

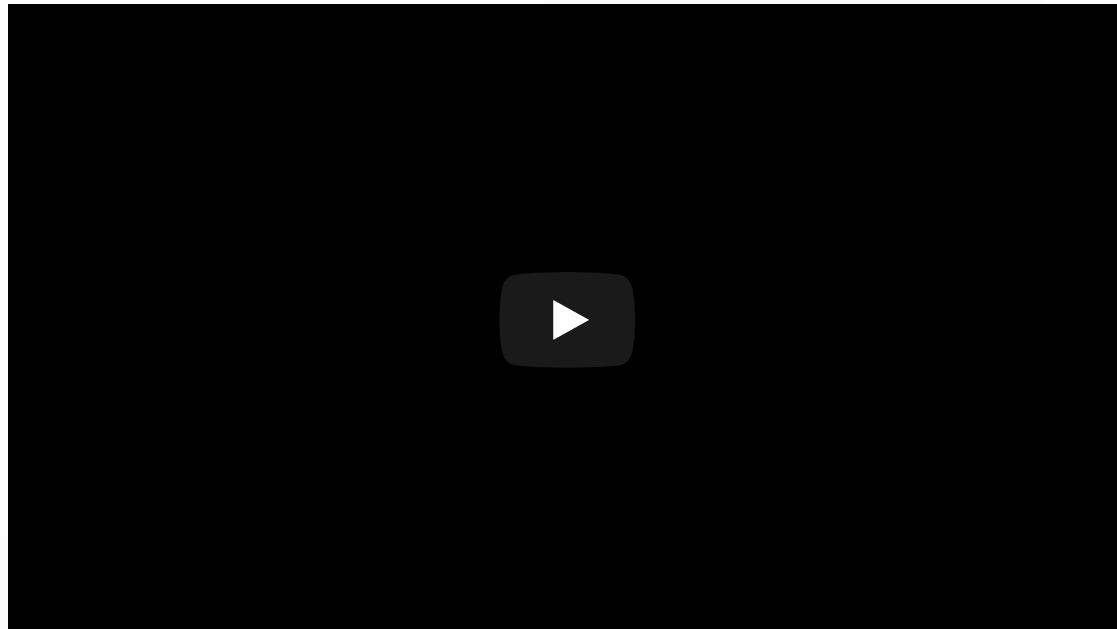
Introduction to Artificial Intelligence

Lecture 3: Games

Today

- How to act rationally in a **multi-agent environment**?
 - How to anticipate and respond to the arbitrary behavior of other agents?
- **Games** as a case study.
- **Adversarial search** (Minimax, $\alpha - \beta$ pruning, H-Minimax, Expectiminimax).
- State-of-the-art agents.

Ignore the Blonde



Games

- A **game** is a multi-agent environment where agents may have either **conflicting** or **common** interests.
- Opponents may act **arbitrarily**, even if we assume a deterministic fully observable environment.
 - Therefore the solution to a game is a **strategy** specifying a move for every possible opponent reply.
 - This is different from search where a solution is a **fixed** sequence.
- Time **limits**.
 - Branching factor is often very large.
 - Unlikely to find goal with standard search algorithms, we need to **approximate**.

Types of games

- Deterministic or stochastic?
- Perfect or imperfect information?
- One, two or more players?

Formal definition

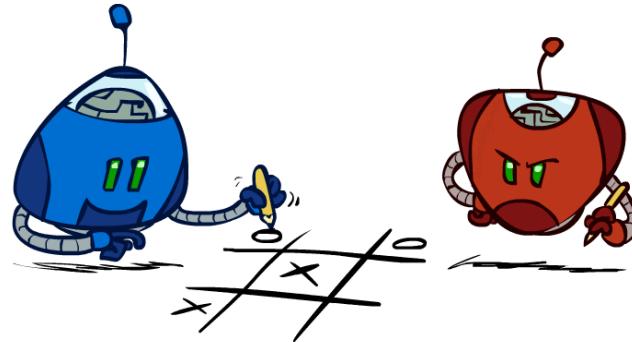
A **game** is formally defined as kind of search problem with the following components:

- The **initial state** s_0 of the game.
- A function **player**(s) that defines which **player** $p \in \{1, \dots, N\}$ has the move in state s .
- A description of the legal **actions** (or **moves**) available to a state s , denoted **actions**(s).
- A **transition model** that returns the state $s' = \text{result}(s, a)$ that results from doing action a in state s .
- A **terminal test** which determines whether the game is over.

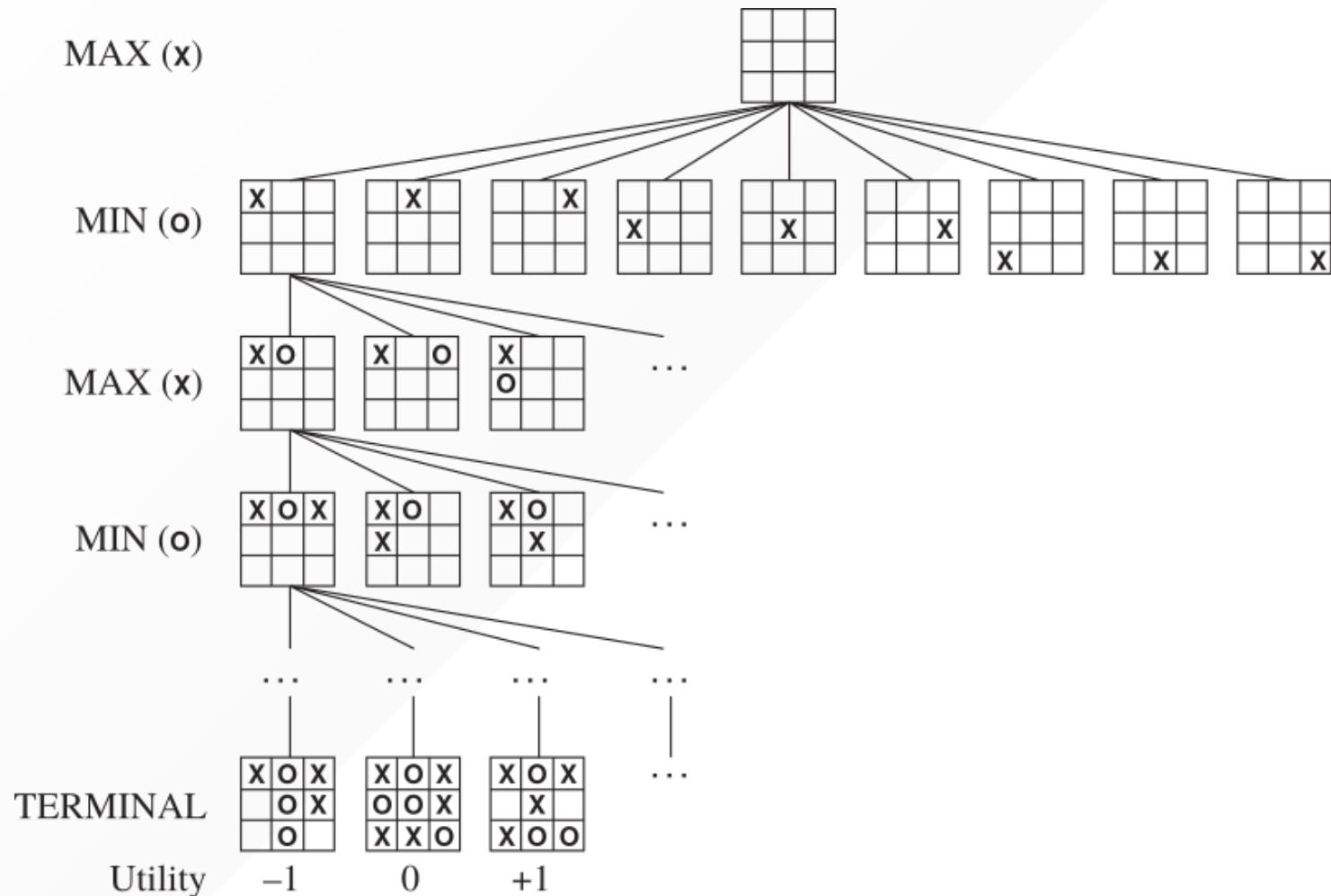
- A utility function $\text{utility}(s, p)$ (or payoff) that defines the final numeric value for a game that ends in s for a player p .
 - E.g., 1, 0 or $\frac{1}{2}$ if the outcome is win, loss or draw.
- Together, the initial state, the $\text{actions}(s)$ function and the $\text{result}(s, a)$ function define the **game tree**.
 - Nodes are game states.
 - Edges are actions.

Assumptions

- We assume a deterministic, turn-taking, two-player zero-sum game with perfect information.
 - e.g., Tic-Tac-Toe, Chess, Checkers, Go, etc.
- We will call our two players MAX and MIN. MAX moves first.



Game tree



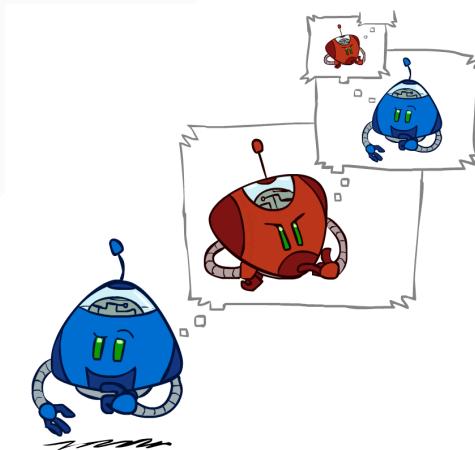
Zero-sum games

- In a **zero-sum** game, the total payoff to all players is **constant** for all games.
 - e.g., in chess: $0 + 1, 1 + 0$ or $\frac{1}{2} + \frac{1}{2}$.
- For two-player games, agents share the **same utility** function, but one wants to **maximize** it while the other wants to **minimize** it.
 - MAX maximizes the game's **utility** function.
 - MIN minimizes the game's **utility** function.
- **Strict competition.**
 - If one wins, the other loses, and vice-versa.



Adversarial search

- In a search problem, the optimal solution is a sequence of actions leading to a goal state.
 - i.e., a terminal state where MAX wins.
- In a game, the opponent (MIN) may react **arbitrarily** to a move.
- Therefore, a player (MAX) must define a contingent **strategy** which specifies
 - its moves in the initial state,
 - its moves in the states resulting from every possible response by MIN,
 - its moves in the states resulting from every possible response by MIN in those states,
 - ...



[Q] What is an optimal strategy (or perfect play)? How do we find it?

Minimax

The **minimax value** $\text{minimax}(s)$ is the largest achievable payoff (for MAX) from state s , assuming an optimal adversary (MIN).

$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

The **optimal** next move (for MAX) is to take the action that maximizes the minimax value in the resulting state.

- Assuming that MIN is an optimal adversary maximizes the **worst-case outcome** for MAX.
- This is equivalent to not making an assumption about the strength of the opponent.

Minimax example

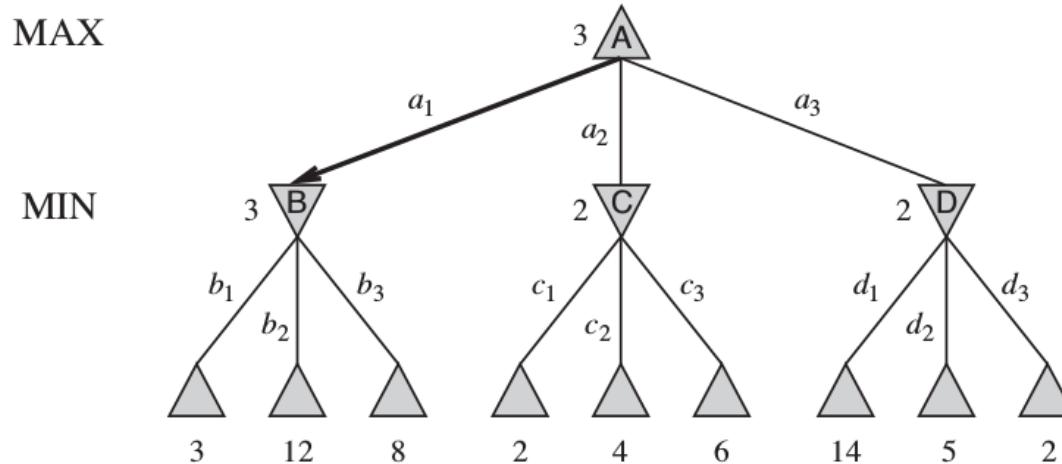


Figure 5.2 A two-ply game tree. The \triangle nodes are “MAX nodes,” in which it is MAX’s turn to move, and the \diamond nodes are “MIN nodes.” The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX’s best move at the root is a_1 , because it leads to the state with the highest minimax value, and MIN’s best reply is b_1 , because it leads to the state with the lowest minimax value.

Properties of Minimax

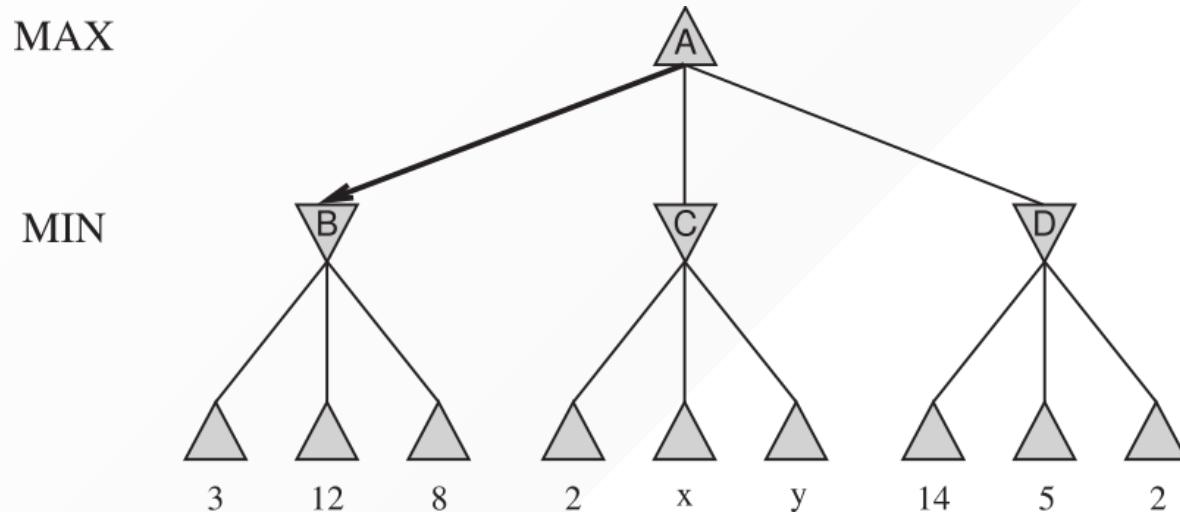
- Completeness:
 - Yes, if tree is finite.
- Optimality:
 - Yes, if MIN is an optimal opponent.
 - What if MIN is suboptimal?
 - Show that MAX will do even better.
 - What if MIN is suboptimal and predictable?
 - Other strategies might do better than Minimax. However they will do worse on an optimal opponent.

Minimax efficiency

- Assume $\text{minimax}(s)$ is implemented using its recursive definition.
- How **efficient** is minimax?
 - Time complexity: same as DFS, i.e., $O(b^m)$.
 - Space complexity:
 - $O(bm)$, if all actions are generated at once, or
 - $O(m)$, if actions are generated one at a time.

[Q] Do we need to explore the whole game tree?

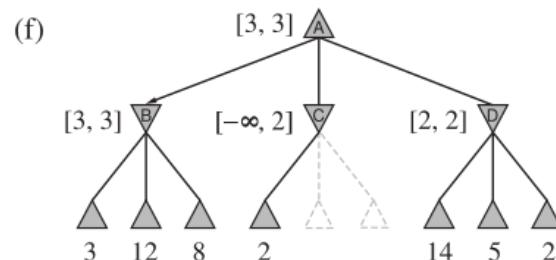
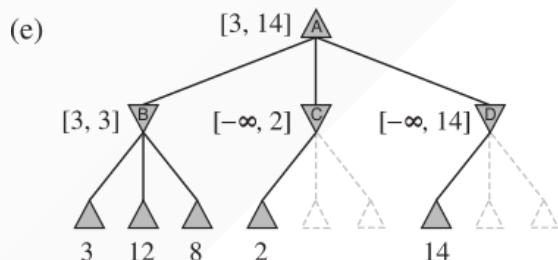
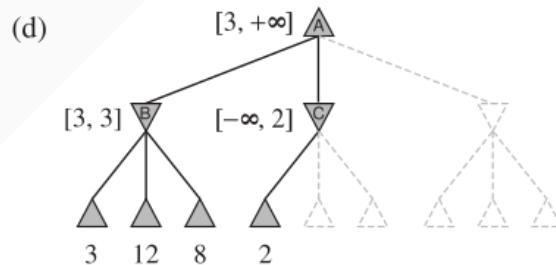
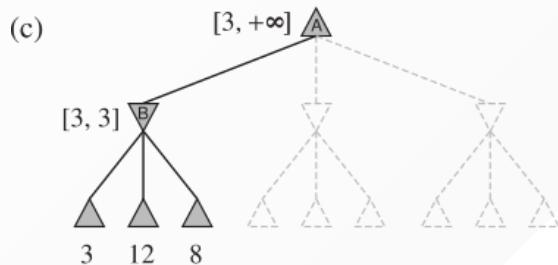
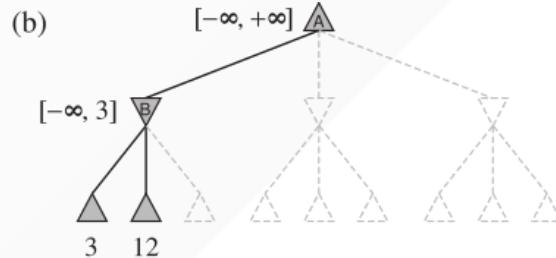
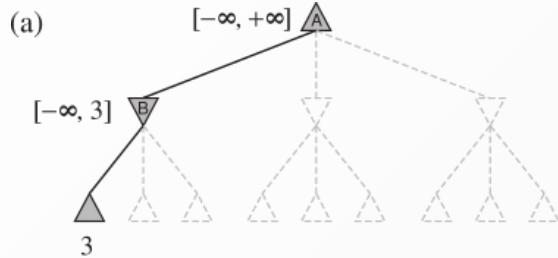
Pruning



$$\begin{aligned}\text{MINIMAX}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

Therefore, it is possible to compute the **correct** minimax decision **without** looking at every node in the tree.

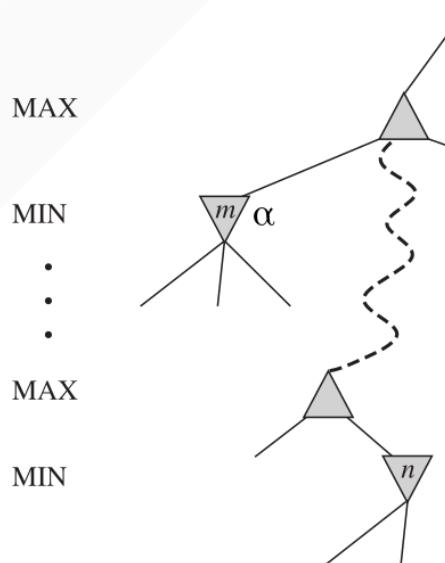
Pruning



α - β pruning

We want to compute $v = \text{minimax}(n)$.

- We loop over n 's children.
- The minimax values are being computed one at a time and v is updated iteratively.
- Let α be the best value that MAX can get, at any choice point along the current path from root.
- If v becomes lower than α , then **MAX will avoid it**.
- Therefore, we can **stop iterating** over the remaining n 's other children.



α - β search

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $-\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow$   $+\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

Properties of α - β search

- Pruning has **no effect** on the minimax values. Therefore, **completeness** and **optimality** are preserved from Minimax.
- **Time complexity:**
 - The effectiveness depends on the order in which the states are examined.
 - If states could be examined in **perfect order**, then $\alpha - \beta$ search examines only $O(b^{m/2})$ nodes to pick the best move, vs. $O(b^m)$ for minimax.
 - $\alpha - \beta$ can solve a tree twice as deep as minimax can in the same amount of time.
 - Equivalent favor to have an effective branching factor \sqrt{b} .
- **Space complexity:** $O(m)$, as for Minimax.

Game tree size



Chess:

- $b \approx 35$ (approximate average branching factor)
- $d \approx 100$ (depth of a game tree for typical games)
- $b^d \approx 35^{100} \approx 10^{154}$.
- For $\alpha - \beta$ search and perfect ordering, we get $b^{d/2} \approx 35^{50} = 10^{77}$.

Finding the exact solution is completely **infeasible**.

Imperfect real-time decisions

- Under **time constraints**, searching for the exact solution is not feasible in most realistic games.
- Solution: cut the search earlier.
 - Replace the $\text{utility}(s)$ function with a heuristic **evaluation function** $\text{eval}(s)$ that estimates the state utility.
 - Replace the terminal test by a **cutoff test** that decides when to stop expanding a state.

$\text{H-MINIMAX}(s, d) =$

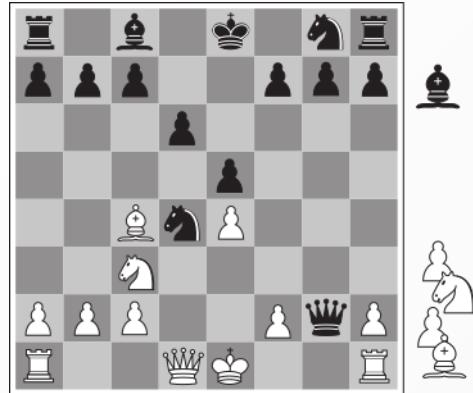
$$\begin{cases} \text{EVAL}(s) & \text{if } \text{CUTOFF-TEST}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MIN}. \end{cases}$$

[Q] Can $\alpha - \beta$ search be adapted to implement H-Minimax?

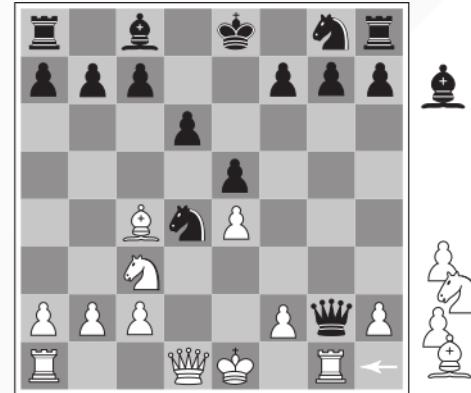
Evaluation functions

- An evaluation function returns an **estimate** of the expected utility of the game from a given position.
- The computation **must be short** (that is the whole point to search faster).
- Ideally, the evaluation should **order** terminal states in the same way as in Minimax.
 - The evaluation values may be different from the true minimax values, as long as order is preserved.
- In non-terminal states, the evaluation function should be strongly **correlated with the actual chances of winning**.
- Like for heuristics in search, evaluation functions can be **learned** using machine learning algorithms.

Quiescence



(a) White to move

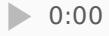


(b) White to move

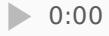
- These states only differ in the position of the rook at lower right.
- However, Black has advantage in (a), but not in (b).
- If the search stops in (b), Black will not see that White's next move is to capture its Queen, gaining advantage.
- A sophisticated cutoff test should be apply to positions that are **quiescent**.
 - i.e., states that are unlikely to exhibit wild swings in value in the near future.

The horizon effect

- Evaluations functions are **always imperfect**.
- Often, the deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters.
- If not looked deep enough, **bad moves may appear as good moves** (as estimated by the evaluation function) because their consequences are hidden beyond the search horizon.
 - and vice-versa!

 0:00

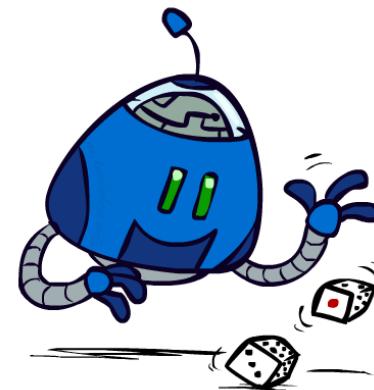
Cutoff at depth 2, evaluation = the closer to the dot, the better.

 0:00

Cutoff at depth 10, evaluation = the closer to the dot, the better.

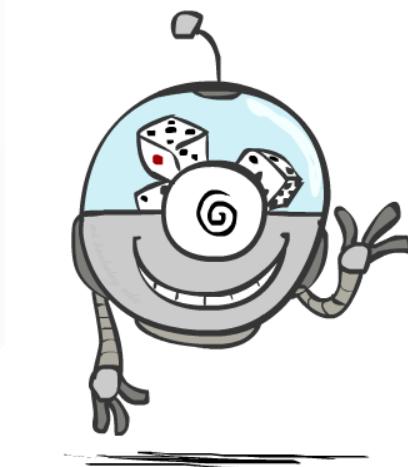
Stochastic games

- In real life, many unpredictable external events can put us into unforeseen situations.
- Games that mirror this unpredictability are called **stochastic games**. They include a random element, such as:
 - explicit randomness: rolling a dice;
 - unpredictable opponents: ghosts respond randomly;
 - actions may fail: when moving a robot, wheels might slip.

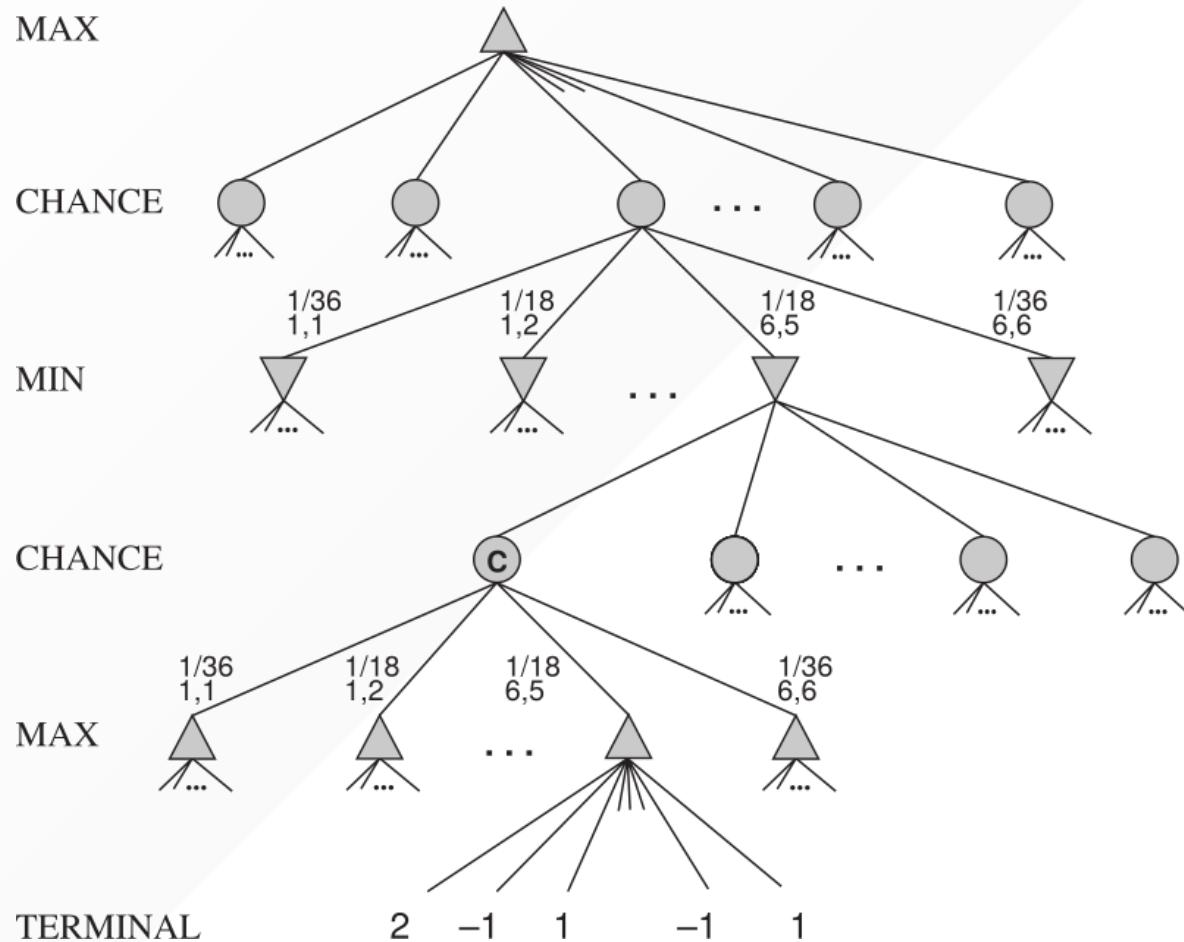


Stochastic games

- In a game tree, this random element can be **modeled** with **chance nodes** that map a state-action pair to the set of possible outcomes, along with their respective **probability**.
- This is equivalent to considering the environment as an extra **random agent** player that moves after each of the other players.



Stochastic game tree



[Q] What is the best move?

Expectiminimax

- Because of the uncertainty in the action outcomes, states no longer have a **definite minimax** value.
- We can only calculate the **expected** value of a state under optimal play by the opponent.
 - i.e., the average over all possible outcomes of the chance nodes.
 - minimax values correspond instead to the worst-case outcome.

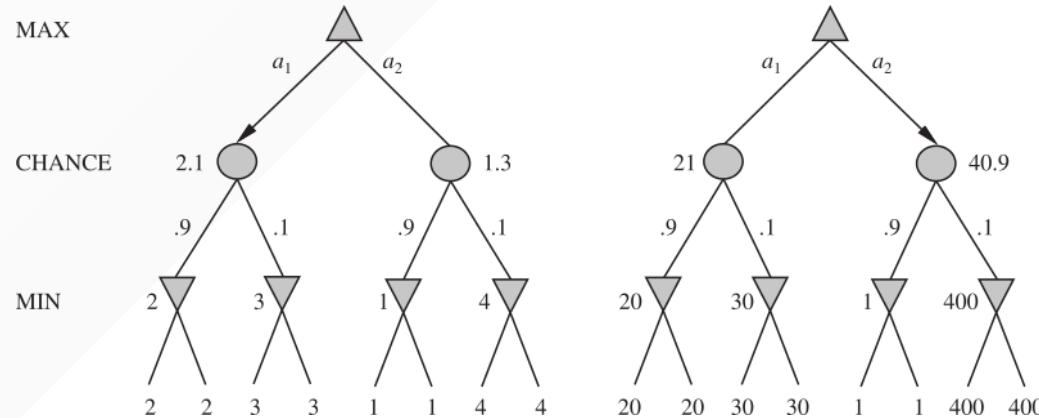
$\text{EXPECTIMINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

[Q] Does taking the rational move mean the agent will be successful?

Evaluation functions

- As for minimax(n), the value of expectiminimax(n) may be approximated by stopping the recursion early and using an evaluation function.
- However, to obtain correct move, the evaluation function should be a **positive linear transformation** of the expected utility of the state.
 - It is not enough for the evaluation function to just be order-preserving.
- If we assume bounds on the utility function, $\alpha - \beta$ search can be adapted to stochastic games.



An order-preserving transformation on leaf values changes the best move.

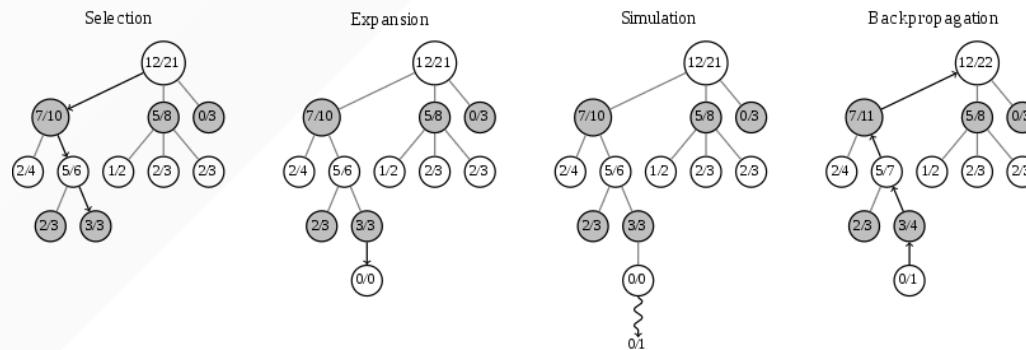
Monte Carlo simulation

- To evaluate a state, have the algorithm play **against itself** using **random moves**, thousands of times.
 - The sequence of random moves is called a **random playout**.
- Use the proportion of wins as the state evaluation.

Monte Carlo tree search

1. **Selection:** start from root, select successive child nodes down to a leaf ℓ .
2. **Expansion:** unless ℓ is a terminal state, create one or more child nodes and choose a node c among them.
3. **Simulation:** play a random playout from c .
4. **Backpropagation:** use the result of the playout to update information in the nodes on the path from c to the root.

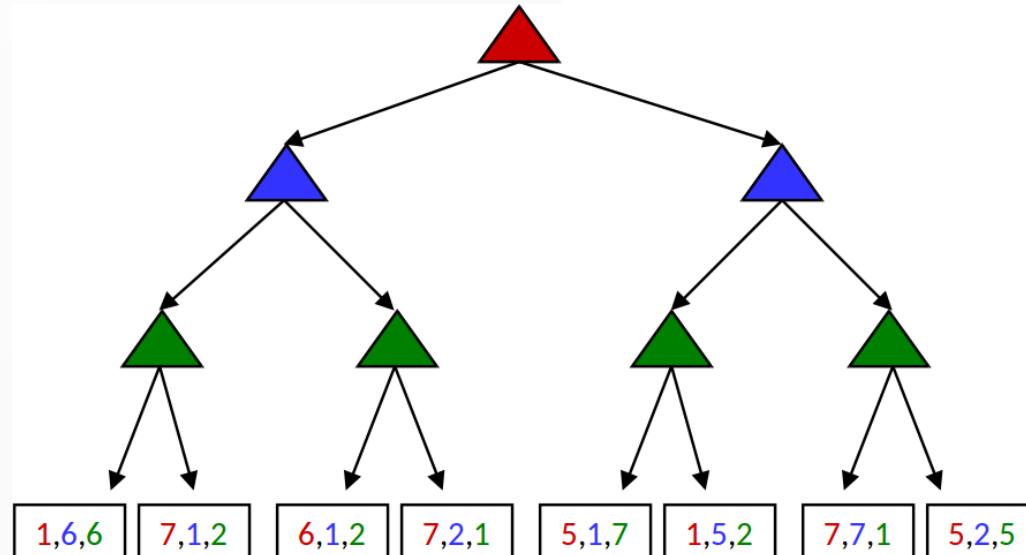
Repeat 1-4 for as long the time budget allows. Pick the best next direct move.



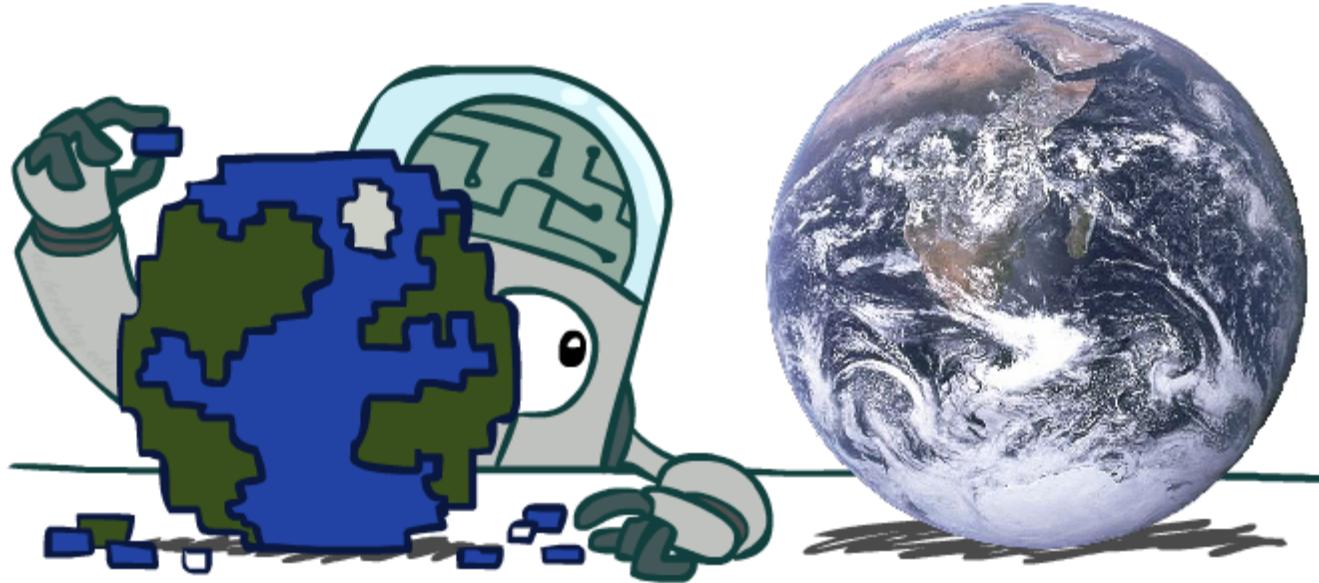
[Q] How to determine the expansion order?

Multi-agent utilities

- What if the game is not zero-sum, or has **multiple players**?
- Generalization of Minimax:
 - Terminal states are labeled with utility **tuples** (1 value per player).
 - Intermediate states are also labeled with utility tuples.
 - Each player maximizes its own component.
 - May give rise to cooperation and competition dynamically

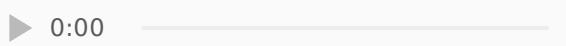


Modeling assumptions



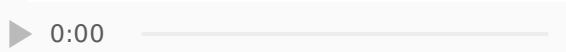
What if our assumptions are incorrect?

Assumptions vs. reality (1)



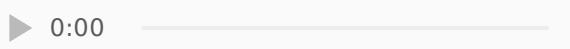
Minimax Pacman vs. Adversarial ghost

Assumptions vs. reality (2)



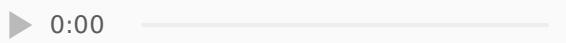
Minimax Pacman vs. Random ghost

Assumptions vs. reality (3)



Expectiminimax Pacman vs. Random ghost

Assumptions vs. reality (4)



Expectiminimax Pacman vs. Adversarial ghost

State-of-the-art game programs

Checkers

- 1951 (Christopher Strachey): first computer player .
- 1994 (Jonathan Schaeffer et al.): first computer champion. [Chinook](#) ended 40-year-reign of human champion Marion Tinsley.
 - Library of opening moves from grandmasters;
 - A deep search algorithm;
 - A good move evaluation function;
 - Based on a linear model.
 - A database for all positions with eight pieces or fewer.
- 2007 (Jonathan Schaeffer et al.): Checkers is [solved](#).
 - A weak solution is computationally proven.
 - The number of involved calculations was 10^{14} , over a period of 18 years.
 - The best an optimal player can achieve against Chinook is a draw.

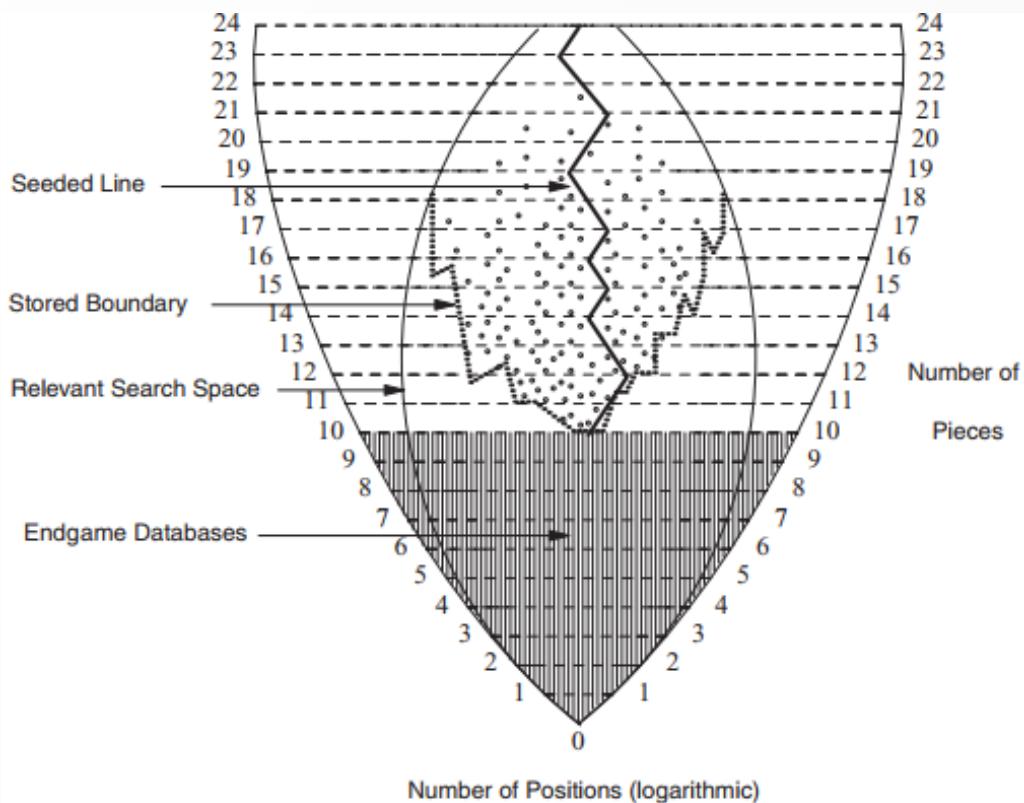


Fig. 2. Forward and backward search. The number of pieces on the board are plotted (vertically) versus the logarithm of the number of positions (Table 1). The shaded area shows the endgame database part of the proof—i.e., all positions with ≤ 10 pieces. The inner oval area shows that only a portion of the search space is relevant to the proof. Positions may be irrelevant because they are unreachable or are not required for the proof. The small open circles indicate positions with more than 10 pieces for which a value has been proven by a solver. The dotted line shows the boundary between the top of the proof tree that the manager sees (and stores on disk) and the parts that are computed by the solvers (and are not saved in order to reduce disk storage needs). The solid seeded line shows a “best” sequence of moves.

State-of-the-art game programs

Chess

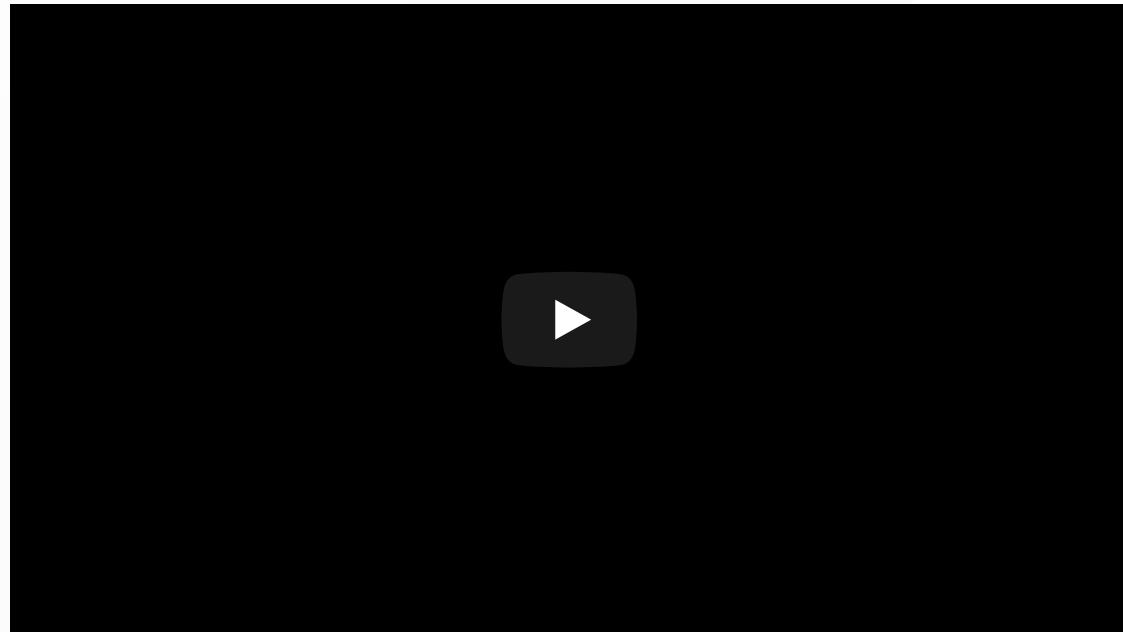
- 1997: Deep Blue defeats human champion Gary Kasparov.
 - 200000000 position evaluations per second.
 - Very sophisticated evaluation function.
 - Undisclosed methods for extending some lines of search up to 40 plies.
- Modern programs are better, if less historic.
- Chess remains unsolved due to the complexity of the game.



State-of-the-art game programs

Go

- On a 19x19, the number of legal positions is $\pm 2 \times 10^{170}$.
- This results in $\pm 10^{800}$ games, considering a length of 400 or less.
- 2010-2014: Using Monte Carlo tree search and machine learning, computer players reach low dan levels.
- 2015-2017 (Google Deepmind): AlphaGo
 - 2015: AlphaGo beat Fan Hui, the European Go Champion.
 - 2016: AlphaGo beat Lee Sedol (4-1), a 9-dan grandmaster.
 - 2017: AlphaGo beat Ke Jie, 1st world human player.
- AlphaGo combines Monte Carlo tree search and deep learning with extensive training, both from human and computer play.



Summary

- Multi-player games are variants of search problems.
- The difficulty rise in the fact that opponents may respond arbitrarily.
 - The optimal solution is a **strategy**, and not a fixed sequence of actions.
- **Minimax** is an optimal algorithm for deterministic, turn-taking, two-player zero-sum game with perfect information.
 - Approximations can be achieved with heuristics, reducing computing times.
 - Minimax can be adapted to stochastic games.
 - Minimax can be adapted to games with more than 2 players.
- Optimal behavior is **relative** and depends on the assumptions we make about the world.
- What about **partially observable** games?
 - See Chapter 17 or INFO8003 Optimal decision making for complex problems.

References

- Schaeffer, Jonathan, et al. "Checkers is solved." *science* 317.5844 (2007): 1518-1522.
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.