GAME THEORY — MINIMAX SEARCH AND ALPHA-BETA PRUNING

Scored games

In the following, we will forget about Conway's theory and drop the normal play convention. Instead, we will consider what we might call *scored games*, namely games that end with a score rather than a winner and a loser. There are two players called Max and Min, and Max wants to maximize the score while Min wants to minimize it.

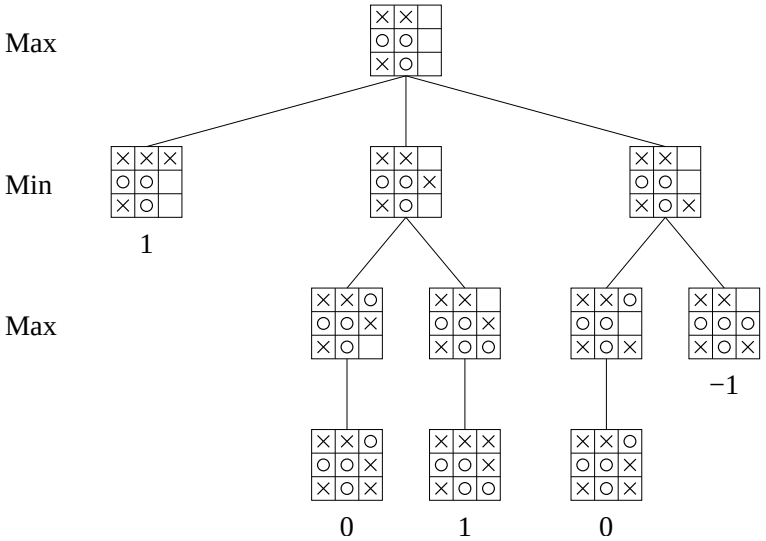
Formally, a scored game is a rooted tree whose leaves are labelled by real numbers and whose inner vertices (non-leaves) are labelled either "Max" or "Min". The game is played with a pebble as follows.

- Initially, the pebble is put on the root vertex.
- The label of the vertex with the pebble tells us who's turn it is.
- A legal move is to move the pebble to any child vertex.
- The game ends when the pebble is moved to a leaf, and the score is the label of that leaf.

The vertices are called *positions* of the game, and the leaves are called *terminal positions*.

Most popular board games (with no element of chance) can be interpreted as a scored game in our sense. Most often, the players will alternate moves, so that non-terminal positions on the same level of the tree (that is, at the same distance from the root) are equally labelled.

Let us take tic-tac-toe as an example (that is, "three in a row" played on a 3 times 3 board). That game ends with a win for crosses, a win for noughts, or a draw when neither player has three in a row and the board is filled. Let us interpret the outcome as a score by the following rule: win for crosses = 1, win for noughts = −1, draw = 0. Max plays crosses and Min plays noughts. The complete game tree is huge, but here is a subtree:



(Since, in this case, every non-terminal position on the same level has the same label, we have labelled the levels instead of the positions.)

Minimax search

Instead of playing a game from the beginning, we can put the pebble at some position P and play from there. Let us denote the resulting score by $S(P)$, *the score of* P, assuming optimal play as always.

Let P' denote a typical child position of P. Obviously, for any non-terminal position P, we have $S(P) = \max_{P'} S(P')$ if P is a Max-position, and $S(P) = \min_{P'} S(P')$ if P is a Min-position, so we can compute the score of any position recursively by the following pseudo-code.

```
function S(P)
    if P is terminal
      return its score
    if P is a Max-position
      α ← −∞
      for each child position P'
          α ← max{α, S(P')}
    return α
    if P is a Min-position
      β ← +∞
      for each child position P'
          β ← min{β, S(P')}
    return β
```

This algorithm is called a *(complete) minimax search* of the game tree.

Alpha-beta pruning

The minimax search consists of many situations of the following kind:

"Hm, I'm Max and I've already found a move from P that yields a score as high as α. But maybe there's a better move. Let's see, this one I haven't tried yet. Excuse me, Min, but what is the lowest score you can achieve if I make this move?"

If the answer from Min is lower than α, Max does not really care about its exact value since he has already found a better move. Thus, Max could spare Min some work by formulating his question like this instead: "Excuse me, Min, but what is the lowest score you can achieve if I make this move? However, if it is $\leq \alpha$, you may answer with any number $\leq \alpha$."

More general: If a query about $S(P)$ is to be made in order to compute $S(Q)$ and we already know that $\alpha \leq S(Q) \leq \beta$, then we might add to the query that if $S(P) \notin (\alpha, \beta)$ we do not really care about its exact value. Thus, we want to implement a function $S(P, \alpha, \beta)$ such that

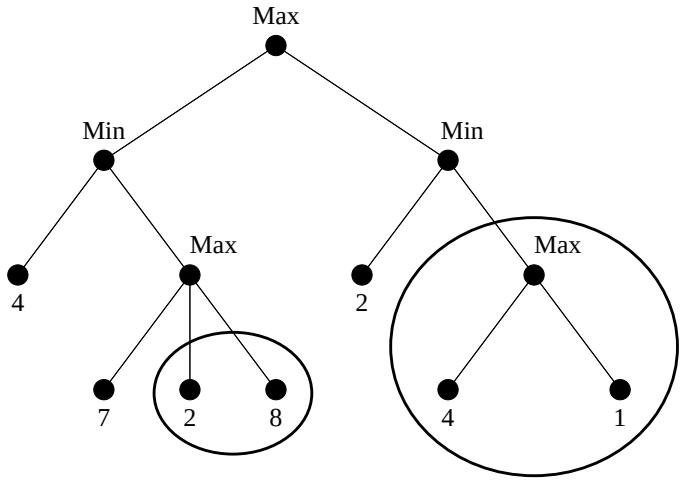
$$\begin{aligned} S(P) \leq \alpha &\Rightarrow S(P, \alpha, \beta) \leq \alpha, \\ \alpha < S(P) < \beta &\Rightarrow S(P, \alpha, \beta) = S(P), \\ \beta \leq S(P) &\Rightarrow \beta \leq S(P, \alpha, \beta). \end{aligned}$$

Here is an implementation in pseudo-code:

```
function S(P, α, β)
    if P is terminal
      return its score
    if P is a Max-position
      for each child position P'
          α ← max{α, S(P', α, β)}
          if α ≥ β
            break
      return α
    if P is a Min-position
      for each child position P'
          β ← min{β, S(P', α, β)}
          if α ≥ β
            break
      return β
```

This algorithm (with initial function call $S(\text{root}, -\infty, +\infty)$) is called *minimax search with alpha-beta pruning* since it typically does not need to search the whole game tree but prunes it. (Prune = beskära in Swedish.) Note that it is important in which order the child positions are explored by the algorithm. One usually draws the game tree so that the possible moves are explored from left to right.

For example, in the following game tree the circled vertices are pruned and will not be explored by the algorithm if moves are tried from left to right.



Heuristics

In real life, most interesting games are far too complex for the minimax search to terminate in a life time, whether we apply alpha-beta pruning or not. The chess computer Deep Blue, for instance, typically manages to look 12 moves (or rather half-moves) ahead, then it evaluates the position heuristically (by counting the pawns and so on). In other words, for real applications we must rely on good guesses at some depth d of the minimax search, and we can approximate the score of a position by the following algorithm, which also is called a *minimax search*.

```
function S(P, d)
    if P is terminal or d = 0
      return the score of P or a good guess
    if P is a Max-position
      α ← −∞
      for each child position P′
          α ← max{α, S(P′, d − 1)}
    return α
    if P is a Min-position
      β ← +∞
      for each child position P′
          β ← min{β, S(P′, d − 1)}
    return β
```

With alpha-beta pruning it looks like this:

```
function S(P, α, β, d)
    if P is terminal or d = 0
      return the score of P or a good guess
    if P is a Max-position
      for each child position P′
          α ← max{α, S(P′, α, β, d − 1)}
          if α ≥ β
            break
      return α
    if P is a Min-position
      for each child position P′
          β ← min{β, S(P′, α, β, d − 1)}
          if α ≥ β
            break
      return β
```

We can improve the algorithm significantly by using the heuristic evaluation function to sort the child positions before they are explored. If good moves are explored before bad moves, larger subtrees may be pruned, so the better the heuristic, the faster the algorithm terminates. As a rule of thumb, alpha-beta pruning with a good heuristic evalutaion function allows a computer to double the search depth compared to a complete minimax search.