Game Playing

Preliminaries
Search in Game Playing .................................................. 2
Useful Definitions ............................................................. 3

Minimax
MAX Procedure .............................................................. 4
MIN Procedure .............................................................. 5
Example ........................................................................... 6

Alpha-Beta Pruning
Idea of Alpha-Beta Pruning ................................................ 7
MAX Procedure .............................................................. 8
MIN Procedure .............................................................. 9
Example ........................................................................... 10

Issues
Performance of Minimax and Alpha-Beta .............................. 11
Other Issues ..................................................................... 12

Search in Game Playing
□ In game playing, the choice of action must take the opponent into account.
□ A search problem for a game is defined by:
  – Initial state. Current position and whose turn.
  – Operators. The legal moves.
  – Terminal test. Is the game over in a given state?
  – Utility function. Is a terminal state win, lose, or draw?

Useful Definitions
□ MAX. The player whose turn it is to move.
□ MIN. The other player.
□ Ply. A synonym for “depth.”

Minimax
MAX Procedure
MAX should maximize evaluation function assuming that MIN minimizes it.

function MAX-VALUE(state, bound)
if TERMINAL(state) or bound = 0
  then return EVALUATION(state)
max ← −∞
for each next in EXPAND(state)
do eval ← MIN-VALUE(next, bound − 1)
  if eval > max then max ← eval
return max
**MIN Procedure**

Assume MIN minimizes MAX's evaluation function.

```
function Min-Value(state, bound)
    if Terminal(state) or bound = 0
        then return Evaluation(state)
    min ← ∞
    for each next in Expand(state)
        do eval ← Max-Value(next, bound - 1)
        if eval < min then min ← eval
    return min
```

**Example**

Minimax
evaluation = +10 if I have 3 in a row
−10 if my opponent has 3 in a row
+1 for each potential 3 in a row for me
−1 for each potential 3 in a row for my opponent

MAX is O; MIN is X

```
\[
\begin{array}{c}
\text{MAX} \\
\text{MIN}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{MIN} \\
\text{MAX}
\end{array}
\]
```

**Alpha-Beta Pruning**

**Idea of Alpha-Beta Pruning**

- The idea of alpha-beta pruning is to avoid search that won’t change the minimax evaluation.
- Example: If MAX has a move which evaluates to 3, then stop searching other moves when their values are known to be ≤ 3.
- General Example:
  Consider a minimizing node n.
  Let α be the maximum-so-far in an ancestor.
  Let β be the minimum-so-far of n.
  If α ≥ β, value of n cannot change α.

**MAX Procedure**

α is a known max-so-far value in an ancestor.
β is a known min-so-far value in an ancestor.

```
function Max-Value(state, bound, α, β)
    if Terminal(state) or bound = 0
        then return Evaluation(state)
    for each next in Expand(state)
        do eval ← Min-Value(next, bound - 1, α, β)
        if eval > α then α ← eval
        if α ≥ β then return α
    return α
```

**MIN Procedure**

\( \alpha \) is a known max-so-far value in an ancestor.
\( \beta \) is a known min-so-far value in an ancestor.

**function MIN-VALUE(state, bound, \( \alpha \), \( \beta \))**

- if \( \text{Terminal}(state) \) or \( \text{bound} = 0 \)
  - then return \( \text{Evaluation}(state) \)
- for each next in \( \text{Expand}(state) \)
  - do \( \text{eval} \leftarrow \text{MAX-VALUE}(next, \text{bound} - 1, \alpha, \beta) \)
  - if \( \text{eval} < \beta \) then \( \beta \leftarrow \text{eval} \)
  - if \( \beta \leq \alpha \) then return \( \beta \)
- return \( \beta \)

---

**Issues**

**Performance of Minimax and Alpha-Beta**

- \( b \) = branching factor
- \( d \) = depth of search
- Minimax visits every state from level 0 to \( d \).
  
  \[
  \sum_{i=0}^{d} b^i = \frac{b^{d+1} - 1}{b - 1} \in O(b^d)
  \]
  
  - Alpha-Beta visits as few as \( \Omega(b^{d/2}) \) states.
    
    Depends on a good ordering from \( \text{Expand} \).
    
    Actual programs approach the minimum bound.
  
  - Alpha-beta pruning allows programs to look ahead nearly twice as many moves as minimax.

**Other Issues**

- Horizon problem
- Quiescence
- Data bases of openings and end games
- Games of chance