Introduction to Artificial Intelligence

Lecture 3: Games and Adversarial search

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Nash Equilibrium| HD | With Subtitles|

Later bekij...

Delen

Today

- How to act rationally in a multi-agent environment?
- How to anticipate and respond to the arbitrary behavior of other agents?
- Adversarial search
  - Minimax
  - $\alpha - \beta$ pruning
  - H-Minimax
  - Expectiminimax
  - Monte Carlo Tree Search
- Modeling assumptions
- State-of-the-art agents.

Minimax

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.
    - 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 is often limited.

Types of games

- Deterministic or stochastic?
- Perfect or imperfect information?
- Two or more players?

Formal definition

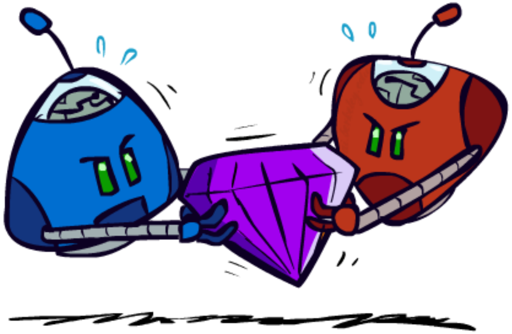
A game is formally defined as a kind of search problem with the following components:

- A representation of the states of the agents and their environment.
- The initial state s_0 of the game.
- A function $\text{player}(s)$ that defines which player $p \in \{1, ..., N\}$ has the move in state s.
- A description of the legal actions (or moves) available to a state s, denoted $\text{actions}(s)$.
- 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 $\mathbf{utility}(s, p)$ (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 $\mathbf{actions}(s)$ function and the $\mathbf{result}(s, a)$ function define the game tree.
    - Nodes are game states.
    - Edges are actions.

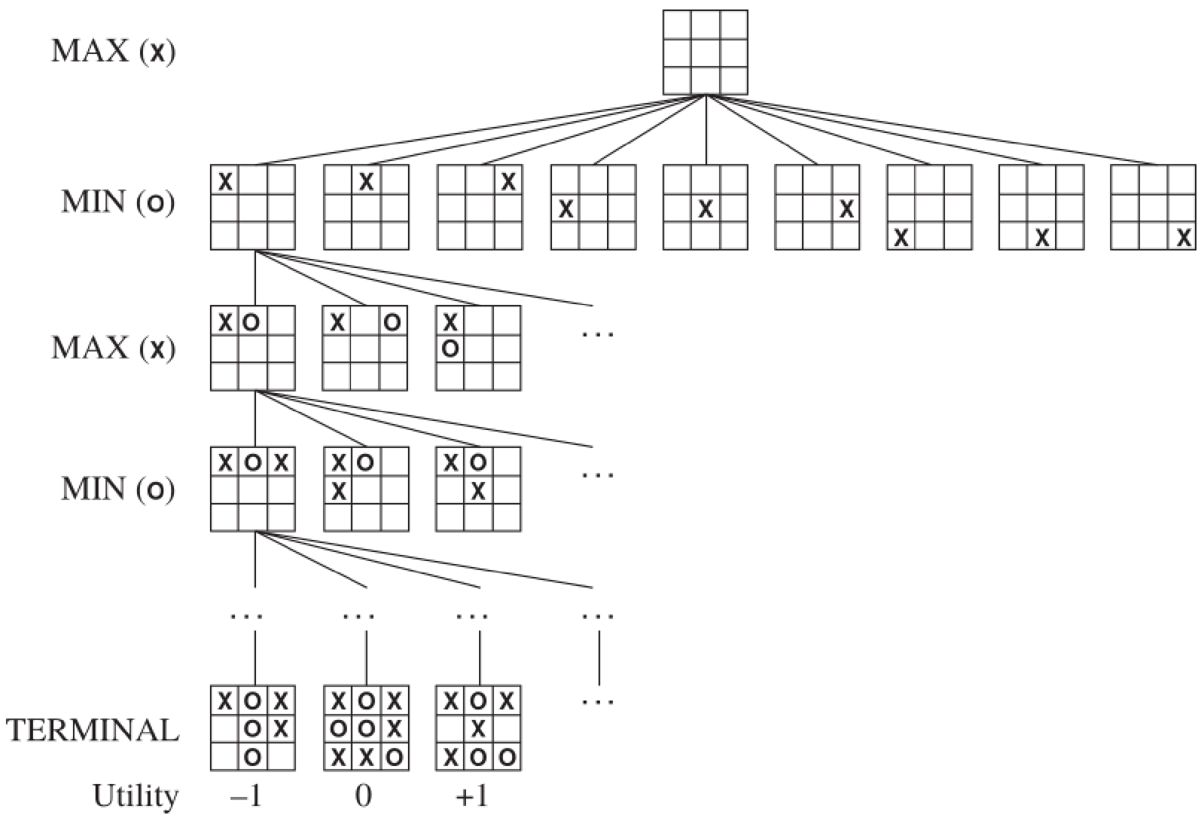
Zero-sum games

- 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 $\texttt{utility}$ function.
  - MIN minimizes the game's $\texttt{utility}$ function.
- Strict competition.
  - If one wins, the other loses, and vice-versa.



—
Image credits: CS188, UC Berkeley.

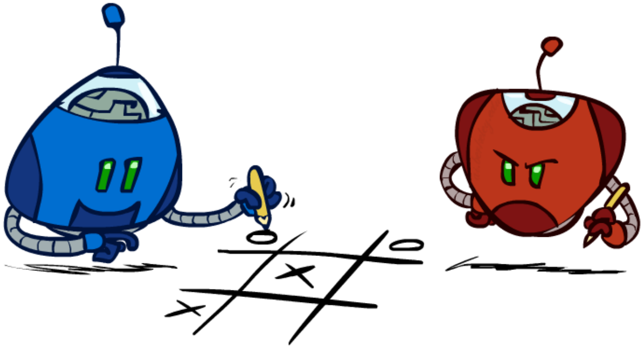
Tic-Tac-Toe game tree



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

Assumptions

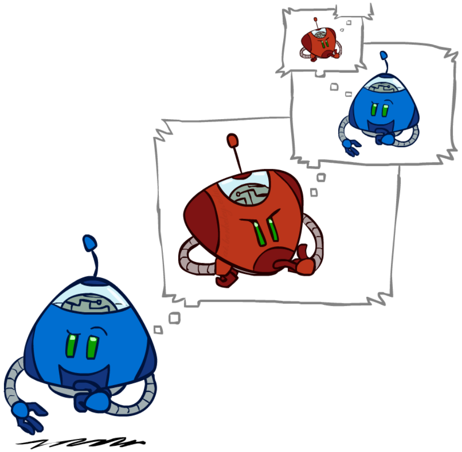
- We assume a deterministic, turn-taking, two-player zero-sum game with perfect information.
  - e.g., Tic-Tac-Toe, Chess, Checkers, Go, etc.
- We will call our two players MAX and MIN. MAX moves first.



—
Image credits: CS188, UC Berkeley.

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, ...



—
Image credits: CS188, UC Berkeley.

Minimax

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

$$
\textsc{Minimax}(s) =
\begin{cases}
\textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\
\max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\
\min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{min}
\end{cases}
$$

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

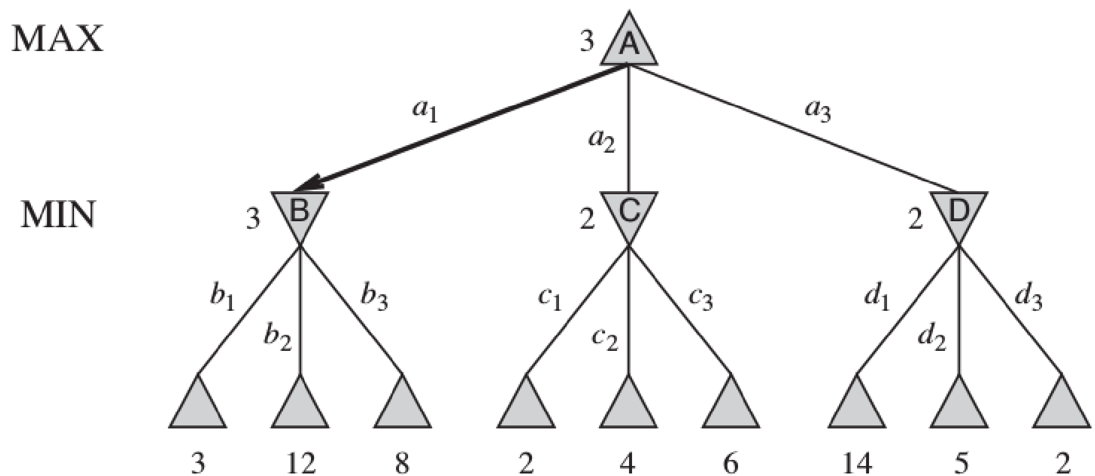


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state 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

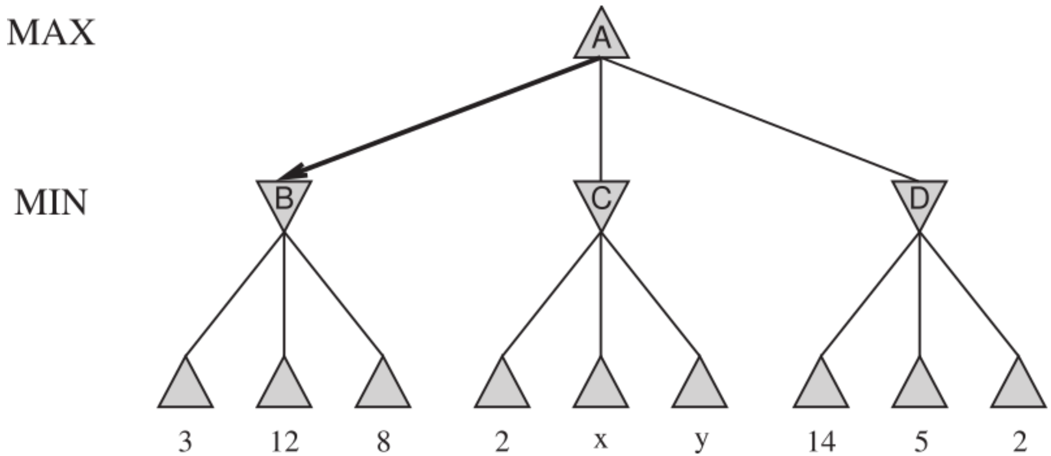
- Completeness:
  - Yes, if tree is finite.
- Optimality:
  - Yes, if MIN is an optimal opponent.
  - What if MIN is suboptimal?
    - Show that MAX will do even better.
  - What if MIN is suboptimal and predictable?
    - Other strategies might do better than Minimax. However they may do worse on an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

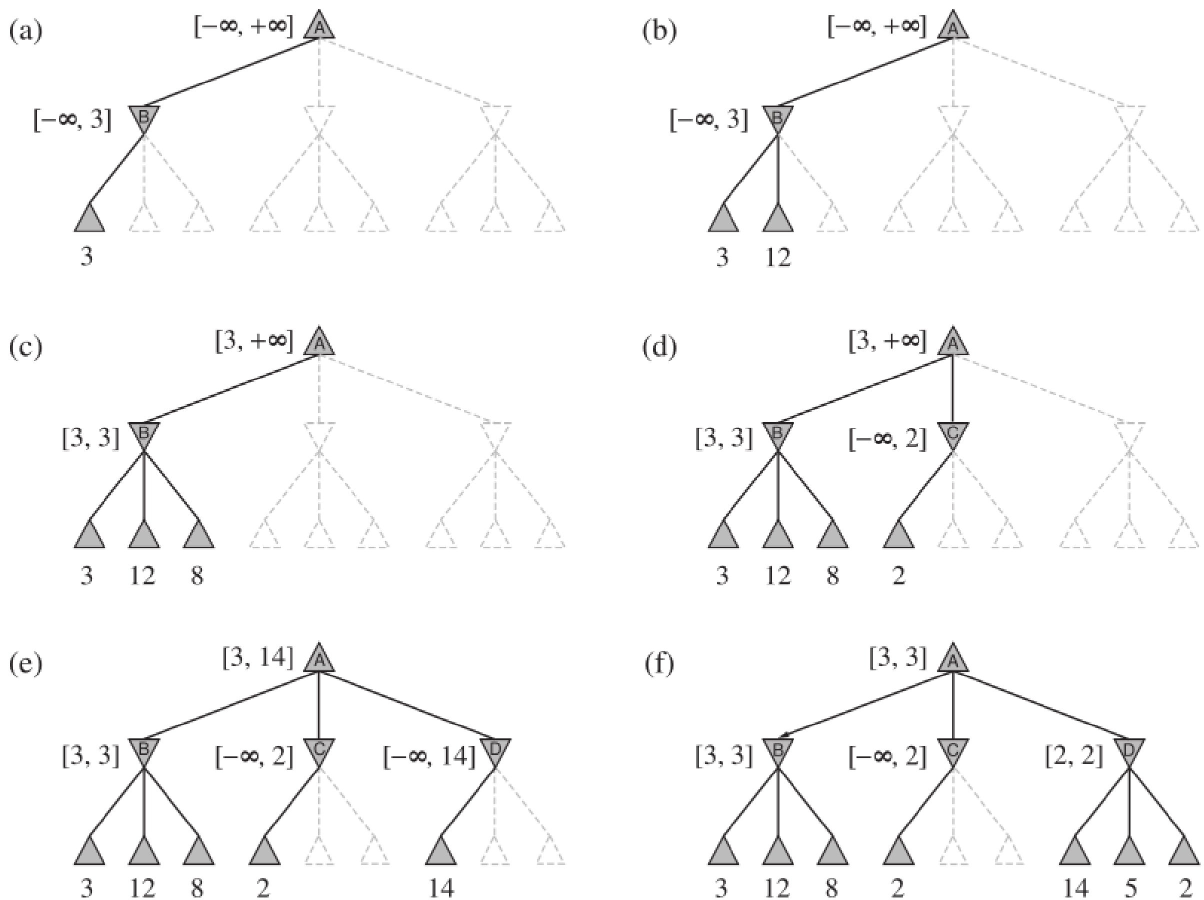
Do we need to explore the whole game tree?

Pruning



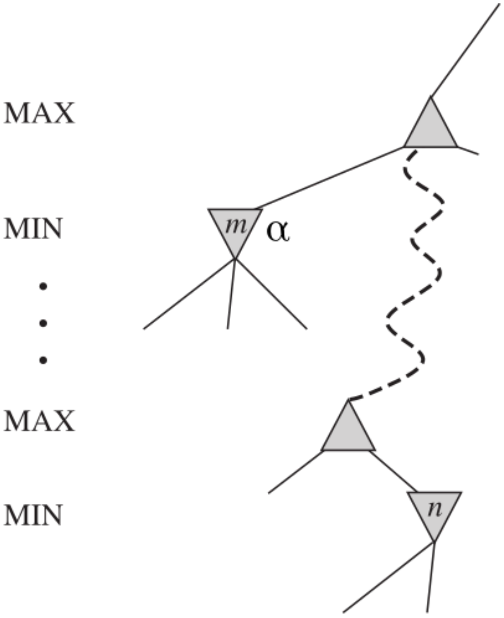
$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\
&= \max(3, \min(2, x, y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2, x, y) \leq 2 \\
&= 3.
\end{aligned}
$$

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


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3 12
(c)
[3, +∞]
A
[3, 3]
B
3 12 8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3 12 8 2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3 12 8 2 14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3 12 8 2 14 5 2

We want to compute $v = \textbf{minimax}(n)$, for $\textbf{player(n)}$=MIN.

- 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 (i.e., the highest) at any choice point along the path for MAX.
- If v becomes lower than α, then n will never be reached in actual play.
- Therefore, we can stop iterating over the remaining n's other children.



Similarly, β is defined as the best value (i.e., lowest) at any choice point along the path for MIN. We can halt the expansion of a MAX node as soon as v becomes larger than β.

α-β pruning

- Updates the values of α and β as the path is expanded.
- Prune the remaining branches (i.e., terminate the recursive calls) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

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*, α, β) **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), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **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) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, 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 to 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 with Minimax remains intractable.

Transposition table

- Repeated states occur frequently because of transpositions: distinct permutations of the move sequence end in a same position.
- Similarly to the `closed` set in `Graph-Search` (Lecture 2), it is worth storing the evaluation of a state such that further occurrences of the state do not have to be recomputed.

> What data structure should be used to efficiently store and look-up values of positions?

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 $\mathtt{utility}(s)$ function with a heuristic evaluation function $\mathtt{eval}(s)$ that estimates the state utility.
  - Replace the terminal test by a cutoff test that decides when to stop expanding a state.

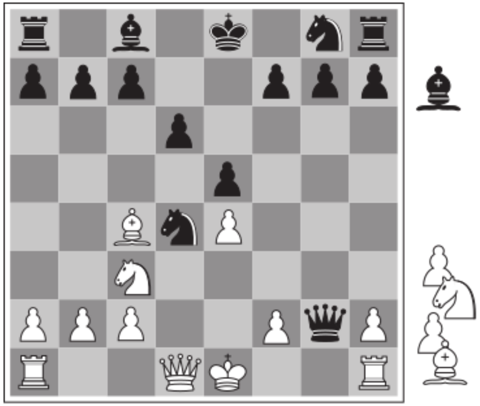
$$
\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}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

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

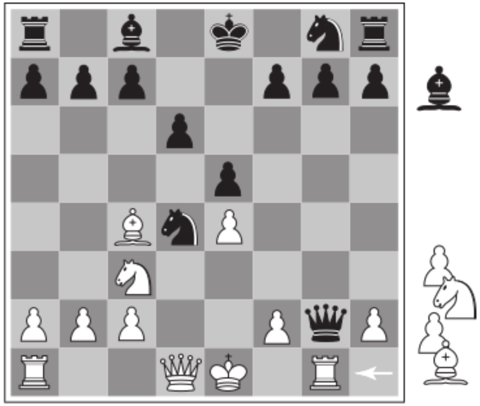
Evaluation functions

- An evaluation function $\mathbf{eval}(s)$ returns an estimate of the expected utility of the game from a given position s.
- The computation must be short (that is the whole point to search faster).
- Ideally, the evaluation should order 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.

Quiescence



(a) White to move



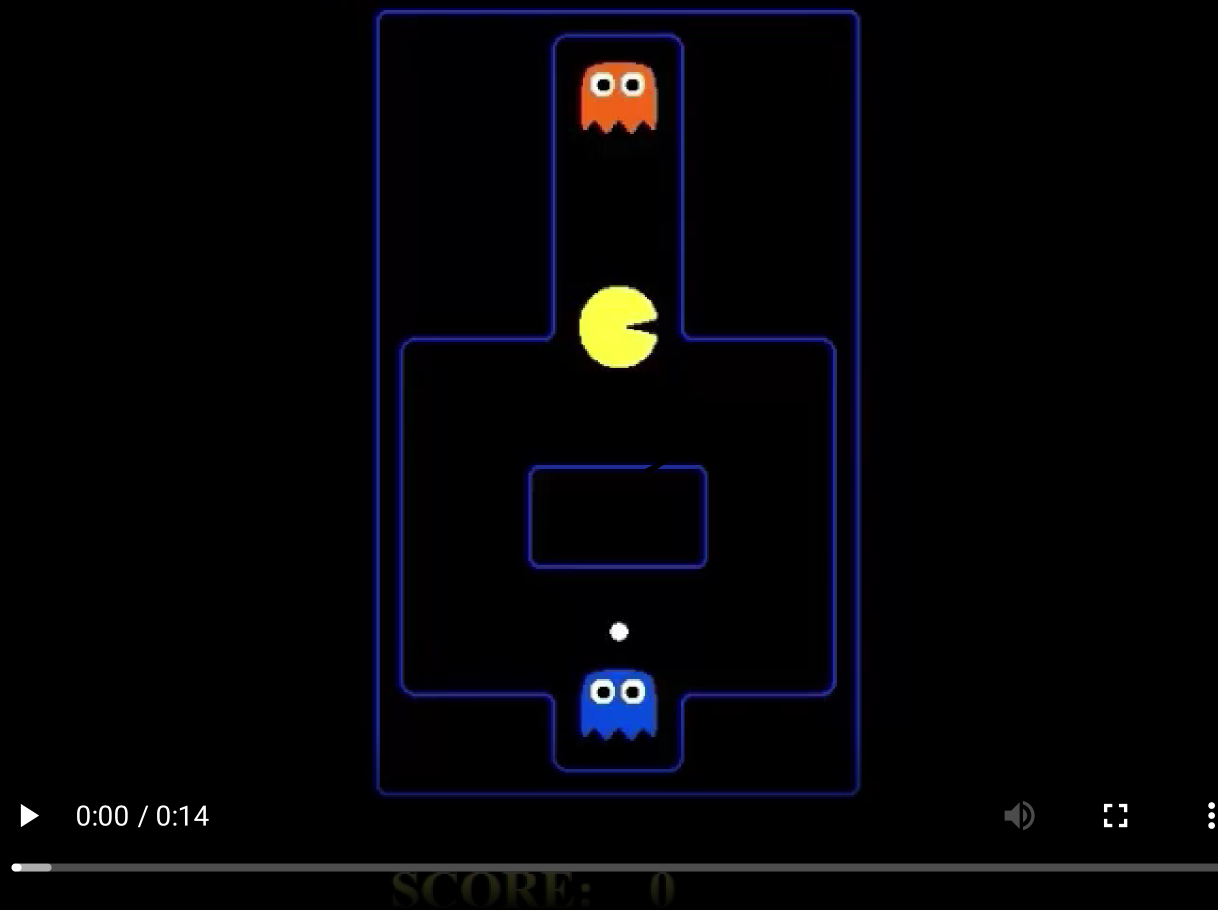
(b) White to move

- 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.
- Cutoff should only be applied to positions that are quiescent.
    - i.e., states that are unlikely to exhibit wild swings in value in the near future.

The horizon effect

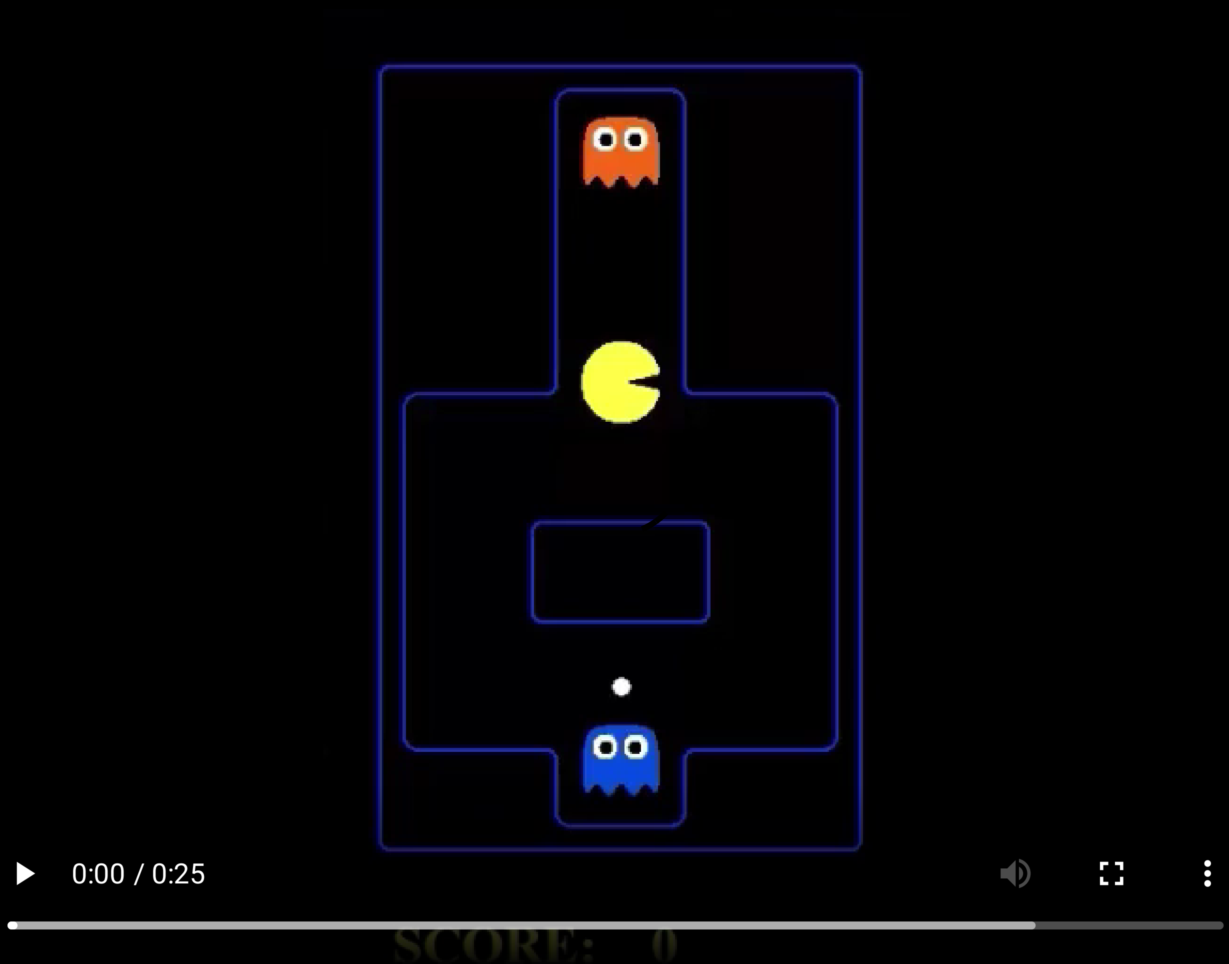
Evaluations functions are always imperfect.

- 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!
- Often, the deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters.



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

—
Image credits: CS188, UC Berkeley.



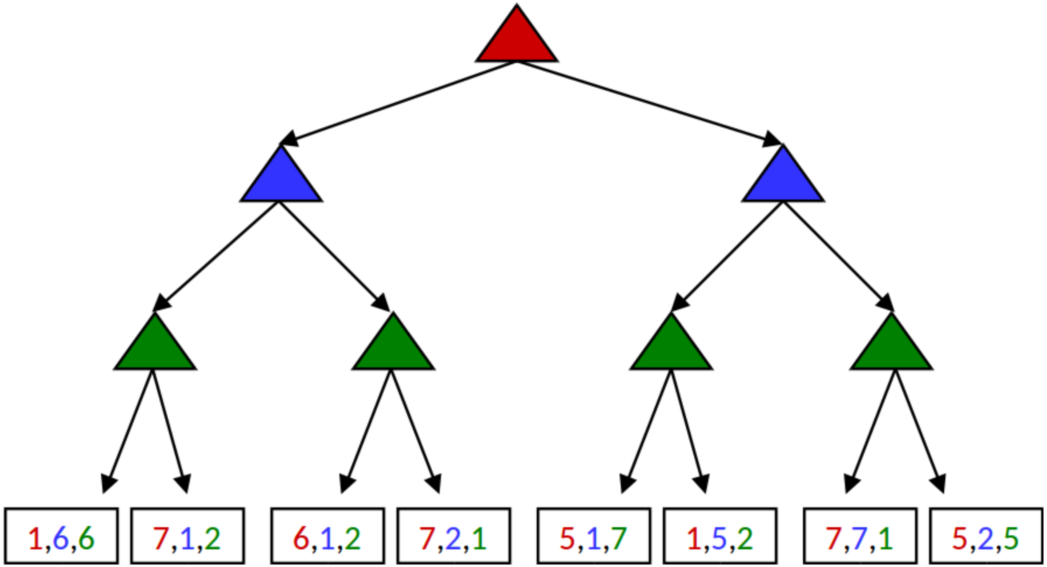
Cutoff at depth 10, evaluation = the closer to the dot, the better.

—

Image credits: CS188, UC Berkeley.

Multi-agent games

- 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.

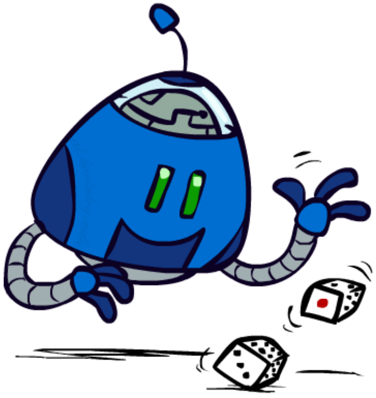


—
Image credits: CS188, UC Berkeley.

Stochastic games

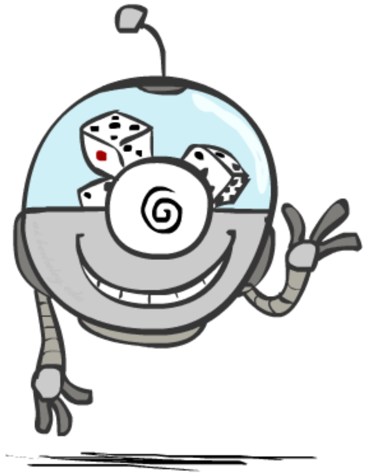
Stochastic games

- 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;
    - actions may fail: when moving a robot, wheels might slip.



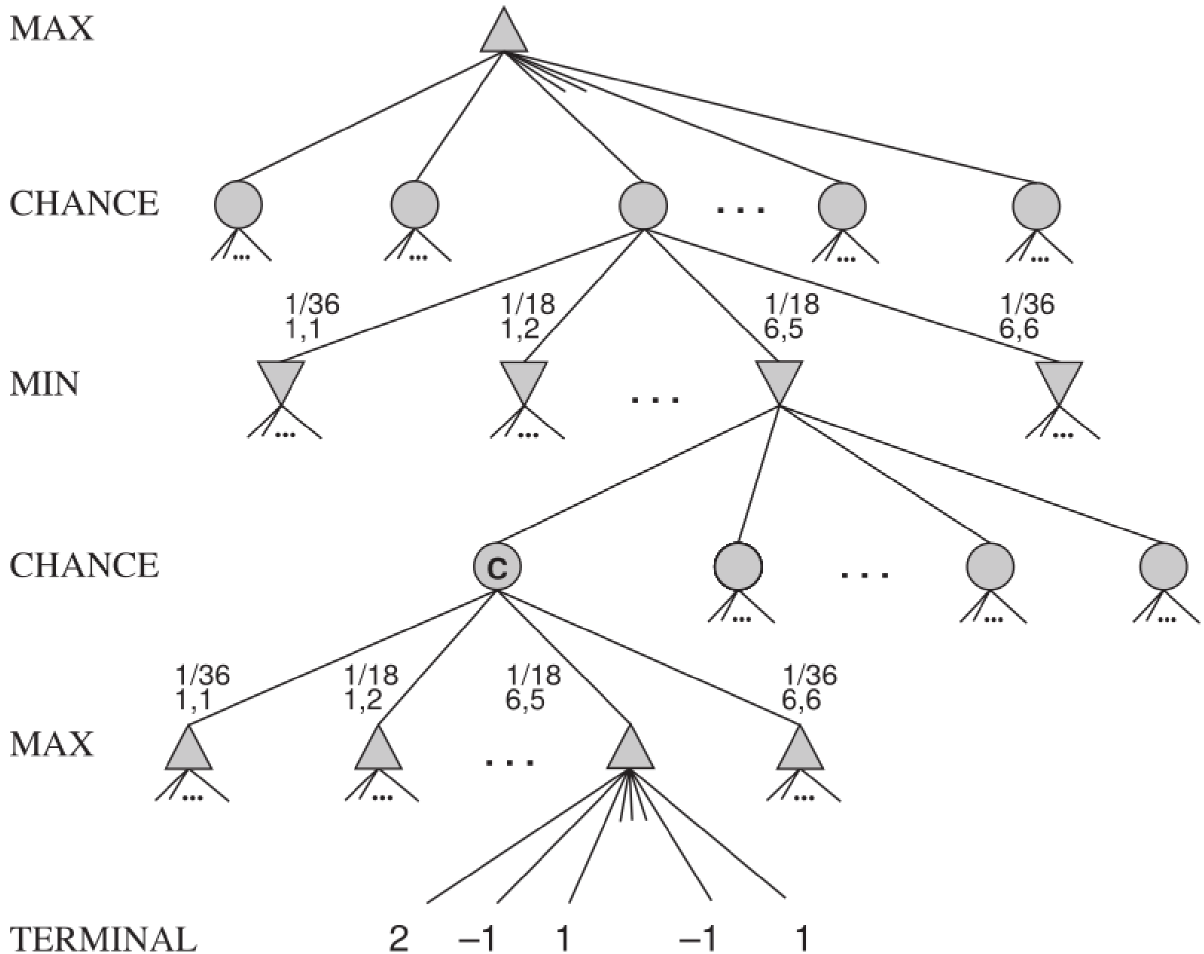
—
Image credits: CS188, UC Berkeley.

- 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.



—
Image credits: CS188, UC Berkeley.

Stochastic game tree



Expectiminimax

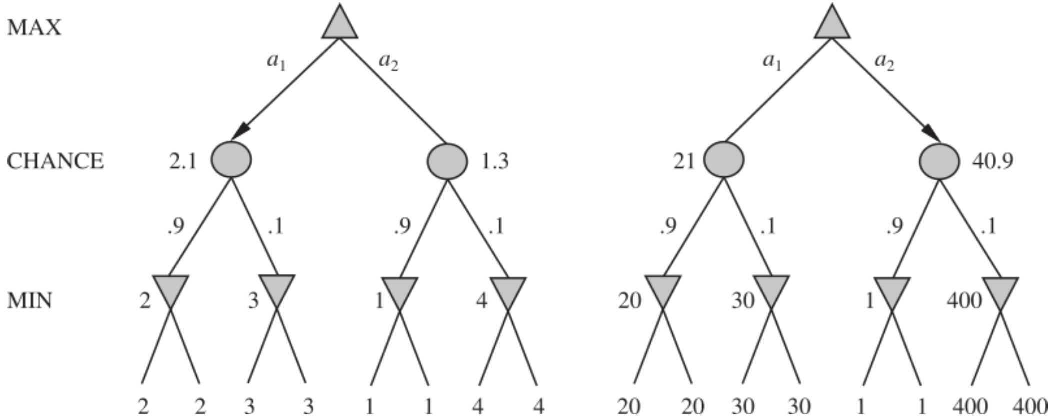
- Because of the uncertainty in the action outcomes, states no longer have a definite minimax value.
- However, we can calculate the expected value of a state under optimal play by the opponent.
  - i.e., the average over all possible outcomes of the chance nodes.
  - minimax values correspond instead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

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Does taking the rational move mean the agent will be successful?

Evaluation functions

- As for $\mathbf{minimax}(n)$, the value of $\mathbf{expectiminimax}(n)$ 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 Tree Search

Random playout evaluation

- 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.
- This strategy does not require domain knowledge!
    - The game engine is all that is needed.

Monte Carlo Tree Search

The focus of MCTS is the analysis of the most promising moves, as incrementally evaluated with random playouts.

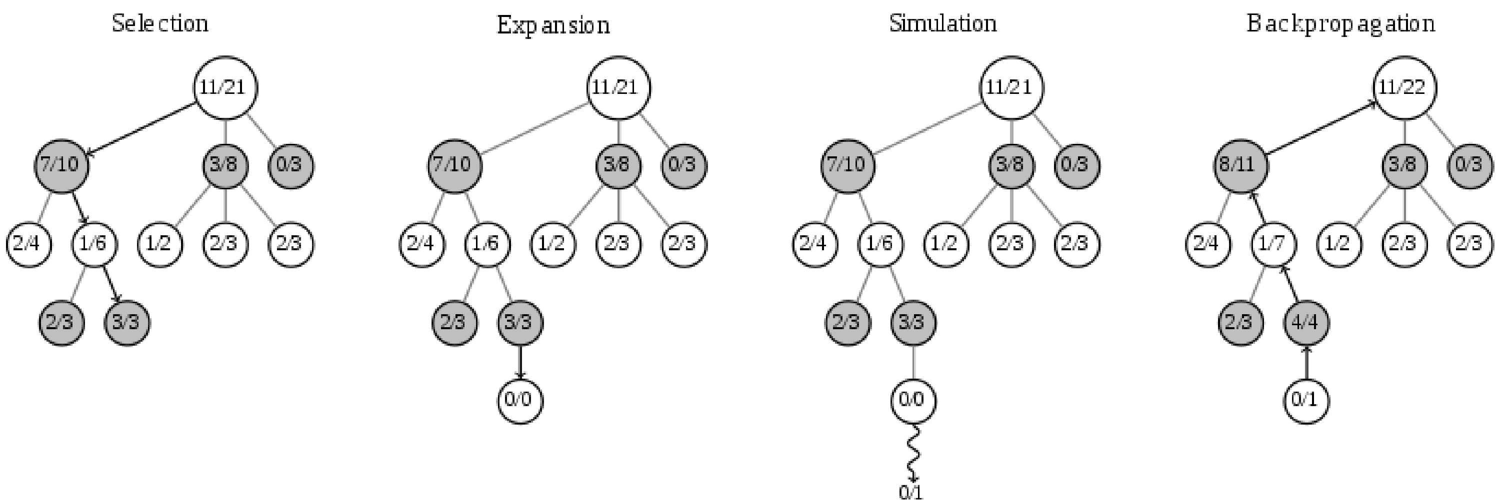
Each node n in the current search tree maintains two values:

- the number of wins $Q(n, p)$ of player p for all playouts that passed through n;
- the number $N(n)$ of times n has been visited.

The algorithm searches the game tree as follows:

1. Selection: start from root, select successive child nodes down to a node n that is not fully expanded.
2. Expansion: unless n is a terminal state, create a new child node n'.
3. Simulation: play a random playout from n'.
4. Backpropagation: use the result of the playout to update information in the nodes on the path from n' to the root.

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


Selection
11/21
7/10
3/8
0/3
2/4
1/6
1/2
2/3
2/3
2/3
3/3
Expansion
11/21
7/10
3/8
0/3
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
Simulation
11/21
7/10
3/8
0/3
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
Backpropagation
11/22
8/11
3/8
0/3
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1

Exploration and exploitation

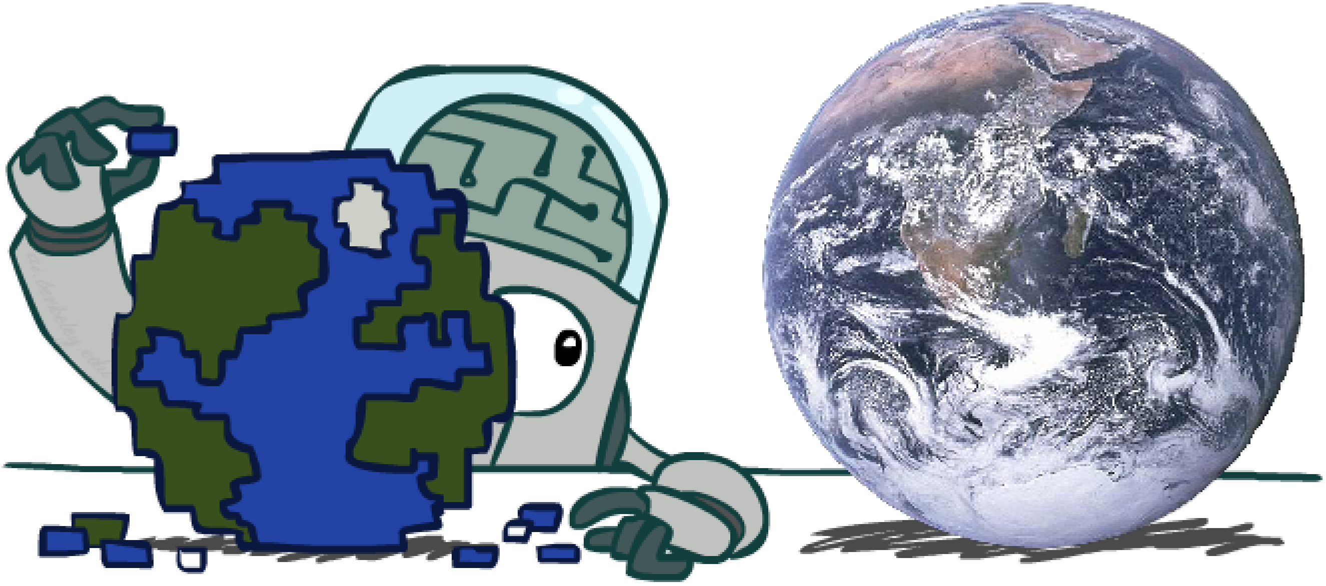
Given a limited budget of random playouts, the efficiency of MCTS critically depends on the choice of the nodes that are selected at step 1.

During the traversal of the branch in the selection step, the UCB1 policy picks the child node n' of n that maximizes

$$\frac{Q(n',p)}{N(n')} + c\sqrt{\frac{\log N(n)}{N(n')}}.$$

- The first term encourages the exploitation of higher-reward nodes.
- The second term encourages the exploration of less-visited nodes.
- The constant $c > 0$ controls the trade-off between exploitation and exploration.

Modeling assumptions

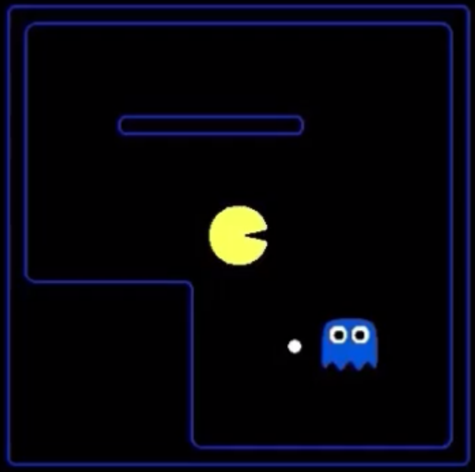


What if our assumptions are incorrect?

—
Image credits: CS188, UC Berkeley.

Setup

- P_1: Pacman uses depth 4 search with an evaluation function that avoids trouble, while assuming that the ghost follows P_2.
- P_2: Ghost uses depth 2 search with an evaluation function that seeks Pacman, while assuming that Pacman follows P_1.
- P_3: Pacman uses depth 4 search with an evaluation function that avoids trouble, while assuming that the ghost follows P_4
- P_4: Ghost makes random moves.



Minimax Pacman (P_1) vs. Adversarial ghost (P_2)

—
Image credits: CS188, UC Berkeley.

Minimax Pacman (P_1) vs. Random ghost (P_4)

—
Image credits: CS188, UC Berkeley.

Expectiminimax Pacman (P_3) vs. Random ghost (P_4)

—
Image credits: CS188, UC Berkeley.

Expectiminimax Pacman (P_3) vs. Adversarial ghost (P_2)

—
Image credits: CS188, UC Berkeley.

State-of-the-art game programs

Checkers

1951

First computer player by Christopher Strachey.

1994

The computer program Chinook ends the 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

Checkers is solved. A weak solution is computationally proven.

- The number of involved calculations was 10^{14}, over a period of 18 years.
- A draw is always guaranteed provided neither player makes a mistake.

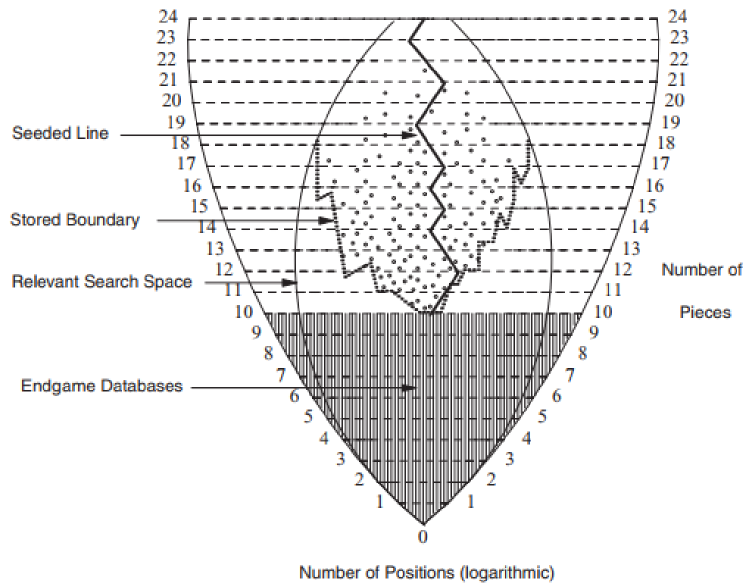


Fig. 2. Forward and backward search. The number of pieces on the board are plotted (vertically) versus the logarithm of the number of positions (Table 1). The shaded area shows the endgame database part of the proof—i.e., all positions with ≤10 pieces. The inner oval area shows that only a portion of 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.

—

Schaeffer, Jonathan, et al. "Checkers is solved." science 317.5844 (2007): 1518-1522.

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 (e.g., Stockfish or AlphaZero) are better, if less historic.
- Chess remains unsolved due to the complexity of the game.

Go

For long, Go was considered as the Holy Grail of AI due to the size of its game tree.

- 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 invents AlphaGo.

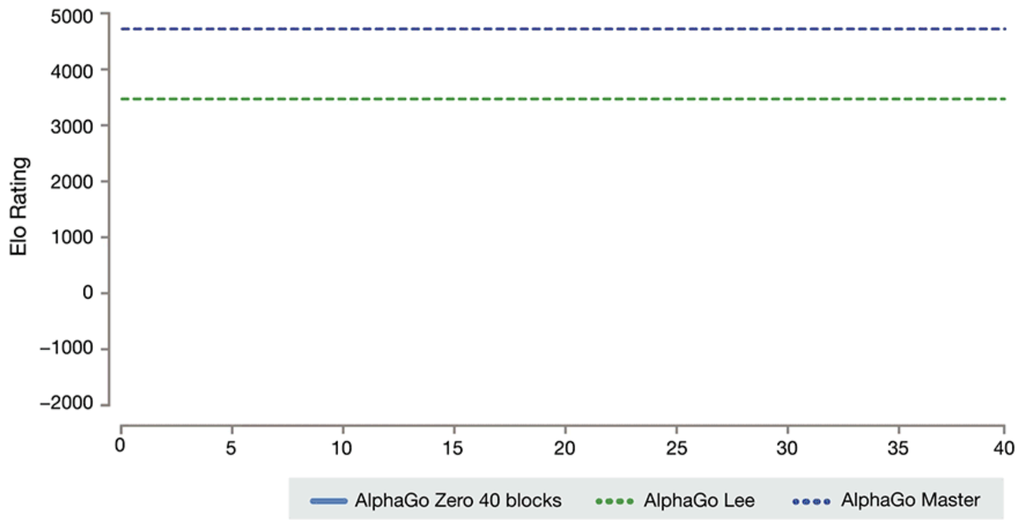
- 2015: AlphaGo beat Fan Hui, the European Go Champion.
- 2016: AlphaGo beat Lee Sedol (4-1), a 9-dan grandmaster.
- 2017: AlphaGo beat Ke Jie, 1st world human player.

AlphaGo combines Monte Carlo tree search and deep learning with extensive training, both from human and computer play.

Press coverage for the victory of AlphaGo against Lee Sedol.

2017

AlphaGo Zero combines Monte Carlo tree search and deep learning with extensive training, with self-play only



—
Credits: AlphaGo Zero: Learning from scratch

Summary

- Multi-player games are variants of search problems.
- The difficulty is to account for the fact that the opponent may act 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.
    - Due to practical time constraints, exploring the whole game tree is often infeasible.
    - 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.

The end.