CSCE 631-D: Intelligent Agents — Computational Game Solving

Lecture 2: Chess, Game Trees, and Why LLM Games Differ

Alan Kuhnle

Fall 2026

Texas A&M University

Today's Plan

Learning objectives:

1. Model chess as a game tree and define strategies formally.
2. Solve game trees by backward induction.
3. State Zermelo's theorem and the minimax theorem.
4. Explain why alpha-beta pruning preserves the minimax value.
5. Identify which assumptions from chess fail for LLM agents.

Roadmap: (1) chess and game trees; (2) backward induction and minimax; (3) alpha-beta and computational limits; (4) computer chess and the bridge from computed to estimated LLM-agent payoffs.

Chess as a Game-Theoretic Object

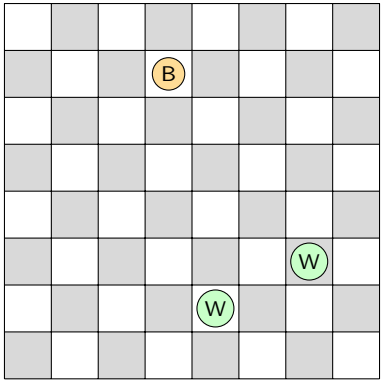
Why Chess?

Chess is the canonical example of a:

- **Two-player** game.
- **Zero-sum** game: $u_1 + u_2 = 0$ at every terminal outcome.
- **Perfect-information** game (both players see the full board; formal definition later).
- **Sequential** (extensive-form) game with alternating moves.
- **Deterministic** game (no chance moves).

Chess motivates the key concepts of this lecture: game trees, backward induction, and the minimax theorem.

A Chess Position Is Fully Visible



Both players see the same board and complete move history, even though finding the best move remains computationally difficult. We will name this property *perfect information* later in the lecture.

Chess — Basic Parameters

Parameter	Value
Players	2 (White, Black)
Outcomes	$\{+1, 0, -1\}$ (win, draw, loss for White)
Average branching factor	≈ 35
Average game length	≈ 40 moves each (≈ 80 plies)
Legal positions (estimate)	$\approx 10^{44}$
Game-tree complexity	$\approx 10^{120}$ (Shannon number)

The enormous game tree makes chess computationally intractable to solve exactly, yet its perfect-information structure makes backward induction directly applicable.

Game Trees (Preview)

Game Trees (Preview)

Definition (Informal)
A **game tree** is a rooted tree:

- internal nodes identify the player who moves;
- edges are legal actions;
- leaves carry payoff vectors (u_1, u_2).

For chess: the root is the initial position, nodes alternate between White and Black, and leaves are terminal positions.

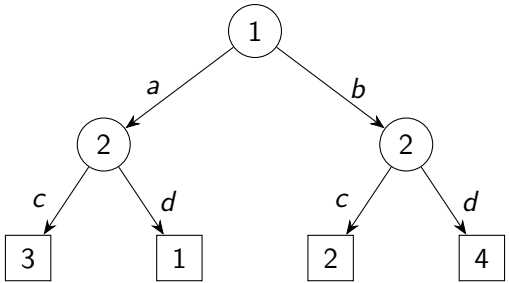
Chess as a Game Tree

Think–pair–share (60 seconds): What must a player's strategy prescribe at every one of their decision nodes?

That question leads from the tree's visible branches to a player's complete contingency plan.

Example: A Tiny Game Tree

Payoff convention. The definition above labels each leaf with a payoff vector (u_1, u_2). Because this is a zero-sum game, we show only Player 1's payoff at each leaf; Player 2's payoff is the negative.



Player 1 chooses $\{a, b\}$; then Player 2 chooses $\{c, d\}$.

Strategies in Extensive-Form Games (Preview)

Given a game tree, how do we describe a player's complete plan?

Key Idea

A **(pure) strategy** for player i specifies an action at *every* decision node of player i, even those not reached under the strategy.

This completeness pins down an outcome for every possible opponent play, which lets backward induction assign values.

Strategies in the Tiny Tree

In the previous example:

- Player 1 has 2 strategies: $\{a, b\}$.
- Player 2 has 4 strategies: $\{cc, cd, dc, dd\}$ (first letter after a; second letter after b).

Example: cd means choose c after a, then choose d after b if that node is reached.

A pure strategy in chess is a contingency plan for every possible board position, so the strategy space is astronomically large.

Formal treatment of information sets and behavioral strategies: Module 4.

Perfect Information and Backward Induction

Perfect Information

Definition
A game has **perfect information** if every player, when making a decision, knows the complete history of all previous moves.

An **information set** is a collection of positions that a player cannot distinguish when it is their turn to move. Perfect information means every information set is a singleton.

Examples: Chess, Go, and Checkers. **Counterexample:** Poker, with hidden cards.

Backward Induction

Algorithm: Backward Induction

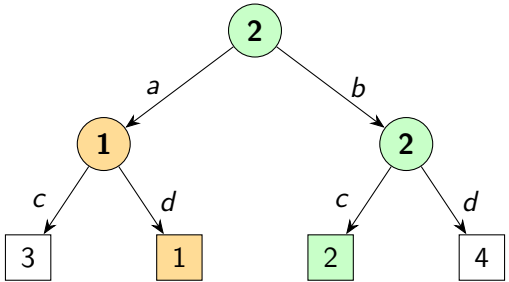
1. Start at the **leaves** of the game tree.
2. At each node where player i moves, assign the value:

$$v(n) = \begin{cases} \max_{a \in A(n)} v(\text{child}(n, a)) & \text{Player 1,} \\ \min_{a \in A(n)} v(\text{child}(n, a)) & \text{Player 2.} \end{cases}$$

3. Propagate values up to the root.

Result: the root value is the **game value**, the payoff under optimal play by both sides. Module 4 names the complete plan a **subgame-perfect equilibrium** (SPE).

Backward Induction: Example



Before we trace: What is the game value of this tree? Can Player 1 guarantee a win?

Player 2 minimizes: left subtree $\rightarrow \min(3, 1) = 1$; right subtree $\rightarrow \min(2, 4) = 2$.

Player 1 maximizes: $\max(1, 2) = 2$. **Game value** $= 2$.

Optimal play: Player 1 chooses b, Player 2 responds c.

Zermelo's Theorem

Zermelo's Theorem (1913)

Determined means optimal play has a definite result: one player can force a win, or both can force a draw.

Theorem (Zermelo)
In any finite two-player perfect-information game with three outcomes (win, lose, draw), exactly one of the following holds:

1. Player 1 has a winning strategy, or
2. Player 2 has a winning strategy, or
3. Both players have strategies that guarantee at least a draw.

Proof idea: Backward induction labels the finite tree with one value in $\{\mathrm{W1}, \mathrm{Draw}, \mathrm{W2}\}$ at its root.

What Zermelo's Theorem Tells Us

Corollary: Chess is *determined*: either White wins, Black wins, or the game is a draw under optimal play. We just don't know which!

Pause and think: Chess is determined. Does this mean chess is "solved"? Why or why not?

Implications of Zermelo's Theorem

Existence is not an efficient algorithm for finding a strategy.

- The backward-induction proof enumerates chess's $\sim 10^{120}$-node tree.
- Some smaller games are solved:

Game	Solved?	Result
Tic-tac-toe	Yes	Draw
Connect Four	Yes (1988)	First player wins
Checkers	Yes (2007, Schaeffer)	Draw
Chess	No	Unknown (conjectured draw)
Go (19×19)	No	Unknown

The Minimax Theorem

Minimax for Game Trees

Backward induction alternates max and min from leaves to root. The **minimax recursion** writes that computation in one equation.

$$v(n) = \begin{cases} \max_a v(\text{child}(n, a)) & \text{Player 1 at } n, \\ \min_a v(\text{child}(n, a)) & \text{Player 2 at } n. \end{cases}$$

At a leaf, $v(\ell) = u_1(\ell)$. At the root, Player 1 guarantees at least $v(\text{root})$ and Player 2 guarantees at most that value.

Quick check: In the tiny tree, which nodes use max and which use min? Compare your answer with the worked trace.

Normal Form and Mixed Strategies: Recall

A **normal-form game** has players choose strategies without observing the other's current choice. Its payoff matrix A has Player 1's strategies as rows and Player 2's as columns; A_{ij} is Player 1's payoff when they choose row i and Player 2 chooses column j.

Notation: S_i is player i's pure-strategy set, and $\Delta(S_i)$ is the set of probability distributions over S_i.

Von Neumann's Minimax Theorem (1928)

The minimax recursion applies to game trees. Von Neumann's theorem extends it to mixed strategies in normal form.

Theorem (von Neumann, 1928)
In any finite two-player zero-sum game with payoff matrix A:

$$\max_{\sigma_1 \in \Delta(S_1)} \min_{\sigma_2 \in \Delta(S_2)} \sigma_1^\top A \, \sigma_2 \;=\; \min_{\sigma_2 \in \Delta(S_2)} \max_{\sigma_1 \in \Delta(S_1)} \sigma_1^\top A \, \sigma_2 .$$

The expression $\sigma_1^\top A \sigma_2$ is Player 1's expected payoff when both players randomize.

Reading the Minimax Equality

- **Left side:** Player 1 chooses a mixed strategy that maximizes the payoff they can guarantee against Player 2's best response.
- **Right side:** Player 2 chooses a mixed strategy that minimizes Player 1's best possible payoff.
- The two quantities are equal, so the zero-sum game has a well-defined value v^*.
- For perfect-information games, the minimax theorem specializes to backward induction.

Proof via LP duality: Lecture 3. Computational treatment: Module 2, Lecture 7.

Minimax in Normal Form vs. Extensive Form

We have now seen minimax in two settings. The table below summarizes the key differences.

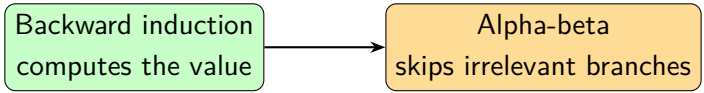
	Normal form	Extensive form
Representation	Payoff matrix	Game tree
Strategy	Row/column choice	Contingency plan
Minimax	LP over mixed strategies	Backward induction
Complexity	Polynomial (LP)	Exponential in tree depth

For perfect-information games, the extensive-form and normal-form representations yield the same game value, but the extensive form is exponentially more compact.

Alpha-Beta Pruning

Checkpoint: From Values to Efficient Search

We can now compute a perfect-information game's value by backward induction. The next question is computational: how can we avoid examining branches that cannot change that value?



Keep in mind: pruning changes search effort, never the minimax value returned at the root.

Alpha-Beta Pruning: Motivation

The minimax recursion gives us the game value exactly, but computing it requires visiting every node in the game tree. For chess, that means $\sim 10^{120}$ nodes.

Idea: Many subtrees cannot affect the root value. Prune them without changing the result.

Alpha-beta pruning maintains two bounds during search:

- α: best value Player 1 can guarantee (lower bound).
- β: best value Player 2 can guarantee (upper bound).

If at any node $\alpha \geq \beta$, the remaining children cannot affect the outcome $\Rightarrow$ **prune**.

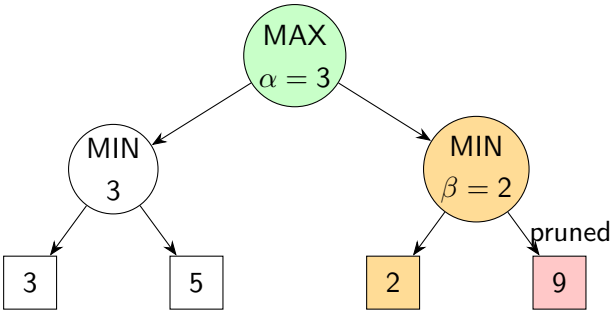
Alpha-Beta Pruning: Algorithm

AlphaBeta(n, α, β)

1. If n is a leaf, return $u_1(n)$.
2. At a Player 1 node, update $\alpha \leftarrow \max(\alpha, \text{AlphaBeta}(c, \alpha, \beta))$ for each child; if $\alpha \geq \beta$, stop (β-*cutoff*) and return α.
3. At a Player 2 node, update $\beta \leftarrow \min(\beta, \text{AlphaBeta}(c, \alpha, \beta))$ for each child; if $\beta \leq \alpha$, stop (α-*cutoff*) and return β.

Worked Alpha-Beta Trace: Find the Cutoff

Predict: after the left subtree returns 3, can the right subtree improve the root's value?



Left: $\min(3, 5) = 3$, so $\alpha = 3$. Right: first child sets $\beta = 2 \leq \alpha$; prune 9. Root value: 3.

Alpha-Beta: Complexity

Theorem

If the game tree has depth d and branching factor b:

- **Worst case:** Alpha-beta visits $O(b^d)$ nodes (no improvement over minimax).
- **Best case** (optimal move ordering): $O(b^{d/2})$ nodes.

Best-case interpretation: alpha-beta effectively **doubles** the search depth for the same computational effort.

- For chess with $b \approx 35$: going from depth 8 to depth 16 with the same number of nodes.
- In practice, good move ordering (killer heuristic, transposition tables) brings performance close to the best case.

Computational Complexity of Chess

How Hard Is Chess?

Question: Backward induction solves any finite perfect-information game. Why can't we just solve chess?

Generalized Chess

Consider the decision problem: "Does White have a winning strategy in a given $n \times n$ chess position?"

- **Fraenkel & Lichtenstein (1981):** Generalized chess is **EXPTIME-complete** (problems requiring time exponential in the input size, strictly harder than NP-complete).
- No polynomial-time algorithm exists (unless P = EXPTIME).

Definition: EXPTIME contains decision problems solvable in time $2^{p(n)}$ for some polynomial p.

Why This Does Not Contradict Backward Induction

Resolving the apparent contradiction:

- Backward induction runs in time polynomial *in the size of the game tree*, which has $O(b^d)$ nodes.
- But the game tree of generalized chess is *exponential in the board size* n. The input to the decision problem is the position (size $O(n^2)$), not the full tree.

Contrast: 8×8 Checkers was weakly solved by exhaustive search (Chinook, 2007), though generalized Checkers is also EXPTIME-complete.

Computer Chess: A Brief History

Stockfish and AlphaZero

Stockfish (2008–present):

- Open-source alpha-beta search with NNUE (efficiently updatable neural-network evaluation).
- Elo $\approx 3600+$ (far beyond any human).

AlphaZero (Silver et al.):

- Self-play reinforcement learning with MCTS (Monte Carlo tree search) and a neural network.
- Learns the policy and evaluator without handcrafted evaluation features.

Contrast: both search game trees; AlphaZero learns how to guide the search, while Stockfish combines search with engineered features.

Why LLM Games Are Different

LLMs as Game Players

Question: When two LLM agents interact strategically, can we still use the game-theoretic tools developed for chess?

- **State:** Chess has a position evaluator; an LLM game has a conversation history with no compact scoring function.
- **Actions:** Chess has enumerable legal moves; an LLM can produce unbounded text.
- **Repeatability:** A board state has a fixed value; the same LLM prompt can yield stochastic responses.

The "game" is accessible only through **sampled play**: run the agents and observe outcomes.

The Empirical-Game Pipeline

Chess: payoffs are *computed* (backward induction, alpha-beta, or exhaustive search).

LLM games: payoffs are *estimated* from repeated interaction.

1. Fix a finite set of **strategies** (prompt/model configurations).
2. **Query agent pairs** repeatedly under each strategy profile.
3. Aggregate outcomes into an **estimated payoff matrix** with confidence intervals.
4. **Analyze** the estimated game: Nash equilibria, dominated strategies, welfare.

This is the *empirical game-theoretic analysis* (EGTA) pipeline. Full treatment: Module 2 Lecture 8 and Module 7.

Demo: Two-Agent Payoff Estimation

Learning goal: estimate a payoff matrix by repeated sampling when backward induction is infeasible.

Setup: two TAMU-API models play a 2×2 Prisoner's Dilemma under fixed system prompts ("cooperate" or "defect"). Notebook: `demos/demo-payoff-estimation.ipynb`. **Time box:** 6 minutes; switch to the fixed example output if the live call runs long.

```
ssh <COURSE_SERVER>
python run_payoff_estimation.py \\
  --game pd --strategies cooperate defect \\
  --model <pinned model> \\
  --reps-per-cell 30 --seed 42
```

Demo: Read the Outputs Before We Run It

Requested outputs:

- 4-cell empirical payoff matrix (means $\pm$ 95% CIs).
- One raw transcript (system prompt $\rightarrow$ response $\rightarrow$ score).
- Total token cost vs. the \$5/day budget.

If the live call is unavailable: use the instructor's pre-recorded output to inspect the matrix, a transcript, and CI widths.

Budget accounting: 4 cells $\times$ 30 reps $=$ 120 game episodes; two agent calls per episode give 240 API calls. **This is your PA1 starting point.**

PA1: Empirical Payoff Estimation and Equilibrium Analysis

Due: Week 3.

PA1 asks how equilibrium conclusions change when payoffs are estimated rather than exact. Starting from the demo scaffold, you will:

- Design and run a **coordination game** (e.g., Battle of the Sexes).
- Design and run a **negotiation game** of your choice.
- Compute Nash equilibria with provided support-enumeration and Lemke–Howson code.
- Test sensitivity to payoff perturbations within the confidence intervals.

Report: seeds, model version, sample counts, confidence intervals for every cell, and cost within the $5/day TAMU API limit.

Summary: Theory

Theory:

- Chess is a two-player, zero-sum, perfect-information, sequential game.
- **Game trees** represent sequential interaction; strategies are complete contingency plans.
- **Backward induction** solves finite perfect-information games; **Zermelo's theorem** guarantees a determined outcome.
- The **minimax theorem** extends the game value to mixed strategies in normal form.
- **Alpha-beta pruning** achieves $O(b^{d/2})$ in the best case, preserving the exact minimax value.

Summary: From Chess to LLM Games

Practice:

- Backward induction is polynomial in the explicit tree; generalized chess ($n \times n$) is **EXPTIME-complete**.
- Computer chess: Stockfish uses alpha-beta; AlphaZero combines MCTS with learned policy and value functions.

Key takeaway: LLM games lack enumerable moves and deterministic evaluation; payoffs must be **estimated**, not computed.

Next: Nash equilibria (Lecture 3), then algorithms for the estimated games (Module 2).