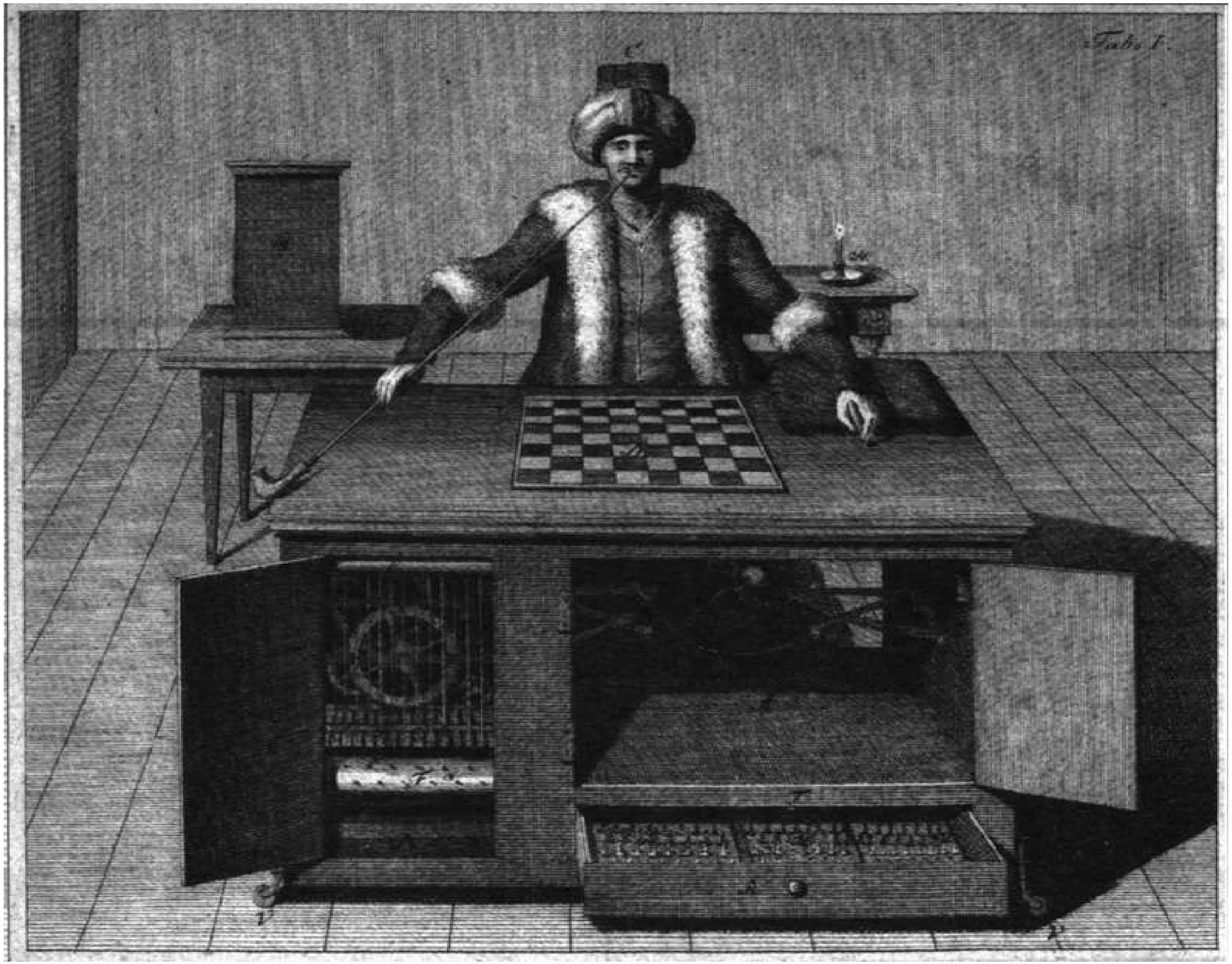
IJCAI'09 Tutorial

New Trends in General Game Playing

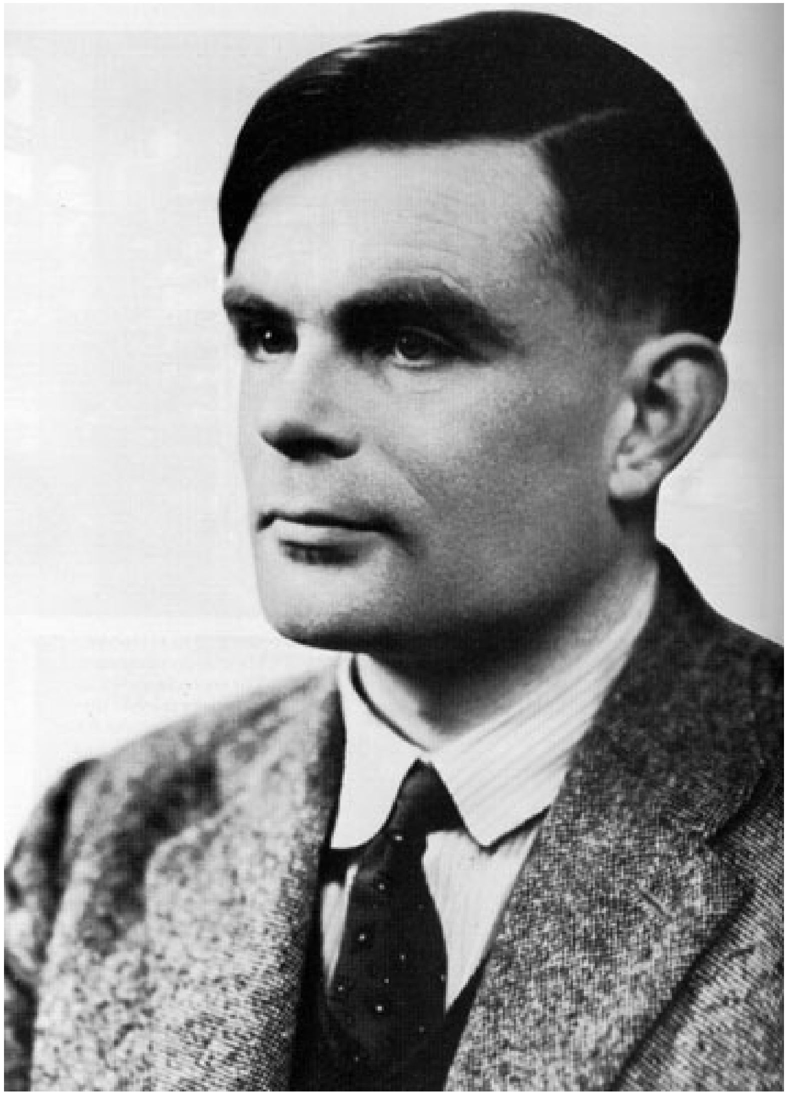
Michael Thielscher, Dresden

Chess Players

The 1st Chess Computer ("Turk", 18th Century)



Alan Turing & Claude Shannon (~1950)



Deep-Blue Beats World Champion (1997)

In the early days, game playing machines were considered a key to Artificial Intelligence.

But Deep Blue is a highly specialized system--it can't even play a decent game of Tic-Tac-Toe or Rock-Paper-Scissors!

A General Game Player is a system that

- understands formal descriptions of arbitrary games
- learns to play these games well without human intervention

General Game Playing is considered a grand AI Challenge

Rather than being concerned with a specialized solution to a narrow problem, General Game Playing encompasses a variety of AI areas:

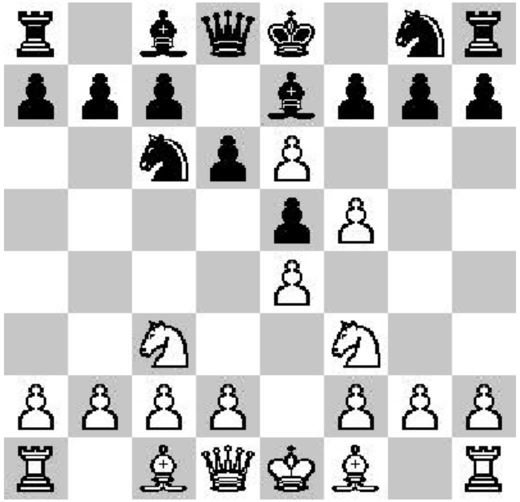
- AI Game Playing
- Knowledge Representation and Reasoning
- Search, Planning
- Learning
- ... and more!

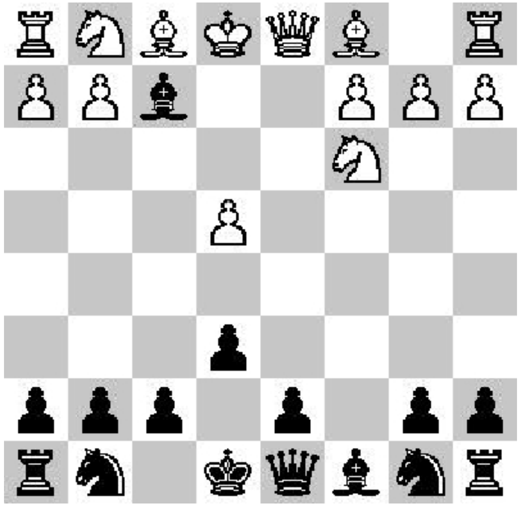
General Game Playing and AI

Agents	Games
Competitive environments	Deterministic, complete information
Uncertain environments	Nondeterministic, partially observable
Unknown environment model	Rules partially unknown
Real-world environments	Robotic player

Application (1)

Commercially available chess computers can't be used for a game of Bughouse Chess.





An adaptable game computer would allow the user to modify the rules for arbitrary variants of a game.

Application (2): General Agents

A General Agent is a system that

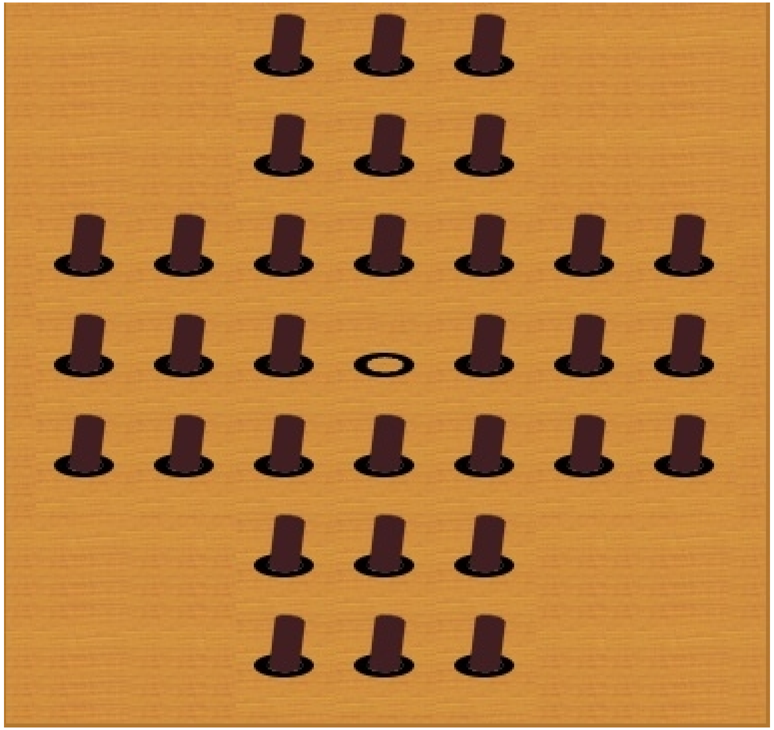
- understands formal descriptions of arbitrary multiagent environments
- learns to function in this environment without human intervention

Examples

- Rules of e-marketplaces made accessible to agents
- Internet platforms that are formally described
- Providing details in agent competitions (eg, TAC) at runtime

Example Games

Single-Player, Deterministic

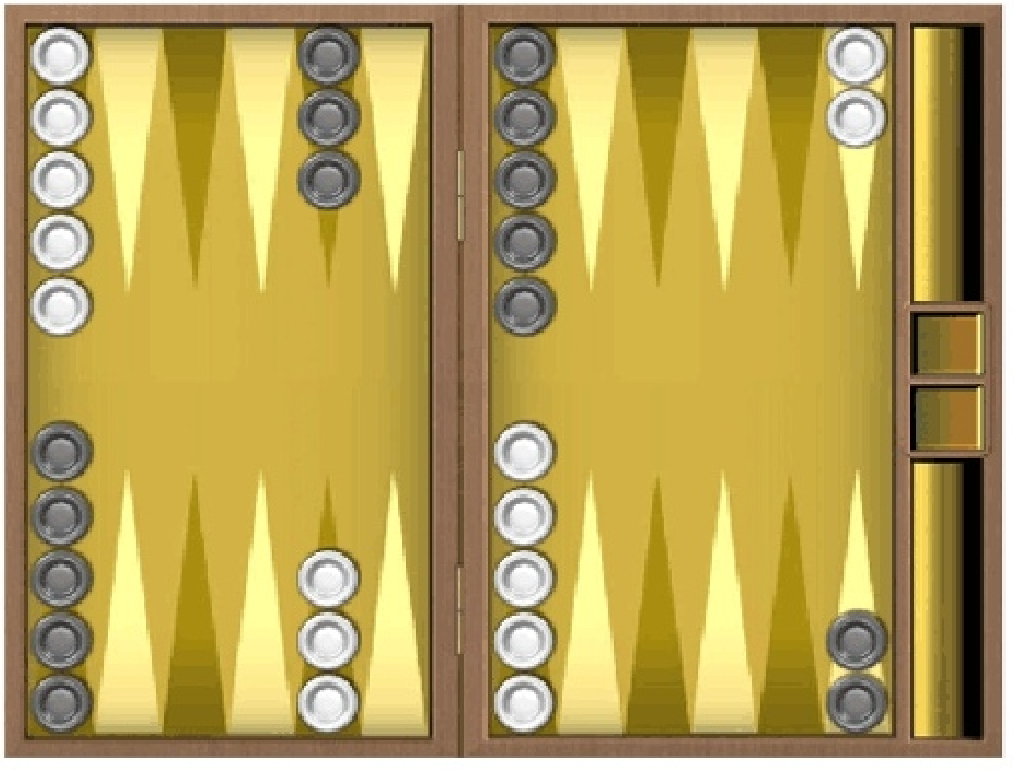


Demo: Single-Player, Deterministic

Two-Player, Zero-Sum, Deterministic

Two-Player, Zero-Sum, Deterministic

Two-Player, Zero-Sum, Nondeterministic



Two-Player, Simultaneous Moves

n-Player, Incomplete Information, Nondeterministic

The History of General Game Playing

- 1993 B. Pell: “Strategy Generation and Evaluation for Meta-Game Playing“ (PhD Thesis, Cambridge)
- 2005 1st AAAI General Game Playing Competition
- 2006 First publications on General Game Playing
- 2009 1st General Game Playing Workshop (GIGA'09)

- Research groups world-wide: Austin, Bremen, Dresden, Edmonton, Liverpool, Paris, Potsdam, Reykjavik, Sydney

Roadmap: New Trends in GGP

- Description Languages
- Reasoning about Game Descriptions
- Generating Evaluation Functions
- Learning by Simulation

Description Languages

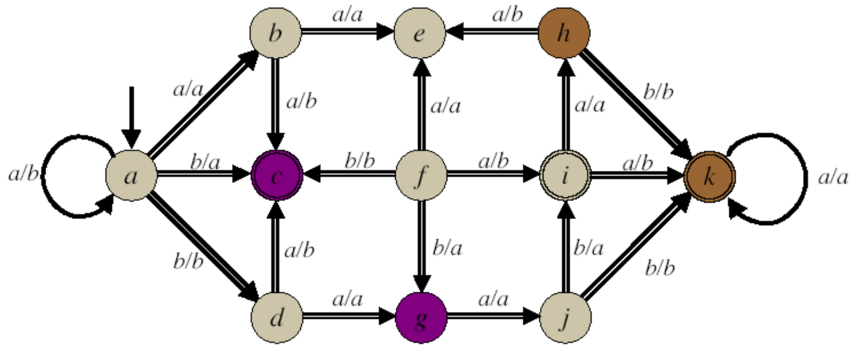
Description Languages: Overview

- The technology of General Agents requires languages to describe the rules that govern an environment

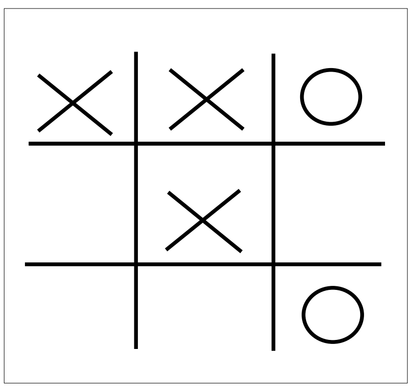
- Descriptions
  - should be easy to understand and maintain
  - can be fully automatically processed by a computer
  - must have a precise semantics

- Examples
  - Game Description Language GDL
  - Market Specification Language MSL

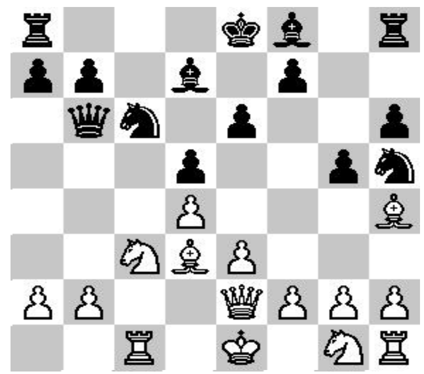
Every finite game can be modeled as a state transition system



But direct encoding impossible in practice

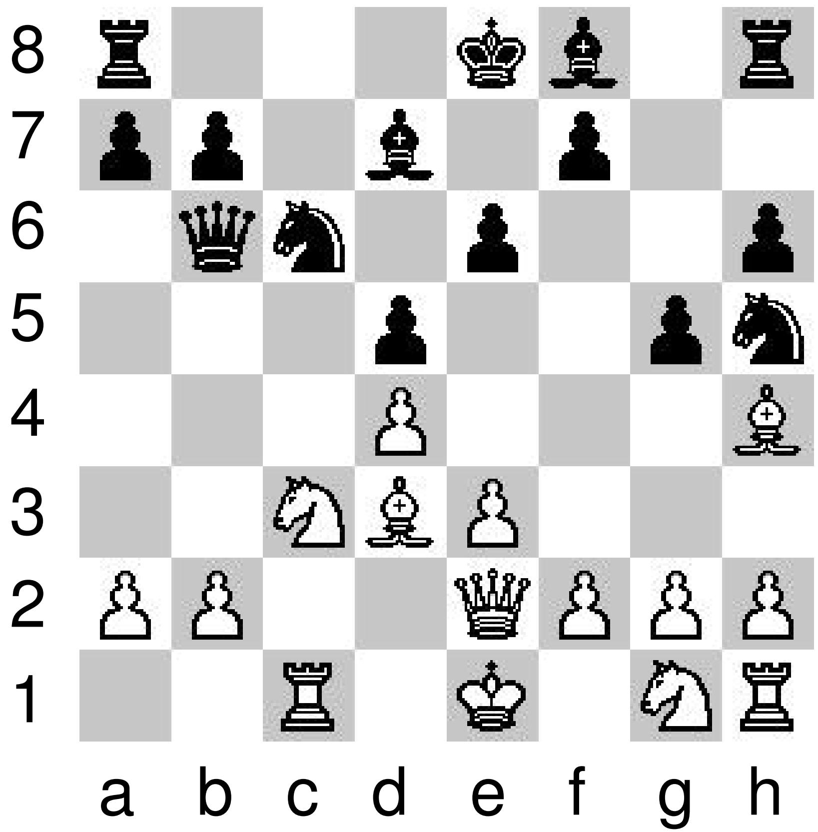


19,683 states



~ 10^{46} legal positions

Modular State Representation: Features



`cell(X,Y,C)`

X ∈ {a,…,h}

Y ∈ {1,…,8}

C ∈ {whiteKing,…,blank}

`control(P)`

P ∈ {white,black}

Feature Representation for Chess (2)

`canCastle(P,S)`

`P ∈ {white,black}`

`S ∈ {kingsSide,queensSide}`

`enPassant(C)`

`C ∈ {a,…,h}`

Moves

`move(U,V,X,Y)`

$U, X \in \{a, \ldots, h\}$

$V, Y \in \{1, \ldots, 8\}$

`promote(X,Y,P)`

$X, Y \in \{a, \ldots, h\}$

$P \in \{whiteQueen, \ldots\}$

Game Description Language GDL

Based on the features and moves of a game, the rules can be described in formal logic using a few standard predicate symbols

`role(P)`	`P` is a player
`init(F)`	`F` holds in the initial position
`true(F)`	`F` holds in the current position
`legal(P,M)`	player `P` has legal move `M`
`does(P,M)`	player `P` does move `M`
`next(F)`	`F` holds in the next position
`terminal`	the current position is terminal
`goal(P,N)`	player `P` gets reward `N` in current position

Elements of a Game Description (1)

- Players

```
role(white) <=
role(black) <=
```

- Initial position

```
init(cell(a,1,whiteRook)) <=
...
```

- Moves

```
legal(white,promote(X,Y,P)) <=
   true(cell(X,7,whitePawn)) ∧ ...
```

Elements of a Game Description (2)

- Moves: Update

```
next(cell(X,Y,C)) <=
     does(P,move(U,V,X,Y))
     ∧ true(cell(U,V,C))
```

- End of game

```
terminal <=
     checkmate ∨ stalemate
```

- Result

```
goal(white,100) <=
        checkmate
        ∧ true(control(black))
goal(white, 50) <= stalemate
```

A Complete Formalization of Tic-Tac-Toe (1/3)

```
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(X,Y)) <=
        true(cell(X,Y,b)) ∧
        true(control(P))

legal(xplayer,noop) <=
        true(cell(X,Y,b)) ∧
        true(control(oplayer))

legal(oplayer,noop) <=
        true(cell(X,Y,b)) ∧
        true(control(xplayer))
```

Rules of Tic-Tac-Toe (2/3)

```
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)) ∧ (¬M=J ∨ ¬N=K)

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

terminal <= line(x) ∨ line(o) ∨ ¬open

line(W) <= row(M,W) ∨ column(M,W) ∨ diagonal(M,W)
open <= true(cell(M,N,b))
```

Rules of Tic-Tac-Toe (3/3)

```
goal(xplayer,100) <= line(x)
goal(xplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(xplayer,0)   <= line(o)
goal(oplayer,100) <= line(o)
goal(oplayer,50)  <= ¬line(x) ∧ ¬line(o) ∧ ¬open
goal(oplayer,0)   <= line(x)

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))
 ∨ true(cell(1,3,W))∧true(cell(2,2,W))∧true(cell(3,1,W))
```

Properties of GDL

- GDL rules are logic programs, including the use of negation-as-failure
- Additional, syntactic restrictions ensure that all relevant derivations are finite
- The language is completely knowledge-free: symbols like `cell` and `control` acquire meaning only through the rules
- To make this clear, GDL descriptions are often obfuscated

For details see [Genesereth, Love & Pell, 2006]

Obfuscated Rules: How the Computer Sees a Game Description

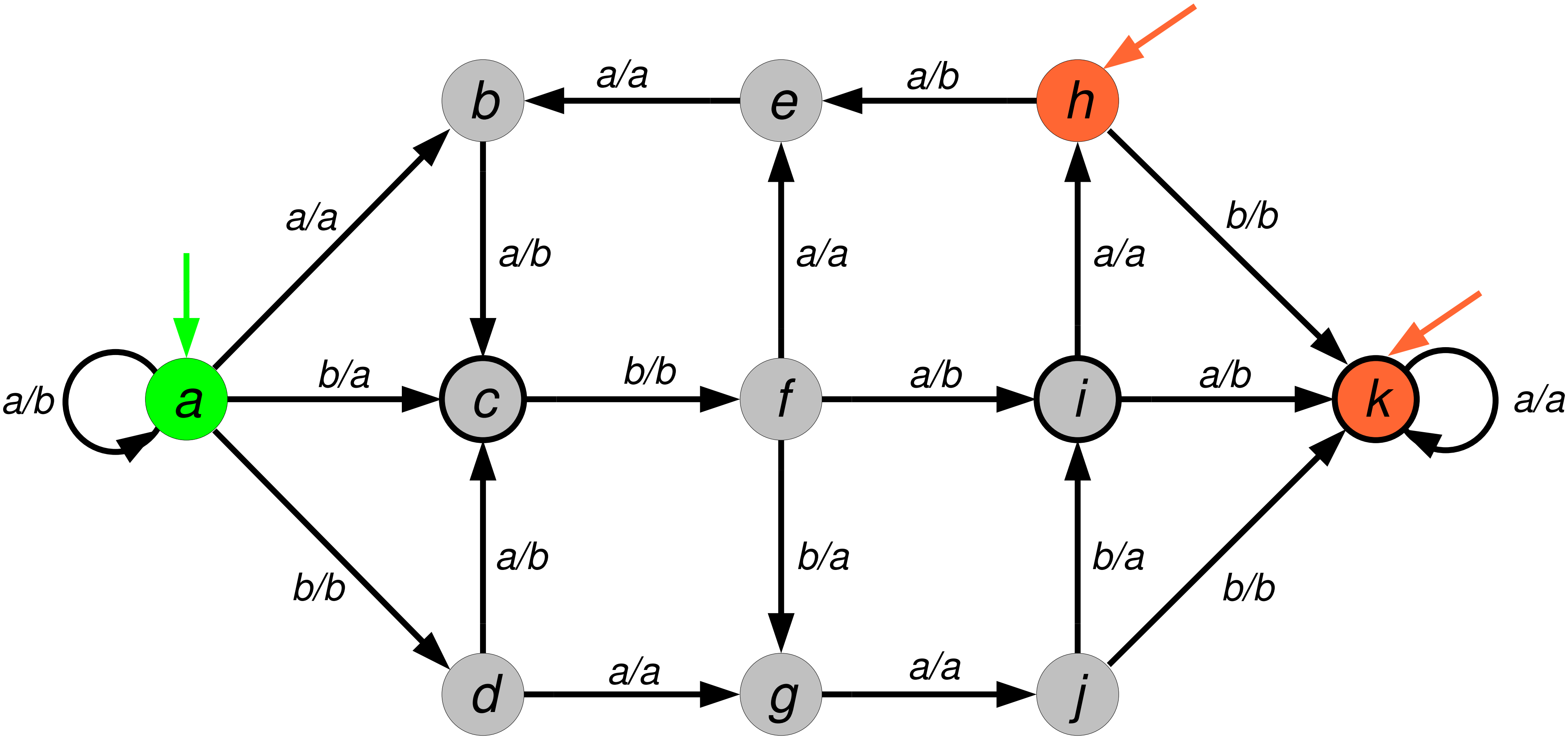
```
next(thuis(M,N,een)) <= does(jij,huur(M,N))
next(thuis(M,N,het)) <= does(wij,huur(M,N))

next(fiets(jij)) <= true(fiets(wij))
next(fiets(wij)) <= true(fiets(jij))

terminal    <= brommer(een) ∨ brommer(het) ∨ ¬keer
brommer(W)  <= gaag(M,W) ∨ daag(M,W) ∨ naar(M,W)

...
```

Semantics: Games as State Machines



Game Model

A game is a structure with the following components:

R – set of players

S – set of states

A – set of moves

$l \subseteq \mathrm{R} \times \mathrm{A} \times \mathrm{S}$ – the legality relation

$u: \mathrm{M} \times \mathrm{S} \rightarrow \mathrm{S}$ – the update function, for joint moves $m: \mathrm{R} \rightarrow \mathrm{A}$

$s_1 \in \mathrm{S}$ – initial game state

$t \subseteq \mathrm{S}$ – terminal states

$g \subseteq \mathrm{R} \times \mathrm{S} \times \mathbb{N}$ – the goal relation

From the Rules to the Game Model (Example): Initial Position

A GDL description P encodes $s_1 = \{ f : P \models \texttt{init}(f) \}$

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

From the Rules to the Game Model: Legality Relation

Let $S^{\texttt{true}} := \{ \texttt{true}(f) : f \in S \}$.

Then P encodes $l = \{ (r \in R, a, S) : P \cup S^{\texttt{true}} \models \texttt{legal}(r,a) \}$

```
legal(P,mark(X,Y)) <= true(cell(X,Y,b)) ∧
                      true(control(P))

...
```

From the Rules to the Game Model: Update Function

Let $m^{\texttt{does}} := \{\ \texttt{does}(r,m(r)) : r \in \mathrm{R}\ \}$.

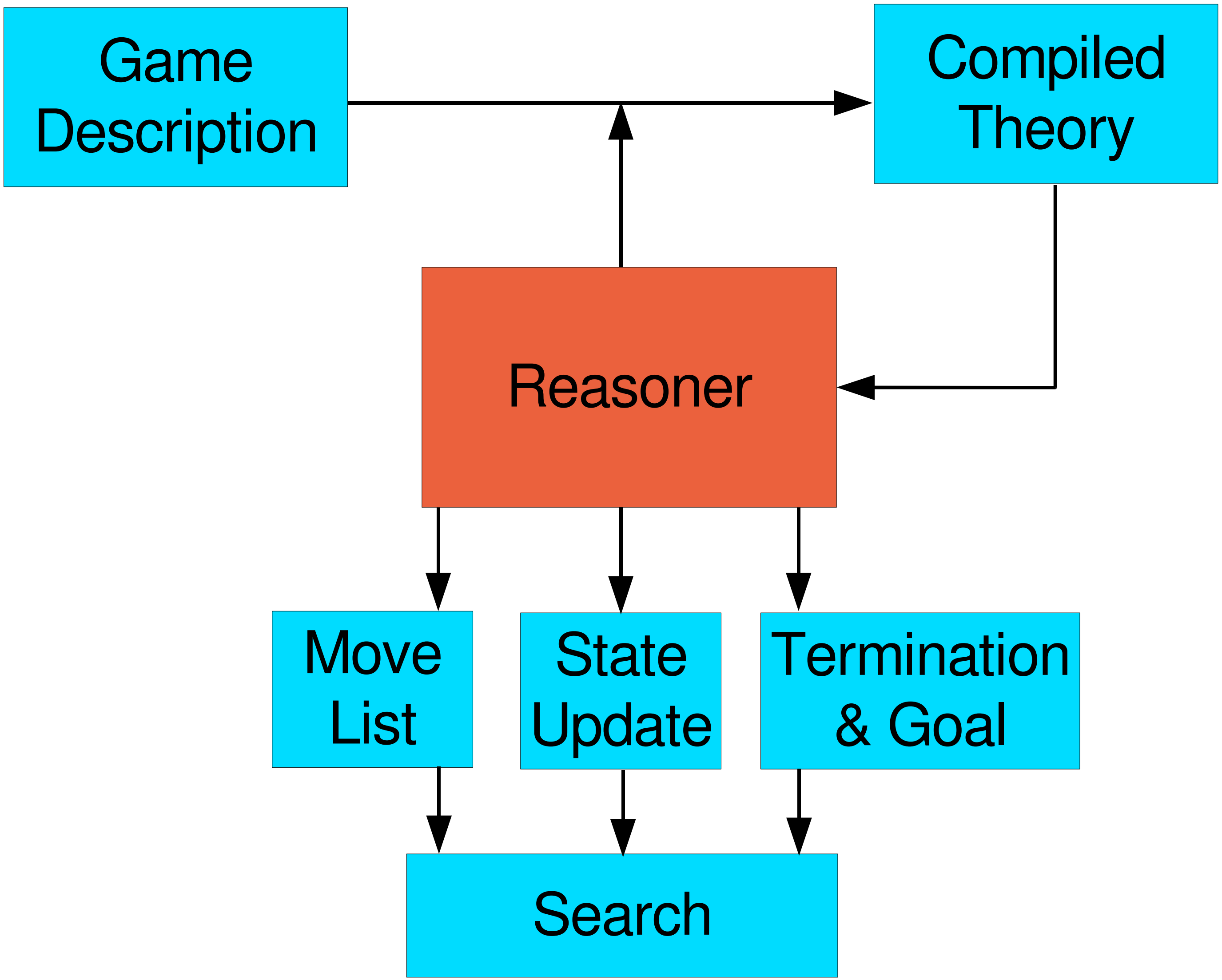
Then P encodes $u(m,S) = \{\ f : P \cup S^{\texttt{true}} \cup m^{\texttt{does}} \models \texttt{next}(f)\ \}$

```
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)) ∧ (¬M=J ∨ ¬N=K)
```

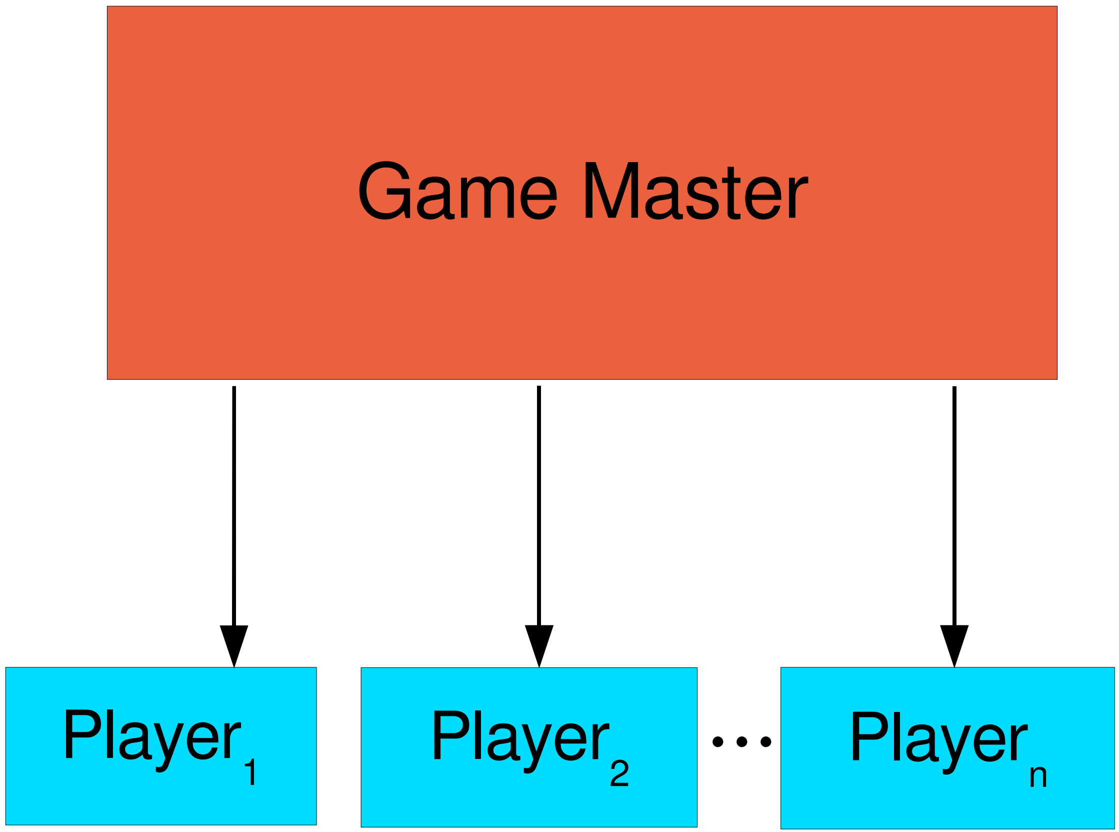
. . .

For details see [Schiffel &Thielscher, 2009a]

A Basic Player

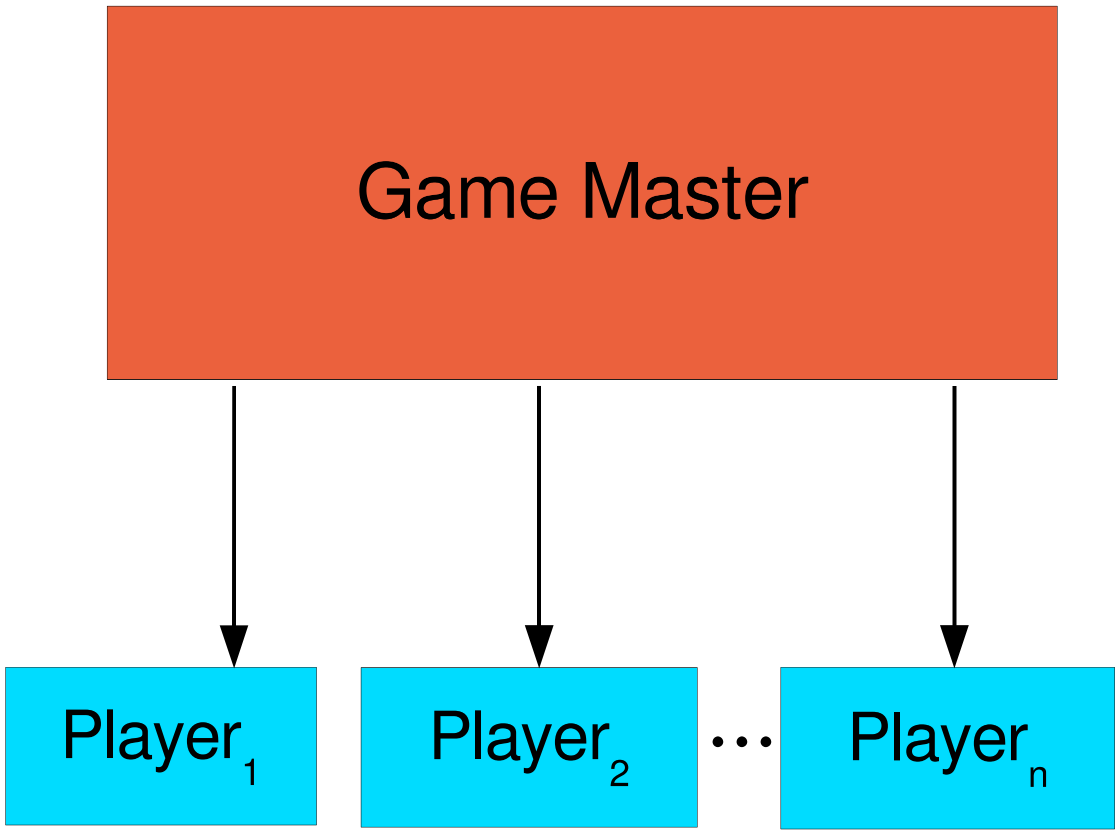


Actual Game Play

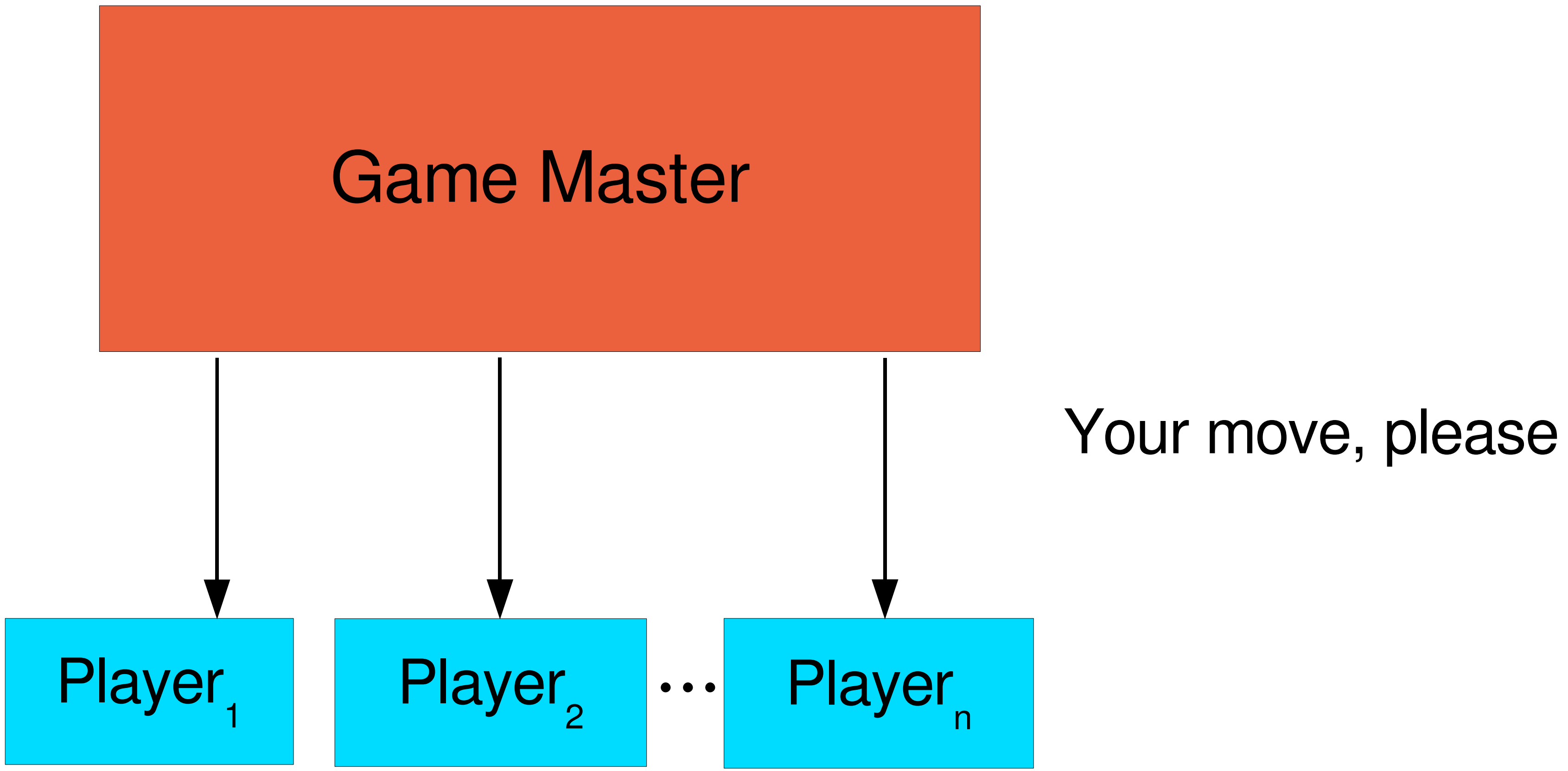


Game description
Time to think: 1,800 sec
Time per move: 45 sec
Your role

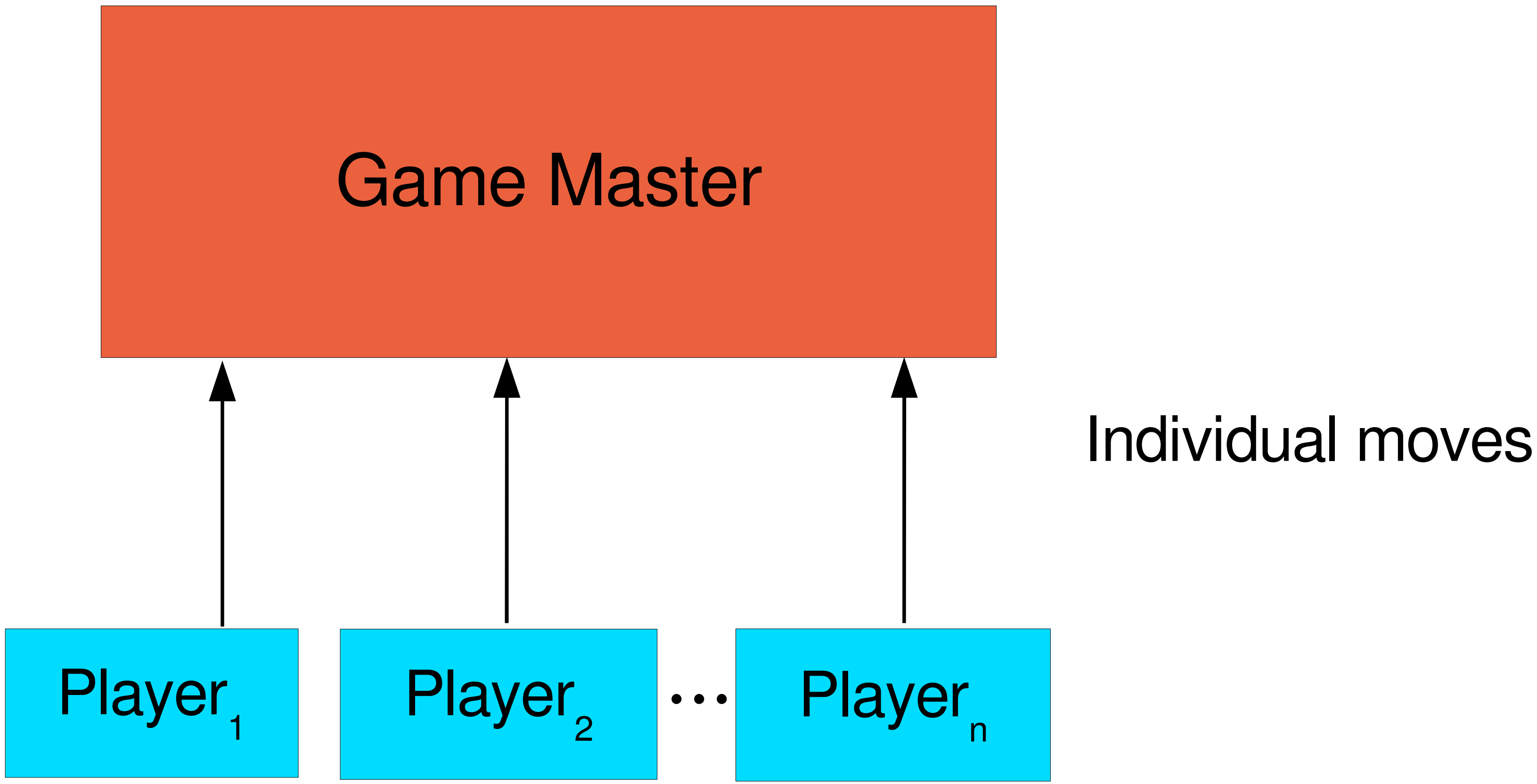
Actual Game Play



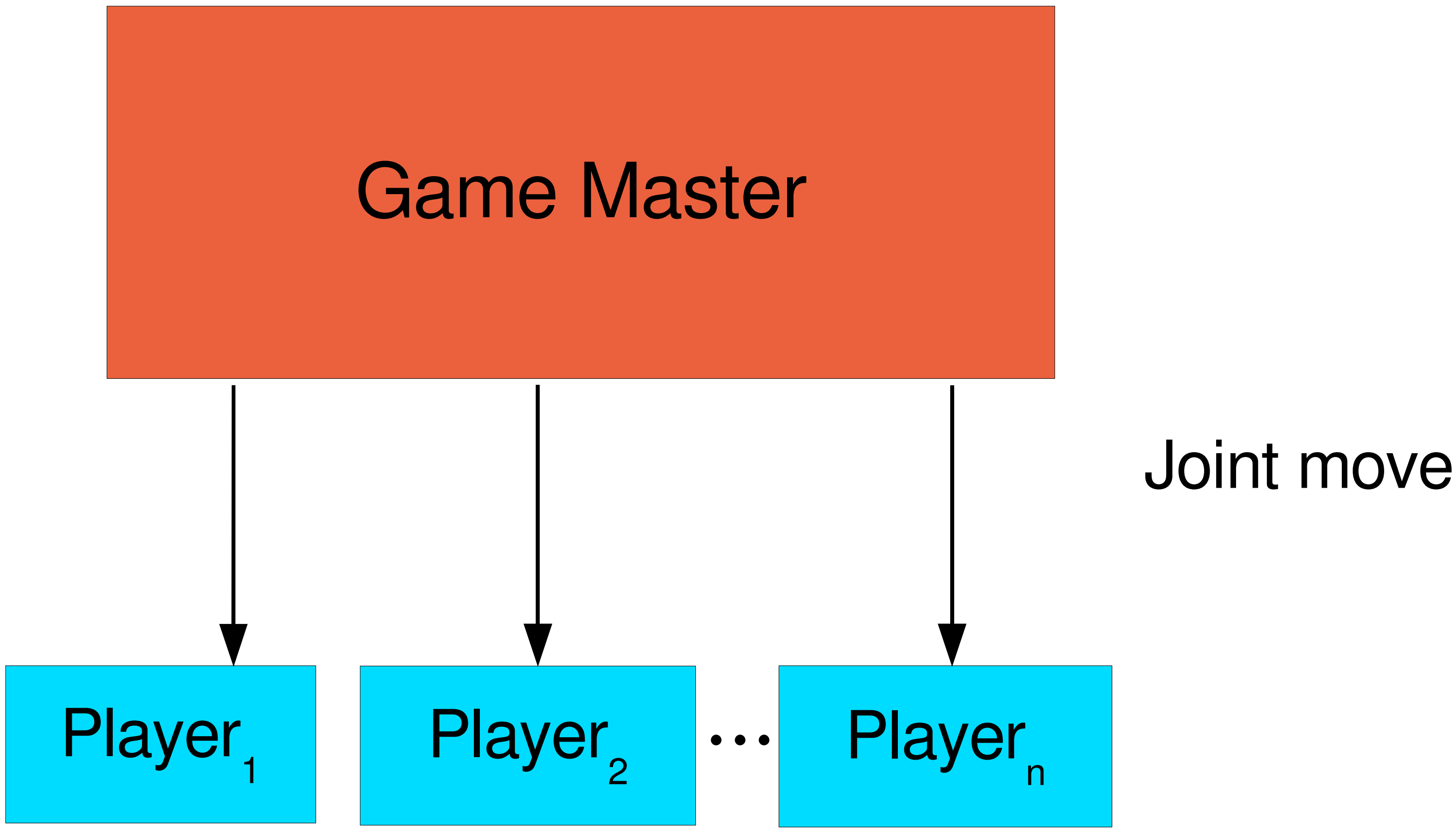
Actual Game Play



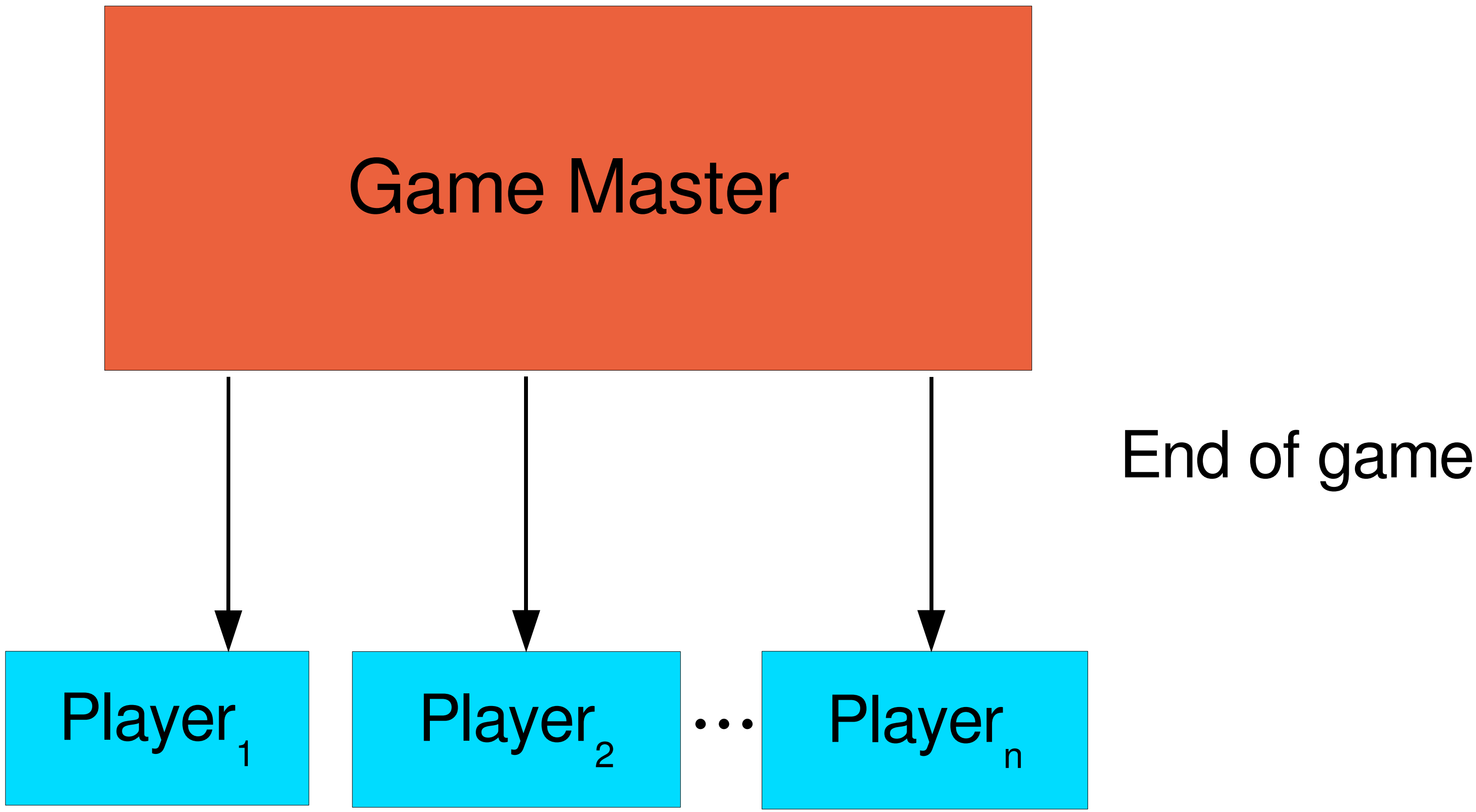
Actual Game Play



Actual Game Play



Actual Game Play



Demo: Bidding Tic-Tac-Toe

Towards Other Description Languages

- The GGP principle can be transferred to other areas
- A General Trading Agent is a system that
  - understands the rules of unknown market places
  - learns how to participate without human intervention
- A specification language for markets must account for
  - information asymmtery
  - asynchronous actions

→ introduce market maker + private message passing

Market Specification Language MDL

`trader(A)`	`A` is a trader
`message(A,M)`	trader `A` can send message `M`
`init(F)`	`F` holds in the initial state
`true(F)`	`F` holds in the current state
`next(F)`	`F` holds in the next state
`legal(A)`	market maker can do action `A`
`does(A)`	market maker does action `A`
`receive(A,M)`	receiving message `M` from trader `A`
`send(A,M)`	sending message `M` to trader `A`
`time(T)`	`T` is the current time
`terminal`	the market is closed

For details see [Thielscher & Zhang, 2009]

Example: Sealed-Bid Auction

```
trader(a_1)<=
...
trader(a_n)<=
message(A,my_bid(P)) <= trader(A) ∧ P ≥ 0

next(bid(A,P))    <= accept(bid(A,P))
accept(bid(A,P)) <= receive(A,my_bid(P)) ∧ time(1)
bestbid(A,P)      <= true(bid(A,P)) ∧ ¬outbid(P)
outbid(P)         <= true(bid(A,P1)) ∧ P1 > P

legal(clearing(A,P)) <= bestbid(A,P) ∧ time(2)

send(A,bid_accepted(P)) <= accept(bid(A,P))
send(A,winner(A1,P))    <= trader(A) ∧ does(clearing(A1,P))
terminal <= time(3)
```

Reasoning about Game Descriptions

The Value of Knowledge

Knowledge-based players try to extract and prove useful knowledge about a game from the mere rules

Some examples of potentially useful game-specific knowledge

- The game is strictly turn-based
- Each board cell (`X,Y`) has a unique contents `M`
- Markers `x` and `o` in Tic-Tac-Toe are permanent

Players systematically search for such properties and use them, eg. to improve their search or to generate an evaluation function

How to Verify Game-Specific Properties

- One approach is to run a number of random games and see if the property never gets violated
- More reliable--and often even more efficient--is to actually prove that the game rules entail the property
- Proof by induction: the property holds initially, and whenever it is true it also holds after a legal joint move

Induction Proofs: Example

Claim

Fluent `control` has a unique argument in every reachable position

P:
```
init(control(xplayer)) <=
next(control(xplayer)) <= true(control(oplayer))
next(control(oplayer)) <= true(control(xplayer))
```

The claim holds if `P` implies that

- uniqueness holds `init`; and
- uniqueness holds `next`,
  provided it is `true` (and every player makes a legal move)

Induction Proofs by Answer Set Programming

ASP is an established method to compute models of logic programs.
Efficient off-the-shelf implementations can be used.
Proof by contradiction: claim follows if its negation admits no model.

P **U**

```
h0 <= 1{init(control(X)): control_dom(X)}1
<= h0
```

weight atom

admits no answer set; same for

P **U**

```
1{true(control(X)): control_dom(X)}1 <=
h <= 1{next(control(X)): control_dom(X)}1
<= h
```

constraint

Another Example

Claim

Every board cell has a unique contents

Let $\mathbb{P}$ be the GDL rules for Tic-Tac-Toe.

$\mathbb{P}$ **U**

```
h0(X,Y) <= 1{init(cell(X,Y,Z)): c_dom(Z)}1
h0 <= ¬h0(X,Y)
<= ¬h0
```

admits no answer set

Another Example (cont'd)

For the induction step, uniqueness of `control` must be known!

P U

```
1{true(control(X)): control_dom(X)}1 <=
```

```
1{does(R,A): move_dom(A)}1 <=
<= does(R,A) ∧ ¬legal(R,A)
```

```
1{true(cell(X,Y,Z)): c_dom(Z)}1 <=
h(X,Y) <= 1{next(cell(X,Y,Z)): c_dom(Z)}1
h <= ¬h(X,Y)
<= ¬h
```

admits no answer set.

For details see [Schiffel & Thielscher, 2009b]

General Search Techniques for Games

- Single-player games: iterative deepening, non-uniform, ie. nodes with high estimated values searched deeper
- Transposition tables to store (position,evaluation)-pairs
- Two-player, zero-sum games with alternating moves: standard minimax with α-β-cutoffs
- Simultaneous moves, non-zero sum, *n*-player games:
  - paranoid search (opponents choose worst move for us)
  - computing equilibria (game theory)

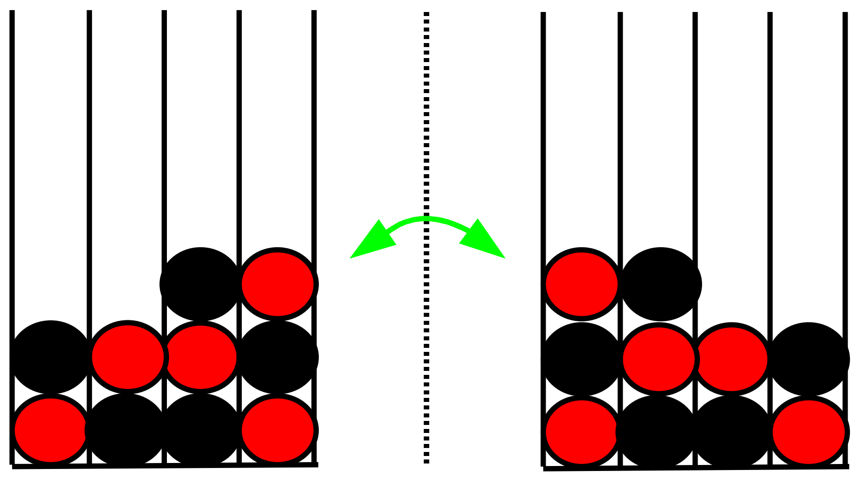
Using Knowledge for Search: Symmetry

Symmetries can be logically derived from the rules of a game

A symmetry relation over the elements of a domain is an equivalence relation such that

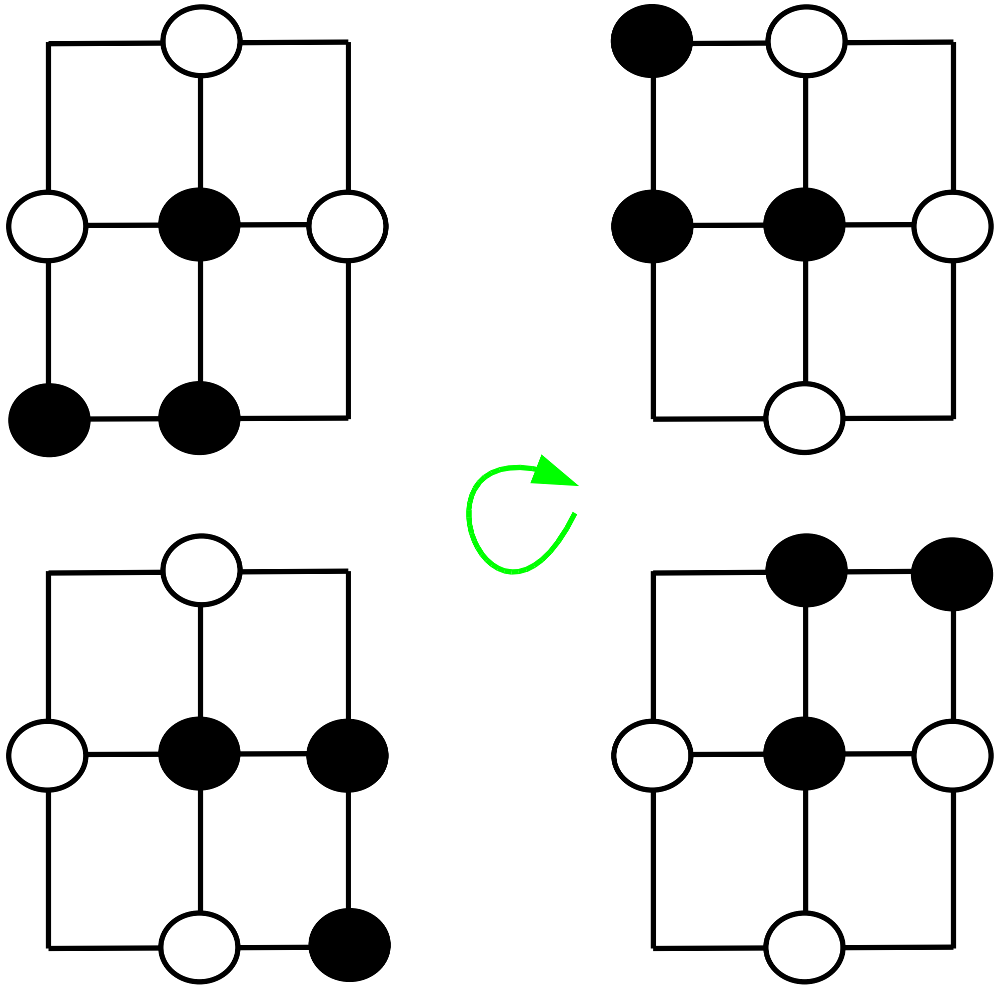
- two symmetric states are either both terminal or non-terminal
- if they are terminal, they have the same goal value
- if they are non-terminal, the legal moves in each of them are symmetric and yield symmetric states

Reflectional Symmetry



Connect-3

Rotational Symmetry

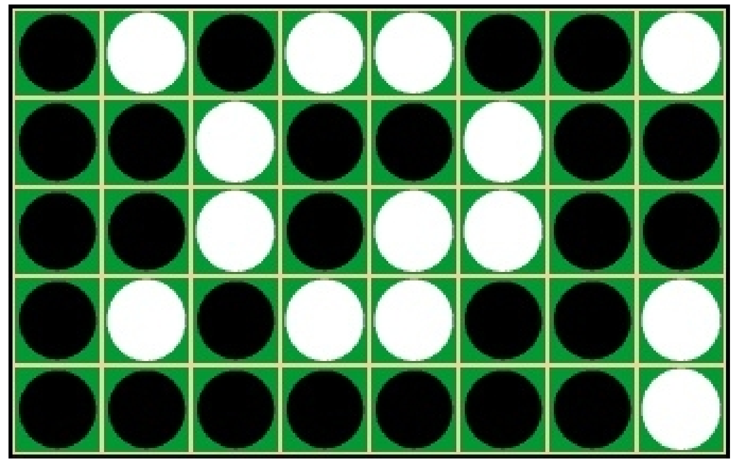


Capture-Go

Using Knowledge for Search: Factoring

Hodgepodge = Chess + Othello

Branching factor: a



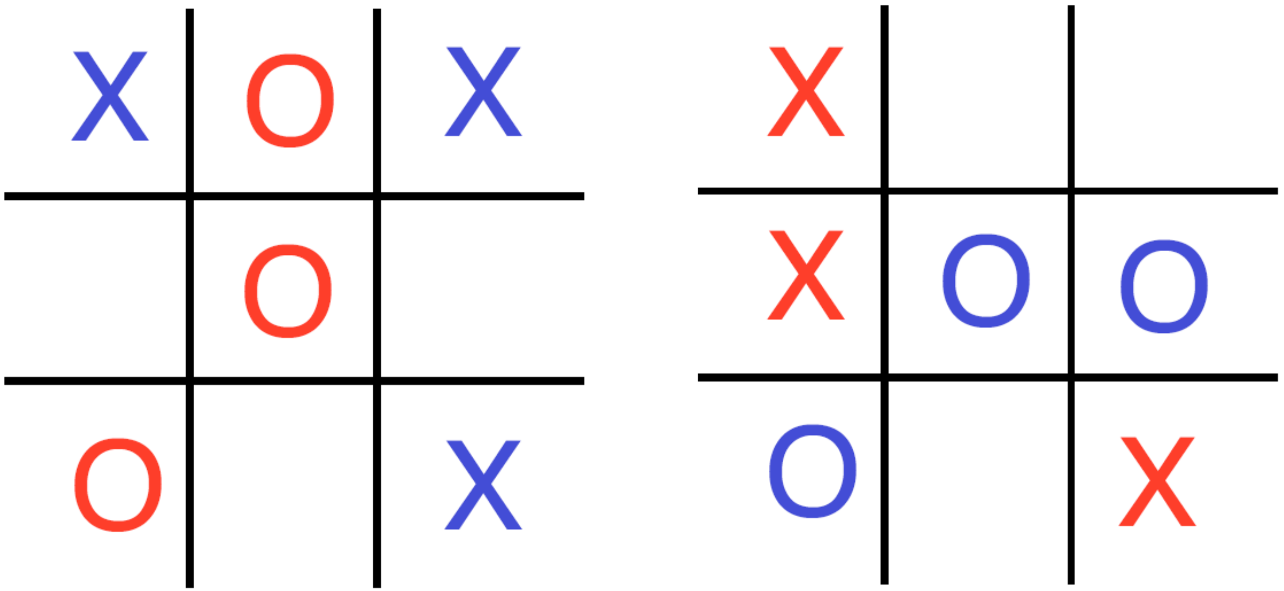
Branching factor: b

Branching factor as given to players: $a \cdot b$

Fringe of tree at depth n as given: $(a \cdot b)^n$

Fringe of tree at depth n if factored: $a^n + b^n$

Double Tic-Tac-Toe



Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1

Branching factor (after factoring): 18, 16, 14, 12, 10, 8, 6, 4, 1

Generating Evaluation Functions

Automatically Generated Evaluation Functions

Besides efficient inference and search algorithms, the ability to automatically generate a good evaluation function distinguishes good from bad general game players

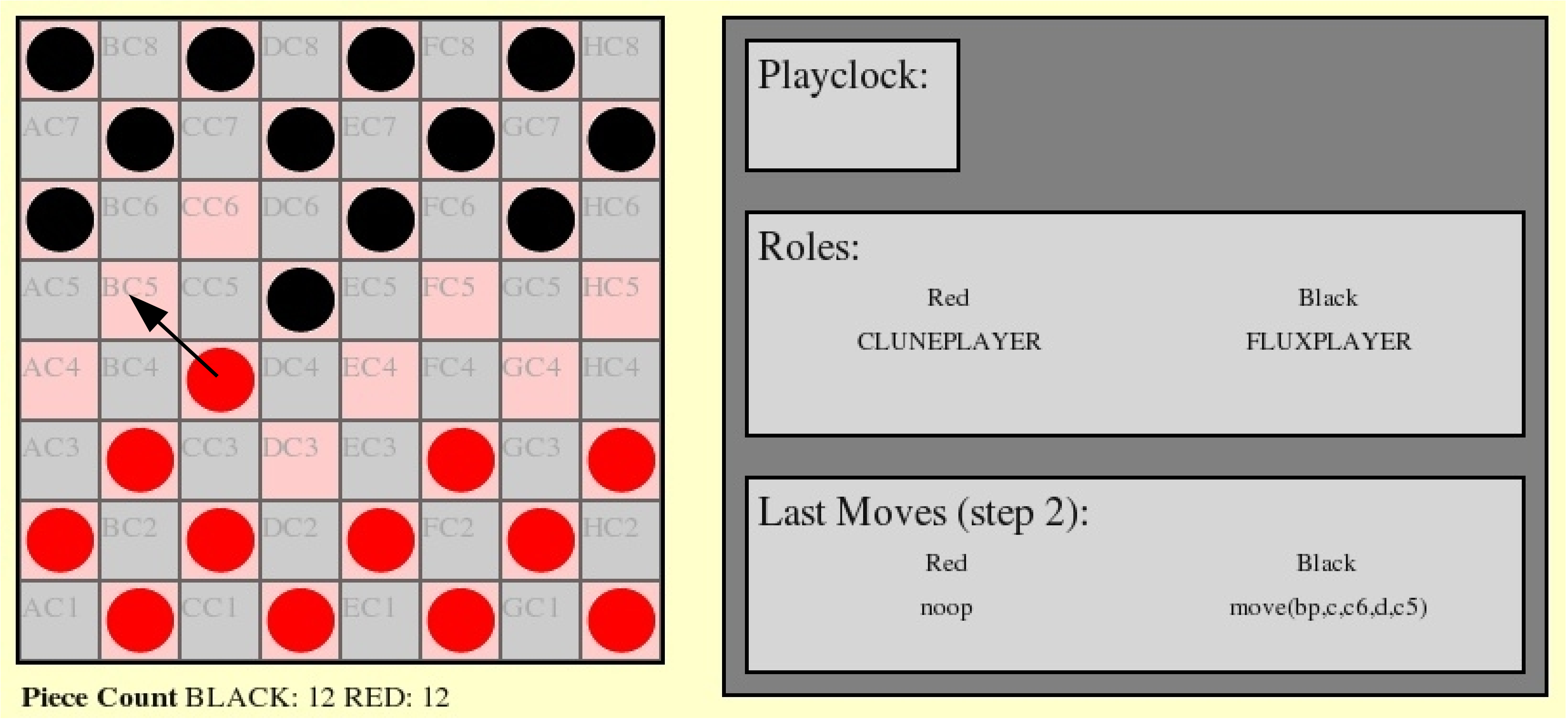
Approaches

- General heuristics: Mobility heuristics, Novelty heuristics, ...
- Recognizing structures: boards, pieces, piece values, ...
- Fuzzy Goal Evaluation

Mobility Heuristics

- Idea
  More moves means better state
- Advantage
  Often, being cornered or forced into making a move is quite bad
  - In Chess, having fewer moves means having fewer pieces or pieces of lower value
  - In Othello, having few moves means you have little control of the board
- Disadvantage
  Mobility is bad for some games

Example: Worldcup 2006 Final



Checkers (on a cylindrical board) with standard “forced capture“ rule

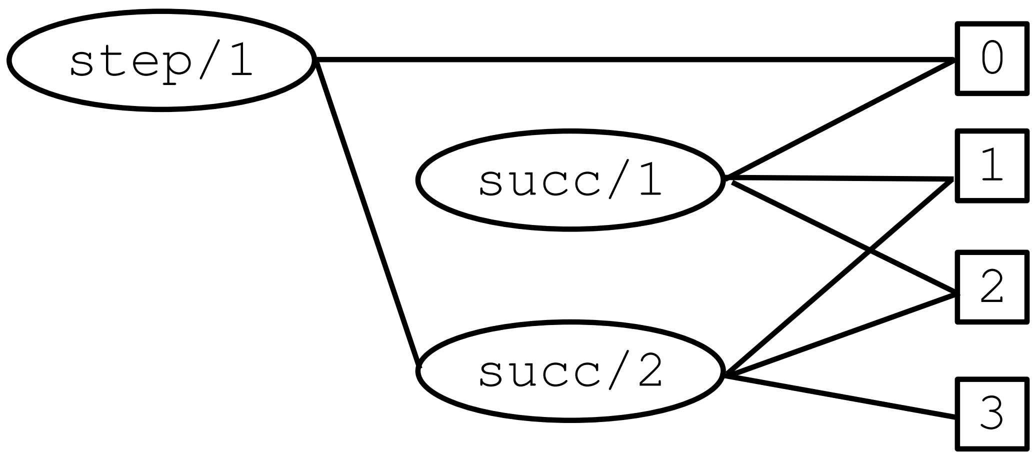
Novelty Heuristics

- Idea
  Changing the game state is better
- Advantage
  - Changing things as much as possible can help avoid getting stuck
  - When it is unclear what to do, maybe the best thing is to throw in some controlled randomness
- Disadvantage
  - Game state can also change if you just throw away own pieces
  - Unclear if novelty per se actually goes anywhere useful

Identifying Structures: Domains

- Domains of fluents identified by dependency graph

```
succ(0,1) ∧ succ(1,2) ∧ succ(2,3)
init(step(0))
next(step(X)) <= true(step(Y)) ∧ succ(Y,X)
```



Identifying Structures: Relations

A successor relation is a binary relation that is antisymmetric, functional, and injective

Example

```
succ(1,2) ∧ succ(2,3) ∧ succ(3,4) ∧ ...
next(a,b) ∧ next(b,c) ∧ next(c,d) ∧ ...
```

An order relation is a binary relation that is antisymmetric and transitive

Example

```
lessthan(A,B) <= succ(A,B)
lessthan(A,C) <= succ(A,B) ∧ lessthan(B,C)
```

Boards and Pieces

An (*m*-dimensional) board is an *n*-ary fluent ($n \geq m+1$) with

- *m* arguments whose domains are successor relations
- 1 output argument

Example

```
cell(a,1,whiterook) ∧ cell(b,1,whiteknight) ∧ ...
```

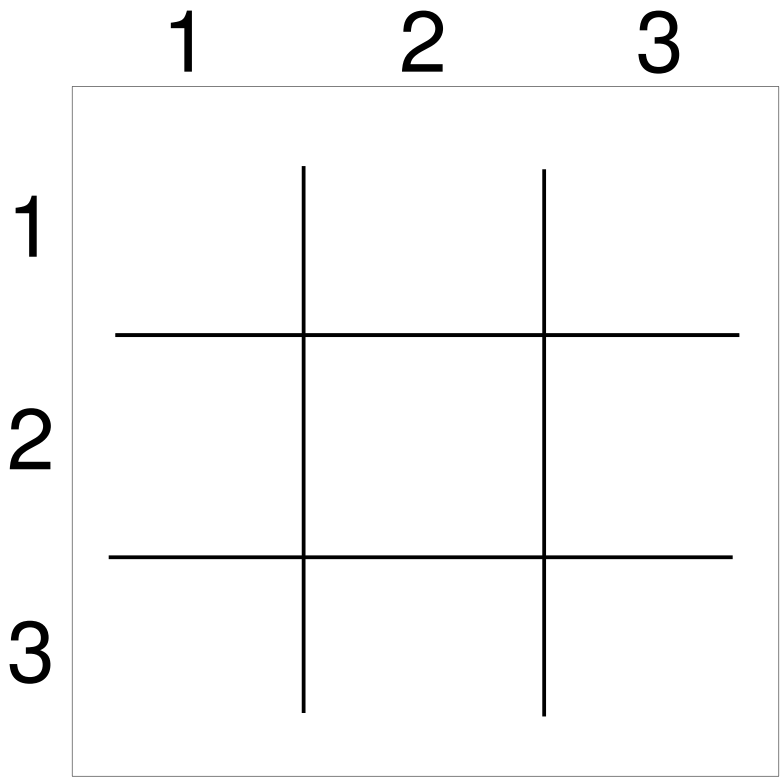
A marker is an element of the domain of a board's output argument

A piece is a marker which is in at most one board cell at a time

Example: Pebbles in Othello, White King in Chess

For details see [Clune, 2007]

Fuzzy Goal Evaluation: Example




```
goal(xplayer,100)<= line(x)

line(P) <= row(P)      V
           column(P)   V
           diagonal(P)
```

Value of intermediate state = Degree to which it satisfies the goal

Full Goal Specification

```
goal(xplayer,100) <= line(x)

line(P)      <= row(P) ∨ column(P) ∨ diagonal(P)

row(P)       <= true(cell(1,Y,P)) ∧ true(cell(2,Y,P)) ∧
                true(cell(3,Y,P))

column(P)    <= true(cell(X,1,P)) ∧ true(cell(X,2,P)) ∧
                true(cell(X,3,P))

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

After Unfolding

```
goal(xplayer,100)
              <=  true(cell(1,Y,x)) ∧ true(cell(2,Y,x)) ∧
                  true(cell(3,Y,x))
                  ∨
                  true(cell(X,1,x)) ∧ true(cell(X,2,x)) ∧
                  true(cell(X,3,x))
                  ∨
                  true(cell(1,1,x)) ∧ true(cell(2,2,x)) ∧
                  true(cell(3,3,x))
                  ∨
                  true(cell(3,1,x)) ∧ true(cell(2,2,x)) ∧
                  true(cell(1,3,x))
```

3 literals are true after `does(x,mark(1,1))`

2 literals are true after `does(x,mark(1,2))`

4 literals are true after `does(x,mark(2,2))`

Fuzzy Goal Evaluation

- Use t-norms, eg. instances of the Yager family (with parameter *q*)

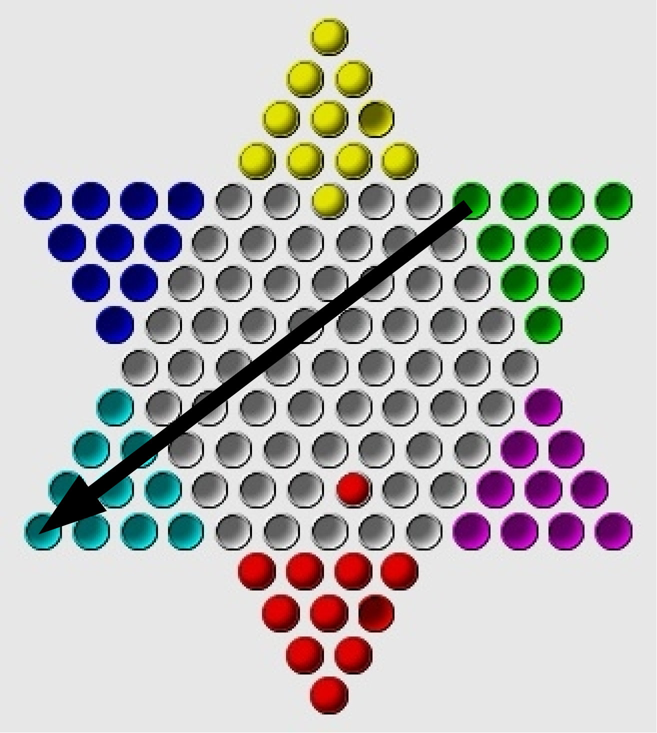
  T(a,b) = 1 – S(1-a,1-b)
  S(a,b) = (a^q + b^q) ^ (1/q)

- Evaluation function for formulas

  $eval(f \wedge g) = T(eval(f),eval(g))$
  $eval(f \vee g) = S(eval(f),eval(g))$
  $eval(\neg f) = 1 - eval(f)$

Advanced Fuzzy Goal Evaluation: Example

(j,13)



(e,5)

```
init(cell(green,j,13))∧ ...

goal(green_player,100)
    <= true(cell(green,e,5))
       ∧...
```

Truth degree of goal literal = (Distance to current value)$^{-1}$

Identifying Metrics

- Order relations Binary, antisymmetric, functional, injective

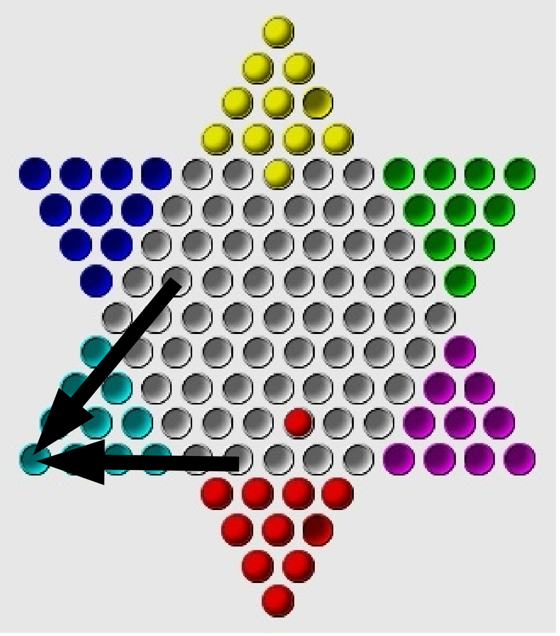
```
succ(1,2).   succ(2,3).   succ(3,4).
file(a,b).   file(b,c).   file(c,d).
```

- Order relations define a metric on functional features

Δ(cell(green,j,13),cell(green,e,5)) = 13

Degree to which *f(x,a)* is true given that *f(x,b)*

(1-p) - (1-p) * Δ(b,a) / |dom(f(x))|



With p=0.9, *eval*(`cell(green,e,5)`) is

0.082 if `true(cell(green,f,10))`

0.085 if `true(cell(green,j,5))`

Assessment

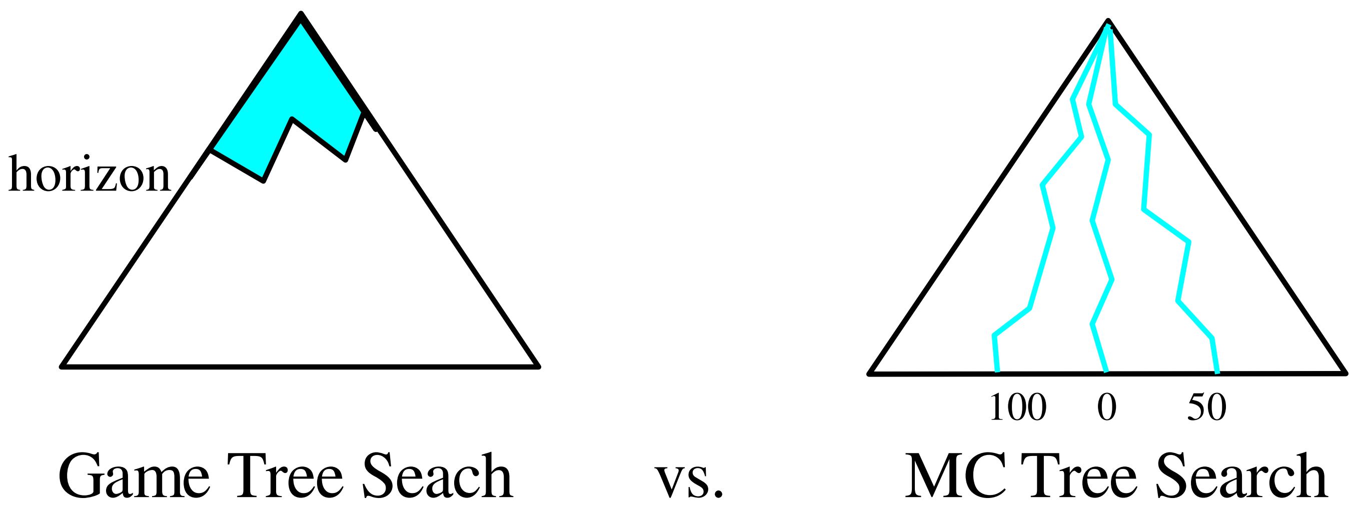
Fuzzy goal evaluation works particularly well for games with

- independent sub-goals
  - 15-Puzzle
- converge to the goal
  - Chinese Checkers
- quantitative goal
  - Othello
- partial goals
  - Peg Jumping, Chinese Checkers with >2 players

For details see [Schiffel & Thielscher, 2007]

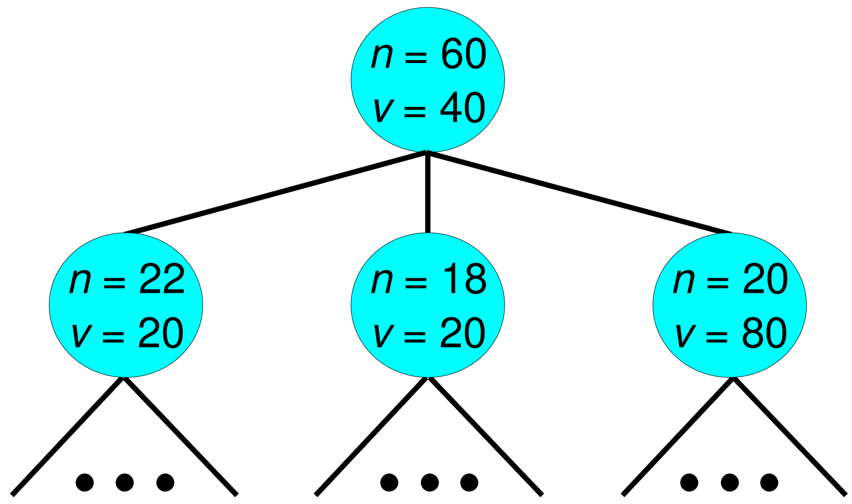
Learning by Simulation

Knowledge-Free General Game Playing: Monte Carlo Tree Search



Monte Carlo Tree Search

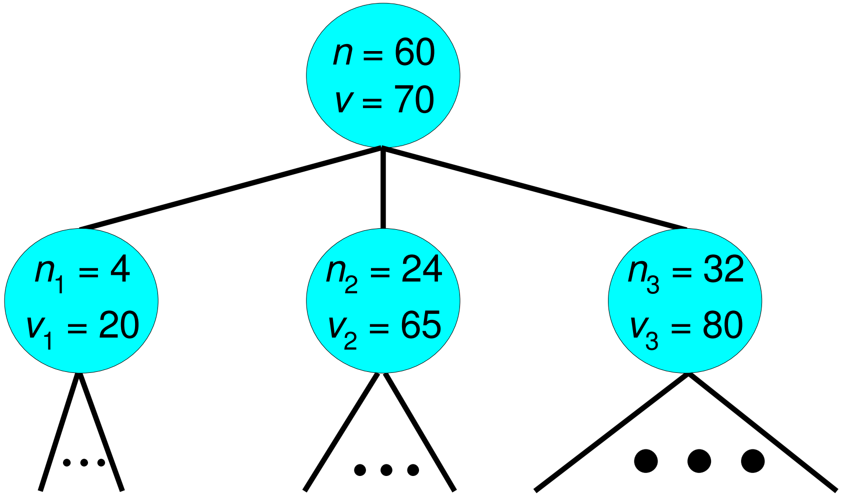
Value of move = Average score returned by simulation



Improvement: UCT Search

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

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



UCT = Upper Confidence bounds applied to Trees

Assessment

UCT Search works particularly well for games which

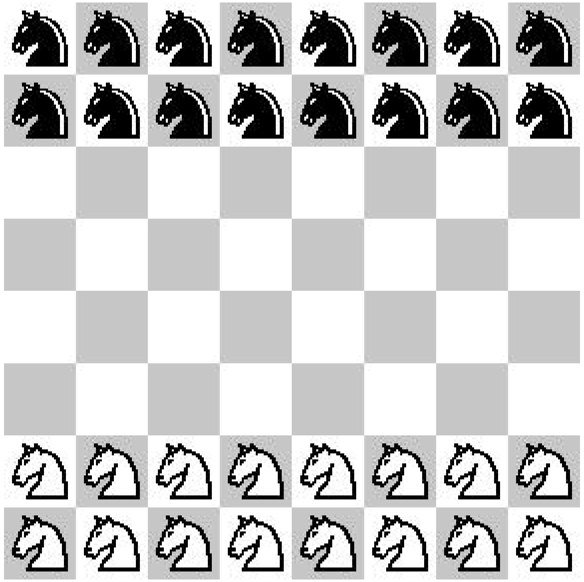
- reward greedy behavior
- do not require long-term strategies
- have a large branching factor
- are difficult for humans to play

For details see [Finnsson & Björnsson, 2008]

Demo: An Unstructered Game

Knowledge-Based vs. Simulation-Based (Championship 2008)

Demo: A Structured Game



Simulation-Based vs. Knowledge-Based (Championship 2008)

Summary

The GGP Challenge

Much like RoboCup, General Game Playing

- combines a variety of AI areas
- fosters developmental research
- has great public appeal
- has the potential to significantly advance AI

In contrast to RoboCup, GGP has the advantage to

- focus on high-level intelligence
- have low entry cost
- make a great hands-on course for AI students

A Vision for GGP

Natural Language Understanding

- Rules of a game given in natural language

Computer Vision

- Vision system sees board, pieces, cards, rule book, ...

Robotics

- Robot playing the actual, physical game

Resources

- Stanford GGP initiative games.stanford.edu
  - GDL specification
  - Basic player
- GGP in Germany general-game-playing.de
  - Game master
  - 24/7 online game playing
  - Extensive collection of GGP literature
- Palamedes palamedes-ide.sourceforge.net
  - GGP/GDL development tool

Papers

[Clune, 2007]
J. Clune. Heuristic evaluation functions for general game playing. AAAI 2007

[Finnsson & Björnsson, 2008]
H. Finnsson, Y. Björnsson. Simulation-based approach to general game playing. AAAI 2008

[Genesereth, Love & Pell, 2006]
M. Genesereth, N. Love, B. Pell. General game playing. AI magazine 26(2), 2006

[Schiffel & Thielscher, 2007]
S. Schiffel, M. Thielscher. Fluxplayer: a successful general game player. AAAI 2007

[Schiffel & Thielscher, 2009a]
S. Schiffel, M. Thielscher. A multiagent semantics for the Game Description Language. ICAART 2009.

[Schiffel & Thielscher, 2009b]
S. Schiffel, M. Thielscher. Automated theorem proving for general game playing. IJCAI 2009.

[Thielscher & Zhang, 2009]
M. Thielscher, D. Zhang. From GDL to a market specification language for general trading agents. GIGA 2009.