Chapter 3 State-Space Search

COMP 6721 Introduction of AI

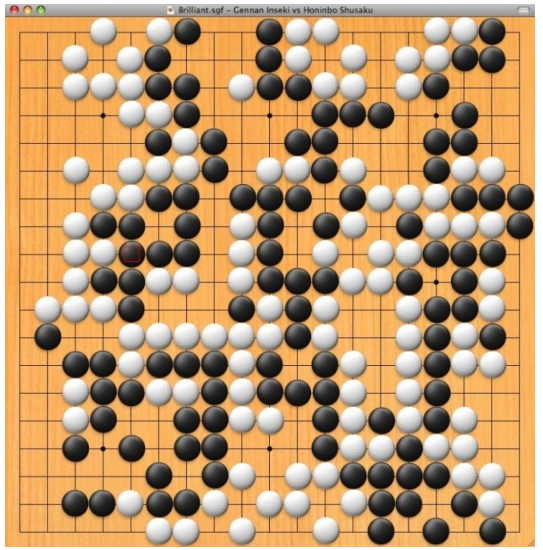
Russell & Norvig – Section 5.1 & 5.4

Some slides from: robotics.stanford.edu/~latombe/cs121/2003/home.htm

State-Space Search

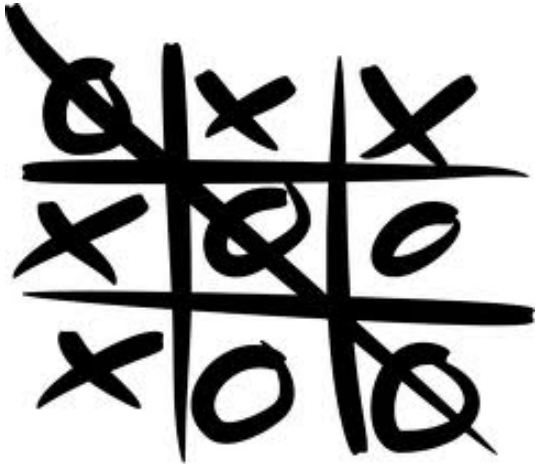
- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - ***Informed Search***
    - ***Hill Climbing***
    - ***Best – Frist***
    - ***Design Heuristics***
    - ***A****

Motivation



GO

chess



tic-tac-toe

Adversarial Search

- MiniMax Search
- Alpha – beta pruning

Adversarial Search

- two or more players with conflicting goals are trying to explore the same search space for the solution, also known as Game Search.
- Classical application for heuristic search
  - simple games: exhaustibly searchable
  - complex games: only partial search possible
  - additional problem: playing against opponent

Adversarial Search

- Type of games:
- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also.

  Examples: Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information.

  Examples: tic-tac-toe, Battleship, blind, Bridge, etc.

Adversarial Search

- Type of games:
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them.

  Examples: chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed.

  Examples: Backgammon, Monopoly, Poker, etc.

Adversarial Search

- Type of games:
- **Zero-Sum Game:**

  -- Zero-sum games are adversarial search which involves pure competition.

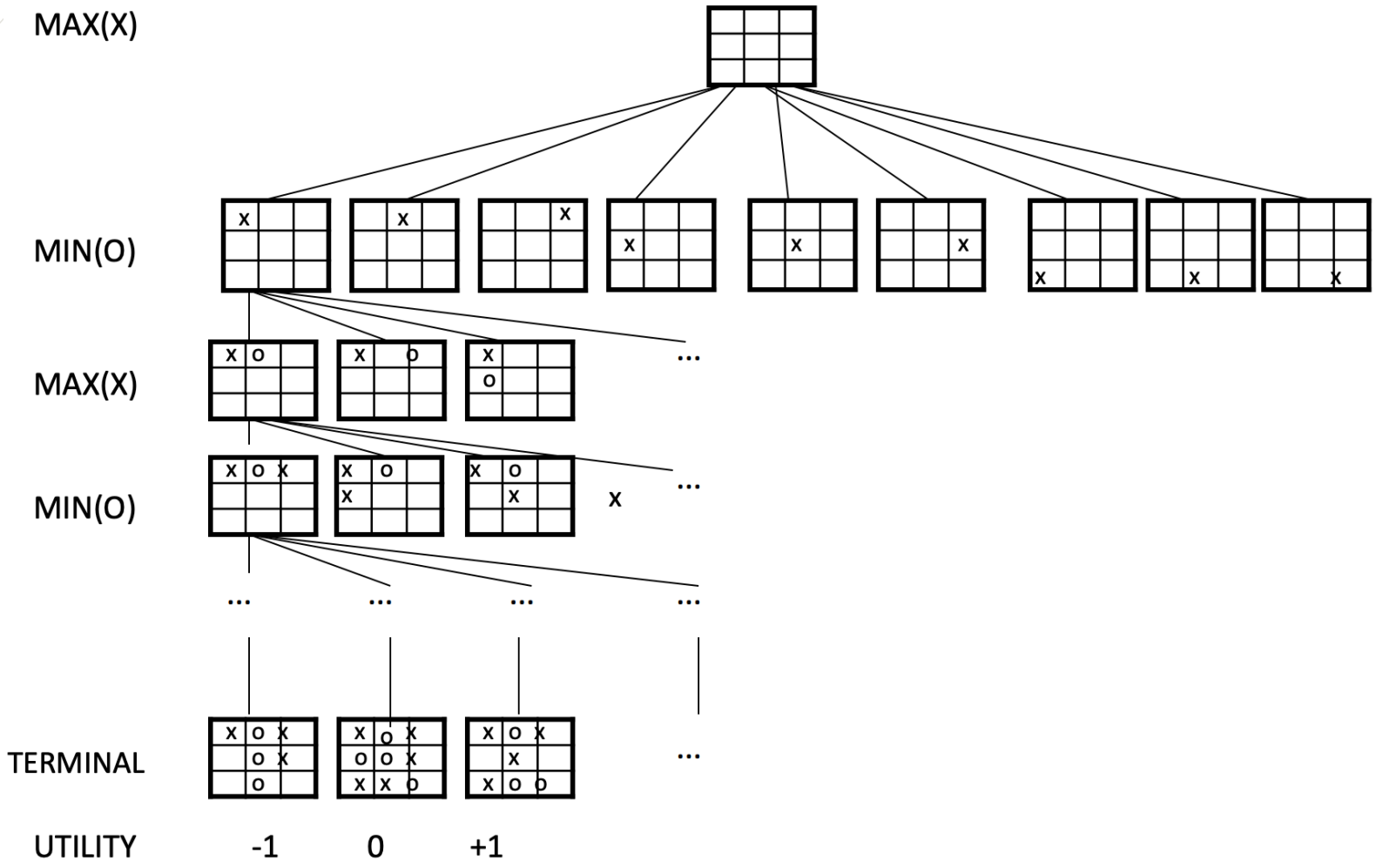
  -- If the total gains of one player are added up, and the total losses are subtracted, they will sum to zero.

  -- One player of the game try to maximize one single value, while other player tries to minimize it.

  -- Each move by one player in the game is called as ply.

  -- Example: Chess, tic-tac-toe

Partial Game Search Tree of Tic-Tac-Toe



MiniMax Search

- Game between two opponents, MIN and MAX
  - MAX tries to win, and
  - MIN tries to minimize MAX's score
- Existing heuristic search methods do not work
  - would require a helpful opponent
  - Need to incorporate "hostile" moves into search strategy

Exhaustive MiniMax Search

- For small games where exhaustive search is feasible
- Procedure:
  1. build complete game tree
  2. label each level according to player's turn (MAX or MIN)
  3. label leaves with a utility function to determine the outcome of the game. e.g., (0, 1) or (-1, 0, 1)
  4. propagate this value up:
     - if parent=MAX, give it max value of children
     - if parent=MIN, give it min value of children
  5. Select best next move for player at root as the move leading to the child with the highest value (for MAX) or lowest values (for MIN)

Example: Game of Nim

- Rules
  - 2 players start with a pile of tokens
  - move: split (any) existing pile into two non-empty differently-sized piles
  - game ends when no pile can be unevenly split
  - player who cannot make his move loses

https://www.youtube.com/watch?v=niMjxNtiuu8

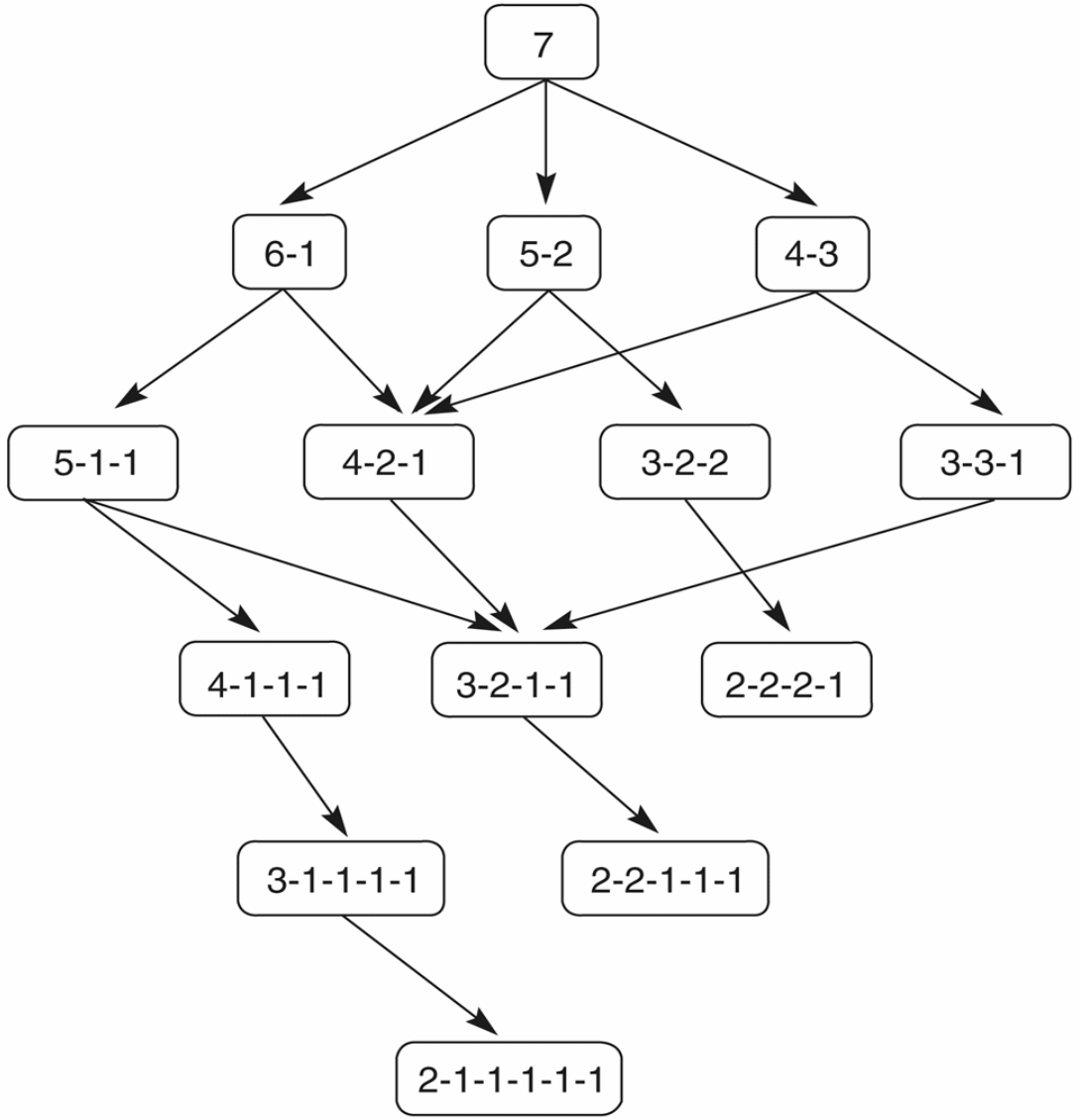
MiniMax Search Algorithms

function MINIMAX-DECISION(*game*) **returns** *an operator*
    for each *op* **in** OPERATORS[game]**do**
        VALUE[*op*]←MINIMAX-VALUE(APPLY(*op*,*game*),*game*)
    end
    return the *op* with the highest VALUE[*op*]

---

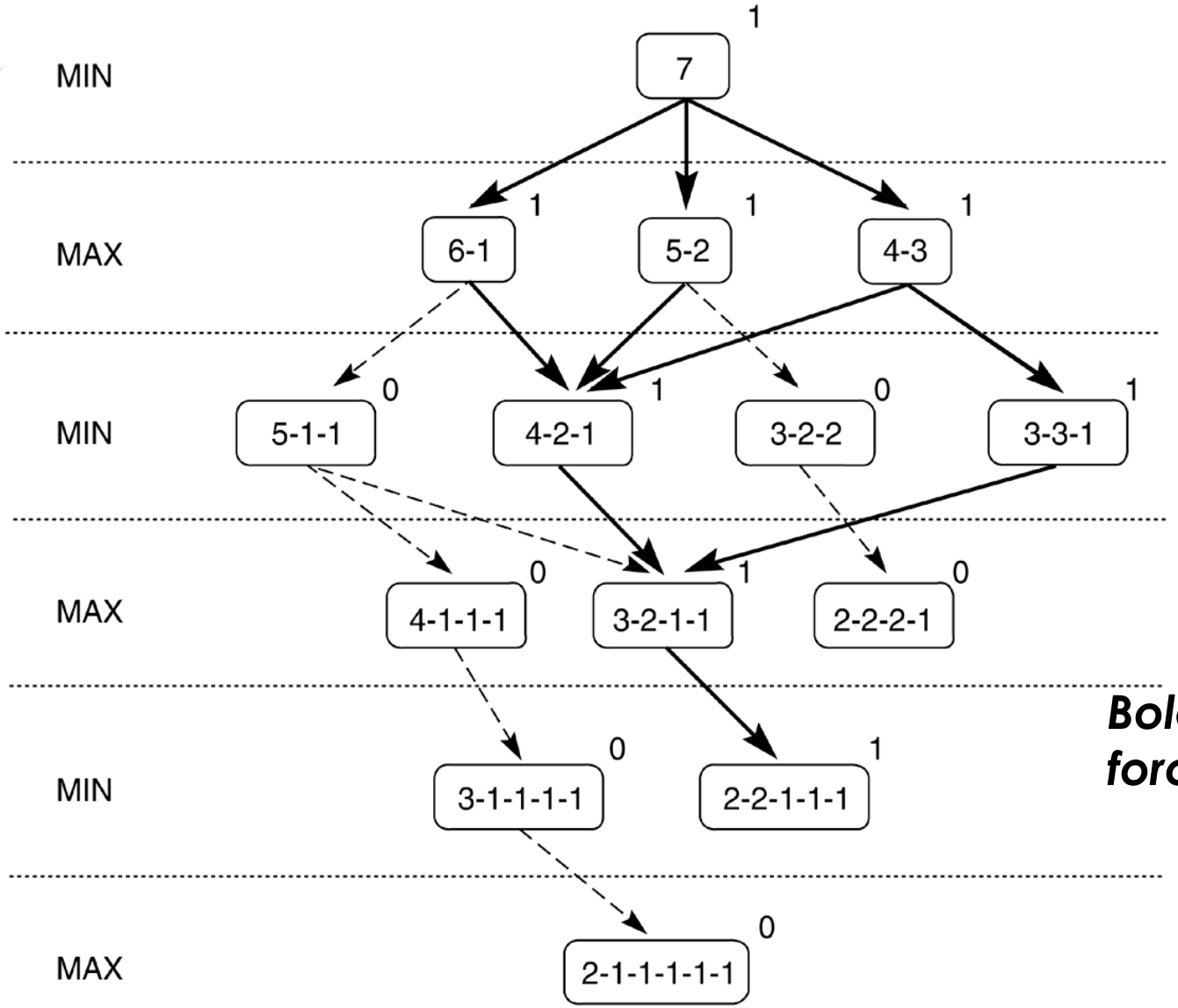
function MINIMAX-VALUE(*state*,*game*) **returns** *a utility value*
    if TERMINAL-TEST[*game*](*state*) **then**
        return UTILITY[*game*](*state*)
    else if MAX is to move in *state* **then**
        return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

State Space of Game Nim



- start with one pile of tokens
- each step has to divide one pile of tokens into 2 non-empty piles of different size
- player without a move left loses game

Exhaustive MiniMax for Nim



Bold lines indicate forced win for MAX

n-ply MiniMax with Heuristic

- Exhaustive search for interesting games is rarely feasible
- Search only to predefined level
  - called n-ply look-ahead
  - n is number of levels
- No exhaustive search
  - nodes evaluated with heuristics and not win/loss
  - indicates best state that can be reached
  - horizon effect
- Games with opponent
  - simple strategy: try to maximize difference between players using a heuristic function e(n)

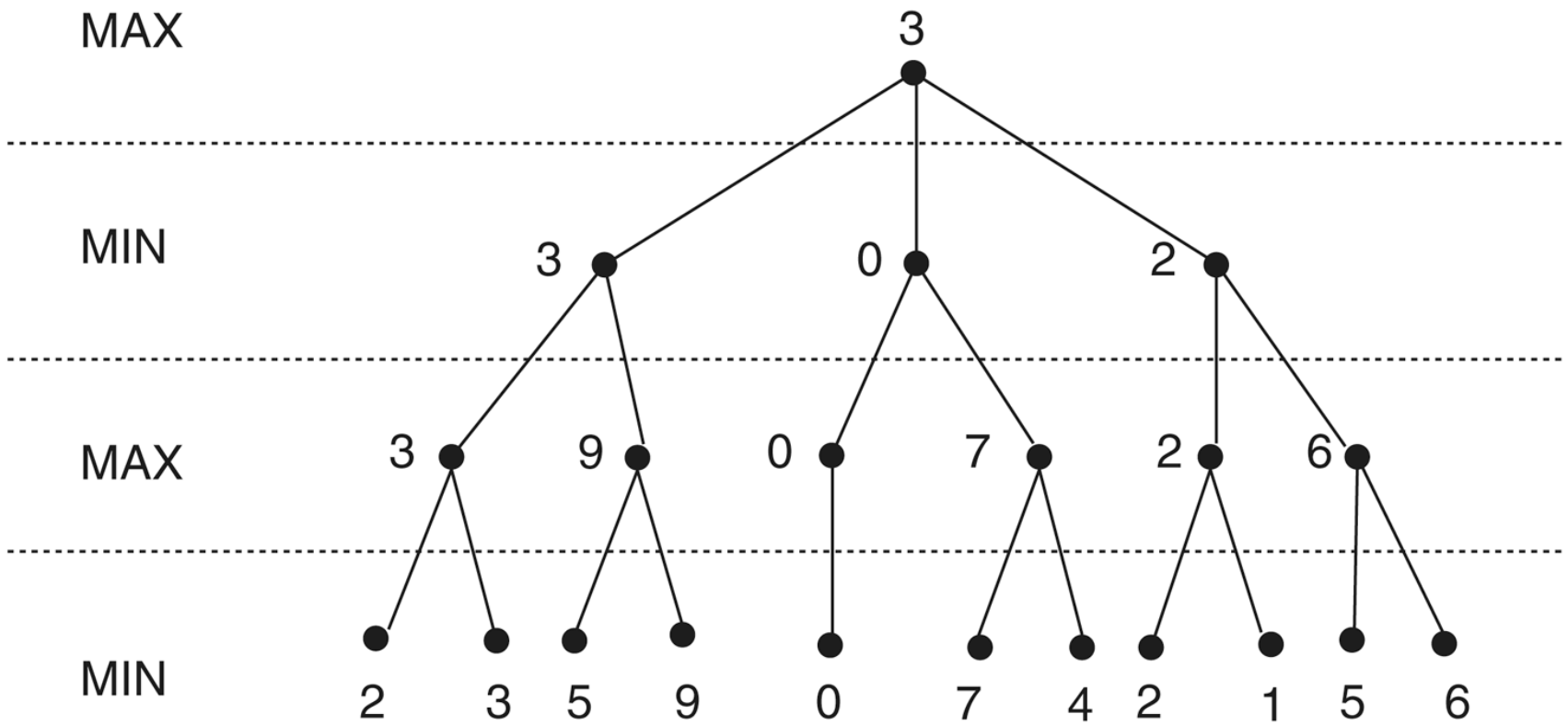
Heuristic Function for 2-player games

- simple strategy:
  - try to maximize difference between MAX's game and MIN's game
- typically called e(n)
- e(n) is a heuristic that estimates how favorable a node n is for MAX
  - e(n) > 0 --> n is favorable to MAX
  - e(n) < 0 --> n is favorable to MIN
  - e(n) = 0 --> n is neutral

Choosing an Heuristic Function e(n)

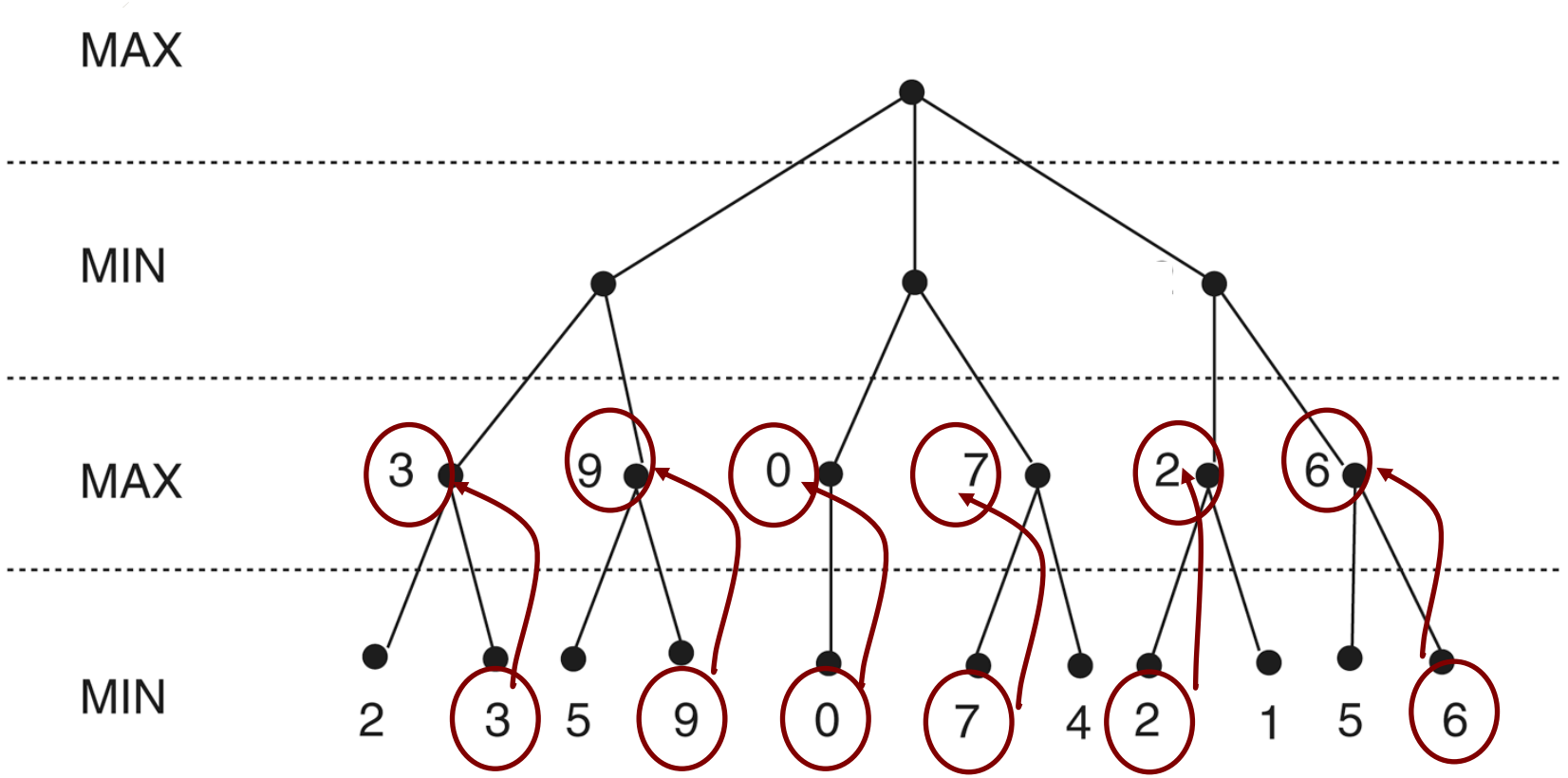
- Usually e(n) is a weighted sum of various features:
  - $e(n) = \sum w_i f_i(n)$
- E.g. of features:
  - f_1 = number of pieces left on the game for MAX
  - f_2 = number of possible moves left for MAX
  - f_3 = -(number of pieces left on the game for MIN)
  - f_4 = -(number of possible moves left for MIN)
- E.g. of weights:
  - w_1 = 0.5 // f_1 is a very important feature
  - w_2 = 0.2 // f_2 is not very important
  - w_3 = 0.2 // f_3 is not very important
  - w_4 = 0.1 // f_4 is really not important

MiniMax with Fixed Ply Depth



Leaf nodes show the actual heuristic value e(n)

MiniMax with Fixed Ply Depth

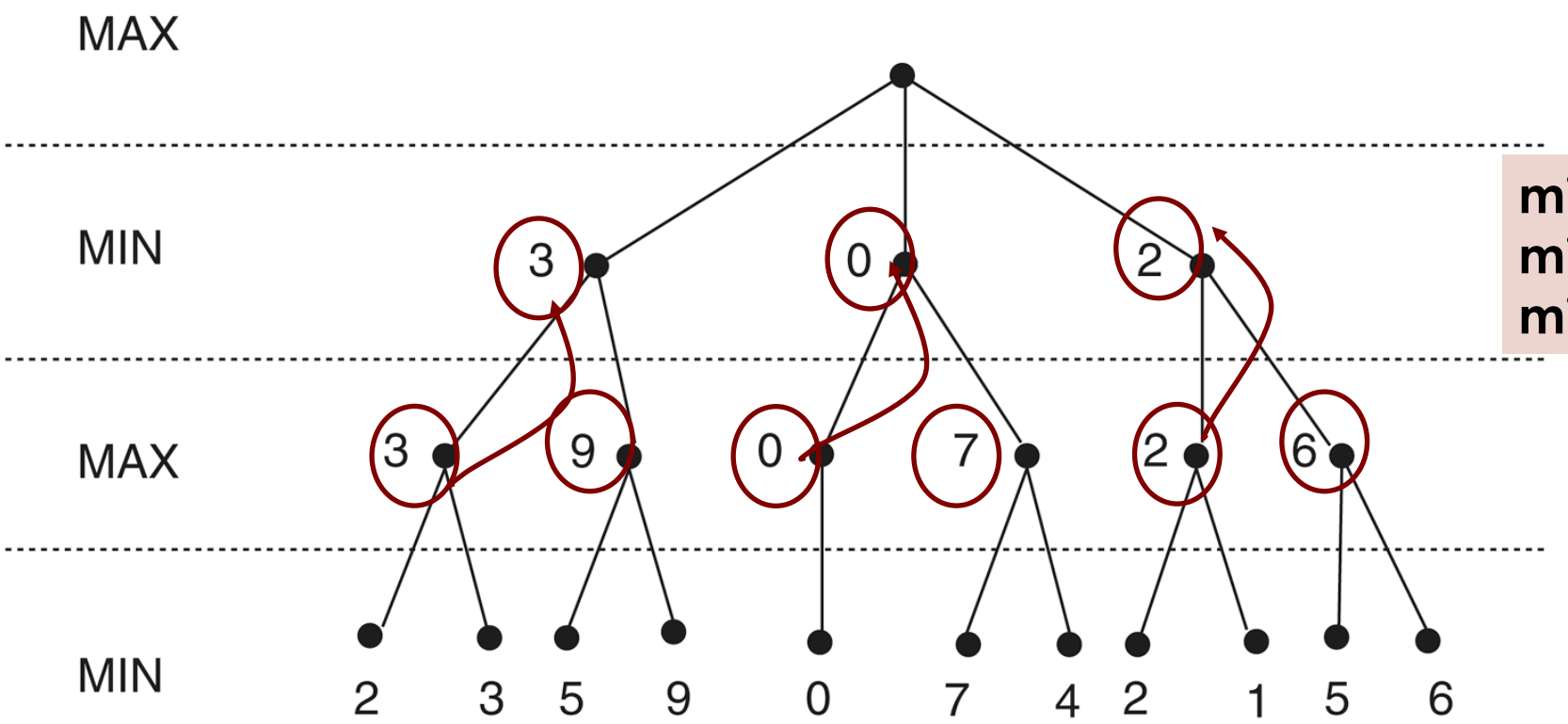


max(2,3) = 3
max(5,9) = 9
….

Leaf nodes show the actual heuristic value e(n)
Internal nodes show back-up heuristic value

source: G. Luger (2005)

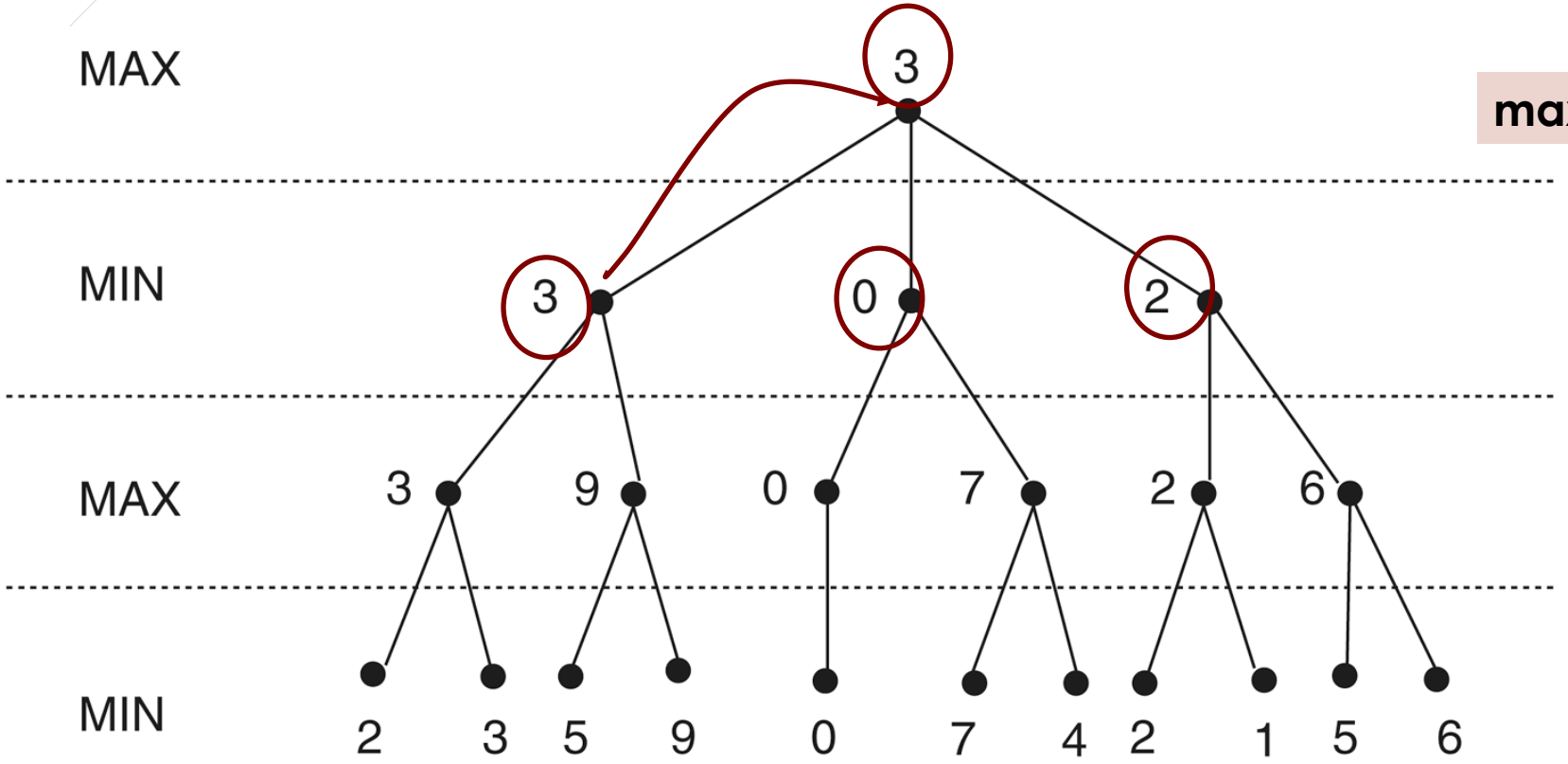
MiniMax with Fixed Ply Depth



min(3,9) = 3
min(0,7) = 9
min(2,6) = 2

Leaf nodes show the actual heuristic value e(n)
Internal nodes show back-up heuristic value

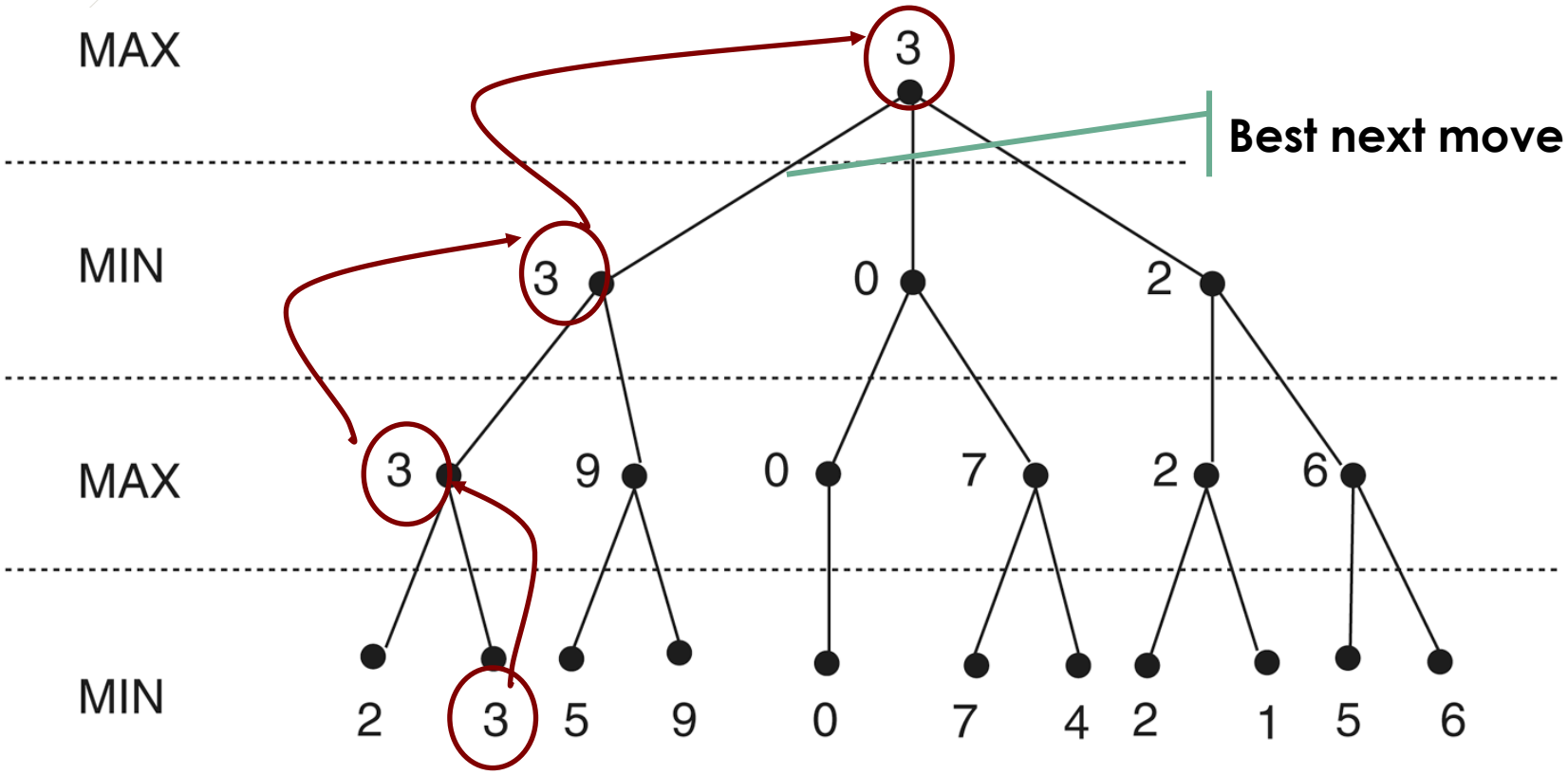
MiniMax with Fixed Ply Depth



max(3, 0, 2) = 3

Leaf nodes show the actual heuristic value e(n)
Internal nodes show back-up heuristic value

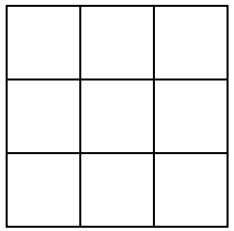
MiniMax with Fixed Ply Depth



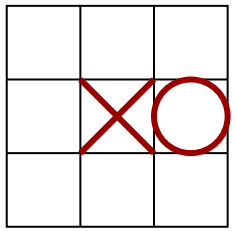
Leaf nodes show the actual heuristic value e(n)
Internal nodes show back-up heuristic value

Example: e(n) for Tic–Tac–Toe

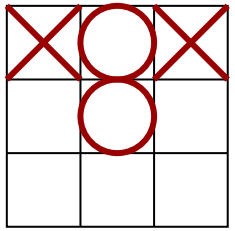
- Possible e(n)
  - e(n) = number of rows, columns, and diagonals open for MAX
    - number of rows, columns, and diagonals open for MIN
  - e(n) = + ∞, if n is a forced win for MAX
  - e(n) = - ∞, if n is a forced win for MIN



e(n) = 8–8 = 0

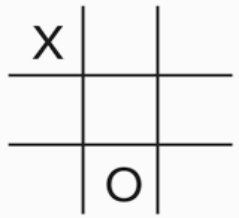


e(n) = 6–4 = 2

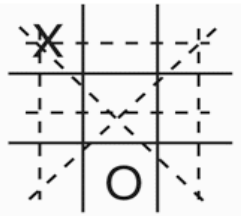


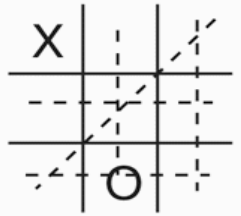
e(n) = 3–3 = 0

More Examples



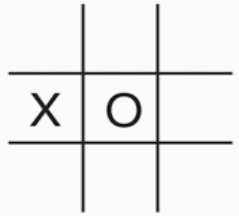
X has 6 possible win paths:





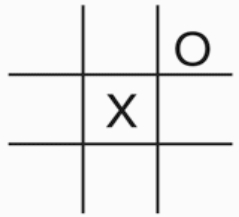
O has 5 possible wins:

$E(n) = 6 - 5 = 1$



X has 4 possible win paths;
O has 6 possible wins

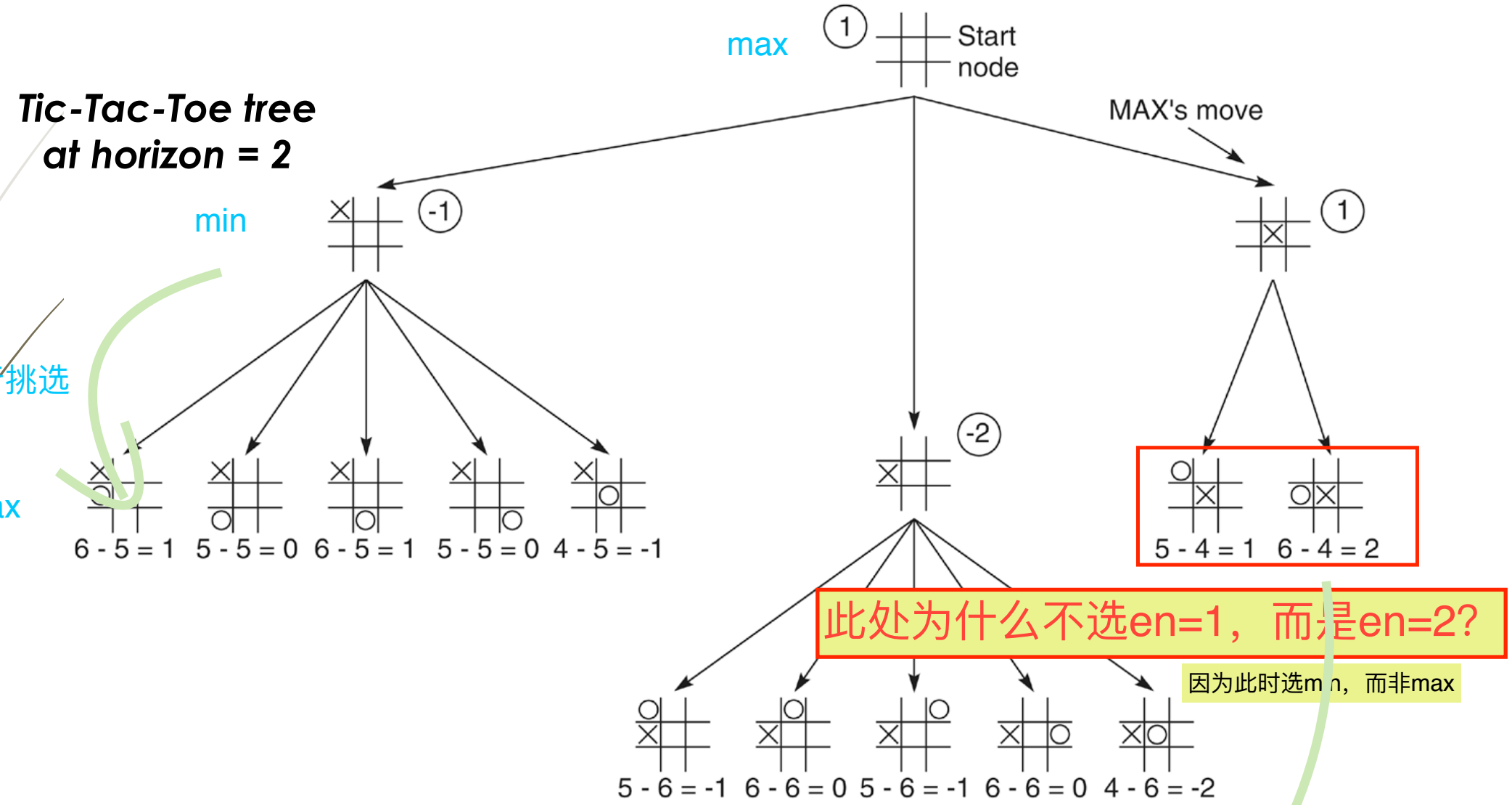
$E(n) = 4 - 6 = -2$



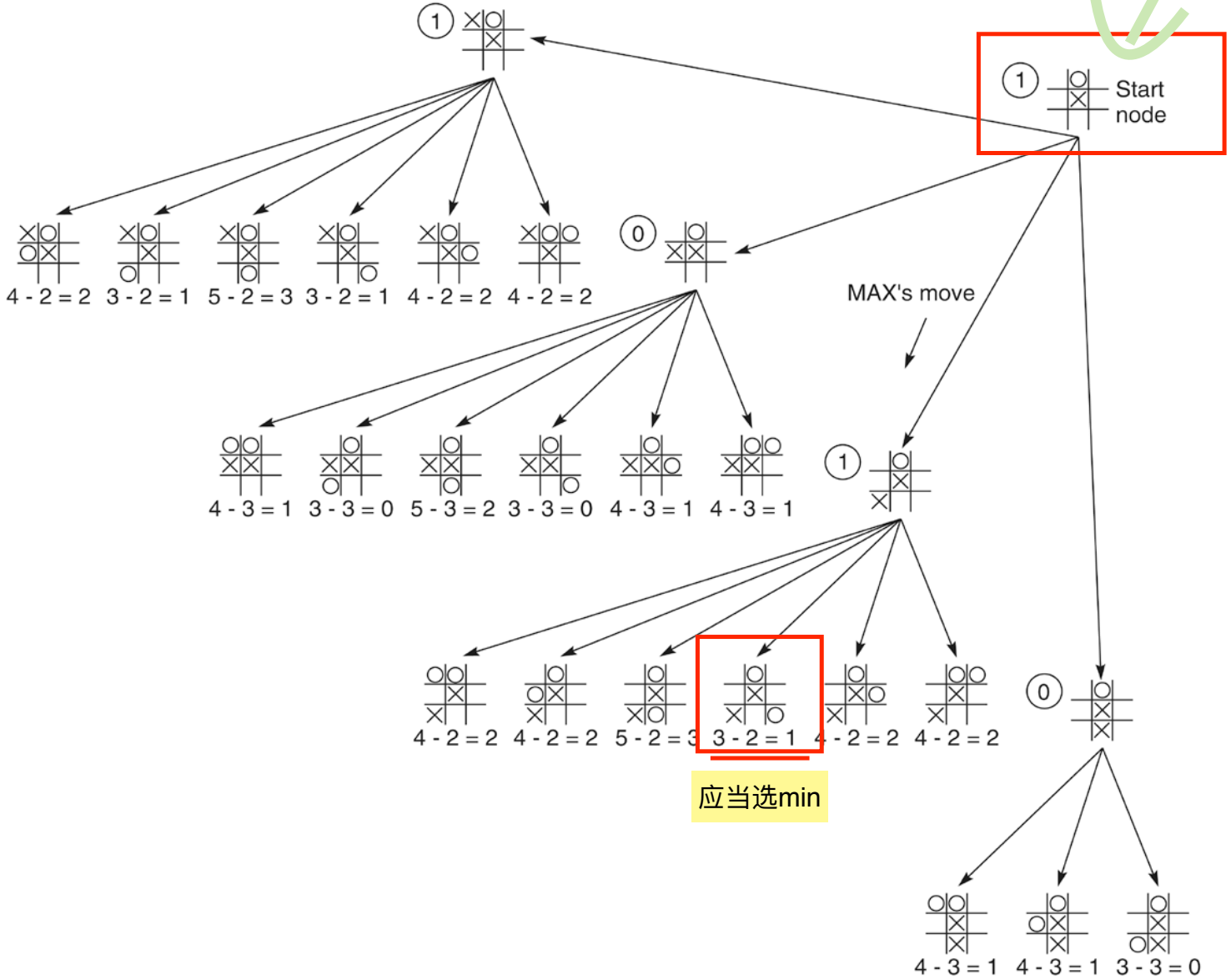
X has 5 possible win paths;
O has 4 possible wins

$E(n) = 5 - 4 = 1$

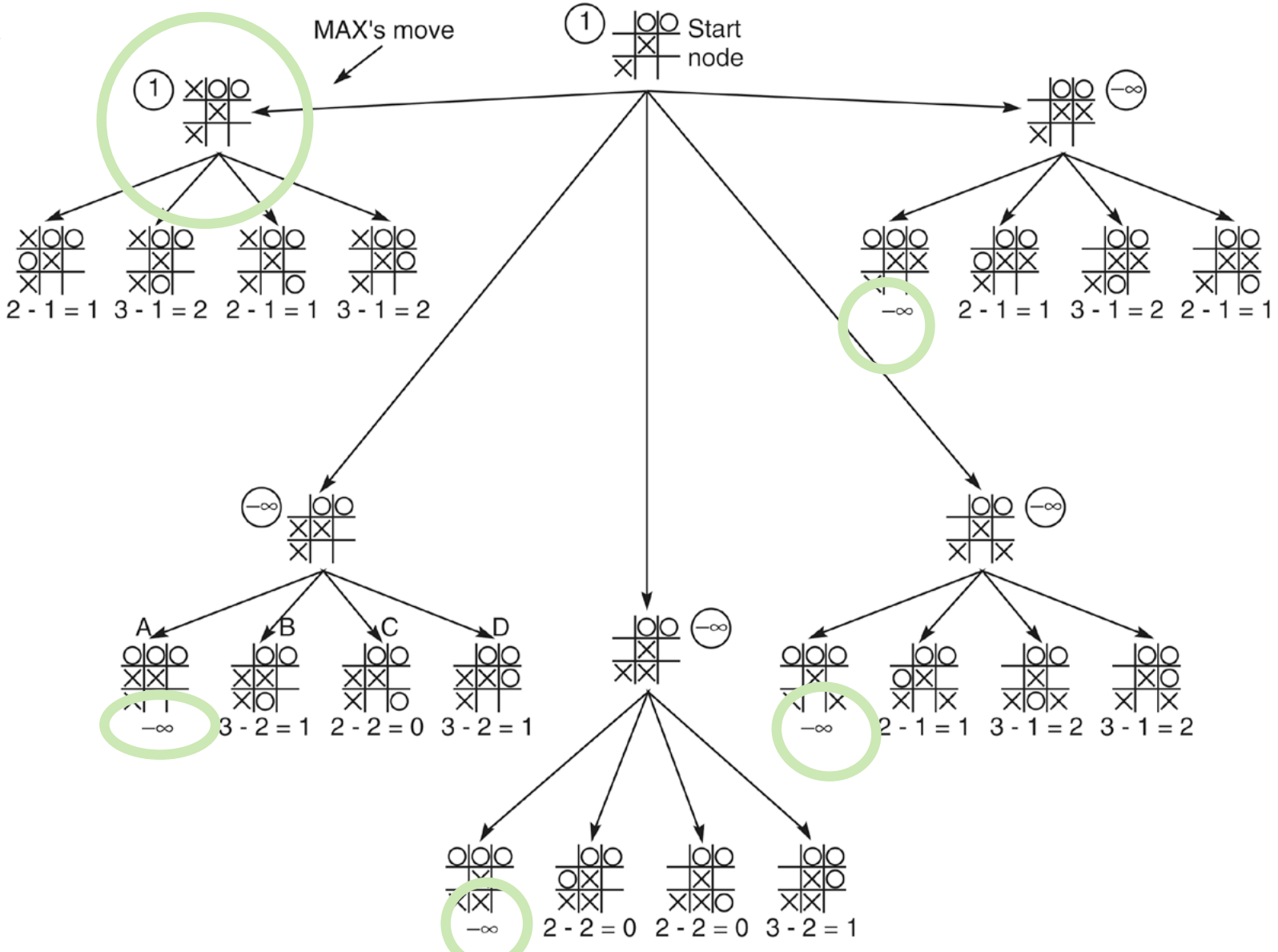
Two-ply MiniMax for Opening Move



Two-ply MiniMax: MAX's possible 2nd Moves



Two-ply MiniMax: MAX's move at end



Example: Game of Nim

- Rules
  - 2 players start with a pile of tokens
  - move: split (any) existing pile into two non-empty differently-sized piles
  - game ends when no pile can be unevenly split
  - player who cannot make his move loses

https://www.youtube.com/watch?v=niMjxNtiuu8

The End