

CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu

TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)

William Yeoh (wyeoh@usc.edu)

Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN

Office hours: By appointment

Class page: <http://www-rcf.usc.edu/~macskass/CS561-Spring2010/>

This class will use <http://www.uscden.net/> and class webpage

- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:

[AIMA] Artificial Intelligence: A Modern Approach,
by Stuart Russell and Peter Norvig. (2nd ed)

This time: Outline (Adversarial Search – AIMA Ch. 6)

Game playing

- Perfect play
 - The minimax algorithm
 - alpha-beta pruning
- Resource limitations
- Elements of chance
- Imperfect information



What kind of games?

- **Abstraction:** To describe a game we must capture every relevant aspect of the game. Such as:
 - Chess
 - Tic-tac-toe
 - ...
- **Accessible environments:** Such games are characterized by perfect information
- **Search:** game-playing then consists of a search through possible game positions
- **Unpredictable opponent:** introduces **uncertainty** thus game-playing must deal with **contingency problems**

Searching for the next move

- **Complexity:** many games have a huge search space
 - **Chess:** $b = 35, m = 100 \Rightarrow \text{nodes} = 35^{100}$
if each node takes about 1 ns to explore
then each move will take about **10⁵⁰ millennia**
to calculate.
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate
 1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality result.
 2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.

Two-player games

- A game formulated as a search problem:
 - Initial state: ?
 - Operators: ?
 - Terminal state: ?
 - Utility function: ?

Two-player games

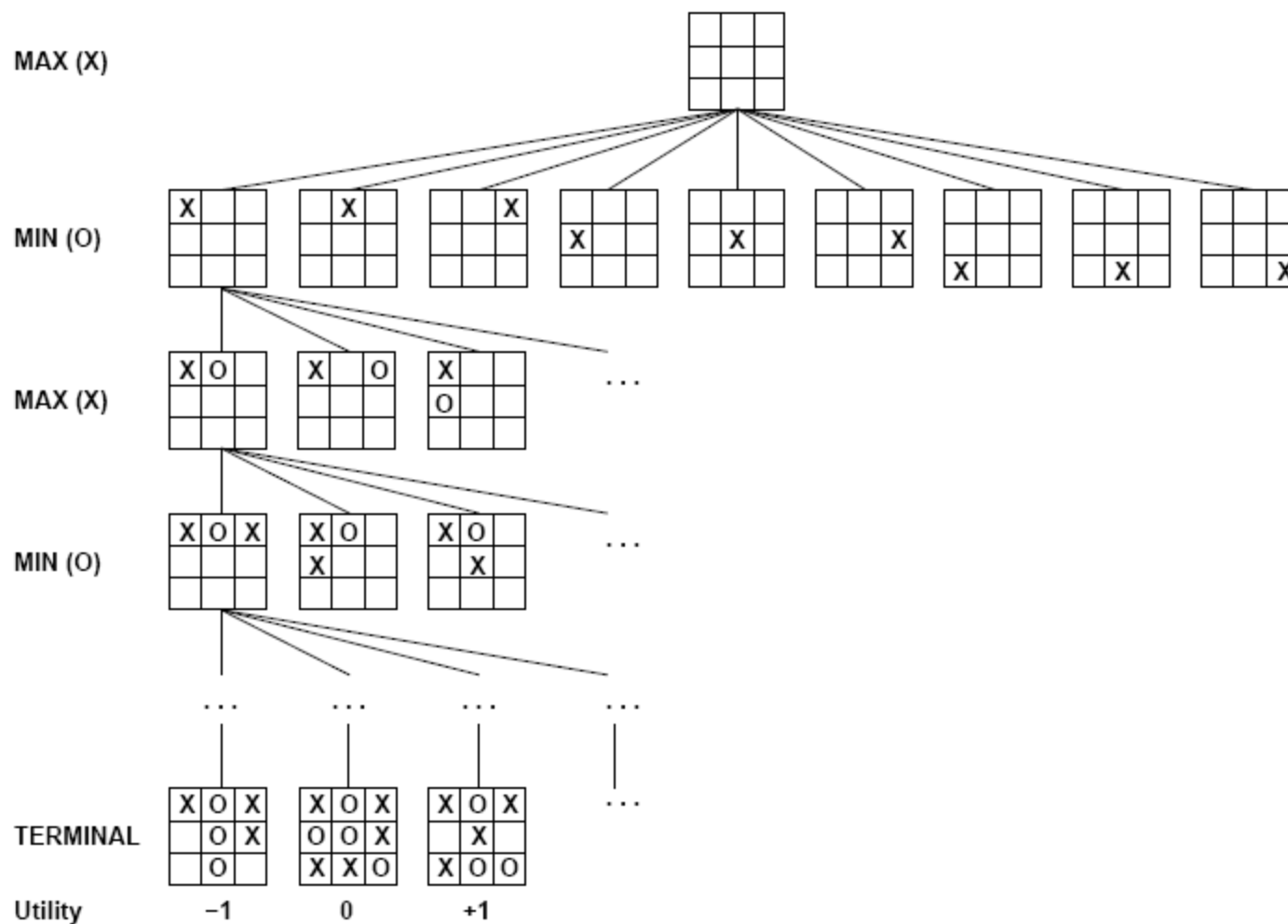
- A game formulated as a search problem:

- Initial state: board position and turn
- Operators: definition of legal moves
- Terminal state: conditions for when game is over
- Utility function: a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)

Games vs. search problems

- "Unpredictable" opponent → solution is a strategy specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate
- Plan of attack:
 - Computer considers possible lines of play (Babbage, 1846)
 - Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
 - Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
 - First chess program (Turing, 1951)
 - Machine learning to improve evaluation accuracy (Samuel, 1952-57)
 - Pruning to allow deeper search (McCarthy, 1956)

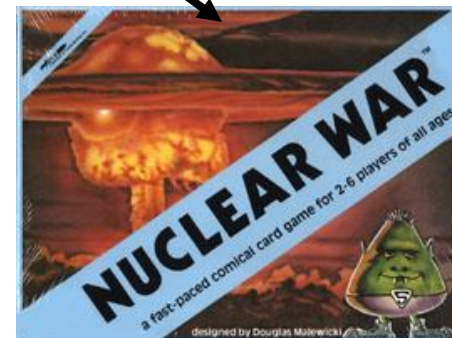
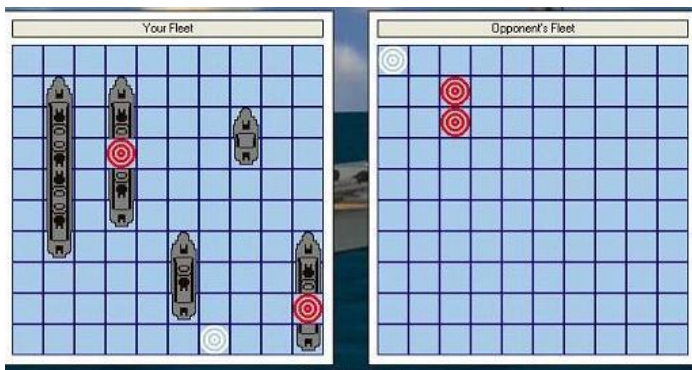
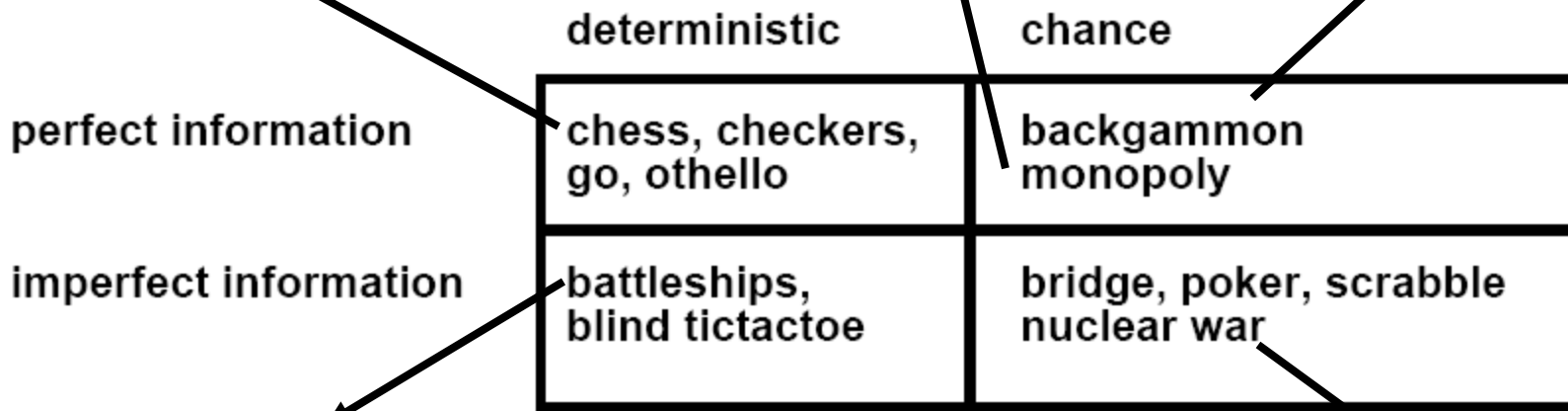
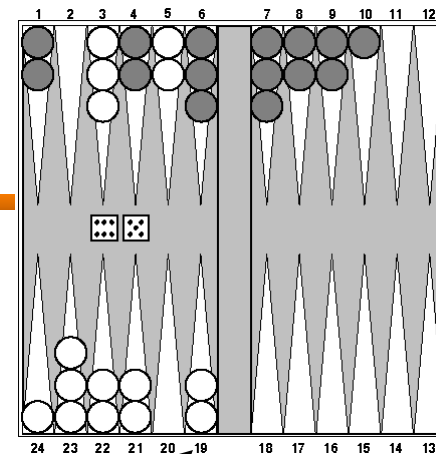
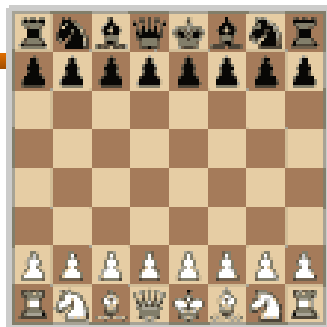
Example: Tic-Tac-Toe



Type of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

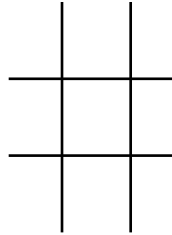
Type of games



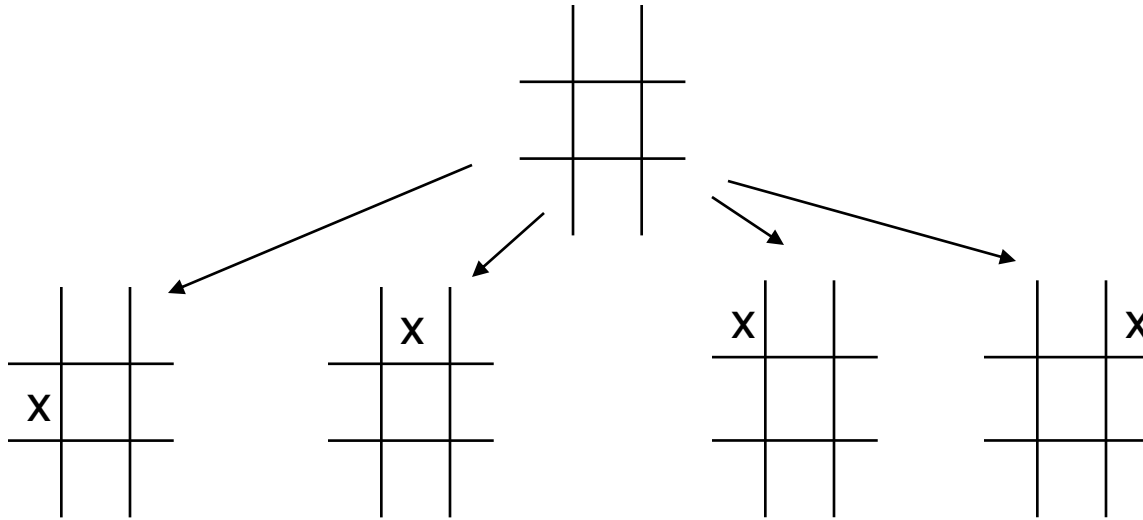
The minimax algorithm

- Perfect play for deterministic environments with perfect information
- **Basic idea:** choose move with highest minimax value
= best achievable payoff against best play
- **Algorithm:**
 1. Generate game tree completely
 2. Determine utility of each terminal state
 3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level
 4. At the root node use minimax decision to select the move with the max (of the min) utility value
- Steps 2 and 3 in the algorithm assume that the opponent will play perfectly.

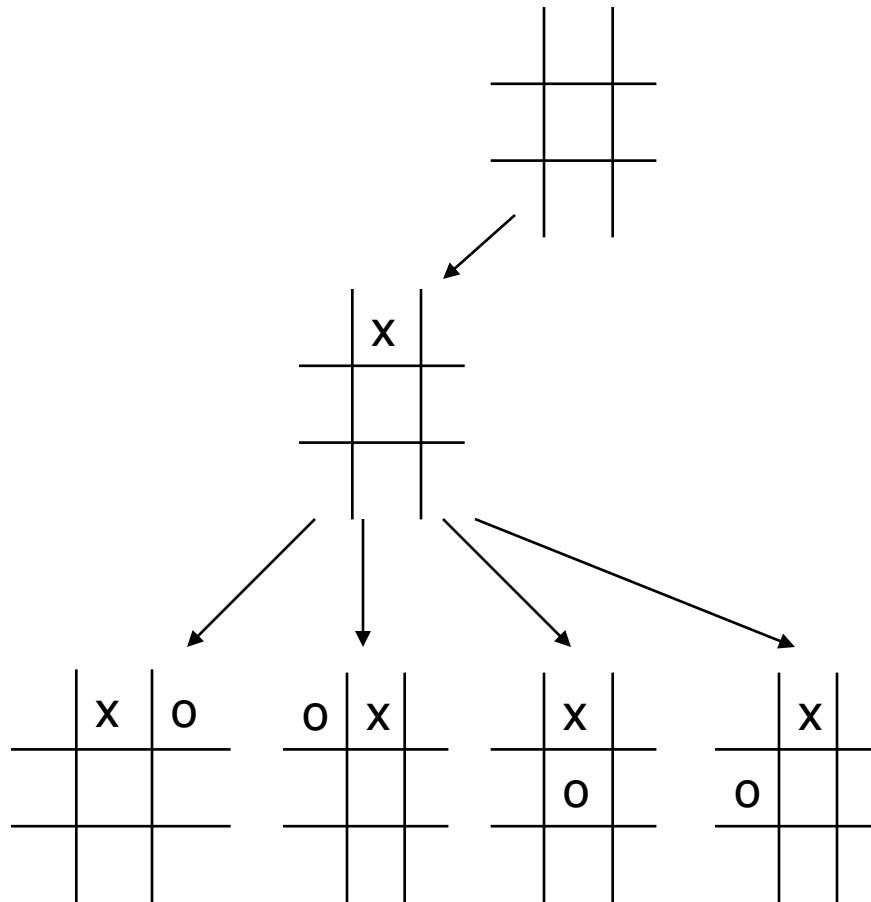
Generate Game Tree



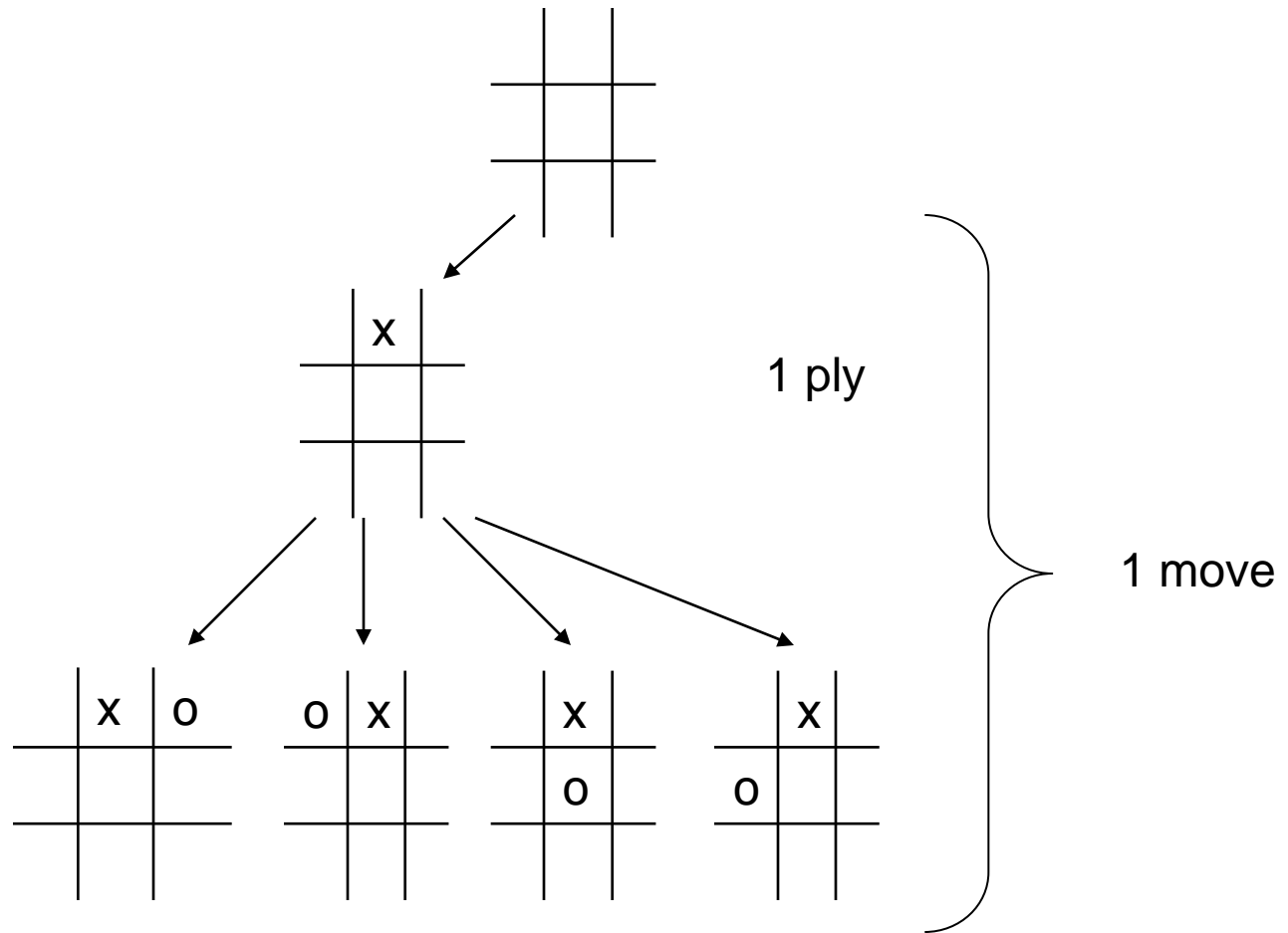
Generate Game Tree



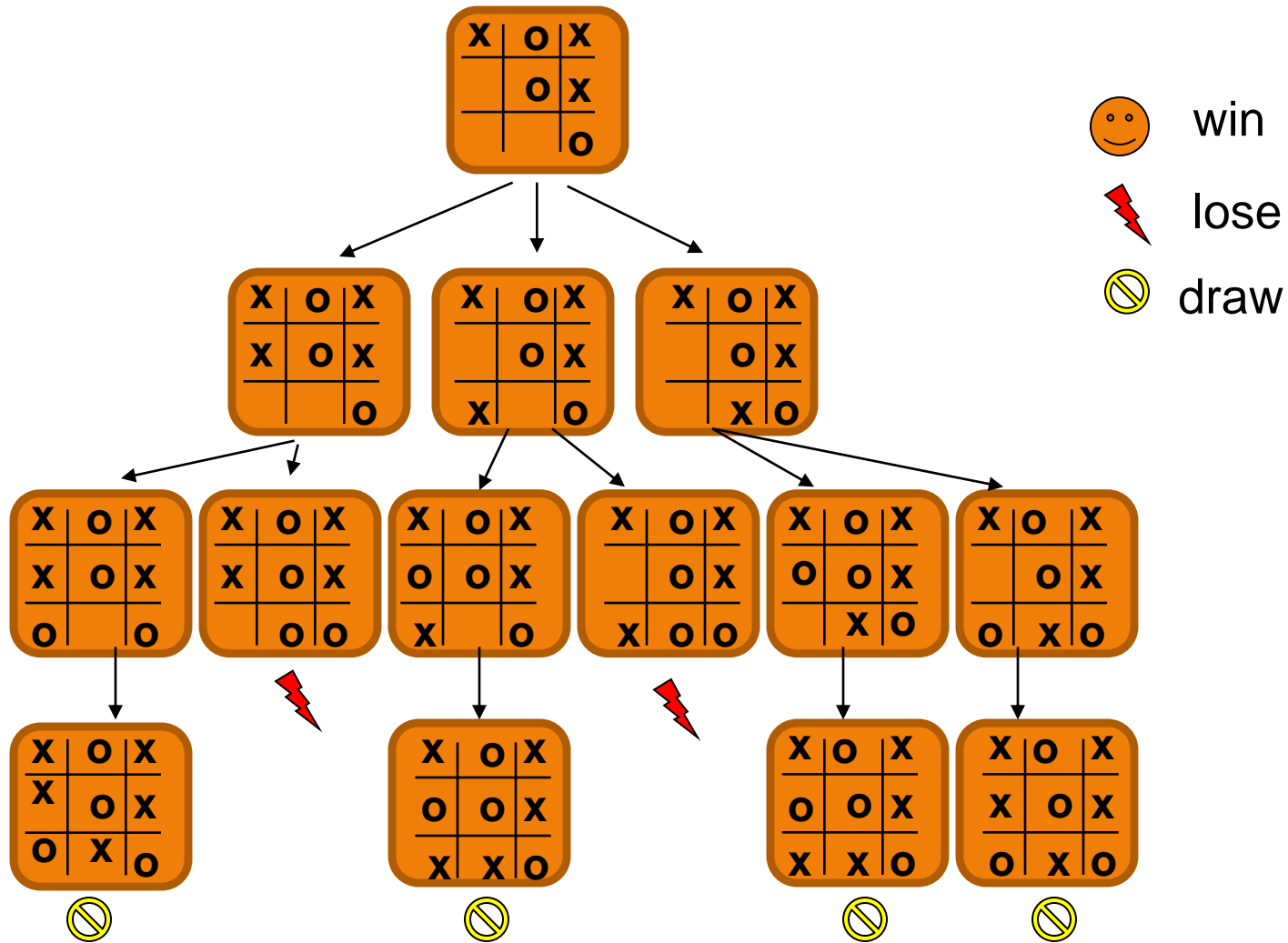
Generate Game Tree



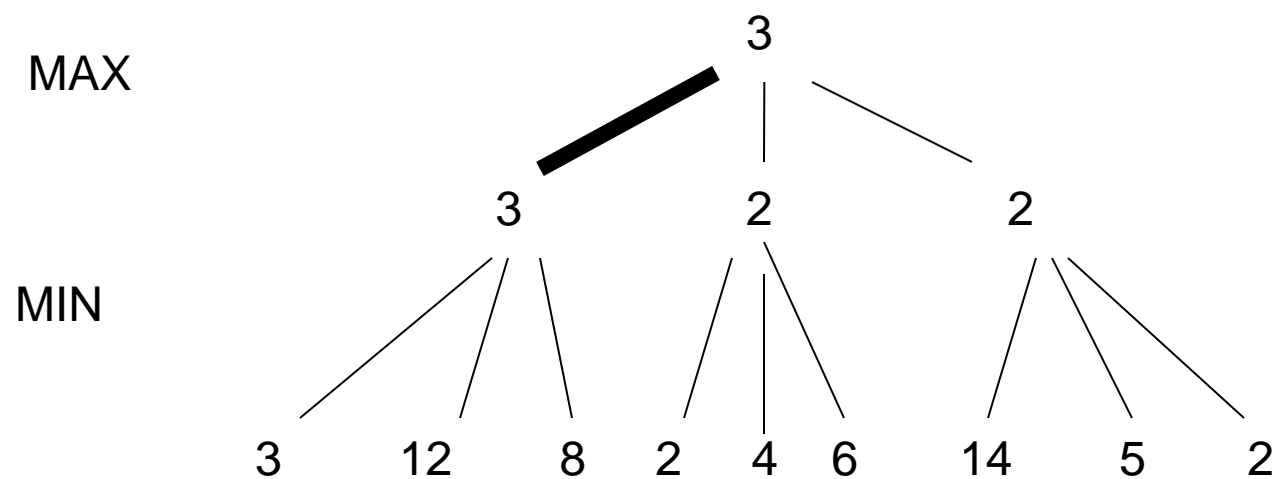
Generate Game Tree



A subtree



Minimax



- Minimize opponent's chance
- Maximize your chance

Minimax: Recursive implementation

```
function MINIMAX-DECISION(state) returns an action  
  inputs: state, current state in game  
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow -\infty$   
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$   
  return v
```

```
function MIN-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow \infty$   
  for a, s in SUCCESSORS(state) do  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$   
  return v
```

Minimax: Recursive implementation

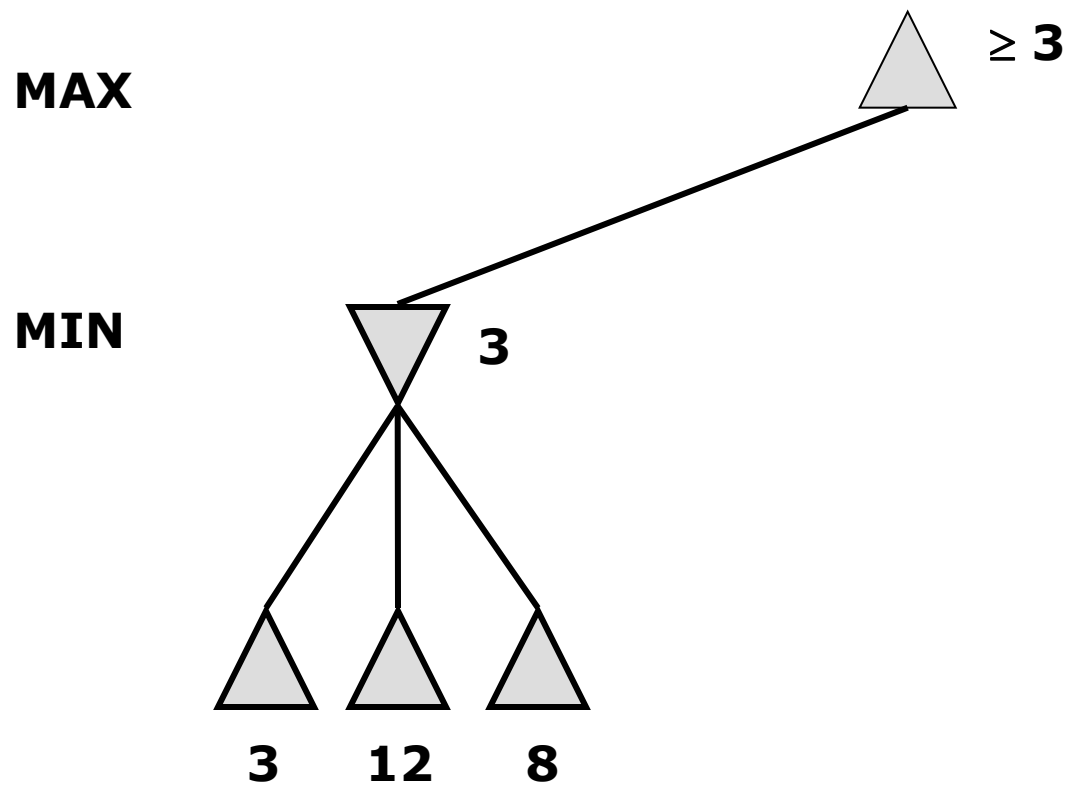
Complete??

Optimal??

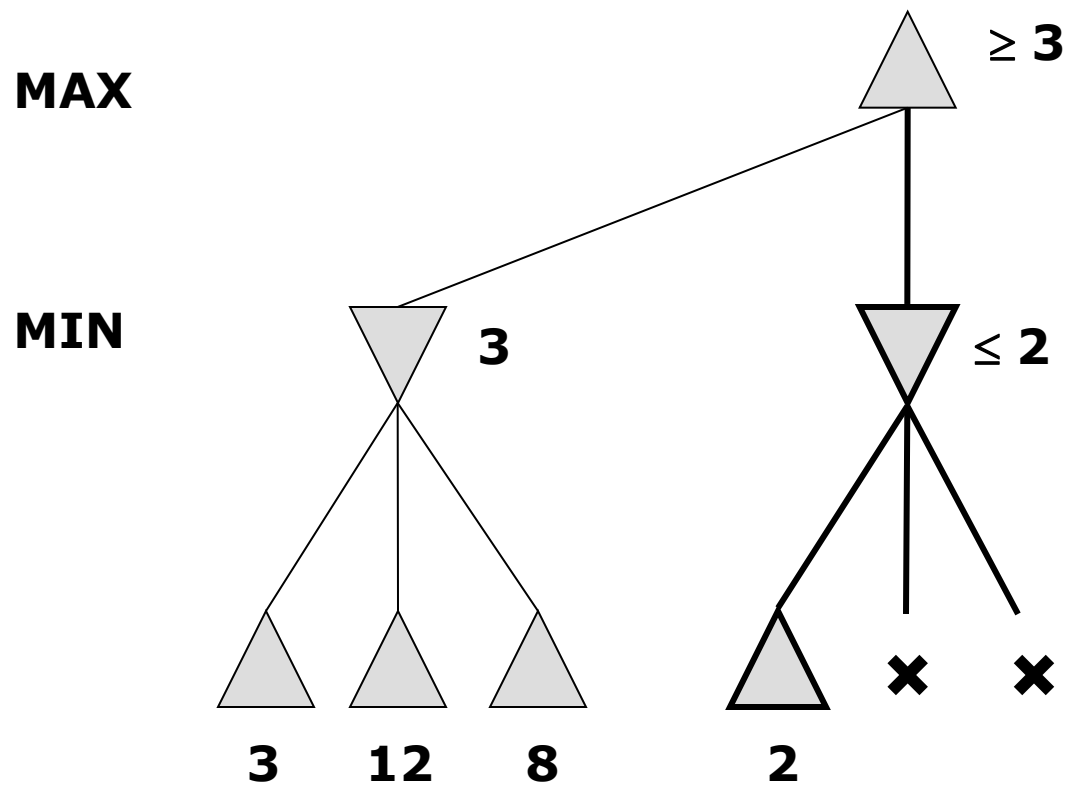
Time complexity??

Space complexity??

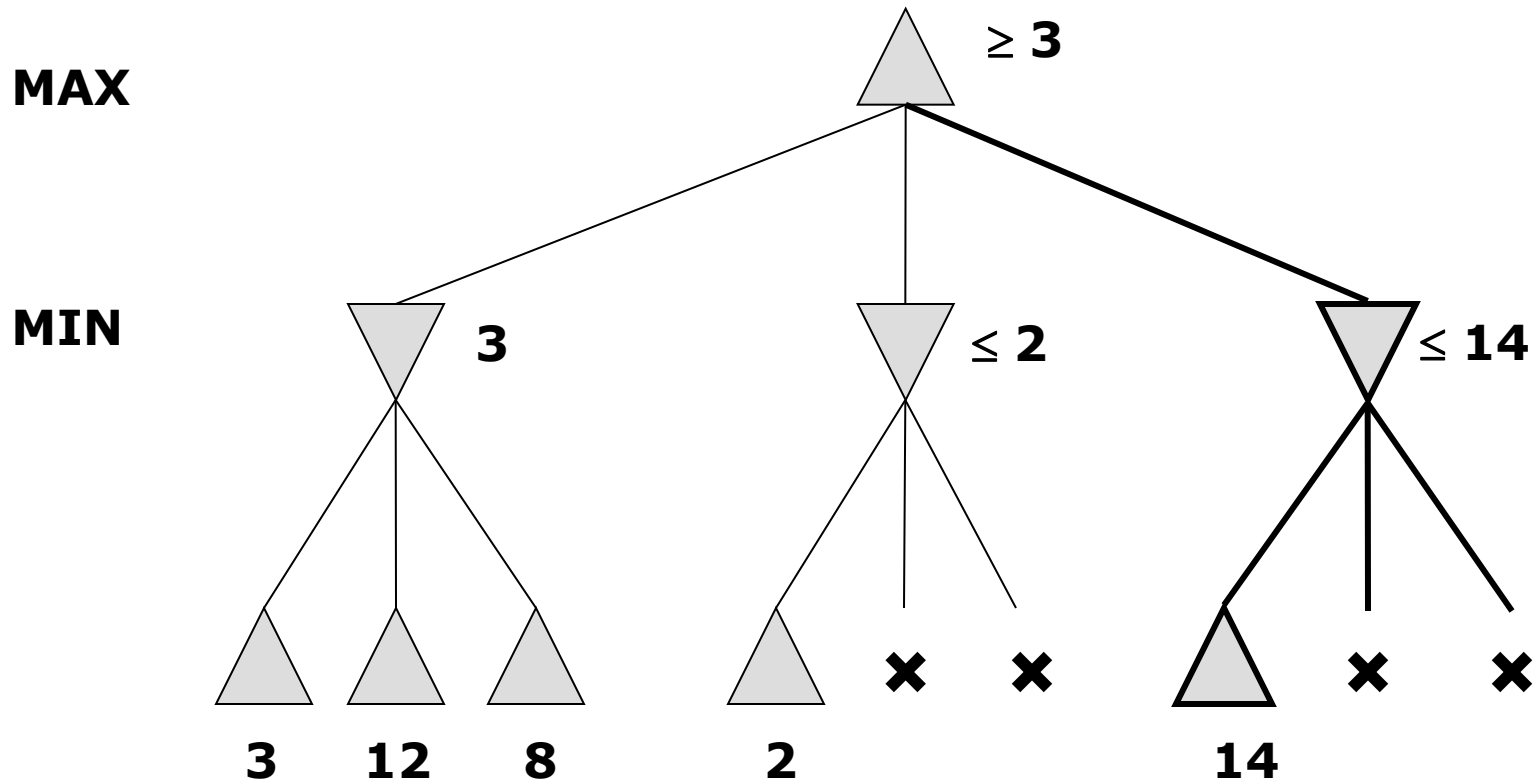
α - β pruning: example



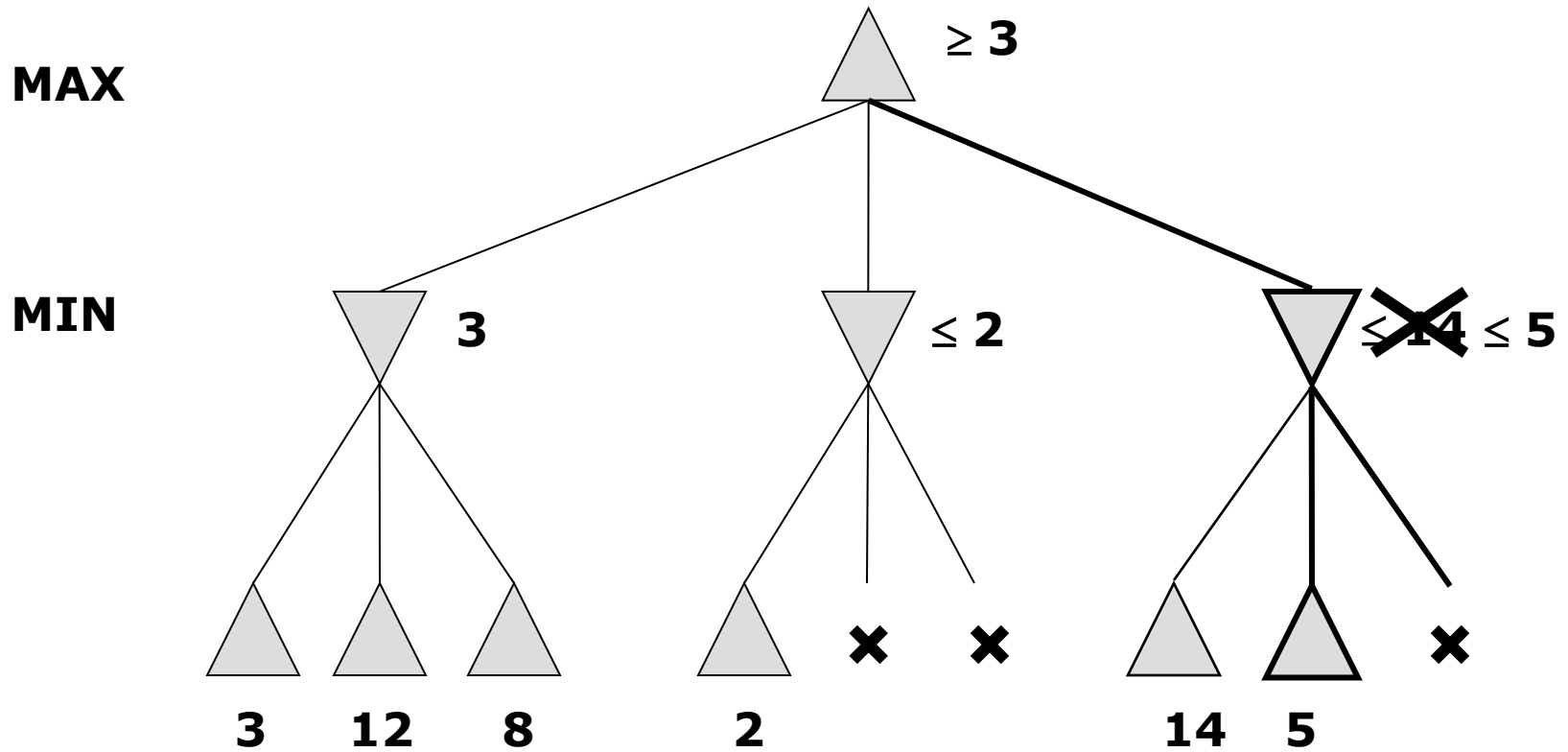
α - β pruning: example



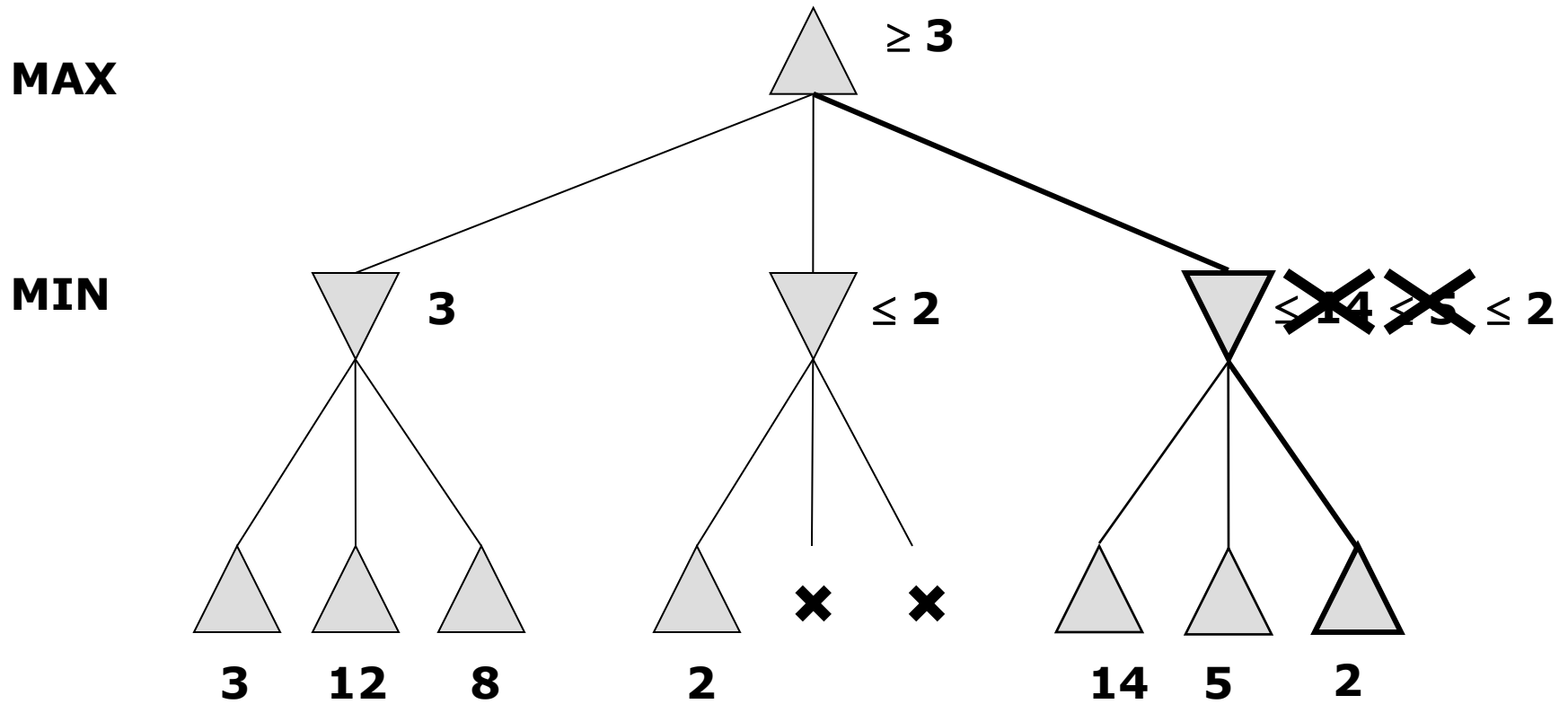
α - β pruning: example



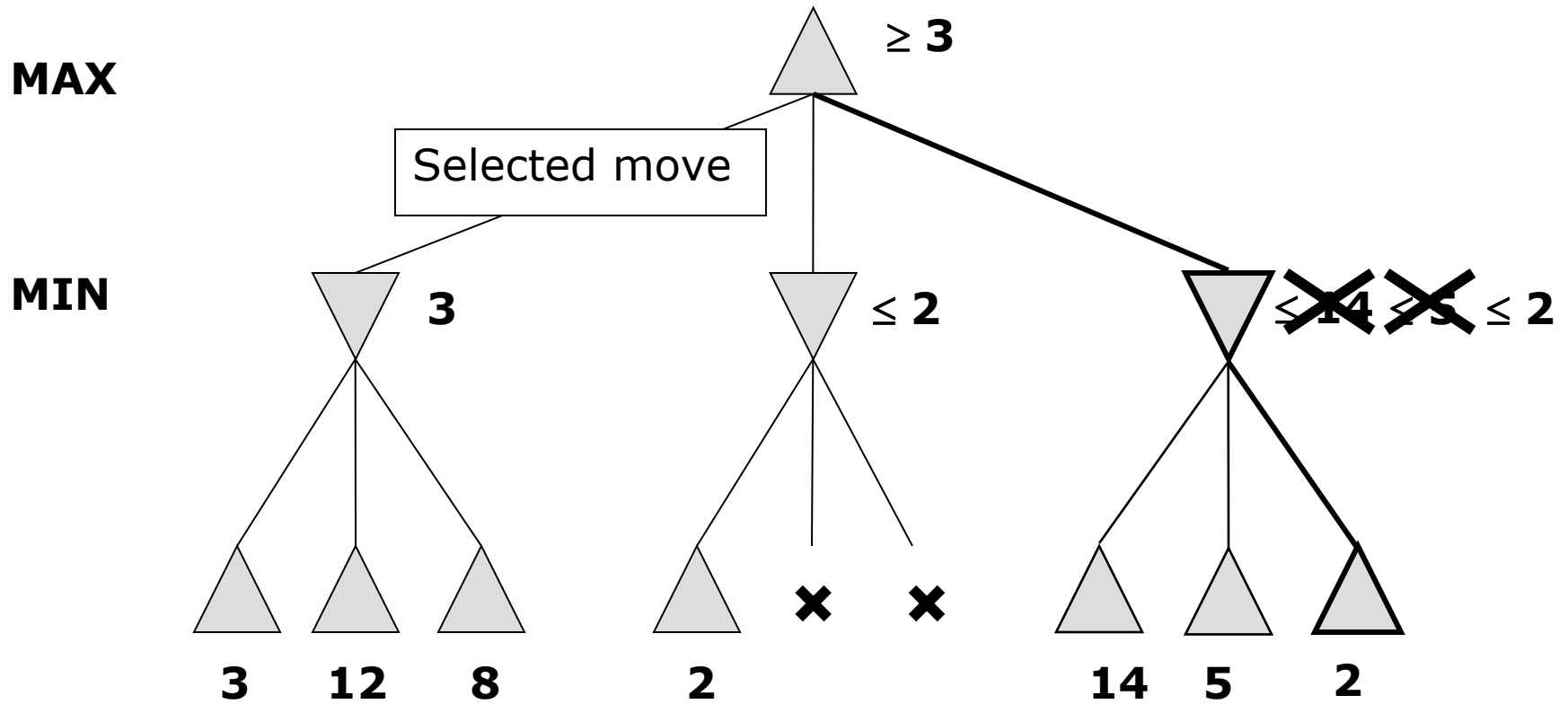
α - β pruning: example



α - β pruning: example



α - β pruning: example



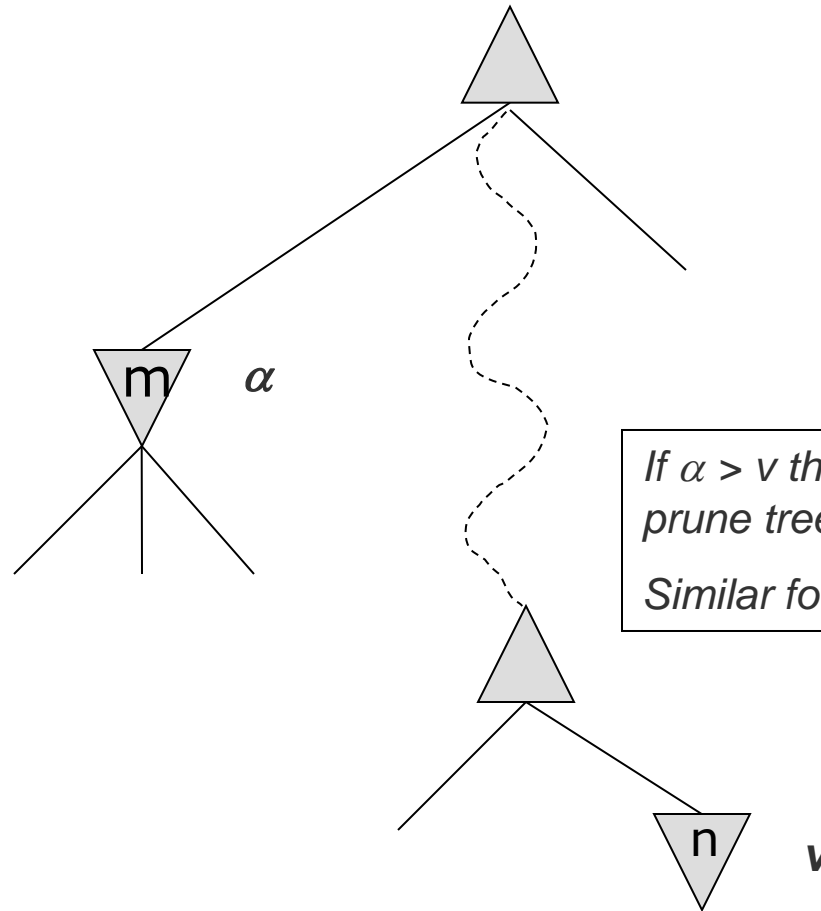
α - β pruning: general principle

Player

Opponent

Player

Opponent



*If $\alpha > v$ then MAX will chose m so
prune tree under n
Similar for β for MIN*

The α - β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

function MAX-VALUE(*state*, α , β) **returns** a utility value
inputs: *state*, current state in game
 α , the value of the best alternative for MAX along the path to *state*
 β , the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
for *a*, *s* in SUCCESSORS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$
 if $v \geq \beta$ **then return** *v*
 $\alpha \leftarrow \text{MAX}(\alpha, v)$
return *v*

function MIN-VALUE(*state*, α , β) **returns** a utility value
same as MAX-VALUE but with roles of α , β reversed

Properties of α - β

Pruning **does not** affect final result

Good move ordering improves effectiveness of pruning

With “perfect ordering,” time complexity = $O(b^{m/2})$
 \Rightarrow **doubles** solvable depth

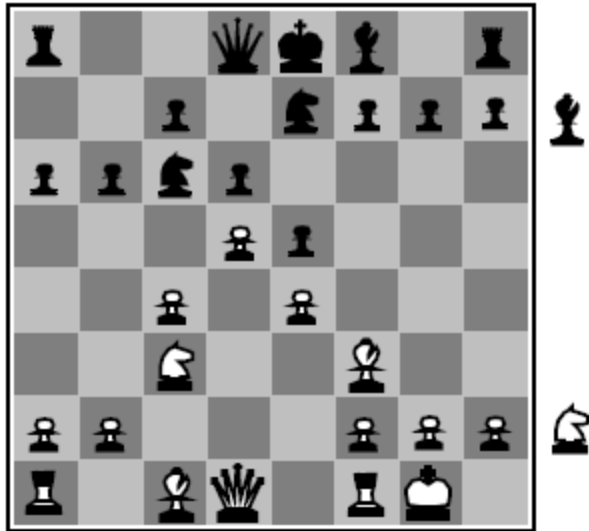
A simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)

Unfortunately, 35^{50} is still impossible!

Resource limits

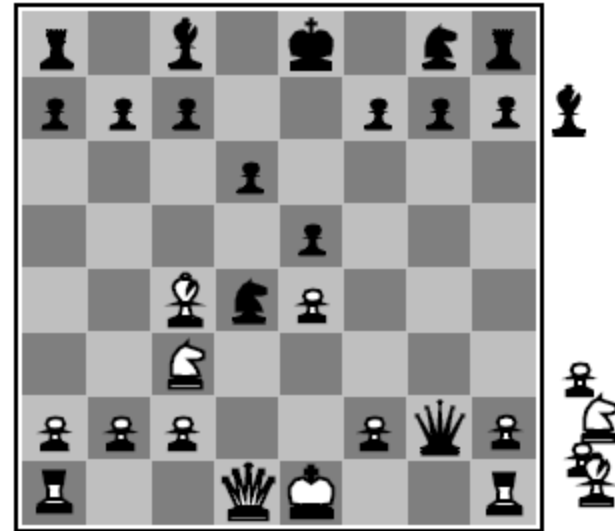
- Standard approach:
 - Use CUTOFF-TEST instead of TERMINAL-TEST
 - e.g., depth limit (perhaps add [quiescence search](#))
 - Use EVAL instead of UTILITY
 - i.e., [evaluation function](#) that estimates desirability of position
- Suppose we have 100 seconds, and can explore 10^4 nodes/second
 - 10^6 nodes per move $\approx 35^{8/2}$
 - α - β reaches depth 8 → pretty good chess program

Evaluation functions



Black to move

White slightly better

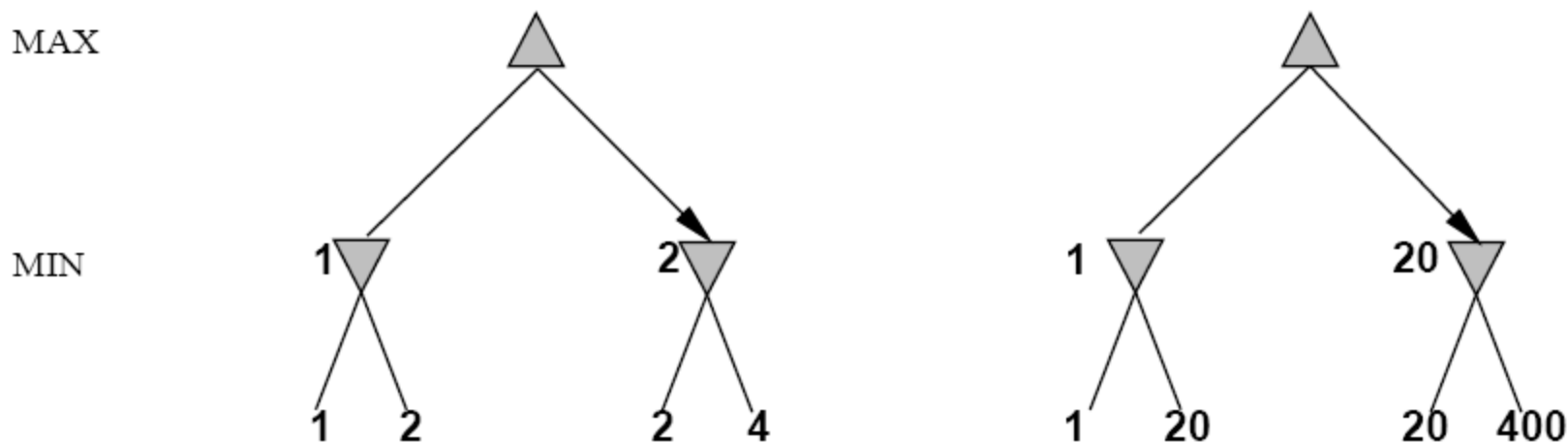


White to move

Black winning

- For chess, typically linear weighted sum of **features**
 - $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
- e.g., $w_1 = 9$ with
 - $f_1(s) = (\text{num white queens}) - (\text{num black queens}), \text{ etc.}$

Digression: Exact values don't matter

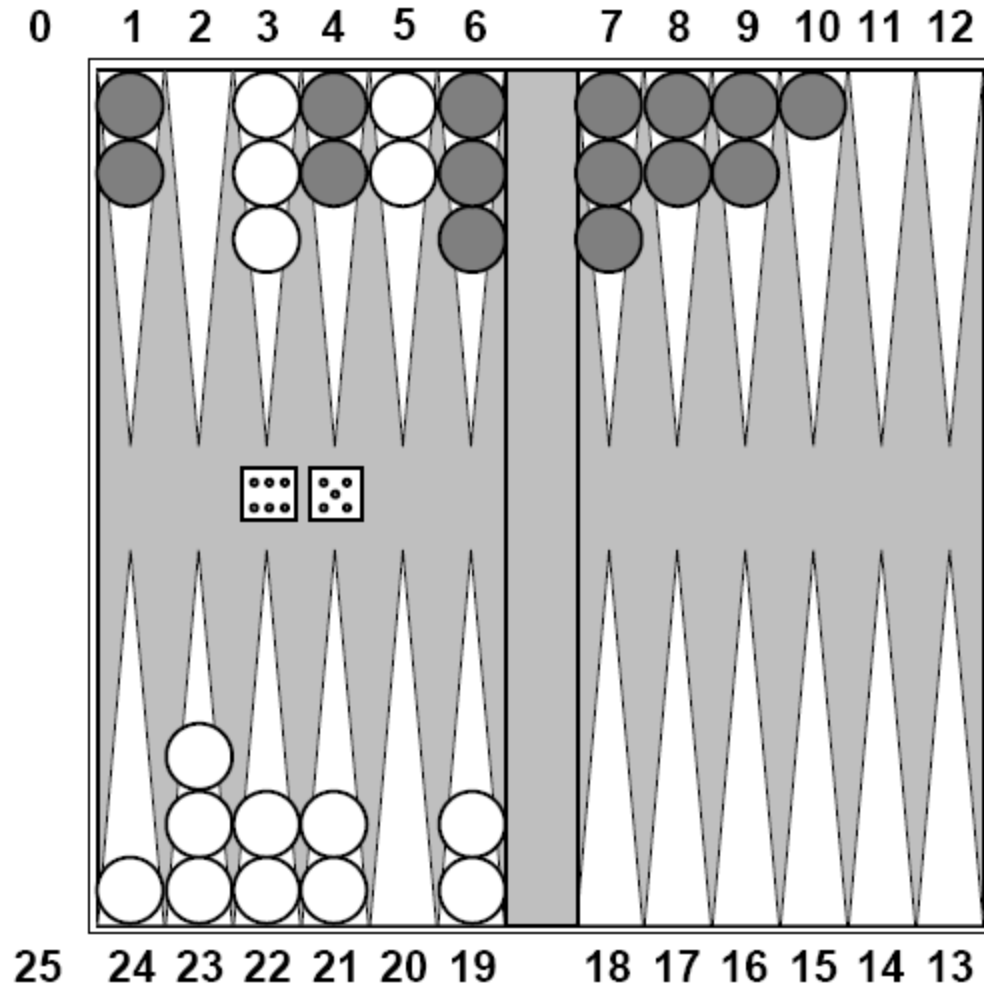


- Behaviour is preserved under any **monotonic** transformation of EVAL
- Only the order matters:
 - Payoff in deterministic games acts as an **ordinal utility** function

Deterministic games in practice

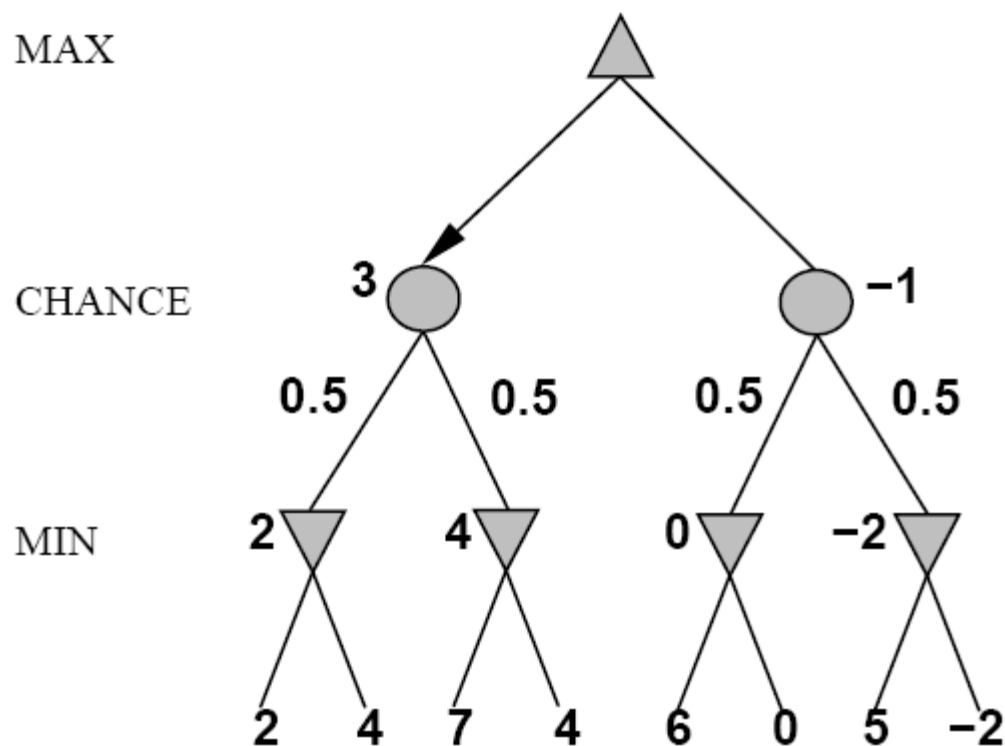
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database dening perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Nondeterministic games: backgammon



Nondeterministic games

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping:



Algorithm for nondeterministic games

- EXPECTIMINIMAX gives perfect play
 - Just like MINIMAX, except we must also handle chance nodes:

...

if *state* is a MAX node then

return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a Min node then

return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a chance node then

return *average* of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

...

Nondeterministic games in practice

Dice rolls increase b : 21 possible rolls with 2 dice

Backgammon \approx 20 legal moves (can be 6,000 with 1-1 roll)
depth 4 = 20 $(21 \times 20)^3 \approx 1.2 \times 10^9$

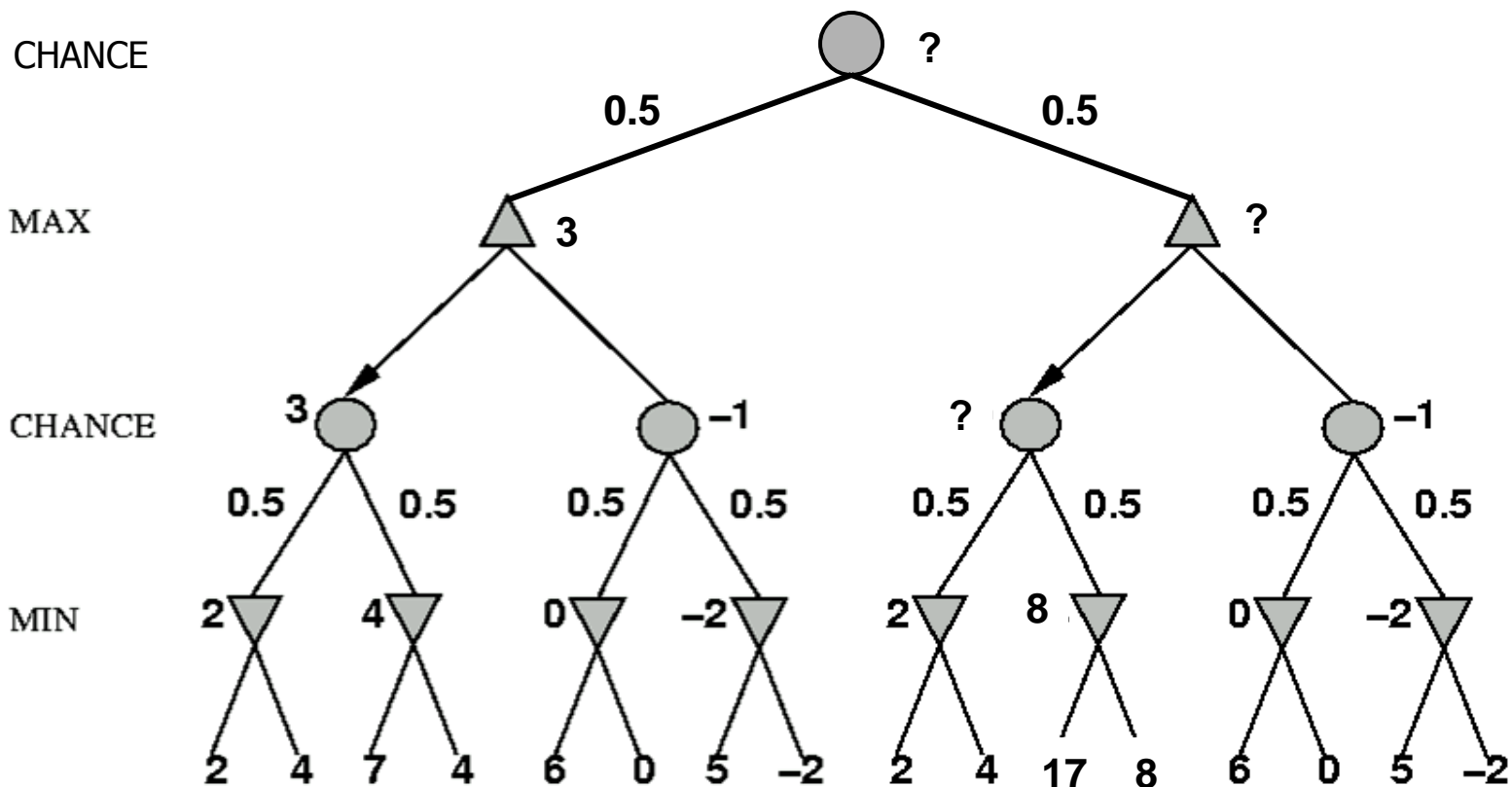
As depth increases, probability of reaching a given node shrinks
→ value of lookahead is diminished

α - β pruning is much less effective

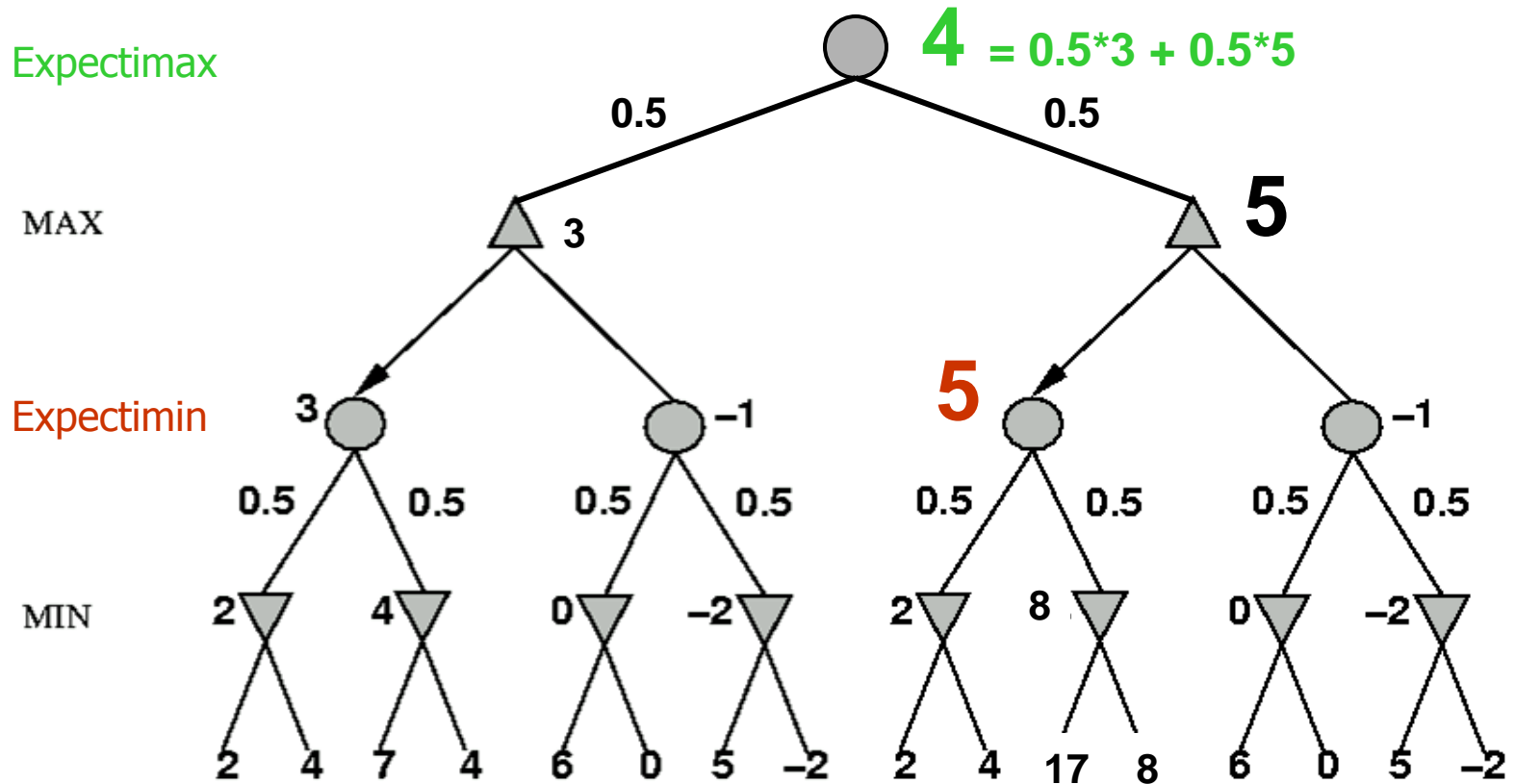
TDGAMMON uses depth-2 search + very good EVAL
 \approx world-champion level

Nondeterministic games: the element of chance

expectimax and **expectimin**, expected values over all possible outcomes

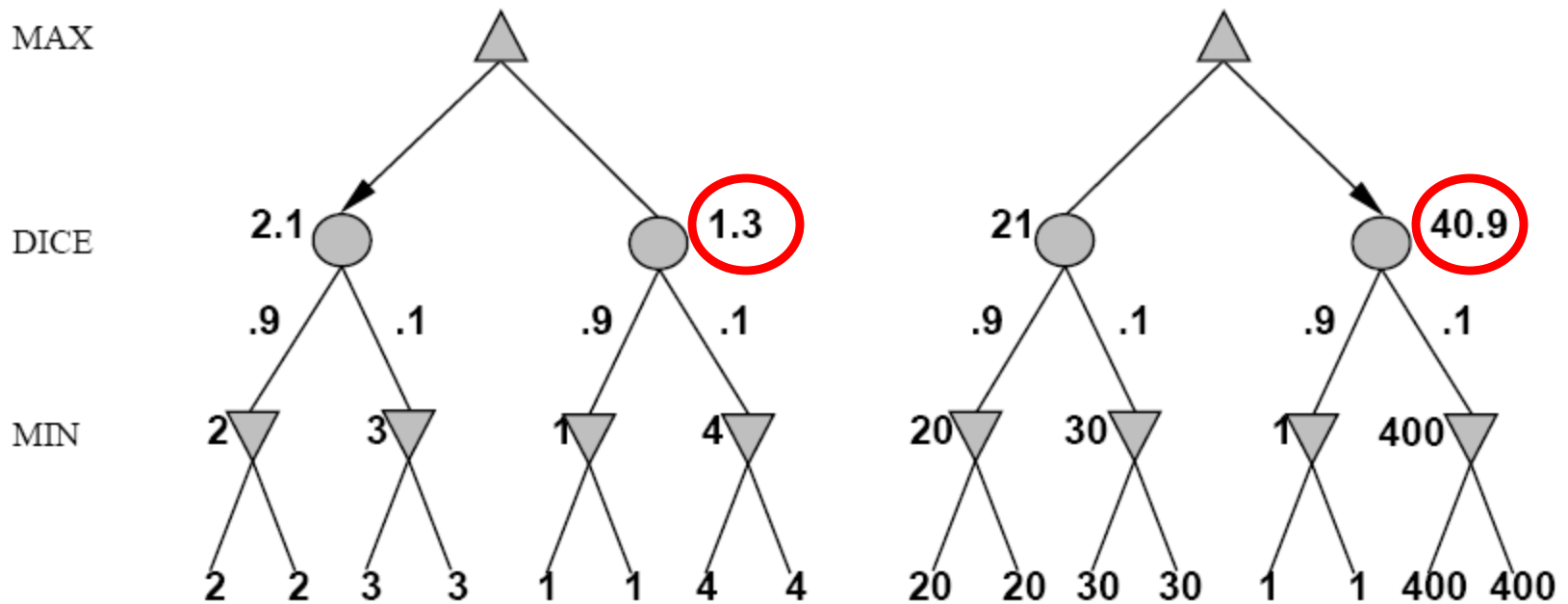


Nondeterministic games: the element of chance



Digression: Exact values DO matter

Order-preserving transformation do not necessarily behave the same!



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of imperfect information

E.g., card games, where opponent's initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game

Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals

Special case: if an action is optimal for all deals, it's optimal.

GIB, current best bridge program, approximates this idea by

- 1) generating 100 deals consistent with bidding information
- 2) picking the action that wins most tricks on average

Commonsense Example

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
 - take the left fork and you'll find a mound of jewels;
 - take the right fork and you'll be run over by a bus.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
 - take the left fork and you'll be run over by a bus;
 - take the right fork and you'll find a mound of jewels.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
 - guess correctly and you'll find a mound of jewels;
 - guess incorrectly and you'll be run over by a bus.

Proper analysis

- Intuition that the value of an action is the average of its values in all actual states is **WRONG**
- With partial observability, value of an action depends on the **information state** or **belief state** the agent is in
- Can generate and search a tree of information states
- Leads to rational behaviors such as
 - Acting to obtain information
 - Signalling to one's partner
 - Acting randomly to minimize information disclosure

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- perfection is unattainable → must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states
- optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design

Exercise: Game Playing

Consider the following game tree in which the evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

- Compute the backed-up values computed by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.
- Compute the backed-up values computed by the alpha-beta algorithm. What nodes will not be examined by the alpha-beta pruning algorithm?
- What move should Max choose once the values have been backed-up all the way?

