Incomplete Search
Evaluation of Non-Terminal States

How do we evaluate non-terminal states?
Choice of Depth

To what depth should we search?
Can we search different branches to different depths?
Persistence

*Can we preserve tree across moves or phases of ID?*
Greedy
Overview

Create a game tree data structure.

*Select* branch of tree, sub-branch, etc. till reach fringe.
*Expand* fringe node.
*Propagate* values back to the root.

Repeat until time runs out.
Select move w/ best estimated value.

Replace game tree with subtree as each move is made.
var role, roles, state, library, startclock, playclock;
var tree = {};

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 makenode (state, mover, reward) {
    {return {state: state,
        actions: [],
        children: [],
        mover: mover,
        utility: reward}}
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-2)*1000;
    while (Date.now()<deadline) {process(tree)};
    return selectaction(tree)
Updating the Tree After Move
function subtree (move, node) {
    if (node.children.length === 0) {expand(node)};
    for (var i=0; i<node.actions.length; i++)
        {if (equalp(move, node.actions[i]))
            {return node.children[i]}}
    return node}
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-2)*1000;
while (Date.now()<deadline) {process(tree)};
return selectaction(tree)}
Processing the Tree

Select → Expand → Backpropagate
function process (node)
    {if (findterminalp(node.state,library)) {return true};
        if (node.children.length===0) {expand(node)}
        else {process(select(node))};
    update(node);
    return true}
function select (node)
{
    var total = node.visits;
    var child = node.children[0];
    var score = child.utility;
    for (var i=1; i<node.children.length; i++)
    {
        var newchild = node.children[i];
        var newscore = newchild.utility;
        if (newscore>score)
        {
            child = newchild; score = newscore;
        }
    }
    return child
}
function expand (node)
    {node.actions = findlegals(node.state,library);
    for (var i=0; i<node.actions.length; i++)
        {var newstate = simulate(node.actions[i],node.state,library);
         var newmover = findcontrol(newstate,library);
         var newscore = parseInt(findreward(role,newstate,library));
         node.children[i] = makenode(newstate,newmover,newscore)};
    return true}

function makenode (state,mover,reward)
    {return {state:state,
             actions:[],
             children:[],
             mover:mover,
             utility:reward}}
function update (node) {
    if (node.mover===role) {
        node.utility = scoremax(node);
    } else {
        node.utility = scoremin(node);
    }
    return true
}

function scoremax (node) {
    var score = node.children[0].utility;
    for (var i=1; i<node.children.length; i++) {
        var newscore = node.children[i].utility;
        if (newscore>score) {score = newscore};
    }
    return newscore
}

function scoremin (node) {
    var score = node.children[0].utility;
    for (var i=1; i<node.children.length; i++) {
        var newscore = node.children[i].utility;
        if (newscore<score) {score = newscore};
    }
    return newscore
}
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-2)*1000);
    while (Date.now()<deadline) {process(tree)};
    return selectaction(tree)}
function selectaction (node)
{var action = node.actions[0];
 var score = node.children[0].utility;
 for (var i=1; i<node.children.length; i++)
 {var newscore = node.children[i].utility;
   if (newscore>score)
     {action = node.actions[i]; score = newscore};
 return action}
Selection Functions
function select (node)
    {var total = node.visits;
     var child = node.children[0];
     var score = child.utility;
     for (var i=1; i<node.children.length; i++)
         {var newchild = node.children[i];
          var newscore = newchild.utility;
          if (newscore>score)
              {child = newchild; score = newscore};
     return child}
function makenode (state,mover,reward)
  {return {state:state, actions:[], children:[], mover:mover,
    utility:reward, visits:0}}

function update (node)
  {if (node.mover===role) {node.utility = scoremax(node)}
    else {node.utility = scoremin(node)};
    node.visits = node.visits+1;
    return true}

function selectnode (node)
  {var child = node.children[0];
    var visits = node.children[0].visits;
    for (var i=1; i<node.children.length; i++)
      {var newvisits = node.children[i].visits;
        if (newvisits<visits)
          {child = node.children[i]; visits = newvisits}};
    return child}
function select (node)
    {var total = node.visits;
     var child = node.children[0];
     var score = value(child.utility,child.visits,total);
     for (var i=1; i<node.children.length; i++)
         {var newchild = node.children[i];
          var newvalue = newchild.utility;
          var newvisits = newchild.visits;
          var newscore = value(newvalue,newvisits,total);
          if (newscore>score)
              {child = newchild; score = newscore}};
    return child}

function value (utility,visits,total)
    {var score = (utility + Math.round((1 - visits/total)*100));
    return score}
Ideas

Some players use different formulas for combing Exploitation and Exploration.

\[ \text{selectValue} = \frac{\text{node.utility}}{\text{node.visits}} + C \times \sqrt{\frac{\ln(\text{node.parent.visits})}{\text{node.visits}}} \]

Some players find value in recording the standard deviation of scores at each node or where the change in estimated values and favor expanding those nodes with the highest value to clarify the uncertainty.
The preceding implementation does not distinguish between values that are *guaranteed* (because the player searched to the end of the tree) from those that are *estimated* (based on heuristic evaluation function).

Fix by adding additional information to each node.
Monte Carlo Tree Search
Monte Carlo Tree Search (MCTS) is a search method that relies on random probes to estimate state values. Blend of Greedy and MCS.

Like MCS:
- Builds up game tree incrementally
- Random probes to end of game to estimate state values

Like Greedy:
- MCTS expands non-uniformly
- Uses combination of Exploitation and Exploration
Overview

Select

[Diagram of tree structure]
Overview
Overview

Select → Expand → Simulate → Backpropagate
Single Player
Select
Select

- u=180, v=7
  - u=100, v=3
    - u=0, v=1
    - u=100, v=2
      - u=0, v=1
  - u=80, v=3
  - u=0, v=1
Select

\[
\text{selectValue} = \frac{\text{node.utility}}{\text{node.visits}} + C \times \sqrt{\frac{\ln(\text{node.parent.visits})}{\text{node.visits}}}
\]
Select
Select

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```

```
Select
```
Simulate
Simulate
Simulate
Backpropagate
Backpropagate
Backpropagate
Backpropagate
Backpropagate
Backpropagate
Multiple Players
Multiple Players

\[
\text{selectValue} = \frac{\text{node.utility}}{\text{node.visits}} + C \times \sqrt{\frac{\ln(\text{node.parent.visits})}{\text{node.visits}}}
\]
Multiple Players

\[
\text{selectValue} = -1 \times \left( \frac{\text{node.utility}}{\text{node.visits}} + C \times \sqrt{\frac{\ln(\text{node.parent.visits})}{\text{node.visits}}} \right)
\]
Multiple Players
Multiple Players

\[
\text{selectValue} = \frac{\text{node.utility}}{\text{node.visits}} + C \times \sqrt{\frac{\ln(\text{node.parent.visits})}{\text{node.visits}}}
\]
Multiple Players

0.2
0.4
0.6
0.8
1.0

u = 10
v = 1

u = 50
v = 1

u = 80
v = 1

u = 20
v = 1

u = 100
v = 2

u = 160
v = 4

u = 60
v = 2

"Multiple Players"
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search