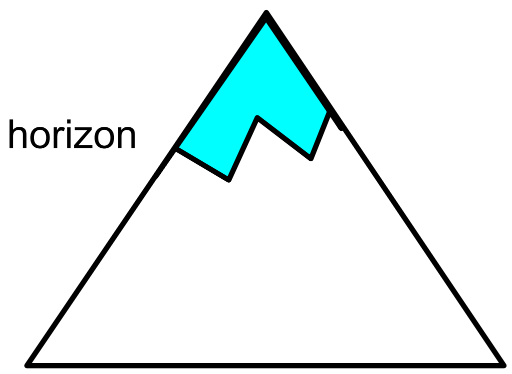
Minimax



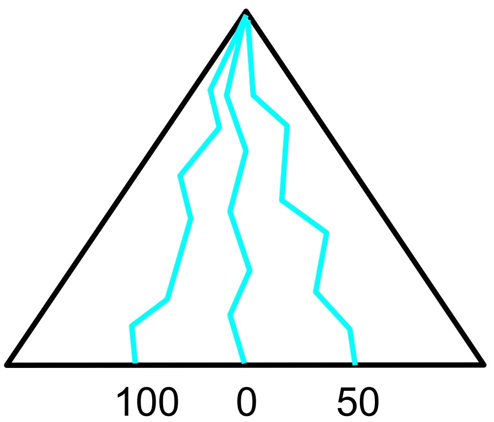
Minimax With α-β-Heuristics



Stoachastic Search (1)



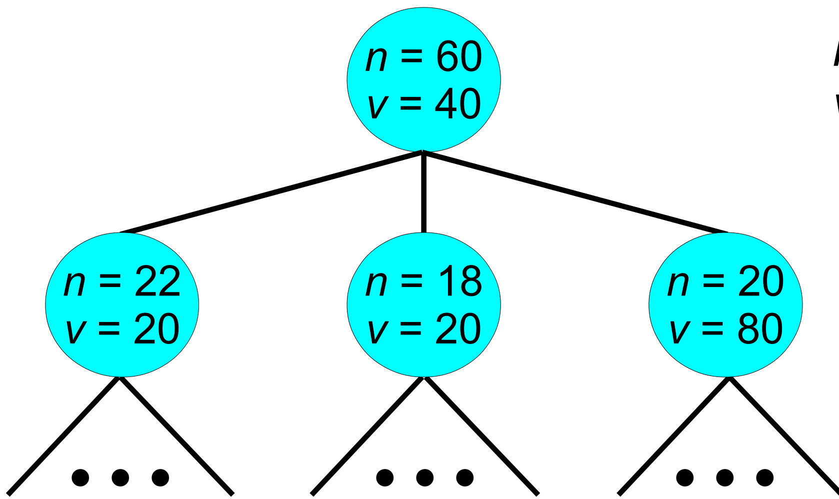
Game Tree Seach



Monte Carlo Tree Search
(random simulations)

Stochastic Search (2)

Value of move = Average score returned by simulation



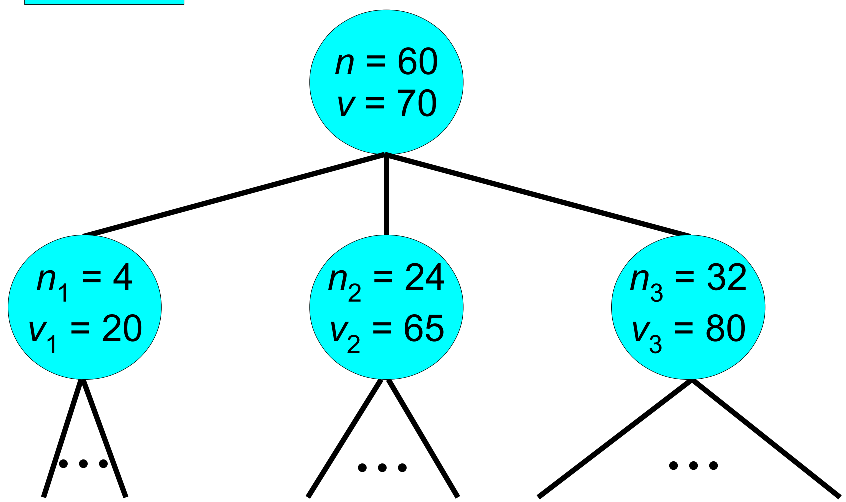
n = # of sample runs
v = average score

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



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`