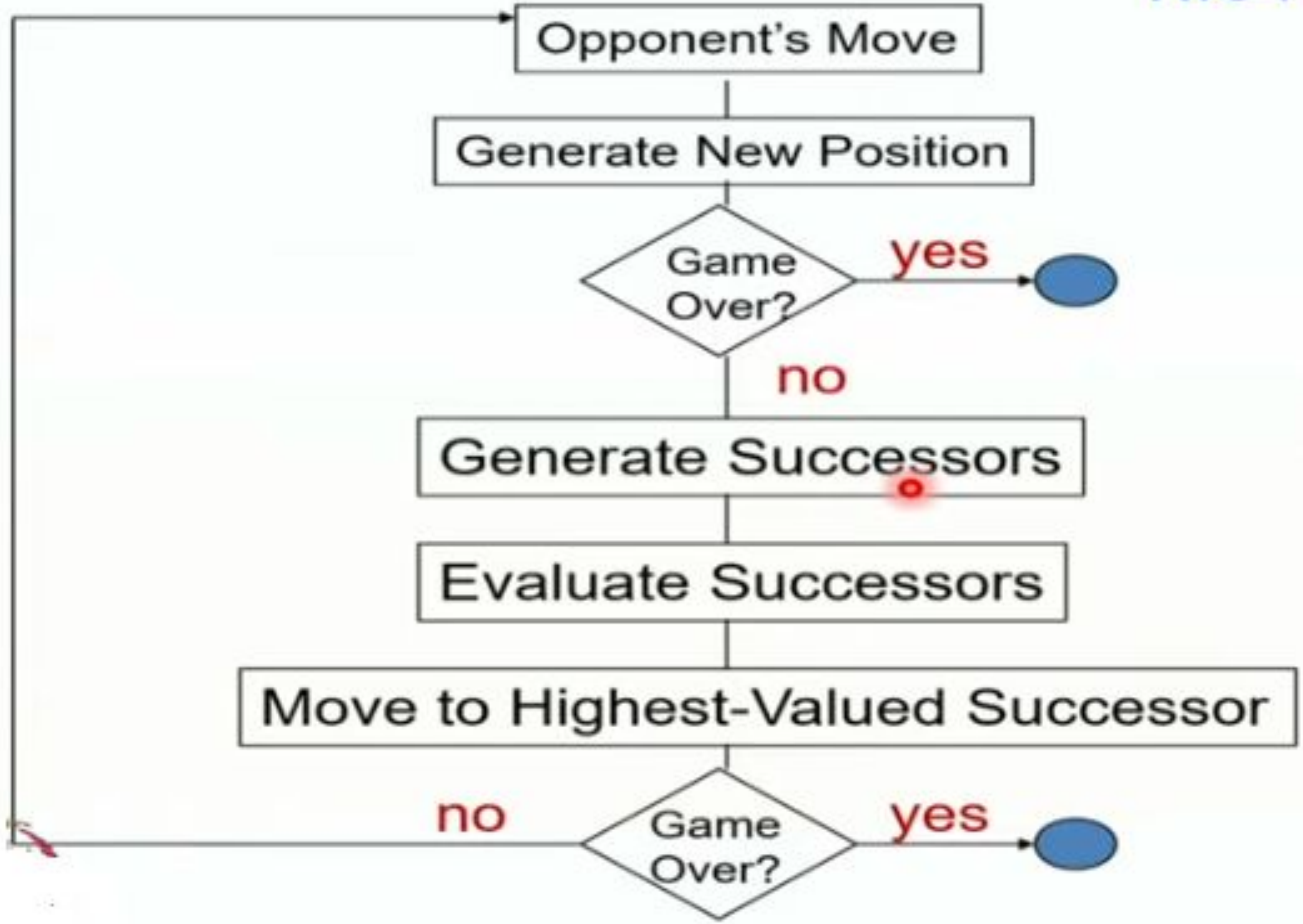
# Game Playing (Adversarial Search)

Unit 4

## Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

## Two-Player Game



## Adversarial Search

- The environment with more than one agent is termed as **multi-agent environment**.
- In multiagent environment, each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.**
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

## Types of Games in AI:

- **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 are 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, such as tic-tac-toe, Battleship, blind, Bridge, etc.

## Types of Games in AI:

**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 are chess, Checkers, Go, tic-tac-toe, etc.

## Types of Games in AI:

**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. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

## Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- 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.
- Chess and tic-tac-toe are examples of a Zero-sum game.
- Utility values at the end of the game are always equal & opposite.

# Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.
- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do
- Each of the players is trying to find out the response of his opponent to their actions.

## Games as Adversarial Search

- States:
  - board configurations
- Initial state:
  - the board position and which player will move
- Successor function:
  - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
  - determines when the game is over
- Utility function:
  - gives a numeric value in terminal states

(e.g., -1, 0, +1 for loss, tie, win)

# Game Tree (2-player, Deterministic, Turns)

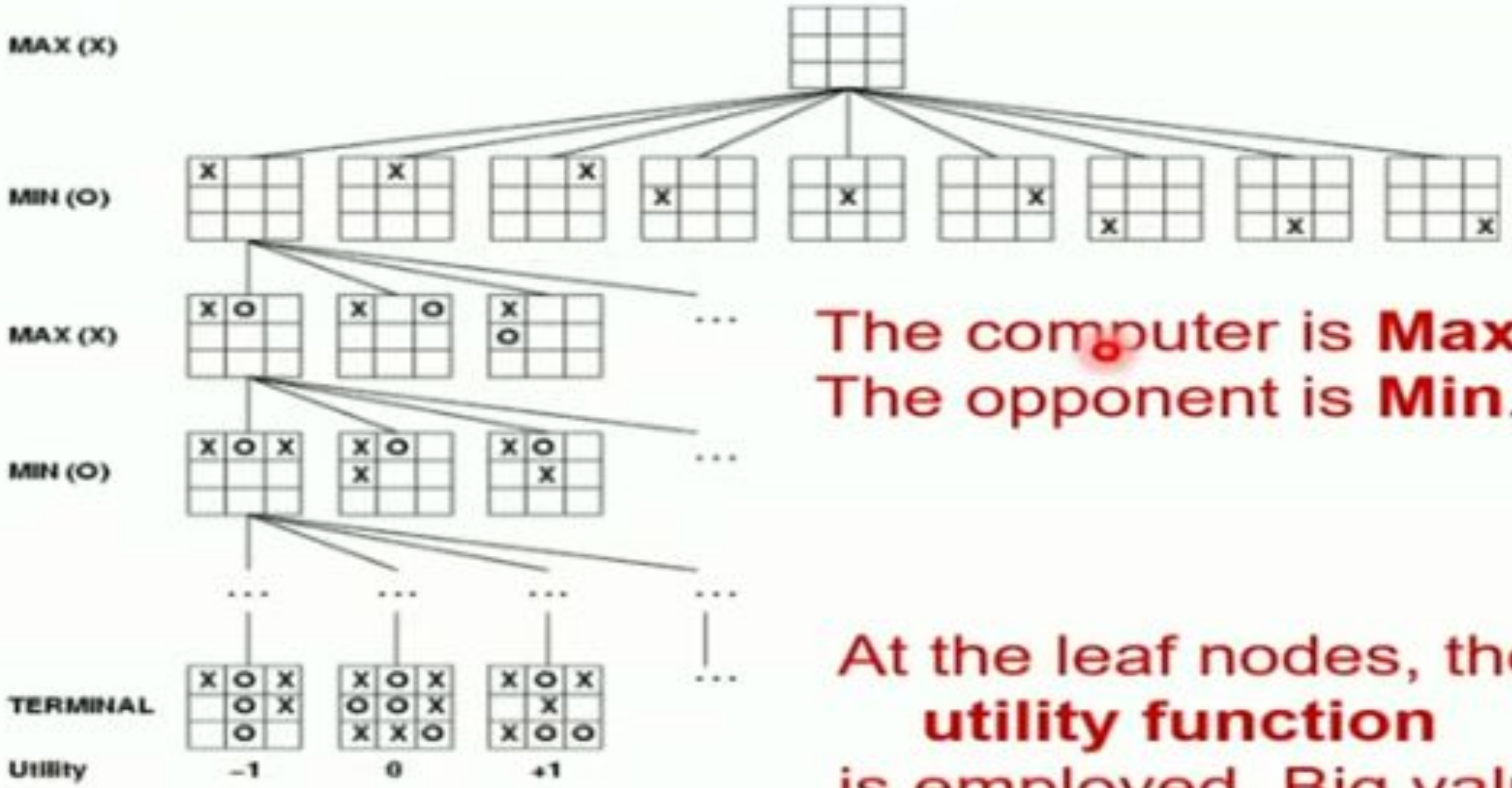
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



The computer is **Max**.
The opponent is **Min**.

At the leaf nodes, the **utility function** is employed. Big value s good, small is bad.

So, search tree formulation.

# Mini-Max Terminology

- **move:** a move by both players
- **ply:** a half-move
- **utility function:** the function applied to leaf nodes
- **backed-up value**
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- **minimax procedure:** search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

## Formalization of the problem:

- **A game can be defined as a type of search in AI which can be formalized of the following elements:**

**1. Initial state:** It specifies how the game is set up at the start.

**2. Player(s):** It specifies which player has moved in the state space.

**3. Action(s):** It returns the set of legal moves in state space.

**4. Result(s, a):** It is the transition model, which specifies the result of moves in the state space.

**5. Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.

**6. Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p.

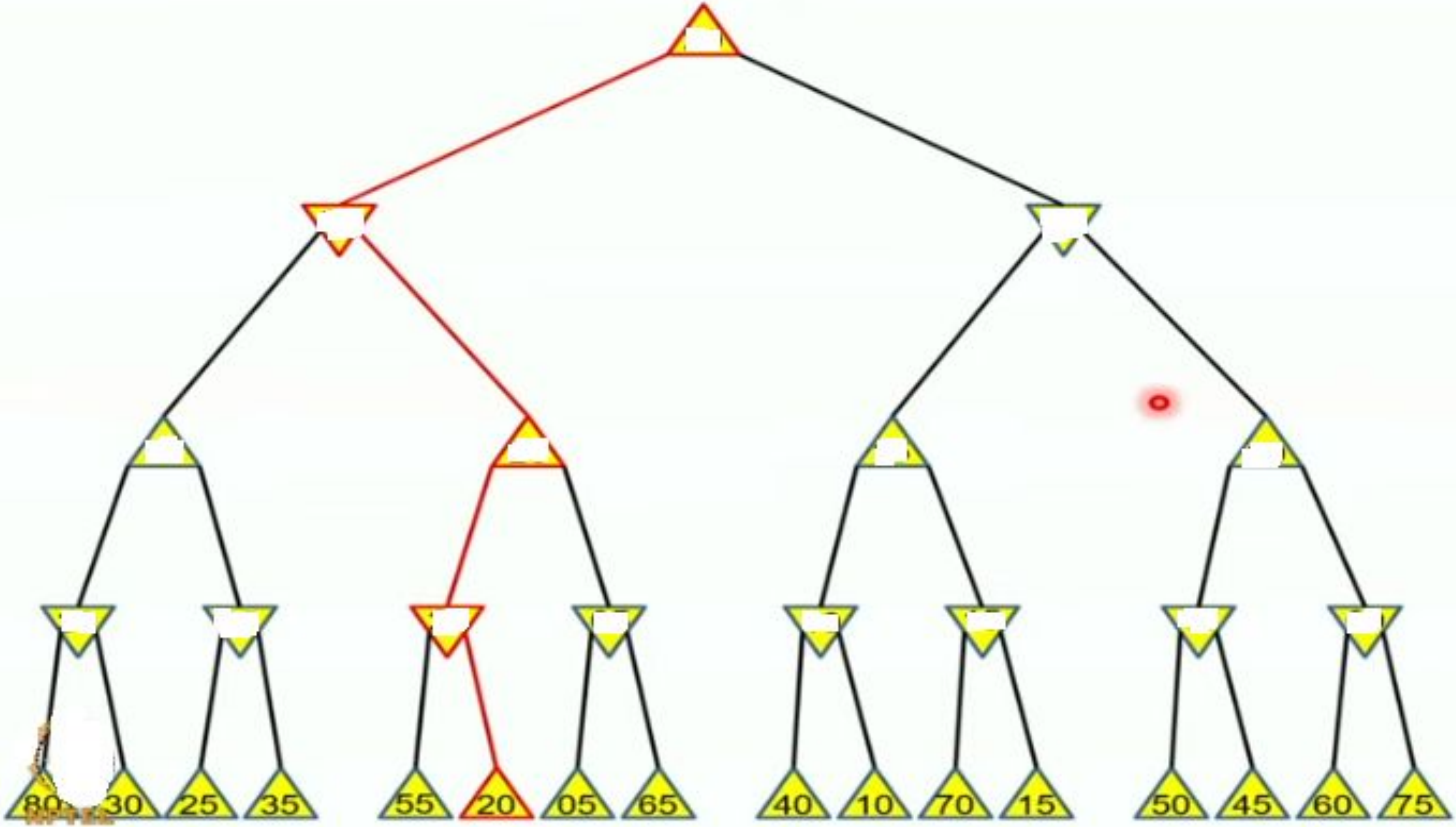
It is also called payoff function. For tic-tac-toe, the outcomes are a win, loss, or draw and its payoff values are +1, -1, and 0.

## Game tree:

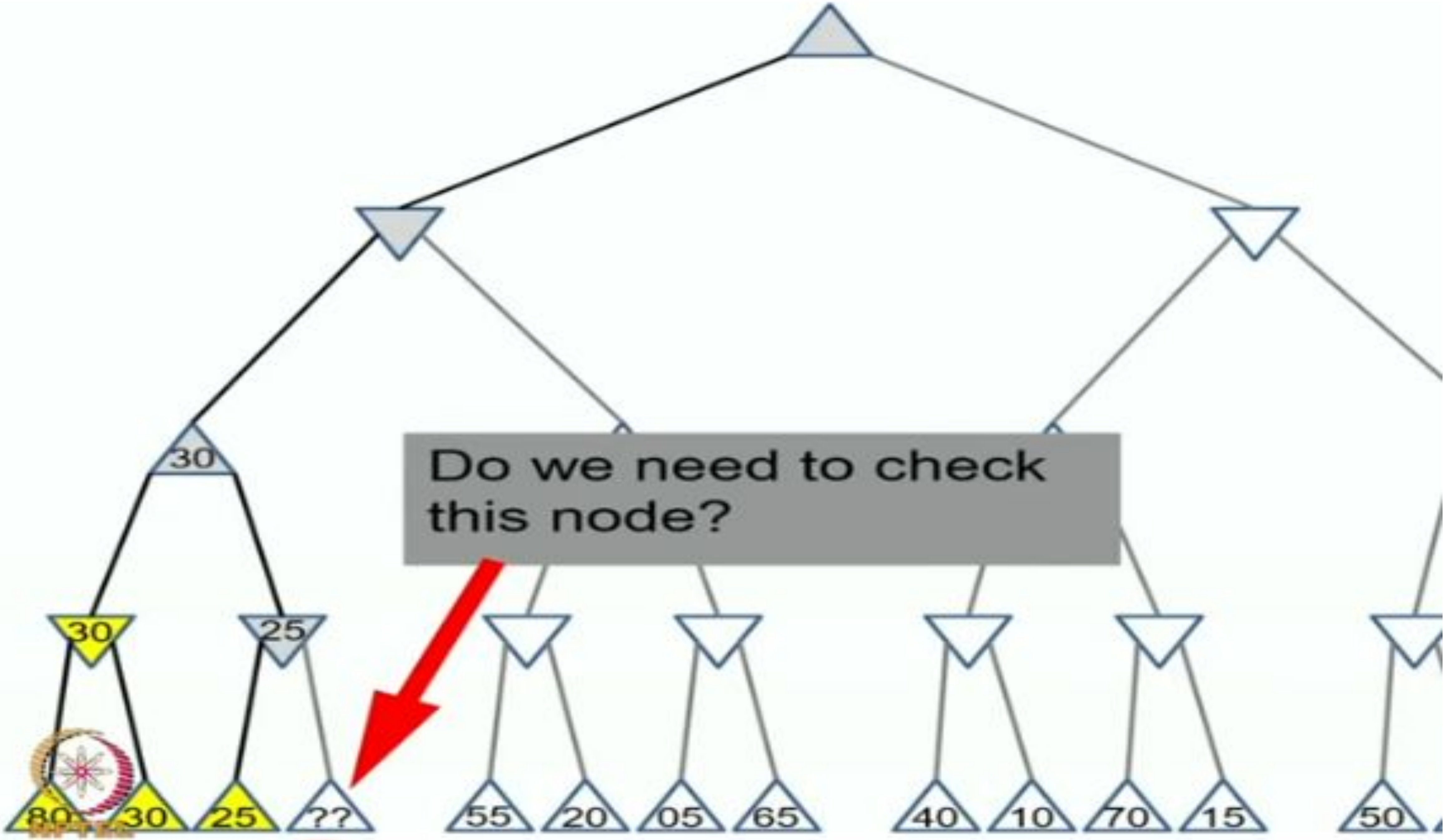
- A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players.
- Game tree involves initial state, actions function, and result Function.
- **Example: Tic-Tac-Toe game tree:**

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

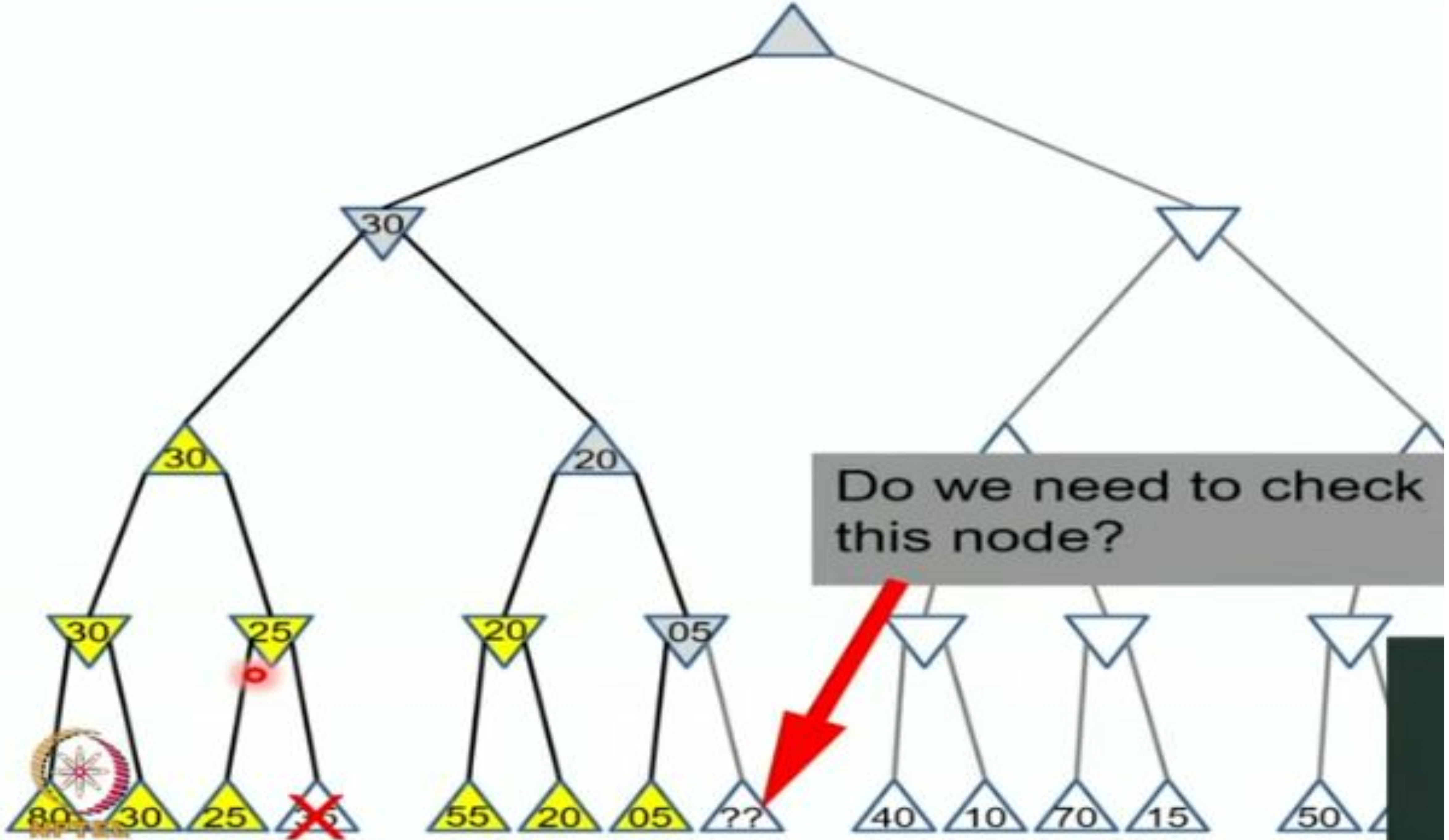
- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.



80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75



Do we need to check this node?

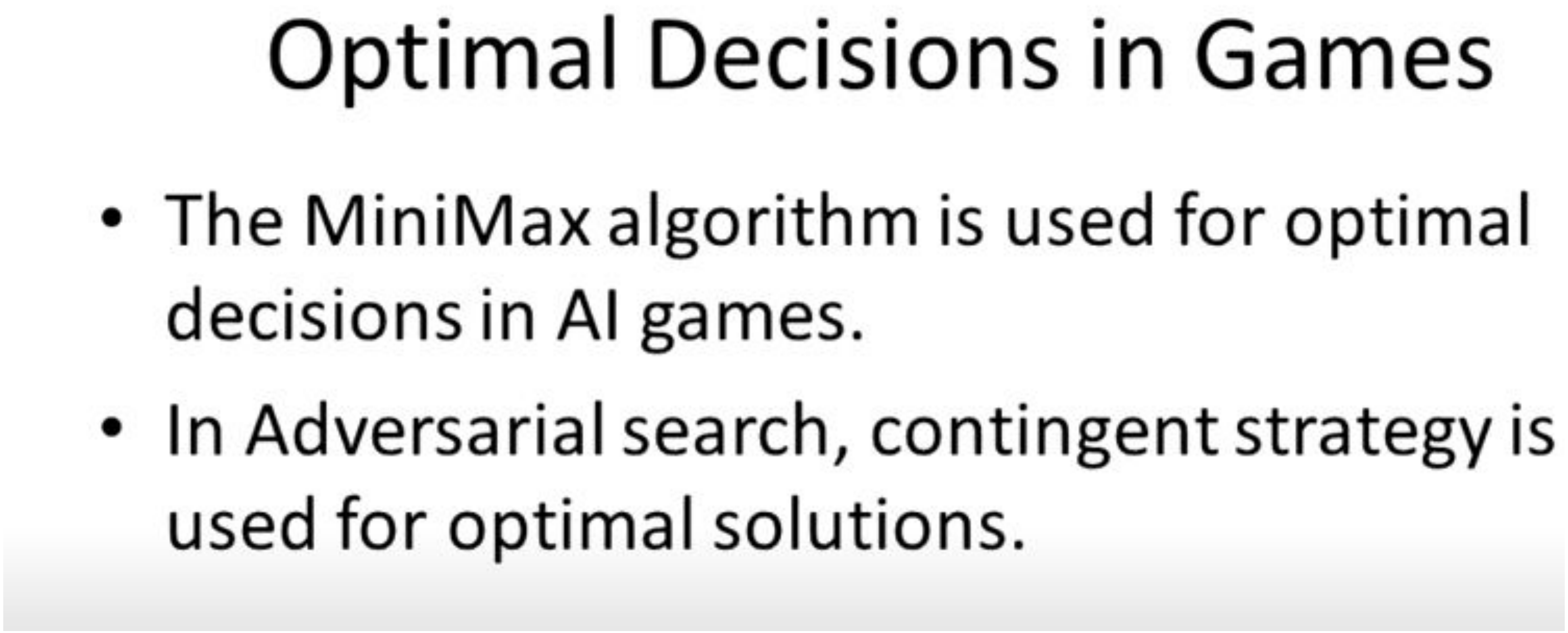


Do we need to check this node?

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Hence adversarial Search for the minimax procedure works as follows:

- It aims to find the optimal strategy for MAX to win the game.
- It follows the approach of Depth-first search.
- In the game tree, optimal leaf node could appear at any depth of the tree.
- Propagate the minimax values up to the tree until the terminal node discovered.



# Optimal Decisions in Games

- The MiniMax algorithm is used for optimal decisions in AI games.
- In Adversarial search, contingent strategy is used for optimal solutions.

## Optimal Decisions in Games (Cont…)

- Given a game tree, the optimal strategy can be determined from the minimax value of each node.
- Minimax value of a node ‘n’ is written as MINIMAX(n).

## MINIMAX(s)

$$\text{MINIMAX}(S) = \begin{cases} \text{UTILITY}(S)\text{ , if TERMINAL\_TEST}(S) \\ \max_{a \epsilon \text{ ACTION}(s)} \text{MINIMAX}(\text{RESULT}(S,a))\text{ , if PLAYER}(s)\text{=MAX} \\ \min_{a \epsilon \text{ ACTION}(s)} \text{MINIMAX}(\text{RESULT}(S,a))\text{ , if PLAYER}(s)\text{=MIN} \end{cases}$$

## Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- This is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**.
- The word ‘pruning’ means cutting down branches and leaves.
- This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**.
- It is also called as **Alpha-Beta Algorithm**.

# Alpha-Beta Pruning

- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

  The two-parameter can be defined as:

  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer.
  - The initial value of alpha is **-∞**.

  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer.
  - The initial value of beta is **+∞**.

## Alpha-Beta Pruning

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.
- Hence by pruning these nodes, it makes the algorithm fast.
- The main condition which required for alpha-beta pruning is:

$$\alpha >= \beta$$

## Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

## Move Ordering in Alpha-Beta pruning

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.