Adversarial Search: Games

In this type of search we have more than one agent, these agents compete for different goals.

In other to classify games we need to follow these criteria:

- Number of players. It can be a game of 2 players or more (multiplayer).
- Properties of the **utility function**:
  - Zero-sum: The sum of the utilities of the agents is zero regardless of the outcome of the game.
  - Constant sum: The sum of the utilities of the agent is constant, regardless of the outcome of the game.
  - Variable sum: The utilities are not zero sum. They can have a complex optimal strategy where sometimes collaboration is involved.
- **Information** available for the players. We can have **perfect information** which means that we know everything about the game, or **partial information** like cards game.
- **Chance** elements. Can be **Deterministic** or **Stochastic**.

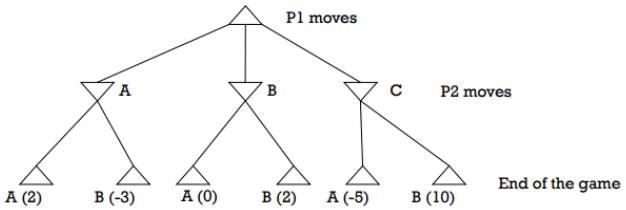
In adversarial search a game tree is an explicit representation of all possible sequences of moves in a game. In this tree the leaves correspond to terminal states of the game.

Representation of games

❑Matrix of final balances: Value of the utility function for each player given the actions of the players

		P2	
		A	B
P1	A	2	-3
	B	0	2
	C	-5	10

❑Game tree



As we can see the players alternate movements. Let's consider now that p1 is **MAX** and p2 is **MIN**. The optimal strategy to perform as good as an infallible problem is the **minimax strategy**.

The strategy consists in using the **minimax value** of a node to guide the search, this value is the utility of a node (from the point of view of MAX) and we are always assuming that both players play optimally.

The minimax algorithm is **complete** only if the tree game is **finite**, it is **optimal** only if the opponent is optimal, it has an **exponential** temporal complexity **$O(b^m)$** and the spatial complexity is **linear** if depth-first search is used.

The first turn is for the MAX, the MAX nodes are the ones where the MAX has the turn, those nodes are the ones located at even depth.

A terminal node is assigned a utility value from the point of view of MAX.

The minimax value of a node is obtained by propagating the values of the utility function corresponding to the terminal nodes towards the root of the tree:

So, the minimax value of the root corresponds to the best value of the utility function that MAX can achieve in the game.

There are some **problems**, usually the utility function **is too expensive** to compute and we have **limited resources**. To solve that we use **limited horizon search**: We define a heuristic evaluation function, **eval(n)**, which is an estimate of the actual utility function. We use a **cutoff test** to determine when to stop the search and compute **eval(n)**.

We also use **evaluation functions**; these functions must return the same value as the utility function if we are evaluating a leaf node. In non-leaf nodes the value is an estimate of the minimax value.

Alpha-beta pruning is a way of performing minimax. Each node has **[α, β]**.

- **α** is the value of the best alternative for MAX found so far, it is a **lower bound** for the minimax value at max. We stop the search of this value on a MIN node whose **β ≤ α** of any of its ancestors.
- **β** is the value of the best alternative for MIN found so far, it is an **upper bound** for the minimax value at min. We stop the search of this value on a MAX node whose **α ≤ β** of any of its ancestors.