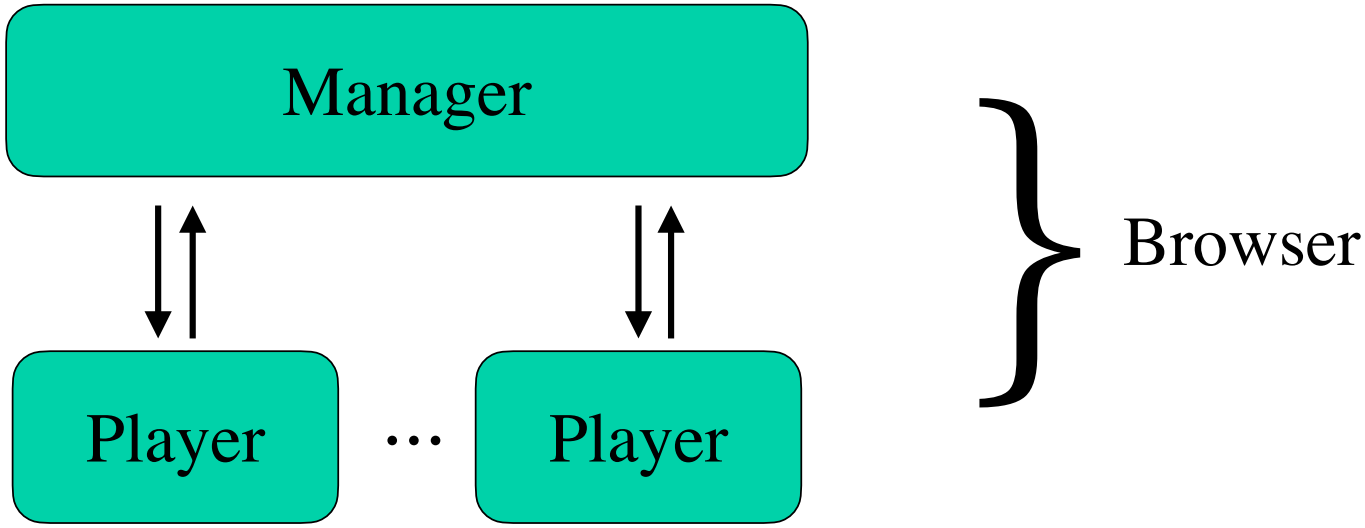
General Game Playing
Game Playing

Michael Genesereth
Computer Science Department
Stanford University

Automatic Game Playing



Communication by message passing between manager and players.

Messages

Manager requests:

`ping()`

`start(`*role*, `ruleset(`$r_1, \ldots, r_k$`),` *startclock*, *playclock*`)`

`play(`*action*`)`

`stop(`*action*`)`

`abort()`

Player responses:

value

Player Architecture

Each player is implemented as an active web page.

Listen Loop

- Receives messages from Game Manager
- Calls appropriate event handler
- Sends result back to Game Manager

Event handlers

- each handles a different type of message
- responds within time bounds

***We** provide listen loop and basic subroutines;*
***you** provide event handlers using those subroutines.*

Event Handlers

`function ping()` - returns ready or busy
{...code that calls predefined subroutines...}

`function start(role,rules,start,play)` - ready
{...code that calls predefined subroutines...}

`function play(move)` - returns action or false
{...code that calls predefined subroutines...}

`function stop(move)` - optional value or false
{...code that calls predefined subroutines...}

`function abort()` - optional value or false
{...code that calls predefined subroutines...}

Predefined Subroutines

```
findroles(rules) - array of roles
findbases(rules) - array of factoids
findactions(rules) - array of actions
findinits(rules) - array of factoids (state)

findcontrol(state, rules) - role
findlegalp(action, state, rules) - boolean
findlegalx(state, rules) - action
findlegals(state, rules) - array of actions
findreward(role, state, rules) - number
findterminalp(state, rules) - boolean

simulate(action, state, rules) - array of factoids
definerules([], rules) - indexed array of rules
```

Legal and Random

Legal Player

Simple behavior

- Maintains current state of the game.
- On each step, selects first legal action it finds.

NB: Selects same action every time in same state.

Implementation (almost)

```
var role, roles, state, library, startclock, playclock;

function ping ()
 {return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  library = rs;
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return findlegalx(state,library)}

function stop (move)
 {return false}

function abort ()
 {return false}
```

Complete Implementation

```
var role, roles, state, library, startclock, playclock;

function ping ()
 {return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  //library = rs;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return findlegalx(state,library)}

function stop (move)
 {return false}

function abort ()
 {return false}
```

Random Player

Simple behavior

- Maintains current state of the game.
- On each step, selects random legal action.

NB: May take different action each time in a state.

Play Handler

Legal Player:

```
function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return findlegalx(state,library)}
```

Random Player:

```
function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  var actions=findlegals(state,library);
  return actions[randomindex(actions.length)]}
```

Implementation

```
var role, roles, state, library, startclock, playclock;

function ping ()
 {return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  var actions=findlegals(state,library);
  return actions[randomindex(actions.length)]}

function stop (move)
 {return false}

function abort ()
 {return false}
```

Remarks on Random Players

Random Players are no "smarter" than legal players.

- Appear more interesting because unpredictable.
- Sometimes avoid traps that befall consistent players like Legal.
- Often used as a comparison to show that a player or method performs statistically better than chance.

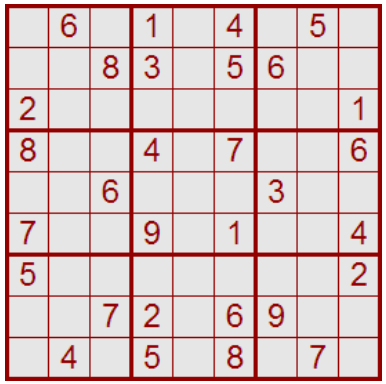
Random players take *slightly* more time than legal

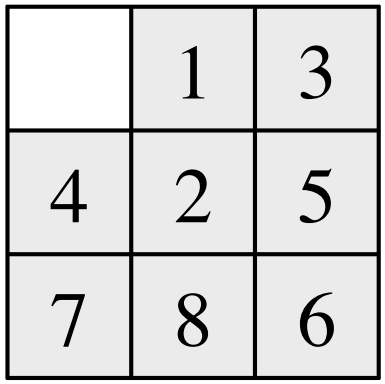
- Compute all legal actions rather than just one
- Noticeable only on games with many legal actions

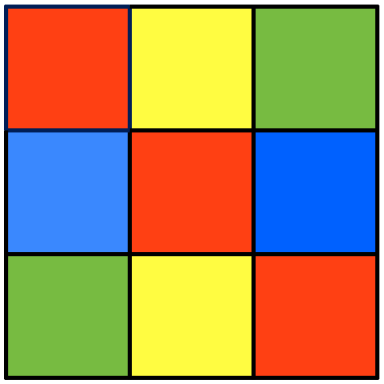
Maximizer

Single Player Games

Example:







Terminology:

- Single Player Games = Puzzles
- Playing Single Player Games = Problem Solving

Easier than multiple player games

- World static (except when single player acts)
- Changes determined entirely by player's actions

Complete Information

Game Description tells player:

- Initial state
- Legal actions in every state
- Results of performing every action in every state
- Reward for every state
- Whether or not a state is terminal

Small Games

Resources

Sufficient time and space
to search the entire game tree

Results

Players can find optimal actions on each time step
NB: *Sometimes* possible without searching entire tree

This rules out puzzles like Rubik's Cube, which requires more complicated techniques, some of which are discussed in subsequent lessons.

Implementation

```
var role, roles, state, library, startclock, playclock;

function ping ()
 {return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  library = rs;   // definerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return bestmove(state)}

function stop (move)
 {return false}

function abort ()
 {return false}
```

Play Handler

Legal Player:

```
function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return findlegalx(state,library)}
```

Maximizer:

```
function play (id,move)
 {if (move!='nil') {state=simulate(move,state,ruleset)};
  if (findcontrol(state,library)!==role) {return false};
  return bestmove(state)}
```

bestmove

```
function bestmove (state)
 {var actions = findlegals(state,library);
  var action = actions[0];
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var result = maxscore(newstate);
       if (result==100) {return actions[i]};
       if (result>score)
          {score = result; action = actions[i]}};
  return action}
```

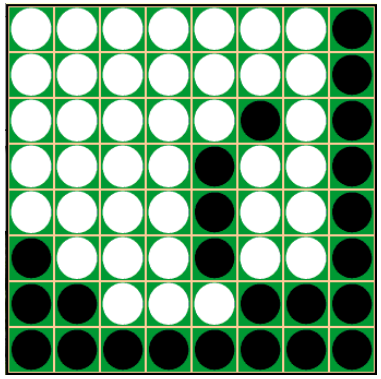
maxscore

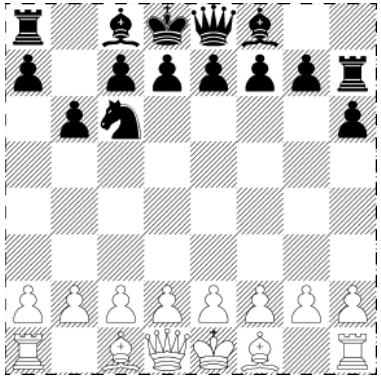
```
function maxscore (state)
 {if (findterminalp(state,ruleset))
     {return findreward(role,state,ruleset)};
  var actions = findlegals(role,state,ruleset);
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,rules);
       var result = maxscore(newstate);
       if (result==100) {return 100};
       if (result>score) {score = result}};
  return score}
```

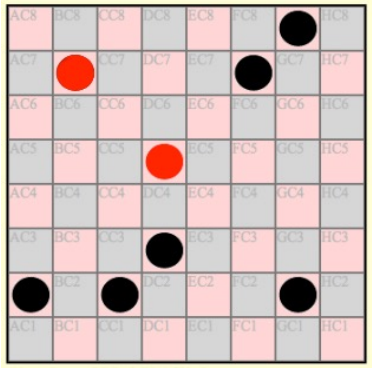
Minimax

Multiple Player Games

Example:







More complicated than single-player games

Changes depend on actions of others

and those actions cannot be controlled

So player must *consider* all possible actions of others

Fixed Sum and Variable Sum Games

Fixed Sum Games

Total reward in all states is the same

For one player to get more, the others must get less

Also called zero sum in cases where sum is 0

Note that, in GGP, all rewards are non-negative.

However, can be scaled - to 50, for example.

Many common games are zero sum

e.g. Chess - winner and loser

Variable Sum Games

Possible for one player to get more without other players getting less

Some games are even cooperative

Small Games

Resources

Sufficient time and space
to search the entire game tree

Results

Players can find optimal *strategy*
Not necessarily *sequential*, as with single player games
Plans may be *conditional* on the actions others
NB: *Sometimes* possible without searching entire tree

Small size rules out games like Chess and Othello, which require more complicated techniques, some of which are discussed in subsequent lessons.

Opponent

Opponent Modeling

May assume opponent will play rationally

but

Assumption may be wrong

Also, players do not know identities of other players

Common Alternative - pessimistic / conservative

Assume the other player will do the *worst* thing for you

*Not maximizing **its** score, minimizing **your** score*

Sometimes called Maximin

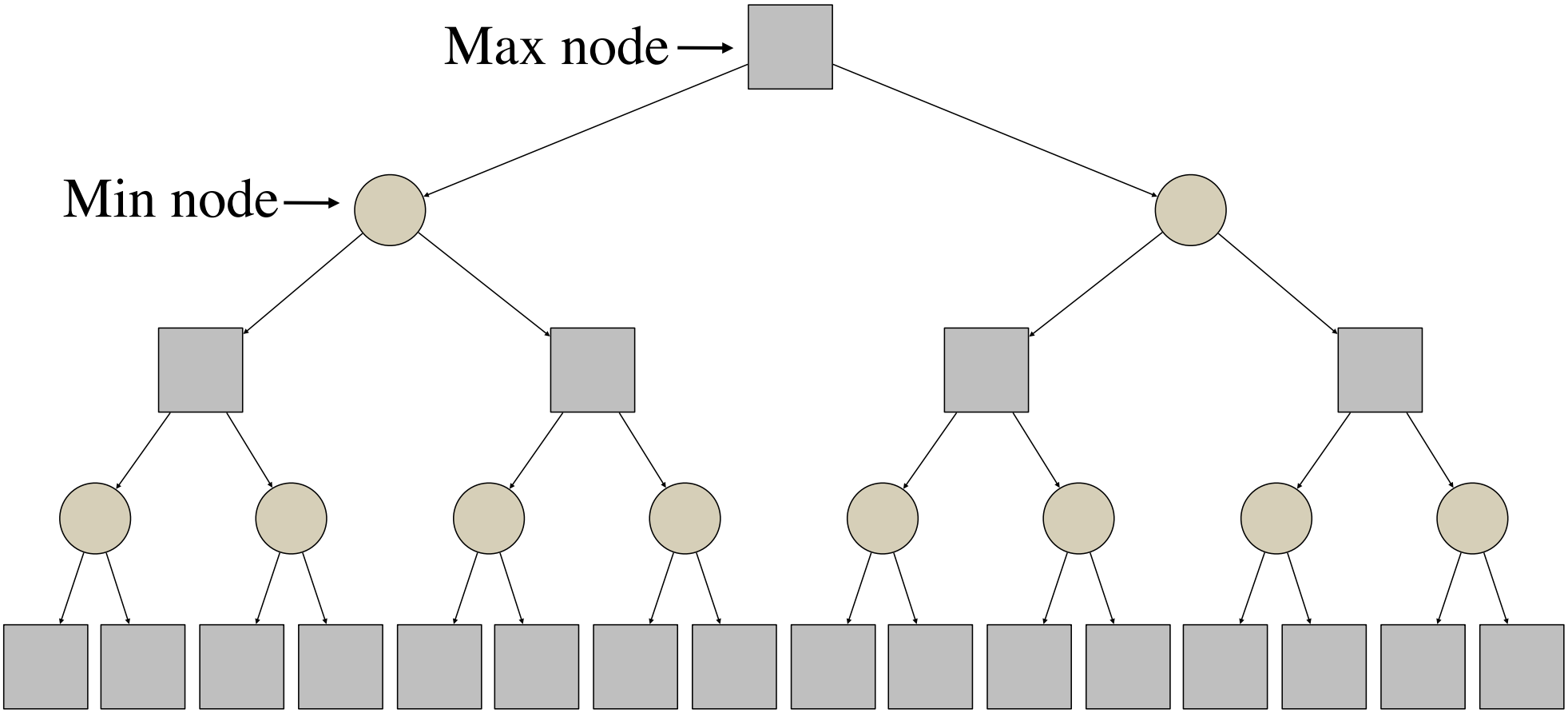
Basic Idea

Intuition - Select a move that is guaranteed to produce the highest possible return no matter what the opponents do.

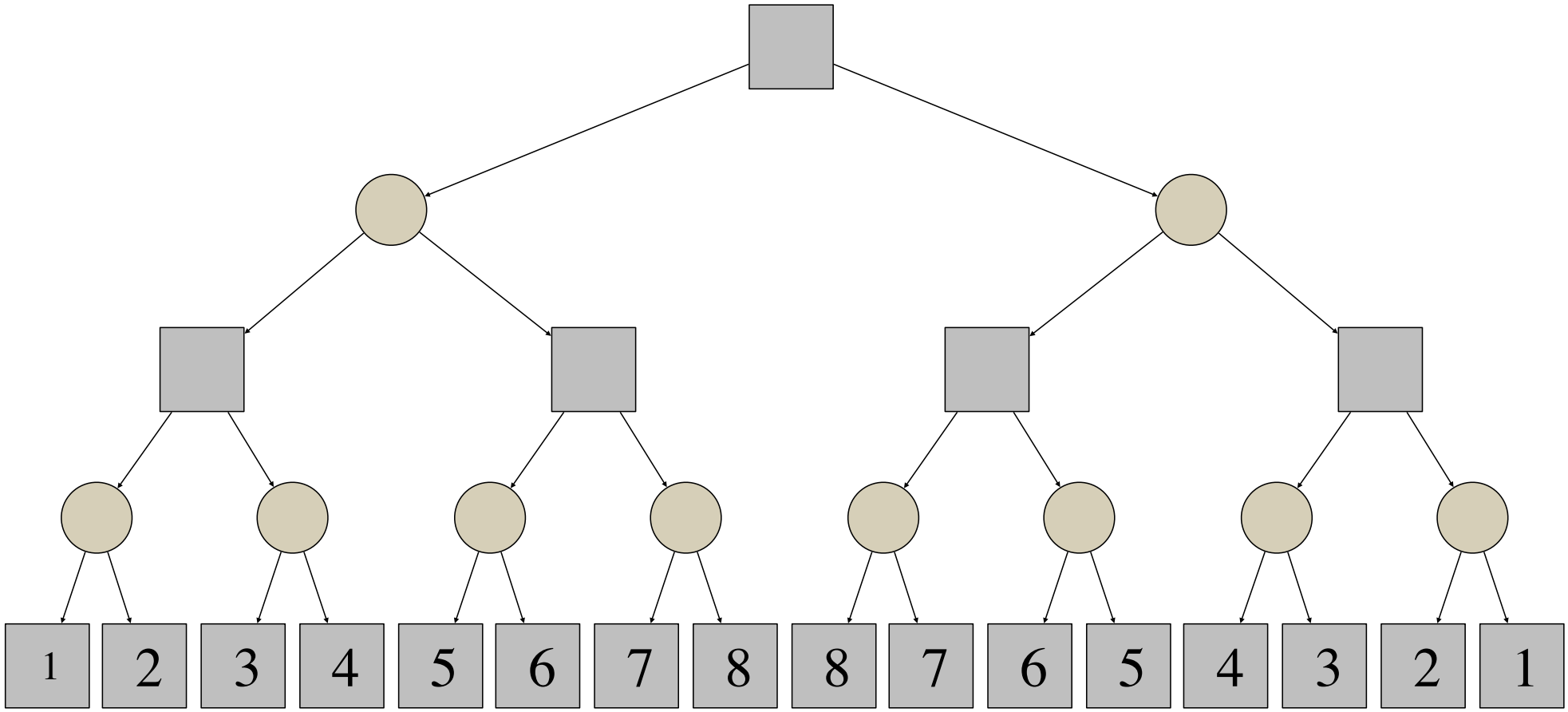
In the case of a one move game, a player should choose an action such that the value of the resulting state for any opponent action is greater than or equal to the value of the resulting state for any other action and opponent action.

In the case of a multi-move game, minimax goes to the end of the game and “backs up” values.

Bipartite Game Tree



Bipartite Game Tree



State Value

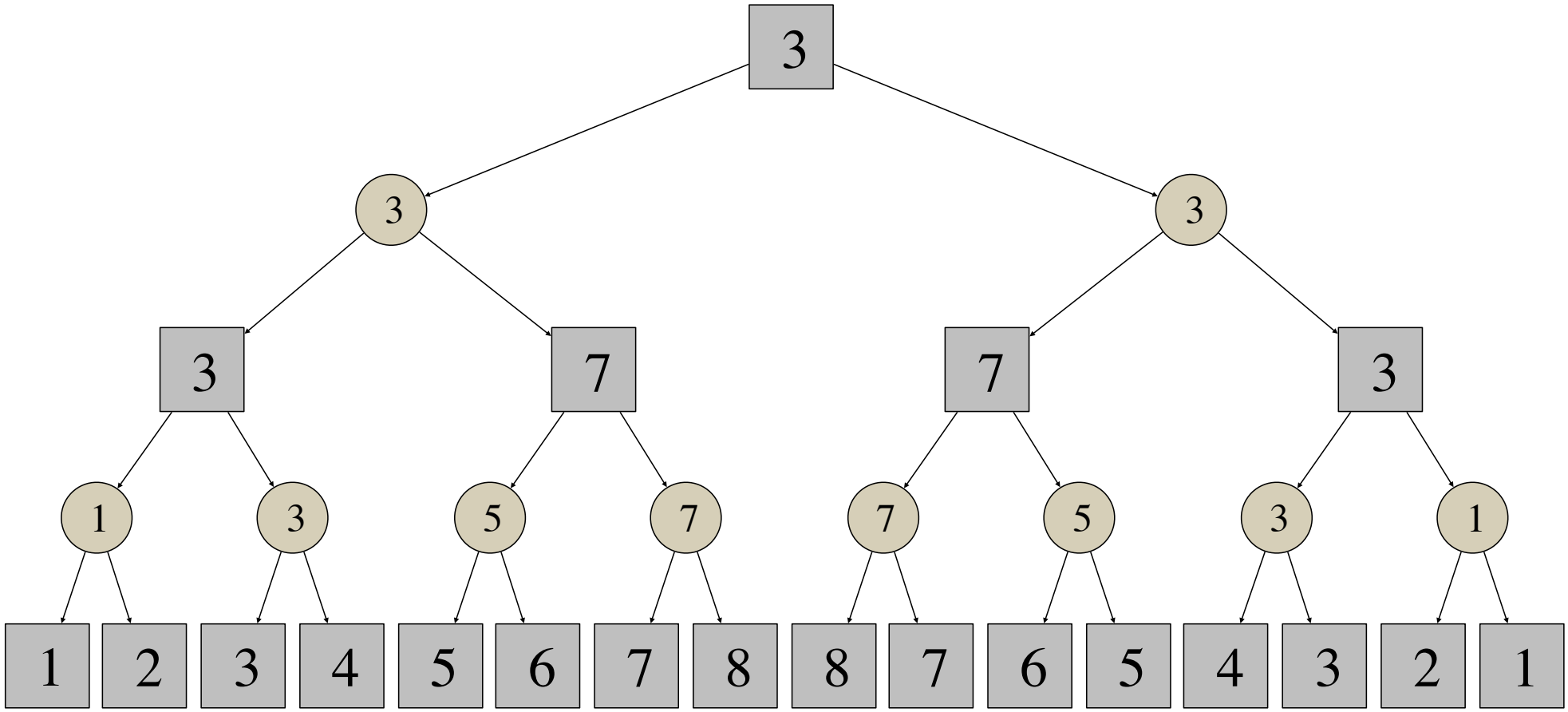
The *value* of a *max* node for player *p* is either the utility of that state if it is terminal or the maximum of all *values* for the *min* nodes that result from its legal actions.

$$value(p,s) =$$
$$goal(p,s) \text{ if } terminal(s)$$
$$max(\{value(p,simulate(a,s)) \mid legal(a,s)\})$$

The *value* of a *min* node is the minimum value that results from any legal opponent action.

$$value(p,s) =$$
$$min(\{value(p,simulate(b,s)) \mid legal(b,s)\})$$

Bipartite Game Tree



Implementation

```
var role, roles, state, library, startclock, playclock;

function ping ()
 {return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  library = rs;   // definerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function play (move)
 {if (move!==nil) {state = simulate(move,state,library)};
  if (findcontrol(state,library)!==role) {return false};
  return bestmove(state)}

function stop (move)
 {return false}

function abort ()
 {return false}
```

bestmove

```
function bestmove (state)
 {var actions = findlegals(role,state,ruleset);
  var action = actions[0];
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
       if (result>score) {score = result; action = actions[i]}};
  return action}
```

minimax

```
function minimax (state)
 {if (findterminalp(state,library))
     {return findreward(role,state,library)*1};
  var active = findcontrol(state,library);
  if (active===role) {return maximize(state)};
  return minimize(state)}
```

maximize and minimize

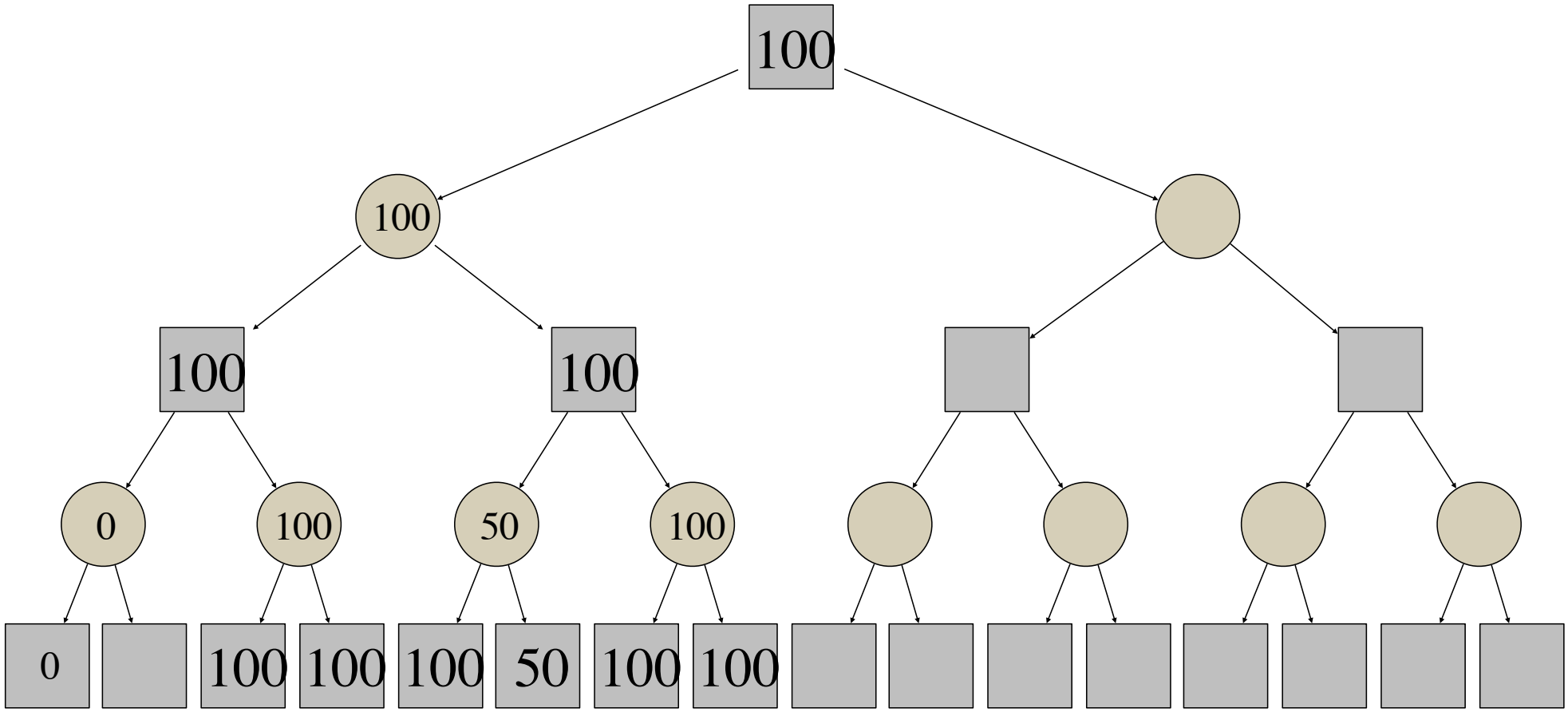
```
function maximize (state)
 {var actions = findlegals(state,library);
  if (actions.length===0) {return 0};
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
      if (newscore>score) {score = newscore}};
  return score}

function minimize (state)
 {var actions = findlegals(state,library);
  if (actions.length===0) {return 0};
  var score = 100;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
      if (newscore<score) {score = newscore}};
  return score}
```

Bounded Minimax

If the minvalue for an action is determined to be 100, then there is no need to consider other actions.

In computing the minvalue for a state if the value to the first player of an opponent's action is 0, then there is no need to consider other possibilities.

Bounded Minimax Example



Generalization

As stated

100 is the limiting case for maxscore

0 is the limiting case for minscore

Other possibilities

Satisficing - fixed minimal score all that is needed

Winner takes all - 51 is sufficient

Maximax

Minimax versus Maximax

Minimax

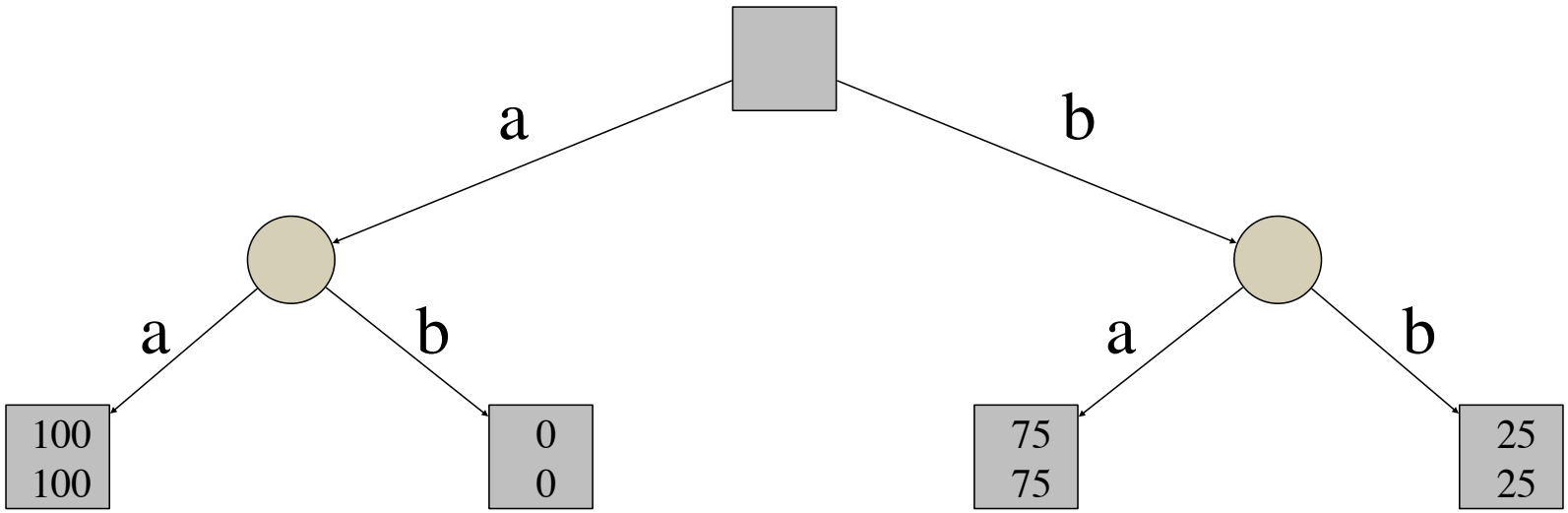
*Not maximizing **its** score, minimizing **your** score*

Maximax

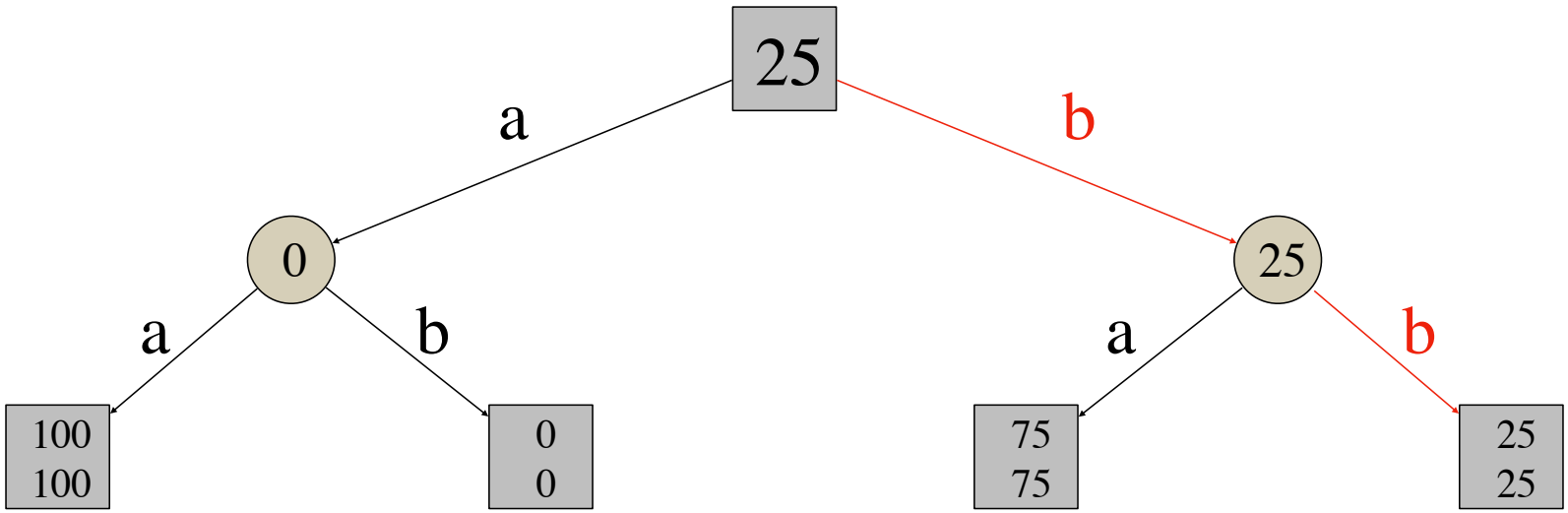
Not minimizing your score, maximizing its own score

For zero-sum games these are equivalent.
Not so for cooperative games.

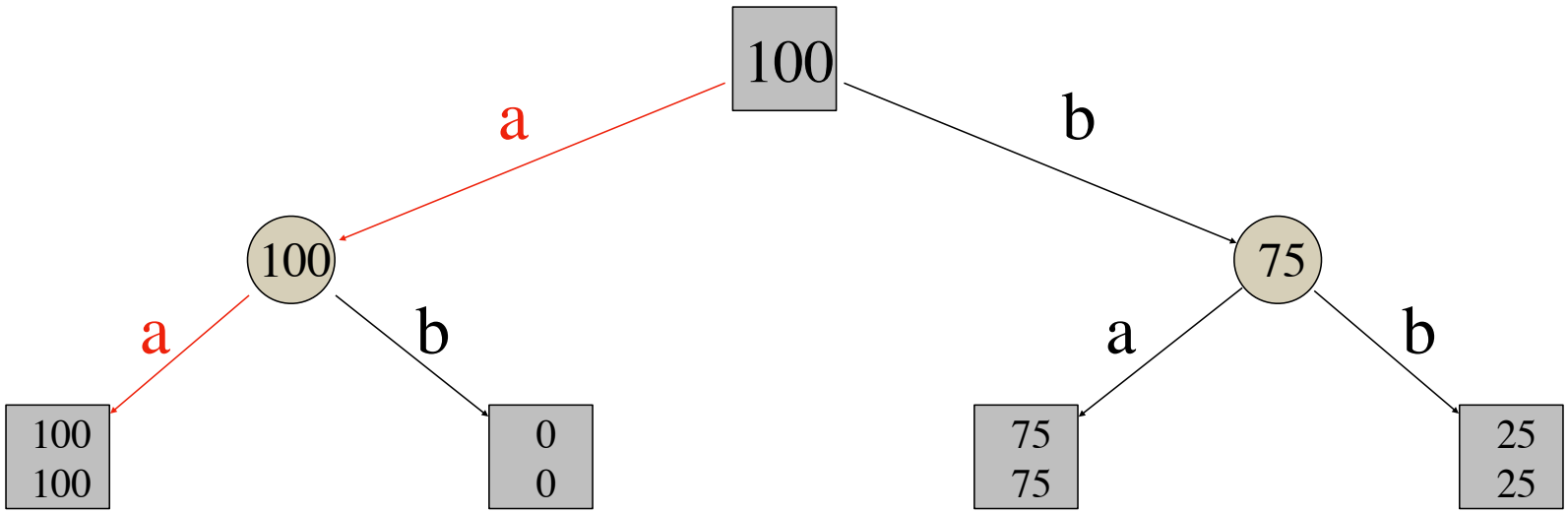
Game Tree for Utopia



Minimax on Utopia



Maximax on Utopia



maximize and minimize

```
function maximize (state)
 {var actions = findlegals(state,library);
  if (actions.length===0) {return 0};
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
      if (newscore>score) {score = newscore}};
  return score}

function minimize (state)
 {var actions = findlegals(state,library);
  if (actions.length===0) {return 0};
  var score = 100;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
      if (newscore<score) {score = newscore}};
  return score}
```

maxvector

```
function maximax (state)
 {if (findterminalp(state,library))
     {var vector = {};
      for (var i=0; i<roles.length; i++)
          {vector[roles[i]]=findreward(roles[i],state,library)*1};
      return vector};
  var active = findcontrol(state,library);
  var actions = findlegals(state,library);
  var vector = {};
  for (var i=0; i<roles.length; i++) {vector[roles[i]]=-1};
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newvector = maximax(newstate);
       if (newvector[active]<vector[active]) {continue};
       if (newvector[active]>vector[active]) {vector = newvector};
       for (var j=0; j<roles.length; j++)
           {if (roles[j]!==active)
               {vector[roles[j]] =
                min(vector[roles[j]],newvector[roles[j]])}}};
  return vector}
```

Alpha-Beta

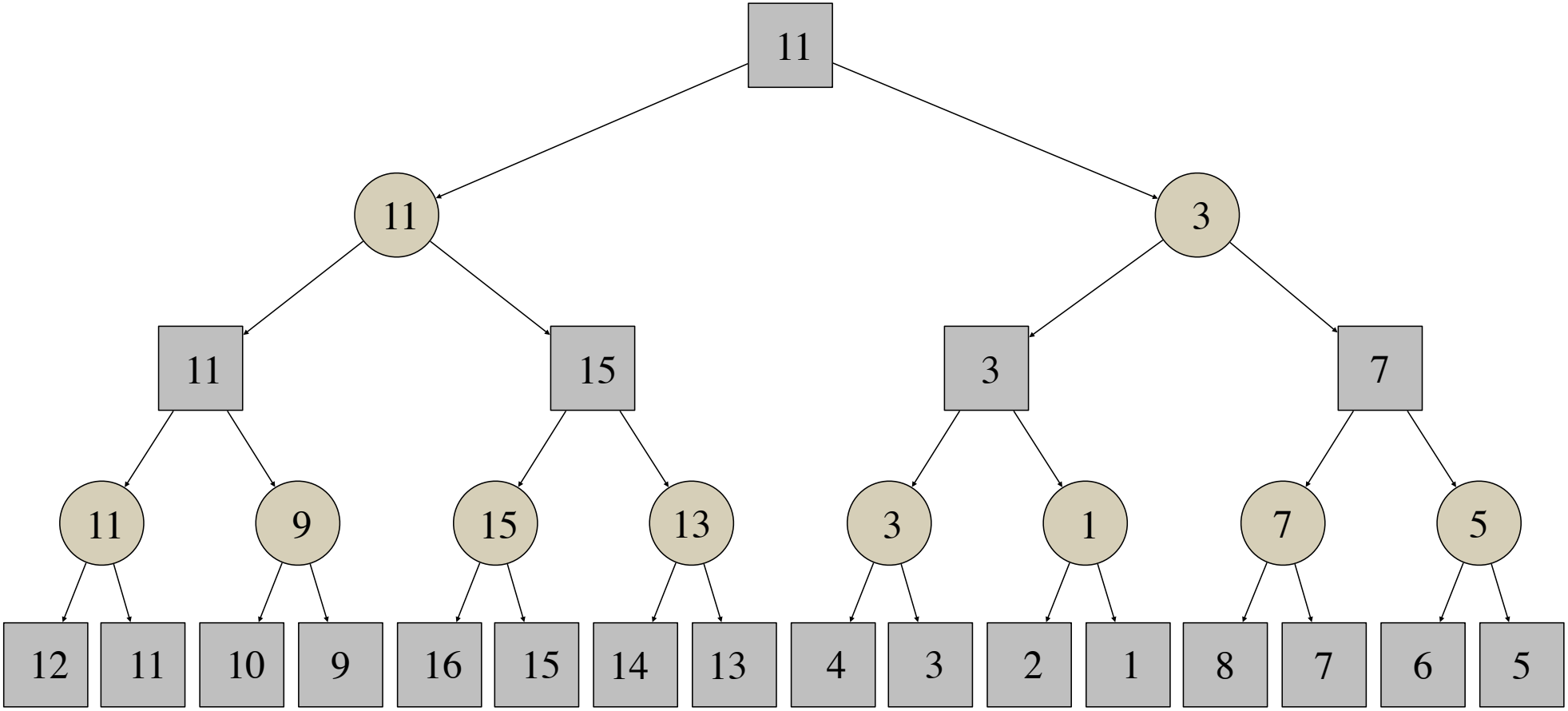
Alpha-Beta Search

Alpha-Beta Search - Same as Bounded Minimax except that bounds are computed dynamically and passed along as parameters.

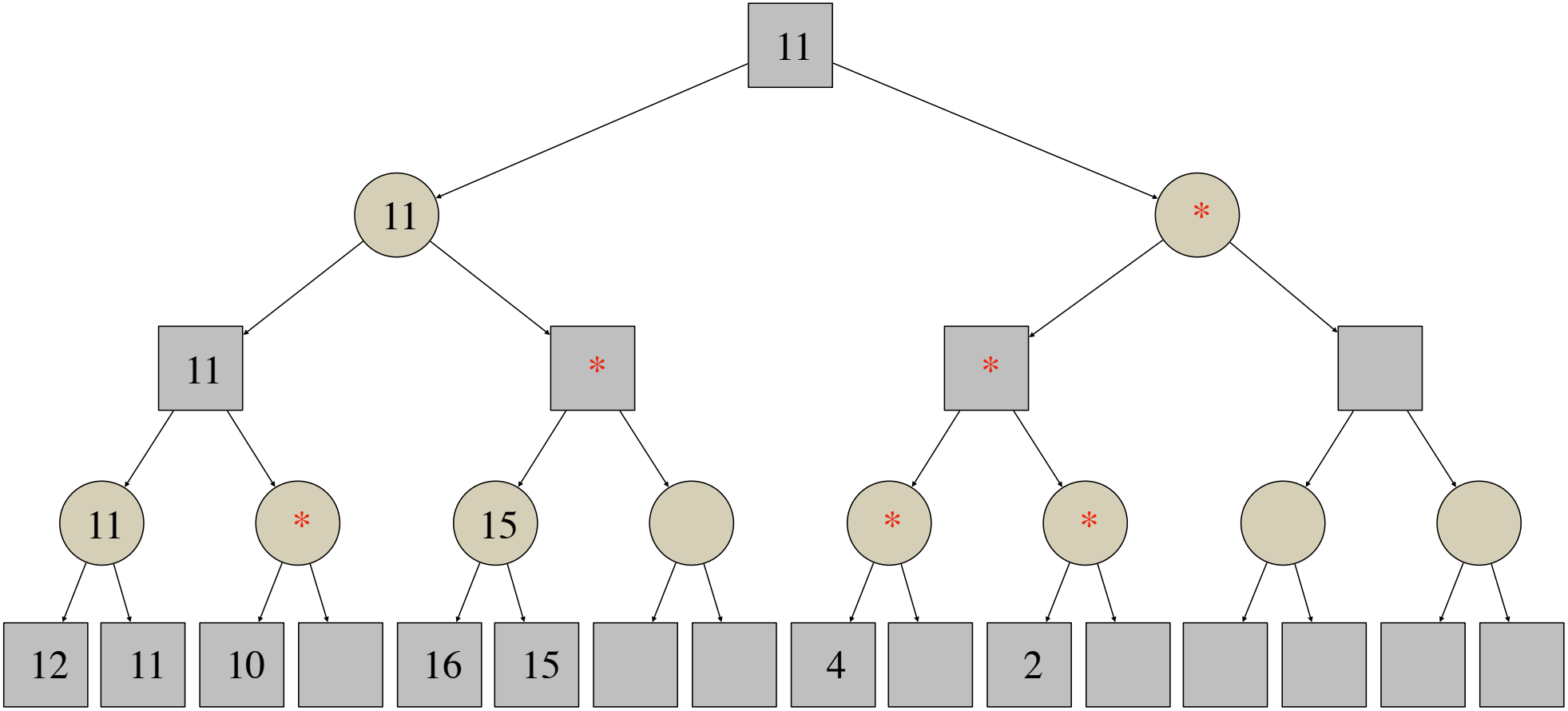
If partial result of min node less than alpha, can only decrease score and player need not consider.

If partial result of max node greater than beta, can only increase score and opponent will not allow.

Minimax Example



Alpha-Beta Example



Benefits

Best case of alpha-beta pruning can reduce search space to *square root* of the unpruned search space, thereby dramatically increasing depth searchable within given time bound.

Benefits

Best case of alpha-beta pruning can reduce search space to *square root* of the unpruned search space, thereby dramatically increasing depth searchable within given time bound.

Gamemaster

Creating Game Players

(1) Create your player code as a javascript file.

(2) Test your player code: Open `playeropen` resource, load player code, use manager to play game.

(3) Download existing player (e.g. Legal) and replace player code with your player code. Then, upload to server and launch from Gamemaster's player page.

Playeropen

Not Secure — gamemaster.stanford.edu

Gamemaster

Sign In

Protocol: localstorage
Strategy: legal
Identifier: anonymous

Playeropen

Not Secure — gamemaster.stanford.edu

Gamemaster

Sign In

Protocol: localstorage
Strategy: custom
Identifier: anonymous

Resources

localhost

Gamemaster

genesereth
Sign Out

Resources

Game Development Utilities

Game Checker - allows users to test game descriptions and stylesheets.

Rule Checker - allows users to test game descriptions (Sierra configuration)

Style Checker - allows users to test Javascript stylesheets on sample states.

Game Management Applications

Standalone - Standalone manager for games on server

Manager - Regular manager for games on server

Stepper - Game stepper for games on server

Standaloneopen - Standalone manager for games described in local files

Manageropen - Regular manager for games described in local files

---

Game Player Applications

Sample - Sample Player to download, customize, and upload.

Human - Human player page for games on server

Humanopen - Human player page for games specified in local files

---

Standard Players

Legal - Legal player

Random - Random player

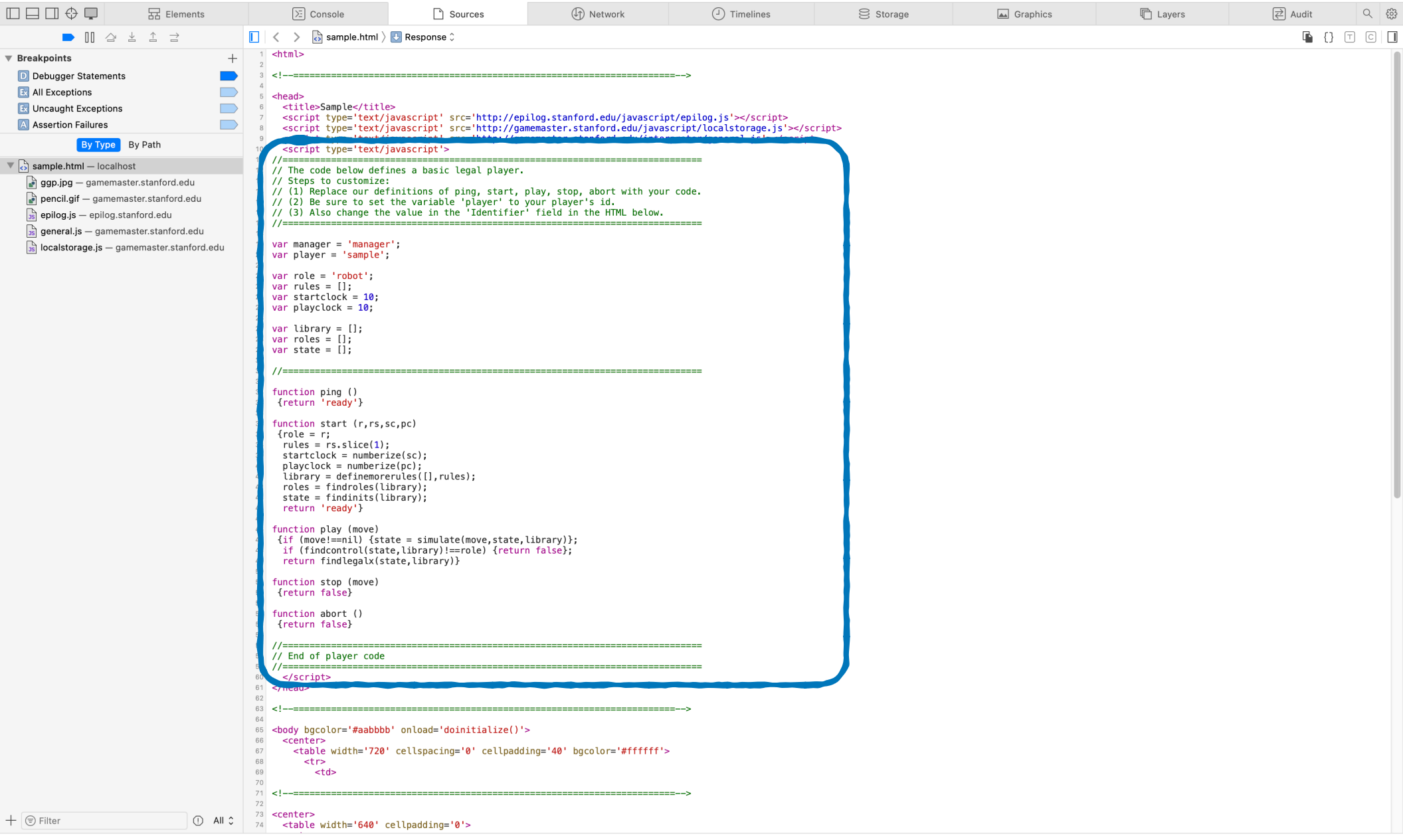
Onestep - One Step player using intermediate values

Twostep - Two Step player using intermediate values

Minimax - Full Minimax player using intermediate values

Minimaxdepth - Minimax player to fixed depth using intermediate values

Page Source


Elements
Console
Sources
Network
Timelines
Storage
Graphics
Layers
Audit
Breakpoints
Debugger Statements
All Exceptions
Uncaught Exceptions
Assertion Failures
By Type
By Path
sample.html — localhost
ggp.jpg — gamemaster.stanford.edu
pencil.gif — gamemaster.stanford.edu
epilog.js — epilog.stanford.edu
general.js — gamemaster.stanford.edu
localstorage.js — gamemaster.stanford.edu
Filter
All
sample.html
Response
<html>
<!--==-->
<head>
<title>Sample</title>
<script type='text/javascript' src='http://epilog.stanford.edu/javascript/epilog.js'></script>
<script type='text/javascript' src='http://gamemaster.stanford.edu/javascript/localstorage.js'></script>
<script type='text/javascript'>
//==
// The code below defines a basic legal player.
// Steps to customize:
// (1) Replace our definitions of ping, start, play, stop, abort with your code.
// (2) Be sure to set the variable 'player' to your player's id.
// (3) Also change the value in the 'Identifier' field in the HTML below.
//==
var manager = 'manager';
var player = 'sample';
var role = 'robot';
var rules = [];
var startclock = 10;
var playclock = 10;
var library = [];
var roles = [];
var state = [];
//==
function ping ()
{return 'ready'}
function start (r,rs,sc,pc)
{role = r;
rules = rs.slice(1);
startclock = numberize(sc);
playclock = numberize(pc);
library = definemorerules([],rules);
roles = findroles(library);
state = findinits(library);
return 'ready'}
function play (move)
{if (move!==nil) {state = simulate(move,state,library)};
if (findcontrol(state,library)!==role) {return false};
return findlegalx(state,library)}
function stop (move)
{return false}
function abort ()
{return false}
//==
// End of player code
//==
</script>
</head>
<!--==-->
<body bgcolor='#aabbbb' onload='doinitialize()'>

<table width='720' cellspacing='0' cellpadding='40' bgcolor='#ffffff'>
<tr>
<td>
<!--==-->

<table width='640' cellpadding='0'>

Profile

Not Secure — gamemaster.stanford.edu

Gamemaster

egghead
Sign Out

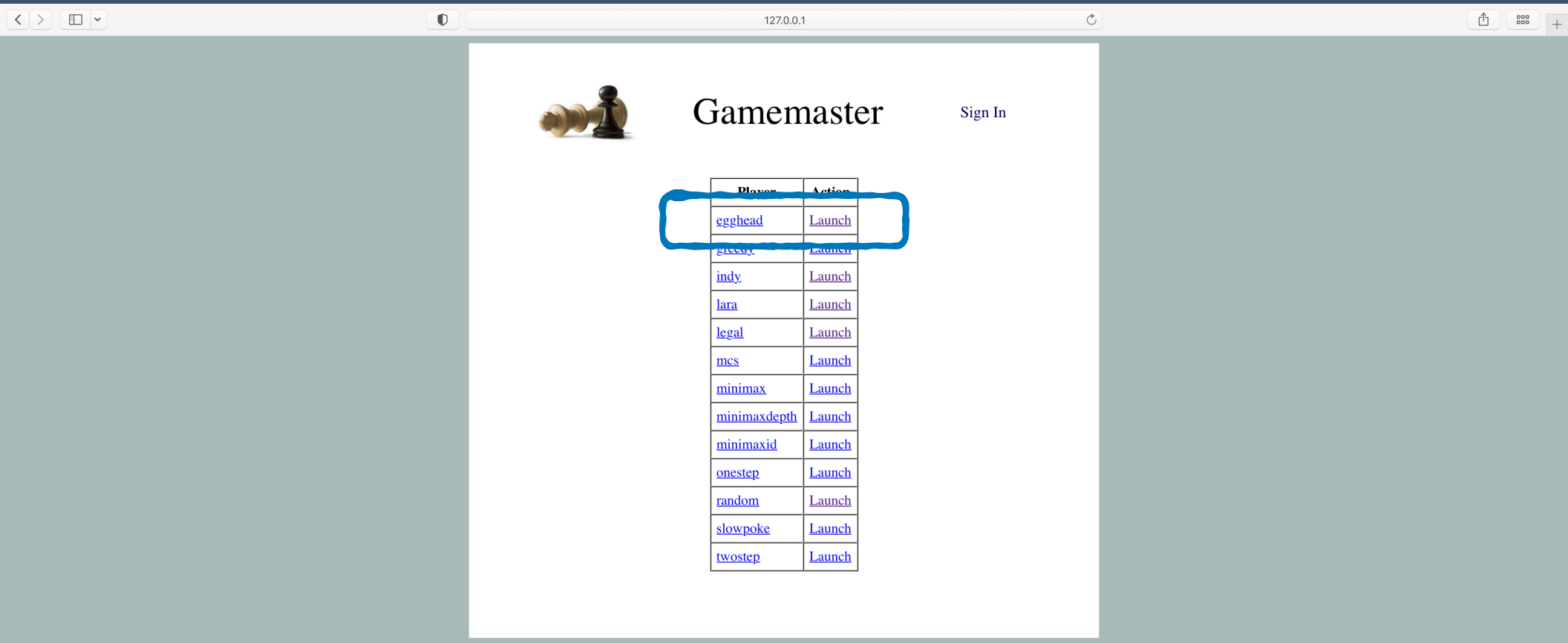
Profile

Id	egghead
Name	Egghead!
Contact	genesereth@stanford.edu
Status	available

Upload Player | Remove Player

Comments and complaints to genesereth@stanford.edu.

Players


127.0.0.1
Gamemaster
Sign In
Player
Action
egghead
Launch
greedy
Launch
indy
Launch
lara
Launch
legal
Launch
mcs
Launch
minimax
Launch
minimaxdepth
Launch
minimaxid
Launch
onestep
Launch
random
Launch
slowpoke
Launch
twostep
Launch

GENERAL
GAME
PLAYING