ARTIFICIAL INTELLIGENCE

ADVERSARIAL SEARCH AND ALPHA-BETA PRUNING ALGORITHM

CHRISTIAN SY

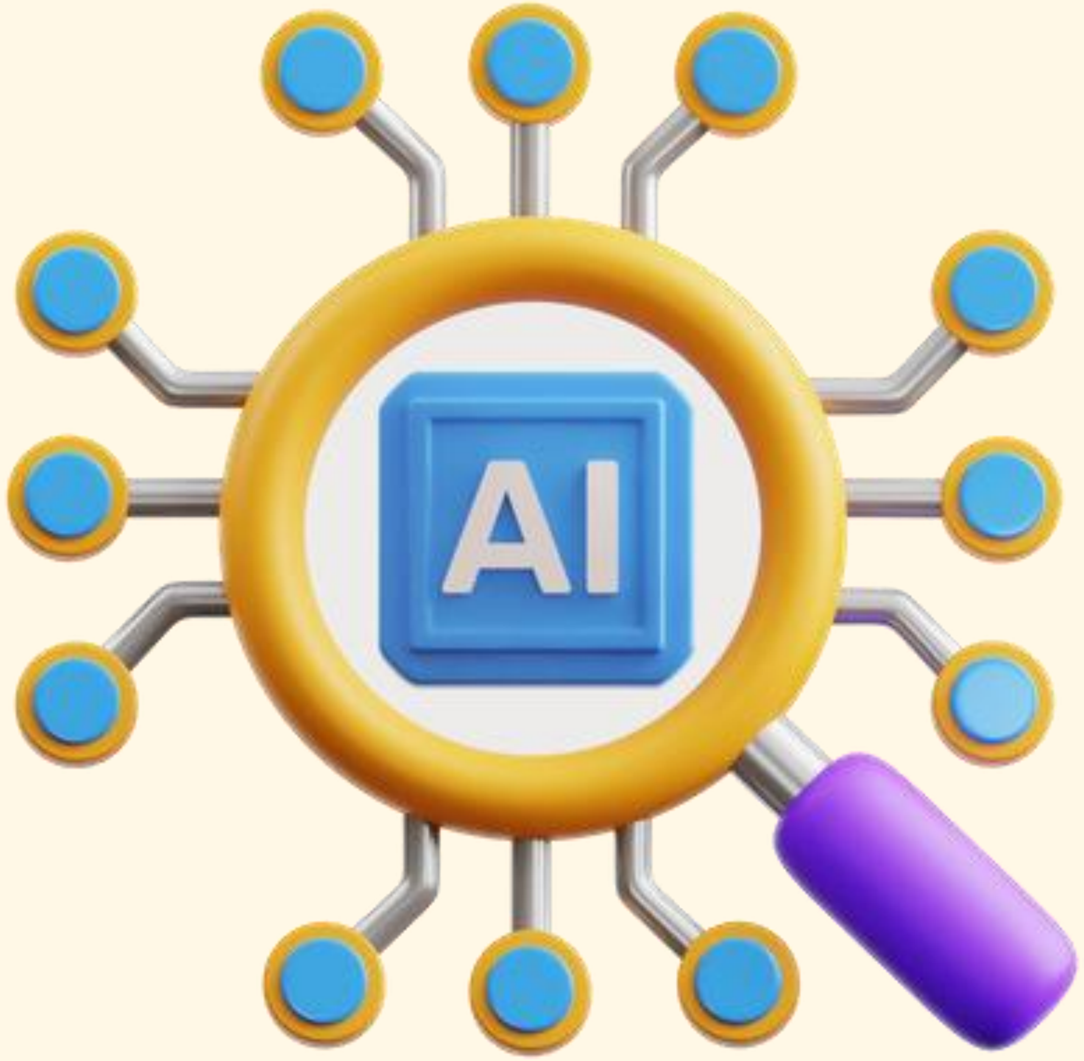
SPECIFIC LEARNING OUTCOMES

By the end of this topic, students will be able to:

1. **Explain** the concept of adversarial search and the role of the Minimax algorithm in two-player games.
2. **Differentiate** between the strategies of MAX (the maximizing player) and MIN (the minimizing player).
3. **Apply** the Minimax formula to compute values of possible moves in Tic-Tac-Toe.
4. **Use** a heuristic evaluation function to estimate the value of non-terminal game states.
5. **Analyze and justify** the selection of optimal moves using backup values in a game tree.

PART I
ADVERSARIAL SEARCH

SEARCH VERSUS GAMES



SEARCH – NO ADVERSARY

- Solution is (heuristic) method for finding goal
- Heuristics and (Constraint Satisfaction Problem) CSP techniques can find an *optimal* solution
- Evaluation function: estimate of cost from start to goal through the given node
- Examples: path planning, scheduling activities

GAMES – ADVERSARY

- Solution is strategy
  - strategy specifies move for every possible opponent reply.
- Time limits force an *approximate* solution
- Evaluation function: evaluate “goodness” of game position
- Examples: tic-tac-toe, chess, checkers, Othello, backgammon

THE ESSENCE OF ADVERSARIAL SEARCH

1. Search in a Competitive Environment

- Unlike classical search (e.g., pathfinding), adversarial search happens when **multiple agents interact**.
- These agents are **competitors**, not collaborators.

Example: In chess, your move and the opponent's move both affect the outcome.

2. Goals in Conflict

- One agent's **gain is often another agent's loss**.
- This is known as a **zero-sum game** (e.g., if you win in chess, the opponent loses).

THE ESSENCE OF ADVERSARIAL SEARCH

3. Games as Ideal Examples

Board games (Chess, Checkers, Tic-Tac-Toe) and strategy games are **classic testbeds**.

Why? Because:

- The environment is **fully observable** (both players see the same board).
- Rules are **well-defined**.
- Outcomes (win, lose, draw) are **clear and measurable**.

4. States Are Easy to Represent

- Each **game position** can be represented as a *state*.
- Example: In chess, a state = board layout + whose turn it is.
- This makes adversarial search a good domain for testing algorithms.

THE ESSENCE OF ADVERSARIAL SEARCH

5. Restricted Number of Actions

In most games, players have a **finite and manageable set of moves** at each turn.

Example:

- Chess: ~35 legal moves on average.
- Tic-Tac-Toe: at most 9 moves initially.

This bounded branching factor helps define the search tree.

6. Outcomes Defined by Precise Rules

The effect of every action is **deterministic and rule-based**.

Example:

- “Knight moves in an L-shape” (chess).
- “Mark an empty square” (tic-tac-toe).

This removes uncertainty — the challenge is not probability, but **opponent strategy**.

THE ESSENCE OF ADVERSARIAL SEARCH

7. Usually Too Hard to Solve

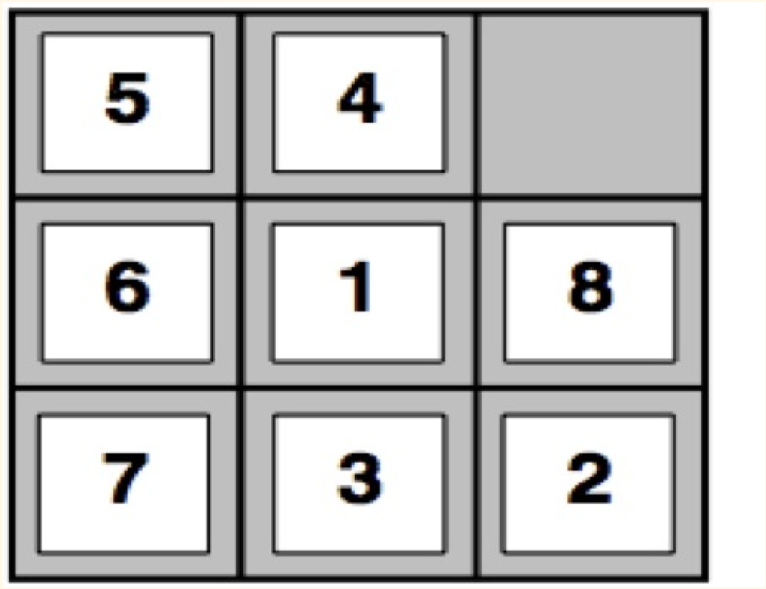
Even though states and rules are clear, the **state space explodes**.

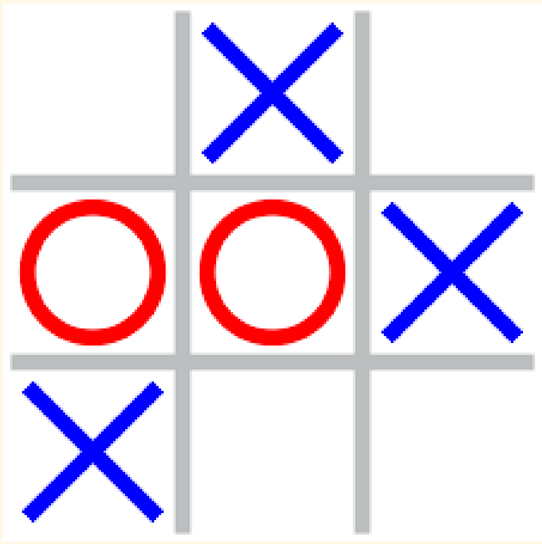
Example:

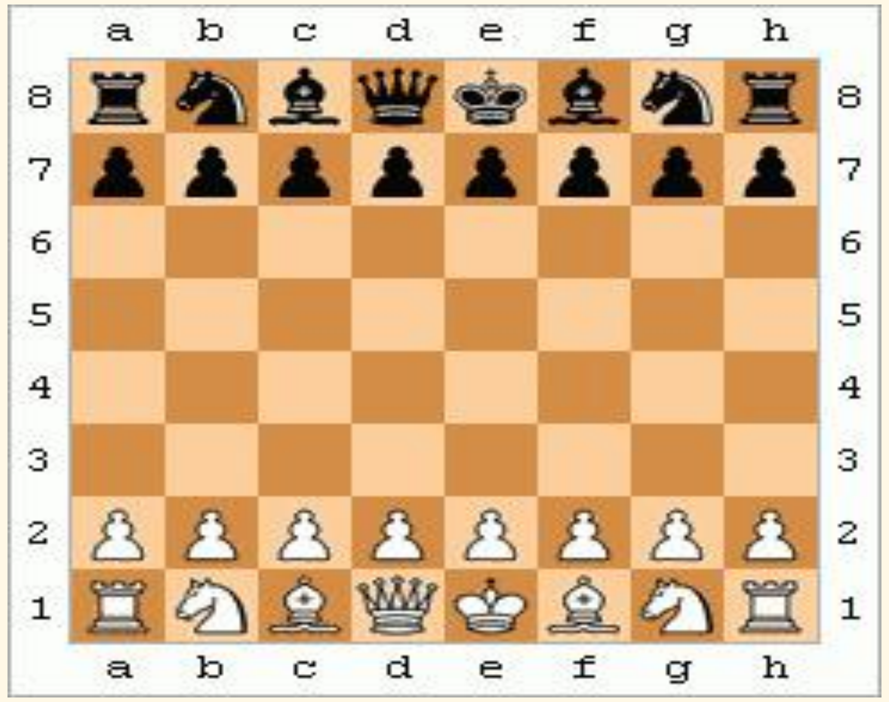
Tic-Tac-Toe: ~26,830 states (solvable with minimax).

Chess: ~10^120 possible games (unsolvable with brute force).

So we need smarter algorithms like **Minimax with Alpha–Beta Pruning**, **heuristics**, or **machine learning** (as in AlphaGo).



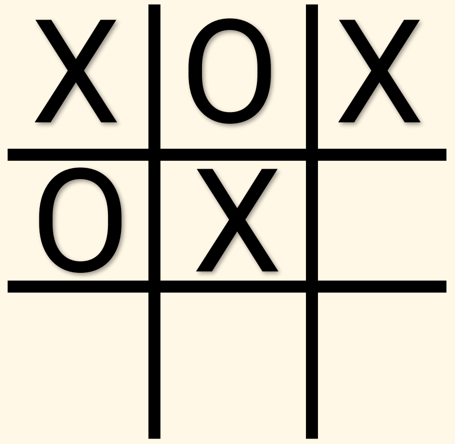




10^120 is an unimaginably huge number. It represents the estimated total number of ways a game of chess could unfold — far too many for any computer to ever calculate exhaustively.

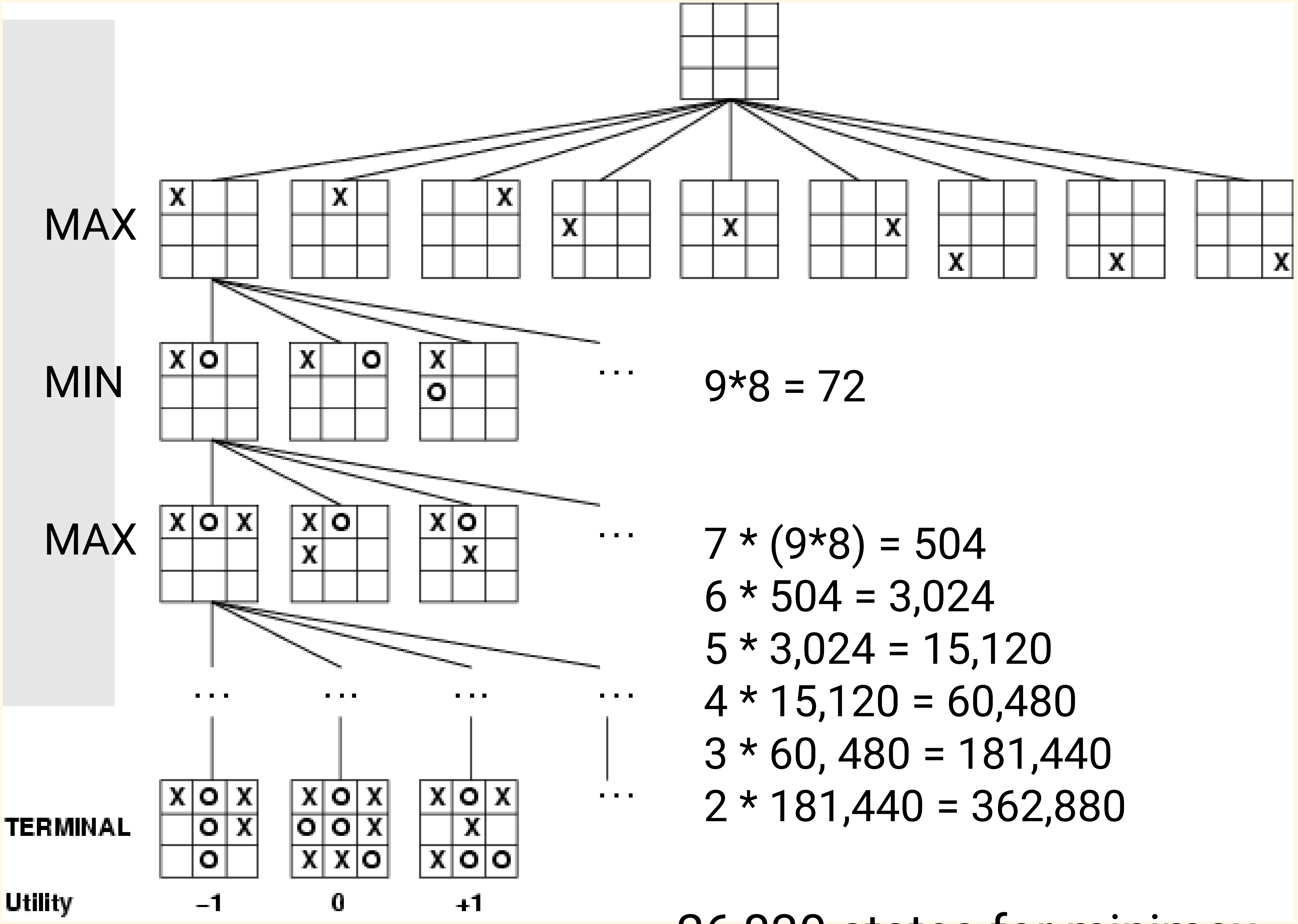
HOW TO FORMALLY DEFINE A GAME (TIC-TAC-TOE)?

- Consider a game with two players: MAX and MIN
  - MAX moves first (places X)
- A Game can be formally defined as a search problem with the following elements:
  - S^0 – the initial state
  - Player (s) – defines which player has the move in a state (s)
  - Actions(s) – returns the set of legal moves in a state
  - Results (s, a) – the transition model which defines the result of a move
  - Terminal-Test (s) – True when game is over, false otherwise
  - Utility (s, p)
    - A utility function (also objective function or payoff function)
    - Defines the final numeric value for a game that ends in a terminal state s for player p – tic-tac-toe: win = +1, loss = -1, draw = 0
  - MAX uses search tree to determine the next move.



GAME TREE FOR TIC TAC TOE

- MAX has 9 possible moves from the initial state
- Play alternates between MAX's placing an X and MIN's placing an O
- The number of each leaf node indicates the utility value of the terminal state from the point of view of MAX
  - High values are assumed to be good for MAX and bad for MIN



9*8 = 72

7 * (9*8) = 504
6 * 504 = 3,024
5 * 3,024 = 15,120
4 * 15,120 = 60,480
3 * 60, 480 = 181,440
2 * 181,440 = 362,880

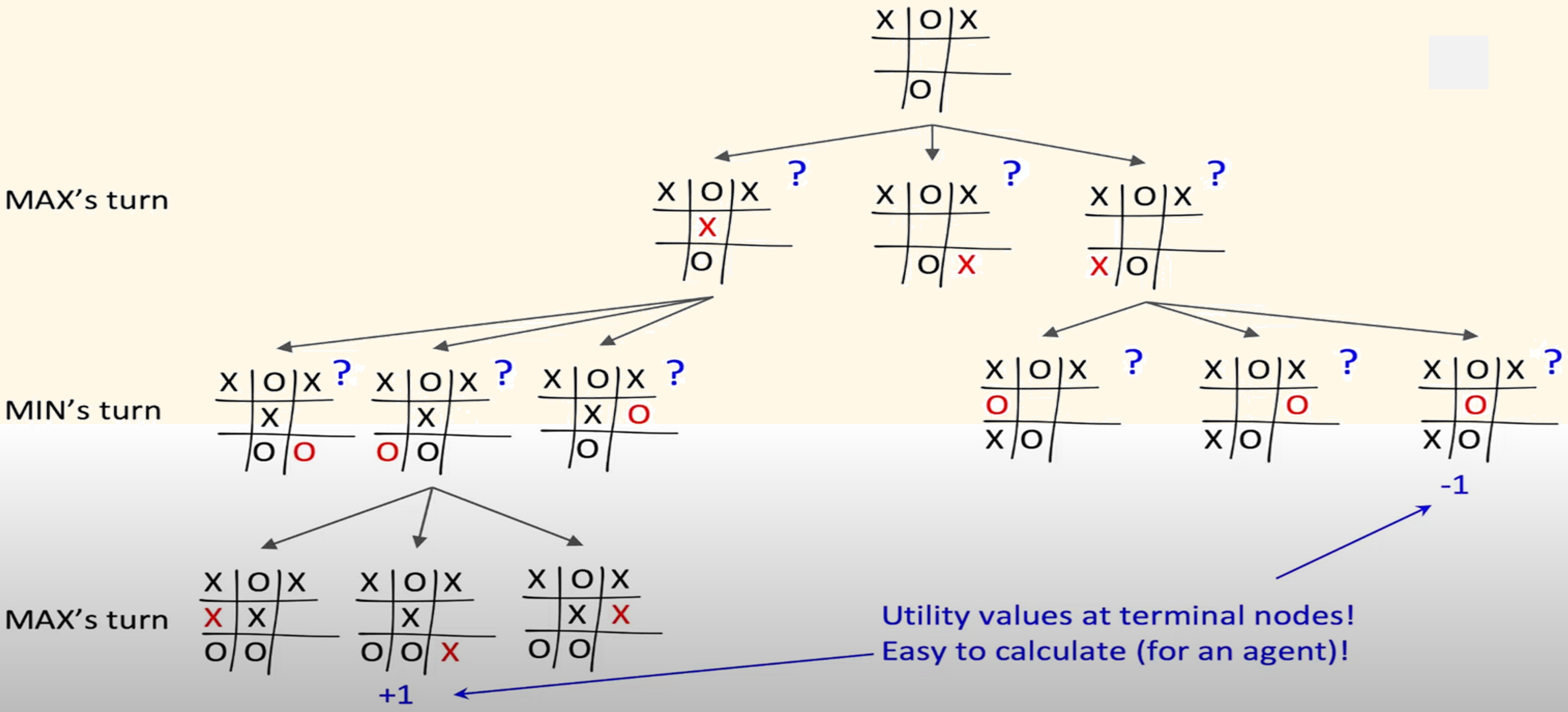
~26,830 states for minimax

AN OPTIMAL PROCEDURE: THE MIN-MAX METHOD

Designed to find the optimal strategy for Max and find the best move:

1. Generate the whole game tree, down to the leaves.
2. Apply the utility (payoff) function to each leaf.
3. Back-up values from leaves through branch nodes:
    1. a Max node computes the Max of its child values
    2. a Min node computes the Min of its child values
4. At root: choose the move leading to the child of highest value.

HOW TO CALCULATE UTILITY OF NON-TERMINAL NODES?



THE MIN-MAX FORMULA

Players:

- **MAX** (wants to maximize utility: +1 for win, 0 for draw, −1 for loss).
- **MIN** (opponent, wants to minimize utility: makes moves that reduce MAX's advantage).

Process:

- Start from the **current board state**.
- **Enumerate all possible legal moves**.
- For each move, simulate what happens after both players play optimally until the game ends.
- Assign a **utility value** to terminal states (e.g., +1, 0, −1).
- **Propagate values back up the tree**:
  - At MAX's turn → choose the **maximum** value among child states.
  - At MIN's turn → choose the **minimum** value among child states.
- The root node value = best achievable outcome for MAX assuming perfect play.

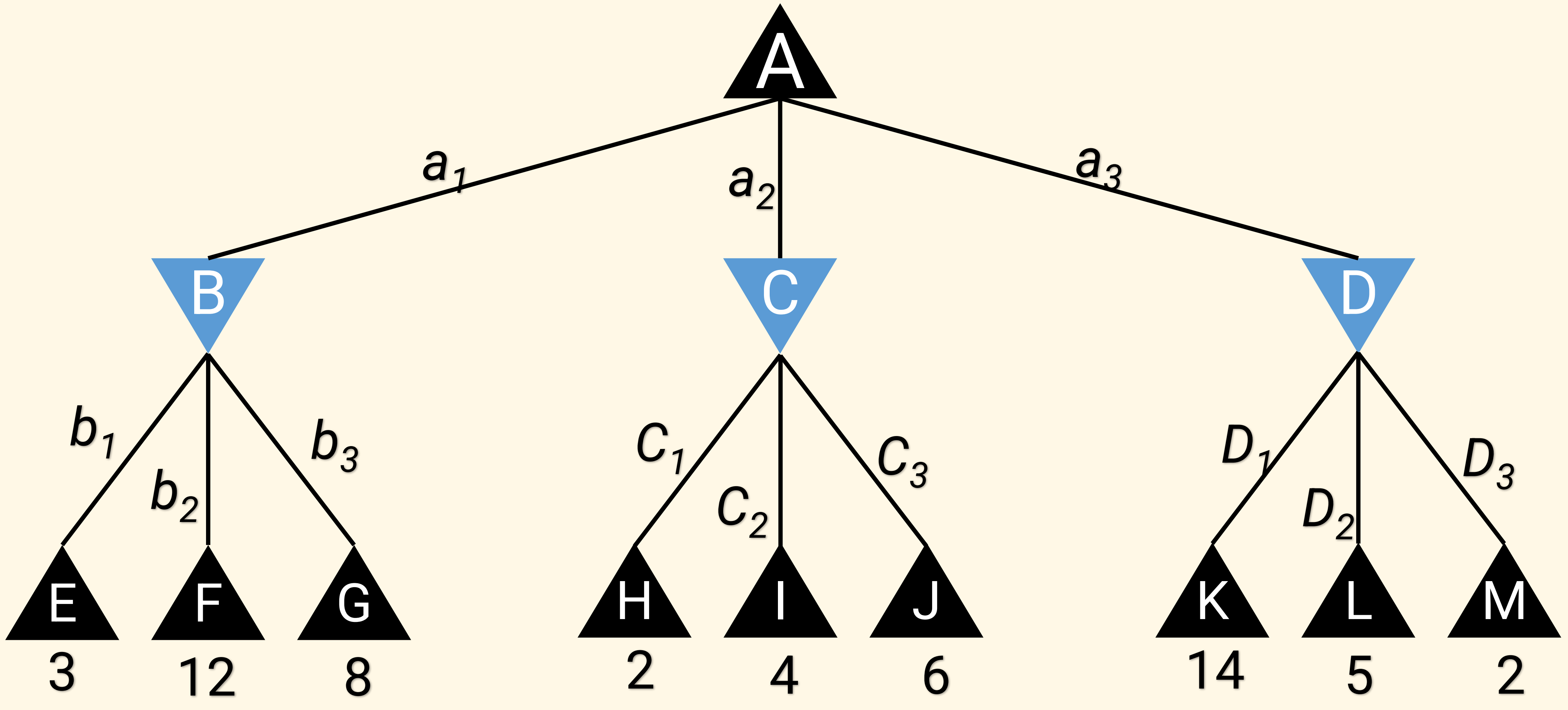
$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max\limits_{a \in Actions(s)} \text{Minimax}(\text{Result}(s, a)), & \text{if Player(s)} = \text{MAX} \\ \min\limits_{a \in Actions(s)} \text{Minimax}(\text{Result}(s, a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$

HOW TO CALCULATE MINIMAX VALUE

Consider a 'reduced' tic-tac-toe game (because game tree for full tic-tac-toe is too big)

- The possible moves for MAX at the root are a1, a2, a3
- Possible replies to a1 for MIN are b1, b2, b3, and so on

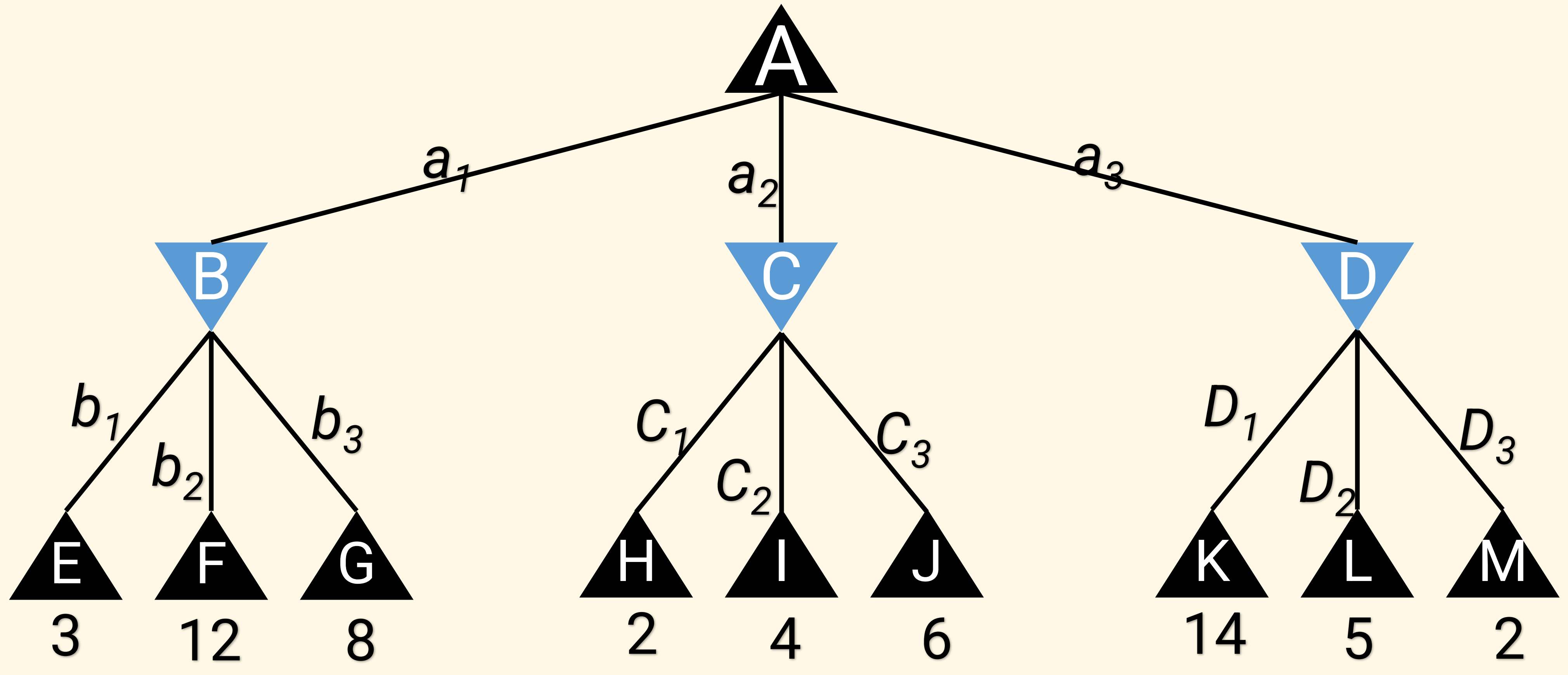
- Optimal strategy can be determined using minimax value of each node MINIMAX(n)
  - MINIMAX(n) for the user MAX is the utility of being in the corresponding state
  - So, MAX will always prefer to move to a state of maximum value



THE MIN-MAX FORMULA

Practice Problem: Calculate the minimax values at nodes A, B, C, and D for the player MAX given the game tree below. The numbers at the leaf nodes represent the values of utility (leafnode, MAX)

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MAX} \\ \min_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$



Utility Values of leaf nodes E to M

Step 1: Check if (s) is terminal – S_0 A is not a terminal.

Step 2: Check A is MAX or MIN -> if MAX Minimax(A) = **max** ((MM(B), (MM(C), (MM(D))

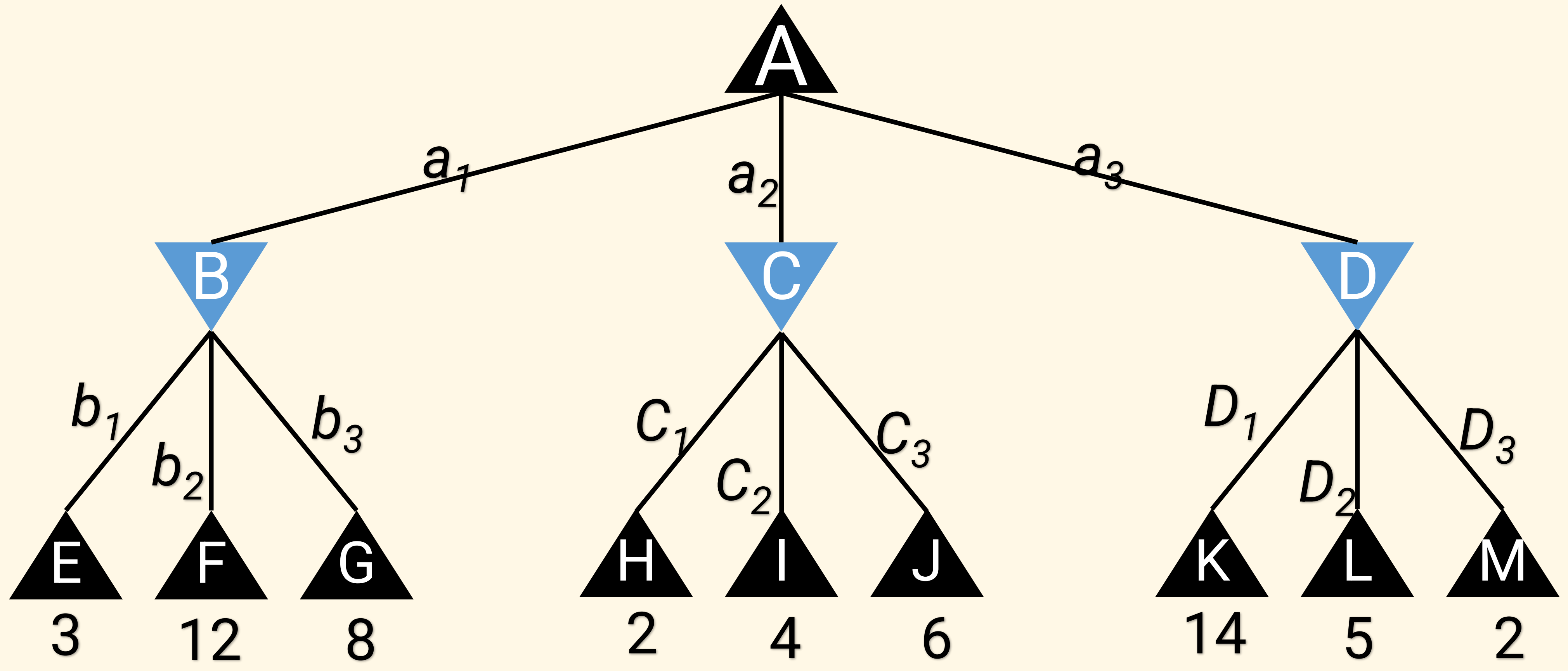
Step 3: Check if (s)-> B is terminal, if not terminal, check if MAX or **MIN** Minimax(B) = **min** ((MM(E), (MM(F), (MM(G)) where min ((MM(3), (MM(12), (MM(8)) -> Minimax(B) = 3

Step 4: Check if (s)-> C is terminal, if not terminal, check if MAX or **MIN** Minimax(C) = **min** ((MM(H), (MM(I), (MM(J)) where min ((MM(2), (MM(4), (MM(6)) -> Minimax(C) = 2

THE MIN-MAX FORMULA

Practice Problem: Calculate the minimax values at nodes A, B, C, and D for the player MAX given the game tree below. The numbers at the leaf nodes represent the values of utility (leafnode, MAX)

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MAX} \\ \min_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$



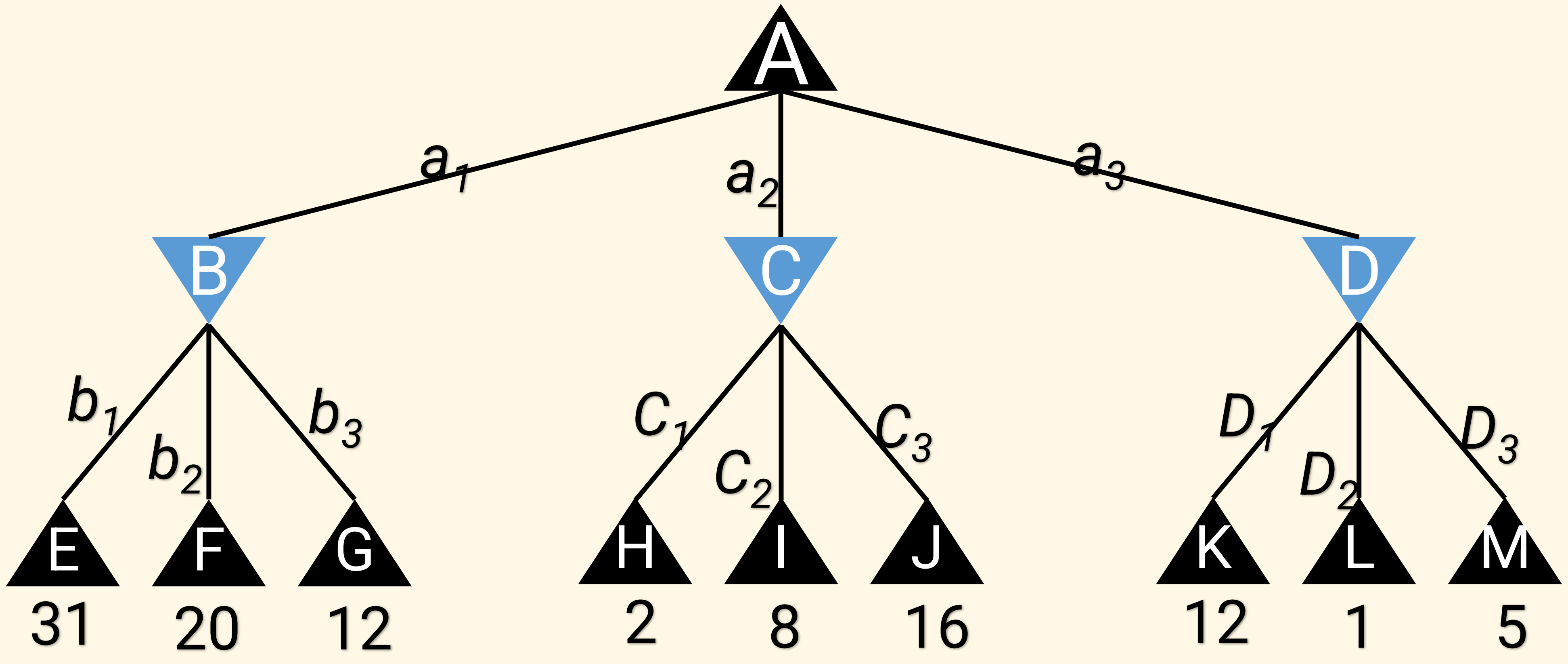
Utility Values of leaf nodes E to M

Step 5: Check if (s)-> D is terminal, if not terminal, check if MAX or **MIN**
Minimax(D) = **min** ((MM(K), (MM(L), (MM(M)) where min ((MM(14), (MM(5), (MM(2)) -> Minimax(C) = 2

Step 6: Minimax(A) = **max** ((MM(3), (MM(2), (MM(2)) -> Minimax (A) = 3

THE MIN-MAX FORMULA

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MAX} \\ \min_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$



Utility Values of leaf nodes E to M

Step 1: Check if (s) is terminal – S_0 A is not a terminal.

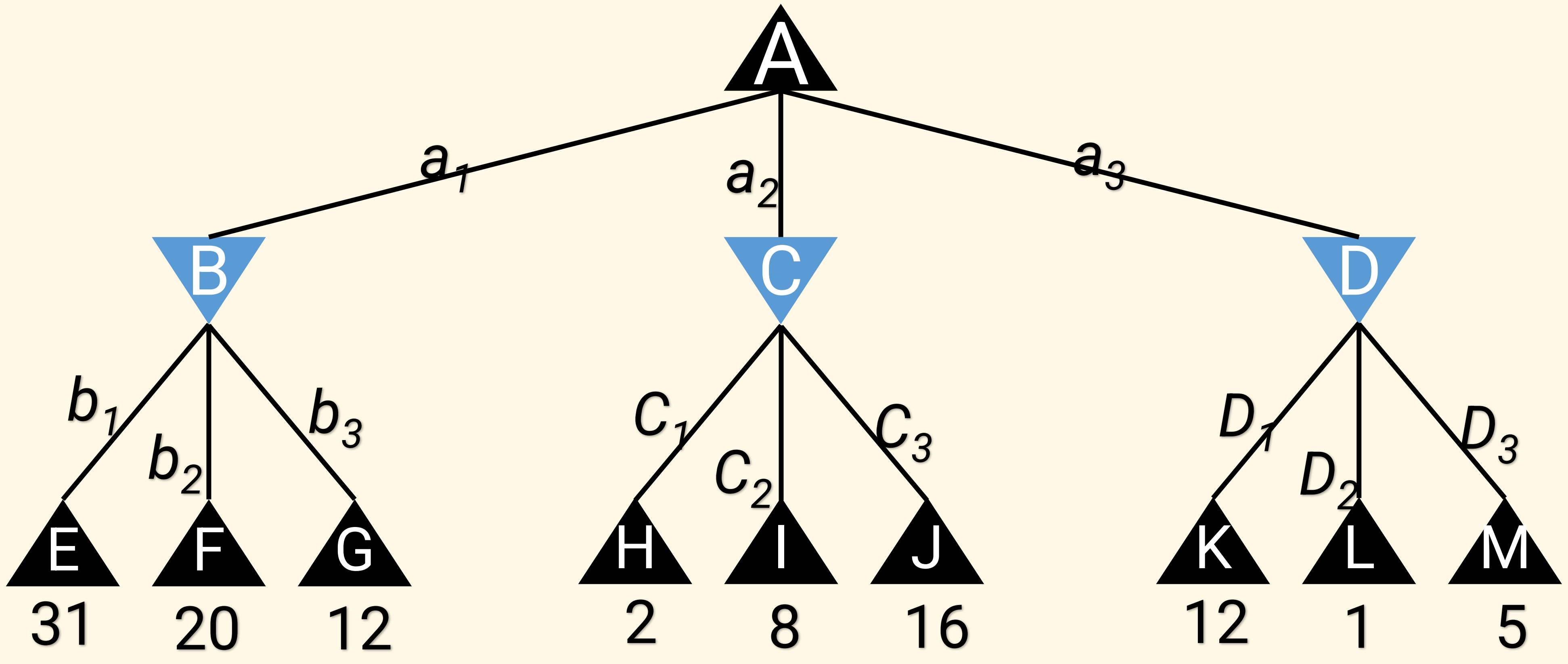
Step 2: Check A is MAX or MIN -> if MAX Minimax(A) = **max** ((MM(B), (MM(C), (MM(D))

Step 3: Check if (s)-> B is terminal, if not terminal, check if MAX or **MIN** Minimax(B) = **min** ((MM(E), (MM(F), (MM(G)) where min ((MM(31), (MM(20), (MM(12)) -> Minimax(B) = 12

Step 4: Check if (s)-> C is terminal, if not terminal, check if MAX or **MIN** Minimax(C) = **min** ((MM(H), (MM(I), (MM(J)) where min ((MM(2), (MM(8), (MM(16)) -> Minimax(C) = 2

THE MIN-MAX FORMULA

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MAX} \\ \min_{a \in Actions(s)} \text{Minimax}(\text{Result}(s,a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$



Utility Values of leaf nodes E to M

Step 5: Check if (s)-> C is terminal, if not terminal, check if MAX or **MIN**
Minimax(D) = **min** ((MM(K), (MM(L), (MM(M)) where min ((MM(12), (MM(1), (MM(5)) -> Minimax(C) = 1

Step 6: Minimax(A) = **max** ((MM(12), (MM(2), (MM(1)) -> Minimax (A) = 12

UTILITY FUNCTION VS MINIMAX FUNCTION

- ✓ **Utility (s, p)** – defines the final numeric value for a game that ends in a terminal state s for player p. Terminal
- ✓ **Minimax (s, p)** – defines the numeric values at all nodes. Non-Terminal and Terminal

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax}(\text{Result}(s, a)), & \text{if Player(s)} = \text{MAX} \\ \min_{a \in Actions(s)} \text{Minimax}(\text{Result}(s, a)), & \text{if Player(s)} = \text{MIN} \end{cases}$$

OPTIMAL DECISIONS IN GAMES

Step 1: Board State - X → **B3**

	1	2	3
A	X	O	X
B	O	X	
C			0

Coordinates:
A1 = X, A2 = O, A3 = X
B1 = O, B2 = X, B3 = *empty*
C1 = *empty*, C2 = *empty*, C3 = O
Available moves for **MAX (X)**: **B3, C1, C2**

Step 2: Define Utilities
We'll use a simple scoring function at terminal states:

- ✓ +1 = MAX (X) wins
- ✓ 0 = Draw
- ✓ −1 = MIN (O) wins

Step 3: Explore Each Move, Move 1: MAX plays at **B3**

	1	2	3
A	X	O	X
B	O	X	X
C			0

Now **O to move**; empties are **C1, C2**.
If O → **C1**, then X has only **C2** left and the final board becomes

	1	2	3
A	X	O	X
B	O	X	X
C	O	X	0

	1	2	3
A	X	O	X
B	O	X	X
C	X	O	0

which is a **draw** → utility 0.
If O → **C2**, then X plays **C1** next and completes the diagonal A3−B2−C1 → **X wins** → utility +1.
MIN (O) will choose the move that minimizes X's outcome, so O chooses **C1** and forces the draw.
Therefore: Minimax(Result(S,B3))=0

OPTIMAL DECISIONS IN GAMES

Step 1: Board State - X → **C1**

	1	2	3
A	X	O	X
B	O	X	
C			0

Coordinates:
A1 = X, A2 = O, A3 = X
B1 = O, B2 = X, B3 = *empty*
C1 = *empty*, C2 = *empty*, C3 = O
Available moves for **MAX (X)**: **B3, C1, C2**

Step 2: Define Utilities
We'll use a simple scoring function at terminal states:

- ✓ +1 = MAX (X) wins
- ✓ 0 = Draw
- ✓ −1 = MIN (O) wins

Step 3: Explore Each Move, Move 1: MAX plays at **C1**

	1	2	3
A	X	O	X
B	O	X	
C	X		0

That completes diagonal A3−B2−C1 = X,X,X → immediate win for X (terminal).

Therefore: Minimax(Result(S,C1))=+1

We stop here since the game ended

OPTIMAL DECISIONS IN GAMES

Step 1: Board State - X → C2

	1	2	3
A	X	O	X
B	O	X	
C			0

Coordinates:
A1 = X, A2 = O, A3 = X
B1 = O, B2 = X, B3 = *empty*
C1 = *empty*, C2 = *empty*, C3 = O
Available moves for **MAX (X)**: **B3, C1, C2**

Step 2: Define Utilities
We'll use a simple scoring function at terminal states:

- ✓ +1 = MAX (X) wins
- ✓ 0 = Draw
- ✓ −1 = MIN (O) wins

Step 3: Explore Each Move, Move 1: MAX plays at **C2**

	1	2	3
A	X	O	X
B	O	X	X
C	O	X	0

Now **O to move**; empties are **B3, C1**.
If O → **B3**, then X plays **C1** next and wins (A3−B2−C1) → +1.
If O → **C1**, then X plays **B3** next and the final position is a draw → 0.
MIN (O) will pick the worst for X (the minimum), i.e. **C1** → draw.
Therefore: Minimax(Result(S,C2))= 0

Root decision (X to move)
Apply the MAX rule at the root:

Minimax(S)= max{0,+1,0} = +1

So **X should play C1** (the immediate winning move).

OPTIMAL DECISIONS IN GAMES

To Summarize X Moves

	1	2	3
A	X	O	X
B	O	X	
C			0

Minimax(B3) = 0 (O can force a draw)
Minimax(C1) = +1 (immediate X win)
Minimax(C2) = 0 (O can force a draw)
Best move for X = **C1** (choose the child with highest minimax value).

(STATIC) HEURISTIC EVALUATION FUNCTIONS

- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent's score from the player's.
  - Often called "static" because it is called on a static board position.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces

- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it's -X for the opponent
  - "Zero-sum game"

(STATIC) HEURISTIC EVALUATION FUNCTIONS

- When the game tree is too large (like in chess), we do not expand all the way to terminal states.
- Instead, we stop at some depth and apply a heuristic evaluation function (h(s)) to estimate how good the board is for MAX (X).
- In **tic-tac-toe**, a simple evaluation function could be:

$$h(s)=(N2,X-N2,O)$$

Where:

$N_{2,X}$ =number of lines (rows, cols, diagonals) where X has 2 and the third is empty

$N_{2,O}$ =number of lines where O has 2 and the third is empty

👉 Intuition:

If X is about to win, the value is high (+).

If O is about to win, the value is low (–).

If the board is neutral, the value is 0.

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A	X	O	X
B	O	X	X
C			0

- Empty cells = B3, C1, C2
- It's **X's turn (MAX)**.
- Move **X** → **B3**

Check all lines:

Rows

Row A (A1,A2,A3): X, O, X → not 2X+empty.

Row B (B1,B2,B3): O, X, X → contains O, so blocked.

Row C (C1,C2,C3): _, _, O → not 2O (only 1 O).

Columns

Col1 (A1,B1,C1): X, O, _ → not 2 of same.

Col2 (A2,B2,C2): O, X, _ → not 2 of same.

Col3 (A3,B3,C3): X, X, O → blocked by O.

Diagonals

Main (A1,B2,C3): X, X, O → blocked.

Anti (A3,B2,C1): X, X, _ → **two X + empty** → **counts as 1** for X.

So: $N2,X$=1 (anti-diagonal A3−B2−C1)$N2,O$=0

Heuristic values:

$h(s)=(N2,X-N2,O)$

$$h(\mathrm{sB3}) = 1 - 0 = \mathbf{+1}$$

No immediate win — game continues; next is O's turn.

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A	X	O	X
B	O	X	
C	X		0

- Empty cells = B3, C1, C2.
- It's **X's turn (MAX)**.
- Move **X** → **C1**

Check diagonal A3–B2–C1: X, X, X → **immediate X win** (terminal).
So:
This is **terminal** → minimax value = **X wins**.
We treat this as the best possible value (e.g. +1 on normalized scale, or a large positive in other heuristics).
No need to compute N_2 for heuristic — terminal detection overrides heuristic.

Heuristic values are only applied to non-terminal instead we assign a terminal value of +1 for a winning move

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A	X	O	X
B	O	X	
C		X	0

- Empty cells = B3, C1, C2.
- It's **X's turn (MAX)**.
- Move **X** → **C2**

Check all lines:

Rows

Row A: X, O, X → blocked.
Row B: O, X, _ → not 2X.
Row C: _, X, O → not 2X or 2O.

Columns

Col1: X, O, _ → no.
Col2: O, X, X → contains O → blocked (it is O, X, X not 2X).
Col3: X, _, O → no.

Diagonals

Main (A1,B2,C3): X, X, O → blocked.
Anti (A3,B2,C1): X, X, _ → **two X + empty** → **counts as 1** for X.

Heuristic values:

$h(s)=(N2{,}X-N2{,}O)$

$$h(sC2) = 1 - 0 = +\mathbf{1}$$

No immediate win — game continues; next is O's turn.

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A	X	O	X
B	O	X	X
C			0

	1	2	3
A	X	O	X
B	O	X	
C	X		0

	1	2	3
A	X	O	X
B	O	X	
C		X	0

Heuristic evaluation of children:

- If X plays **B3**:
  - Board creates potential win along Row 2 (O|X|X).
  - h(s)=(N2,X−N2,O) - formula
  - Look for X two-in-a-row lines (2 X + 1 empty):

Diagonal A3−B2−C1: A3=X, B2=X, C1=_ → **one** (X-two-in-a-row → N2,X = 1)

O two-in-a-row lines: none → N2,O = 0.

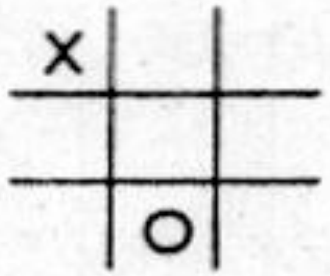
  - h(sB3)=(1-0) = +1
- If X plays **C1**:
  - Check: diagonal A3−B2−C1 = X, X, X → **immediate X win**.

So by the rule, no need to compute for h(s).

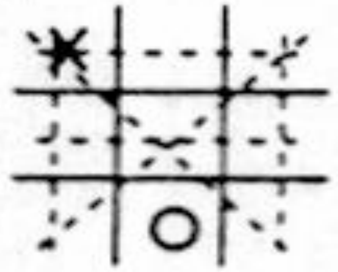
- Empty cells = B3, C1, C2.
- It's **X's turn (MAX)**.

- If X plays **C2**:
  - Count two-in-a-row:
  - X two-in-a-row: diagonal A3−B2−C1 = X, X, _ → N2,X = 1 (C1 empty).
  - O had no 2-in-a-row threat there
  - So N2,X = 1 and N2,O = 0
  - h(sC1)=(1-0) = +1

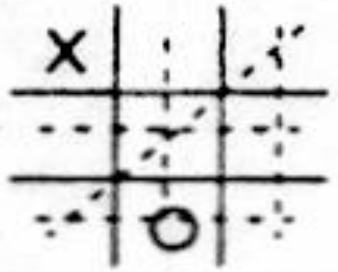
(STATIC) HEURISTIC EVALUATION FUNCTIONS



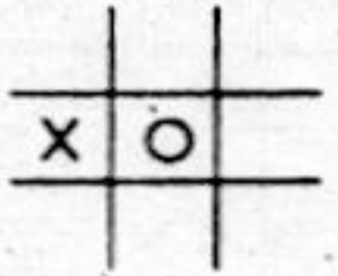
X has 6 possible win paths:



O has 5 possible wins:



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



X has 4 possible win path

O has 6 possible wins

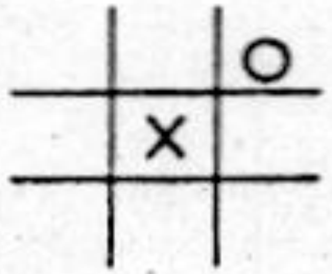
$E(n) = 4 - 6 = -2$

Heuristic is $E(n) = M(n) - O(n)$

where $M(n)$ is the total of My possible winning lines

$O(n)$ is total of Opponent's possible winning lines

$E(n)$ is the total Evaluation for state n



X has 5 possible win paths:

O has 4 possible wins

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

BACKUP VALUES

- In Minimax with heuristics, the heuristic value is assigned at leaf nodes (where you stop search).
- Then, these values are backed up through the Minimax formula:

$$\text{Minimax}(s) = \begin{cases} \max_a \ \text{Minimax}(\text{Result}(s,a)) & \text{if player} = \text{MAX (X)} \\ \min_a \ \text{Minimax}(\text{Result}(s,a)) & \text{if player} = \text{MIN (O)} \end{cases}$$

Backup means:

✓ Leaf nodes get heuristic values (h(s)).
✓ Their parent nodes take min/max depending on whose turn it is.
✓ This process continues until the root.

BACKUP VALUES

Board before X moves (X to move), children states after X plays:

$s_{B3}$with heuristic $h(s_{B3}) = +1$) leaf value)

$s_{C1}$is **terminal win** $\longrightarrow$ value $= +1$

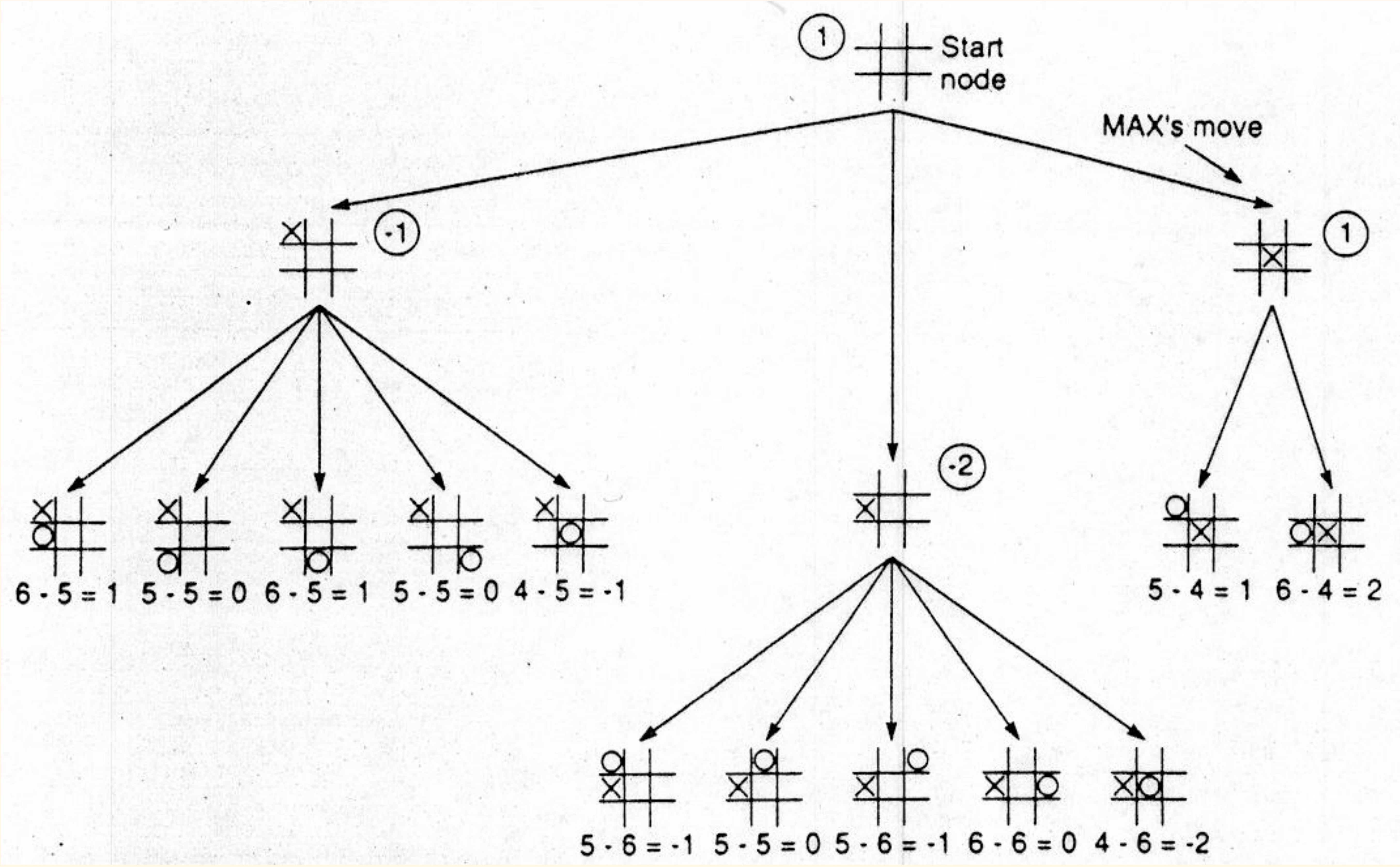
$s_{C2}$with heuristic $h(s_{C2}) = +1$) leaf value)

- Here the **root is a MAX node** (X to move). The leaves are already the backed values (we treated them as leaves by using heuristics or terminal utility). So we back up:
- Interpretation: from the root X can guarantee outcome **+1** (a win). In practice X picks **C1** because it wins immediately — terminal children are preferred when equal values, but numerically equal leaf values yield the same max.
- Backup Value = +1

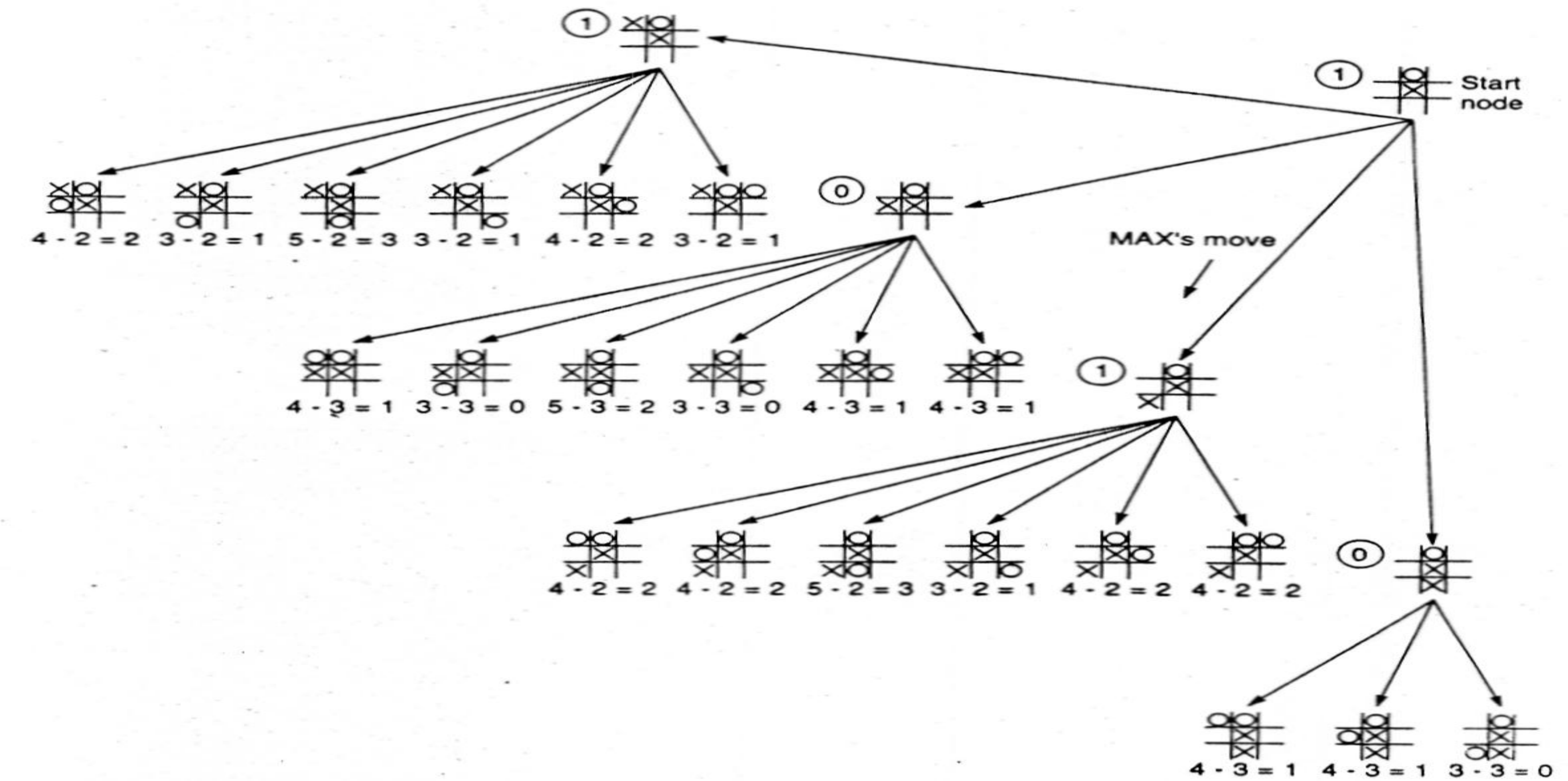
$$\text{value}(n) = \begin{cases} \text{Utility}(n) & \text{if } n \text{ is terminal} \\ \max_{c \in Children(n)} \text{value}(c) & \text{if } n \text{ is a MAX node} \\ \min_{c \in Children(n)} \text{value}(c) & \text{if } n \text{ is a MIN node} \end{cases}$$

```
        (MAX)
       /     \
   (MIN)      (MIN)
   /   \      /   \
 +1     0   -1    +1
```

BACKUP VALUES

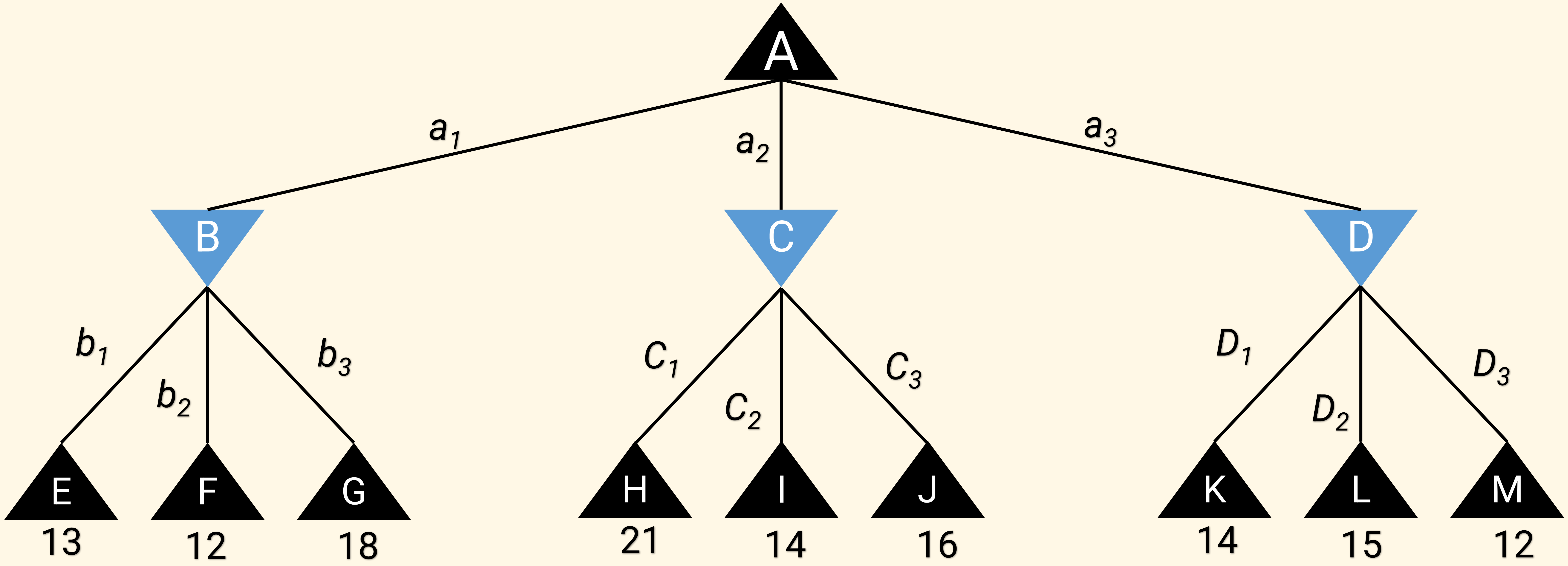


BACKUP VALUES



POST-ACTIVITY FOR ADVERSARIAL SEARCH

1. Calculate the minimax values at nodes A, B, C, and D for the player MAX given the game tree below. The numbers at the leaf nodes represent the values of utility (leafnode, MAX). Show your solutions.



Utility Values of leaf nodes E to M

POST-ACTIVITY FOR ADVERSARIAL SEARCH

2. Using the MINIMAX function, list all possible moves for **X**. Compute Minimax(S) for each available move. Show which move has the best backup value for MAX.

Board State

	1	2	3
A		O	
B		0	X
C	O	X	X

Possible Moves for X: A1, A3, B1

	1	2	3
A	X	O	
B		0	X
C	O	X	X

X moves to A1

Now **O to move**; empties are **A3, B1**

If O → **A3**, then O wins (terminal) -> -1

	1	2	3
A	X	O	O
B		O	X
C	O	X	X

If O → **B1**, then X plays **A3,** resulting in X winning → utility +1.

	1	2	3
A	X	O	X
B	O	O	X
C	O	X	X

MIN (O) will choose the move that minimizes X's outcome, or to win, so O chooses **A3** and wins.

Therefore:

Minimax(Result(S,A1))= -1

POST-ACTIVITY FOR ADVERSARIAL SEARCH

2. Using the MINIMAX function, list all possible moves for **X**. Compute Minimax(S) for each available move. Show which move has the best backup value for MAX.

Board State

	1	2	3
A		O	
B		0	X
C	O	X	X

Possible Moves for X: A1, A3, B1

	1	2	3
A		O	X
B		0	X
C	O	X	X

X moves to A3, then X wins (terminal) -> +1

Therefore: Minimax(Result(S,A3))= +1

POST-ACTIVITY FOR ADVERSARIAL SEARCH

2. Using the MINIMAX function, list all possible moves for **X**. Compute Minimax(S) for each available move. Show which move has the best backup value for MAX.

Board State

	1	2	3
A		O	
B		0	X
C	O	X	X

Possible Moves for X: A1, A3, B1

	1	2	3
A		O	
B	X	0	X
C	O	X	X

X moves to B1

Now **O to move**; empties are **A1, A3**

If O → **A1**, then X moves to A3 where X wins (Terminal) -> +1

	1	2	3
A	O	O	X
B	X	O	X
C	O	X	X

If O → **A3**, then O wins (terminal) -> -1.

	1	2	3
A	X	O	O
B	X	O	X
C	O	X	X

MIN (O) will choose A3, the move that minimizes X's outcome or O wins, so O chooses A3 and wins (terminal) -1

Therefore: Minimax(Result(S,B1))= -1

Root decision (X to move)

Apply the MAX rule at the root:

Minimax(S)= max{-1,+1,-1} = +1

So **X should play A3** (the immediate winning move).

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A	X	O	
B		0	X
C	O	X	X

- Empty cells = A1, A3, B1
- It's **X's turn (MAX)**.
- Move **X** → **A1**

Check all lines:

Rows

Row A (A1,A2,A3): X, O, _ → not 2X+empty.
Row B (B1,B2,B3): _, O, X → not 2X+empty.
Row C (C1,C2,C3): O, X, X → blocked by O.

Columns

Col1 (A1,B1,C1): X, _, O → not 2 of same.
Col2 (A2,B2,C2): O, O, X → blocked by X.
Col3 (A3,B3,C3): _, X, X → two X plus empty

Diagonals

Main (A1,B2,C3): X, O, X → blocked.
Anti (A3,B2,C1): O, O, _ → **two O + empty** → **counts as 1** for O.

So: $N2,X$=1 (anti-diagonal A3−B2−C1)$N2,O$=0

Heuristic values:

$h(s)=(N2,X-N2,O)$

$$h(\mathrm{sB3}) = 1 - 1 = 0$$

No immediate win — game continues; next is O's turn.

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A		O	X
B		0	X
C	O	X	X

- Empty cells = A1, A3, B1
- It's **X's turn (MAX)**.
- Move **X** → **A1**

Check Column 3: A3–B3–C3: X, X, X → **immediate X win** (terminal).
So:
This is **terminal** → minimax value = **X wins**.
We treat this as the best possible value (e.g. +1 on normalized scale, or a large positive in other heuristics).
No need to compute N_2 for heuristic — terminal detection overrides heuristic.

(STATIC) HEURISTIC EVALUATION FUNCTIONS

	1	2	3
A		O	
B	X	0	X
C	O	X	X

- Empty cells = A1, A3, B1
- It's **X's turn (MAX)**.
- Move **X** → **A1**

Check all lines:

Rows

Row A (A1,A2,A3): _, O, _ → not 2X+empty.

Row B (B1,B2,B3): X, O, X → not 2X+empty.

Row C (C1,C2,C3): O, X, X → blocked by O.

Columns

Col1 (A1,B1,C1): _, X, O → not 2 of same.

Col2 (A2,B2,C2): O, O, X → blocked by X.

Col3 (A3,B3,C3): _, X, X → two X plus empty

Diagonals

Main (A1,B2,C3): _, O, X → not 2 of same.

Anti (A3,B2,C1): O, O, _ → **two O + empty** → **counts as 1** for O.

So: $N2{,}X$=1 (anti-diagonal A3−B2−C1)$N2{,}O$=0

Heuristic values:

$h(s)=(N2{,}X-N2{,}O)$

$$h(\text{sB3}) = 1 - 1 = 0$$

No immediate win — game continues; next is O's turn.

POST-ACTIVITY FOR ADVERSARIAL SEARCH

3. Do the Tic-Tac-Toe coding in Google Colab and save the file as: Tic-Tac-Toe-Sy-Christian

PARTII

ALPHA-BETA PRUNING ALGORITHM