University of Malta
BSc(Hons.) IT Year II

# Problem Solving Techniques in Artificial Intelligence

## Lecture 1

## Search

- Games as Search Problems
- Minimax Algorithm

Department of Computer Science and AI

# Why Study Games?

Games offer

- Intellectual Engagement
- Abstraction
- Representability
- Performance Measure

Not all games are suitable for AI research. We will restrict ourselves to 2 person perfect information board games.

*Reference for this lecture: Russell and Norvig chapter 5*

# Game Playing as Search

- Playing a game involves **searching for the best move**.

- Board games clearly involve notions like start state, goal state, operators, etc.

  We can this usefully import **problem solving techniques that we have already met**.

- There are nevertheless important differences from standard search problems.

# Game Playing Search: Special Characteristics

Main differences are **uncertainties** introduced by

- **Presence of an opponent.** (since you don't know what the opponent will do until he/she does it). Game playing programs must solve the *contingency problem*.

- **Complexity.** Most interesting games are simply too complex to solve by exhaustive means. Chess, for example, has an average branching factor of 35. Uncertainty also arises from not having the resources to compute a move which is guaranteed to be the best.

## Ingredients of 2-Person Games

- **Players:** We call them Max and Min.

- **Initial State:** Includes board position and whose turn it is.

- **Operators:** These correspond to legal moves.

- **Terminal Test:** A test applied to a board position which determines whether the game is over. In chess, for example, this would be a checkmate or stalemate situation.

- **Utility Function:** A function which assigns a numeric value to a terminal state. For example, in chess the outcome is win (+1), lose (-1) or draw (0). Note that by convention, we always measure utility relative to Max.

# Strategy

- **Normal search problem:**

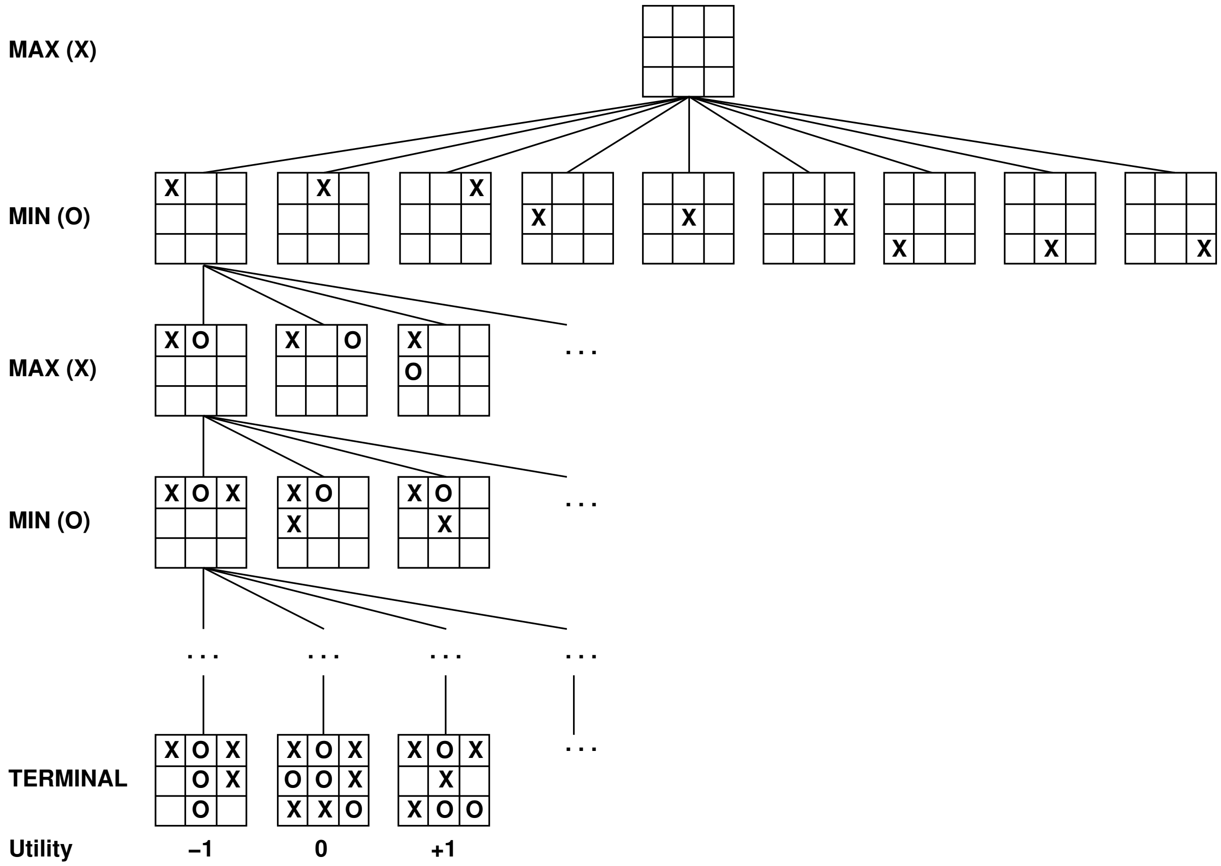
  Max searches for a sequence of moves yielding a winning position and then makes the first move in the sequence.

- **Game search problem:**

  Clearly, this is not feasible in in a game situation where Min's moves must be taken into consideration.

  Max must devise a **strategy** which leads to a winning position no matter what moves Min makes.

# A Game Tree for Naughts and Crosses




- Non Terminal Nodes

- Leaf Nodes

# A Perfect Decision Strategy Based on Minimaxing

1. Generate the whole game tree.

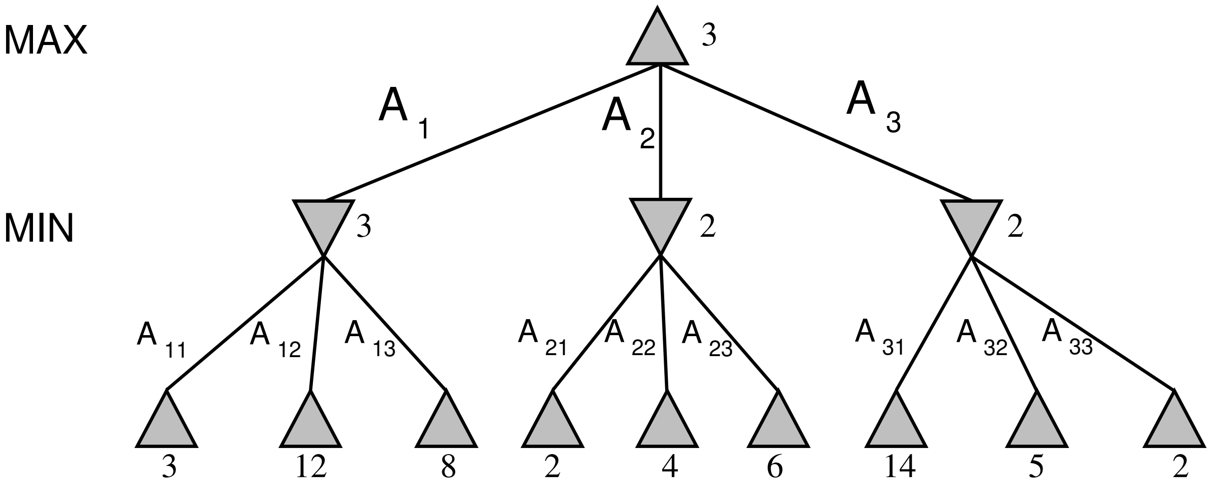
2. Apply the utility function to leaf nodes to get their values.

3. Use the utility of nodes at level n to derive the utility of nodes at level n-1.

4. Continue backing up values towards the root (one layer at a time).

5. Eventually the backed up values reach the top of the tree, at which point Max chooses the move that yields the highest value. This is called the **minimax decision** because it maximises the utility for Max on the assumption that Min wil play perfectly to minimise it.

# 2-ply Game Tree




# Minimax Algorithm

**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*)

## Imperfect Decisions

The strategy descibed above is perfect because it only assigns a utility when it has reached a terminal state of the game.

However, this is normally not feasible due to the sheer number of intermediate states.

How much space/time resources are available?

The remedy is to prune the search tree before reaching a leaf of the game tree and estimate the utility of the non-terminal state of the

This raises two issues:

1. How to assign a utility to a non-terminal state.

2. When to stop searching.

# Evaluation Function

## 1 ply

| | | |
|---|---|---|
| X | - | - |
| - | - | - |
| - | - | - |

| | | |
|---|---|---|
| - | X | - |
| - | - | - |
| - | - | - |

| | | |
|---|---|---|
| - | - | - |
| - | X | - |
| - | - | - |

## 2 ply

| | | |
|---|---|---|
| X | O | - |
| - | - | - |
| - | - | - |

| | | |
|---|---|---|
| X | - | O |
| - | - | - |
| - | - | - |

| | | |
|---|---|---|
| X | - | - |
| - | - | O |
| - | - | - |

| | | |
|---|---|---|
| X | - | - |
| - | - | - |
| - | - | O |

| | | |
|---|---|---|
| X | - | - |
| - | O | - |
| - | - | - |