General Game Playing
Game Playing

Michael Genesereth
Computer Science Department
Stanford University

Gamemaster

About Resources Software
Games Players Metagamers
Matches Run Leaderboard

Gamemaster

Sign In

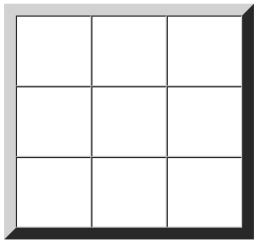
Game	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
alquerque	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badconnectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badhex	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badskirmish	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badtictactoe	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
battleofnumbers	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
bestbuttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
breakthrough	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
buttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
capturetheflag	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
cbnk	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
chinesecheckers3	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
connectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
crusade	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
cryptarithmetic	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
donttouch	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
doubleconnectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hamilton	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hex	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hex7x7	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hunter	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
jointbuttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
knightstour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
knightthrough	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
kono	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
leafymcfallface	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
lightboard	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
majorities	Description	Rulesheet	Stylesheet	Standalone	Human	Manager

Gamemaster

Sign In

Protocol: standalone
Strategy: manager

Game: tictactoe
Startclock: Infinity
Playclock: Infinity



Control: x

Move:

Gamemaster

About Resources Software
Games Players Metagamers
Matches Run Leaderboard

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

Gamemaster

General Game Playing

Protocol: autoplayer
Identifier: legal

Clear Connect Disconnect

file:///Library/WebServer/Documents/ggp/solutions/assignment_09/egghead.html

Gamemaster

General Game Playing

Protocol: autoplayer
Identifier: egghead

Clear Connect Disconnect

Gamemaster

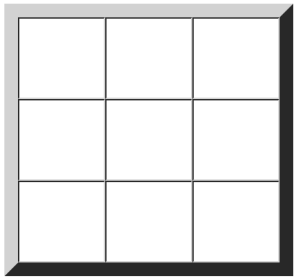
Sign In

Game	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
alquerque	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badconnectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badhex	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badskirmish	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
badtictactoe	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
battleofnumbers	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
bestbuttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
breakthrough	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
buttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
capturetheflag	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
cbnk	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
chinesecheckers3	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
connectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
crusade	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
cryptarithmetic	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
donttouch	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
doubleconnectfour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hamilton	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hex	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hex7x7	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
hunter	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
jointbuttonsandlights	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
knightstour	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
knightthrough	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
kono	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
leafymcfallface	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
lightboard	Description	Rulesheet	Stylesheet	Standalone	Human	Manager
majorities	Description	Rulesheet	Stylesheet	Standalone	Human	Manager

Gamemaster

Protocol: manager
Identifier: manager

Game: tictactoe
Startclock: 10
Playclock: 10



Control: x

Roles	x	o
Players	indy	lara
Score	50	50
Errors	0	0

Clear Begin Pause Resume End

Gamemaster

Protocol: manager
Identifier: manager

Game: tictactoe
Startclock: 10
Playclock: 10

x	x	x
	o	
	o	

Game over

Roles	x	o
Players	indy	lara
Score	100	0
Errors	0	0

Clear Begin Pause Resume End


```
message(1649703514,manager,indy,start(1,x,ruleset(role(x),role(o),base(cell(M,N,x))
:- index(M) & index(N),base(cell(M,N,o)) :- index(M) & index(N),base(cell(M,N,b)) :-
index(M) & index(N),base(control(R)) :- role(R),action(mark(M,N)) :- index(M) &
index(N),index(1),index(2),index(3),init(cell(M,N,b)) :- index(M) &
index(N),init(control(x)),legal(mark(M,N)) :- cell(M,N,b),mark(M,N) :: control(R)
==> cell(M,N,R) & ~cell(M,N,b),mark(M,N) :: control(x) ==> ~control(x) &
control(o),mark(M,N) :: control(o) ==> ~control(o) & control(x),goal(x,100) :-
line(x) & ~line(o),goal(x,50) :- line(x) & line(o),goal(x,50) :- ~line(x) &
~line(o),goal(x,0) :- ~line(x) & line(o),goal(o,100) :- ~line(x) &
line(o),goal(o,50) :- line(x) & line(o),goal(o,50) :- ~line(x) & ~line(o),goal(o,0)
:- line(x) & ~line(o),row(M,X) :- cell(M,1,X) & cell(M,2,X) & cell(M,3,X),col(N,X)
:- cell(1,N,X) & cell(2,N,X) & cell(3,N,X),diag(X) :- cell(1,1,X) & cell(2,2,X) &
cell(3,3,X),diag(X) :- cell(1,3,X) & cell(2,2,X) & cell(3,1,X),line(X) :-
row(M,X),line(X) :- col(N,X),line(X) :- diag(X),terminal :- line(x),terminal :-
line(o),terminal :- ~open,open :- cell(M,N,b)),10,10))
message(1649703514,manager,lara,start(1,o,ruleset(role(x),role(o),base(cell(M,N,x))
:- index(M) & index(N),base(cell(M,N,o)) :- index(M) & index(N),base(cell(M,N,b)) :-
```

Programme

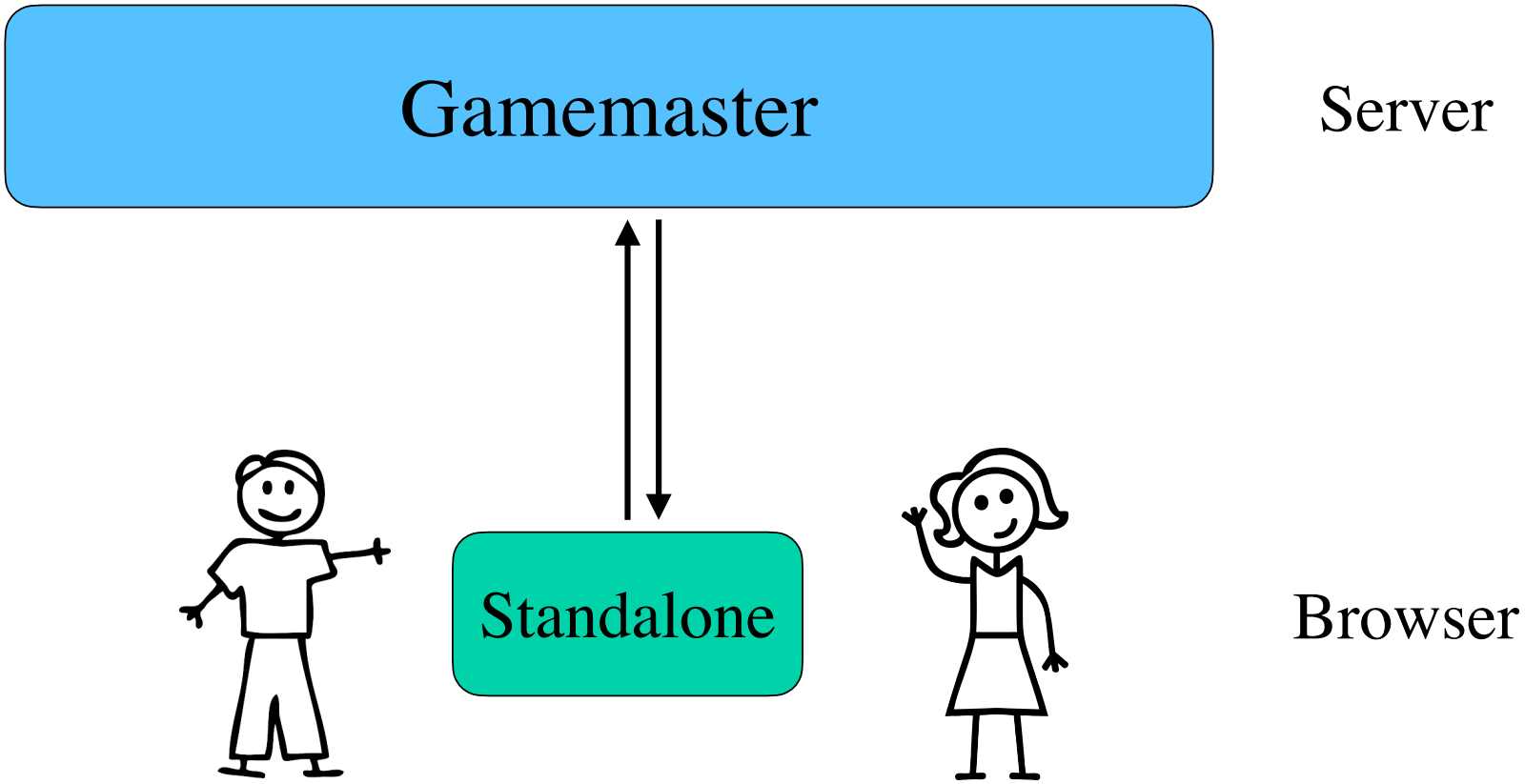
Game Management

Game Players

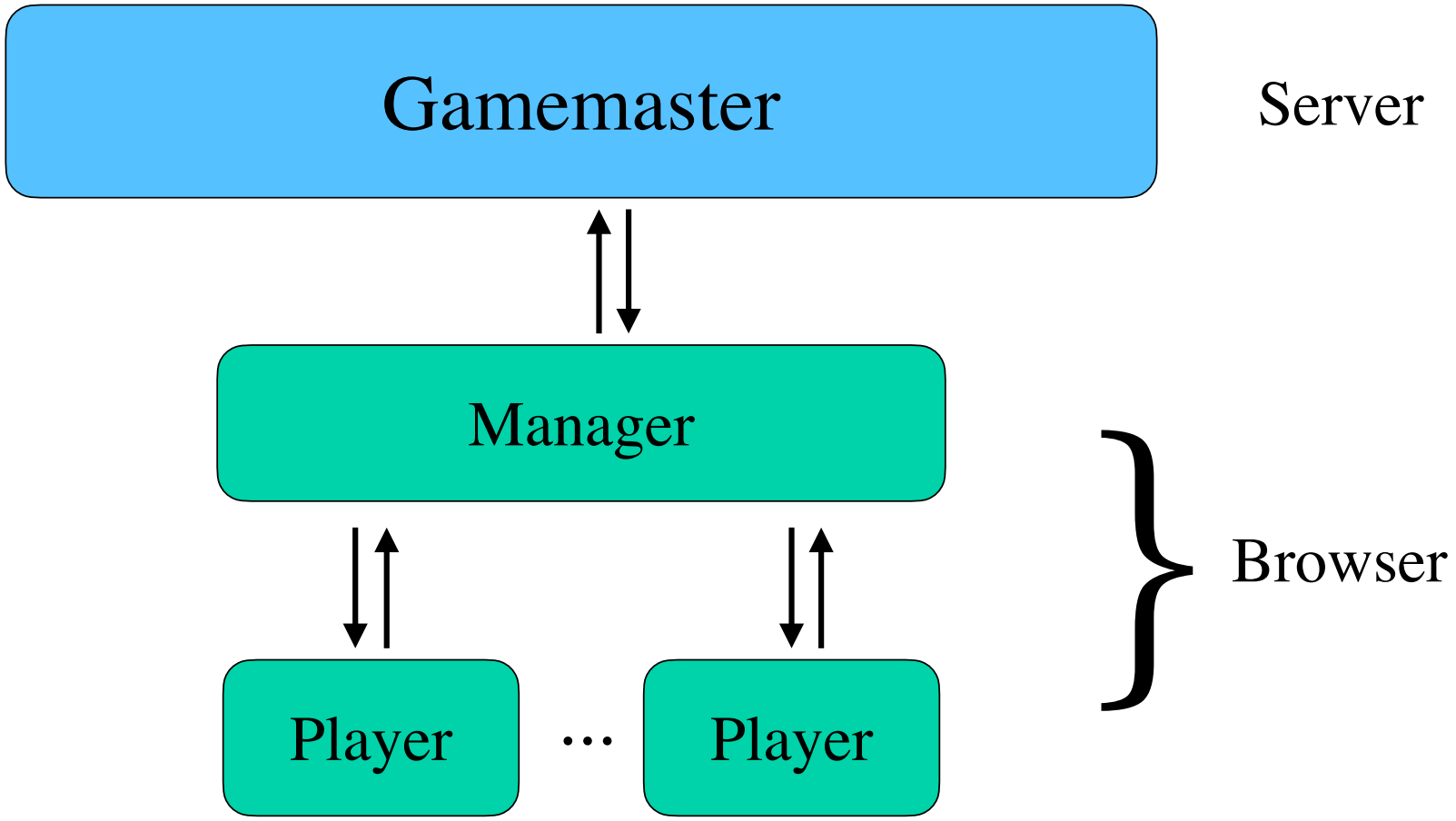
Complete Search Techniques for Single Player Games
Complete Search Techniques for Multiple Player Games

Game Management

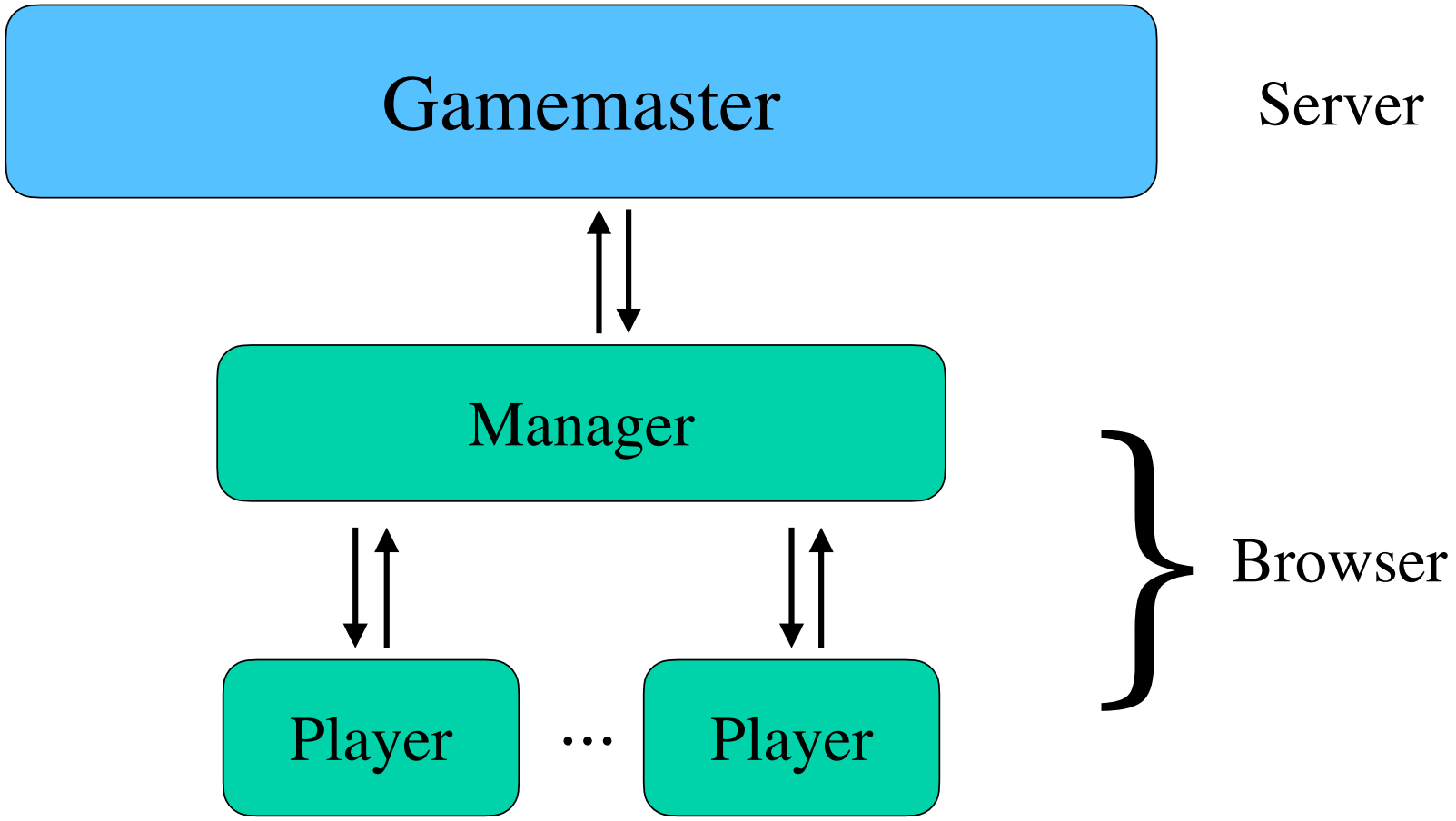
Standalone Game Playing



Automatic Game Playing

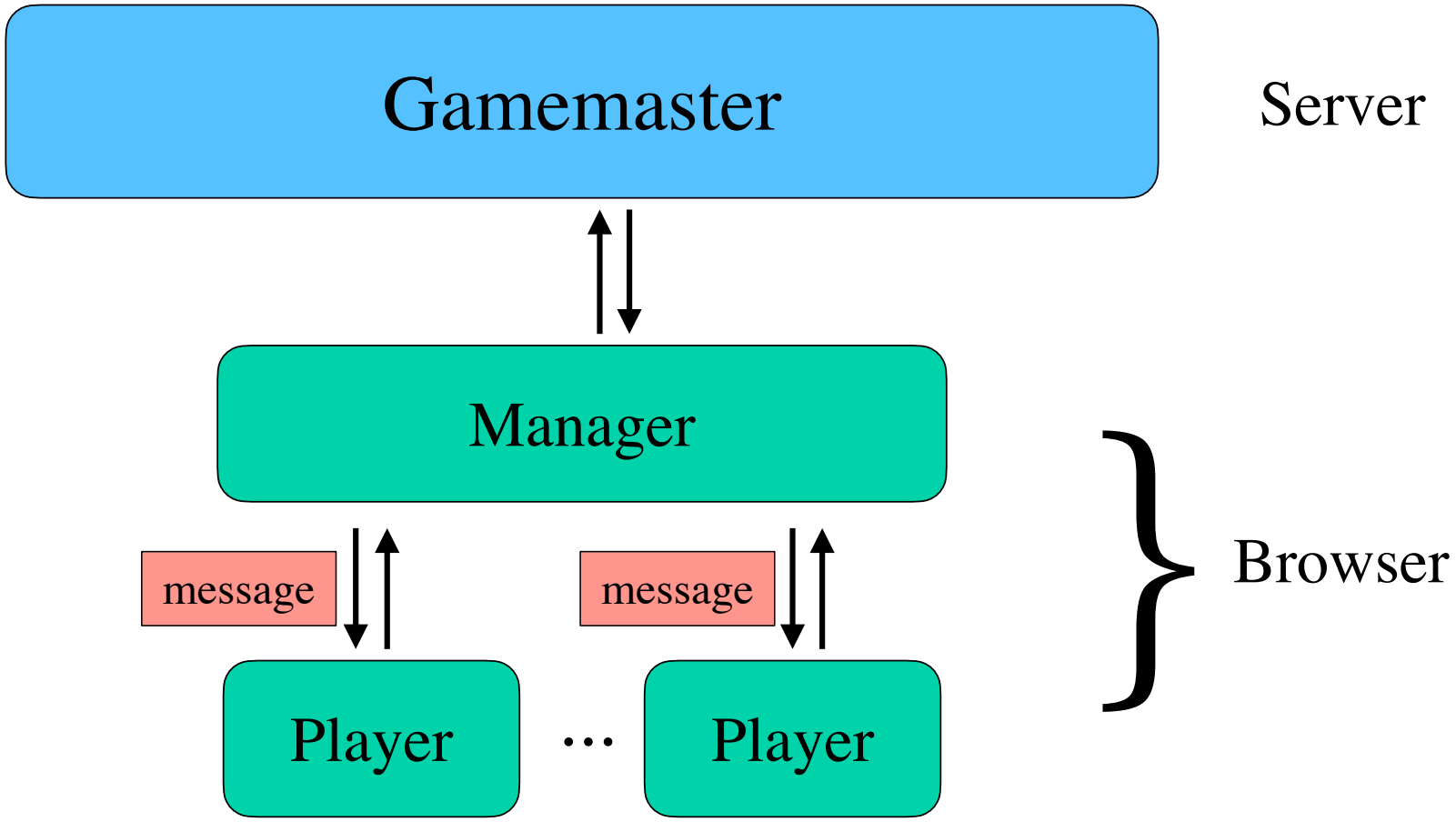


Automatic Game Playing



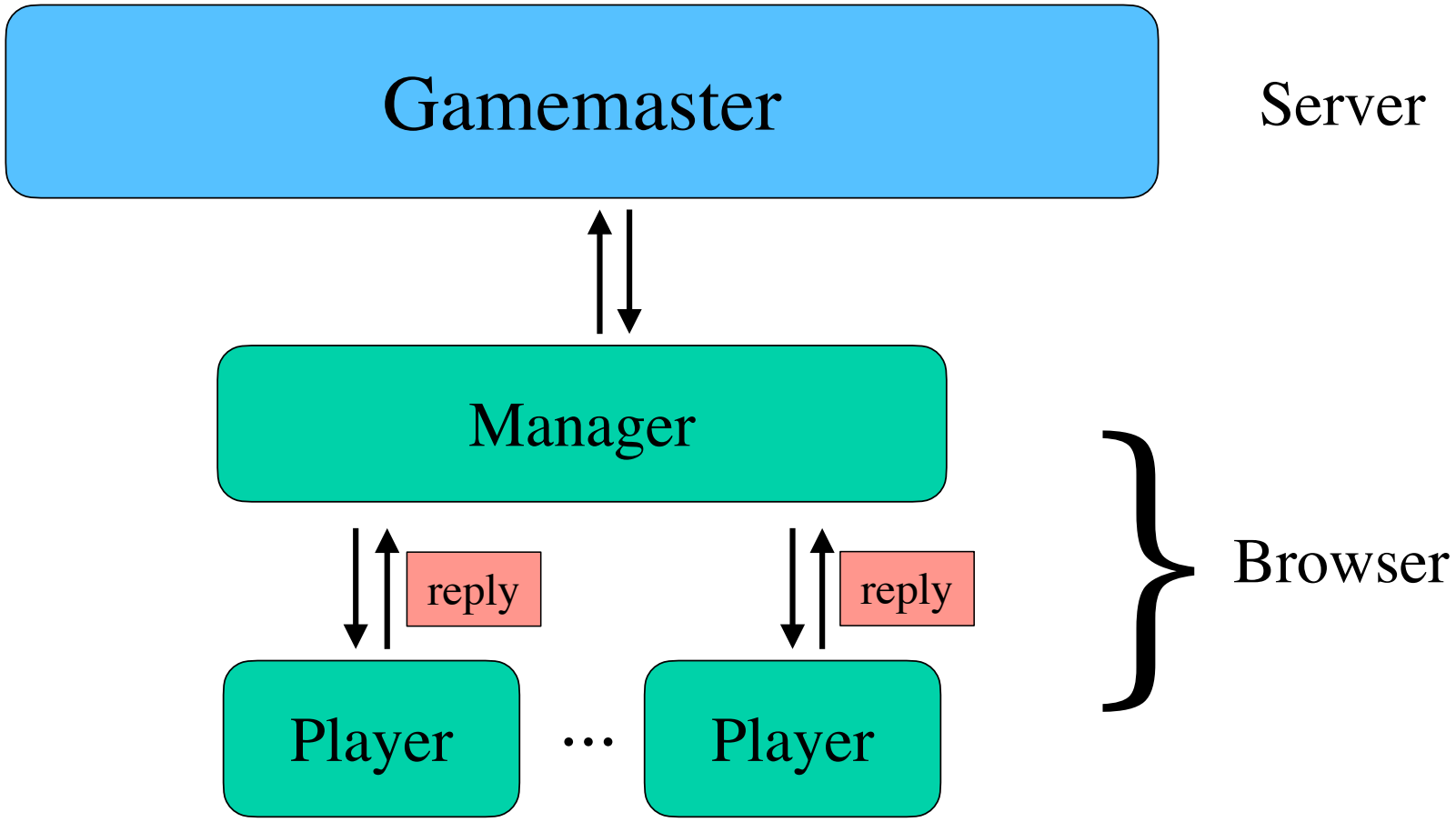
Communication by message passing between manager and players.

Automatic Game Playing



Manager writes messages into browser's local storage.

Automatic Game Playing



Players receive messages, process, write replies into local storage.

Messages

Manager requests:

`ping(`n`)`

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

`play(`n, a`)`

`stop(`n, a`)`

`abort(`n`)`

Player responses:

`reply(`n`,`$value$`)`

The ping Message

A *ping* message is used to determine if a player is running, communicating, and ready to play a match.

General Form:

`ping(`*messageid*`)`

Replies:

`reply(`*messageid*`,ready)` - ready to play

`reply(`*messageid*`,busy)`- not ready

No reply after 5 seconds interpreted as not ready

The start Message

A start message initiates a match.

General Form:

`start(`*messageid*`,`*role*`,ruleset(`r_1`,…,`r_k`),`*start*`,`*play*`)`

Reply:

`reply(`*messageid*`,ready)` - ready to begin play

NB: Match begins as soon as all players have replied *or*
when *startclock* seconds have elapsed whichever comes first.

The play Message

A play message is an update and a request for an action.

General Form:

`play(`*messageid*`, nil)`
`play(`*messageid*`,` *action*`)`

NB: `nil / []` is argument on first step, *action* thereafter, where *action* is the action performed on preceding step.

Reply:

`reply(`*messageid*`,` *action*`)`

NB: No need to reply if not in control.

NB: If player in control returns an illegal action *or* playclock has elapsed, the Manager substitutes a random legal action.

The stop Message

A stop message is used to inform players that the match has terminated successfully.

General Form:

`stop(`*messageid*`,` *action*`)`

Reply:

optional

The abort Message

An abort message is used to tell players that a match has terminated abnormally.

General Form:

`abort(`*messageid*`)`

Reply:

optional

Match Management Procedure

Begins when manager receives a request to run a match of a given game with given players and given start clock and play clock.

(1) optionally sends ping messages to players
(2) sends start message with appropriate parameters
(3) sends play messages to all players
(4) sends stop message when game terminates
(5) sends abort for abnormal termination

Tic-Tac-Toe Example

```
message(1,manager,indy,ping())
message(1,manager,lara,ping())
message(1,indy,manager,ready)
message(1,lara,manager,ready)
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,o,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))

message(4,manager,indy,play(m01,mark(1,1)))
message(4,manager,lara,play(m01,mark(1,1)))
message(4,lara,manager,mark(1,2))
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))

message(4,manager,indy,play(m01,mark(1,1)))
message(4,manager,lara,play(m01,mark(1,1)))
message(4,lara,manager,mark(1,2))

message(5,manager,indy,play(m01,mark(1,2)))
message(5,manager,lara,play(m01,mark(1,2)))
message(5,indy,manager,mark(2,2))
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))

message(4,manager,indy,play(m01,mark(1,1)))
message(4,manager,lara,play(m01,mark(1,1)))
message(4,lara,manager,mark(1,2))

message(5,manager,indy,play(m01,mark(1,2)))
message(5,manager,lara,play(m01,mark(1,2)))
message(5,indy,manager,mark(2,2))

message(6,manager,indy,play(m01,mark(2,2)))
message(6,manager,lara,play(m01,mark(2,2)))
message(6,lara,manager,mark(1,3))
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))

message(4,manager,indy,play(m01,mark(1,1)))
message(4,manager,lara,play(m01,mark(1,1)))
message(4,lara,manager,mark(1,2))

message(5,manager,indy,play(m01,mark(1,2)))
message(5,manager,lara,play(m01,mark(1,2)))
message(5,indy,manager,mark(2,2))

message(6,manager,indy,play(m01,mark(2,2)))
message(6,manager,lara,play(m01,mark(2,2)))
message(6,lara,manager,mark(1,3))

message(7,manager,indy,play(m01,mark(1,3)))
message(7,manager,lara,play(m01,mark(1,3)))
message(7,indy,manager,mark(3,3))
```

Tic-Tac-Toe Example

```
message(2,manager,indy,start(m01,x,ruleset(...),10,10))
message(2,manager,lara,start(m01,x,ruleset(...),10,10))
message(2,indy,manager,ready)
message(2,lara,manager,ready)

message(3,manager,indy,play(m01,[]))
message(3,manager,lara,play(m01,[]))
message(3,indy,manager,mark(1,1))

message(4,manager,indy,play(m01,mark(1,1)))
message(4,manager,lara,play(m01,mark(1,1)))
message(4,lara,manager,mark(1,2))

message(5,manager,indy,play(m01,mark(1,2)))
message(5,manager,lara,play(m01,mark(1,2)))
message(5,indy,manager,mark(2,2))

message(6,manager,indy,play(m01,mark(2,2)))
message(6,manager,lara,play(m01,mark(2,2)))
message(6,lara,manager,mark(1,3))

message(7,manager,indy,play(m01,mark(1,3)))
message(7,manager,lara,play(m01,mark(1,3)))
message(7,indy,manager,mark(3,3))

message(8,manager,indy,stop(m01,mark(1,3)))
message(8,manager,lara,stop(m01,mark(1,3)))
```

Tic-Tac-Toe Example

```
message(9,manager,indy,abort(m01))
message(9,manager,lara,abort(m01))
```

Game Players

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 with time bounds

***We** provide listen loop, **you** provide event handlers.*

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...}

Subroutines defined using epilog.js

```
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
```

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.

Complete 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 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.

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};
  var actions=findlegals(state,library);
  return actions[randomindex(actions.length)]}

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)}
```

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)]}
```

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.

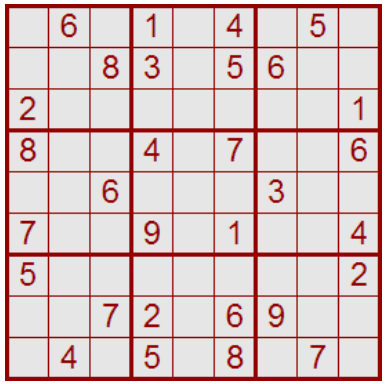
Random players take *slightly* more time than legal

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

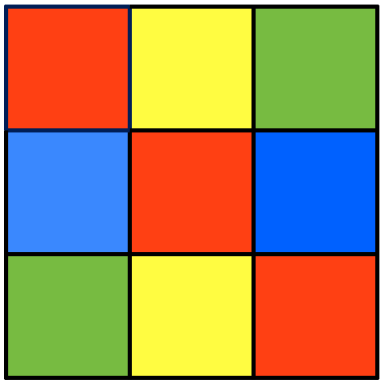
Maximizer

Single Player Games

Example:



	1	3
4	2	5
7	8	6



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}
```

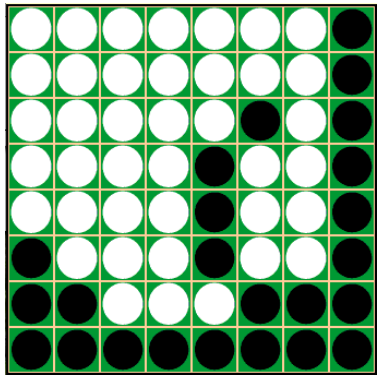
maxscore

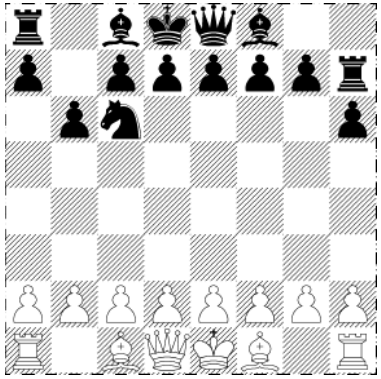
```
function maxscore (state)
 {if (findterminalp(state,ruleset))
     {return findreward(role,state,ruleset)};
  var actions = findlegals(role,state,ruleset);
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,rules);
       var result = maxscore(newstate);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

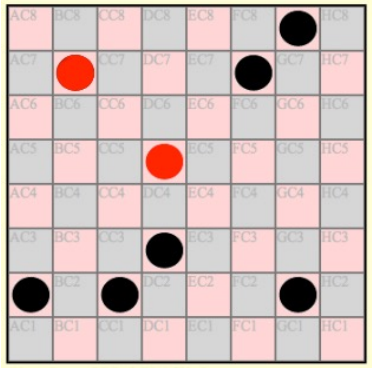
Minimax

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* possible thing

Not maximizing its score, minimizing your score

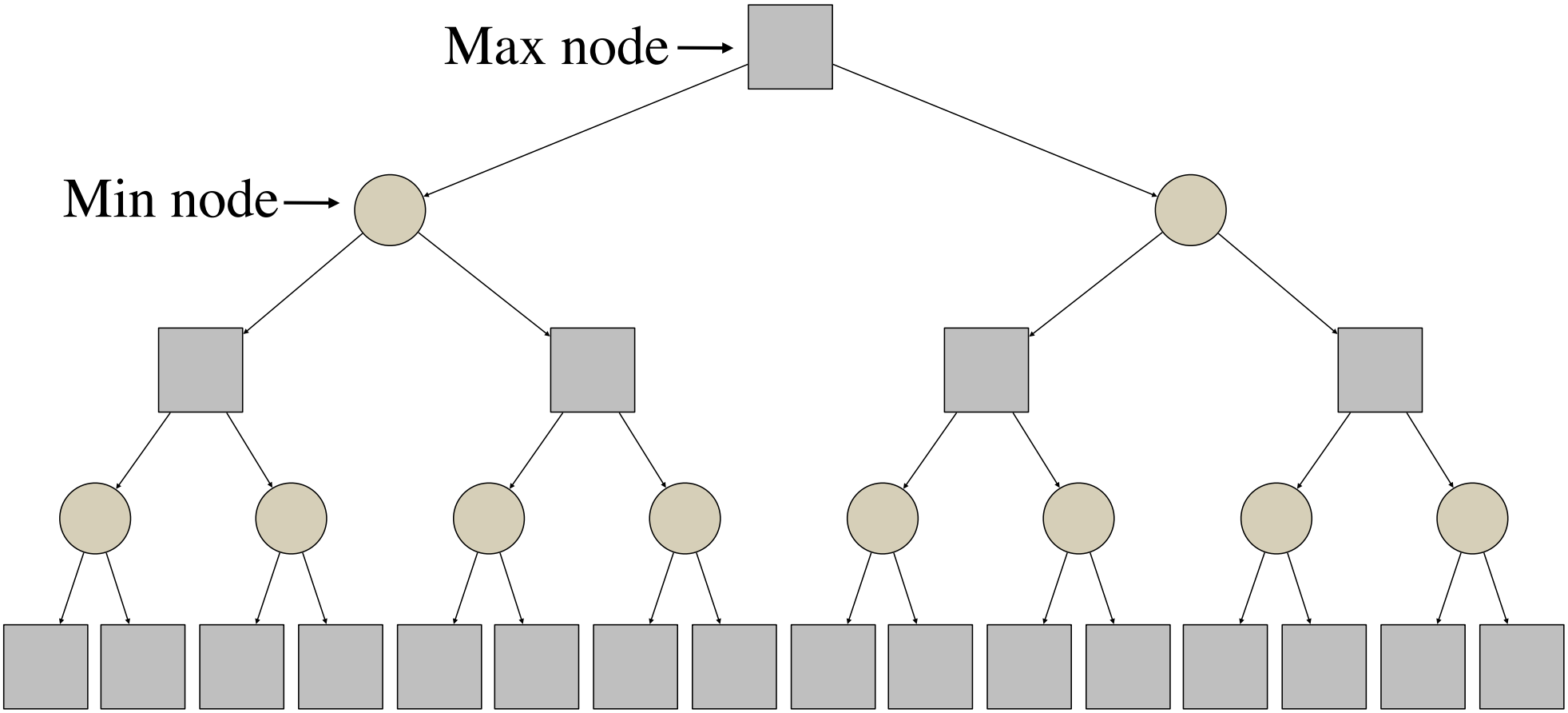
Basic Idea

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

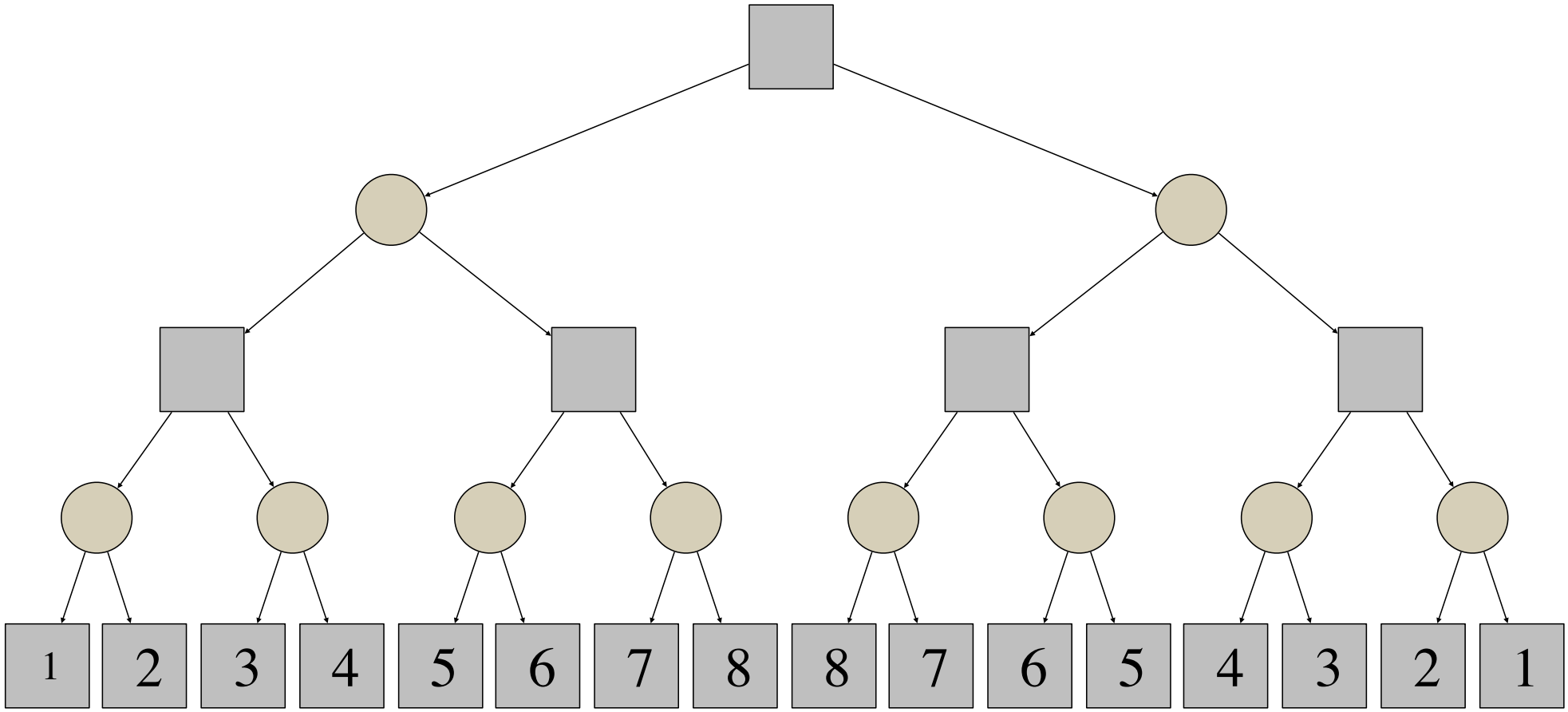
In the case of a one move game, a player should choose an action such that the value of the resulting state for any opponent action is greater than or equal to the value of the resulting state for any other action and opponent action.

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

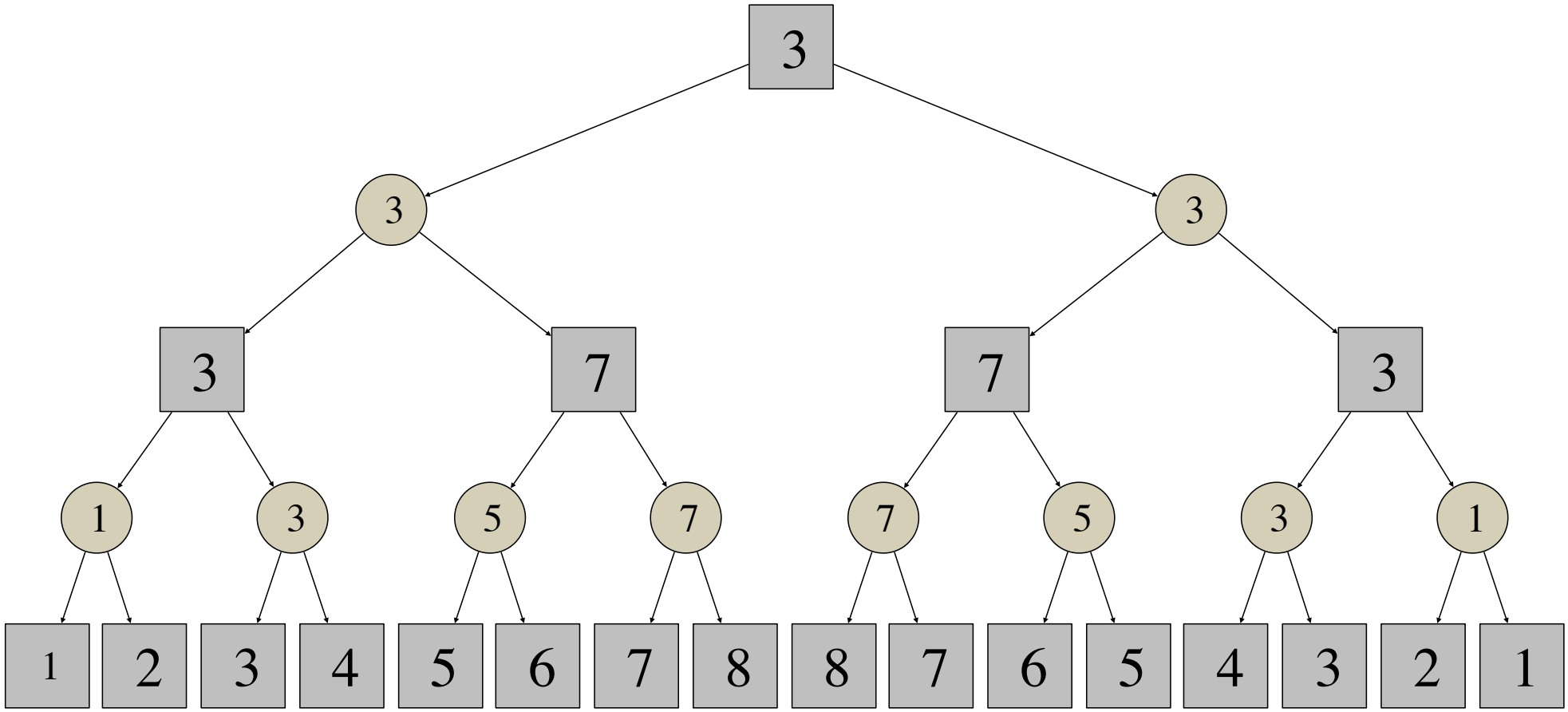
The *value* of a *max* node for player *p* is either the utility of that state if it is terminal or the maximum of all *values* for the *min* nodes that result from its legal actions.

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

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

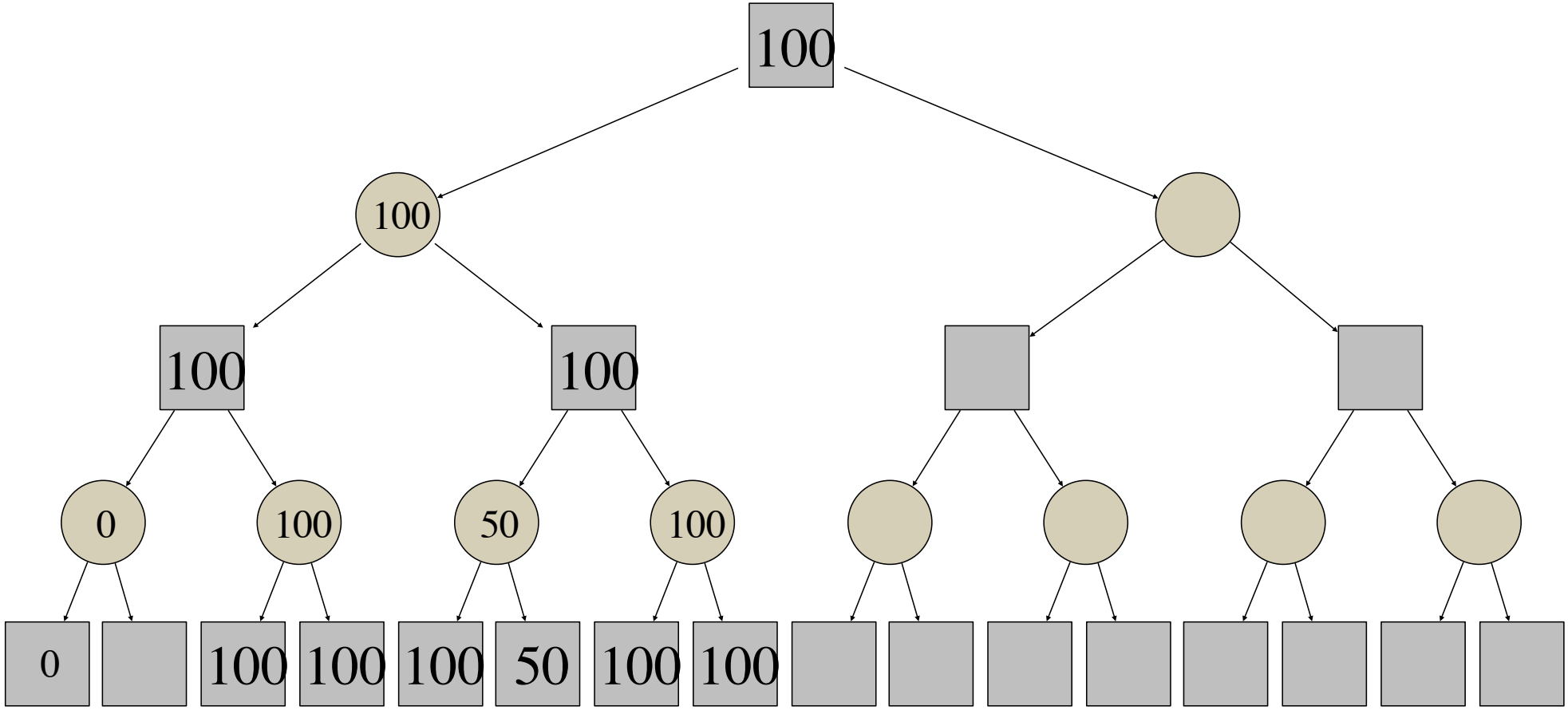
```
function maximize (state)
 {var actions = findlegals(state,library);
  if (actions.length===0) {return 0};
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var newstate = simulate(actions[i],state,library);
       var newscore = minimax(newstate);
      if (newscore>score) {score =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

Fixed sum game - 51 is sufficient

Alpha-Beta

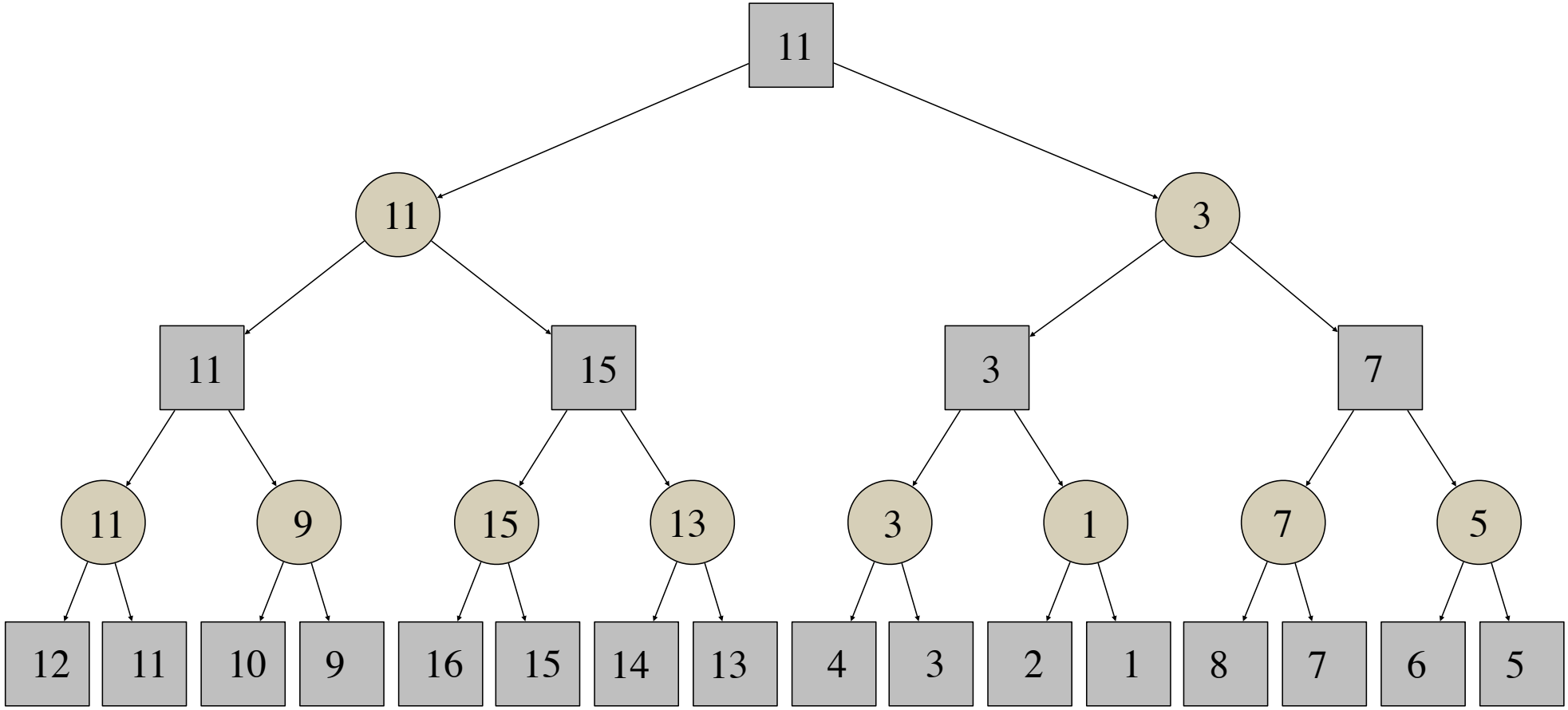
Alpha-Beta Search

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

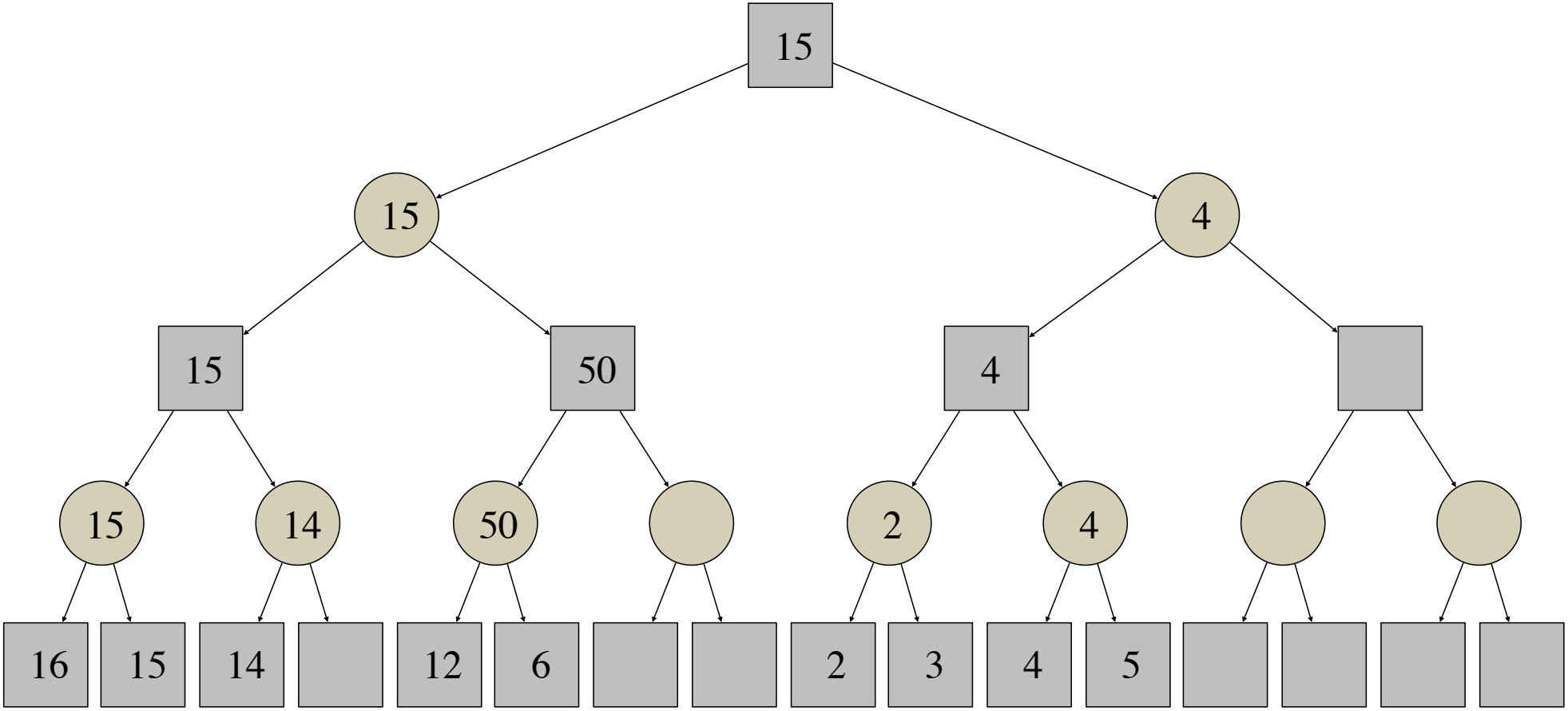
If partial result of min node less than alpha, can only decrease score and player need not consider.

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

Alpha-Beta 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.

For example, it could in some cases reduce a tree with branching factor of 25 to branching factor of 5.

Details

Creating Game Players

(1) Create your player code.

(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.

Gamemaster

Sign In

Resources

Game Development Utilities

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

Standaloneopen - Standalone manager for games described in local files

Manageropen - Regular manager for games described in local files

Human - Human player page for games on server

Player - Automated player page for games on server

Humanopen - Human player page for games described in local files

Playeropen - Automated player page for games described in local files

---

Standard Players

Legal - Legal player

Random - Random player

Onestep - One Step player

Twostep - Two Step player

Minimax - Full Minimax player

Minimaxdepth - Minimax player with fixed depth

Minimaxid - Minimax player with iterative deepening

Greedy - Greedy player

MCS - Monte Carlo Search player

Not Secure — gamemaster.stanford.edu

Gamemaster

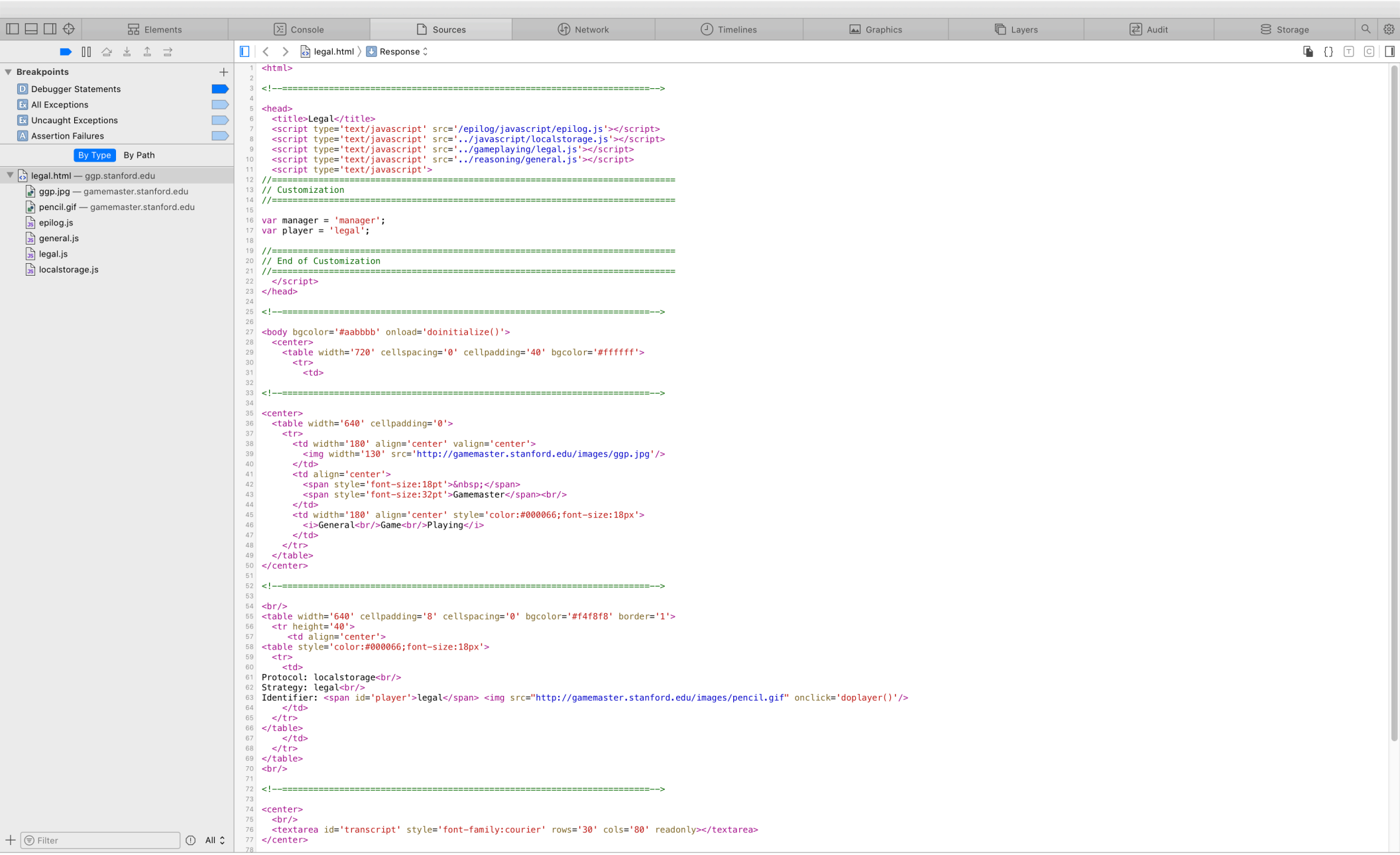
Sign In

Protocol: localstorage
Strategy: legal
Identifier: anonymous

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


Elements
Console
Sources
Network
Timelines
Graphics
Layers
Audit
Storage
Breakpoints
Debugger Statements
All Exceptions
Uncaught Exceptions
Assertion Failures
By Type
By Path
legal.html — ggp.stanford.edu
ggp.jpg — gamemaster.stanford.edu
pencil.gif — gamemaster.stanford.edu
epilog.js
general.js
legal.js
localstorage.js
Filter
All
legal.html
Response
<html>
<!--===-->
<head>
<title>Legal</title>
<script type='text/javascript' src='/epilog/javascript/epilog.js'></script>
<script type='text/javascript' src='../javascript/localstorage.js'></script>
<script type='text/javascript' src='../gameplaying/legal.js'></script>
<script type='text/javascript' src='../reasoning/general.js'></script>
<script type='text/javascript'>
//===
// Customization
//===
var manager = 'manager';
var player = 'legal';
//===
// End of Customization
//===
</script>
</head>
<!--===-->
<body bgcolor='#aabbbb' onload='doinitialize()'>

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

<table width='640' cellpadding='0'>
<tr>
<td width='180' align='center' valign='center'>

</td>
<td align='center'>

Gamemaster
</td>
<td width='180' align='center' style='color:#000066;font-size:18px'>
General
Game
Playing
</td>
</tr>
</table>

<!--===-->

<table width='640' cellpadding='8' cellspacing='0' bgcolor='#f4f8f8' border='1'>
<tr height='40'>
<td align='center'>
<table style='color:#000066;font-size:18px'>
<tr>
<td>
Protocol: localstorage
Strategy: legal
Identifier: legal
</td>
</tr>
</table>
</td>
</tr>
</table>

<!--===-->

<textarea id='transcript' style='font-family:courier' rows='30' cols='80' readonly></textarea>


GENERAL
GAME
PLAYING


GENERAL
GAME
PLAYING