Game Playing
function MiniMax(State s, Event e, boolean isMax)
    State s1 = updateState(s, e)
    if (isLeaf(s1))
        return eval(s1)
    if (isMax)
        highest = -\infty
        foreach (Event el in maxmoves(s1))
            tmp = MiniMax(s1, el, !isMax)
            if (tmp > highest)
                highest = tmp
                move = el
        return highest, move
    else
        lowest = \infty
        foreach (Event el in minmoves(s1))
            tmp = MiniMax(s1, el, !isMax)
            if (tmp < lowest)
                lowest = tmp
                move = el
        return lowest, move
MiniMax and Heuristics

• What about more complex games?
  – If the search space is too large
    • We could use a heuristic function to determine which nodes to expand first.
    • Unfortunately, not very useful/helpful
MiniMax and heuristics

- Practically, rather than implement MiniMax, we implement MiniCutoff
  - Don’t build the whole tree. Takes lots of memory.
  - Limit the depth of “look ahead”
  - Use a utility function rather than an evaluation function.
Cutting Off Search

• MiniMaxCutoff is identical to the MiniMax algorithm from before except:
  – if (isLeaf(s1))
  – Is replaced with
  – if (isLeaf(s1) or cutoff(S))

• Chess:
  – 4 ply lookahead is a human novice
  – 8 ply lookahead is a typical PC or human master
  – 12 ply lookahead is Deep Blue or Gary Kasparov
Pruning

- Pruning is used to narrow the search down
  - e.g. if we find a branch where we will win no matter what, we don’t need to search that part of the tree anymore
  - And if we find a guaranteed losing line we don’t need to search that part of tree any more
MiniMax

- NIM Search tree

```
Max moves
Min moves
Max moves
Min moves
```
MiniMax

- NIM Search tree

Max moves
Min moves
Max moves
Min moves

1-5
1-1-4
1-1-1-3
1-1-1-1-2

2-4
1-2-3
1-1-2-2

3-3
2-2-2

6(?)

Max moves
Min moves
MiniMax

• NIM Search tree

1-5(?) → 1-1-4 → 1-1-1-3 → 1-1-1-1-2

6(?) → 2-4(?) → 1-2-3 → 1-1-2-2

3-3(?) → 2-2-2

Max moves
Min moves
Max moves
Min moves
MiniMax

- NIM Search tree

Max moves
Min moves
Max moves
Min moves
MiniMax

• NIM Search tree

- 6(?)
  - 1-5(?)
    - 1-1-4(?)
      - 1-1-1-3(?)
        - 1-1-1-1-2
  - 2-4(?)
    - 1-2-3(?)
      - 1-1-2-2(?)
  - 3-3(?)
    - 2-2-2

Max moves
Min moves
Max moves
Min moves
MiniMax

- NIM Search tree

1-5(?)  2-4(?)  3-3(?)

1-1-4(?)  1-2-3(?)  2-2-2

1-1-1-3(?)  1-1-2-2(?)

1-1-1-1-2(?)

Max moves
Min moves
Max moves
Min moves
MiniMax

• NIM Search tree

Max moves
Min moves
Max moves
Min moves
Max moves
Min moves

1-1-1-1-2(-1)
1-1-1-2(?)
1-1-4(?)
1-5(?)

1-1-2-2(?)
1-2-3(?)
2-4(?)
3-3(?)
6(?)
MiniMax

- NIM Search tree

```
       6(?)
        /   \
   1-5(?)  2-4(?)  3-3(?)
     /     /      /     \
1-1-4(?) 1-2-3(?)  2-2-2
     /     /      /     \   \
1-1-1-3(-1) 1-1-2-2(?)
   /       /           /   \
1-1-1-1-2(-1)
```

Max moves
Min moves
Max moves
Min moves
MiniMax

- NIM Search tree

```
Max moves
Min moves
Max moves
Min moves
```
MiniMax

• NIM Search tree

Max moves
Min moves
Max moves
Min moves
MiniMax

• NIM Search tree

Max moves

Min moves

Max moves

Min moves
MiniMax

- NIM Search tree

```
1-1-1-1-2(-1)  1-1-1-3(-1)  1-1-4(1)
|       |       |       |
|       |       |       |
| 1-1-2-2(1)  1-2-3(1)  2-4(?)
|       |       |       |
|       |       |       |
| 6(?)  3-3(?)  2-2-2
```

Max moves
Min moves
Max moves
Min moves
MiniMax

- NIM Search tree

```
   6(1)
  /   \
1-5(1) 2-4(?) 3-2(?)
 /       /   /
1-1-4(1) 1-2-3(1) 2-1-2
 /     /   /
1-1-1-3(-1) 1-1-2-2(1) 2
 /     \
1-1-1-1-2(-1)
```
Alpha Beta Pruning

• A generalized pruning procedure
  – During our depth-first search we remember
    • The score of the best choice so far for Max, alpha
    • And the best score so far for Min, beta
  – If we find a point where MIN can choose a move with the utility ≤ alpha. Then we can stop search there since
    • MAX will not choose a move since there are a alternative route that is better. (Where alpha was found)
procedure Minimax_alpha_beta(n, α, β)

Inputs
n a node in a game tree
α, β real numbers

Output
A pair of a value for node n, path that gives this value

best := None

if n is a leaf node then
return evaluate(n), None
else if n is a MAX node then
for each child c of n do
    score, path := MinimaxAlphaBeta(c, α, β)
    if score ≥ β then
        return score, None
    else if score > α then
        α := score
        best := c : path
    return α, best
else
for each child c of n do
    score, path := MinimaxAlphaBeta(c, α, β)
    if score ≤ α then
        return score, None
    else if score < β then
        β := score
        best := c : path
return β, best
Alpha-Beta (α-β) Pruning Example
Alpha Beta Pruning

• What can it do for us?
  – Allows deeper searches without penalty
  – Allows deeper searches at same time cost
    • The result is always the same as without pruning, but a lot faster
Do Exact Values Matter?

MAX

MIN

Behaviour is preserved under any monotonic transformation of Eval

Only the order matters:

payoff in deterministic games acts as an ordinal utility function