General Game Playing
Metagaming

Michael Genesereth
Computer Science Department
Stanford University

Metagaming

Metagaming is match-independent game processing, i.e. game processing that is done independent of any particular opponent or any particular state.

Objective of metagaming - to optimize performance in playing specific matches of the game.

Usually done offline, i.e. during the *startclock* or *between moves or in parallel with regular game play*.

#1 - Headstart

Headstart = game tree search during the start clock.

```
function start (r,rs,sc,pc)
 {role = r;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = parseInt(sc);
  playclock = parseInt(pc);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,findcontrol(state,library),reward);
  return 'ready'}

function play (move)
 {if (move!==nil)
      {tree = subtree(move,tree); state = tree.state};
  if (findcontrol(state,library)!==role) {return false};
  var deadline = Date.now()+playclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return selectaction(tree)}
```

Headstart

```
function start (r,rs,sc,pc)
 {role = r;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = parseInt(sc);
  playclock = parseInt(pc);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,findcontrol(state,library),reward);
  var deadline = Date.now()+startclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return 'ready'}

function play (move)
 {if (move!==nil)
     {tree = subtree(move,tree); state = tree.state};
  if (findcontrol(state,library)!==role) {return false};
  var deadline = Date.now()+playclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return selectaction(tree)}
```

#2 - Bigswitch

Bigswitch is using the startclock to analyze a game and use the analysis to adjust runtime play.

(1) runtime players (e.g. iterative deepening vs greedy)

(2) evaluation functions (e.g. mobility, intermediate values, depth charges)

(3) selection functions or coefficients (e.g. coefficients for exploration and exploitation)

How to Decide

Analyze rules *somehow*

Run some depth charges:

- Check for intermediate state values
- Branching factor
- Depth
- Cost of expanding

#3 Rewrite Rules

Source-to-Source Transformations

changing game descriptions before runtime

possibly changing players

(a) Logical Optimization (week 6 - today!)

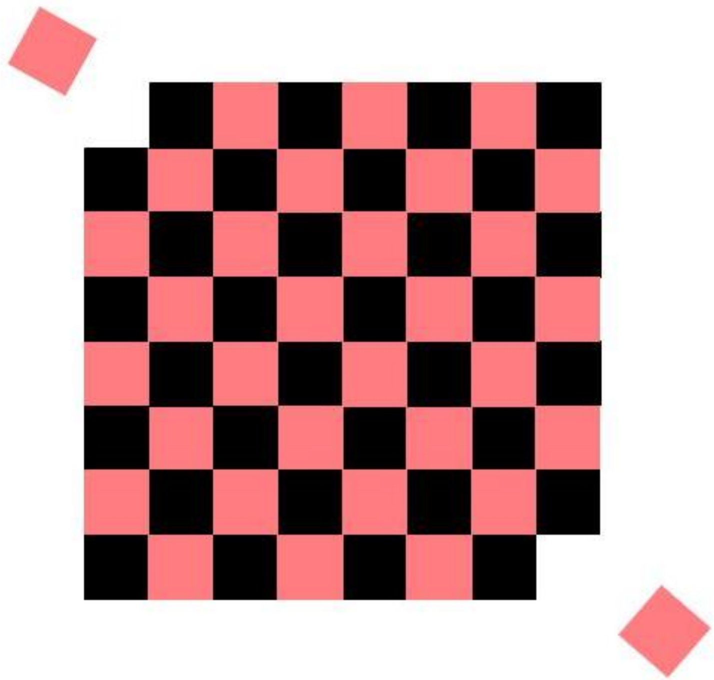
(b) Game Tree Restructuring (week 7)

(c) Game Grounding, Symbolizing, and Vectorizing (week 8)

#4 - Symbolic Reasoning

Symbolic Reasoning (week 9)

- Automatic Theorem Proving
- Automatic Programming



Logical Simplification

Example

```
 SEND
+MORE
-----
MONEY
```

Solution

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)

solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) &
  S!=0 & E!=S & N!=S & N!=E & D!=S & D!=E & D!=N &
  M!=0 & M!=S & M!=E & M!=N & M!=D &
  O!=S & O!=E & O!=N & O!=D & O!=M &
  R!=S & R!=E & R!=N & R!=D & R!=M & R!=O &
  Y!=S & Y!=E & Y!=N & Y!=D & Y!=M & Y!=O & Y!=R
  evaluate(S*1000+E*100+N*10+D,U) &
  evaluate(M*1000+O*100+R*10+E,V) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,W) &
  evaluate(plus(U,V),W)
```

Analysis

Data

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

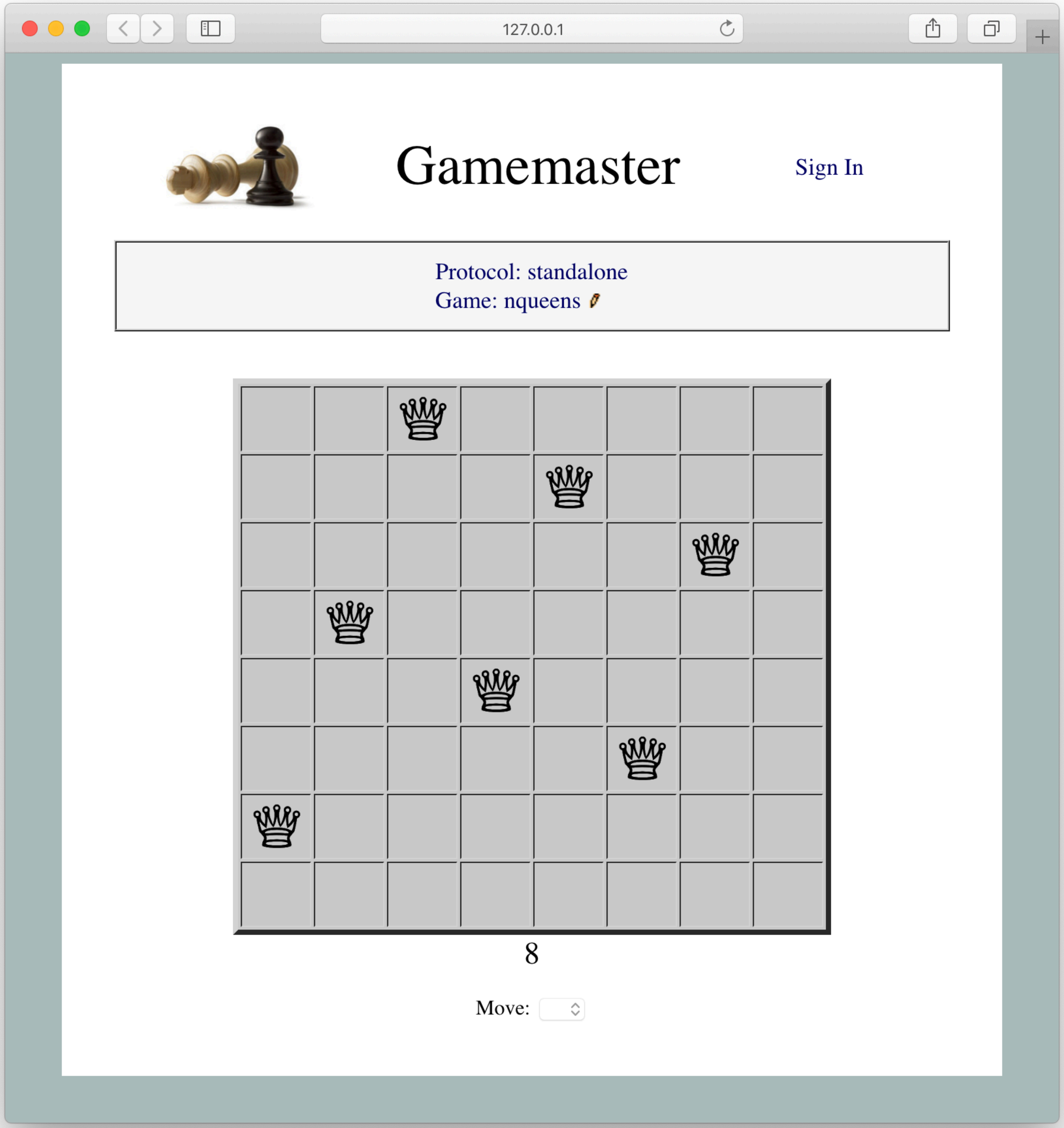
Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) & ...
```

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 instances

8 Queens


127.0.0.1
Gamemaster
Sign In
Protocol: standalone
Game: nqueens
8
Move:

Goal Description

```
digit(1)      digit(5)
digit(2)      digit(6)
digit(3)      digit(7)
digit(4)      digit(8)

base(cell(X,Y)) :- digit(X) & digit(Y)

solution(C1,C2,C3,C4,C5,C6,C7,C8) :-
  cell(C1) & cell(C2) & cell(C3) & cell(C4) &
  cell(C5) & cell(C6) & cell(C7) & cell(C8) &
  ~attacks(C1,C2) & ... & ~attacks(C7,C8)
```

Analysis

64 x 64 x 64 x 64 x 64 x 64 x 64 x 64 =

281,474,976,710,656 instances

Subgoal Ordering

Original Rule

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
```

Reformulation

```
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

Analysis

Original Rule

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
```

$$(n^2 + 2n) + n*((n^2 + 2n) + n*(n^2 + 2n)) =$$
$$n^4 + 3n^3 + 3n^2 + 2n$$

Reformulation

```
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

$$(n^2 + 2n) + n*((n^2 + 2n) + 1*(n^2 + 2n)) =$$
$$2n^3 + 5n^2 + 2n$$

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :-
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X)
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X) & q(X)
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

Example

```
 SEND
+MORE
-----
MONEY
```

One Solution

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) &
  S!=0 & E!=S & N!=S & N!=E & D!=S & D!=E & D!=N &
  M!=0 & M!=S & M!=E & M!=N & M!=D &
  O!=S & O!=E & O!=N & O!=D & O!=M &
  R!=S & R!=E & R!=N & R!=D & R!=M & R!=O &
  Y!=S & Y!=E & Y!=N & Y!=D & Y!=M & Y!=O & Y!=R
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
```

Computational Analysis

Data

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

Rule

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) & ...
```

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 cases

111,111,110 unifications

Running time ~ minutes

Subgoals Reordered

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R &
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
```

Computational Analysis

Rule

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R & ...
```

Analysis

10x9x8x7x6x5x4x3 = 1,814,400 cases

5,989,558 / 7,921,010 unifications

Interpreted ~ 40 seconds   Compiled ~ 4 seconds

Computational Analysis

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(D) &
  digit(E) & mutex(E,D) &
  digit(Y) & mutex(Y,D,E) &
             evaluate(remainder(plus(D,E),10),Y) &
  digit(S) & distinct(S,0) & mutex(S,D,E,Y) &
  digit(N) & mutex(N,D,E,Y,S) &
  digit(M) & distinct(M,0) & mutex(M,D,E,Y,S,N) &
  digit(O) & mutex(O,D,E,Y,S,N,M) &
  digit(R) & mutex(R,D,E,Y,S,N,M,O) & ...
```

Analysis

438,728 / 7,921,010 unifications

Interpreted ~ 4 seconds

Subgoal Removal

Original Rule:

```
solution(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

Equivalent Reformulation:

```
solution(X,Y) :- p(X,Y) & q(Y)
```

Rule Removal

Redundant Rules:

```
solution(X) :- p(X,b) & q(b) & r(Z)
solution(X) :- p(X,Y) & q(Y) & r(Z)
```

Non-Redundant Rules:

```
solution(X) :- p(X,b) & q(b) & r(Z)
solution(X) :- p(X,Y) & q(Y) & r(c)
```

127.0.0.1

Standard Players

Legal - Legal player

Random - Random player

Onestep - One Step player using intermediate values

Twostep - Two Step player using intermediate values

Minimax - Full Minimax player using intermediate values

Minimaxdepth - Minimax player to fixed depth using intermediate values

Minimaxid - Minimax player with iterative deepening using intermediate values

MCS - Monte Carlo Search - one step player using depth charges

PTS - Minimax player with persistent breadth-first search using intermediate values

Greedy - PTS player with search based on exploration and exploitation

---

Standard Metagamers

Optimizer - performs optimizations on game descriptions.

Materializer - materializes relations used in game descriptions.

Simplifier - simplifies games by eliminating subgoals or rules based on ground facts in the game description.

Grounder - converts game descriptions with variables to fully grounded versions.

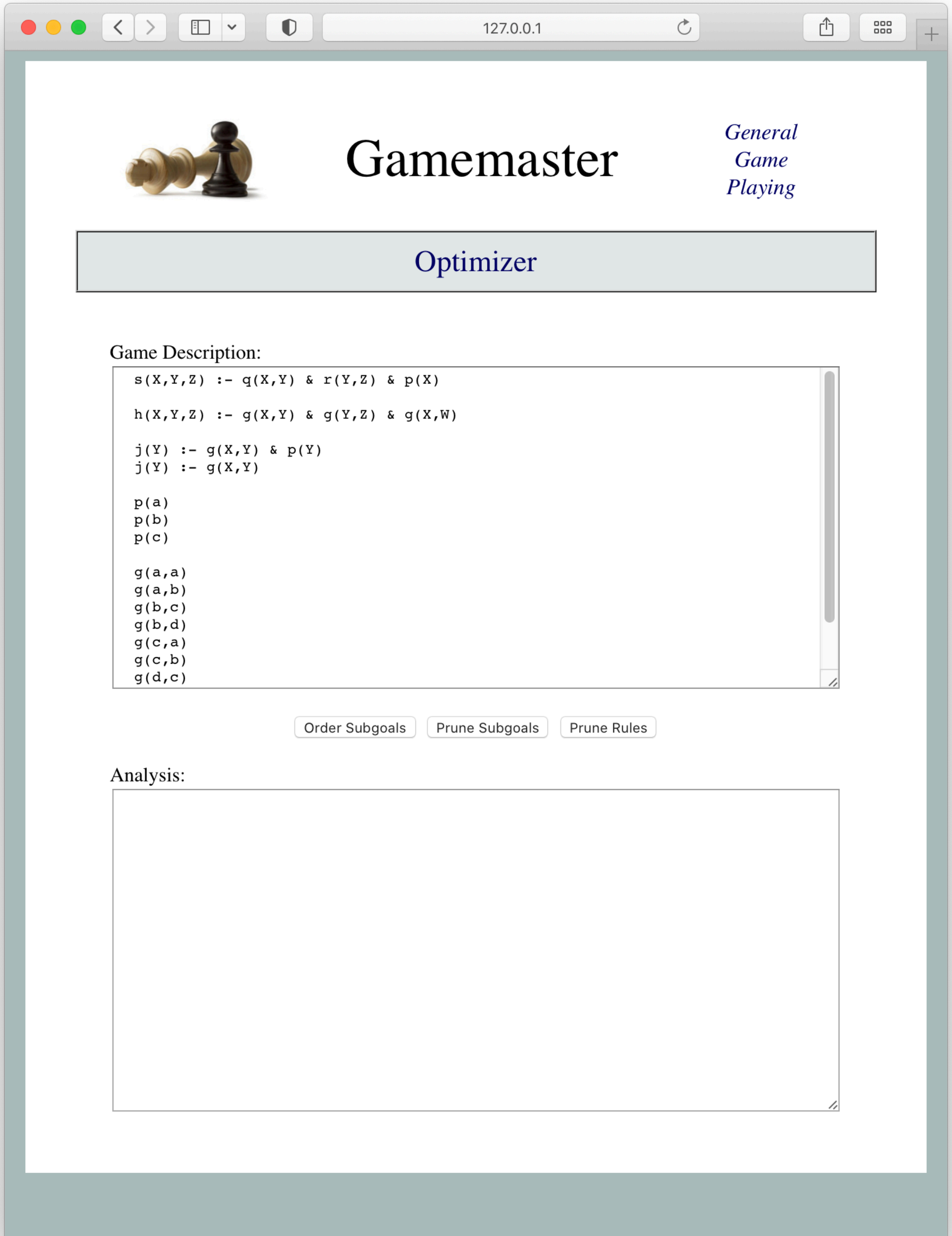
Symbolizer - converts game descriptions with variables to fully grounded and symbolized versions (i.e. with all ground atoms converted to propositions).

Pruner - prunes games to include only potentially relevant actions based on dependency analysis of fully grounded game descriptions.

Older General Game Playing Websites

Tiltyard (web site) - allows users to register players for automatic round robin competition against other general game playing programs. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.

GGP.org (web page) - General website on GGP. Contains information on how to develop software for GGP. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.


127.0.0.1
Gamemaster
General
Game
Playing
Optimizer
Game Description:
s(X,Y,Z) :- q(X,Y) & r(Y,Z) & p(X)
h(X,Y,Z) :- g(X,Y) & g(Y,Z) & g(X,W)
j(Y) :- g(X,Y) & p(Y)
j(Y) :- g(X,Y)
p(a)
p(b)
p(c)
g(a,a)
g(a,b)
g(b,c)
g(b,d)
g(c,a)
g(c,b)
g(d,c)
Order Subgoals
Prune Subgoals
Prune Rules
Analysis:

Loading Metagaming Code

```
<script src='http://epilog.stanford.edu/javascript/
epilog.js'></script>

<script src='http://gamemaster.stanford.edu/javascript/
localstorage.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/
optimizer.js'></script>

<script src='http://gamemaster.stanford.edu/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
general.js'></script>
```

Usage

```
function start (r,rs,sc,pc)
 {role = r;
  rules = rs.slice(1);
  startclock = numberize(sc);
  playclock = numberize(pc);

  rules = prunerulesubgoals(rules);
  rules = prunerules(rules);
  rules = fixrules(rules);

  library = definemorerules([],rs);
  roles = findroles(library);
  state = findinits(library);
  var active = findcontrol(state,library);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,active,reward,false);
  return 'ready'}
```

Materialization

Basic Idea

Materialization is the process of precomputing a view relation and storing the result so that subsequent queries can be answered by lookup rather than computation.

Advantage: Look up is often much cheaper than computation. Efficiency advantage if the relation is queried often.

Disadvantages: Relation may be *large* (e.g. third generation relatives of US population) or even *infinite* (e.g. the set of all primes) and/or *expensive* to compute (e.g. factors of a large number) and may be accessed only infrequently.

Example

```
goal(robot,0)   :- cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100) :- cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(M,N) & distinct(M,N)

solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) :-
  index(X1) & index(Y1) &
  index(X2) & index(Y2) &
  index(X3) & index(Y3) &
  index(X4) & index(Y4) &
  index(X5) & index(Y5) &
  ~queenmove(X1,Y1,X2,Y2) &
  ~queenmove(X1,Y1,X3,Y3) &
  ~queenmove(X1,Y1,X4,Y4) &
  ~queenmove(X1,Y1,X5,Y5) &
  ~queenmove(X2,Y2,X3,Y3) &
  ~queenmove(X2,Y2,X4,Y4) &
  ~queenmove(X2,Y2,X5,Y5) &
  ~queenmove(X3,Y3,X4,Y4) &
  ~queenmove(X3,Y3,X5,Y5) &
  ~queenmove(X4,Y4,X5,Y5)
```

Example

```
goal(robot,0)   :- cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100) :- cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(M,N) & distinct(M,N)

solution(1,1,2,3,3,5,4,2,5,4)
...
```

127.0.0.1

Standard Players

Legal - Legal player

Random - Random player

Onestep - One Step player using intermediate values

Twostep - Two Step player using intermediate values

Minimax - Full Minimax player using intermediate values

Minimaxdepth - Minimax player to fixed depth using intermediate values

Minimaxid - Minimax player with iterative deepening using intermediate values

MCS - Monte Carlo Search - one step player using depth charges

PTS - Minimax player with persistent breadth-first search using intermediate values

Greedy - PTS player with search based on exploration and exploitation

---

Standard Metagamers

Optimizer - performs optimizations on game descriptions.

Materializer - materializes relations used in game descriptions.

Simplifier - simplifies games by eliminating subgoals or rules based on ground facts in the game description.

Grounder - converts game descriptions with variables to fully grounded versions.

Symbolizer - converts game descriptions with variables to fully grounded and symbolized versions (i.e. with all ground atoms converted to propositions).

Pruner - prunes games to include only potentially relevant actions based on dependency analysis of fully grounded game descriptions.

Older General Game Playing Websites

Tiltyard (web site) - allows users to register players for automatic round robin competition against other general game playing programs. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.

GGP.org (web page) - General website on GGP. Contains information on how to develop software for GGP. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.


127.0.0.1
Gamemaster
General
Game
Playing
Materializer
Game Description:
role(robot)
base(f(X,Y)) :- g(X,Y)
base(e(X,Y)) :- g(X,Y)
action(doit(a))
init(f(a,b))
init(e(a,b))
s(X,Y,Z) :- q(X,Y) & r(Y,Z) & p(X)
h(X,Y,Z) :- g(X,Y) & g(Y,Z) & j(Z)
j(Y) :- g(X,Y) & k(Y)
k(Y) :- f(X,Y)
l(Y) :- g(Y,Y)
g(a,a)
g(a,b)
Static Bases
Static Views
Dynamic Bases
Dynamic Views
Materialize Static Views
Materialize Dynamic Views
Analysis:

Loading Metagaming Code

```
<script src='http://epilog.stanford.edu/javascript/
epilog.js'></script>

<script src='http://gamemaster.stanford.edu/javascript/
localstorage.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/
materializer.js'></script>

<script src='http://gamemaster.stanford.edu/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
general.js'></script>
```

Usage

```
function start (r,rs,sc,pc)
 {role = r;
  rules = rs.slice(1);
  startclock = numberize(sc);
  playclock = numberize(pc);

  rules = materializestaticrelations(rules);

  library = definemorerules([],rs);
  roles = findroles(library);
  state = findinits(library);
  var active = findcontrol(state,library);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,active,reward,false);
  return 'ready'}
```

Simplification

Basic Idea

Simplification is the process of substituting static facts into rules and eliminating true subgoals and eliminating rules with false subgoals.

Example

```
h(a,b,c) :- g(a,b) & g(b,c)

j(a) :- g(a,a) & k(a)
j(b) :- g(a,b) & k(b)
j(c) :- g(b,c) & k(c)
j(a) :- g(c,c) & k(a)
j(b) :- g(c,b) & k(b)

k(a) :- f(a,a)
k(b) :- f(a,b)
k(c) :- f(b,c)

g(a,b)
g(b,c)
g(c,a)
g(c,b)
```

Example

```
h(a,b,c) :- g(a,b) & g(b,c)

j(a) :- g(a,a) & k(a)
j(b) :- g(a,b) & k(b)
j(c) :- g(b,c) & k(c)
j(a) :- g(c,c) & k(a)
j(b) :- g(c,b) & k(b)

k(a) :- f(a,a)
k(b) :- f(a,b)
k(c) :- f(b,c)

g(a,b)
g(b,c)
g(c,a)
g(c,b)
```

```
h(a,b,c)

j(b) :- k(b)
j(c) :- k(c)

k(a) :- f(a,a)
k(b) :- f(a,b)
k(c) :- f(b,c)

g(a,b)
g(b,c)
g(c,a)
g(c,b)
```

Example of Materialization + Simplification

```
goal(robot,0)   :- cell(1,1) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(2,2) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,100) :- cell(3,3) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(4,4) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(5,5) & solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5)
goal(robot,0)   :- cell(M,N) & distinct(M,N)

solution(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) :-
  index(X1) & index(Y1) &
  index(X2) & index(Y2) &
  index(X3) & index(Y3) &
  index(X4) & index(Y4) &
  index(X5) & index(Y5) &
  ~queenmove(X1,Y1,X2,Y2) &
  ~queenmove(X1,Y1,X3,Y3) &
  ~queenmove(X1,Y1,X4,Y4) &
  ~queenmove(X1,Y1,X5,Y5) &
  ~queenmove(X2,Y2,X3,Y3) &
  ~queenmove(X2,Y2,X4,Y4) &
  ~queenmove(X2,Y2,X5,Y5) &
  ~queenmove(X3,Y3,X4,Y4) &
  ~queenmove(X3,Y3,X5,Y5) &
  ~queenmove(X4,Y4,X5,Y5)
```

Example of Materialization + Simplification

```
goal(robot,0)   :- cell(1,1)
goal(robot,0)   :- cell(2,2)
goal(robot,100) :- cell(3,3)
goal(robot,0)   :- cell(4,4)
goal(robot,0)   :- cell(5,5)
goal(robot,0)   :- cell(M,N) & distinct(M,N)
```

127.0.0.1

Standard Players

Legal - Legal player

Random - Random player

Onestep - One Step player using intermediate values

Twostep - Two Step player using intermediate values

Minimax - Full Minimax player using intermediate values

Minimaxdepth - Minimax player to fixed depth using intermediate values

Minimaxid - Minimax player with iterative deepening using intermediate values

MCS - Monte Carlo Search - one step player using depth charges

PTS - Minimax player with persistent breadth-first search using intermediate values

Greedy - PTS player with search based on exploration and exploitation

---

Standard Metagamers

Optimizer - performs optimizations on game descriptions.

Materializer - materializes relations used in game descriptions.

Simplifier - simplifies games by eliminating subgoals or rules based on ground facts in the game description.

Grounder - converts game descriptions with variables to fully grounded versions.

Symbolizer - converts game descriptions with variables to fully grounded and symbolized versions (i.e. with all ground atoms converted to propositions).

Pruner - prunes games to include only potentially relevant actions based on dependency analysis of fully grounded game descriptions.

Older General Game Playing Websites

Tiltyard (web site) - allows users to register players for automatic round robin competition against other general game playing programs. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.

GGP.org (web page) - General website on GGP. Contains information on how to develop software for GGP. Warning: Games on this website do not necessarily comply with the current General Game Playing standard.

Questions and Comments

Gamemaster

General Game Playing

Simplifier

Game Description:

```
role(robot)
init(f(a,b))
init(e(a,b))
h(a,a,a) :- g(a,a) & g(a,a) & j(a)
h(a,a,b) :- g(a,a) & g(a,b) & j(b)
h(a,b,c) :- g(a,b) & g(b,c) & j(c)
h(a,b,d) :- g(a,b) & g(b,d) & j(d)
h(b,c,a) :- g(b,c) & g(c,a) & j(a)
h(b,c,b) :- g(b,c) & g(c,b) & j(b)
h(b,d,c) :- g(b,d) & g(d,c) & j(c)
h(b,d,d) :- g(b,d) & g(d,d) & j(d)
h(c,a,a) :- g(c,a) & g(a,a) & j(a)
h(c,a,b) :- g(c,a) & g(a,b) & j(b)
h(c,b,c) :- g(c,b) & g(b,c) & j(c)
h(c,b,d) :- g(c,b) & g(b,d) & j(d)
h(d,c,a) :- g(d,c) & g(c,a) & j(a)
h(d,c,b) :- g(d,c) & g(c,b) & j(b)
h(d,d,c) :- g(d,d) & g(d,c) & j(c)
```

Simplify

Result:

```
role(robot)
init(f(a,b))
init(e(a,b))
h(a,a,a) :- j(a)
h(a,a,b) :- j(b)
h(a,b,c) :- j(c)
h(a,b,d) :- j(d)
h(b,c,a) :- j(a)
h(b,c,b) :- j(b)
h(b,d,c) :- j(c)
h(b,d,d) :- j(d)
h(c,a,a) :- j(a)
h(c,a,b) :- j(b)
h(c,b,c) :- j(c)
h(c,b,d) :- j(d)
h(d,c,a) :- j(a)
h(d,c,b) :- j(b)
h(d,d,c) :- j(c)
```

Loading Metagaming Code

```
<script src='http://epilog.stanford.edu/javascript/
epilog.js'></script>

<script src='http://gamemaster.stanford.edu/javascript/
localstorage.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/
simplifier.js'></script>

<script src='http://gamemaster.stanford.edu/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
general.js'></script>
```

Usage

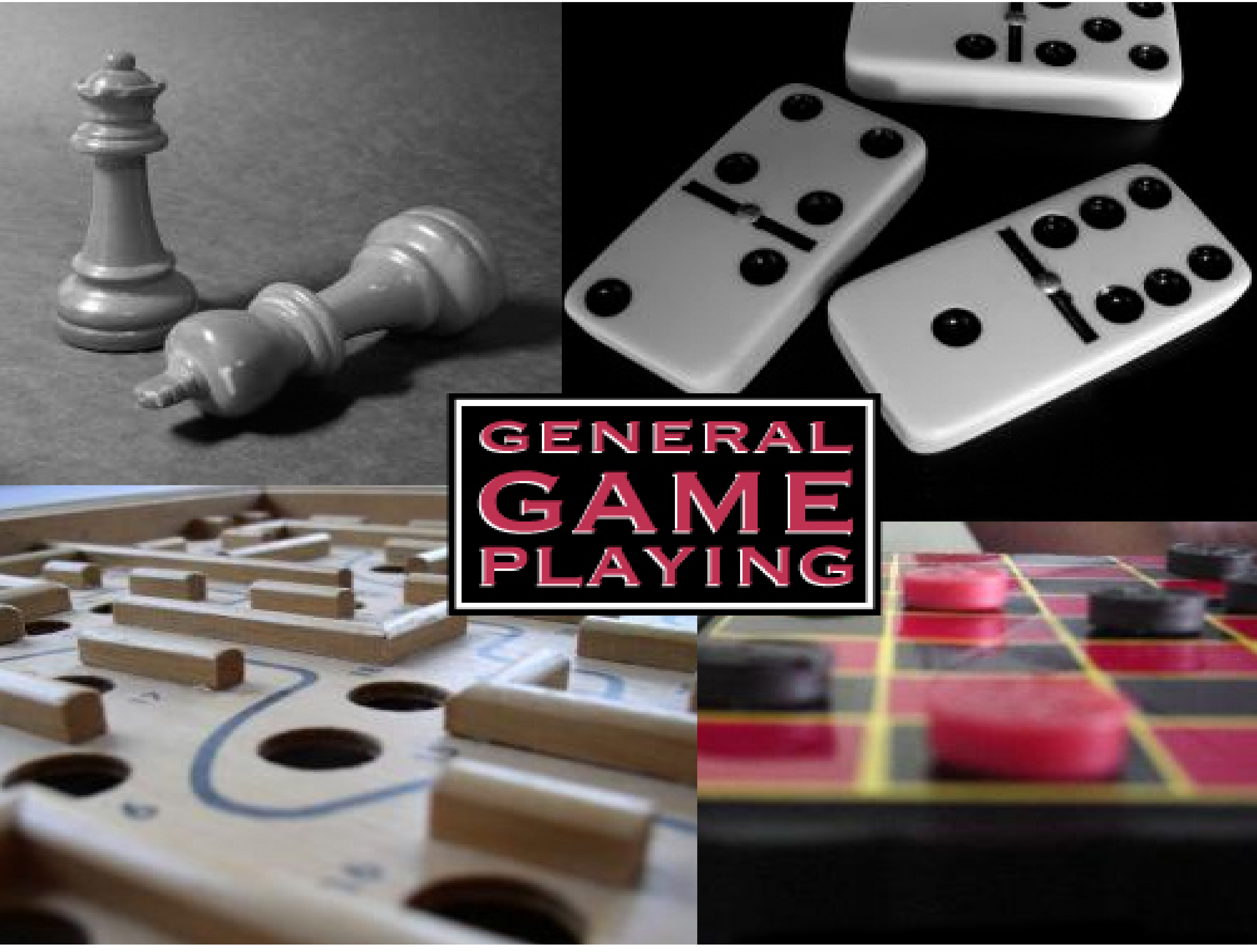
```
function start (r,rs,sc,pc)
 {role = r;
  rules = rs.slice(1);
  startclock = numberize(sc);
  playclock = numberize(pc);

  rules = simplifyrules(rules);

  library = definemorerules([],rs);
  roles = findroles(library);
  state = findinits(library);
  var active = findcontrol(state,library);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,active,reward,false);
  return 'ready'}
```


General Game Playing
HELLESPONT ALHAMBRA


General Game Playing
HELLESPONT ALHAMBRA
Mike


GENERAL
GAME
PLAYING