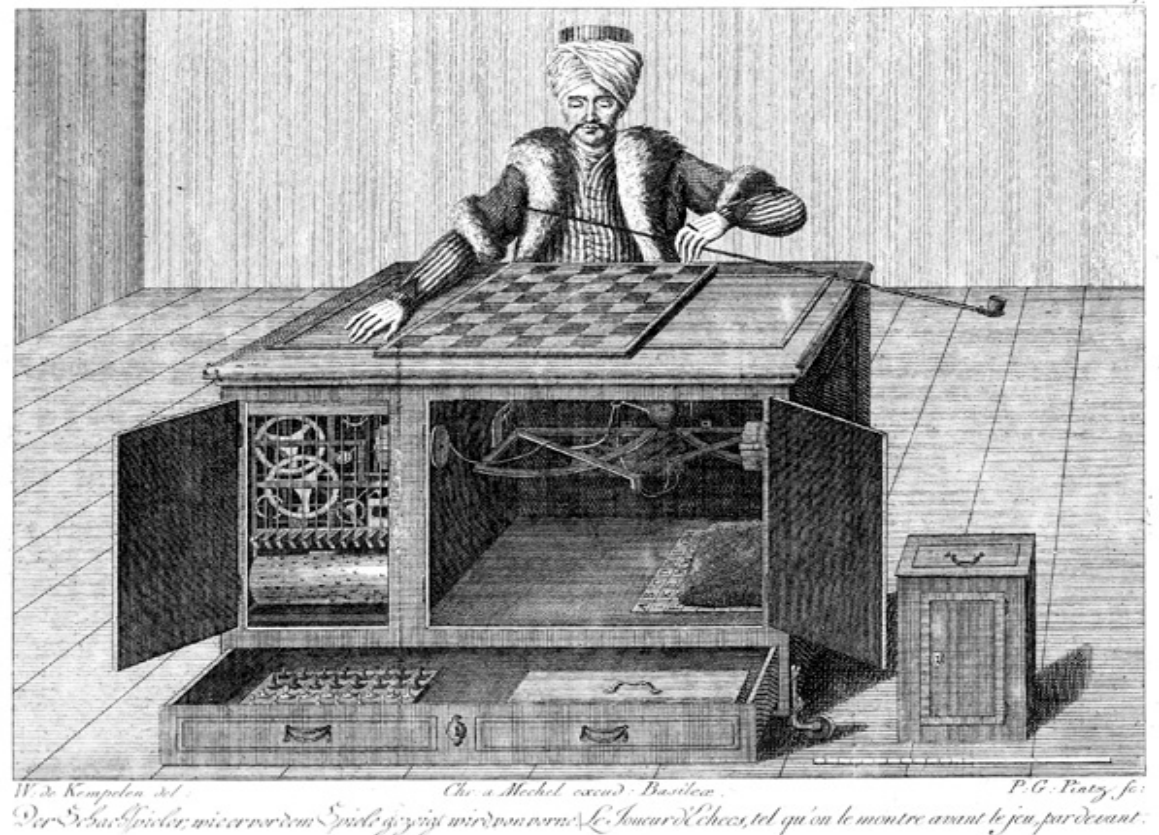


CS440/ECE448 Lecture 8: Two-Player Games

Slides by Svetlana Lazebnik 9/2016

Modified by Mark Hasegawa-Johnson 2/2018



Why study games?

- Games are a traditional hallmark of intelligence
- Games are easy to formalize
- Games can be a good model of real-world competitive or cooperative activities
 - Military confrontations, negotiation, auctions, etc.

Game AI: Origins

- Minimax algorithm: Ernst Zermelo, 1912
- Chess playing with evaluation function, quiescence search, selective search:
Claude Shannon, 1949 ([paper](#))
- Alpha-beta search: John McCarthy, 1956
- Checkers program that learns its own evaluation function by playing against itself: Arthur Samuel, 1956

Types of game environments

	Deterministic	Stochastic
Perfect information (fully observable)	Chess, checkers, go	Backgammon, monopoly
Imperfect information (partially observable)	Battleship	Scrabble, poker, bridge

Zero-sum Games

Alternating two-player zero-sum games

- Players take turns
- Each game outcome or **terminal state** has a **utility** for each player (e.g., 1 for win, 0 for loss)
- The sum of both players' utilities is a constant

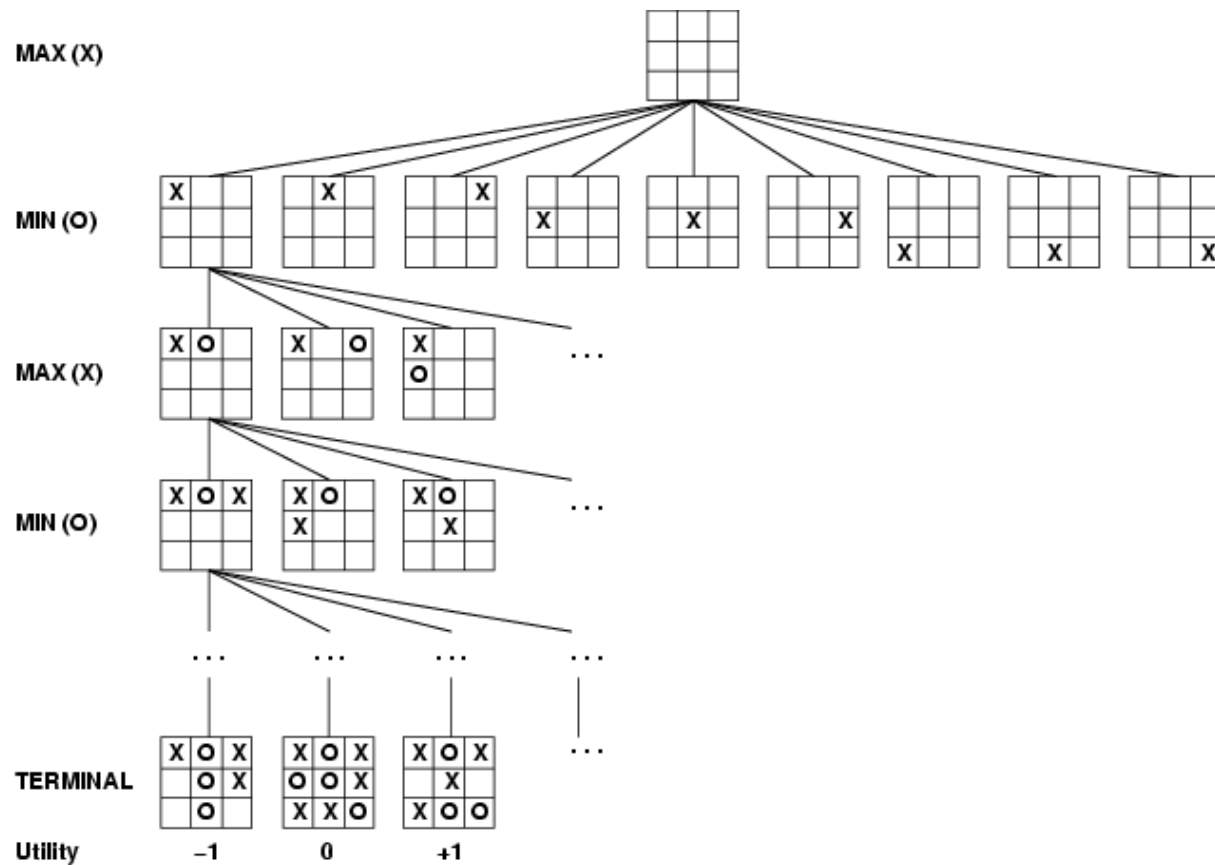


Games vs. single-agent search

- We don't know how the opponent will act
 - The solution is not a fixed sequence of actions from start state to goal state, but a *strategy* or *policy* (a mapping from state to best move in that state)

Game tree

- A game of tic-tac-toe between two players, “max” and “min”

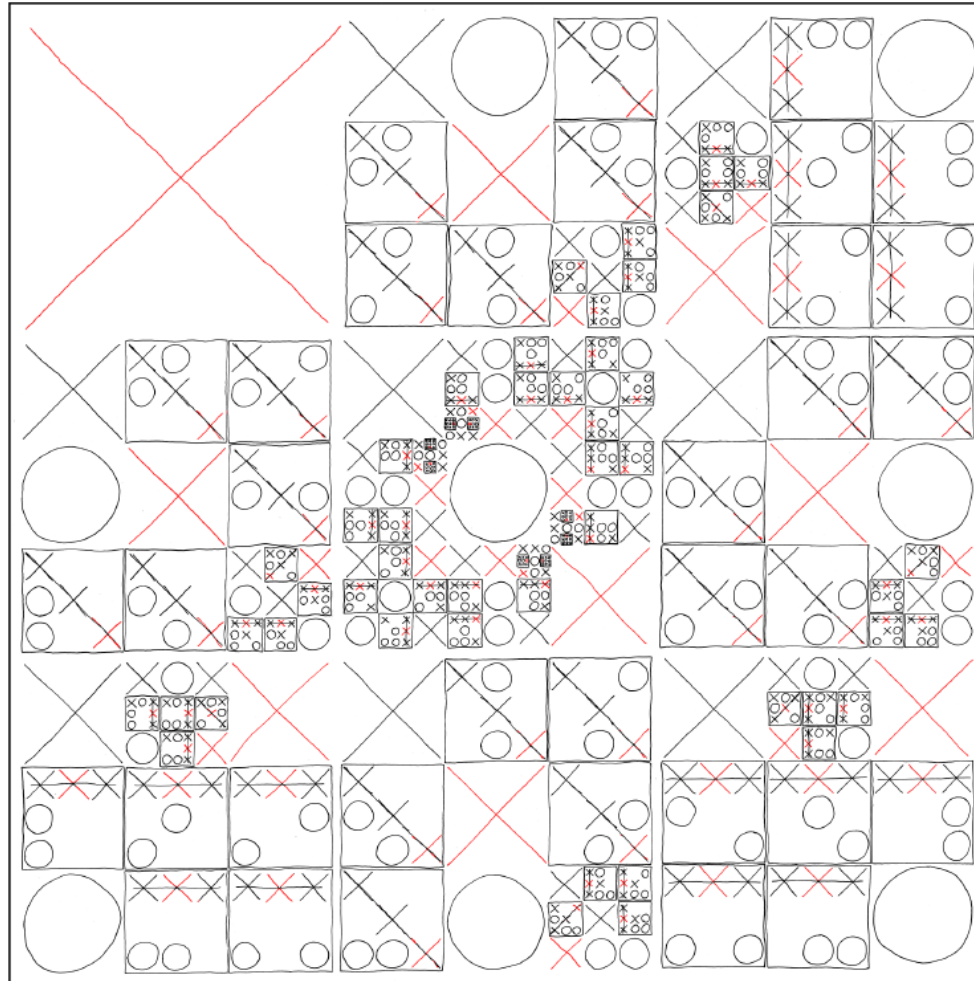


COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

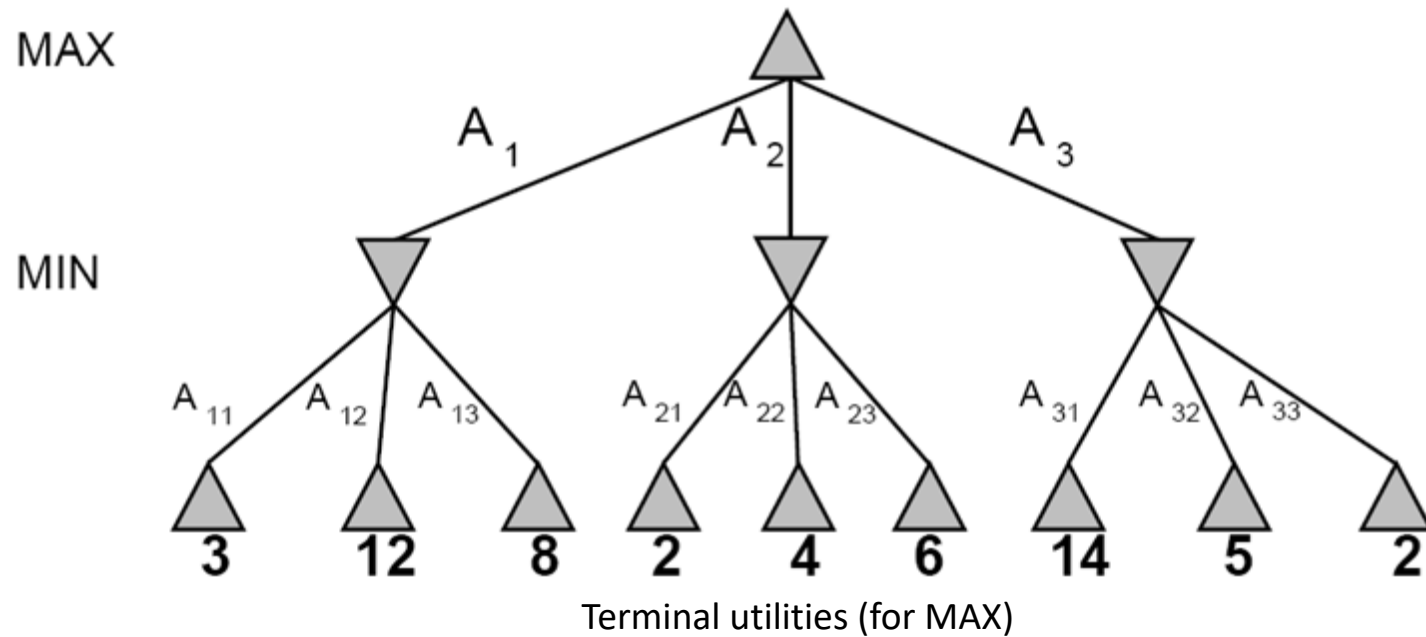
YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

<http://xkcd.com/832/>

MAP FOR X:



A more abstract game tree



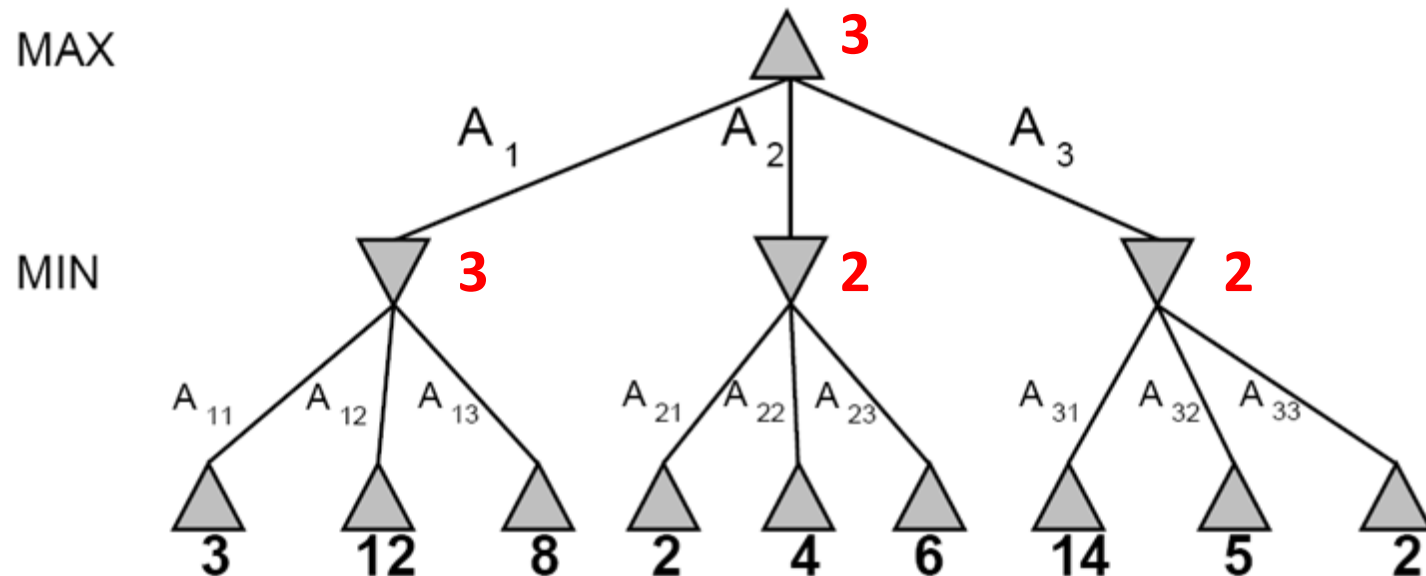
A two-ply game

Minimax Search

The rules of every game

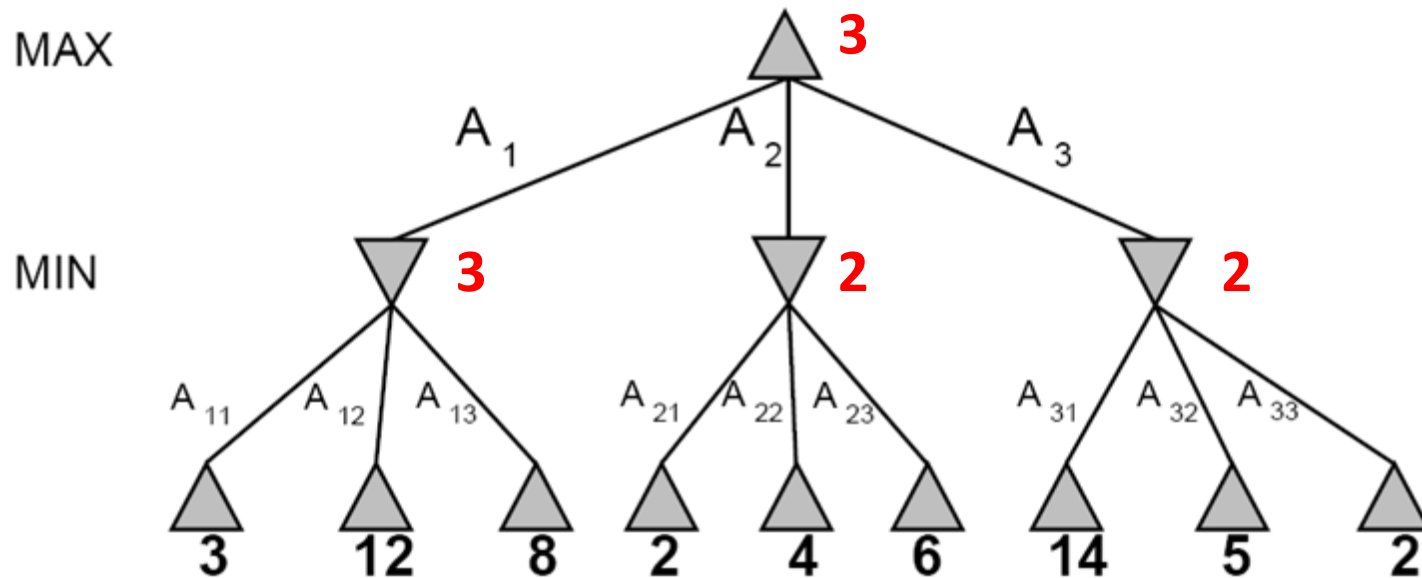
- Every possible outcome has a value (or “utility”) for me.
- Zero-sum game: if the value to me is $+V$, then the value to my opponent is $-V$.
- Phrased another way:
 - My rational action, on each move, is to choose a move that will maximize the value of the outcome
 - My opponent’s rational action is to choose a move that will minimize the value of the outcome
- Call me “Max”
- Call my opponent “Min”

Game tree search



- **Minimax value of a node:** the utility (for MAX) of being in the corresponding state, assuming perfect play on both sides
- **Minimax strategy:** Choose the move that gives the best worst-case payoff

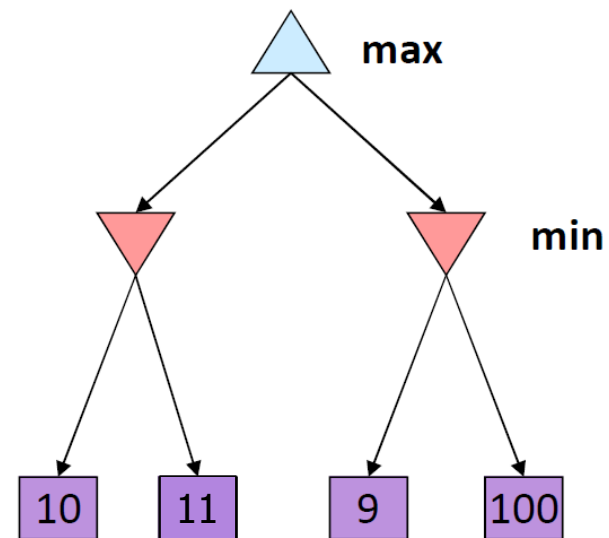
Computing the minimax value of a node



- **Minimax**(*node*) =
 - $\text{Utility}(\text{node})$ if *node* is terminal
 - $\max_{\text{action}} \text{Minimax}(\text{Succ}(\text{node}, \text{action}))$ if *player* = MAX
 - $\min_{\text{action}} \text{Minimax}(\text{Succ}(\text{node}, \text{action}))$ if *player* = MIN

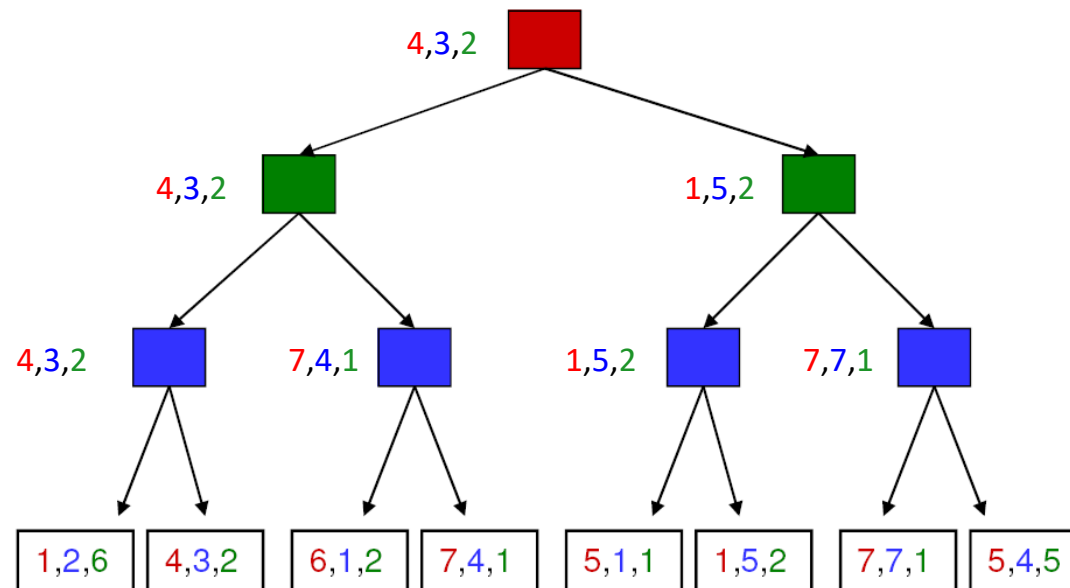
Optimality of minimax

- The minimax strategy is optimal against an optimal opponent
- What if your opponent is suboptimal?
 - Your utility will ALWAYS BE HIGHER than if you were playing an optimal opponent!
 - A different strategy may work better for a sub-optimal opponent, but it will necessarily be worse against an optimal opponent



Example from D. Klein and P. Abbeel

More general games

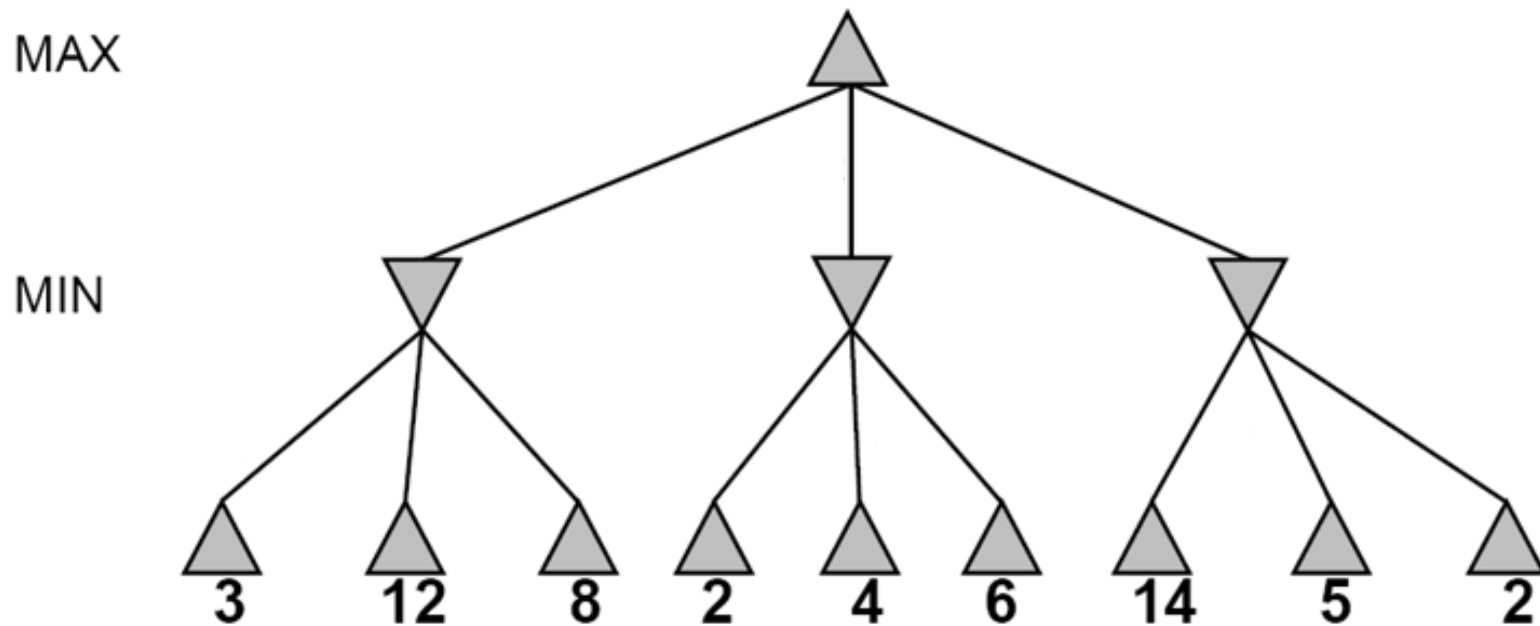


- More than two players, non-zero-sum
- Utilities are now tuples
- Each player maximizes their own utility at their node
- Utilities get propagated (*backed up*) from children to parents

Alpha-Beta Pruning

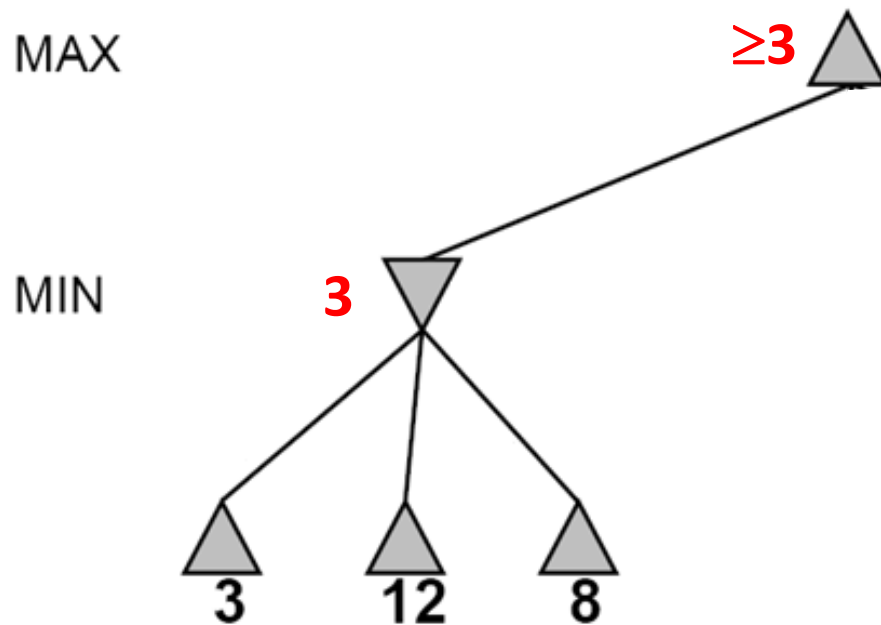
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



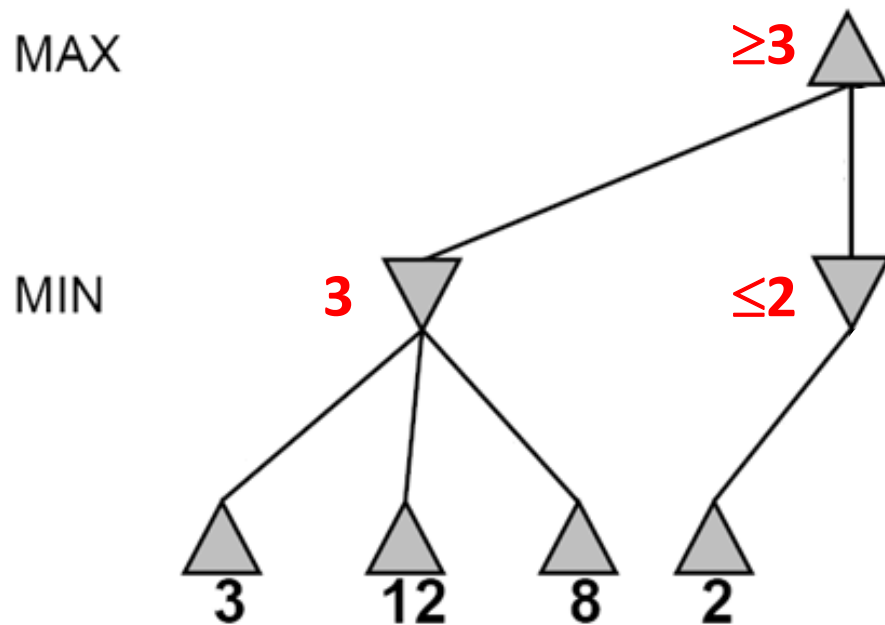
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



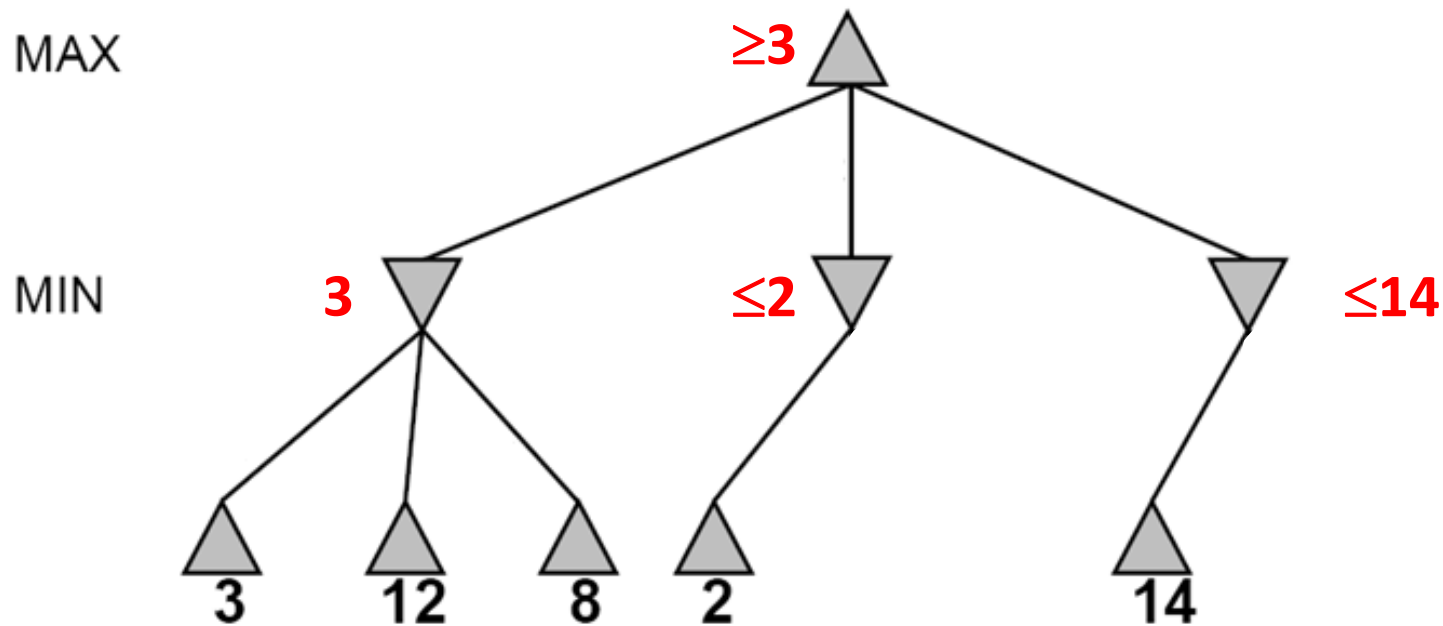
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



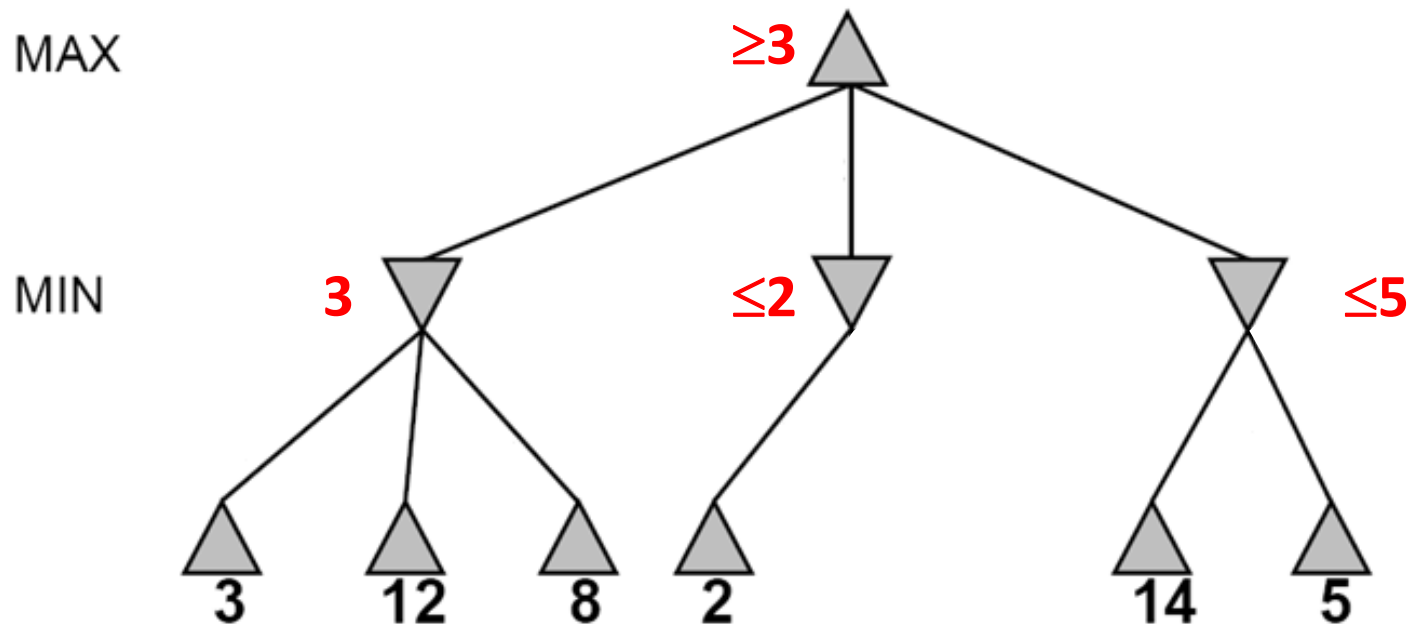
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



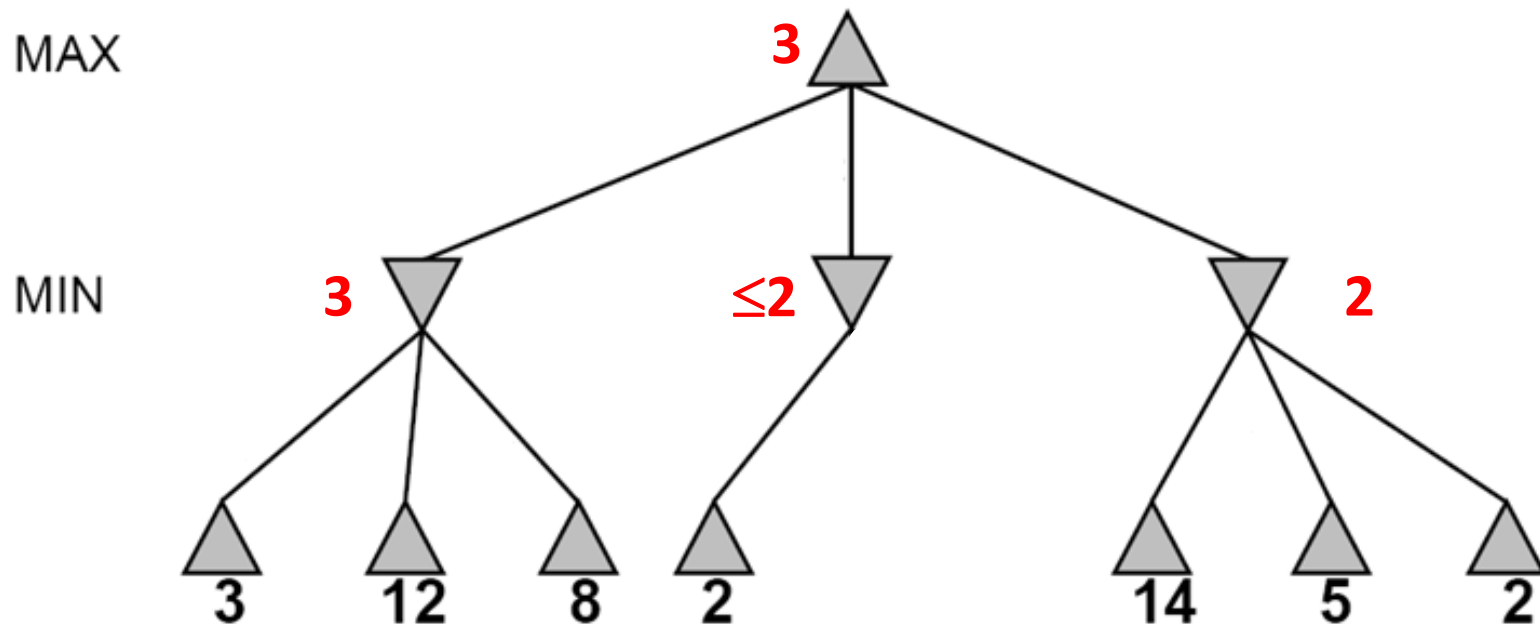
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



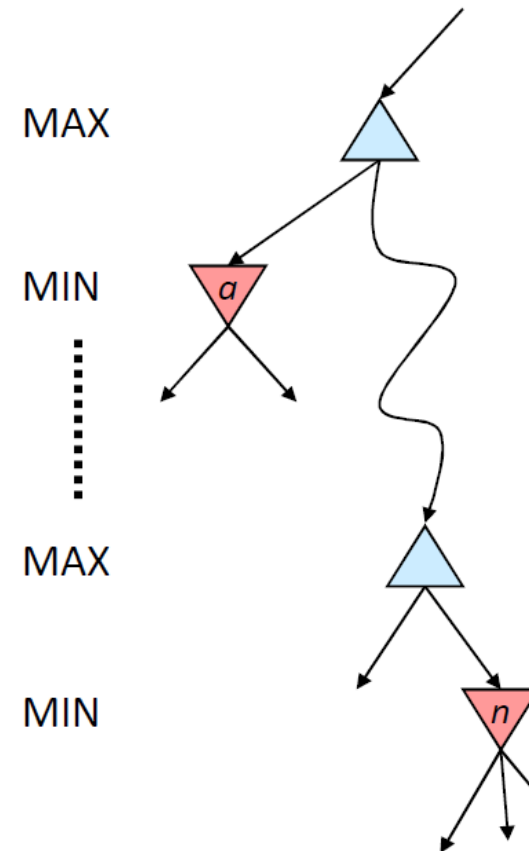
Alpha-Beta Pruning

Key point that I find most counter-intuitive:

- MIN needs to calculate which move MAX will make.
- MAX would never choose a suboptimal move.
- So if MIN discovers that, at a particular node in the tree, she can make a move that's REALLY REALLY GOOD for her...
- She can assume that MAX will never let her reach that node.
- ... and she can prune it away from the search, and never consider it again.

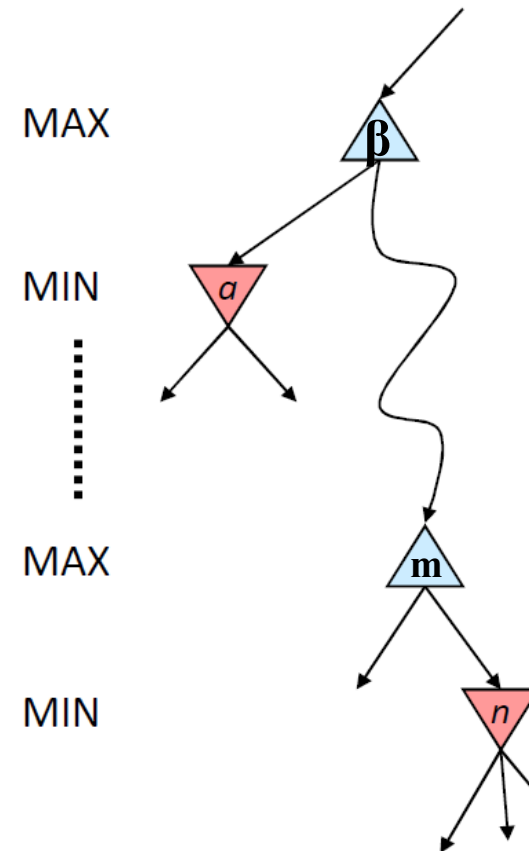
Alpha-beta pruning

- α is the value of the best choice for the MAX player found so far at any choice point above node n
- More precisely: α is the highest number that MAX knows how to force MIN to accept
- We want to compute the MIN-value at n
- As we loop over n 's children, the MIN-value decreases
- If it drops below α , MAX will never choose n , so we can ignore n 's remaining children



Alpha-beta pruning

- β is the value of the best choice for the MIN player found so far at any choice point above node n
- More precisely: β is the lowest number that MIN know how to force MAX to accept
- We want to compute the **MAX**-value at m
- As we loop over m 's children, the **MAX**-value increases
- If it rises above β , MIN will never choose m , so we can ignore m 's remaining children



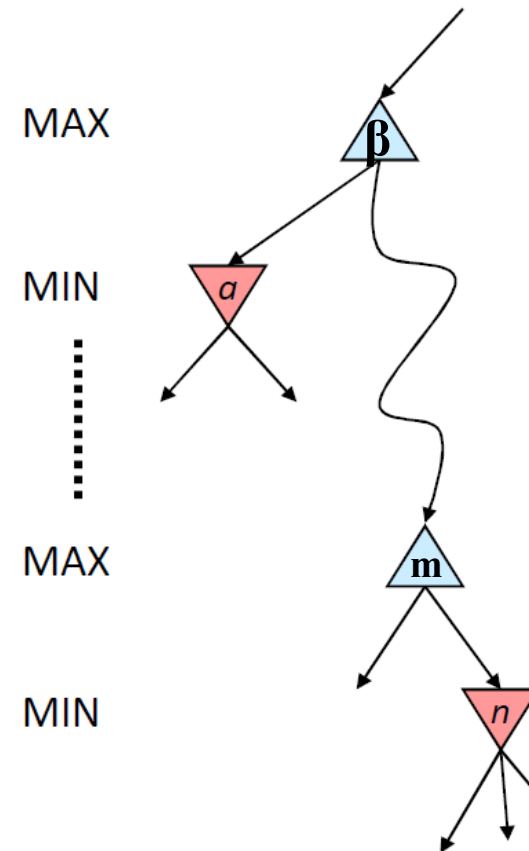
Alpha-beta pruning

An unexpected result:

- α is the highest number that MAX knows how to force MIN to accept
- β is the lowest number that MIN know how to force MAX to accept

So

$$\alpha \leq \beta$$



Alpha-beta pruning

Function $action = \text{Alpha-Beta-Search}(node)$

$v = \text{Min-Value}(node, -\infty, \infty)$

return the $action$ from $node$ with value v

α : best alternative available to the Max player

β : best alternative available to the Min player

Function $v = \text{Min-Value}(node, \alpha, \beta)$

if $\text{Terminal}(node)$ return $\text{Utility}(node)$

$v = +\infty$

for each $action$ from $node$

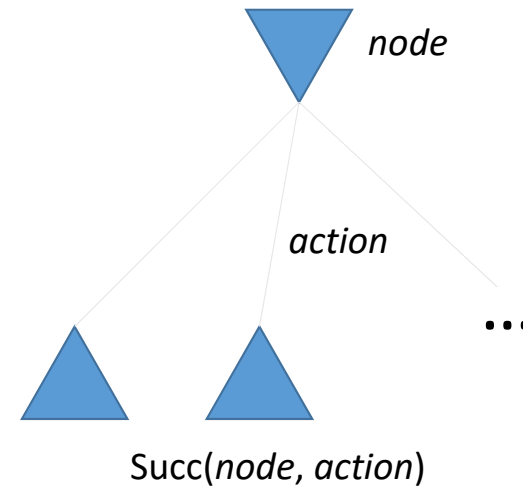
$v = \text{Min}(v, \text{Max-Value}(\text{Succ}(node, action), \alpha, \beta))$

if $v \leq \alpha$ return v

$\beta = \text{Min}(\beta, v)$

end for

return v



Alpha-beta pruning

Function $action = \text{Alpha-Beta-Search}(node)$

$v = \text{Max-Value}(node, -\infty, \infty)$

return the $action$ from $node$ with value v

α : best alternative available to the Max player

β : best alternative available to the Min player

Function $v = \text{Max-Value}(node, \alpha, \beta)$

if $\text{Terminal}(node)$ return $\text{Utility}(node)$

$v = -\infty$

for each $action$ from $node$

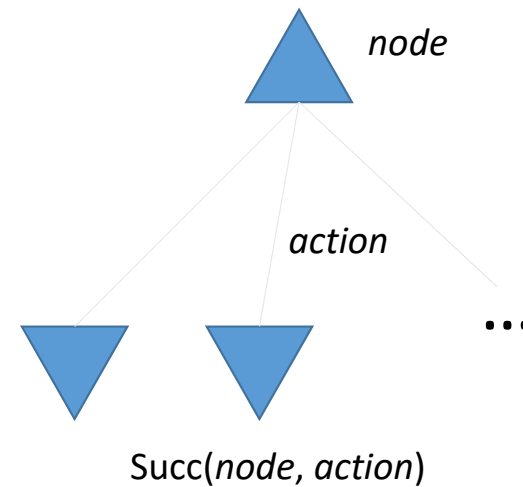
$v = \text{Max}(v, \text{Min-Value}(\text{Succ}(node, action), \alpha, \beta))$

if $v \geq \beta$ return v

$\alpha = \text{Max}(\alpha, v)$

end for

return v



Alpha-beta pruning

- Pruning does not affect final result
- Amount of pruning depends on move ordering
 - Should start with the “best” moves (highest-value for MAX or lowest-value for MIN)
 - For chess, can try captures first, then threats, then forward moves, then backward moves
 - Can also try to remember “killer moves” from other branches of the tree
- With perfect ordering, the time to find the best move is reduced to $O(b^{m/2})$ from $O(b^m)$
 - Depth of search is effectively doubled

Limited-Horizon Computation

Games vs. single-agent search

- We don't know how the opponent will act
 - The solution is not a fixed sequence of actions from start state to goal state, but a *strategy* or *policy* (a mapping from state to best move in that state)

Games vs. single-agent search

- We don't know how the opponent will act
 - The solution is not a fixed sequence of actions from start state to goal state, but a *strategy* or *policy* (a mapping from state to best move in that state)
- Efficiency is critical to playing well
 - The time to make a move is limited
 - The branching factor, search depth, and number of terminal configurations are huge
 - In chess, *branching factor* ≈ 35 and *depth* ≈ 100 , giving a search tree of 10^{154} nodes
 - Number of atoms in the observable universe $\approx 10^{80}$
 - This rules out searching all the way to the end of the game

Evaluation function

- Cut off search at a certain depth and compute the value of an **evaluation function** for a state instead of its minimax value
 - The evaluation function may be thought of as the probability of winning from a given state or the *expected value* of that state
- A common evaluation function is a weighted sum of *features*:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- For chess, w_k may be the **material value** of a piece (pawn = 1, knight = 3, rook = 5, queen = 9) and $f_k(s)$ may be the advantage in terms of that piece
- Evaluation functions may be *learned* from game databases or by having the program play many games against itself

Cutting off search

- **Horizon effect:** you may incorrectly estimate the value of a state by overlooking an event that is just beyond the depth limit
 - For example, a damaging move by the opponent that can be delayed but not avoided
- Possible remedies
 - **Quiescence search:** do not cut off search at positions that are unstable – for example, are you about to lose an important piece?
 - **Singular extension:** a strong move that should be tried when the normal depth limit is reached

Advanced techniques

- **Transposition table** to store previously expanded states
- **Forward pruning** to avoid considering all possible moves
- **Lookup tables** for opening moves and endgames

Chess playing systems

- Baseline system: 200 million node evaluations per move (3 min), minimax with a decent evaluation function and quiescence search
 - 5-ply \approx human novice
- Add alpha-beta pruning
 - 10-ply \approx typical PC, experienced player
- Deep Blue: 30 billion evaluations per move, singular extensions, evaluation function with 8000 features, large databases of opening and endgame moves
 - 14-ply \approx Garry Kasparov
- More recent state of the art ([Hydra](#), ca. 2006): 36 billion evaluations per second, advanced pruning techniques
 - 18-ply \approx better than any human alive?

Summary

- A zero-sum game can be expressed as a minimax tree
- Alpha-beta pruning finds the correct solution. In the best case, it has half the exponent of minimax (can search twice as deeply with a given computational complexity).
- Limited-horizon search is always necessary (you can't search to the end of the game), and always suboptimal.
 - Estimate your utility, at the end of your horizon, using some type of learned utility function
 - Quiescence search: don't cut off the search in an unstable position (need some way to measure "stability")
 - Singular extension: have one or two "super-moves" that you can test at the end of your horizon