6.034 Recitation 3
Search in Games: Addendum

Luis E. Ortiz

10/01/04

1 Minimax Search

The pseudocode for the Minimax algorithm follows. Given a node n in the game tree, it returns the value of n (i. e., the value that the player who has the choice to move at n can achieve assuming the player and its opponent play optimally thereafter). The algorithm is initially called with n as some starting node in the tree for which a player needs to make a choice. Recall that it usually runs only up to a certain depth of the full game tree. A terminal node of the subtree rooted at the starting node is either a leaf node of the game tree or a node at the given depth used.

```
MINIMAX(n)
if n is a terminal node then
  return value(n)
end if
if n is a maximizer node then
  v ← −∞
  for all successors s of n do
    v ← max(v, MINIMAX(s))
  end for
  return v
end if
if n is a minimizer node then
  v ← ∞
  for all successors s of n do
    v ← min(v, MINIMAX(s))
  end for
  return v
end if
```

2 α-β Search

The pseudocode for α-β follows. Given as inputs a node n and the bound $[\alpha, \beta]$, it returns the value assigned to n conditioned on α and β. That is, if n is a maximizer node and the maximum value the algorithm has found so far for its successors is greater than or equal to β, it returns the maximum value found so far; otherwise, if all of its successors values are less than β, it returns the maximum of those values. Similarly, if n is a minimizer node and the minimum value the algorithm has found so far for its successors is less than or equal to α, it returns the minimum value found so far; otherwise, if all of its successors values are greater than α, it returns the minimum of those values.

The algorithm is initially called with n as the starting node in the tree for which a player needs to make a choice, $\alpha = -\infty$ and $\beta = \infty$.

```
ALPHA-BETA(n, α, β)
if n is a terminal node then
  return value(n)
end if
if n is a maximizer node then
  v ← −∞
  for all successors s of n do
    v ← max(v, ALPHA-BETA(s, α, β))
    if v ≥ β then
      return v
    end if
    α ← max(v, α)
  end for
  return v
end if
if n is a minimizer node then
  v ← ∞
  for all successors s of n do
    v ← min(v, ALPHA-BETA(s, α, β))
    if v ≤ α then
      return v
    end if
    β ← max(v, β)
  end for
  return v
end if
```

The parameter α can be interpreted as the maximum value that the *maximizer* player can guarantee itself from any choice point (i. e., node or state) in the path to the node n. Similarly, β corresponds to the minimum value that the *minimizer* player can guarantee itself from any choice point in the path to n.

The effectiveness of α-β search is largely dependent on the order of the nodes in the game tree. If we consider nodes from left to right in the game tree, for α-β to be most effective, the successors of the *maximizer* nodes need to be ordered from *the highest to the lowest* valued, while the successors of the *minimizer* nodes need to be ordered from *the lowest to the highest* valued.

3 Progressive deepening

Progressive deepening, also known as iterative deepening, solves search problems by systematically increasing the search depth or horizon in the search or game tree. Progressive deepening is a very useful search technique in general and can be used along with any of the traditional search methods. It allows us to have an answer ready if one is needed by first solving the problem up to some small depth. Also, the longer we run the algorithm, the deeper we go into the game tree and the better the decision about which of the choices for the player is best. (Therefore, applications of progressive deepening lead to what are called *anytime algorithms.*)

For instance, we can use progressive deepening along with α-β search. The solutions from applying α-β up to a certain depth can be used to reorder the nodes in the game tree so as to hopefully make later runs of α-β that go deeper into the tree more efficient. Another heuristic that can be used during progressive deepening is to use the α-β bounds obtained in previous steps of progressive deepening to initialize the bounds for further applications of progressive deepening (instead of just using $[-\infty, \infty]$ every time we restart from the starting node of the tree).