Lecture 8 - Alpha-Beta Pruning

The number of game states that minimax search has to examine is exponential $O(b^m)$. Can we get the correct minimax decision without looking at every state in the game tree? Yes.

Alpha-beta pruning:

α is the alue of the best (i.e. highest-value) choice we have found so far at any choice point along the path for MAX.

β is the best of the best (lowest-value) choice we have found so far at any choice point along the path for MIN.

Start with $(v, -\infty, \infty)$ for (α, β) where v is a temporary value. α and β values are passed in the minimax call: minimax(s, α, β). For MAX, we prune if $v \geq \beta$. For MIN, we prune if $v \leq \alpha$. We cannot do this for the first call, only for the second onwards. This is best demonstrated through an example. Hard concept to grasp.

Properties:

- Complete? Yes, assuming finite state space.
- Optimal? Yes, assuming optimal opponent
- Time complexity? $O(b^m)$, but $O\left(b^{\frac{m}{2}}\right)$ if we have optimal ordering of nodes when pruning
- Space complexity? $O(bm)$.

Pseudo-code for implementation:

function Alpha-Beta-Search(*state*) **returns** an action
  $v \longleftarrow$ Max-Value(*state*, $-\infty, \infty$)
return the *action* in Actions(*state*) with value v.

function Max-Value(*state*, α, β) **returns** a *utility value*
  if Terminal-Test(*state*) **then return** Utility(*state*)
  $v \longleftarrow -\infty$
  for each a **in** Actions(*state*) **do**
    $v \longleftarrow$ Max(v, Min-Value(Result(s, a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \longleftarrow$ Max(α, v)
return v

function Min-Value(*state*, α, β) **returns** a *utility value*
  if Terminal-Test(*state*) **then return** Utility(*state*)
  $v \longleftarrow +\infty$
  for each a **in** Actions(*state*) **do**
    $v \longleftarrow$ Min(v, Max-Value(Result(s, a), α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \longleftarrow$ Min(β, v)
return v

Approximate real-time decisions: The minimax algorithm generates the entire game tree, and alpha beta pruning allows us to prune large parts of the search space. However, we still have to search all the way to the terminal states. We have two points here: we can replace the terminal test with cut off tests, and we can replace the utility function with some sort of heuristic / evaluation function.

Expectimax Search: Let's assume that the opponent plays randomly. Now, we have max nodes and chance nodes. We need to compute the chance values as expected utilities. That's the only difference. We just have an added chance term. Easier demonstrated with examples.