CS4000 Intelligent Systems

Games and Adversarial Search

Dr. Santiago Conant P.
sconant@tec.mx

Content

Why talk about games?

- Historical reasons
- Requiring intelligence
- Ease of representation
- Accessible environment
- Simple rules

Chess

It has received great attention since the beginning of artificial intelligence.

Current programs on modern computers can defeat most chess masters in tournament situations.

In timed chess the computer Deep Blue was able to defeat the current world champion (Gary Kasparov) in 1997:

- Program written in C.
- 200 million positions per second.
- 25th Super-computer more powerful then.
- 11.38 gigaflops.

In more recent times (May 2008) Rybka was the best ranked chess player system.

Checkers

In 1952, Arthur Samuel of IBM developed a program capable of learning to play when competing against humans.

Jonathan Schaeffer created the Chinook program that managed to beat the world champion in the official 1994 tournament.

Chinook is currently invincible.

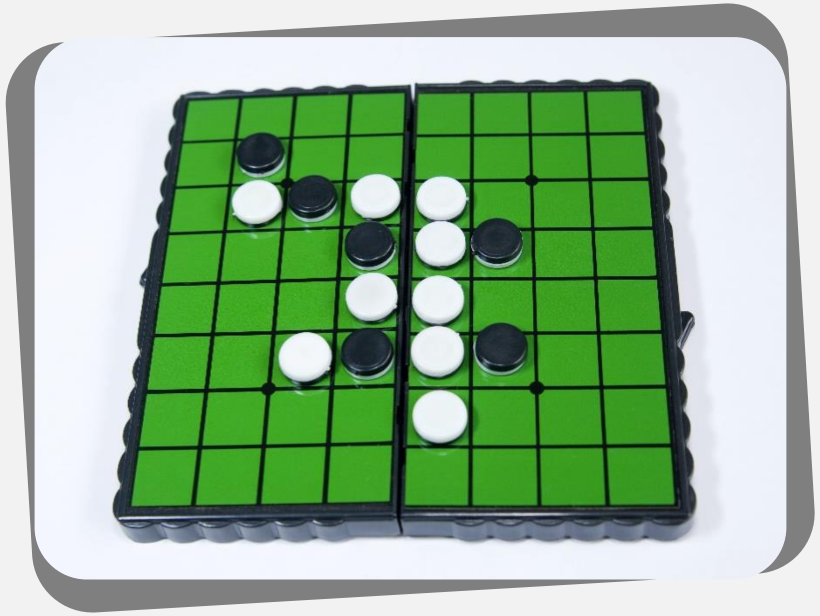
- It guarantees not to lose even if it plays against an infallible player.

Othello (Reversi)

It is better known as a computer game than a board game.

In 1997 the program Logistello defeated the current world champion (Takeshi Murakami) six games to zero.

It is recognized that computers in Othello are much better than humans.



Go

- Popular game in Asia that is played on a 19x19 board.
- Its branching factor starts at 361, so regular search methods fail.
- Rules based on knowledge-based systems to suggest successful moves are more successful but continue playing poorly.
- For 40 years computers had played at the level of a human rookie.
- In August 2008 the Huygens supercomputer with the MoGo Titan program defeated the first professional human player in an official match.

Intelligent Agent

Task environment

- Perfect Information (***fully observable***).
- Games without elements of chance where states depend on the moves of both adversaries (***strategic***).
- Success depends on every decision of the players (***sequential***).
- Finite number of possible moves and game settings (***discrete***).
- Two players (***multi-agent***).
- Players interleaved turns.

Type of agent

- **Utility-based agent:** Each player tries to win the game and maximize their profits.
- **Zero-sum games:** The utilities at the end of the game are the same and opposite.

Games like search

Goal

- Decide on each turn the best play to make.

Strategy

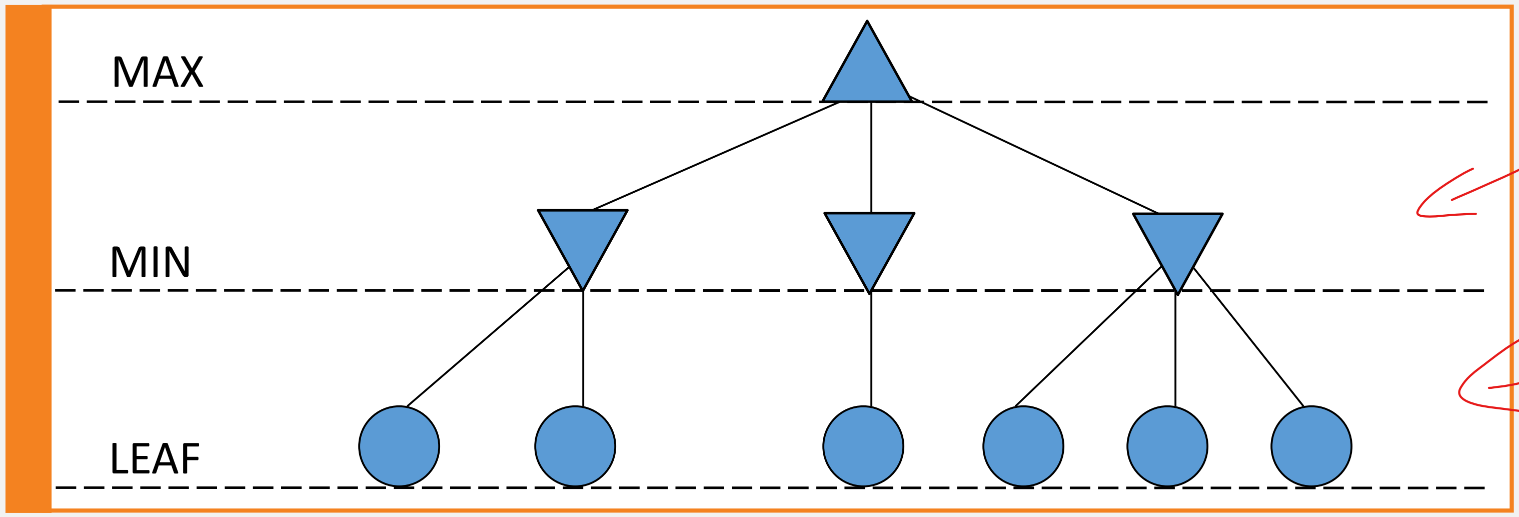
- Look ahead technique that analyzes several forward moves before deciding the best move for the current state.

Formulation

- **States:** Board and who moves.
- **Successor function:** Legal movements and generated states.
- **Goal test:** End of game.
- **Utility function:** It depends on the game.

Search space (Minimax)

- **Two players:** MAX (commonly the computer) and MIN (its opponent).
- Alternating movements starting with MAX.
- Search space = game tree (**contingent strategy**).



Actions by Max

Actions by Min

Motivation of the MINIMAX search

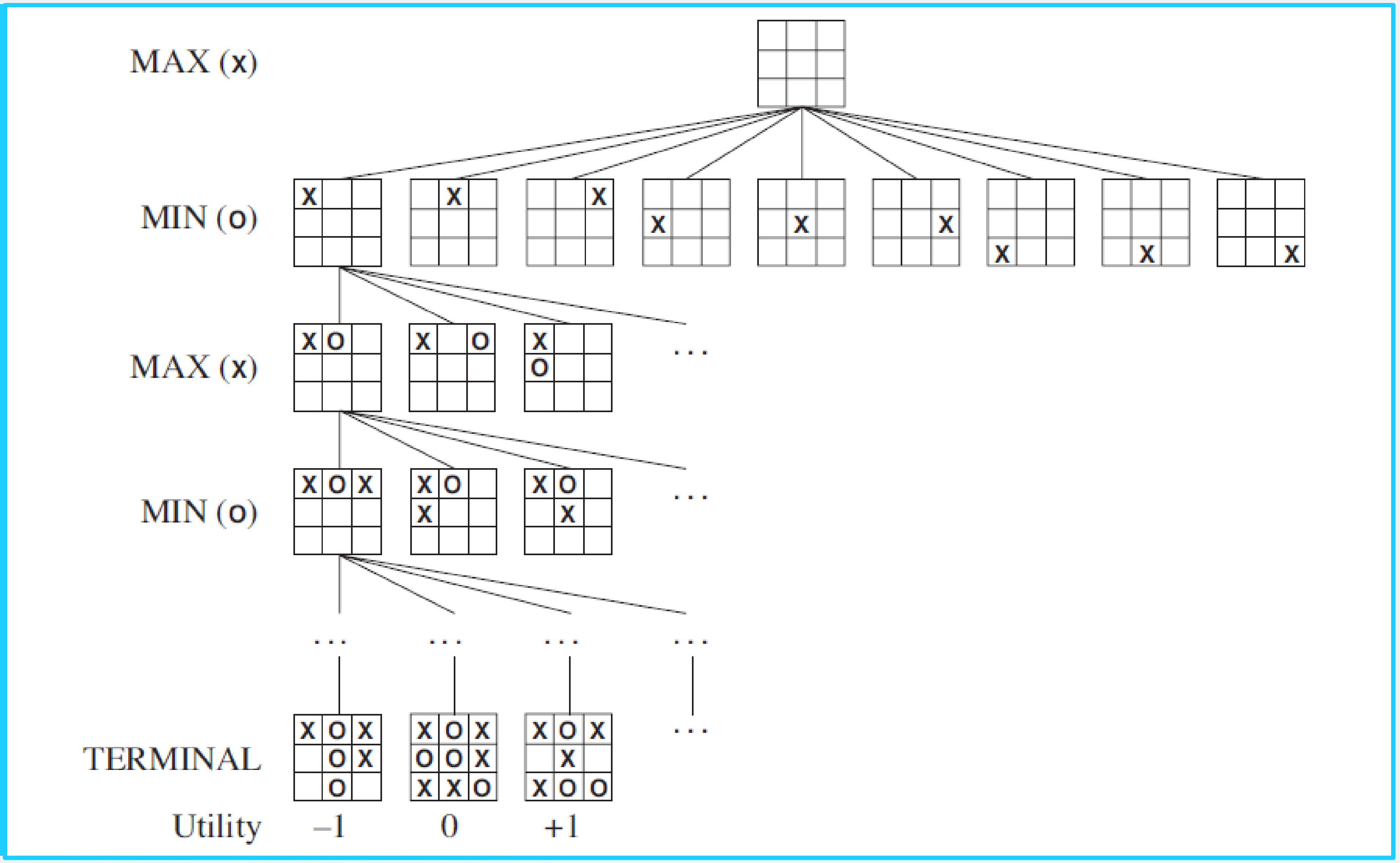
Adversarial search problem:

The player (MAX) can not control what his opponent does (MIN).

Modeling the opponent:

Infallible opponent (MIN always chooses the movement that is least suited to MAX).

Example: Tic-tac-toe



Minimax algorithm

1. Generate the complete tree of the game.
2. Apply the utility function to each terminal state to obtain its value.
3. Use the utility of terminal states to determine the utility of nodes that are one level higher in the tree (**Minimax value**).
4. Continue to get the values of the internal nodes towards the root of the tree, one level at a time.
5. Eventually, the values reach the top of the tree. At that point, MAX chooses the move that takes the highest value (**Minimax decision**).

Optimal strategy: MINIMAX algorithm

- **Terminal nodes** ($TERMINAL\_TEST(s)$)

$$Utility(s)$$

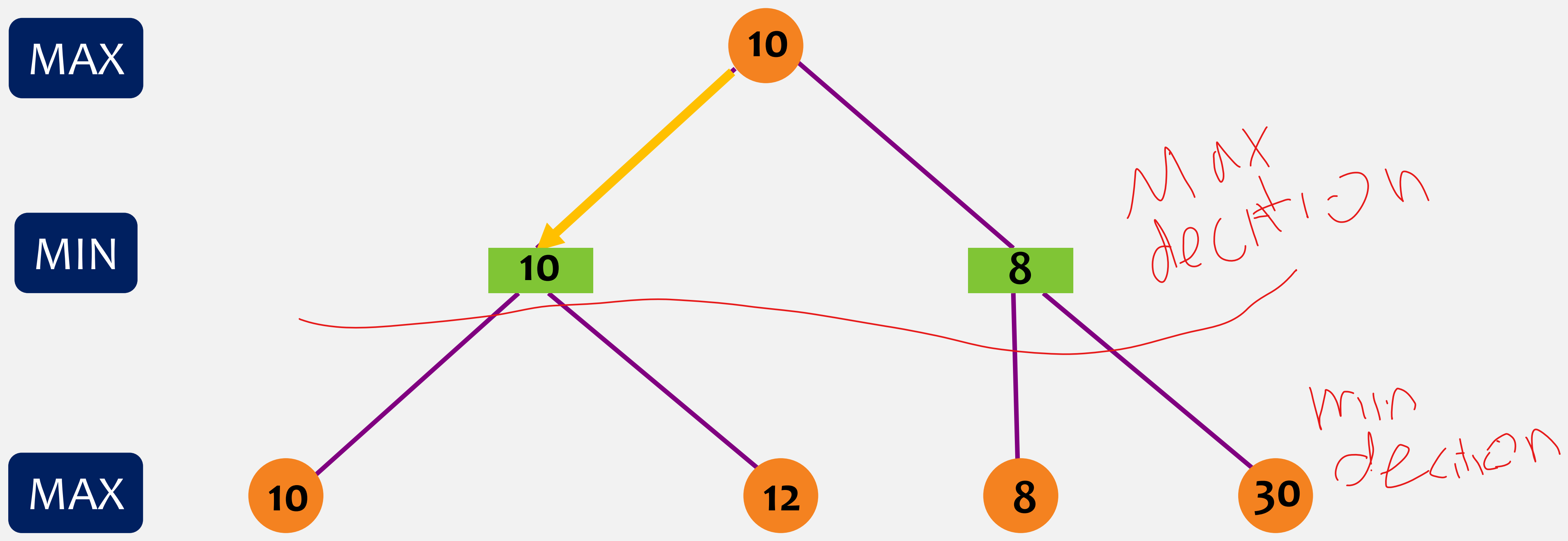
- **Internal nodes**
  - **MAX player** ($PLAYER(s) = MAX$)

$$max_{a \in Actions(s)} MINIMAX\big(RESULT(s, a)\big)$$

  - **MIN player** ($PLAYER(s) = MIN$)

$$min_{a \in Actions(s)} MINIMAX\big(RESULT(s, a)\big)$$

Example



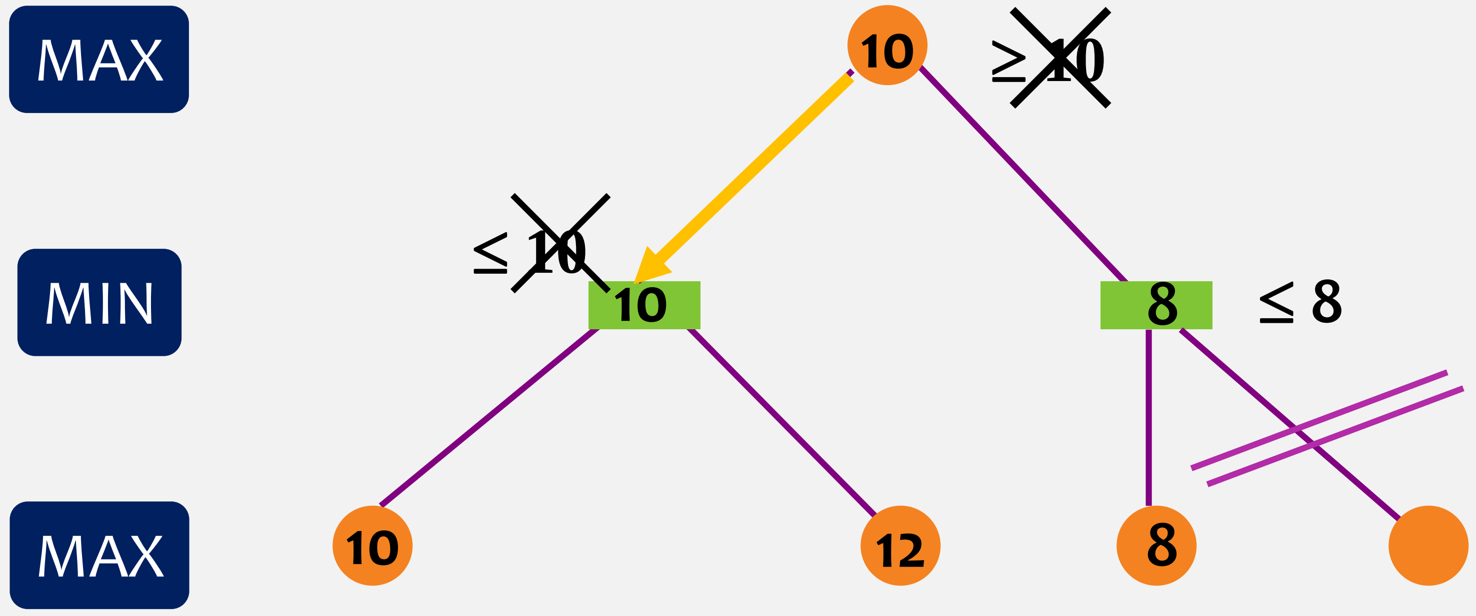
Minimax optimization (α–β pruning)

- **Idea:** Keep a record of partial evaluations of the nodes.
- **Variables:**
  - **v** = Minimax value.
  - **α** = Best evaluation found so far following the strategy of MAX.
  - **β** = Best evaluation found so far following the strategy of MIN.

- α and β are modified as the algorithm proceeds.
  - α only modified in the MAX nodes.
  - β only modified in the MIN nodes.

If, while adjusting the values of a node, α equals or exceeds β, the rest of its branches are "pruned" .

Example



Minimax with imperfect decisions

- **Main problem:** Size of the search space.
  - Tic-tac-toe: 362880
  - Othello: 3^{60}
  - Checkers: 5×10^{20}
  - Chess: 10^{154}

- **Solution:** Cut the search before reaching the Goal state => Define a level of exploration.

- **Effect:** replace the *utility function* by a ***heuristic evaluation function*** (estimation) for leaf nodes.

Heuristic evaluation of MINIMAX nodes

- **Leaf nodes ($CUTOFF\_TEST(s, dept)$)**

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

where w_i is a weight and f_i is a feature of the state.

- **Internal nodes**
  - **MAX player ($PLAYER(s) = MAX$)**

$$max_{a \in Actions(s)} H\_MINIMAX(RESULT(s, a), d + 1)$$

  - **MIN player ($PLAYER(s) = MIN$)**

$$min_{a \in Actions(s)} H\_MINIMAX(RESULT(s, a), d + 1)$$

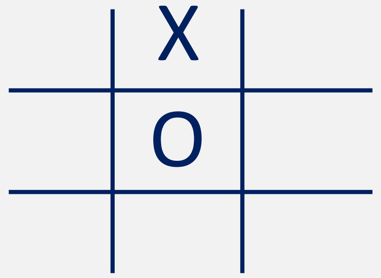
FEV requirements

- Evaluate boards in a "static“ way. Same state = same evaluation.
- Consider positive (in favor of MAX) and negative (in favor of MIN) aspects.
- Sorts the terminal states as you would sort the utility function.
- Reflect the chances of winning or losing according to the evaluated state.
- Cheap computational cost.

Example: fev for TIC-TAC-TOE game

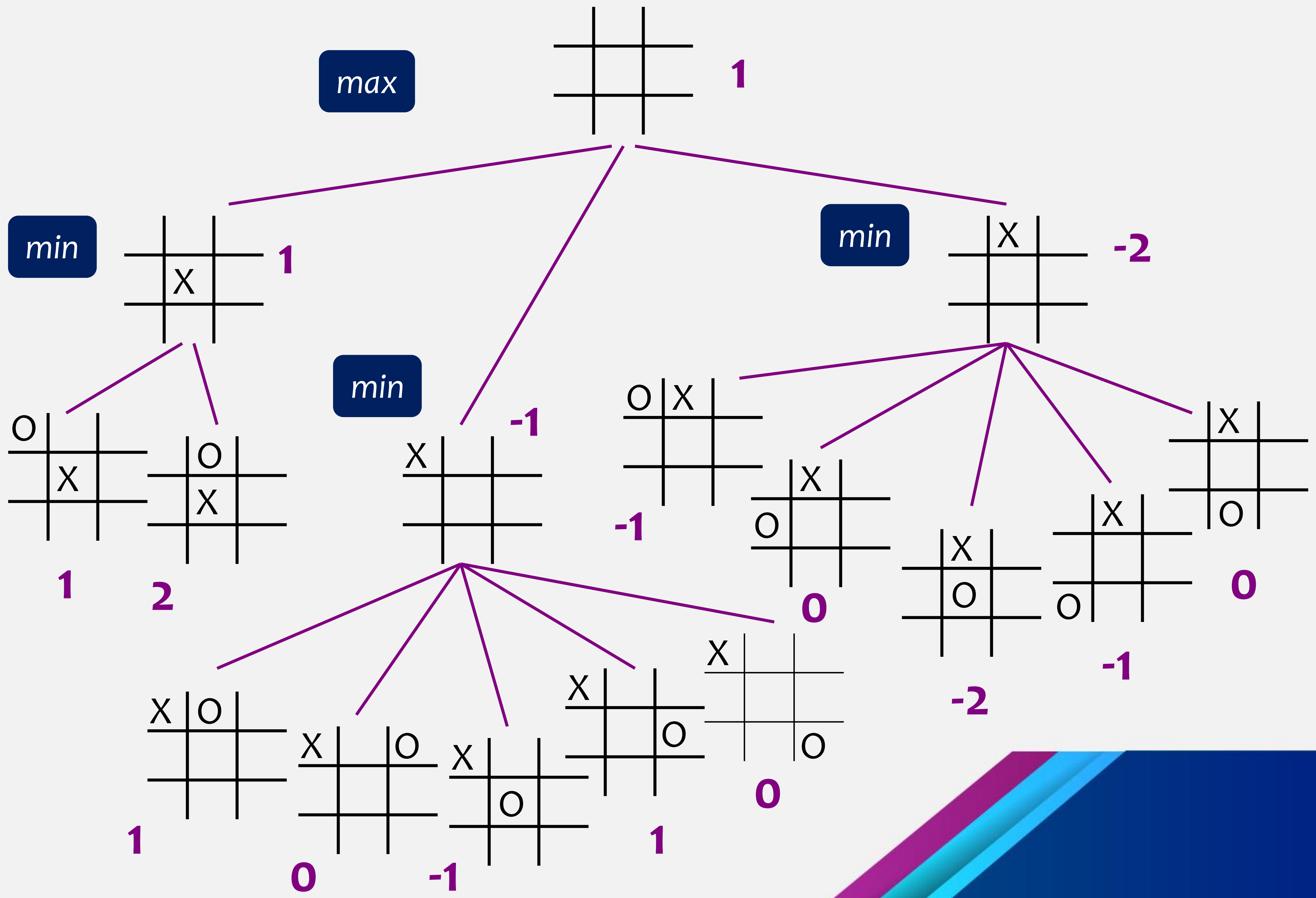
Count rows and columns where players can still win.

$$fev_x(n) = (FreeRow_x(n) + FreeCol_x(n) + FreeDiag_x(n)) - (FreeRow_o(n) + FreeCol_o(n) + FreeDiag_o(n))$$

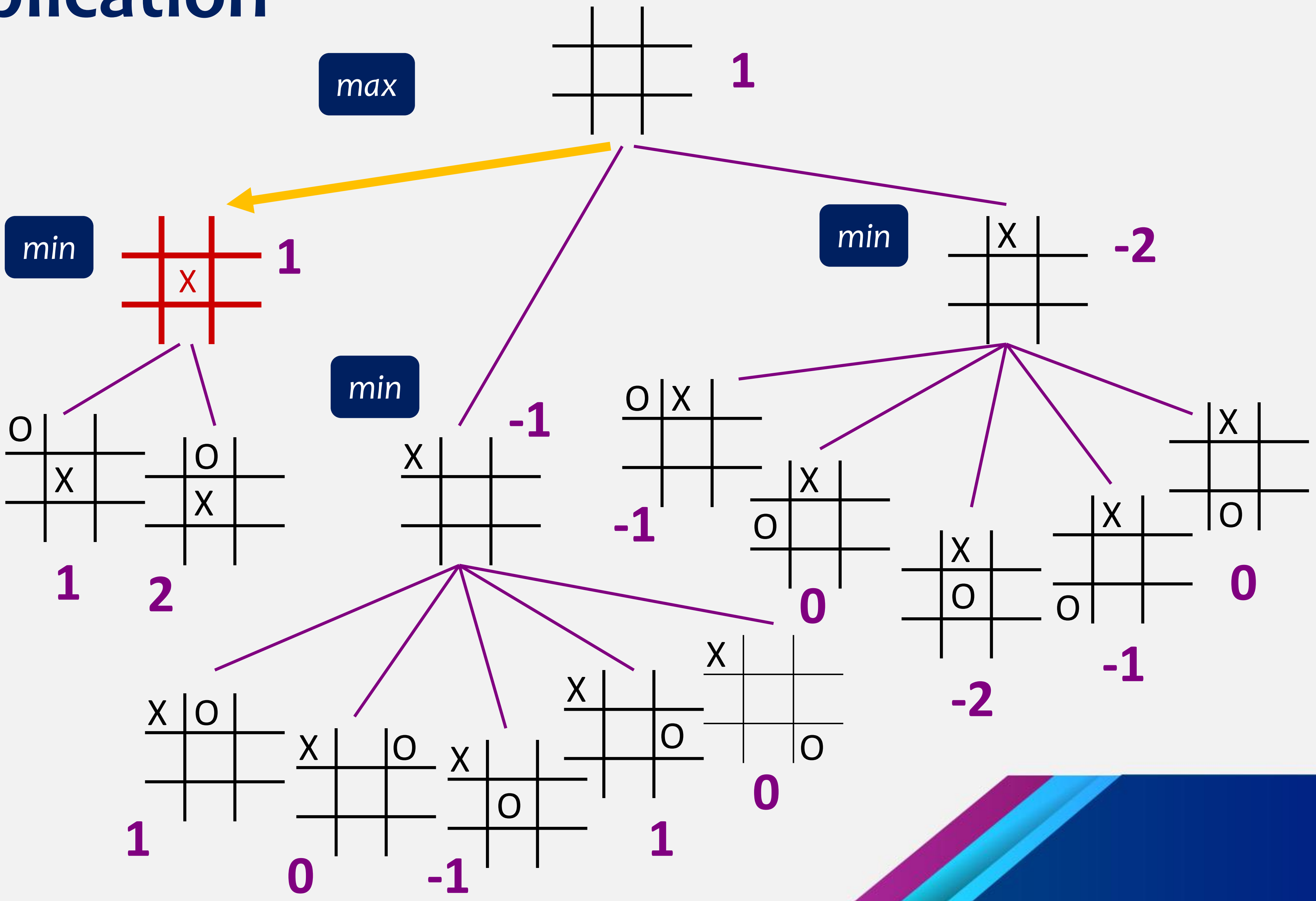


$$fev_x(n) = (2 + 2 + 0) - (2 + 2 + 2) = -2$$

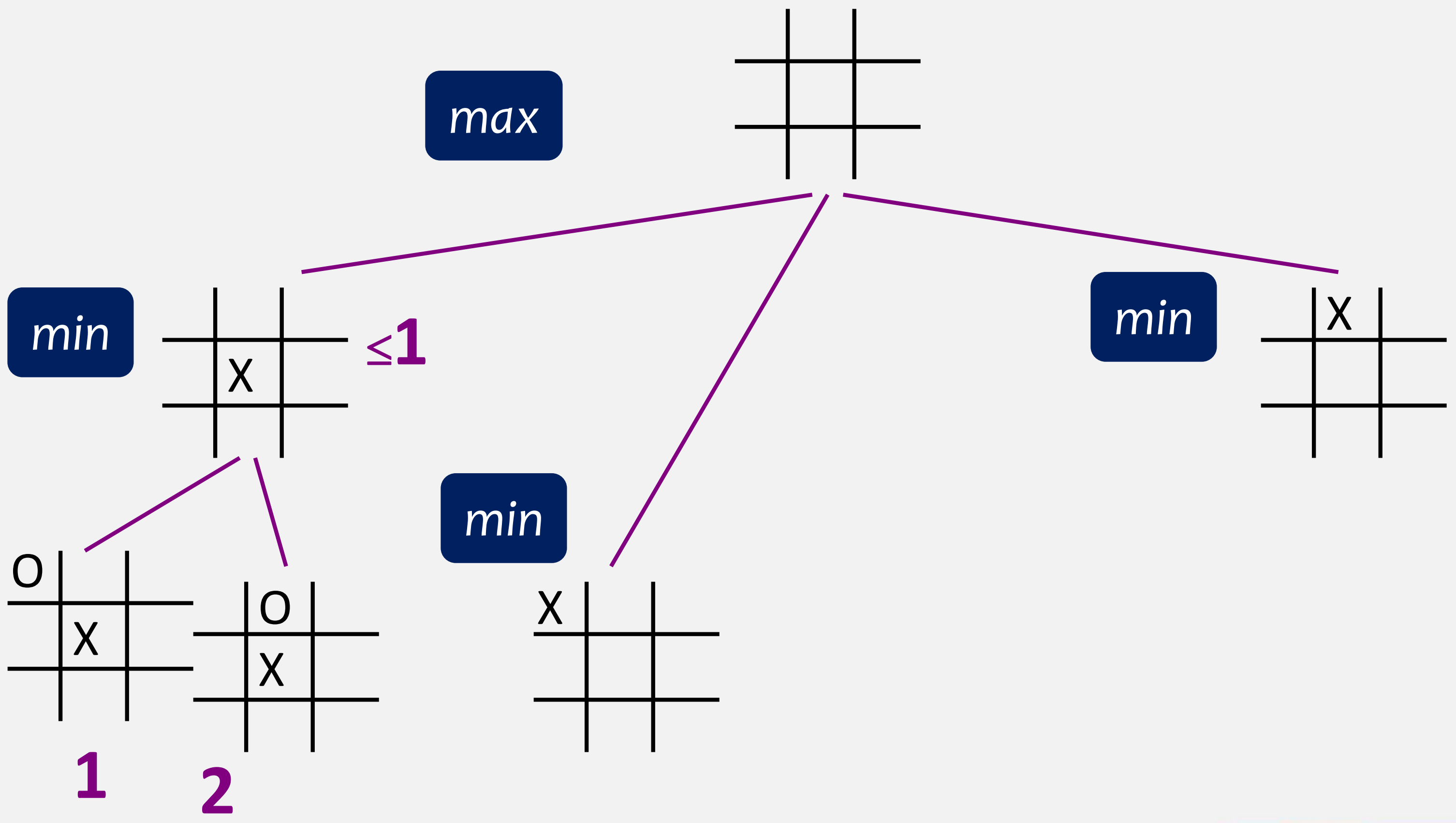
Application of Minimax



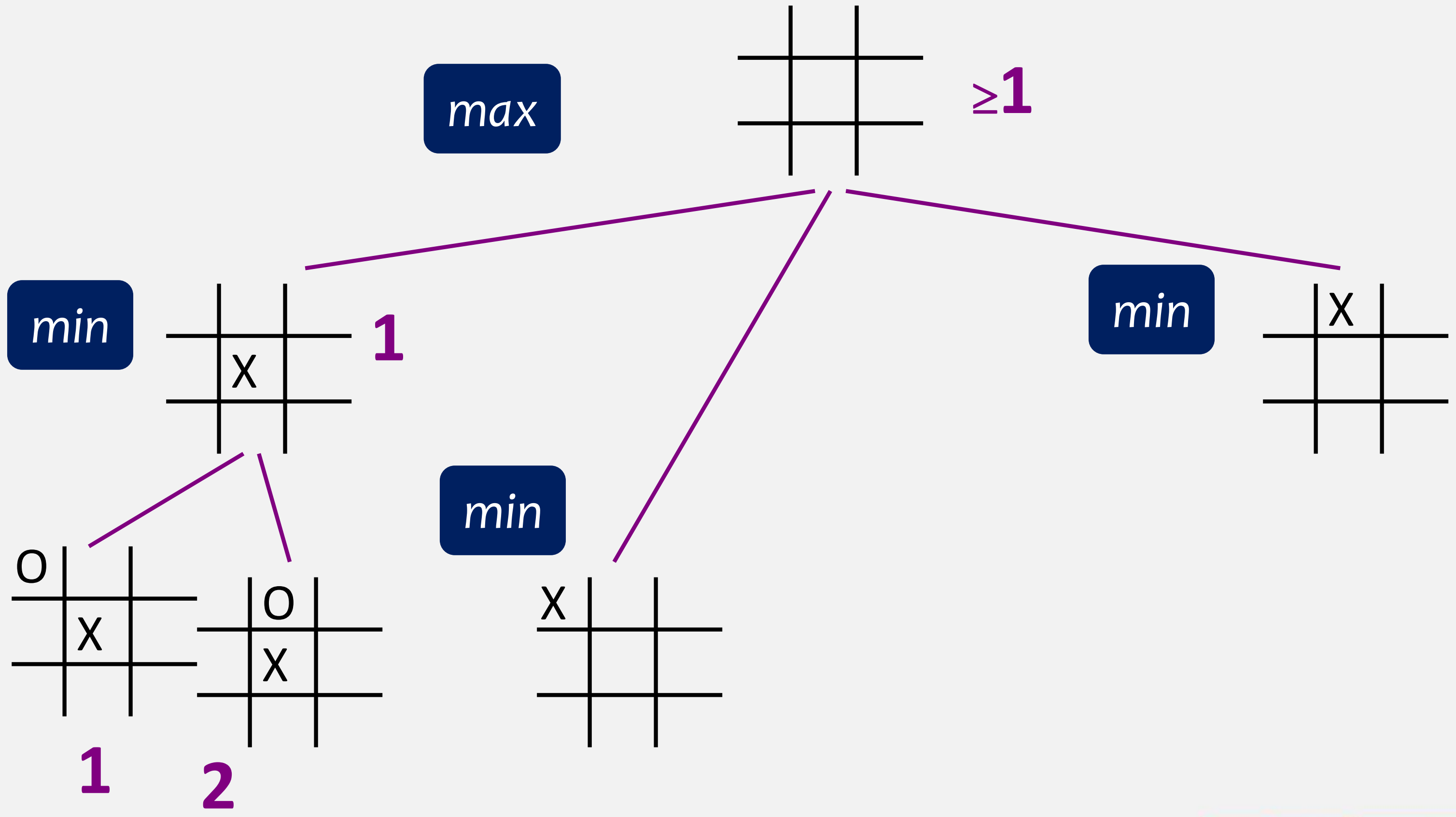
Application



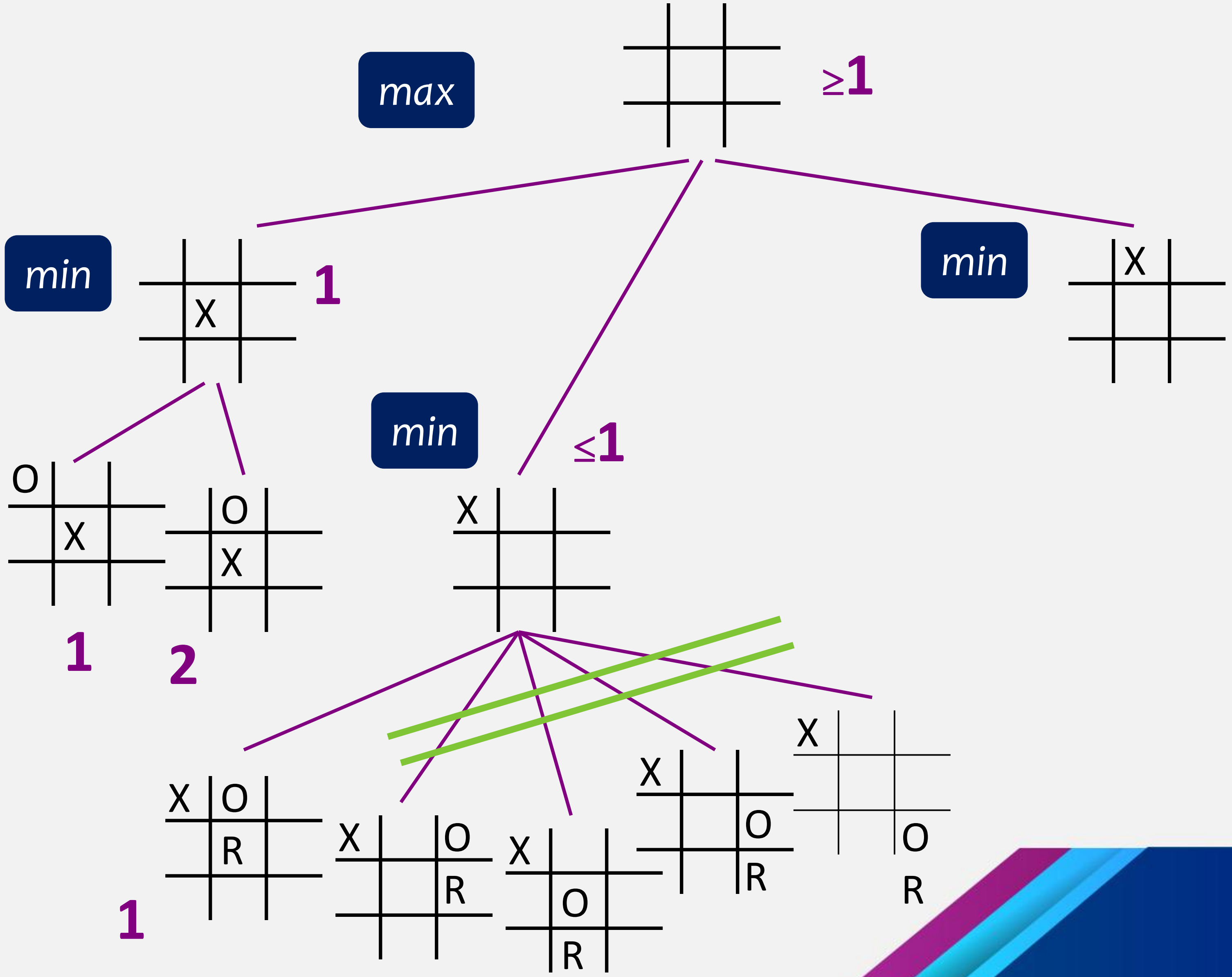
Application



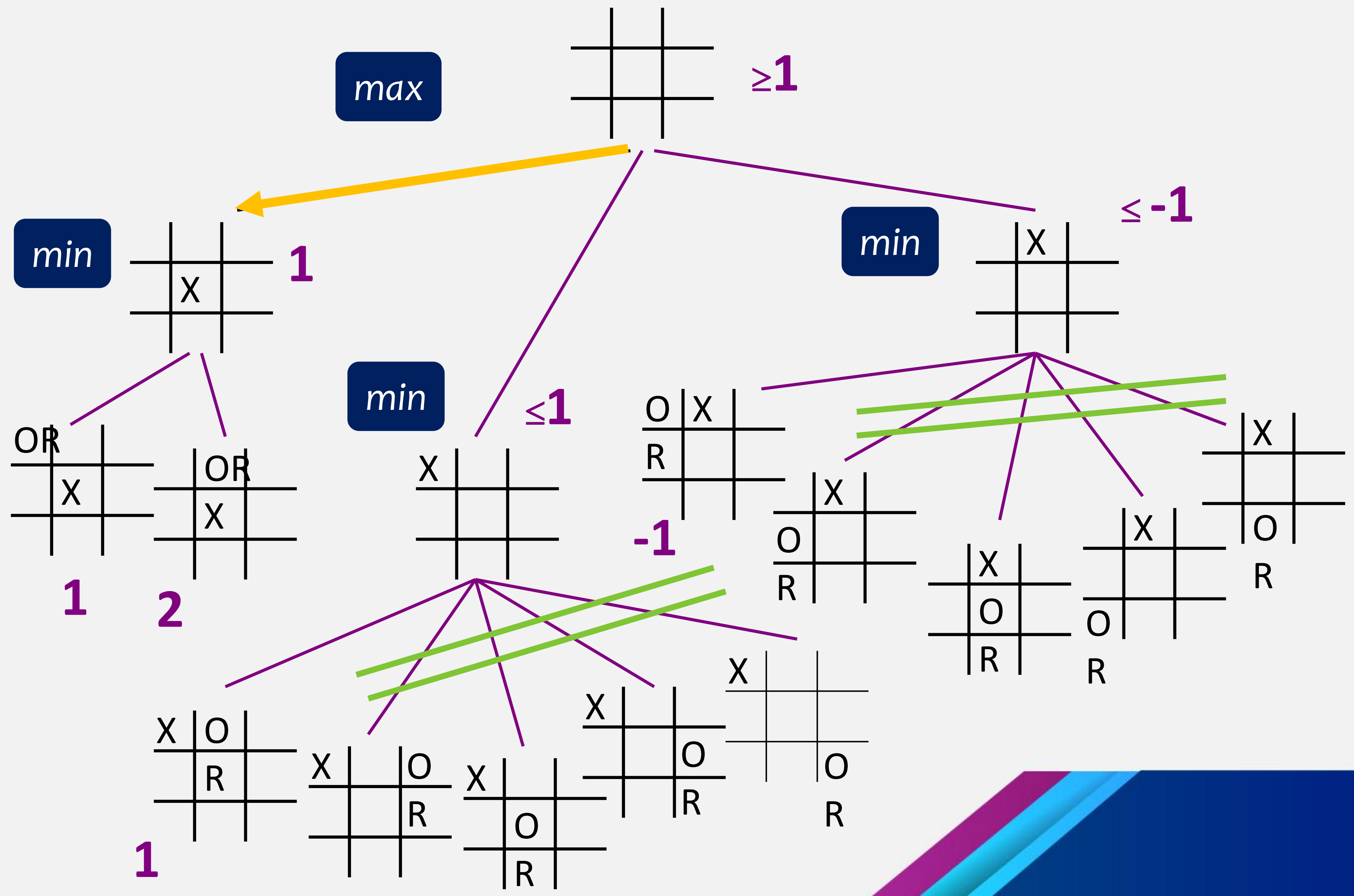
Application



Application

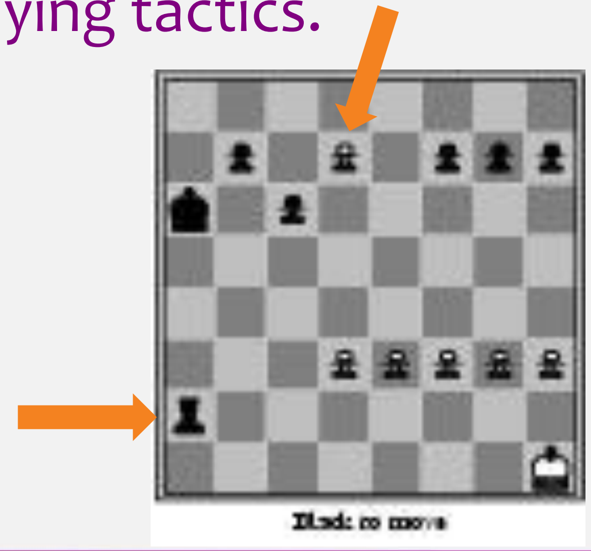


Application



Serious problems

- Quiescent positions
  - Positions unlikely to exhibit wild swings in value in the near future.
  - Restrict the quiescent search to see only certain types of moves, such as capture moves, that will quickly resolve uncertainties in the position.
- Horizon effect
  - It arises when the program is facing an opponent’s move that causes serious damage and is ultimately unavoidable, but can be temporarily avoided by delaying tactics.



A depth-limited search think that can avoid the coronation move.

Stochastic Games

- In real life there are unpredictable external events that put us in unexpected situations.

- Some games copied this situation including a random element such as throwing a dice.

- **Chance nodes:**
  - Additional nodes to the MAX and MIN nodes used to represent the possible outcomes of the random event.

Games of chance

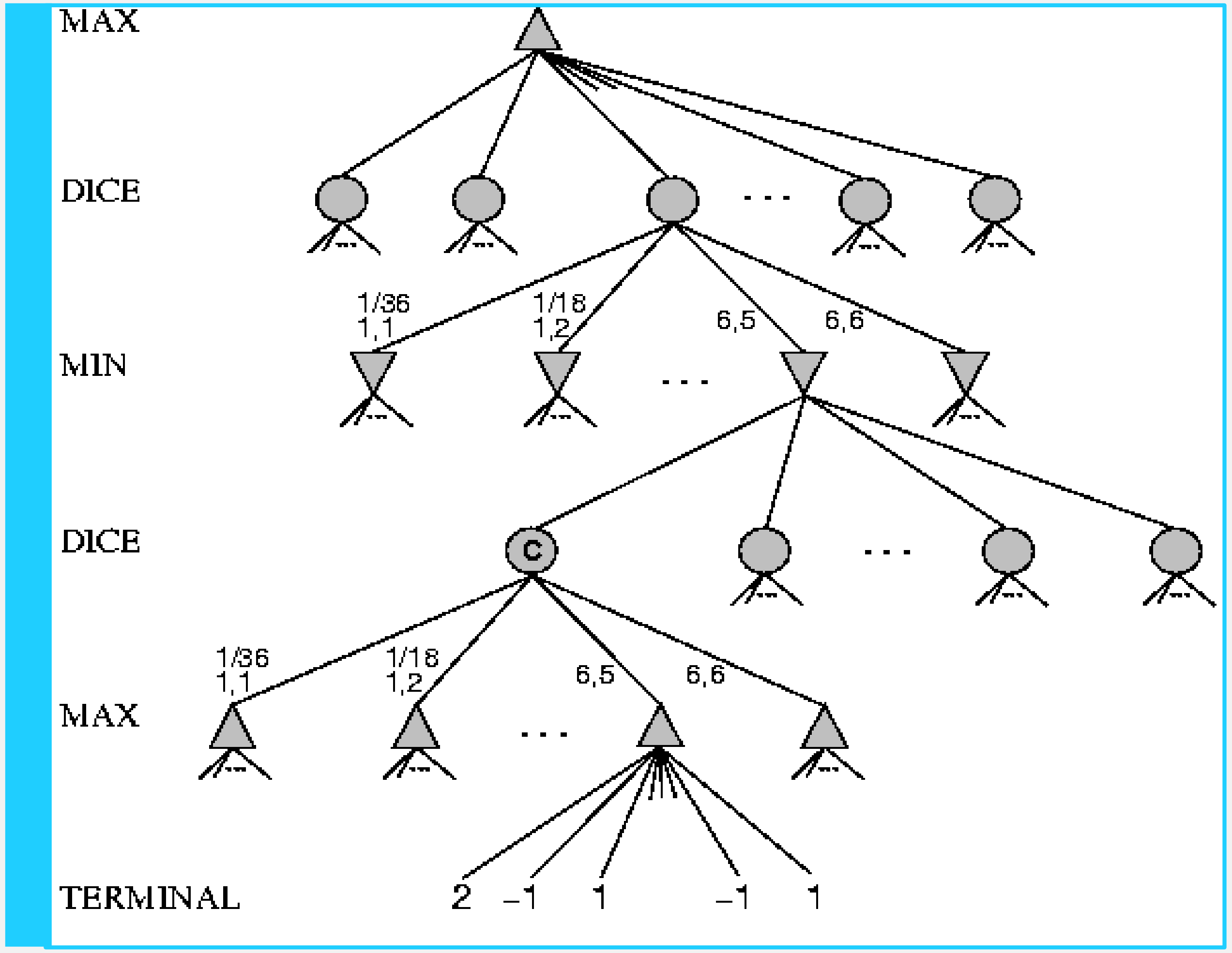
Example: backgammon

BKG was the first program to achieve serious progress.

Rules of backgammon: https://bkgm.com/rules.html

Modifying the search space

Introduction of chance nodes



Evaluation of nodes

- **Leaf nodes** ($CUTOFF\_TEST(s, dept)$)

  $Eval(s) =$ positive linear transformation of the probability of winning from s

- **Internal nodes**
  - **MAX player** ($PLAYER(s) = MAX$)

    $$max_{a \in Actions(s)} EXPECTIMINIMAX(RESULT(s, a))$$

  - **MIN player** ($PLAYER(s) = MIN$)

    $$min_{a \in Actions(s)} EXPECTIMINIMAX(RESULT(s, a))$$

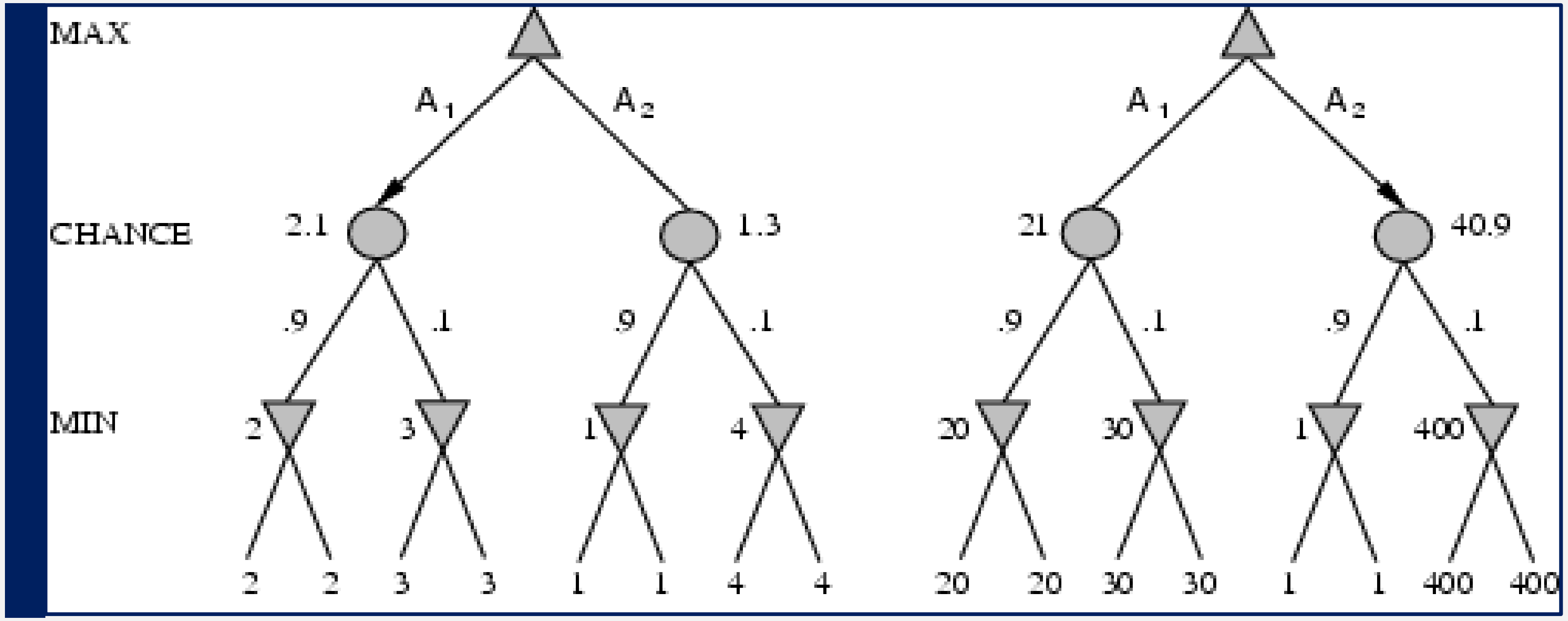
  - **CHANCE node** ($PLAYER(s) = CHANCE$)

    $$\sum_r P(r)\, EXPECTIMINIMAX(RESULT(s, r))$$

    where r represents a possible dice roll (or other chance event)

Effect of FEV in games of chance

- The program behaves totally differently if we make a change in the scale of some evaluation values!



Conclusions

Conclusions

- The techniques of games in AI have been relevant to their development.
- There are games whose strategies depend on the features of those games
- In many games it is necessary to combine solution strategies with heuristics to be able to handle the complexity of the search spaces.

Tecnológico de Monterrey