Topics contained herein:

Strategies for 2 player games

<u>Strategies:</u>

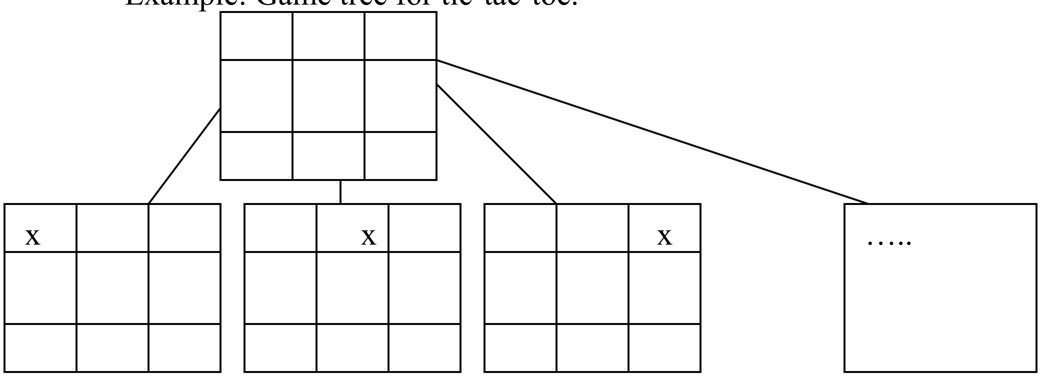
Max – Player who moves first

Wants to come up with a strategy for what to do contingent upon Min playing his best.

An optimal strategy is a sequence of contingent decisions that will lead to outcomes at least as good as any other strategy when one is playing an infallible opponent.

It is useful to use a game tree when trying to reason about strategies.

Example: Game tree for tic-tac-toe.



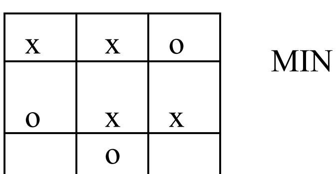
<u>Minimax value of a node</u>

Useful for determining optimal strategy

Minimax – value(n)

$$= \begin{cases} \text{Utility}(n) \text{ if } n \text{ is a terminal} \\ \text{Max}_{s\, \varepsilon\, succ(n)} \text{ MiniMax-value}(S) \text{ if } n \text{ is a max node} \\ \text{Min}_{s\, \varepsilon\, succ(n)} \text{ MiniMax-value}(S) \text{ if } n \text{ is a min node} \end{cases}$$

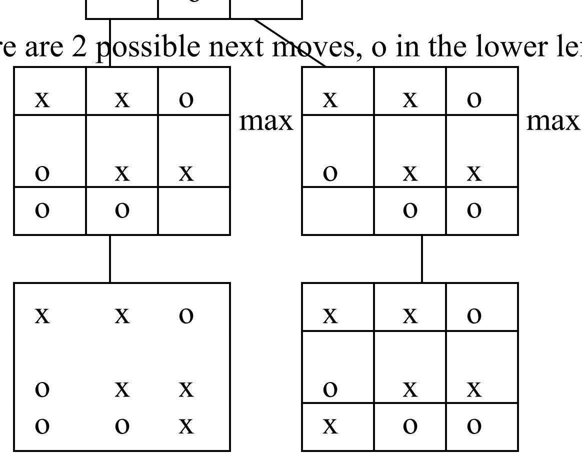
Example



Terminal Values

+1	Max wins
-1	Min wins
0	Draw

There are 2 possible next moves, o in the lower left, or lower right corner.



value of terminal board is 0 (right board)

value of terminal board is 1 (left board)

Our goal is to have a board of value –1 to win the game, or at worst 0 to draw the game. An outcome of +1 means you lose the game. (if the system is min and player is max)

<u>Minimax Algorithm</u>

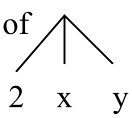
Given a current state, if player is MAX, choose a move so successor node of largest minimax value.
If player is min, choose a move so successor node is of least value.

If the maximum depth of the game tree is m, and expected branching factor is b, then time complexity of minimax is **$O(b^m)$**

It is possible depending on implementation to have a linear space complexity, ergo space complexity is not an issue.

$O(b^m)$ for time complexity is impractical. We can do better on average, and get **$O(b^{m/2})$**. Consider two level tree

```
        MAX
   3/     |    \
 /  |\    /|\  /|\ MIN
3  4 7  2
```

For the next subtree,
since backed up value of  <= 2, and in doing the traversal of game tree,

Max has already seen a backed-up value 3, so Max doesn't need to expand x & y, therefore the backed-up value 3 is called the alpha value, and ignoring x & y is called an alpha pruning, or alpha cut of tree.

The analogous thing for Min is a beta value, beta pruning or beta cut of tree.

For min, the beta value is the largest value as opposed to alpha's smallest value.

On average, beta/alpha pruning makes the minimax algorithm time complex **$O(b^{m/2})$**