

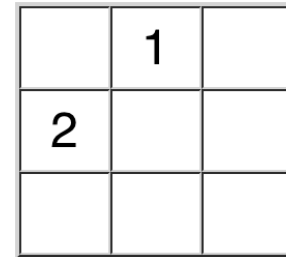
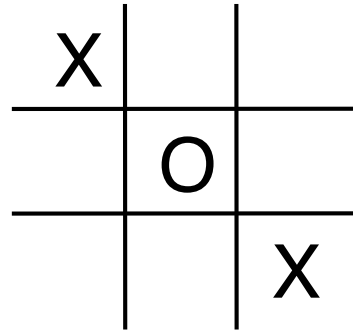
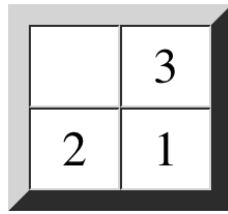
General Game Playing

Incomplete Search

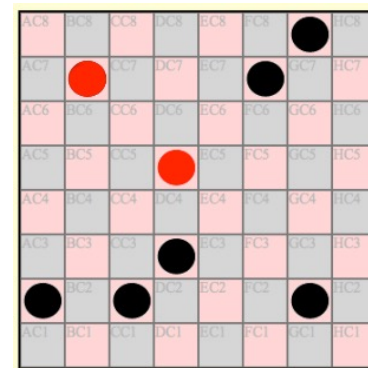
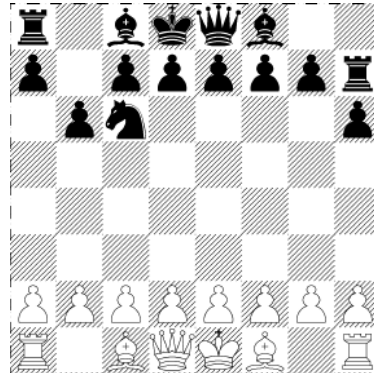
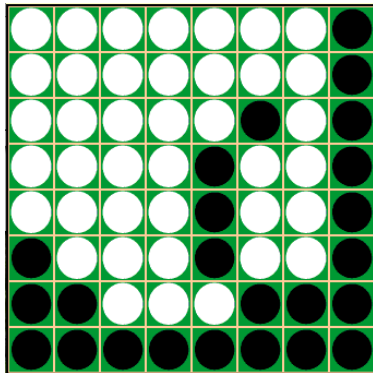
Michael Genesereth
Computer Science Department
Stanford University

Game Variety

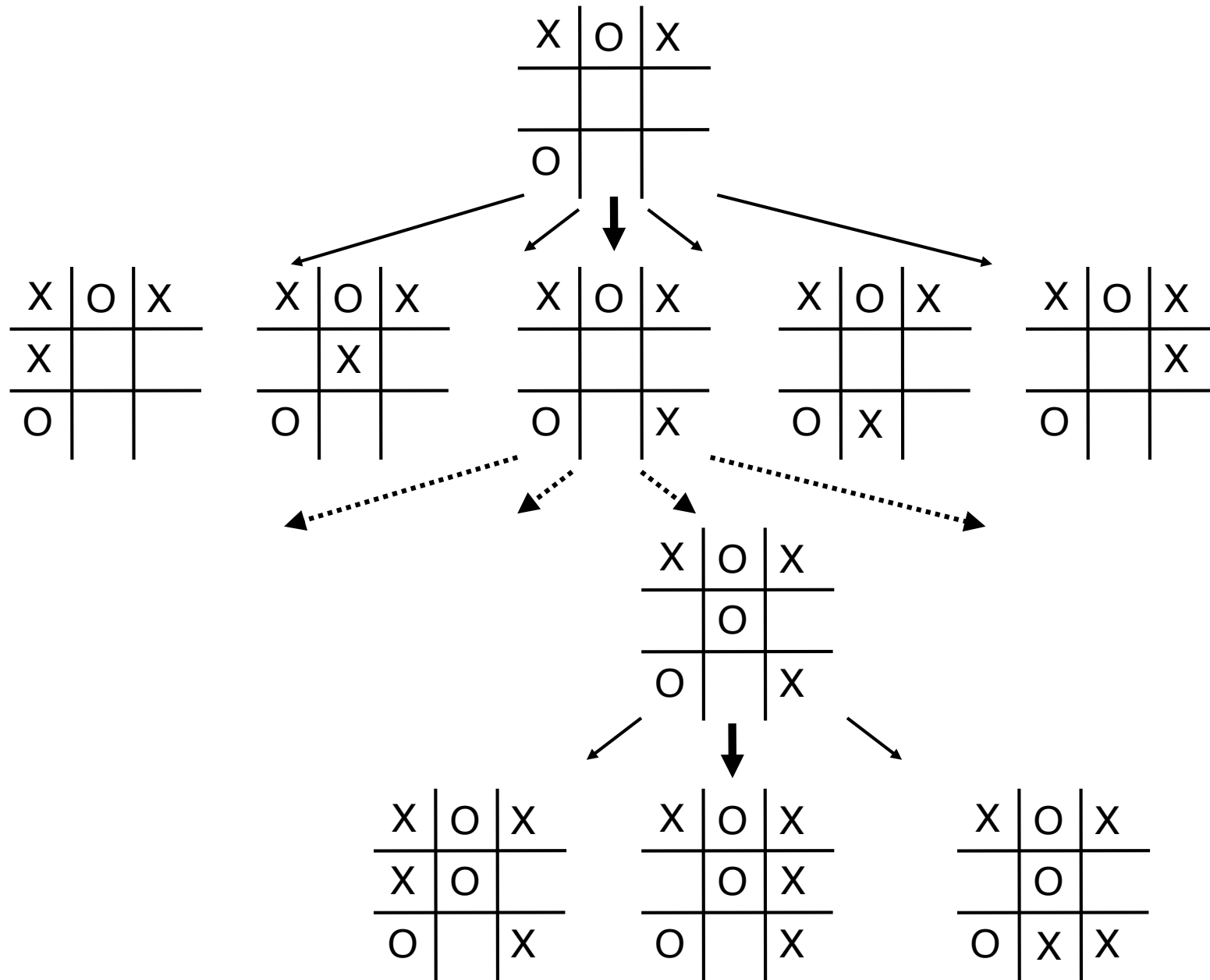
Small Games



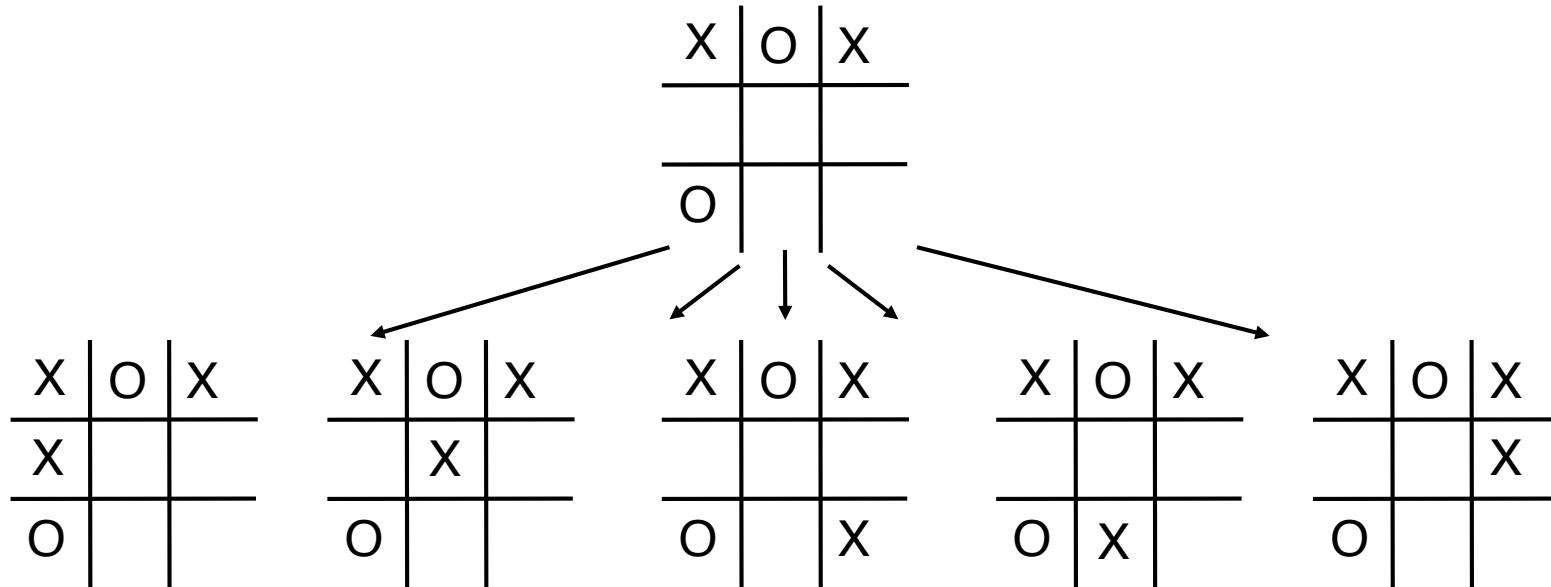
Large Games



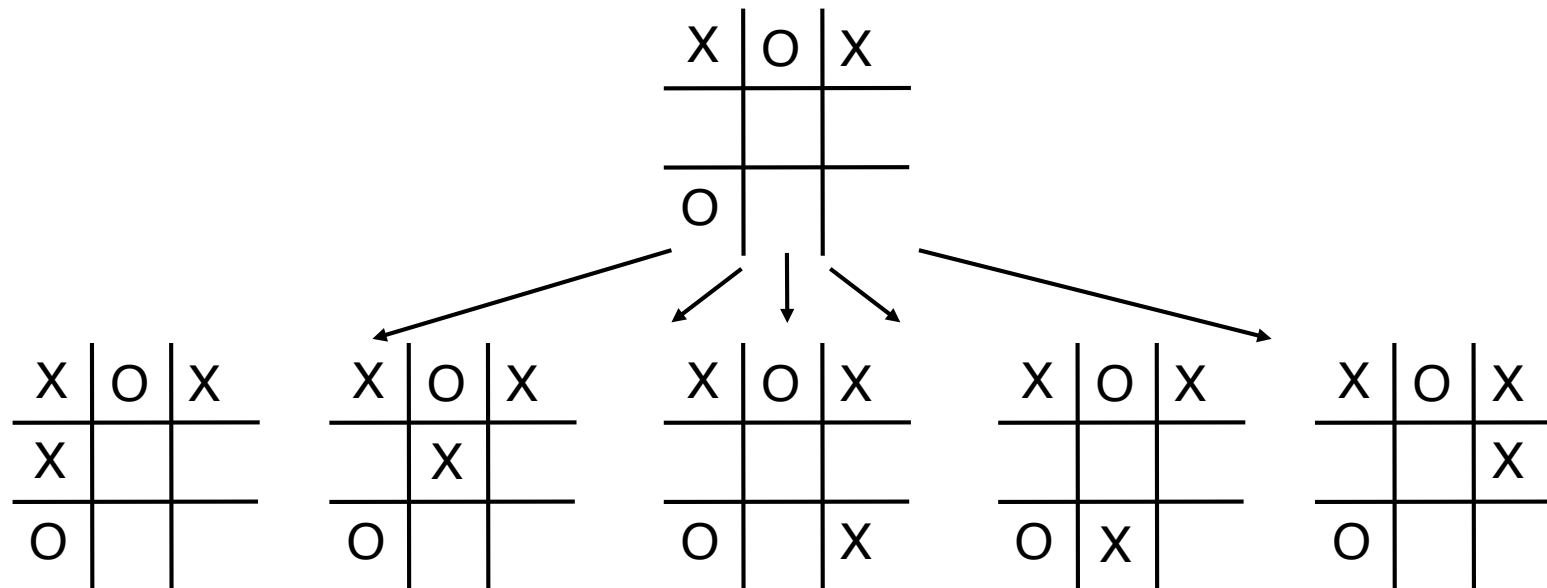
Complete Game Graph Search



Incomplete Search

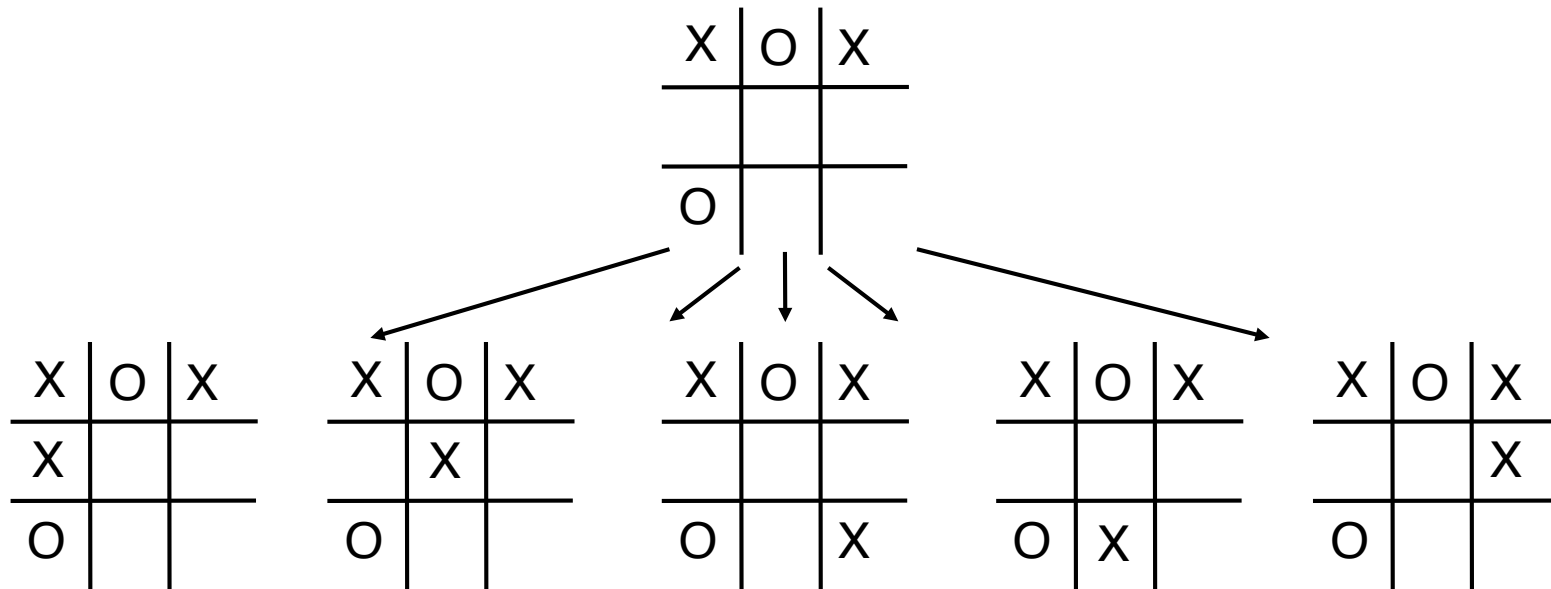


Evaluation of Non-Terminal States



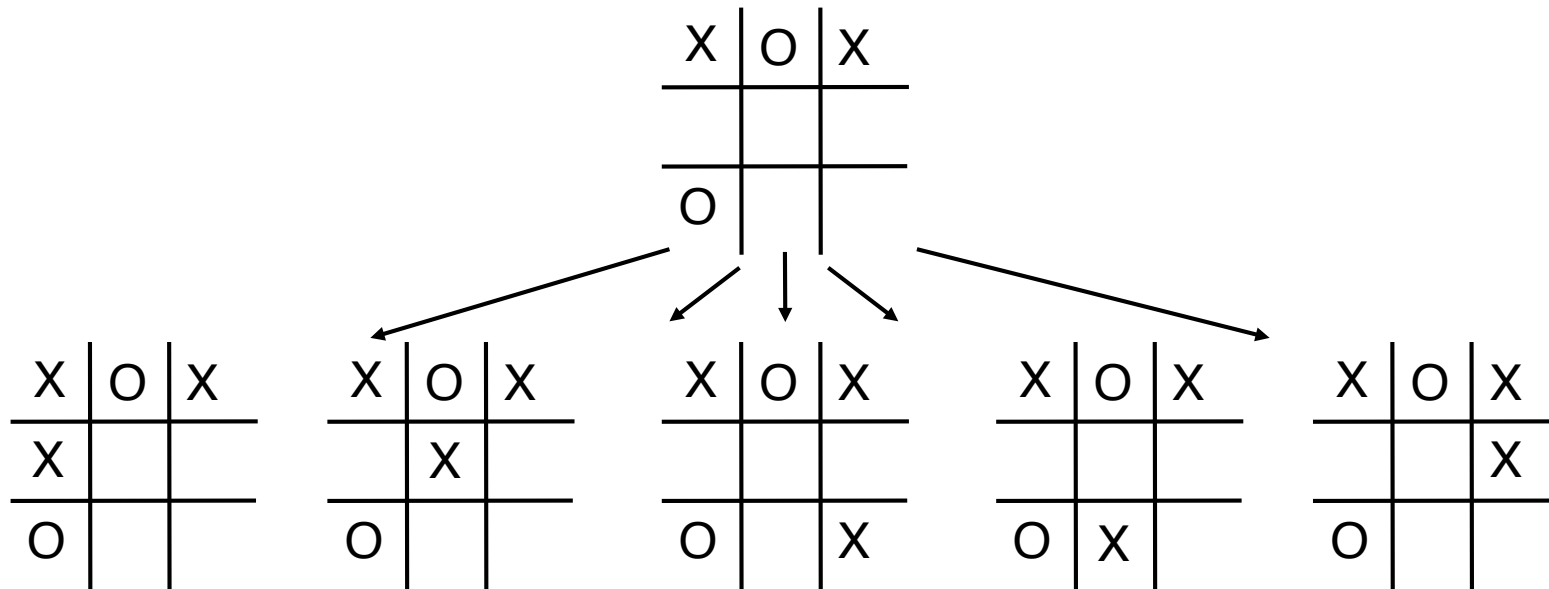
How do we evaluate non-terminal states?

Choice of Depth



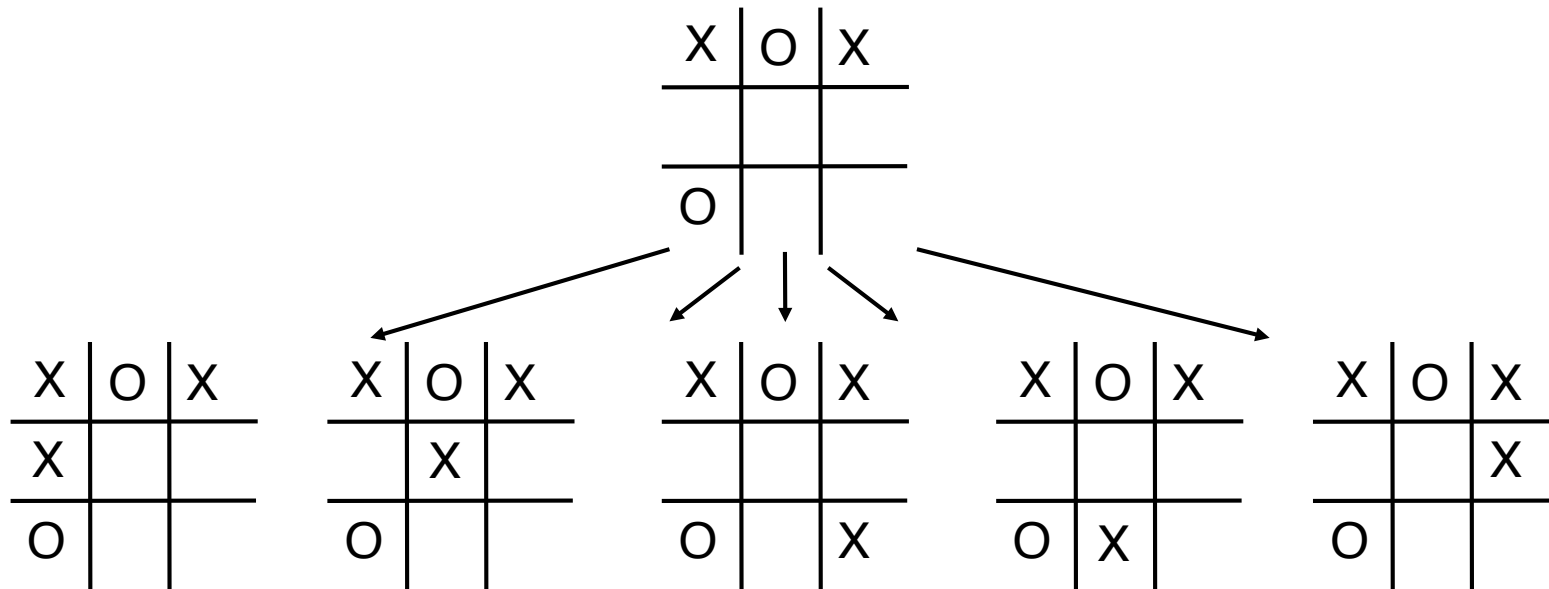
To what depth should we search?

Variable Depth Search



Should we search different branches to different depths?

Persistence



Can we preserve results across moves?

Evaluation Functions

Evaluation Functions

Chess examples:

Piece count

Board control

Comments

Not *necessarily* successful

Game-specific but this is *general* game playing

Heuristic #1 - Mobility / Focus

Mobility is a measure of the number of things a player can do. *Focus* is a measure of the narrowness of the search space. It is the opposite of mobility.

Basis - number of actions in a state or number of states reachable from that state. Horizon - current state or n moves away.

Sometimes it is good to focus to cut down on search space. Often better to restrict opponents' moves while keeping one's own options open.

Heuristic #1 - Mobility / Focus

Mobility is a measure of the number of things a player can do. *Focus* is a measure of the narrowness of the search space. It is the opposite of mobility.

Basis - number of actions in a state or number of states reachable from that state. Horizon - current state or n moves away.

Sometimes it is good to focus to cut down on search space. Often better to restrict opponents' moves while keeping one's own options open.

Implementation

```
function mobility (state)
  {var actions = findlegals(state,library);
   var feasibles = findactions(library);
   return (actions.length/feasibles.length * 100)}
```

```
function focus (state)
  {var actions = findlegals(state,library);
   var feasibles = findactions(library);
   return (100 - actions.length/feasibles.length * 100)}
```

GGP-06 Final - Cylinder Checkers

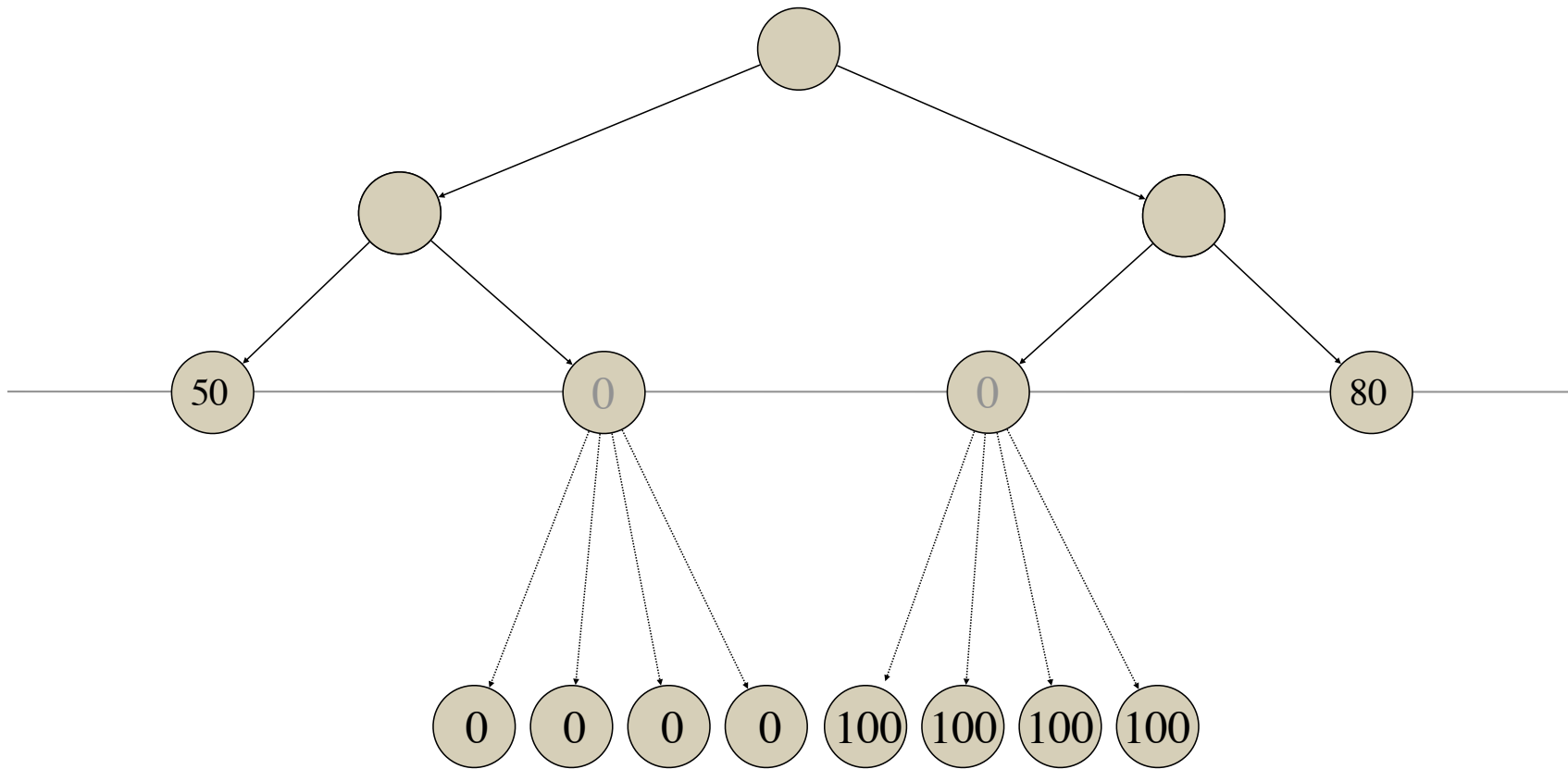
AC8	BC8	CC8	DC8	EC8	FC8	GC8	HC8
AC7	●	CC7	DC7	EC7	●	GC7	HC7
AC6	BC6	CC6	DC6	EC6	FC6	GC6	HC6
AC5	BC5	CC5	●	EC5	FC5	GC5	HC5
AC4	BC4	CC4	DC4	EC4	FC4	GC4	HC4
AC3	BC3	CC3	●	EC3	FC3	GC3	HC3
●	BC2	●	DC2	EC2	FC2	●	HC2
AC1	BC1	CC1	DC1	EC1	FC1	GC1	HC1

Heuristic #2 - Pessimism

Assume value of 0 for non-terminal states.

$$\begin{aligned} \textit{value}(\textit{state}) &= \textit{goal}(\textit{role}, \textit{state}) && \textit{if } \textit{terminal}(\textit{state}) \\ \textit{value}(\textit{state}) &= 0 && \textit{otherwise} \end{aligned}$$

Example



*Grey - estimates of rewards in non-terminal states - here 0.
Black - rewards in terminal states.*

Heuristic #3 - Intermediate Values

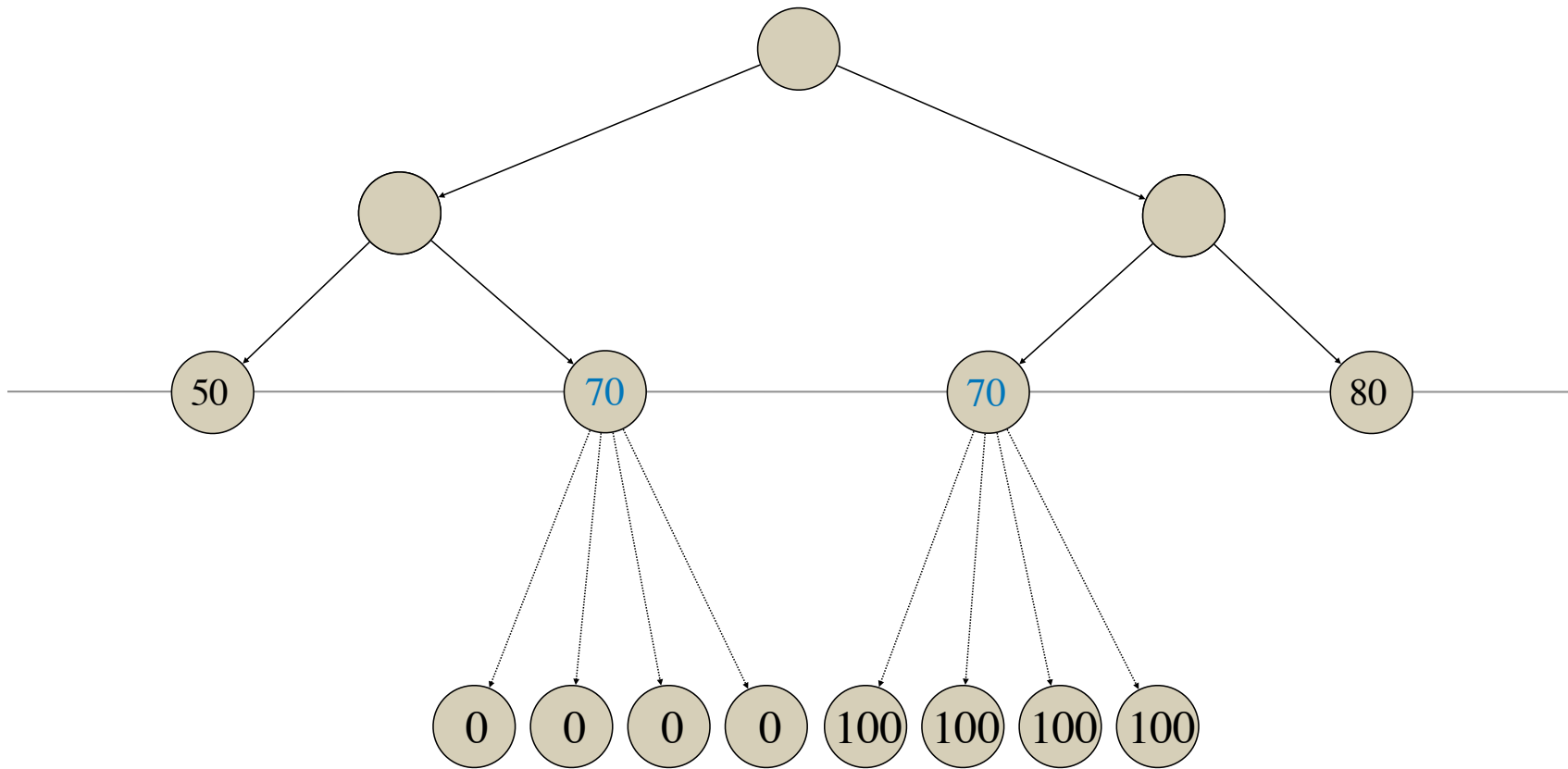
Assume reward for non-terminal states.

$$value(state) = goal(role, state)$$

Good on monotonic games (where utility accumulates as the game progresses), e.g. alquerque.

Not so good on nonmonotonic games. Susceptible to "false summits".

Example



*Blue - rewards in non-terminal states.
Black - rewards in terminal states.*

Weighted Linear Combinations

Definition

$$f(s) = w_1 \times f_1(s) + \dots + w_n \times f_n(s)$$

Examples:

Mobility / Focus

Intermediate State Values

Other

Some players estimate weights by experimentation during the start clock. *More on this in a few weeks.*

Weighted Linear Combinations

Definition

$$f(s) = w_1 \times f_1(s) + \dots + w_n \times f_n(s)$$

Examples:

Mobility / Focus

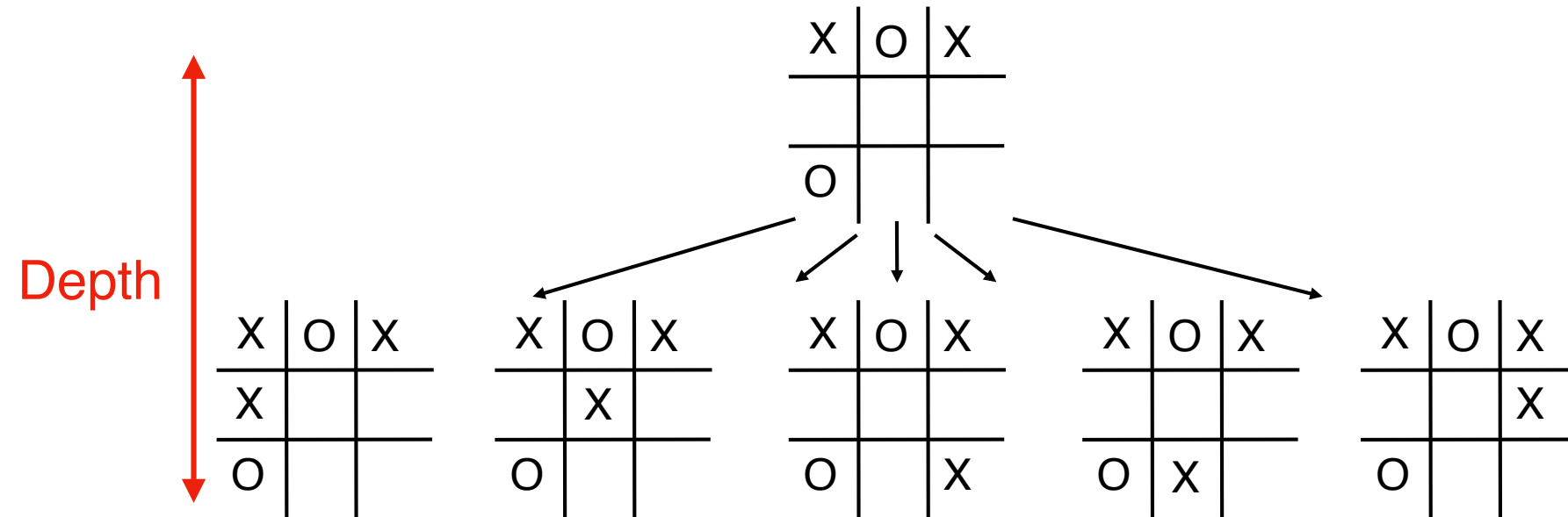
Intermediate State Values

Other

Some players estimate weights by experimentation during the start clock. *More on this in a few weeks.*

Depth-Limited Search

Depth-Limited Search



Depth-Limited Minimax

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)}
```

Depth-Limited Minimax

```
function minimaxdepth (state,depth)
  {if (findterminalp(state,library))
    {return findreward(role,state,library)*1};
  if (depth<=0) {return evalfun(state,library)};
  var active = findcontrol(state,library);
  if (active===role) {return maxscore(state,depth-1)};
  return minscore(state,depth-1)}
```

maxscore and minscore

```
function maxscore (state,depth)
{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 = minimaxdepth(newstate,depth);
      if (newscore===100) {return 100};
      if (newscore>score) {score = newscore}};
  return score}
```

```
function minscore (state,depth)
{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 = minimaxdepth(role,newstate,depth);
      if (newscore===0) {return 0};
      if (newscore<score) {score = newscore}};
  return score}
```

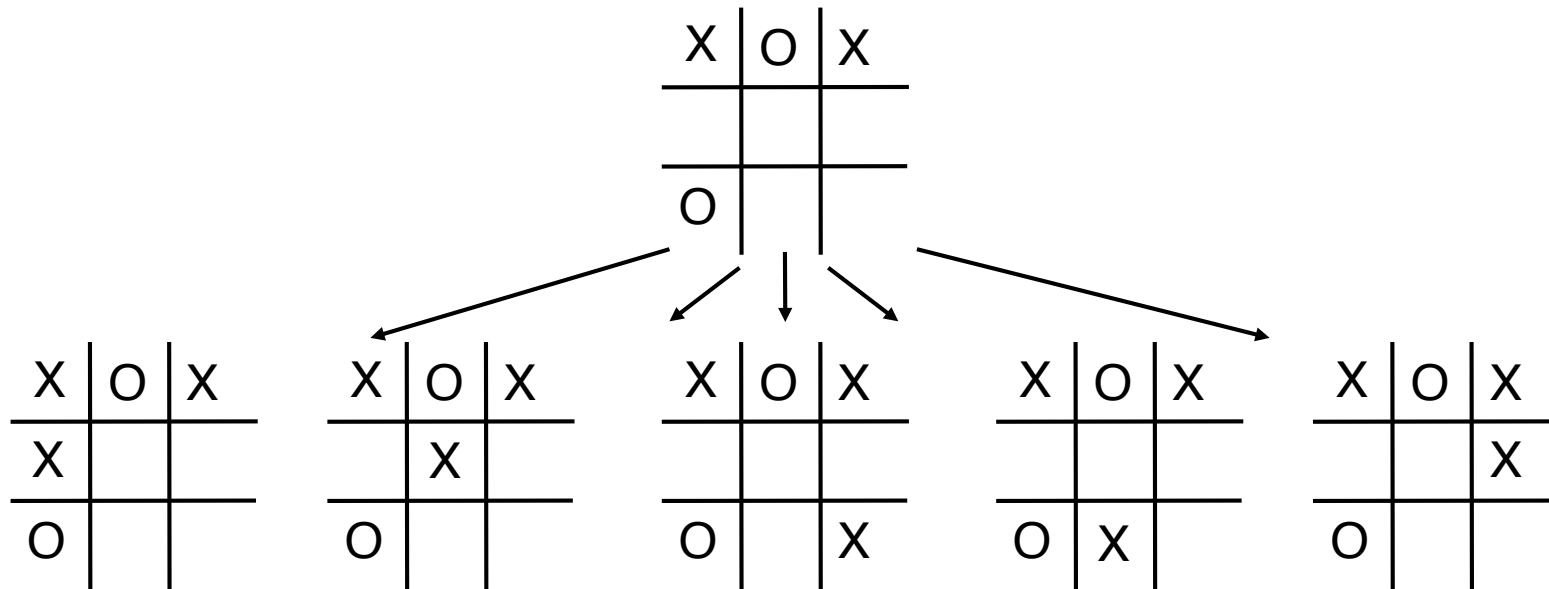

Remarks

Legal and random players are degenerate depth-limited search with depth 0.

Onestep and Twostep are degenerate depth-limited search with depths 1 and 2.

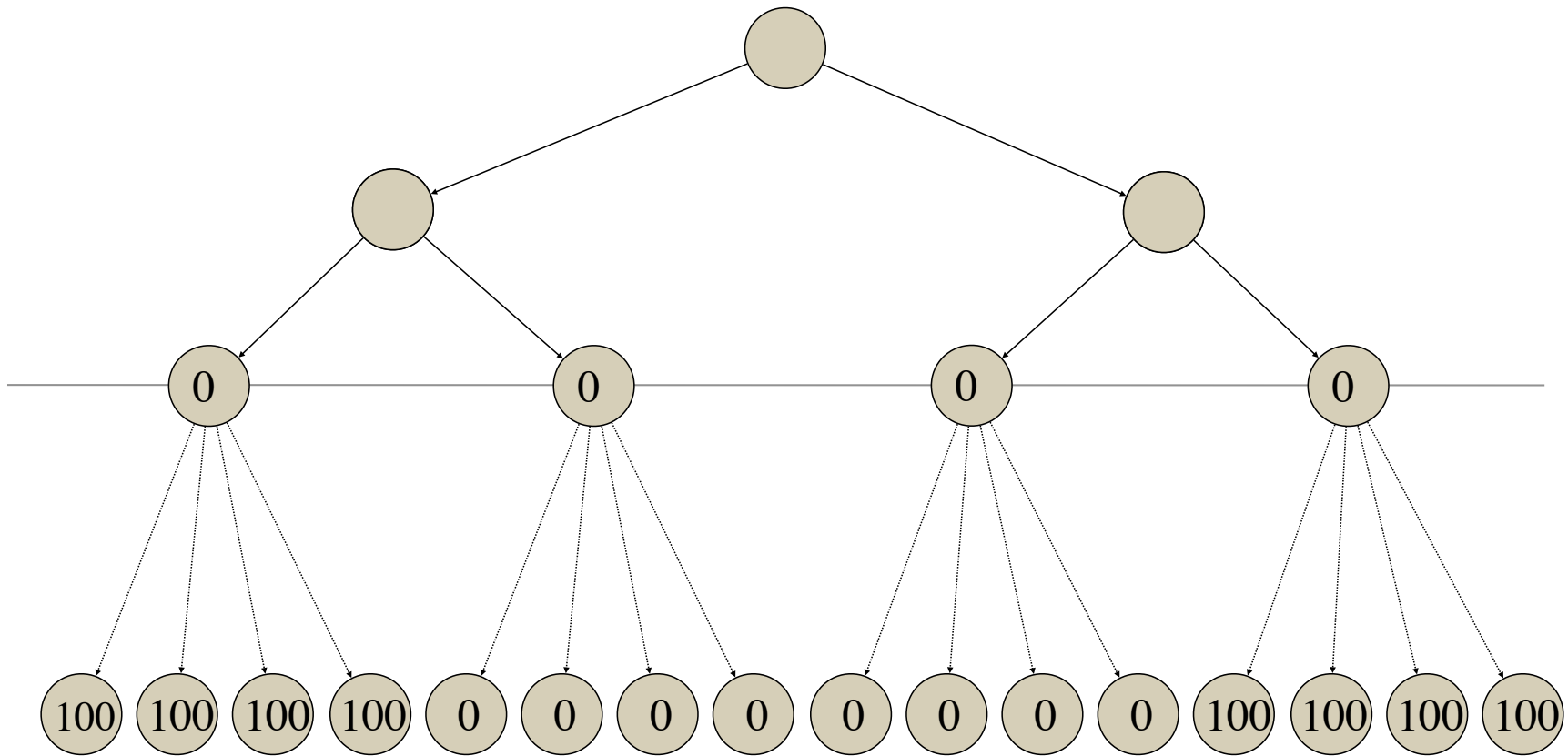
In general, we would like to allow greater depths.

Problem

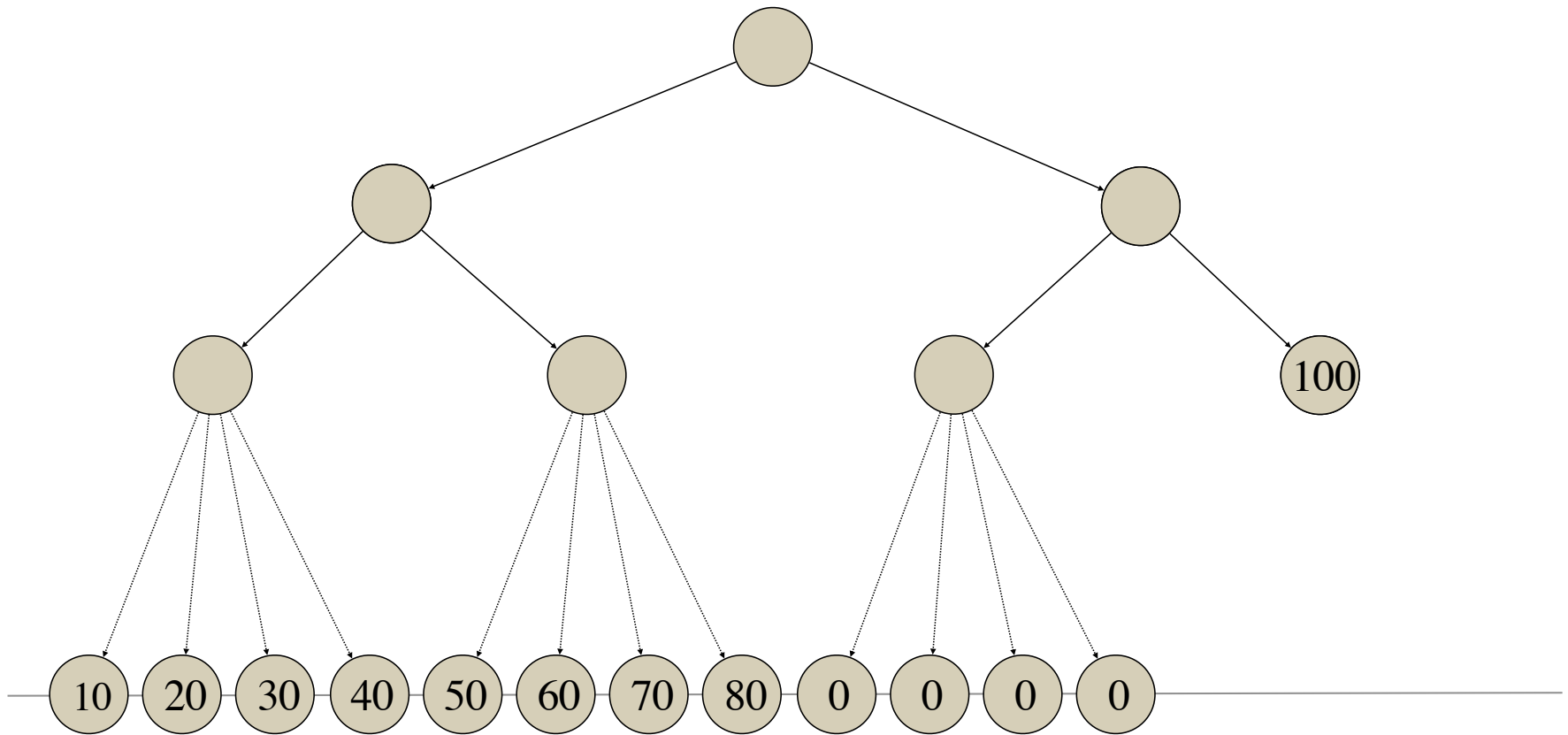


To what depth should we search?

Problem - Insufficient Depth



Problem - Excessive Depth



Iterative Deepening

Iterative Deepening

Use depth-limited search to explore entire tree to level 1

Use depth-limited search to explore entire tree to level 2

Use depth-limited search to explore entire tree to level 3

And so forth

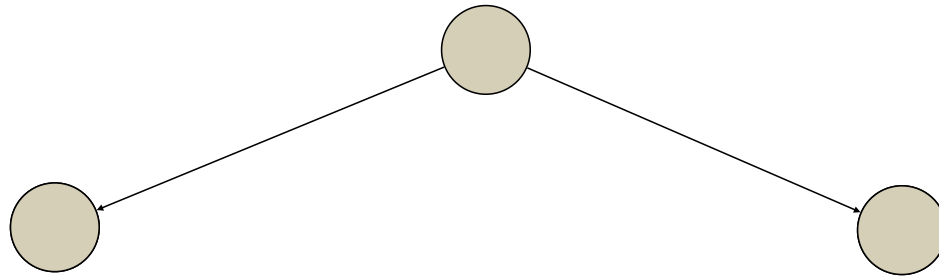
Continue till time runs out

Choose action that gives maximal value

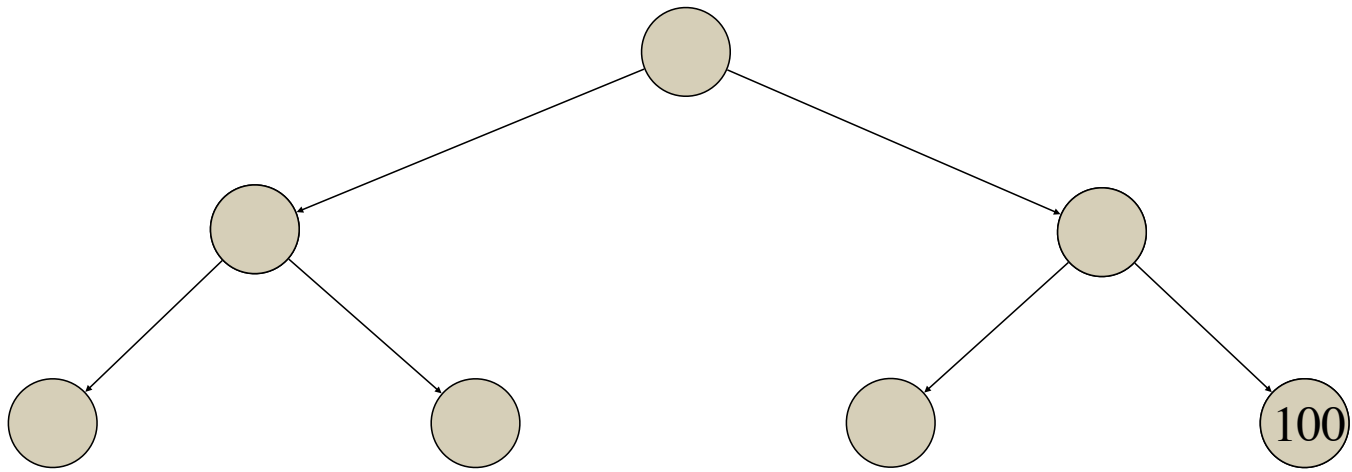
Level 1



Level 2



Level 3



Naive Implementation

```
function playminimaxid ()  
  {var best = findlegalx(state,library);  
  for (var depth=1; depth<10; depth++)  
    {var action = minimaxdepth(state,depth);  
    best = action};  
  return best}
```

At what depth do we stop?

Implementation

```
function playminimaxid ()
{var deadline = Date.now()+(playclock-1)*1000;
  var best = findlegalx(state,library);
  for (var depth=1; depth<10; depth++)
    {var action = playminimaxidinner(state,depth,deadline);
      if (action===false) {return best};
      best = action};
  return best}
```

Implementation

```
function playminimaxidinner (state,depth,deadline)
{var actions = shuffle(findlegals(state,library));
  var best = actions[0];
  var score = 0;
  for (var i=0; i<actions.length; i++)
    {var newstate = simulate(actions[i],state,library);
      var newscore = minimaxid(newstate,depth,deadline);
      if (newscore===false) {return false};
      if (newscore===100) {return actions[i]};
      if (newscore>score) {best = actions[i]; score=newscore}};
  return best}
```

Implementation

```
function minimaxid (state,depth,deadline)
{if (findterminalp(state,library))
  {return findreward(role,state,library)*1};
if (depth<=0) {return evalfun(state,library)*1};
if (Date.now()>deadline) {return false};
if (findcontrol(state,library)===role)
  {return maxscoreid(state,depth,deadline)};
return minscoreid(state,depth,deadline)}
```

maxscore and minscore

```
function maxscore (state,depth,deadline)
{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 = minimaxid(newstate,depth,deadline);
      if (newscore===false) {return false};
      if (newscore===100) {return 100};
      if (newscore>score) {score = newscore}};
  return score}
```

```
function minscore (state,depth,deadline)
{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 = minimaxid(newstate,depth,deadline);
      if (newscore===false) {return false};
      if (newscore===0) {return 0};
      if (newscore<score) {score = newscore}};
  return score}
```

Advantages and Disadvantages

Advantages

requires storage linear in depth

still finds shortest path to an optimal solution

Disadvantages (?)

Repeated work

but

Cost only a constant factor more than depth-first search

Why? Tree is growing exponentially, so fringe of tree and size of tree above fringe are approximately same

More Information

[https://en.wikipedia.org/wiki/
Iterative_deepening_depth-first_search](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search)

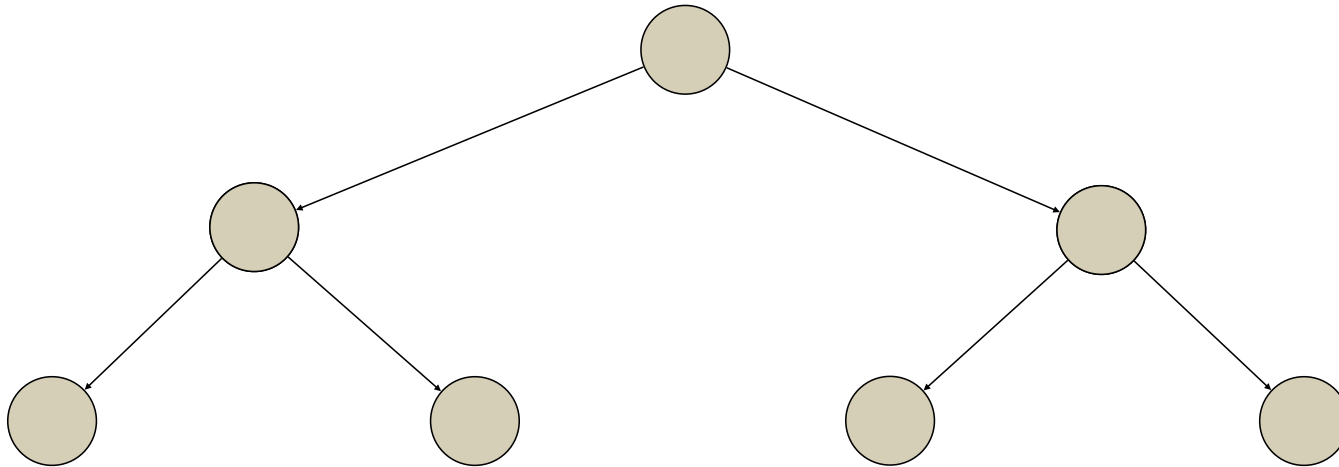
Monte Carlo Search

Basic Idea

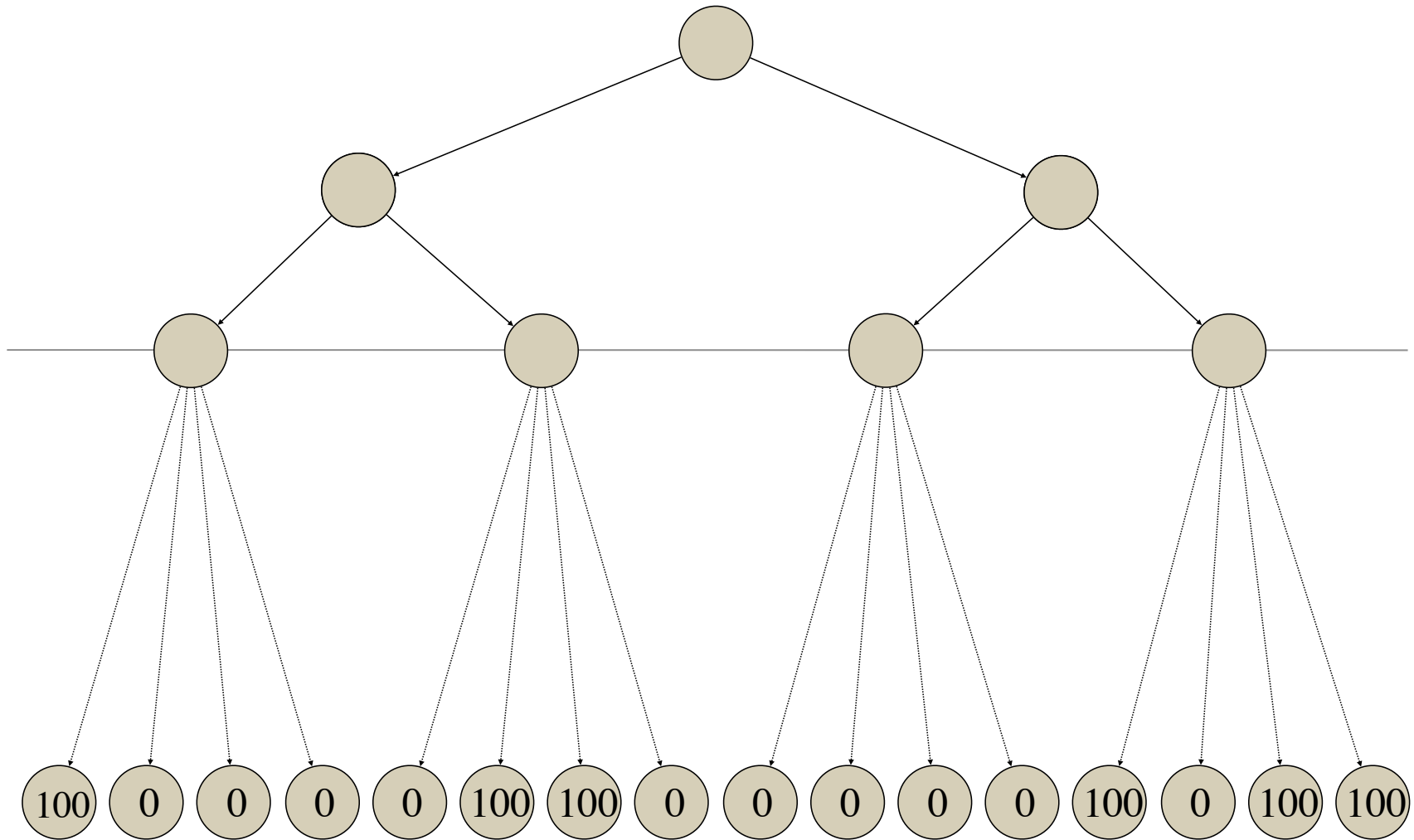
Sample a few branches of the game tree and use results to estimate values.

- (1) Optionally explore game graph to some level.
- (2) Beyond this, explore to end of game from fringe nodes, making random choices for moves of all players.
- (3) Assign expected utilities to fringe states by summing utilities and dividing by number of trials.

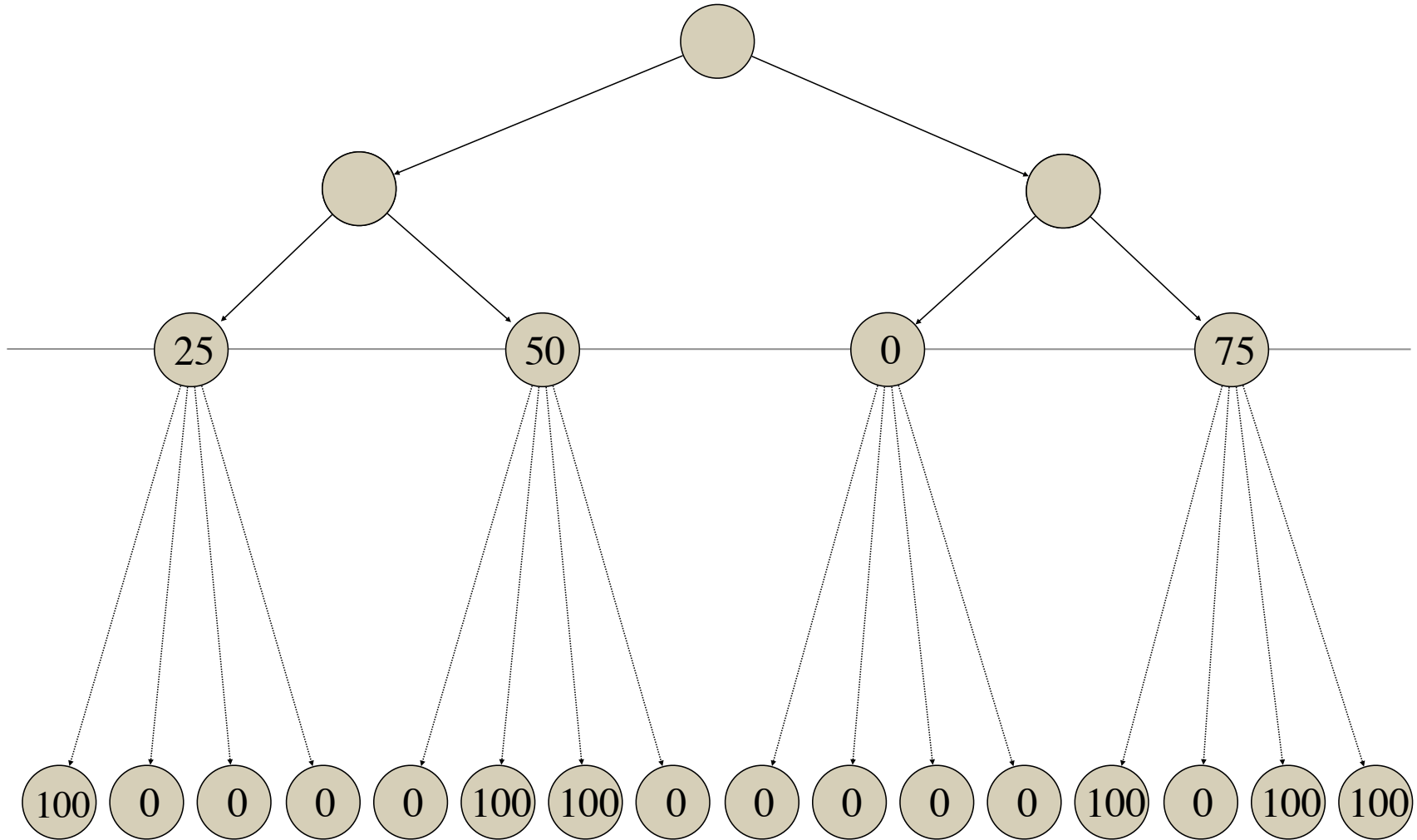
Example



Example



Example



mcs

```
function mcs (state, level)
  {if (findterminalp(state, library))
    {return findreward(role, state, library)*1};
  if (level>levels) {return montecarlo(state)};
  var active = findcontrol(state, library);
  if (active===role) {return maxscore(state, level+1)};
  return minscore(state, level+1)}
```

maxscore and minscore

```
function maxscore (state,level)
  {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 = mcs(newstate,level);
      if (newscore===100) {return 100};
      if (newscore>score) {score = newscore}};
   return score}
```

```
function minscore (state,level)
  {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 = mcs(role,newstate,level);
      if (newscore===0) {return 0};
      if (newscore<score) {score = newscore}};
   return score}
```

Implementation

```
function montecarlo (state)
  {var total = 0;
   for (var i=0; i<count; i++)
     {total = total + depthcharge(state)};
   return total/count}

function depthcharge (state)
  {if (findterminalp(state,ruleset))
    {return findreward(role,state,ruleset)}*1;
   var move = seq();
   for (var i=0; i<roles.length; i++)
     {var options = findlegals(roles[i],state,library);
      var best = randomindex(options.length);
      move[i] = options[best]};
   var newstate = simulate(move,state,library);
   return depthcharge(newstate)}
```


Problems and Features

Problems

Optimistic - opponent might not respect probabilities
Does not utilize game structure in any useful way

Problems and Features

Problems

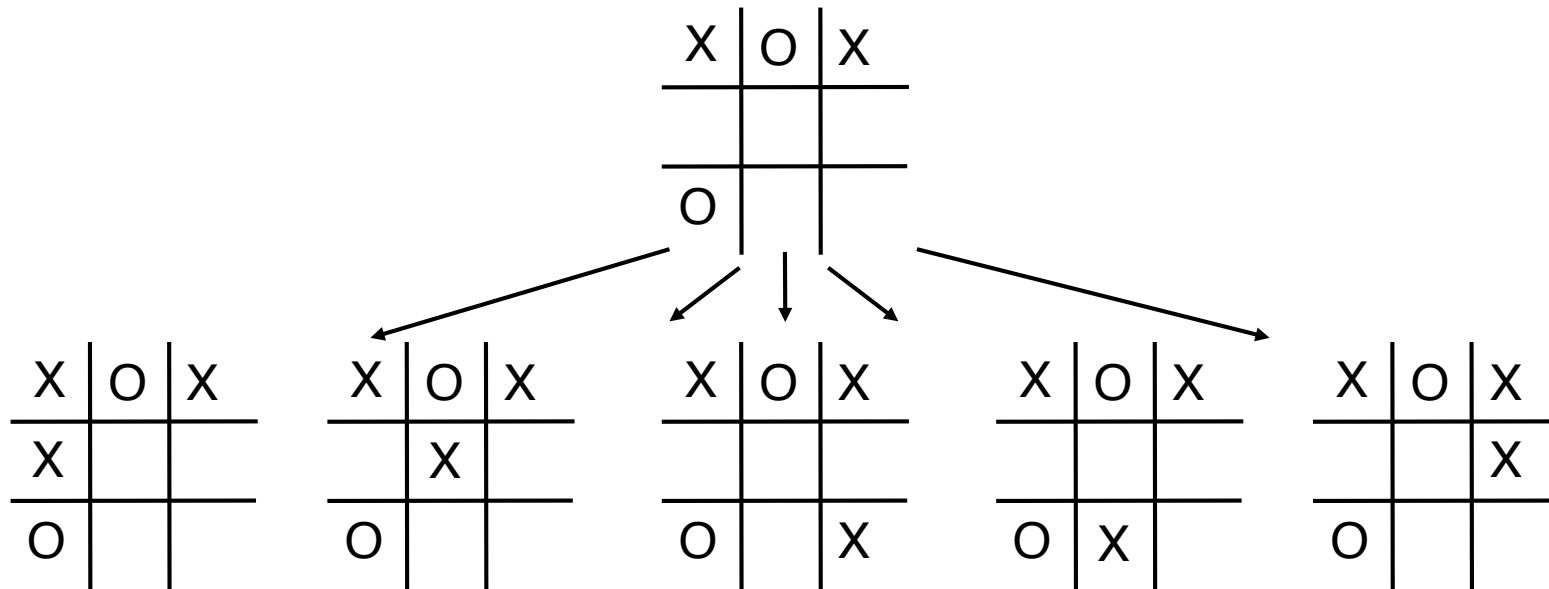
- Optimistic - opponent might not respect probabilities
- Does not utilize game structure in any useful way

Benefits

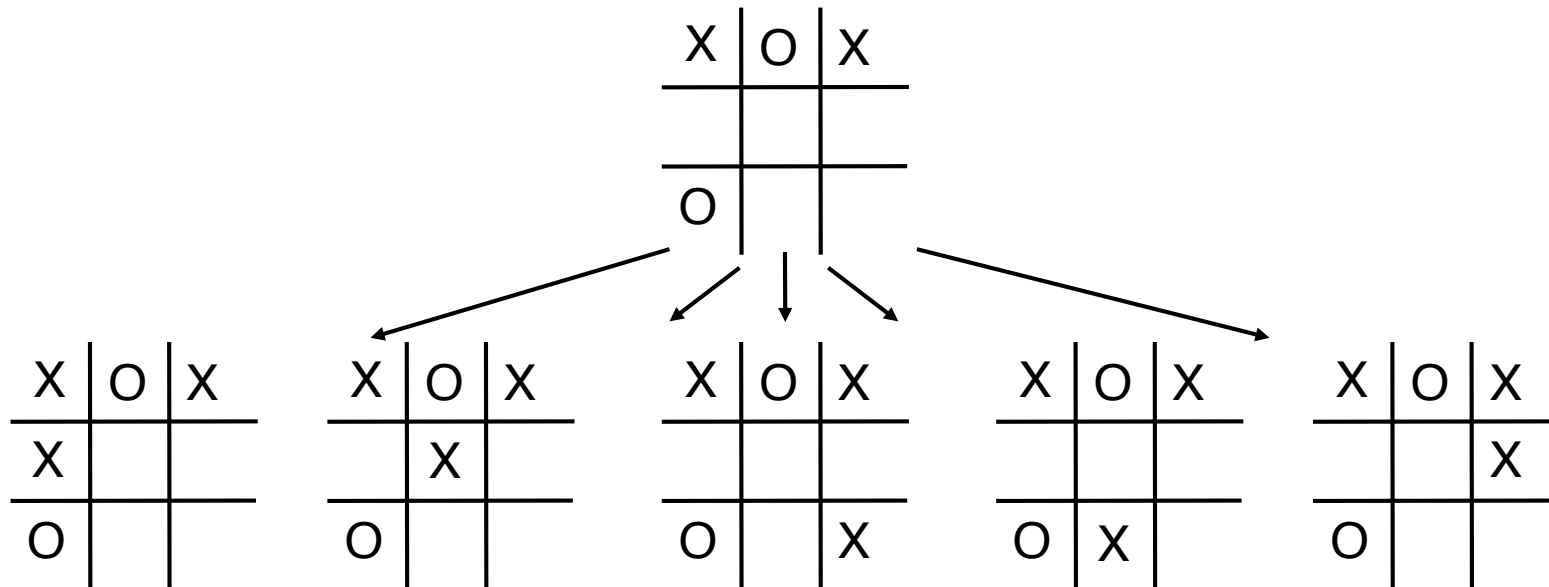
- Fast because no branching in depth charges
- Small space because nothing stored in probes
- Provides guidance when other heuristics fail

Issues

Incomplete Search

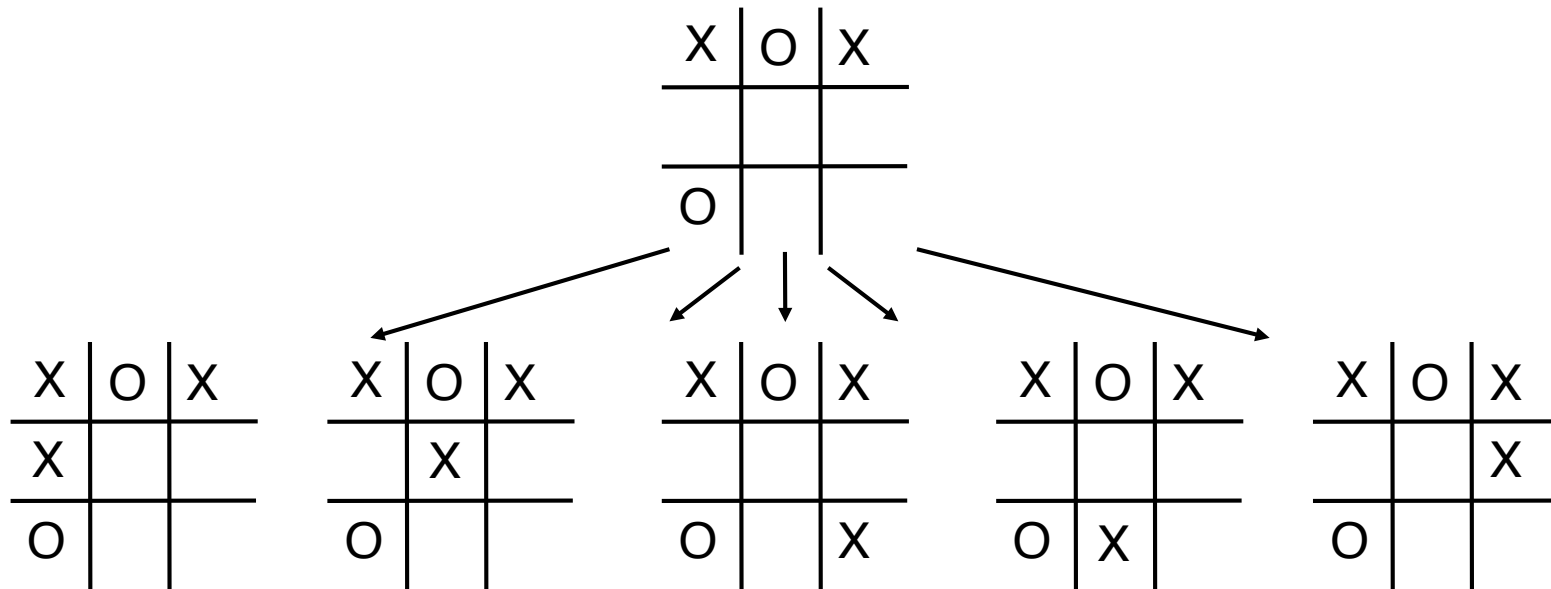


Evaluation of Non-Terminal States



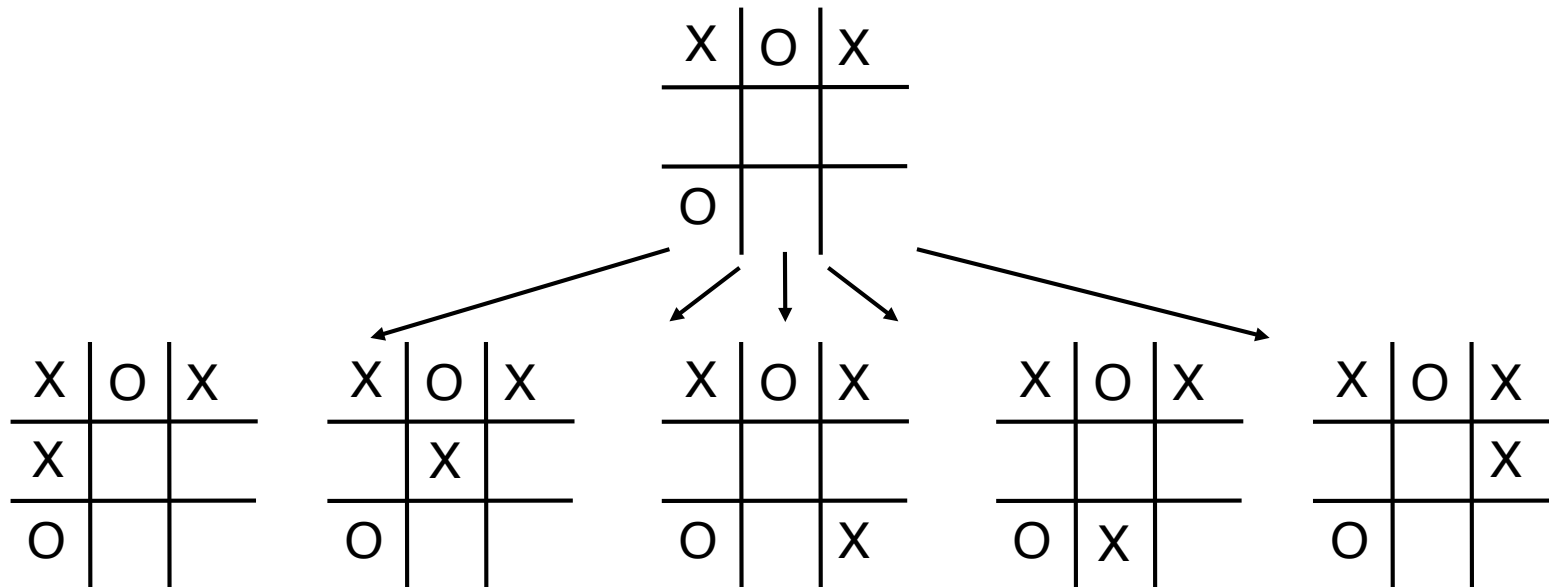
How do we evaluate non-terminal states? ✓

Choice of Depth



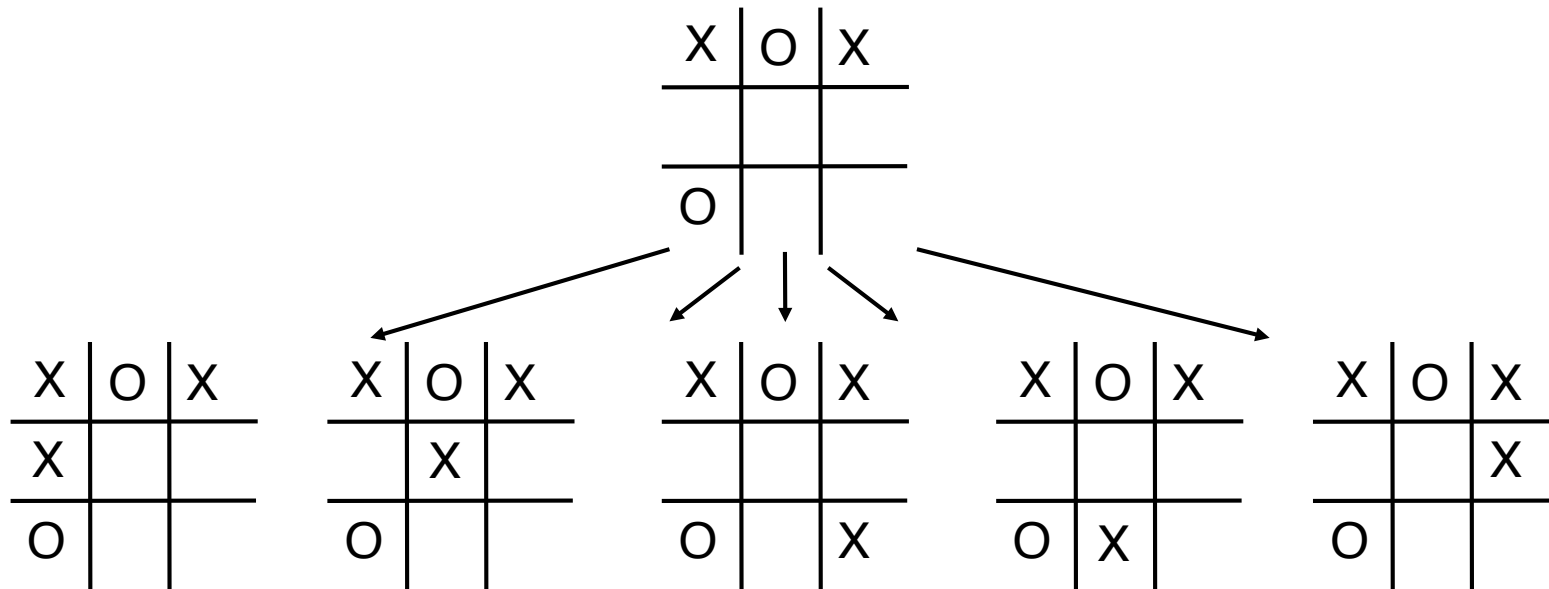
To what depth should we search? ✓

Variable Depth Search



Can we search different branches to different depths?

Persistence



Can we preserve tree across moves or phases of ID?



**GENERAL
GAME
PLAYING**





**GENERAL
GAME
PLAYING**

