**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

# PROBLEM-SOLVING AND SEARCH METHODS

## Search in competitive contexts

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2025/26

ISLab
Synthetic Intelligence Lab

# Agents and the Environment

Agents:

- They usually only have partial control over the environment, in the sense that they can influence it through their actions;
- They have to decide which action to take in order to achieve their goals;
- They can only carry out some of their actions, depending on the state of the environment (preconditions).

Source: Matrix

ISLab
Synthetic Intelligence Lab

## Environment
## Accessible vs. Inaccessible

- An accessible environment is one in which the agent can obtain complete, accurate and up-to-date information about the state of the environment;

- More moderately complex environments are inaccessible.

ISLab
Synthetic Intelligence Lab

## Environment
## Deterministic vs. Non-deterministic

- A deterministic environment is one in which any action is guaranteed to have only one effect;

- The real world is, for us humans, non-deterministic.

ISLab
Synthetic Intelligence Lab

## Environment
## Static vs. Dynamic

- a static environment, the world doesn't change while the agent is deliberating (acting);
- A non-static environment is called dynamic.


ISLab
Synthetic Intelligence Lab

## Environment
## Discrete vs. Continuous

- In a discrete environment, there is a fixed and finite number of possible actions and perceptions;

- A non-discrete environment is said to be continuous.

ISLab
Synthetic Intelligence Lab

## Environment
## Episodic vs. Non-episodic

In an episodic environment, the agent's execution time can be divided into a series of intervals (episodes) that are independent of each other, in the sense that what happens in one episode has no influence on the other episodes.

ISLab
Synthetic Intelligence Lab

# Environment
# The agent as an Interface

- Sensors: define the list of perceptions that the agent can use to perceive the world;

- Actuators: define the list of possible actions that the agent can take in the world

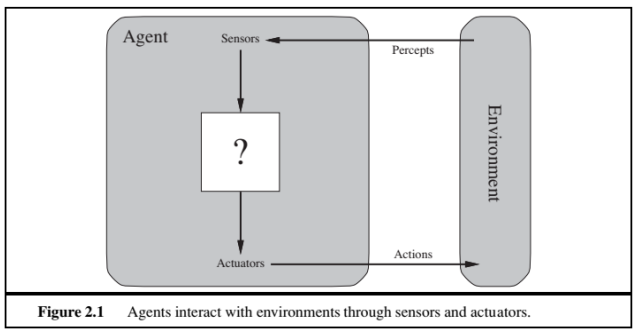


**Figure 2.1** Agents interact with environments through sensors and actuators.

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

| Sensors | Actuators |
|---|---|
| Cameras | Starting the car; acceleration; braking. |
| Speed Senosr | Turn right; turn left. |
| GPS | Speed changes |
| Microphone | |

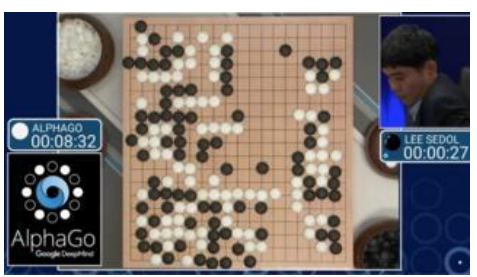
"a little intelligence goes a long way!"

ISLab
Synthetic Intelligence Lab

# Games as Search Problems

Source: https://milanwittpohl.com/projects/adversarial/index.html

Source: https://www.wired.com/2017/05/what-deep-blue-tells-us-about-ai-in-2017/




Source: https://www.bbc.com/news/technology-35785875

ISLab
Synthetic Intelligence Lab

# Agenda

- Games
- Games as Search Problems
- Minimax algorithm
- Alpha-beta pruning
- Imperfect Decisions in Real Time
- Deterministic vs Stochastic Games
- Stochastic Games
  - Perfect information
  - Imperfect (partial) information

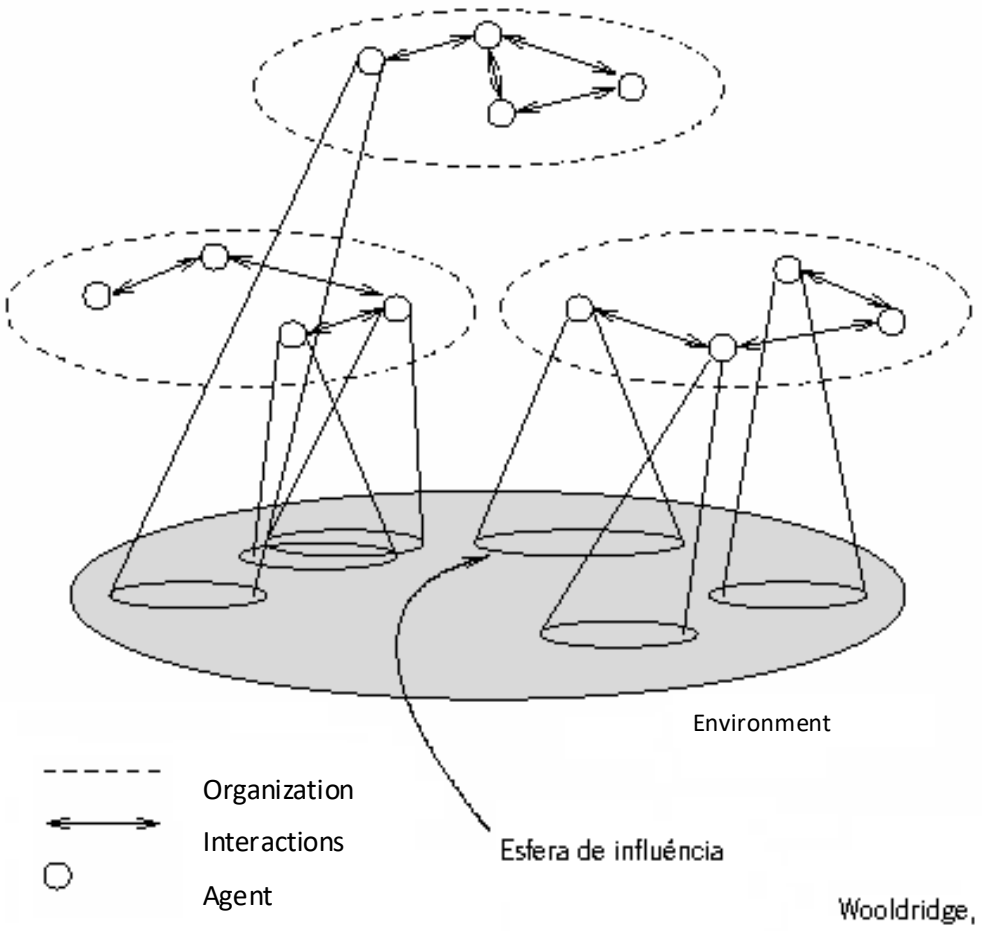
Conclusions

## Games as Search Problems

- In previous topics, we saw that search strategies are only associated with one entity (agent) that wants to find a solution (sometimes expressed in a sequence of actions);
- However, there may be situations in which there is more than one agent searching for solutions in a search space (which is often the case in games);
- In multi-agent environments like these, each agent needs to consider the actions of the other agents and how they affect it.

ISLab
Synth

# Multi-agent Systems




Wooldridge, 1999

## Games as Search Problems

- The unpredictability of these agents can create **contingencies** in the agent's problem-solving process.
- These competitive environments, in which agents' objectives are in conflict, give rise to **adversarial search problems**.
- Game Theory considers any multi-agent environment to be a **game**, in which the impact of one agent on the other agents is "significant", regardless of whether the agents are competitive or cooperative.

ISLab
Synthetic Intelligence Lab

## Why Study Games?

- Games are traditionally a mark of intelligence;
- Games are (relatively) easy to formalize;
- Games can be a good model of real-world competitive or cooperative activities.
  - e.g. military confrontations, negotiations, auctions.

ISLab
Synthetic Intelligence Lab

# Game theory

**Move**

- A move is the process by which the game progresses from one stage to another. Moves can alternate between players according to specific rules or occur simultaneously. It can result from a player's decision or because of a random event.

**Payoff**

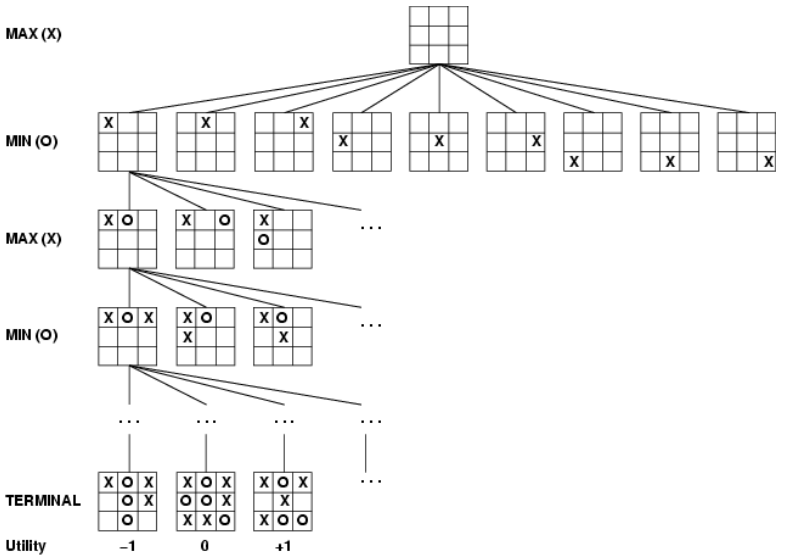
- At the end of the game, each player gets a payoff. We can associate this number with the amount that has been won or lost, or say, for example, that the payoff is +1 for the winner, 0 if there is a draw, and -1 for the loser.

**Strategy**

- A set of the best choices for a player, in which all the possible situations the player may face are already foreseen. Thus, with a strategy, the player will know how to act in any given move, regardless of the opponent or random events.

**Game tree**

- In which each 'node' represents a possible move, and each 'line' represents a move (one player after another).



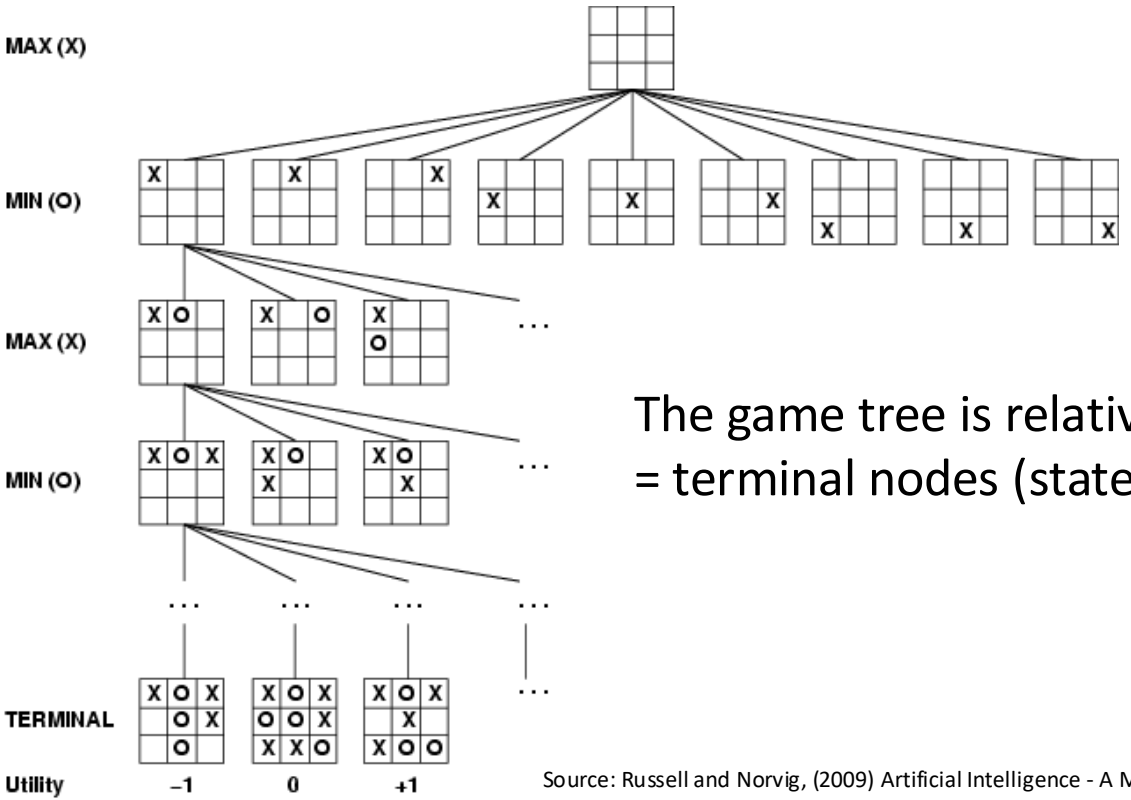

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

## Games vs. Single Agent Search

- We don't know how the opponent will act
  - The solution is not a fixed sequence of actions from the initial state to the goal state, but a strategy or policy (a mapping of the state to the best move in that state);;
- Efficiency is fundamental to playing well
  - The time to make a move is limited;
  - The branching factor, the depth of the search and the number of termination configurations are enormous.

ISLab
Synthetic Intelligence Lab

# A game of rooster between two players, "max" and "min"




The game tree is relatively small - less than 9!
= terminal nodes (states) (362 880)

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

# Games as Search Problems

- **Types of Games :**
  - Information:
    - Perfect: Chess, Checkers, Go, Othello, Backgammon, Monopoly
    - Imperfect: Poker, Scrabble, Bridge, King
  - Lucky/Deterministic:
    - Deterministic: Chess, Checkers, Go, Othello
    - Lucky Game: Backgammon, Monopoly, Poker, Scrabble, Bridge, King
- **Plan of Attack:**
  - Algorithm for the perfect game
  - Finite horizon, approximate evaluation
  - Pruning trees to reduce costs

## Games as Search Problems

- **Characteristics :**
  - Hostile Agent (adversary) included in the world
  - Unpredictable Opponent => Solution is a Contingency Plan
  - Time Limit => Unlikely to find the objective. An approach is required
  - One of the oldest areas of AI. In 1950 Shannon and Turing created the first chess programs.
  - Chess:
    - Everyone thinks you have to be "smart" to play it
    - Simple rules but the game is complex
    - World fully accessible to the agent
    - Average branching factor of 35, game with 50 moves => 35100 leaves in the search tree (although there are only 1040 legal positions)

## Games as Search Problems

- Considering a game of perfect information with two opponents, we can formally define it as a search problem through the following elements :
  - **Initial State** (position on the board and which player will play next)
  - **Set of Operators** (which define the legal moves)
  - **Terminal Test** (which determines whether the game is over - terminal state)
  - **Utility Function** (which gives a numerical value for the outcome of the game, e.g. 1 (win), 0 (draw), -1 (loss))

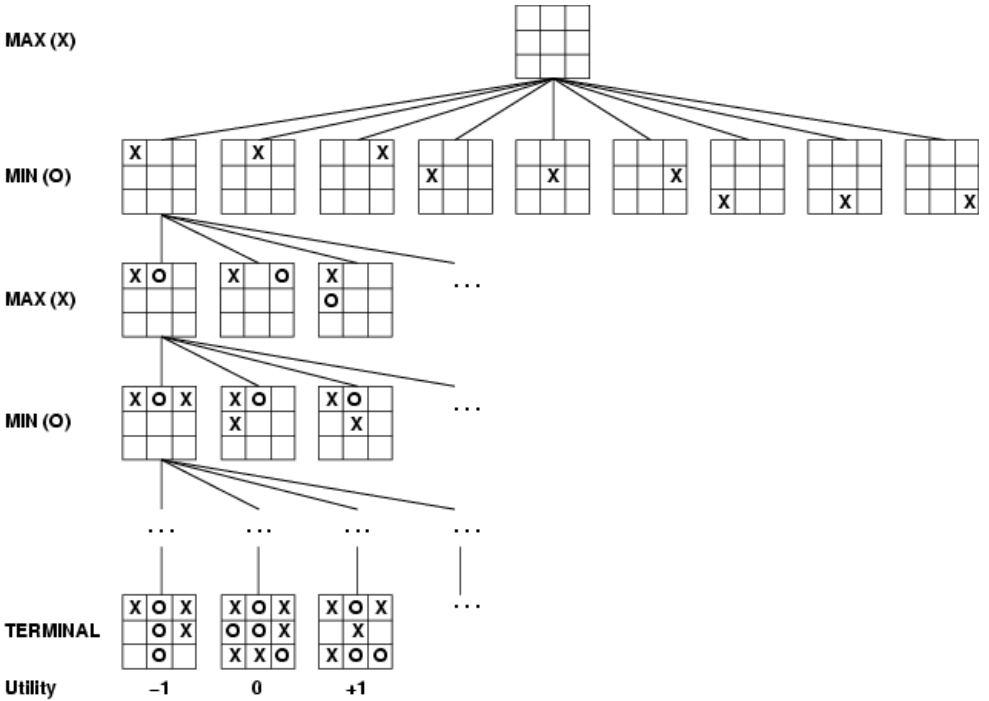
- The MiniMax algorithm can be applied as a solving strategy in this type of game (with two opponents and perfect information).

ISLab
Synthetic Intelligence Lab

# Games as Search Problems

- Minimax: Rooster Game

- The top node is the initial state
- MAX plays first, placing an X in the empty square
- The image shows part of the search tree, showing the possible alternative moves for MIN(o) and MAX(x) until they reach the terminal states (where utility values can be assigned according to the rules of the game).




Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

# Ideal Strategies

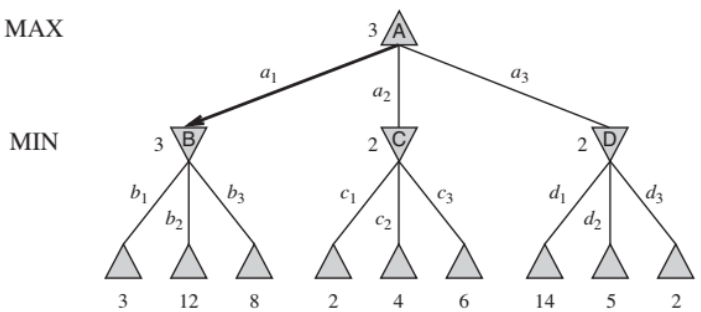
- In a normal search problem, the ideal solution would be a sequence of moves that lead to a goal state (an end state that is a win).

- In a game, MIN has something to say about this and MAX must therefore find a contingent strategy that specifies :
  - the movement of NAX in the initial state,
  - then MAX's movements in the states resulting from all possible MIN responses,
  - MAX's movements in the states resulting from every possible MIN response to these movements

- An ideal strategy leads to results that are at least as good as any other strategy when you're playing an infallible opponent.

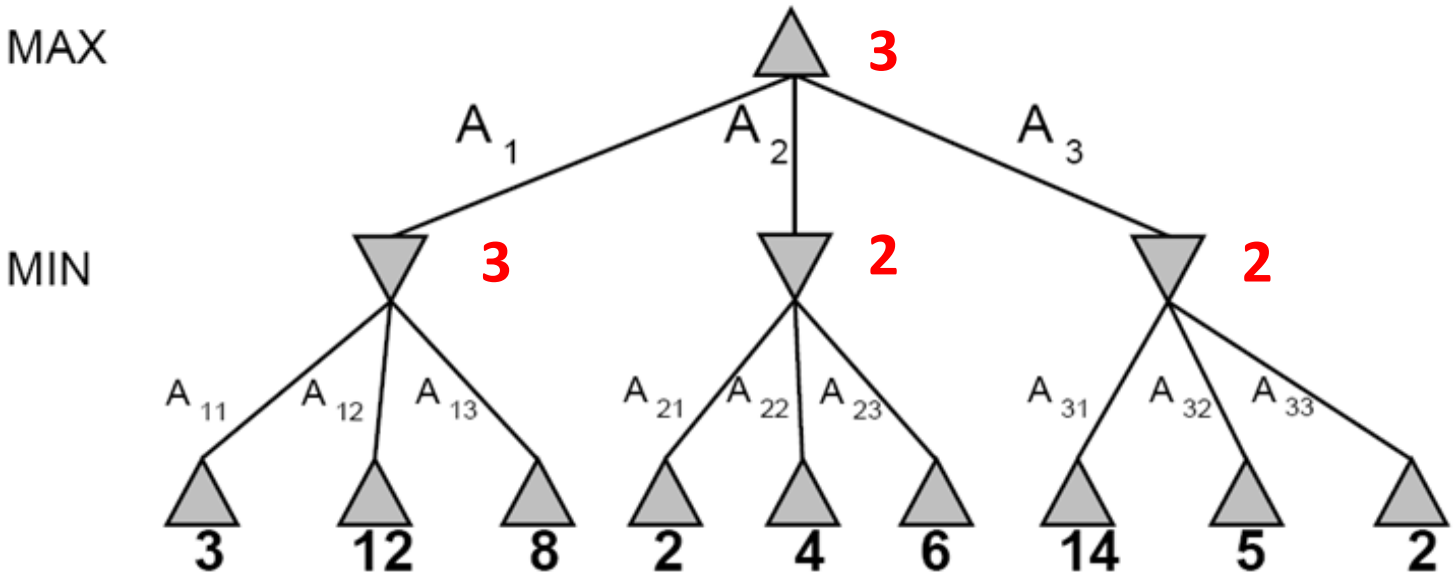
# Games as Search Problems

- The MiniMax algorithm can be applied as a solving strategy in this type of game (deterministic, with two opponents and perfect information) and consists of :
  - Generate the complete tree up to the terminal states
  - Apply the utility function to these states
  - Calculate the utility values up to the root of the tree, one layer at a time
  - Choose the move with the highest value




Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

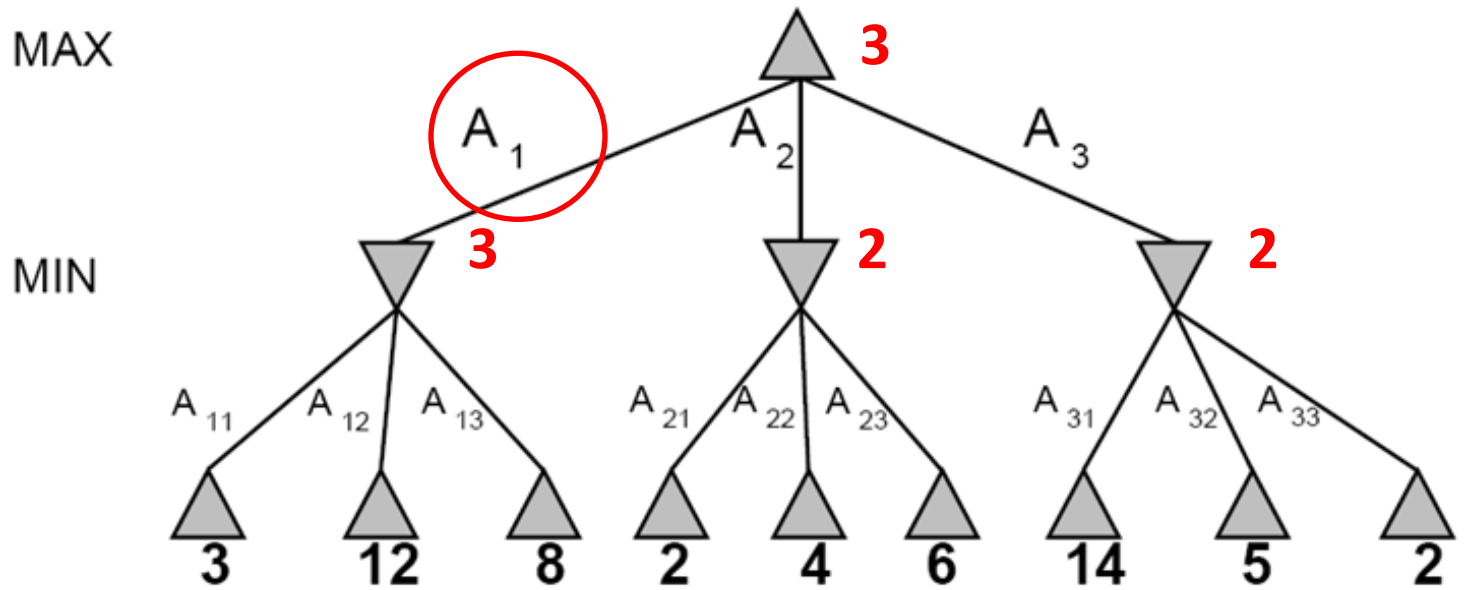
# Game Tree




- **Minmax value of a node**: the utility (for MAX) of being in the corresponding state, assuming perfect performance on both sides
- **Minimax strategy:** choose the move that offers the best worst-case return

ISLab
Synthetic Intelligence Lab

# Calculating the minimax value of a node




- Minimax (node) =
  - utility (nó) if o node is terminal
  - $Max_{action}$ Minimax (Sucessor(node, action)) if player = MAX
  - $min_{action}$ Minimax (Sucessor(node, action)) if player = MIN

## Minimax Value

- Given a game tree, the optimal strategy can be determined by analyzing the minimax value of each node (MINIMAX-VALUE (n))
- The minimax value of a node is the utility of being in the corresponding state, assuming that the two players play optimally from then until the end of the game.
- MAX prefers to move to a state of maximum value, while MIN prefers a state of minimum value.

# Algorithm Minimax

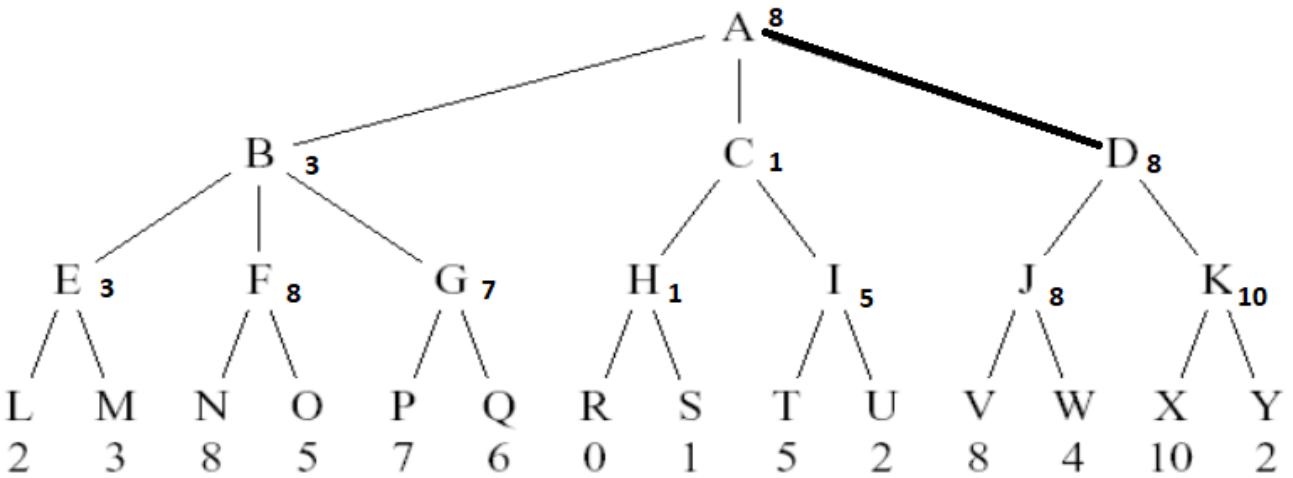
```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

# Exercise – MINIMAX

Solution:

- Assuming that MAX is the first to play, apply the Minimax Algorithm to the following tree, indicating the move selected by the algorithm and its estimated value.




Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

ISLab
Synthetic Intelligence Lab

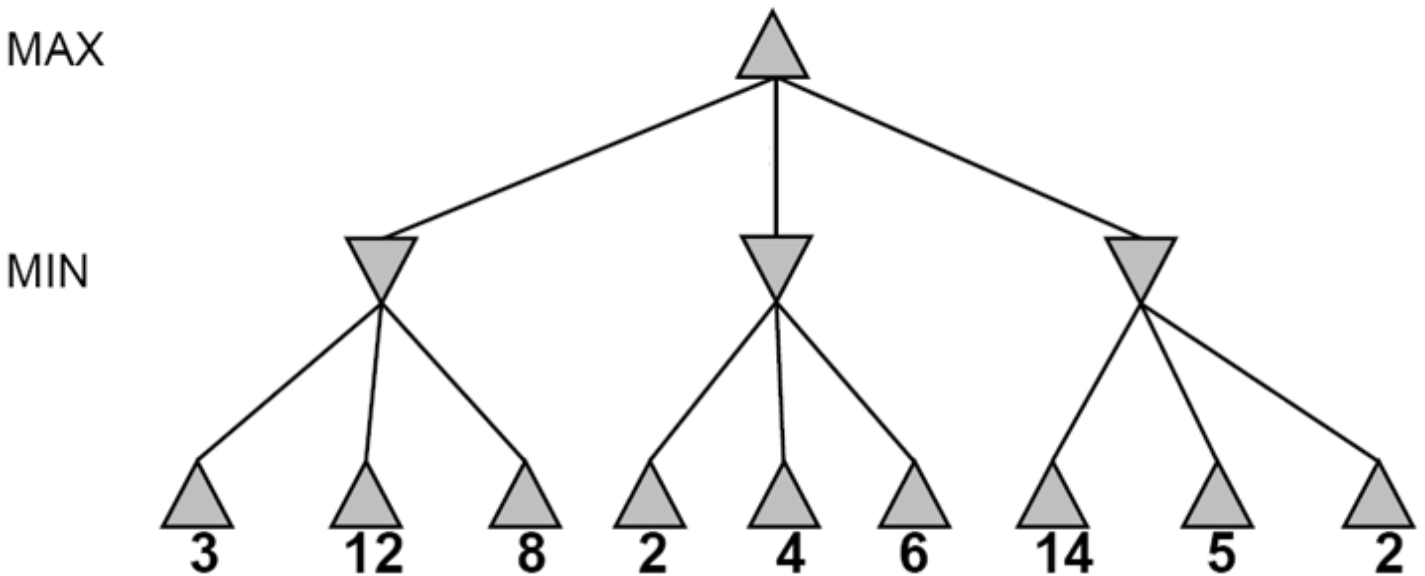
# Minimax Properties

- Properties:
  - Complete? Yes, if the tree is finite!
  - Great? Yes, against an optimal opponent! Otherwise?
  - Time complexity? $O(b^m)$
  - Complexity in Space? O(bm) (first in-depth exploration)
- Problem:
  - Unviable for any minimally complex game
- Example:
  - For Chess(b=35, m=100), $b^m=35^{100}=2.5*10^{154}$
  - Assuming 450 million hypotheses are analyzed per second => $2*10^{138}$ years to find a solution!

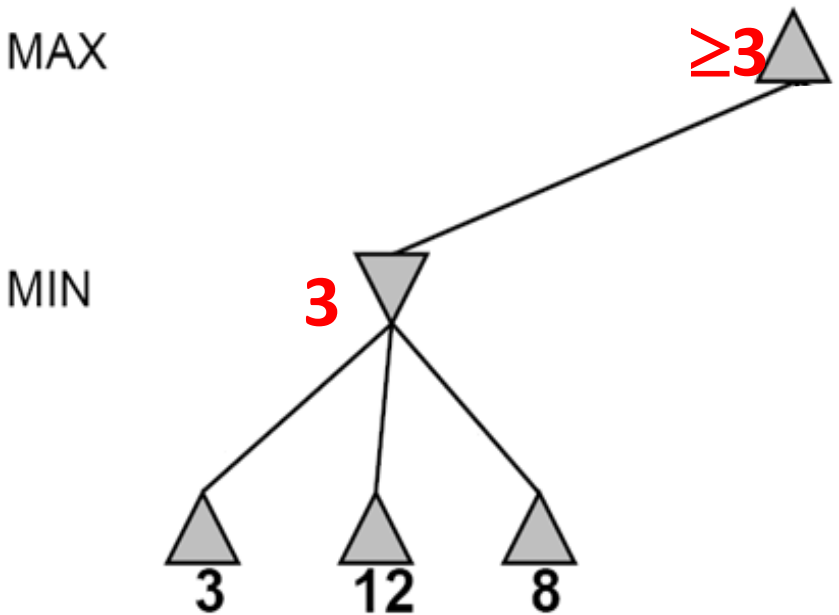
- b: the maximum branching factor (the maximum number of successors to a node) of the search tree
- d: the depth of the best solution
- m: the maximum depth of the state space

# Alpha-beta Pruning

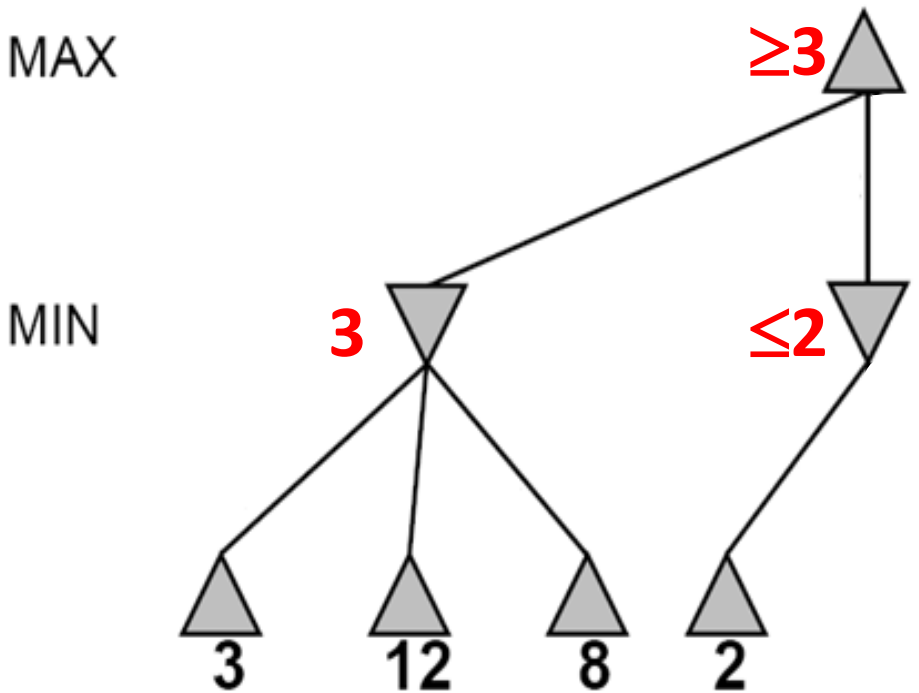
- It is possible to calculate the exact minimax decision without expanding all the nodes in the game tree…



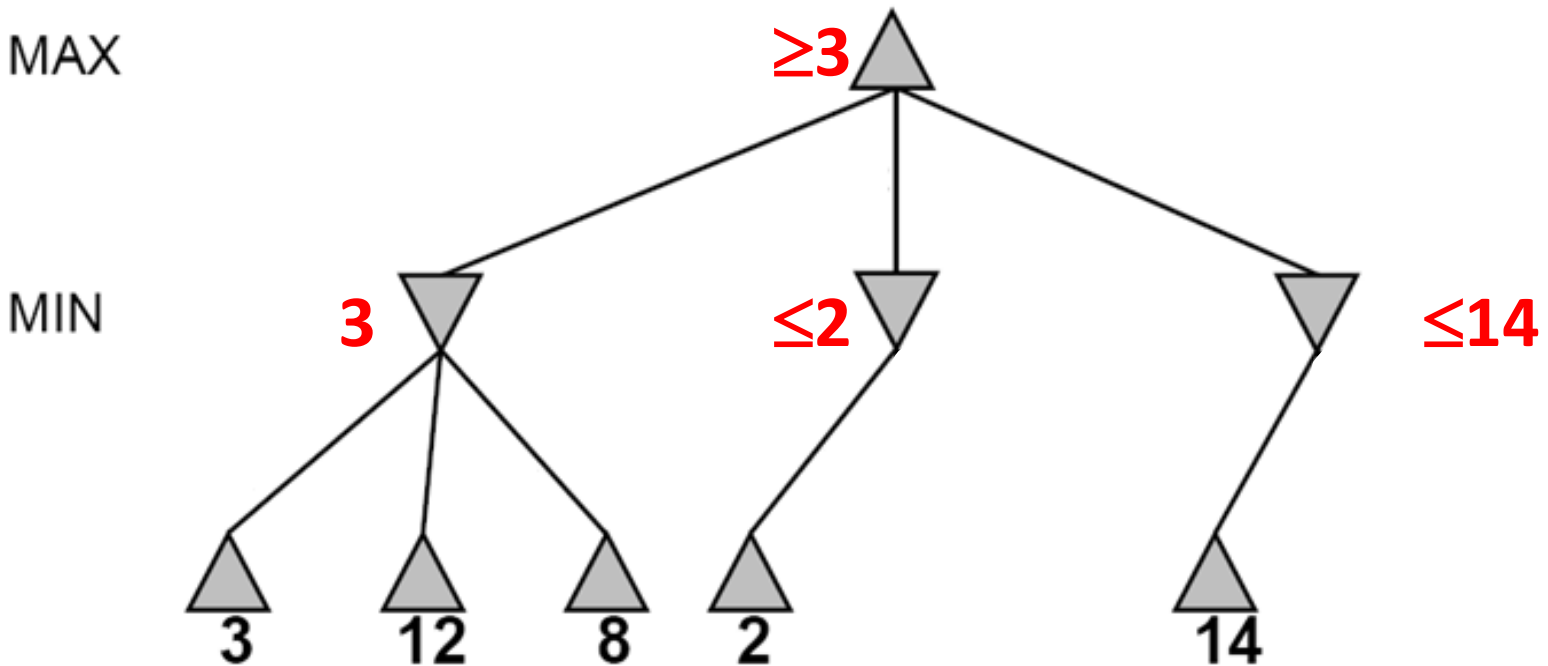
# Alpha-beta Pruning



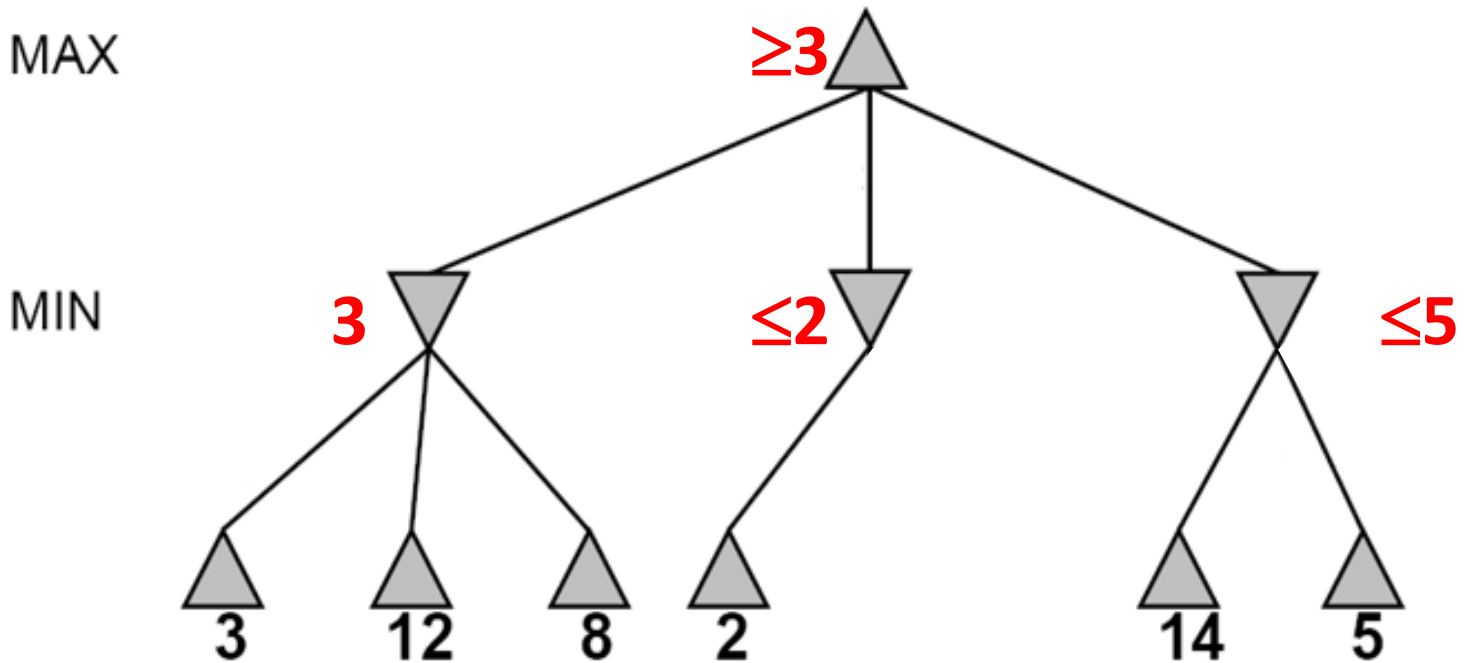
# Alpha-beta Pruning



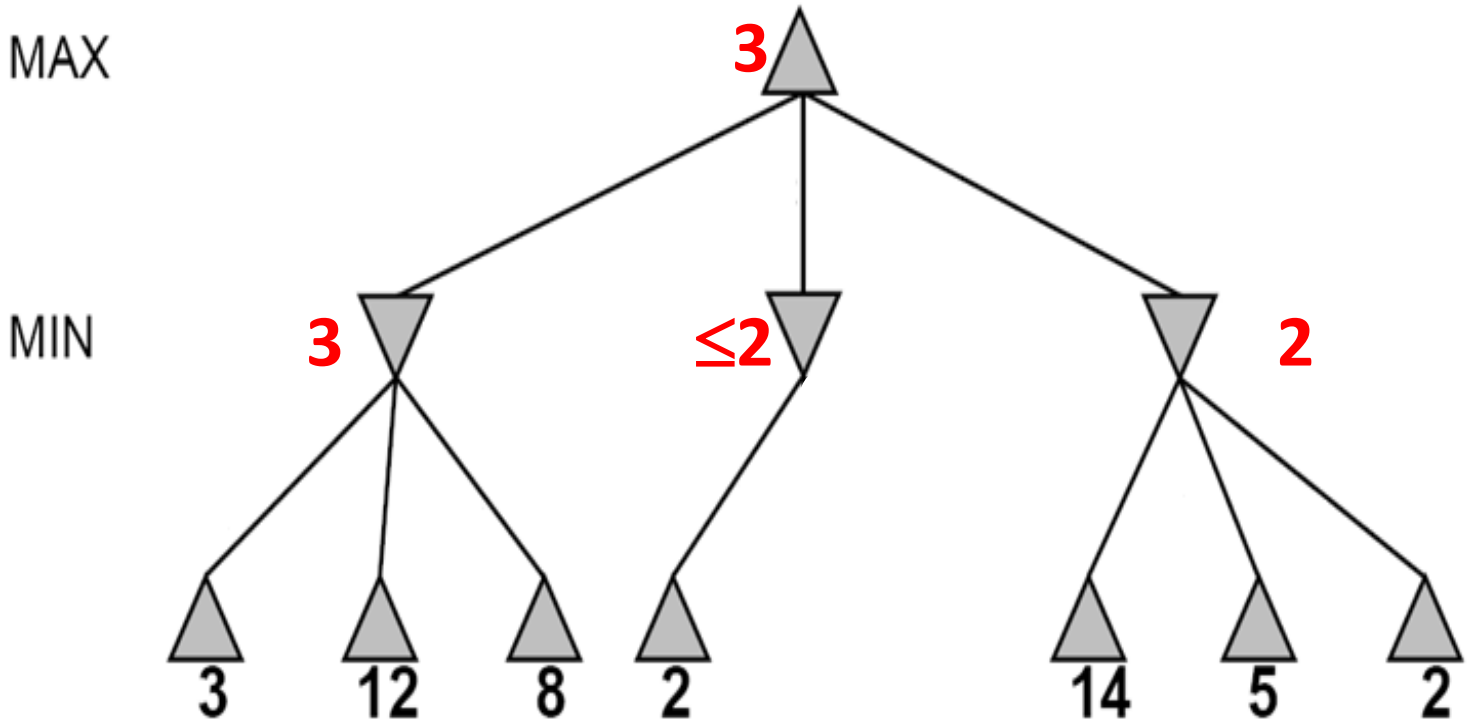
# Alpha-beta Pruning



# Alpha-beta Pruning



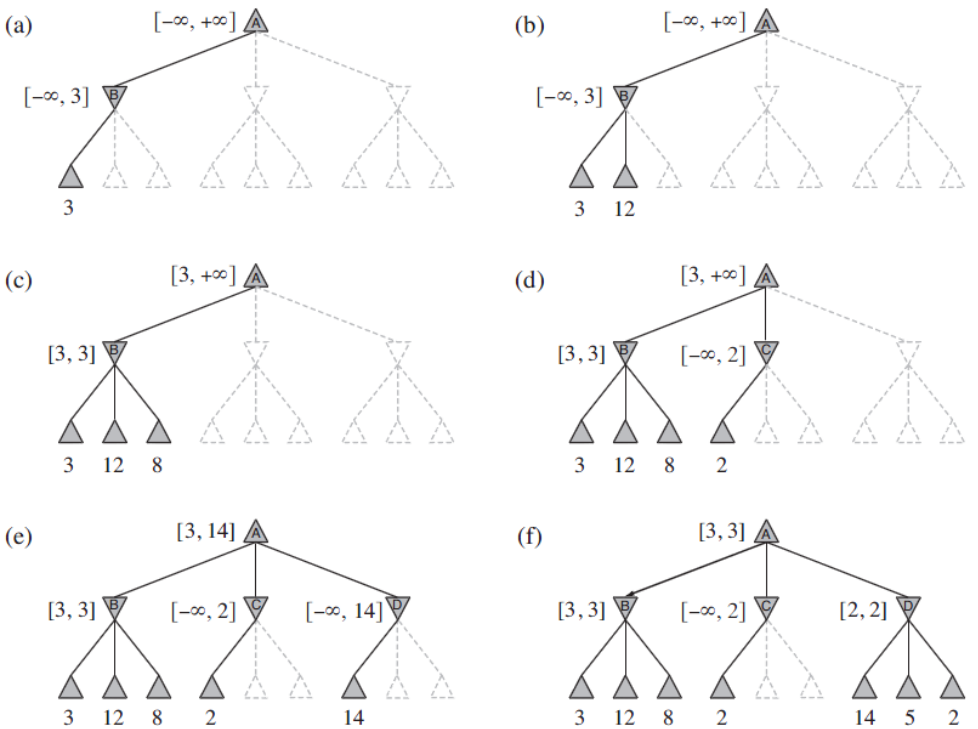
# Alpha-beta Pruning



ISLab
Synthetic Intelligence Lab

# Alpha-beta pruning
# Pruning of Alfa-Beta

- $\alpha$ is the best value (for Max) found so far on the current path
- If V is worse then $\alpha$, Max should avoid it => cut the branch
- $\beta$ is defined in the same way for Min

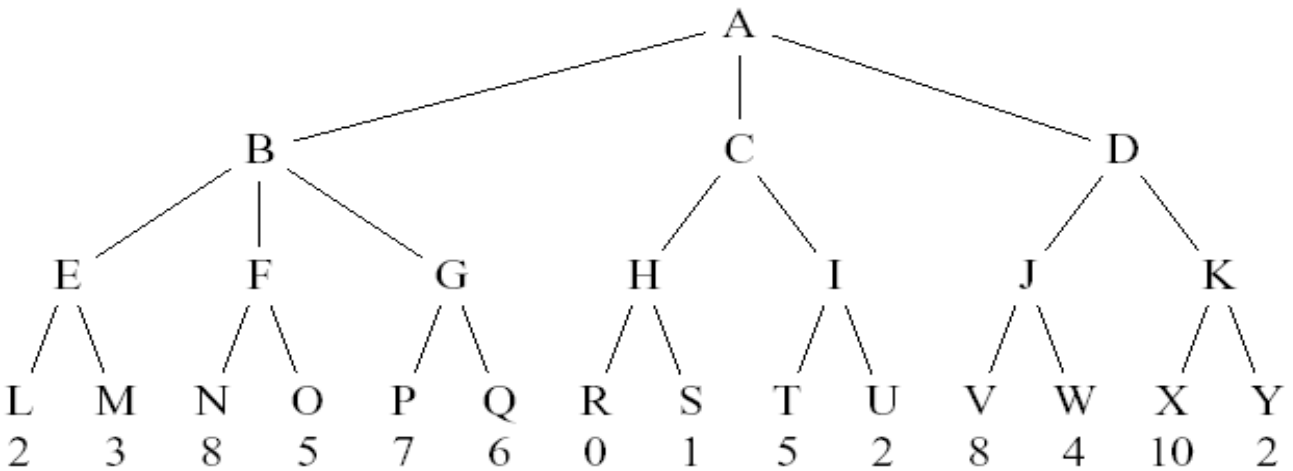



Alpha-Beta does not affect the final result
Good sorting improves pruning efficiency

ISLab
Synthetic Intelligence Lab

# Exercise - MINIMAX with pruning

- Assuming that MAX is the first to play, apply the Minimax algorithm with Alpha-Beta pruning to the following tree, indicating the move selected by the algorithm. Indicate graphically and justify the pruning I perform when applying the Minimax algorithm

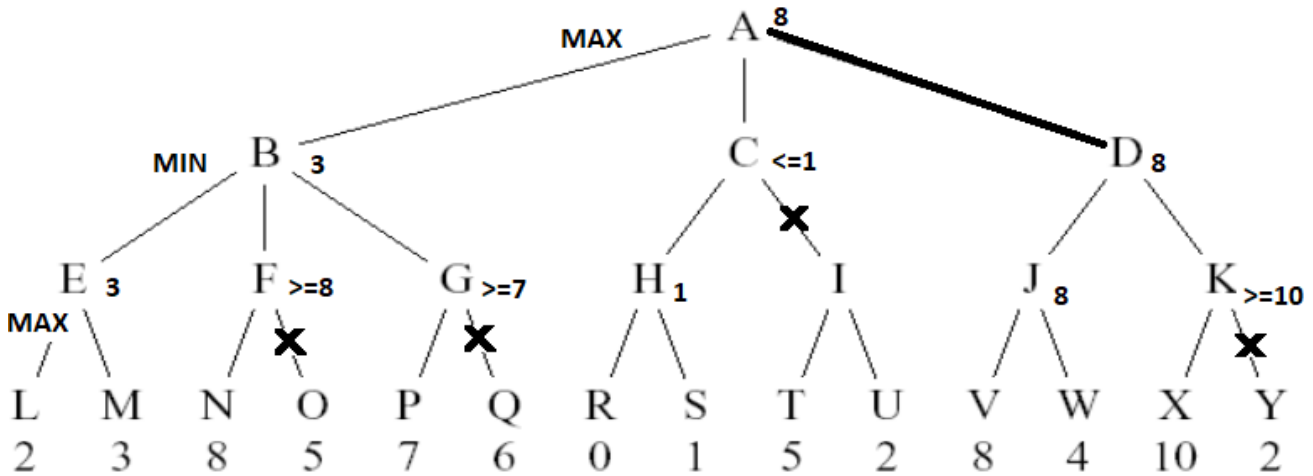



Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

# Exercise - MINIMAX with pruning

Solution:

- Assuming that MAX is the first to play, apply the Minimax algorithm with Alpha-Beta pruning to the following tree, indicating the move selected by the algorithm. Indicate graphically and justify the pruning I perform when applying the Minimax algorithm.




Source: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

# Alpha-beta pruning
# Pruning of Alfa-Beta

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v          ← pruning point
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v          ← Pruning point
    β ← MIN(β, v)
  return v
```

# The Problem
# Imperfect Decisions in Real Time

- The minimax algorithm generates the entire search space of the game, while the alpha - beta algorithm allows us to remove a large part of it. However, alpha - beta still needs to perform a search up to the terminal states in at least part of the search space;
- This depth is generally not practical, because changes must be made in a reasonable amount of time - usually a few minutes at most;
  - Evaluation Function: Estimated utility (interest) for the position
  - Cutoff Test: Depth Limit
- Another problem may arise here: the horizon problem!
  - Transposition table to store previously expanded states;
  - Direct pruning to avoid considering all possible moves;
  - Lookup tables for opening moves and endgames

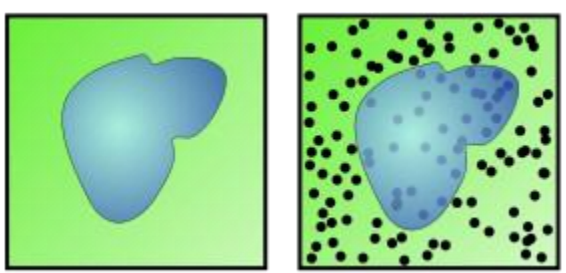
# Chess

- On base: 200 million node evaluations per move (3 min), minimax with a decent evaluation function and search for quiescence
  - 5 moves ≈ human novice
- Add alpha-beta pruning
  - 10 moves, experienced player
- Deep Blue: 30 billion evaluations per move, evaluation function with 8000 features, large databases of opening and endgame moves
  - 14 plays ≈ Garry Kasparov
- Current state of the art (Hydra, et al., 2006): 36 billion evaluations per second, advanced pruning techniques
  - 18 moves - better than any human being alive?

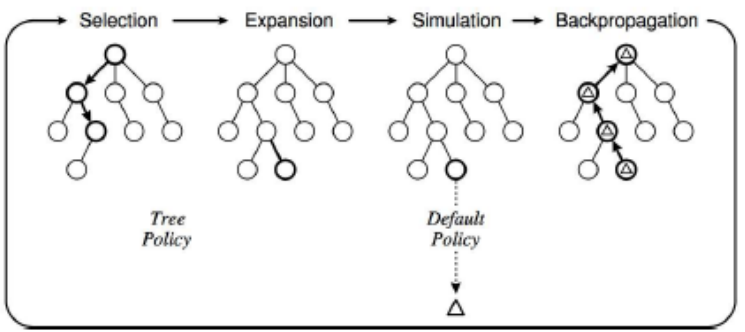
ISLab
Synthetic Intelligence Lab

# Monte Carlo Tree Search

- It applies to games with deep trees, a large branching factor and no good heuristics - like Go?
- Instead of limited depth search with an evaluation function, random simulations should be used.
- Starting from the current state (root of the search tree), iterate:
  - Select a leaf node for expansion using a tree policy (discovery and exploration conflict)
  - Run a simulation using a standard policy (e.g. random moves) until a terminal state is reached
  - Propagate the result again to update the value estimates of the internal nodes of the tree



Source: https://pt.wikipedia.org/wiki/M%C3%A9todo_de_Monte_Carlo



# Monte Carlo Tree Search

- The process is repeated many times, and the final decision is based on the child node of the root that has the best statistics (e.g. the highest win rate).

**Algorithm 1** Original MCTS

**function** MCTSEARCH($s_0$)
  create root node $v_0$ with state $s_0$.
  **while** within computational budget **do**
    $v_l \leftarrow$ *TreePolicy*($v_0$)
    $\Delta \leftarrow$ *Simulate*($s(v_l)$)
    *Backprog*($\Delta$, $v_l$)
  **return** *Bestchild*($v_0$)

# Deterministic Games

- Checkers:
  - **Chinook** ended the 40-year reign of human champion Marion Tinsley in 1994. It used a database for endgames defining the perfect way to win for all positions involving 8 or fewer pieces (a total of 443748401247 positions). Today it's a **solved problem**.
- Chess:
  - **Deep Blue** defeated human world champion **Gary Kasparov** in a 6-game match in 1997. Deep Blue searched 200 million positions per second and used an extremely sophisticated evaluation function and (undisclosed) methods to extend some search lines beyond depth 40!
- Go (2015):
  - Human champions refuse to compete with computers **because machines can't play reasonably (b>300)**
- Go (2017):
  - AlphaGo (Fan, Lee), AlphaGo Master, AlphaGo Zero and AlphaZero! Machines beat human and machine champions from previous generations 100-0.

Sorce: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

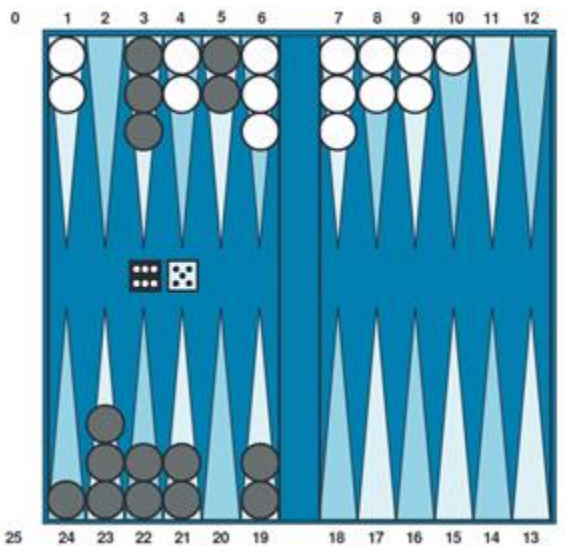
## Stochastic Games

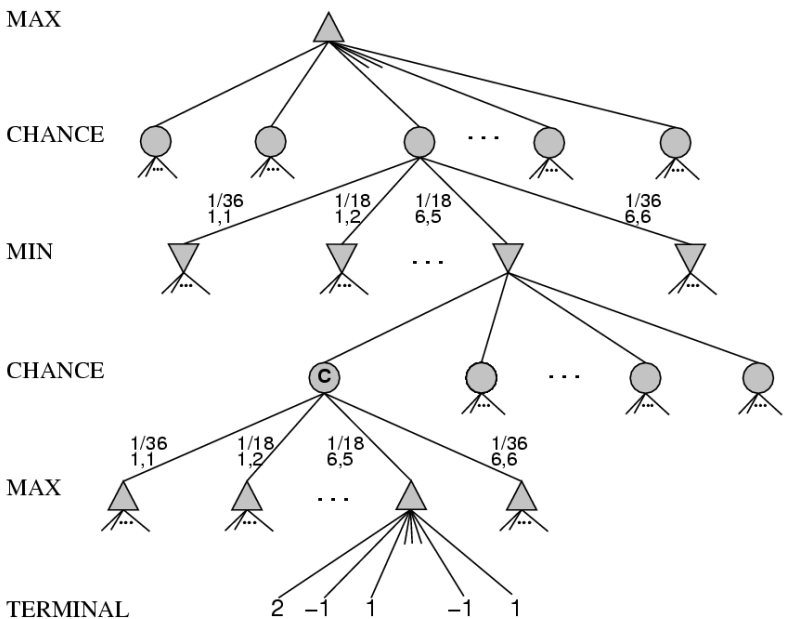
- In real life, there are many unpredictable external events that can put us in unforeseen situations. Many games reflect this unpredictability by including a random element (e.g. dice rolling).

| | **Deterministic** | **Stochastic** |
|---|---|---|
| Perfect information (fully observable) | Chess<br>Go,<br>Checkers | Backgammon,<br>Monopoly |
| Imperfect information (partially observable) | Battleship | Scrabble,<br>Poker,<br>Bridge |

# Stochastic Games

- Stochastic games are games that typically combine skill and luck;
- The search tree should include probability nodes;
- The decision is made based on the expected value
- ExpectiMiniMax algorithm.




Source: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

# Minimax vs. Expectiminimax

- **Minimax:**
  - **Maximize** (over all possible moves I can make) the
  - **Minimum** (over all possible moves Min can make) of the
  - Reward

$$Value(node) = \max_{my\ moves} \left( \min_{Min's\ moves} (Reward) \right)$$

- **Expectiminimax:**
  - **Maximize** (over all possible moves I can make) the
  - **Minimum** (over all possible moves Min can make) of the
  - **Expected** reward

$$Value(node) = \max_{my\ moves} \left( \min_{Min's\ moves} (\mathbb{E}[Reward]) \right)$$

$$\mathbb{E}[Reward] = \sum_{outcomes} Probability(outcome) \times Reward(outcome)$$

## Stochastic Games

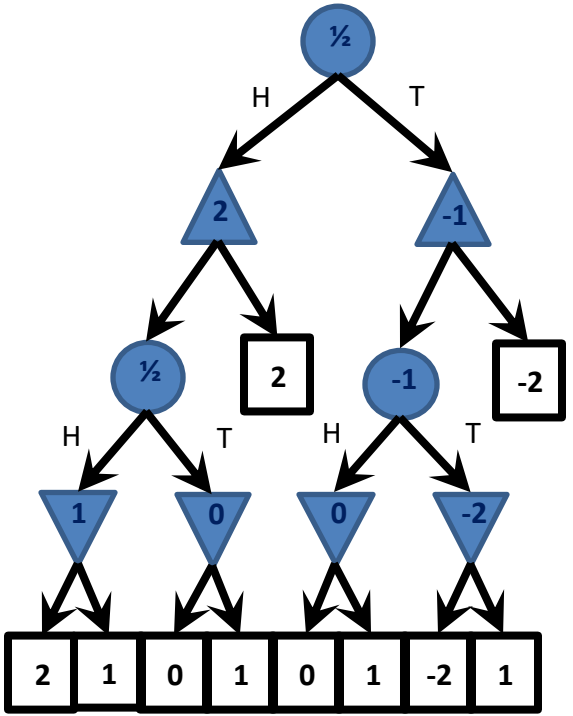
- **Expectiminimax:** for lucky nodes (chance) add up the values of the successor states weighted by the probability of each successor.

  - **Value**(*node*) =
    - Utility(*node*) if *node* is terminal
    - $\max_{action}$ **Value**(Succ(*node, action*)) if *type* = MAX
    - $\min_{action}$ **Value**(Succ(*node, action*)) if *type* = MIN
    - $\text{sum}_{action}$ P(Succ(*node, action*)) * **Value**(Succ(*node, action*)) if *type* = CHANCE

ISLab
Synthetic Intelligence Lab

# Stochastic Games Example

- ALEATORY: Max tosses a coin. It's either heads (H) or tails (T).
- MAX: Max stops or continues.
  - Stop on Heads: the game ends, Max wins(value = € 2).
  - Stop on Tails: the game ends, Max loses (value = € -2).
  - Continue: the game continues
- ALEATORY : Min tosses a coin.
  - Head-Head: value = € 2
  - Tail-Tail: value = € - 2
  - Head-Tail or Tail-Head: value = 0
- MIN: Min decides whether to keep the current result (value above) or pay a penalty (value = € 1).



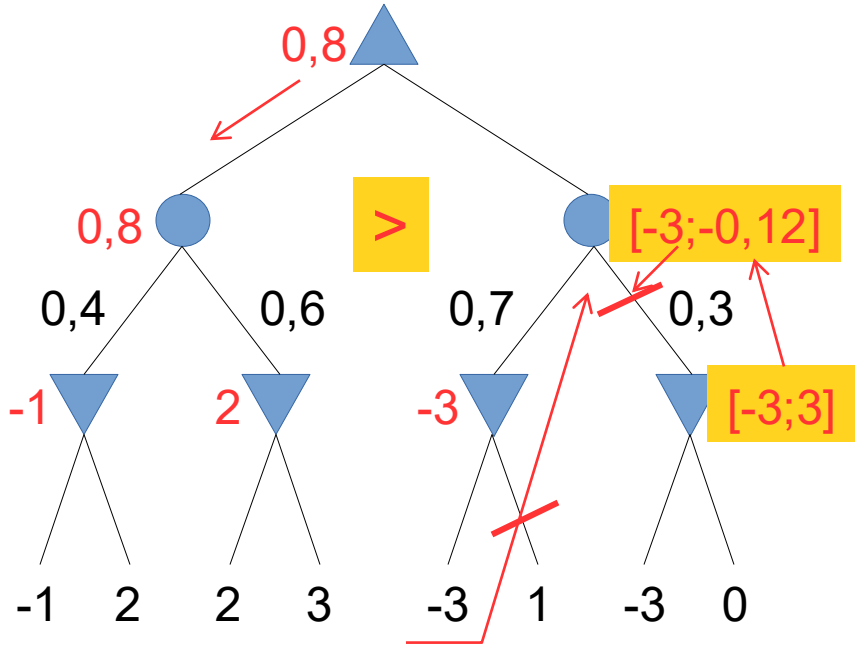
# Expectiminimax

- All the above methods are useful :
  - Alpha-beta pruning
  - Evaluation function
  - etc
- The computational complexity is very complicated
  - The branching factor of the random choice can be high
  - Twice as many "levels" in the tree

- In Expectiminimax: for us luck (Change), it adds up the values of the successor states weighted by the probability of each successor.
  - Branching factor can be very unpleasant, defining more difficult evaluation functions and pruning algorithms
- If necessary, use Monte Carlo Simulation: when you reach a lucky node, simulate a large number of games with random dice rolls and use the winning percentage as an evaluation function.
  - It can work well for games like backgammon.

ISLab
Synthetic Intelligence Lab

## Pruning in Stochastic Games




- If the gain ∈ [-3; 3]

limits can be calculated and exact pruning

ISLab
Synthetic Intelligence Lab

## Stochastic Games with imperfect (partial) information

Card games provide one of the best examples of stochastic games with imperfect (partial) information, where the unknown (imperfect) information comes from randomness.

Example: in many games, the cards are dealt randomly at the start of the game, each player receives a suit of cards that is neither visible nor known to the other players.

Games: Bridge, Hearts and Poker.

## Partially observable Games

The state of the game is not completely known

ex: poker, bridge, king,... (initial randomness and then incomplete information)

Solutions

- abstraction - considering different states as similar

what varies between them is of little relevance

- meta-reasoning - reasoning about whether or not it is worth reasoning more in certain states/ranges of the game

ISLab
Synthetic Intelligence Lab

## Miniminimax with imperfect information

- **Minimax:**
  - **Maximize** (over all possible moves I can make) the
  - **Minimum**
    - (over all possible states of the information I don't know,
    - … over all possible moves Min can make) the
  - Reward.

$$Value(node) = \max_{my\ moves}\left(\min_{\substack{missing\ info,\\ Min's\ moves}}(Reward)\right)$$

## Stochastic games with imperfect information Techniques

- If we know the probabilities of different configurations and want to maximize the average winnings (for example, if we can play the game many times): **expectiminimax** is used.
- If we have no idea of the probabilities of different configurations; or, if we can only play once and can't (and don't want to) lose: **miniminimax** is used.
- If the unknown information was intentionally selected by the opponent: **game theory** is used.

ISLab
Synthetic Intelligence Lab

## Conclusions

- Working with games is extremely interesting because it:
  - easy to test new ideas
  - easy to compare agents with other agents
  - easy to compare agents with humans
- The games illustrate several interesting points about AI:
  - Perfection is unattainable => you have to get closer!
  - It's a good idea to think about what to think about
  - Uncertainty restricts the assignment of values to states

- Games are to AI as Formula 1 is to car construction…

Fonte: Luís Paulo Reis, Artificial Intelligence (2019), Universidade do Porto

# References

Recommended Bibliography

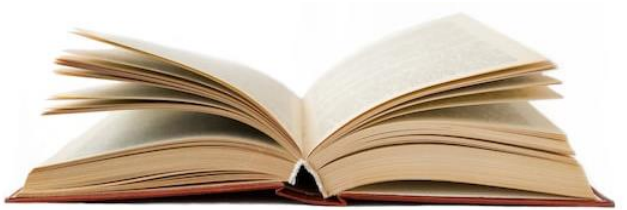

- Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594, Chapter 5.


Other Material:


- Svetlana Lazebnik, Lecture notes Fall 2017 Artificial Intelligence, University of Illinois.
- Luís Paulo Reis, Lecture notes Artificial Intelligence (2019), Universidade do Porto


For those who want to go further


**AlphaGo**: Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

# PROBLEM-SOLVING AND SEARCH METHODS

## Search in competitive contexts

LICENCIATURA EM ENGENHARIA INFORMÁTICA
MESTRADO integrado EM ENGENHARIA INFORMÁTICA
Inteligência Artificial
2025/26