- The results of alpha-beta depend on the order in which moves are considered among the children of a node.
- If possible, consider better moves first!

Real-world use of alpha-beta

- (Regular) minimax is normally run as a preprocessing step to find the optimal move from every possible situation.
- Minimax with alpha-beta can be run as a preprocessing step, but might have to re-run during play if a non-optimal move is chosen.
- Save states somewhere so if we re-encounter them, we don't have to recalculate everything.

Real-world use of alpha-beta

- States get repeated in the game tree because of *transpositions*.
- When you discover a best move in minimax or alpha-beta, save it in a lookup table (probably a hash table).
  - Called a *transposition table*.

Real-world use of alpha-beta

- In the real-world, alpha-beta does not "pre-generate" the game tree.
  - The whole point of alpha-beta is to not have to generate all the nodes.
- The DFS part of minimax/alpha-beta is what generates the tree.

Improving on alpha-beta

- Alpha-beta still has to search down to terminal nodes sometimes.
  - (and minimax has to search to terminal nodes all the time!)
- Improvement idea: can we get away with only looking a few moves ahead?

Heuristic minimax algorithm

h-minimax(s, d) =

$$\text{h-minimax}(s,d) = \begin{cases} \text{heuristic-eval}(s) & \text{if cutoff}(s, d) \\ \max_{a \text{ in actions}(s)} \text{h-minimax}(\text{result}(s, a), d+1) & \text{if player}(s)=\text{MAX} \\ \min_{a \text{ in actions}(s)} \text{h-minimax}(\text{result}(s, a), d+1) & \text{if player}(s)=\text{MIN} \end{cases}$$

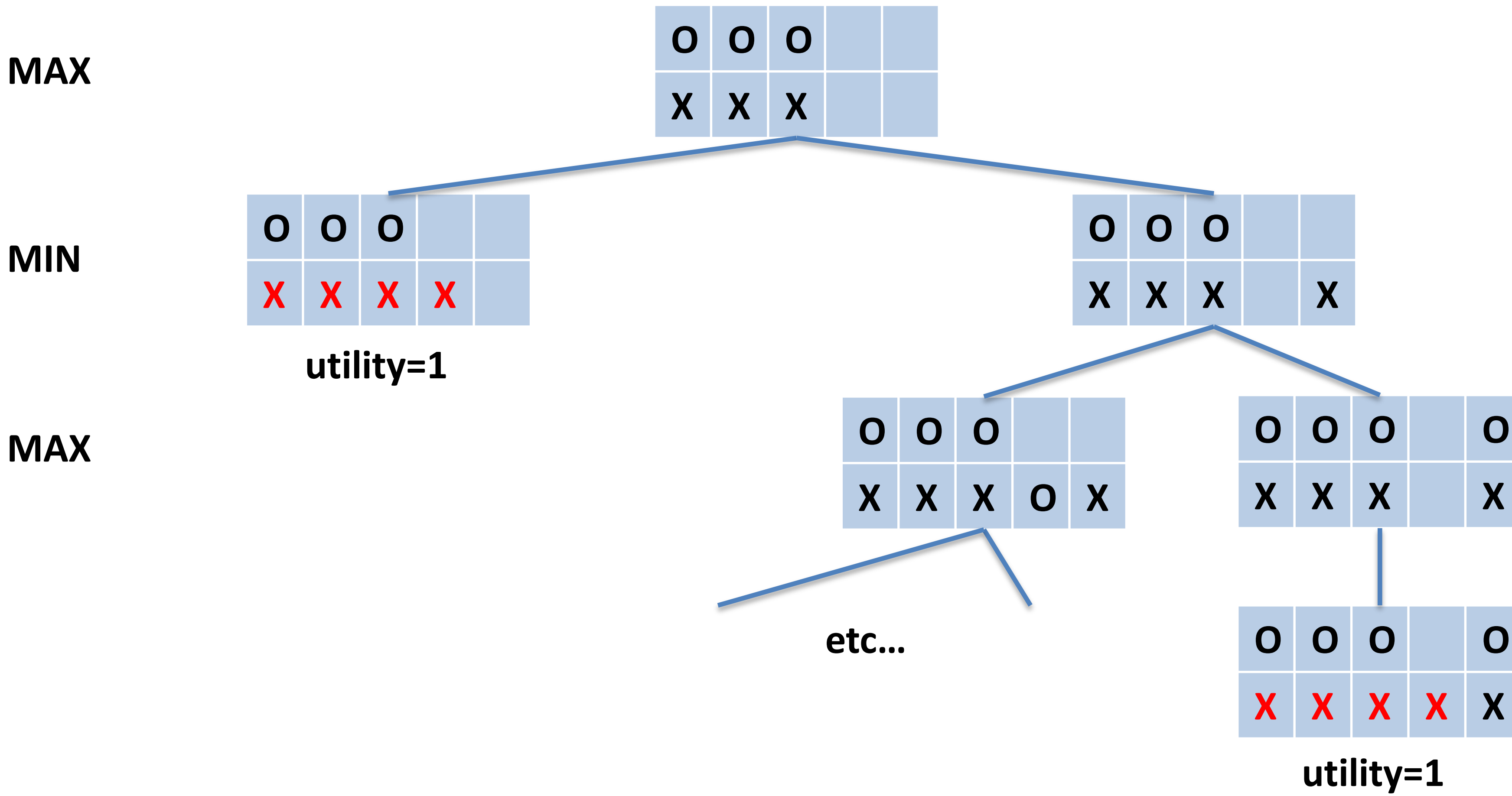
result(s, a) means the new state generated by taking action *a* in state *s*.

cutoff(s, d) is a boolean test that determines whether we should stop the search and evaluate our position.

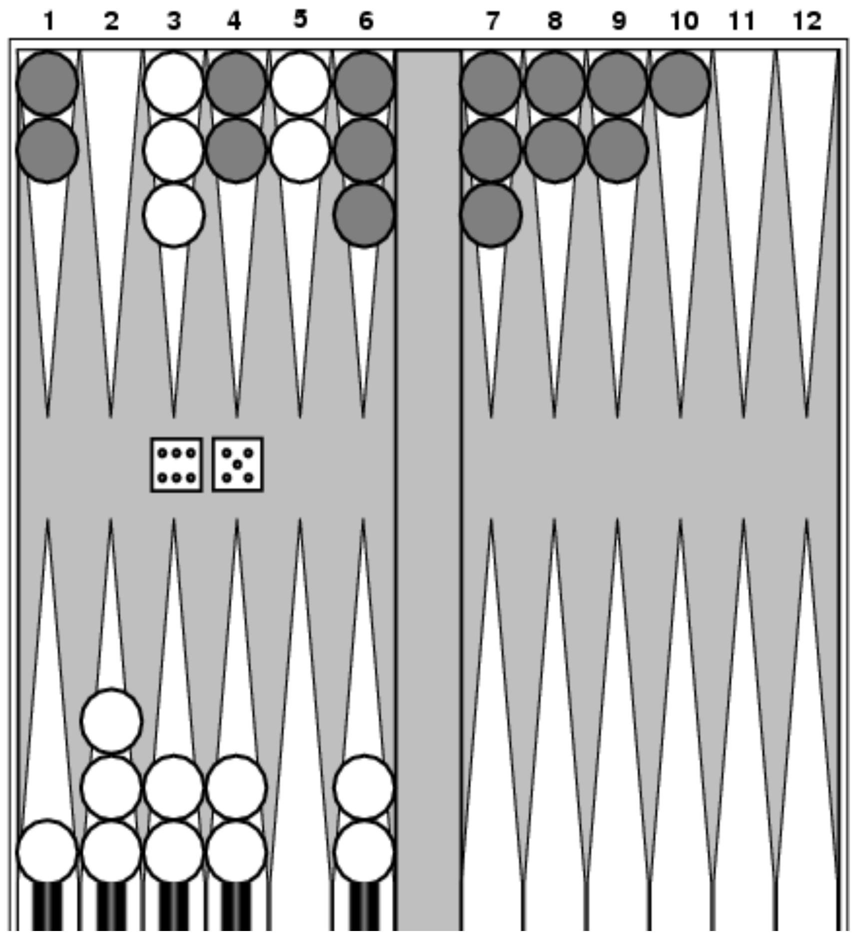
How to create a good evaluation function?

- Trying to judge the probability of winning from a given state.
- Typically use features: simple characteristics of the game that correlate well with the probability of winning.

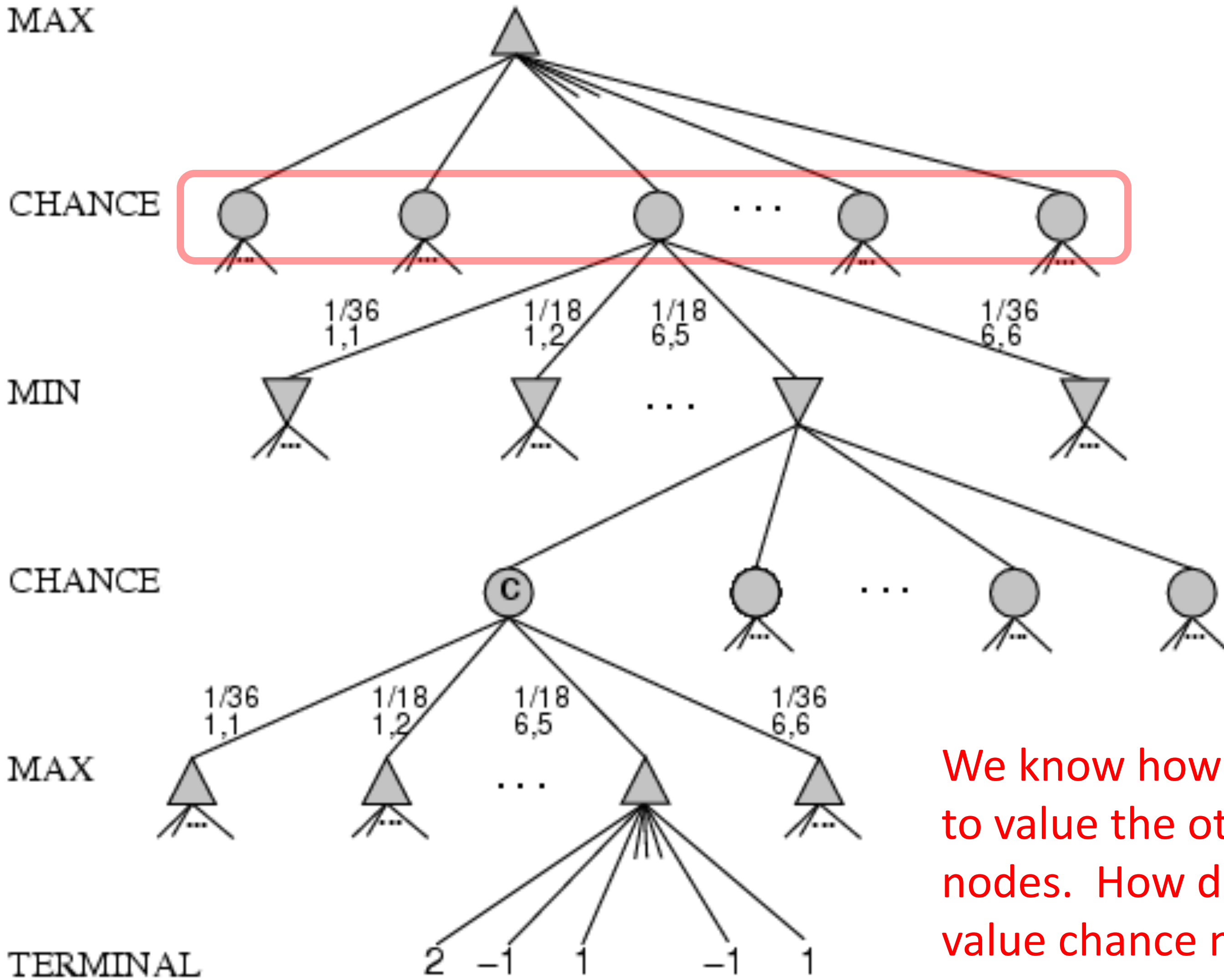
One last point



What if a game has a "chance element"?



What if a game has a "chance element"?



We know how to value the other nodes. How do we value chance nodes?

Expected value

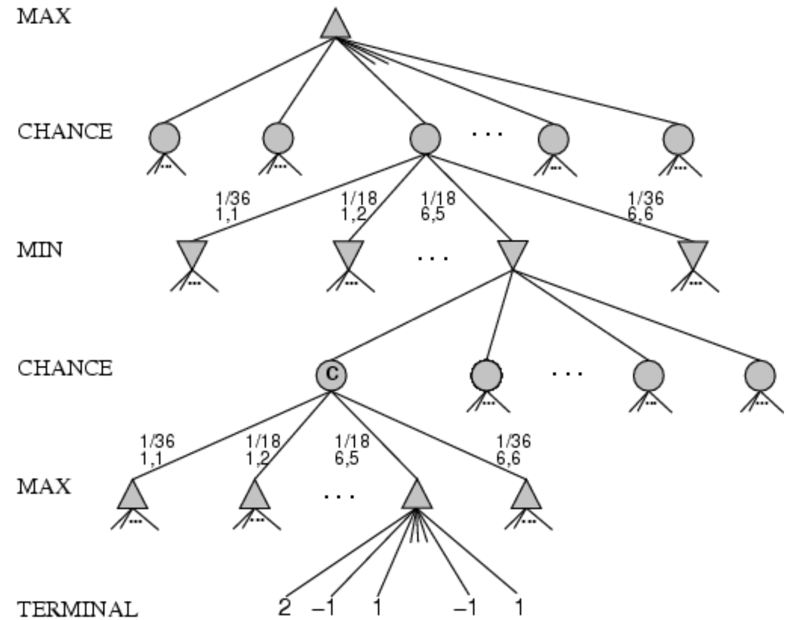
- The sum of the probability of each possible outcome multiplied by its value:

$$E(X) = \sum_i p_i x_i$$

- x_i is a possible value of (random variable) X.
- p_i is the probability of xi happening.

Expected minimax value

- Now *three* different cases to evaluate, rather than just two.
  - MAX
  - MIN
  - CHANCE



EXPECTED-MINIMAX-VALUE(n) =

UTILITY(n), If terminal node

$\max_{s \in successors(n)}$ MINIMAX-VALUE(s), If MAX node

$\min_{s \in successors(n)}$ MINIMAX-VALUE(s), If MIN node

$\sum_{s \in successors(n)} P(s)$ • EXPECTEDMINIMAX(s), If CHANCE node