AI in Games: Basic Concepts and Foundations

Course Notes 2

Nadjib Lazaar
Université Paris-Saclay
lazaar@lisn.fr

Why Games?

Games are a privileged domain for AI because they provide a formal, structured, and measurable framework for evaluating the intelligence and effectiveness of algorithms. By modeling competitive environments with precise rules, games offer an ideal experimental ground for developing agents capable of planning and acting optimally. These agents must analyze situations, anticipate adversarial actions, and make decisions that maximize their chances of success, relying on appropriate strategies and advanced algorithmic techniques.

1 Game and Intelligent Agent

A **two-player game** is a competitive game where two players alternate their actions at each turn. Each player seeks to maximize their own gain while minimizing the other player's gain. These games are often **zero-sum**, meaning that one player's gain is exactly the other player's loss. In other words, the sum of both players' gains is constant and equal to zero at each terminal state of the game.

Two-Player Game

A two-player game $\mathcal{J}$ is defined by a quintuple $\langle \mathcal{P}, S, A, T, U \rangle$, where:

- S: The set of **states** of the game, representing all possible configurations of the game system.
- A: The set of **actions** available to each player. Each player chooses an action at each turn, depending on the current state.
- $T : S \times A \to S$: The **transition function**, which determines the next state based on the current state and the action chosen by each player.
- $U : S \to \mathbb{R}^2$: The **utility function**, which associates with each terminal state $s \in S$ a pair of values (u_1, u_2), representing the gains of players p_1 and p_2, respectively.
- $\mathcal{P} = \{p_1, p_2\}$: The set of **players**, where each player has an objective opposite to the other's (maximizing utility for each player).

In this type of game, each player's objective is to maximize their own utility $U_1(s)$ for p_1, or $U_2(s)$ for p_2, while taking into account the opponent's actions. The players alternate turns: player p_1 chooses an action each time, then p_2 reacts, and so on until the game reaches a terminal state.

An **intelligent agent** is an autonomous entity that perceives its environment and makes decisions to achieve a given goal, based on its perceptions.

Intelligent Agent

Let $\mathcal{J} = \langle \mathcal{P}, S, A, T, U \rangle$ be a game, where The agent $\mathcal{A}$ uses a **perception function** $f : S \to O$, which allows it to perceive its environment and gather information about the current state s. The **reward function** $r : S \to \mathbb{R}$ associates a value to each state s, representing the quality of that state for the agent.
The agent chooses its actions $a \in A$ based on its perceptions $o \in O$ in order to maximize a utility or reward function over a given time horizon. This action is determined by a policy $\pi : O^* \to A$, which specifies the action to choose based on past observations.

In a competitive game, the intelligent agent seeks to maximize its own gain while considering the opponent's actions. The agent adopts a strategy that aims to approach the **oracle** — that is, the optimal strategy that would maximize its reward assuming it knows the environment and the opponent's actions perfectly. The agent adjusts its choices based on the history of its perceptions, attempting to imitate or approximate the optimal behavior in the game.

The agent's performance is measured by the sum of the rewards received over a sequence of states $s_0, s_1, \ldots, s_t$:

$$\max_\pi \mathbb{E}\left[\sum_{t=0}^{T} r(s_t)\right]$$

where $\mathbb{E}$ represents the expected value and T is the planning horizon. The agent can also use learning strategies to approach the optimal strategy, adjusting its actions over time based on accumulated experience.

2 Game Tree

A **game tree** is a structure that models all possible configurations of a game and the decisions made by players at each step. In a two-player game framework, the game tree represents the alternation of players' actions, starting from an initial state and exploring all possible sequences of moves until a terminal state is reached.

Game Tree

Let a two-player game $\mathcal{J} = \langle \mathcal{P}, S, A, T, U \rangle$. The associated game tree is a tree-like structure defined as follows:

- The **nodes** of the tree represent the **states** of the game $s \in S$,
- The **edges** between nodes represent the **actions** $a \in A$ performed by the players,
- The **root** of the tree corresponds to the initial state s_0,
- The **leaves** of the tree represent the **terminal states** where the game ends.

Example: Placing Dominos on an $n \times n$ Board

Consider an $n \times n$ board where players take turns placing a domino: one places them horizontally, while the other places them vertically. Each domino occupies exactly two adjacent squares. The game ends when no more dominos can be placed. The player who makes the last move wins the game. Figure 1 illustrates the game tree for a 3×3 board.

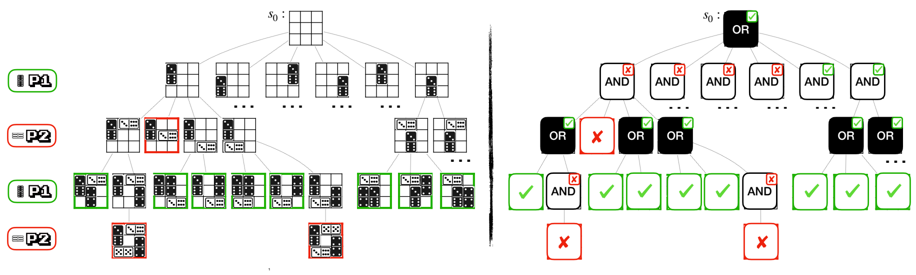

Figure 1: Domino game tree (3 × 3). On the left is the game tree, while on the right is the AND/OR tree. Note that the nodes with ellipses at the bottom are **symmetrical** to nodes that have already been expanded. AND nodes correspond to Player 1's decisions, who seeks to guarantee victory by maximizing their options. OR nodes represent Player 2's decisions, who aims to minimize Player 1's chances of winning. The leaves of the tree are labeled "Victory" or "Defeat" depending on the outcome of the game for Player 1.

Question: Finding the Winning Strategy

After exploring the complete game tree for domino placement on a board, think about the optimal strategy for the starting player.

1. By consulting the complete game tree, what strategy guarantees a win for player 1, regardless of player 2's behavior?
2. Identify critical configurations where a bad choice could lead to a loss.

This reflection will help better understand how an intelligent agent can analyze a game tree and adopt an optimal strategy based on possible configurations.

2.1 Winning Strategy for Player 1

In a game tree, each player's decision influences the future state of the game. To determine the winning strategy, an AND/OR tree is particularly useful, as it models both players' decisions, where:

- AND nodes correspond to player 1's decisions, who seeks to maximize their gain.
- OR nodes correspond to player 2's decisions, who seeks to minimize player 1's gain.

The AND/OR tree represents an alternation between these two types of nodes and helps determine the best strategy for each player.

Here's how to determine the winning strategy by following this game tree through the following steps:

1. **Analyze the leaves:** The leaves of the tree represent the terminal states of the game. Each leaf is labeled "Player-1" (player 1 wins) or "Player-2" (player 2 wins), depending on the game's outcome. These final states determine the win or loss.
2. **Propagate values up the nodes:**
   - For an **OR node**, representing player 2's choice, it is assumed that player 2 seeks to minimize player 1's chances. They will choose the branch with the minimum value (i.e., the branch leading to a "Loss" for player 1 if possible).
   - For an **AND node**, representing player 1's choice, player 1 seeks to maximize their chances of winning. They will choose the branch that leads to "Victory" if available, ensuring that all branches lead to favorable outcomes for them.
3. **Root (AND node):** As we move up the tree, we reach the root node, which represents the initial game state. If this node can be evaluated as "Victory" considering both players' choices and available branches, player 1 has a winning strategy. If the root is evaluated as "Loss," player 2 can force a loss for player 1, regardless of player 1's strategy. Figure 1 illustrates the AND/OR tree for the domino game.

2.2 From the Winning Strategy to a Policy to Follow

The winning strategy, determined from the game tree, represents the set of optimal decisions that allow player 1 to guarantee a victory, regardless of the opponent's responses. To make this strategy practically usable, it is transformed into a **game policy**, which associates each state s with the optimal action a .

A game policy acts as a lookup table or a simple rule to follow:

- On each turn, the player observes the current state of the game.
- The player consults the policy to determine the optimal action based on this state.
- The player executes this action, while anticipating the opponent's countermoves.

The policy ensures that the player stays on the favorable paths identified in the tree. In the case of an opponent's error (the opponent chooses a non-optimal action), the policy can be adjusted locally to exploit new opportunities. Thus, the winning strategy translates into a clear and exploitable plan to maximize the chances of victory.

2.3 Complexity of Game Trees

Game trees suffer from combinatorial explosion, which is an exponential increase in the size of the tree as the branching factor and depth increase. Generally, the size of the tree depends on two main parameters:

- n, the **branching factor**, which corresponds to the number of possible choices (actions) from a given state.
- d, the **depth**, which corresponds to the maximum number of levels in the tree. This is often related to the total number of actions required before reaching a terminal state.

The size of the tree in the worst case is given by the following complexity:

$$O(n^d)$$

This exponential behavior quickly makes exhaustive exploration impossible as n or d increases.

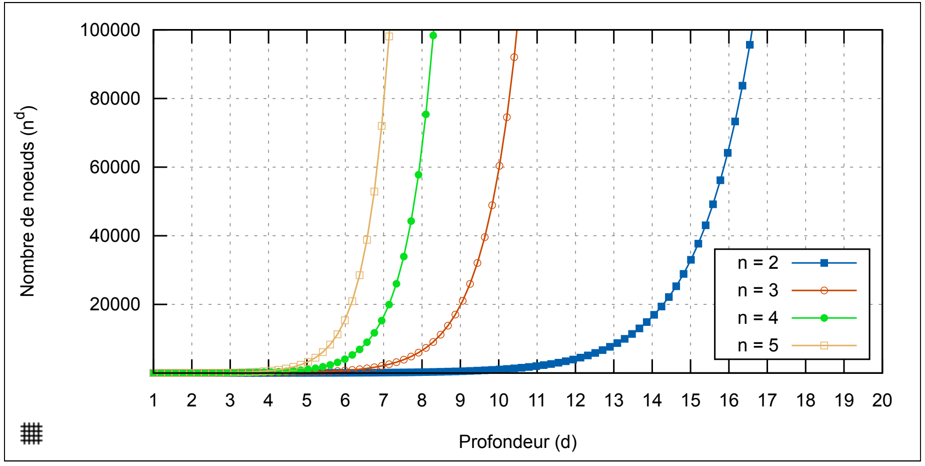

Figure 2: Combinatorial explosion in a game tree with $n \in [2, 5]$.

Reducing Complexity

To limit combinatorial explosion, optimization techniques such as alpha-beta pruning are used. These methods allow:

- Ignoring unnecessary branches that cannot influence the final decision.
- Reducing the effective depth by applying heuristics or limits to the search.

Thus, while the theoretical complexity remains exponential, these optimizations make exploration more feasible in practical cases.

2.4 Introduction of Heuristics

Complete exploration of a game tree often suffers from combinatorial explosion. One optimization is to transform the tree into a graph by avoiding re-exploring common subgraphs, which requires storing already explored nodes. However, this approach can become costly in terms of time and memory, especially for complex games. Another improvement is to introduce **intelligent pruning**: if a branch guarantees a victory (Player 1) or a loss (Player 2), neighboring branches are ignored. This reduces the size of the tree while maintaining the properties of complete exploration.

To go further, the exploration is limited to a **maximum depth p** , where the leaves of the tree are evaluated using a **heuristic**. A heuristic is a static function that assigns a numerical value to a game board, reflecting its quality for a player. It is positive when the state is close to victory, negative when it is close to defeat, and consistent with the actual scores in terminal configurations. In practice, the heuristic helps guide the exploration by prioritizing promising branches, thus reducing complexity without exploring the entire tree.