Last time: Simulated annealing algorithm

function \text{SIMULATED-ANNEALING}(\text{problem}, \text{schedule}) \text{ returns} a solution state

inputs: \text{problem}, a problem
\text{schedule}, a mapping from time to “temperature”

local variables: \text{current}, a node
\text{next}, a node
\text{T}, a “temperature” controlling the probability of downward steps

\text{current} \leftarrow \text{MAKE-NODE(\text{INITIAL-STATE}[\text{problem}])}

\text{for} \ t \leftarrow 1 \ \text{to} \ \infty \ \text{do}
\quad \text{T} \leftarrow \text{schedule}[t]
\quad \text{if} \ \text{T} = 0 \ \text{then return} \ \text{current}
\quad \text{next} \leftarrow \text{a randomly selected successor of} \ \text{current}
\quad \Delta \text{E} \leftarrow \text{VALUE}[\text{next}] - \text{VALUE}[\text{current}]
\quad \text{if} \ \Delta \text{E} > 0 \ \text{then} \ \text{current} \leftarrow \text{next}
\quad \text{else} \ \text{current} \leftarrow \text{next} \ \text{only with probability} \ \text{e}^{\Delta \text{E}/\text{T}}

Note: goal here is to maximize E.
Last time: Simulated annealing algorithm

function SIMULATED-ANNEALING(problem, schedule) returns a solution state

inputs: problem, a problem
         schedule, a mapping from time to “temperature”

local variables: current, a node
                  next, a node
                  T, a “temperature” controlling the probability of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T=0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] - VALUE[current]
    if ΔE < 0 then current ← next
    else current ← next only with probability $e^{-ΔE/T}$

Algorithm when goal is to minimize E.
This time: Outline

- Game playing
  - The minimax algorithm
  - Resource limitations
  - alpha-beta pruning
  - Elements of chance
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
What kind of games?

- **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 \( 10^{50} \) millennia to calculate.
Searching for the next move

- **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: ?
Game vs. search problem

“Unpredictable” opponent \(\Rightarrow\) solution is a contingency plan

Time limits \(\Rightarrow\) unlikely to find goal, must approximate

Plan of attack:

- algorithm for perfect play (Von Neumann, 1944)
- finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
- pruning to reduce costs (McCarthy, 1956)
Example: Tic-Tac-Toe

Question:
1. \( b \) (branching factor) = ?
2. \( m \) (max depth) = ?
Type of games

- Deterministic
  - Perfect information: chess, checkers, go, othello
  - Imperfect information: backgammon, monopoly

- Chance
  - Bridge, poker, scrabble, nuclear war
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
The minimax algorithm

**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 mins) utility value

- Steps 2 and 3 in the algorithm assume that the opponent will play perfectly.
Generate Game Tree
Generate Game Tree

1 ply

1 move
A subtree

- win
- lose
- draw
What is a good move?
Minimax

- Minimize opponent’s chance
- Maximize your chance
minimax = maximum of the minimum

1st ply

2nd ply

MAX

MIN
JavaApplet

-Minimax java applet

dd
Minimax: Recursive implementation

function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← Max(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← Min(v, MAX-VALUE(s))
    return v

Complete: ?
Optimal: ?
Time complexity: ?
Space complexity: ?
1. Move evaluation without complete search

- Complete search is too complex and impractical
- **Evaluation function:** evaluates value of state using heuristics and cuts off search
- **New MINI MAX:**
  - **CUTOFF-TEST:** cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
  - **EVAL:** evaluation function to replace utility function (e.g., number of chess pieces taken)
Do We Have To Do All That Work?

MAX

MIN

3 12 8
Evaluation functions

- Weighted linear evaluation function:
  - to combine $n$ heuristics: $f = \omega_1 f_1 + \omega_2 f_2 + \ldots + \omega_n f_n$

E.g,

- $\omega$’s could be the values of pieces (1 for prawn, 3 for bishop)
- $f$’s could be the number of type of pieces on the board
Note: exact values do not matter

Ordering is preserved

MAX

MIN

Behaviour is preserved under any monotonic transformation of Eval

Only the order matters: payoff in deterministic games acts as an ordinal utility function
Minimax with cutoff: viable algorithm?

MinimaxCutoff is identical to MinimaxValue except
1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!

4-ply \(\approx\) human novice
8-ply \(\approx\) typical PC, human master
12-ply \(\approx\) Deep Blue, Kasparov

Assume we have 100 seconds, evaluate \(10^4\) nodes/s; can evaluate \(10^6\) nodes/move