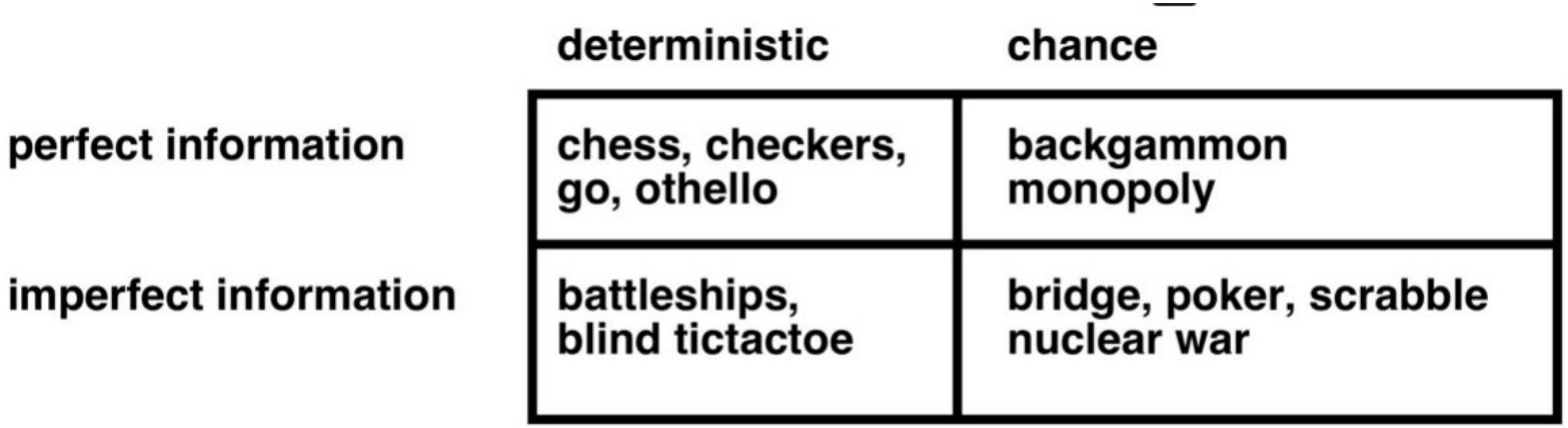
Adversarial Search

- Adversarial Search = Games
- Multi Agent Competitive Environment
- Game Vs Search: Optimal Solution is not a sequence of actions but a strategy. If opponent does a, agent does b, else if opponent does c, agent does d, etc.

"Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us."

Types of Games



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

Zero Sum Game

- Pure Competition
- 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.

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. This requires embedded thinking or backward reasoning to solve the game problems in AI.

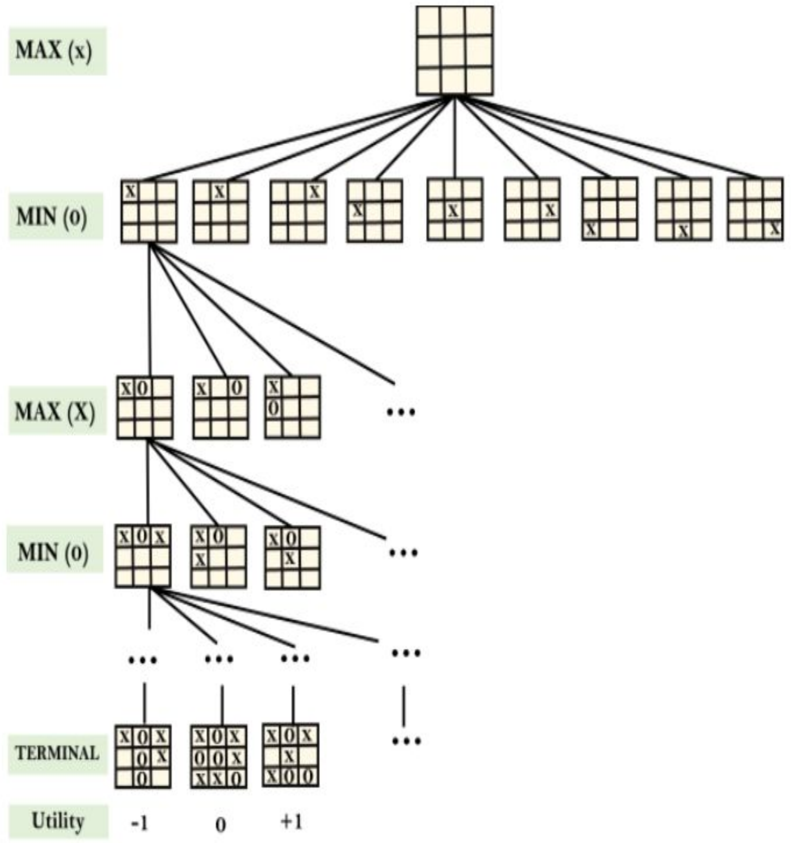
Formalization of the problem

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **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.
- **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 Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

Game Tree

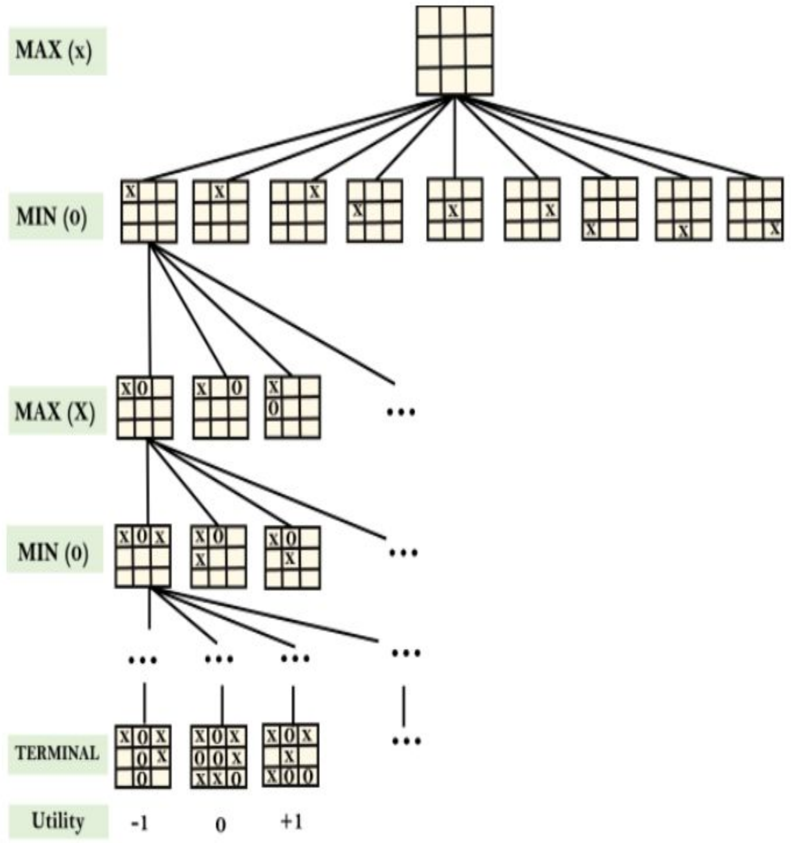
- Nodes of the tree are the game states
- Edges of the tree are the moves by players.
- Game tree involves initial state, actions function, and result Function.

Tic-Tac-Toe Game Tree



- 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.

Tic-Tac-Toe Game Tree



- 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.

Adversarial Search

For a state s minimax(s) =

$$\begin{cases} Utility(s) & \text{if Terminal-test(s)} \\ max_{a \in Actions(s)} \text{ minimax(Result(s,a))} & \text{if Player(s) = Max} \\ min_{a \in Actions(s)} \text{ minimax(Result(s,a))} & \text{if Player(s) = Min} \end{cases}$$