Lecture 9.1: Game-Playing AI at Scale

From Self-Play to Leagues to Language: Accessible Companion Notes

CSCE 631-D: Intelligent Agents: Computational Game Solving
Alan Kuhnle, Texas A&M University

Fall 2026 (Module 9, Lecture 1; overall Lecture 20, Tuesday, November 10, 2026)

Using these notes. These notes expand the Lecture 9.1 slides into readable prose, give the short derivations the slides only state (the PFSP weighting, the PSRO loop, the OpenAI Five opponent sampler, and the piKL objective), describe every diagram and table in words, and suggest answers to the "pause and predict" prompts. Headings follow the deck in order. Slide references use **PDF page numbers in the 43-page slide deck**, counting the title page and the maroon part dividers. Page 41 is a backup slide (not presented by default), and the references occupy pages 42 and 43. Three topics appear only here, not on the slides: MuZero (Part I) and the 2025 to 2026 hidden-information results Ataraxos and Obscuro (Part VI).

Figures. The deck shows eleven figures taken from primary sources: the papers' own figures, DeepMind's and OpenAI's own publications, and the official competition report. Each is described in words below at the slide where it appears. The file list, source, figure number, and credit for every image are in `m9-lecture1-figures.md`.

Source status. Every number and date below comes from a source opened while preparing the lecture; the claim-by-claim trace is in `m9-lecture1-claims.md`. The sources differ in standing. AlphaGo, AlphaGo Zero, AlphaZero, MuZero, AlphaStar, Libratus, Pluribus, Cicero, and Ataraxos appeared in journals (*Nature* or *Science*). PSRO, "Spinning Tops," piKL, the KataGo attack, and Obscuro appeared at peer-reviewed conferences. The OpenAI Five report is an arXiv preprint. Pluto has no paper: we rely on the author's technical README, which the competition organizer links from the official competition report. One press account (Vice/Motherboard, 2019) is quoted and labeled as journalism. Two DeepMind blog posts (2019) are lab publications, not peer-reviewed, and are used only as figure sources. Where these notes interpret or speculate, they say so.

Contents

1 Why Games Keep Being the Benchmark **2**

2 Part I: AlphaGo and AlphaZero (MuZero in the Notes) **4**

3 Part II: AlphaStar and the League **6**

4 Part III: OpenAI Five **10**

5 Part IV: Strong Is Not Unexploitable **12**

6 Part V: Cicero and Diplomacy **14**

7 Part VI: Pluto, Other Recent Results, and Synthesis **17**

Appendix: Pseudocode for the Two Opponent Samplers **20**

References **20**

Notation used in this lecture

- s: a game state (a position); a: an action; o_t: an observation at time t.
- π: a policy (a map from states or histories to distributions over actions). π_i is player i's policy, π_{-i} the others'.
- u_i: player i's expected payoff. In a two-player zero-sum (2p0s) game, $u_2 = -u_1$ and v^* denotes the game value.
- $f_\theta(s) = (\mathbf{p}, v)$: a network with parameters θ returning move probabilities $\mathbf{p}$ and a value estimate v.
- $\boldsymbol{\pi}$ (bold): the visit-count distribution that MCTS returns at a position; $z \in \{-1, 0, +1\}$: the final game outcome.
- Π_i: a finite population of policies for player i; U^{Π}: the empirical payoff table among them; σ_i: a meta-strategy, that is, a mixture over Π_i.
- $\Pr[A \text{ beats } B]$: the estimated win probability of policy A against frozen policy B.
- τ_i: an anchor policy learned from human data; $\lambda_i > 0$: the anchor strength in piKL. (In Part III, τ without a subscript is OpenAI Five's team-spirit parameter, an unrelated quantity.) $D_{\mathrm{KL}}(p\|q) = \sum_a p(a) \log(p(a)/q(a))$.
- $\mathrm{expl}(\pi)$: exploitability, defined in Part IV.

1 Why Games Keep Being the Benchmark

1.1 Slide 1: Title

The title names the lecture's arc. AlphaZero improves by playing copies of itself; AlphaStar improves by playing a curated population (a league); Cicero plays a seven-player game in natural language, where neither self-play nor an equilibrium is the right target. Pluto, a 2026 StarCraft: Brood War bot, closes the lecture by returning to pure self-play in a game where earlier work had needed a league. Throughout, we read each system as a game-theoretic algorithm from Modules 2 to 7, run with neural networks and very large compute.

1.2 Slide 3: Games as AI's Yardstick, 1992 to 2026

Figure description. A horizontal timeline with seven milestones. Above the line: 1992 backgammon, 1997 chess, 2013 Atari, 2016 Go, 2019 StarCraft II and Dota 2, 2022 Diplomacy, and 2025 to 2026 Stratego, Fog of War chess, and StarCraft: Brood War. Below each milestone, in italics, is the property it added: chance, depth, pixels, breadth, real time with hidden information and teams, language with seven players, and hidden information at scale.

The first four milestones are listed in the introduction of the OpenAI Five report (Berner et al. 2019, §1). The labels below the line are our summary, not a claim from any one paper. Stratego and Fog of War chess are discussed only in these notes (Part VI). Games remain the yardstick because they offer three things real deployments rarely do: fixed rules, an unambiguous score, and expert humans to compare against.

Why the left half matters. Backgammon, chess, and Go are two-player zero-sum; Atari is a single-agent environment. StarCraft II and Dota 2 remain zero-sum between two sides (Dota is two *teams*), while Diplomacy is general-sum with seven players. The change matters because the guarantee behind self-play, "beat your copy and you approach an unbeatable policy," is a 2p0s fact (Part I).

1.3 Slide 4: What Makes the Newer Games Hard

Table, read as prose. The table compares Go, StarCraft II, Dota 2, and Diplomacy on five properties. Information: Go is perfect; StarCraft II hides units under fog and outside the camera; Dota 2 has fog of war; Diplomacy has private negotiation. Timing: alternating turns in Go, real time in StarCraft and Dota, simultaneous moves in Diplomacy. Actions per step: Go's breadth is about 250; StarCraft II has about 10^{26} possible actions per step; OpenAI Five chose among 8,000 to 80,000 actions per step; Diplomacy requires orders for every unit plus language. Players: two, two, two teams of five, and seven. Payoffs: zero-sum, zero-sum with cyclic strategies, two-team zero-sum, and general-sum.

The slide omits one row of the earlier version, the horizon: Go's typical depth is about 150 moves; a StarCraft II game is tens of thousands of time-steps; Dota 2 is about 20,000 decisions over about 45 minutes; and in Diplomacy Cicero sent and received about 292 messages per game. The numbers come from Silver et al. (2016, p. 484), Vinyals et al. (2019, pp. 1 to 2), Berner et al. (2019, §2), and FAIR et al. (2022, p. 1067). Of these properties, nontransitivity is the one this course is best equipped to analyze. If strategy A beats B, B beats C, and C beats A, then the learning rule "replace my policy by a best response to my current policy" can cycle forever. Module 3's fictitious play addresses exactly this problem: it best-responds to the *empirical average* of past play, not to the latest policy.

1.4 Slide 5: Today: Learning Objectives and Plan

There are four objectives: (1) explain why self-play tends to work in 2p0s perfect-information games and where the argument breaks; (2) compute prioritized fictitious self-play (PFSP) probabilities and read AlphaStar's league as PSRO; (3) separate average-case strength from exploitability; (4) state the piKL objective and say why Cicero does not simply play an equilibrium. The six parts are perfect information (Part I), AlphaStar (II), OpenAI Five (III), exploitability (IV), Cicero (V), and Pluto with a synthesis (VI).

Pacing plan for 75 minutes. Why games and the poker recap, 6 minutes; Part I, 9; Part II, 18; Part III, 10; Part IV, 7; Part V, 11; Part VI with the synthesis, 9. That is 70 planned minutes, leaving a 5-minute buffer for questions or one discussion question. The slide on defenses against exploitation is a backup (page 41); the remaining discussion questions can serve as project or reading-response prompts.

1.5 Slide 6: Recall: Where Poker Left Us

This is the only poker slide; the summer Module 9 lectures on the poker case study and Deep CFR covered the details. Libratus (Brown & Sandholm 2018) played heads-up no-limit hold'em, a 2p0s game. It combined three modules: a blueprint computed by solving an abstraction of the game, nested subgame solving during play, and a self-improver that patched weaknesses opponents had found. It defeated four top specialist professionals in a 20-day, 120,000-hand competition (pp. 418 to 419). Pluribus (Brown & Sandholm 2019) played six-player no-limit hold'em. Its blueprint came from Monte Carlo CFR self-play, computed in 8 days on a 64-core server for about $144 at cloud spot rates, and it searched in real time with depth-limited search over a few continuation strategies. The authors are explicit that their algorithms "are not guaranteed to converge to a Nash equilibrium outside of two-player zero-sum games"; the goal was to defeat humans empirically (p. 886).

The point for today is the target. In 2p0s games, CFR or self-play aims at a Nash equilibrium, and a Nash equilibrium is unexploitable in expectation. The rest of the lecture follows that target as the games get too large to solve, cyclic, team-based, and finally general-sum with communication.

2 Part I: AlphaGo and AlphaZero (MuZero in the Notes)

2.1 Slide 8: AlphaGo (2016)

Figure description. Panel a of Figure 1 of Silver et al. (2016). Four small network icons sit in a row, labeled rollout policy p_π, SL policy network p_σ, RL policy network p_ρ, and value network v_θ. Arrows labeled "classification" run from a stack of human expert positions up to the first two networks. A thick arrow labeled "policy gradient" runs from p_σ to p_ρ. An arrow labeled "self play" runs from p_ρ down to a stack of self-play positions, and an arrow labeled "regression" runs from those positions up to v_θ.

Exhaustive game-tree search costs roughly b^d positions, where b is the number of legal moves per position and d the game length; for Go, $b \approx 250$ and $d \approx 150$ (Silver et al. 2016, p. 484). Monte Carlo tree search (MCTS) grows a search tree selectively, spending simulations on promising lines. AlphaGo adds learned guidance in four steps; the figure shows the training pipeline for the first three. (The fast rollout policy p_π in the figure is used inside search; the slide omits it.)

1. A *supervised policy network* $p_\sigma(a \mid s)$ is trained to predict expert human moves. In search it narrows the breadth.
2. A *reinforcement-learning policy network* p_ρ starts from p_σ and is improved by policy gradient in games between the current network and a randomly selected previous iteration (p. 485). Sampling opponents from a pool of past versions is already a mild form of fictitious self-play, a theme that returns in Part II.
3. A *value network* $v_\theta(s)$ is trained to predict the winner of games played by p_ρ from position s. In search it truncates the depth: instead of playing to the end, evaluate.
4. MCTS combines these at play time.

AlphaGo won 99.8% of games against other Go programs and defeated the European Go champion 5 games to 0, the first time a program beat a human professional in the full-sized game (abstract). Strictly, v_θ is trained to predict outcomes under the RL policy p_ρ, so it estimates that policy's value; the course reading is that, to the extent p_ρ plays strongly, it serves as a stand-in for the minimax value $v^*(s)$ of a perfect-information 2p0s game, the quantity that backward induction (M4) computes exactly on trees small enough to enumerate.

2.2 Slide 9: AlphaGo Zero and AlphaZero

Figure description. Three boxes in a loop. Box one: the network $(\mathbf{p}, v) = f_\theta(s)$. An arrow leads to box two: MCTS guided by f_θ, which returns a search policy $\boldsymbol{\pi}$. An arrow leads to box three: self-play games in which moves are sampled from $\boldsymbol{\pi}$ and the outcome z is recorded. A return arrow from box three to box one is labeled "train: $\mathbf{p} \to \boldsymbol{\pi}$, $v \to z$."

AlphaGo Zero (Silver et al. 2017) removes human data and merges the two networks into one, $(\mathbf{p}, v) = f_\theta(s)$. Training minimizes

$$\ell = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c\|\theta\|^2,$$

where z is the game outcome, $\boldsymbol{\pi}$ the MCTS visit distribution, and c an L_2 regularization weight (author manuscript, Eq. 1, p. 6). The paper describes MCTS as "a powerful policy improvement operator" (p. 3): its output $\boldsymbol{\pi}$ is usually stronger than the raw prior $\mathbf{p}$. Fitting $\mathbf{p}$ to $\boldsymbol{\pi}$ is then the improvement step of policy iteration, and fitting v to z is the evaluation step. Starting from random play, AlphaGo Zero beat the published, champion-defeating AlphaGo by 100 games to 0 (abstract). AlphaZero (Silver et al. 2018) applies the same algorithm, given only the rules, to chess, shogi, and Go, reaching a superhuman level within 24 hours of training and defeating a world-champion program in each case (arXiv:1712.01815, abstract). The slide's last bullet points to the next subsection: MuZero learns the model as well.

2.3 MuZero (2020) (Notes Only)

AlphaZero's search calls the game's true transition function. MuZero (Schrittwieser et al. 2020) learns three functions instead: a *representation* $s^0 = h_\theta(o_1, \ldots, o_t)$ mapping observations to a hidden state; a *dynamics* function $r^k, s^k = g_\theta(s^{k-1}, a^k)$ that predicts an immediate reward and the next hidden state for a hypothetical action; and a *prediction* function $\mathbf{p}^k, v^k = f_\theta(s^k)$ (arXiv:1911.08265, pp. 3 to 4). The tree search runs entirely inside this learned model. The hidden state is never asked to reconstruct the screen; it is trained only to make reward, policy, and value predictions accurate, which is all planning needs. On 57 Atari games MuZero set a new state of the art, and on Go, chess, and shogi, without the rules, it matched AlphaZero's superhuman performance (abstract). For this course MuZero is the plainest example of planning in a learned, task-specific abstraction (compare M6), which is the situation of an LLM agent that has no simulator of its environment.

2.4 Slide 10: Why Self-Play Tends to Work Here

The Cicero paper states the standard view: "In finite 2p0s games, certain reinforcement learning (RL) algorithms that learn by playing against themselves ... will converge to a policy that is unbeatable in expectation in balanced games" (FAIR et al. 2022, p. 1067). Three facts from earlier modules make this plausible for AlphaZero-style training.

1. **Zero-sum.** By the minimax theorem every 2p0s game has a value, and all equilibrium strategies of a player are interchangeable (M2). So there is a single, well-defined target. Whether a particular learning process reaches it is a separate question.
2. **Perfect information.** Backward induction yields a pure equilibrium (M4), and each position has a value $v^*(s)$ that does not depend on how play reached it. Sokota et al. (2026) make the same contrast: in perfect-information games the right move is the one that "maximizes the value of the resulting position," while under imperfect information the value of a decision depends on the policies used before and at the time of the decision.

3. **Search as improvement.** MCTS with a learned value function is an approximate lookahead, so the AlphaZero loop is approximate policy iteration. Exact policy iteration in a finite 2p0s perfect-information game has the minimax value v^* as its fixed point; the approximate version carries no such guarantee.

This is intuition, not a theorem about deep networks. AlphaStar's authors warn that self-play "may chase cycles ... indefinitely without making progress" (Vinyals et al. 2019, p. 3), and in Part IV a superhuman AlphaZero-style Go program loses consistently to a weak adversary.

Pause and predict. StarCraft II keeps zero-sum but loses perfect information (fact 2) and, with its cyclic strategy space, makes the "single target" of fact 1 hard to reach by naive self-play even though it exists. Diplomacy loses fact 1 entirely: it is general-sum with seven players, so there is no game value and equilibria are not interchangeable.

3 Part II: AlphaStar and the League

3.1 Slide 12: AlphaStar: The Game and the Interface

Figure description. A StarCraft II game screen from DeepMind's January 2019 blog post: a Protoss base with buildings and units, a minimap in the lower left, and a score bar comparing AlphaStar and the professional player LiquidTLO on supply, minerals, gas, workers, army, and actions per minute. The match was played in December 2018 by an earlier agent than the one in the 2019 *Nature* paper, so the on-screen numbers say nothing about the constraints described below.

AlphaStar (Vinyals et al. 2019) plays full 1v1 StarCraft II for all three races. The game involves hundreds of units, imperfect information, about 10^{26} possible actions per step, and tens of thousands of time-steps per game (pp. 1 to 2). The interface was designed with professional players and Blizzard employees to approximate human constraints (Methods, p. 11): the agent sees full detail only inside a movable camera view, and moving the camera is an action; a monitoring layer limits it to at most 22 non-duplicate actions in any five-second window; and in live play there is a delay of about 110 ms between observing a frame and executing an action. The action-rate limiter is an external monitor wrapped around an agent's interface, the same design pattern as the monitors of Module 8.

Training has two stages. First, supervised learning on 971,000 replays from players with MMR above 3500, the top 22% (Methods, p. 13). The policy is conditioned on a statistic z that summarizes a strategy sampled from human data, for example a build order (p. 2), which keeps the imitation policy diverse. AlphaStar Supervised reached an average rating of 3,699 MMR, above 84% of human players (p. 4). Second, reinforcement learning in a league.

3.2 Slide 13: Why Plain Self-Play Is Not Enough: Cycles

Figure description. Figure 1 of DeepMind's October 2019 AlphaStar blog post. On the left, rock, paper, and scissors sit on a cycle, with the note that StarCraft units have balanced strengths and weaknesses in the same way. On the right, two rows of three triangle plots show which units (void rays, stalkers, immortals) an agent builds. Top row, self-play: the agent starts with a mixture of units, specializes, and its specialization narrows. Bottom row, with exploiters: exploiters highlight a weakness, and the agent moves to more robust strategies. The blog explains that naive reinforcement learning focuses on strategies that are easier to improve.

In rock-paper-scissors (RPS), self-play that best-responds to its latest policy produces the sequence Rock, Paper, Scissors, Rock, and so on: it never settles. Fictitious self-play (FSP) best-responds instead to the uniform mixture of all previous policies. AlphaStar's paper summarizes the classical result that this mixture "converges to a Nash equilibrium in two-player zero-sum games" (p. 3). That classical guarantee assumes exact (or suitably controlled) best responses; with deep-RL approximate best responses it is a heuristic, not a theorem, and in RPS the time-averaged policy approaches $(1/3, 1/3, 1/3)$. This is the deep-RL version of fictitious play from Module 3. Its weakness is efficiency: "many games are wasted against players that are defeated in almost 100% of games" (Methods, p. 15), and a rare but strong counter-strategy receives only its uniform share of attention.

3.3 Slide 14: Prioritized Fictitious Self-Play (PFSP)

Definition. Given a learner A and candidate set $\mathcal{C}$ of frozen opponents, PFSP samples opponent B with probability

$$\Pr[B] = \frac{f(x_B)}{\sum_{C \in \mathcal{C}} f(x_C)}, \qquad x_B = \Pr[A \text{ beats } B], \tag{1}$$

for a weighting function $f : [0, 1] \to [0, \infty)$ (Methods, pp. 15 to 16), provided at least one candidate has $f(x_C) > 0$. (If the learner beats every candidate with certainty, f_{hard} gives all of them weight zero and the rule needs a fallback, such as uniform sampling.) Two choices are used. $f_{\text{hard}}(x) = (1 - x)^p$ with $p > 0$ is the default; since $f_{\text{hard}}(1) = 0$, no games are played against opponents the agent always beats. $f_{\text{var}}(x) = x(1 - x)$ favors opponents near the learner's own level and is used for main exploiters and struggling main agents. Uniform FSP corresponds to $f \equiv 1$.

PFSP comes before the league slide so that the term is defined before the league diagram uses it.

Example 1 (the slide's table)**.** Let $x = (0.9, 0.5, 0.2)$ for opponents B_1, B_2, B_3.

- f_{hard}, $p = 1$: weights $(0.1, 0.5, 0.8)$, sum 1.4, probabilities $(0.071, 0.357, 0.571)$.
- f_{hard}, $p = 2$: weights $(0.01, 0.25, 0.64)$, sum 0.90, probabilities $(0.011, 0.278, 0.711)$.
- f_{var}: weights $(0.09, 0.25, 0.16)$, sum 0.50, probabilities $(0.18, 0.50, 0.32)$.

Note that f_{var} prefers B_2, the even match, over B_3, the opponent the learner usually loses to; f_{hard} prefers B_3.

Why "smooth max-min." The paper calls f_{hard} "a form of smooth approximation of max-min optimisation, as opposed to max-avg that FSP imposes" (p. 16). A short calculation makes this precise. Suppose $x_{\min} = \min_C x_C$ is attained by a unique opponent $B^\star$. Then for every other opponent C,

$$\frac{\Pr[C]}{\Pr[B^\star]} = \left(\frac{1 - x_C}{1 - x_{\min}}\right)^p \xrightarrow[p \to \infty]{} 0,$$

since $1 - x_C < 1 - x_{\min}$. So as $p \to \infty$ the learner trains only against the opponent it beats least often, and improving its expected win rate under PFSP approaches improving $\min_C x_C$: a best response to the worst case in the pool. Uniform FSP instead improves the average $\frac{1}{|\mathcal{C}|} \sum_C x_C$, which can be high while one strong counter-strategy still wins. The paper's motivation is exactly this: "in highly non-transitive games like StarCraft, ... the pursuit of the mean win-rate might lead to policies that are easy to exploit" (p. 16).

3.4 Slide 15: The League: Three Kinds of Players

Figure description. A green box on the left, "Main agents (3): 35% self-play, 50% PFSP, 15% forgotten players and exploiters," points to a gold box in the center, "League pool: frozen snapshots, about 900 by the end." On the right, a red box "Main exploiters (3): current main agents only" and an orange box "League exploiters (6): PFSP versus the whole pool" both point into the pool.

League training keeps a growing pool of frozen snapshots and three kinds of learners that differ only in whom they play, when they are snapshotted, and whether they are reset (Methods, p. 16).

- **Main agents** train with 35% self-play, 50% PFSP against all past league players, and 15% PFSP against "forgotten" main players they can no longer beat and past main exploiters. A snapshot joins the pool every $2 \cdot 10^9$ steps. Main agents never reset; they are the product.
- **Main exploiters** play only the current main agents. They join the pool when they beat all three main agents in more than 70% of games (or after a timeout) and are then reset to the supervised policy. Their job is to find weaknesses of the current main agents.
- **League exploiters** use PFSP against the whole pool, join it when they beat every league player in more than 70% of games (or after a timeout), and are then reset with probability 25%. They find "global blind spots in the League (strategies that no player in the League can beat, but that are not necessarily robust themselves)."

The run used 12 learners, one main agent, one main exploiter, and two league exploiters per race (Methods, p. 17), each trained on 32 third-generation TPUs for 44 days; almost 900 distinct players were created (p. 4).

3.5 Slide 16: The League as PSRO in Practice

Table, read as prose. Four meta-solver choices and what PSRO becomes under each. If the meta-strategy puts all weight on the latest policy, PSRO is self-play, also called iterated best response. If it is uniform over past policies, PSRO is fictitious play. If it is a Nash equilibrium of the meta-game and there are two players, PSRO is double oracle. If it is the PFSP distribution together with dedicated exploiter roles, PSRO resembles AlphaStar's league.

Module 7 introduced policy-space response oracles (PSRO) as the deep-RL version of the double-oracle method. Algorithm 1 of Lanctot et al. (2017) reads as follows.

> **PSRO.** Input: initial policy sets Π_i for each player.
>
> 1. Estimate the empirical payoff table U^{Π} for every joint profile in Π by simulation. Initialize each meta-strategy σ_i uniform over Π_i.
> 2. Repeat for epochs $e = 1, 2, \ldots$:
>   (a) For each player i: for many episodes, sample $\pi_{-i} \sim \sigma_{-i}$ and train a new oracle policy π_i' by reinforcement learning against it. Add π_i' to Π_i.
>   (b) Fill in the missing entries of U^{Π} by simulation.
>   (c) Compute new meta-strategies σ from U^{Π} with the meta-solver.
> 3. Output σ_i for each player.

Lanctot et al. show that iterated best response, independent RL, and fictitious play are instances of PSRO with particular meta-strategies, and that double oracle is the instance with $n = 2$ and σ a Nash equilibrium of the meta-game (p. 4). They also note why neither extreme is ideal: fictitious

play "can only sharpen the meta-strategy distribution by repeatedly generating the same best responses," while best responses to an equilibrium "will ... overfit to a specific equilibrium" (p. 4).

AlphaStar's authors place league training in this family: PSRO "is related to League training, although League training specifies unique targets for approximate best responses (i.e. PFSP and exploiters)" (Methods, p. 13). Their central coordinator "maintains an estimate of the payoff matrix" (p. 17), which is U^{Π}. Our reading: the main agents run PSRO with a PFSP meta-solver, while exploiters are additional oracles whose training target is a single opponent (the current main agents) or the pool's blind spots, rather than the meta-strategy. PSRO permits this, but nothing in the PSRO template prescribes it.

3.6 Slide 17: Real-World Games Look Like Spinning Tops

Figure description. Figure 1 of Czarnecki et al. (2020), "Game geometry" and "Game profile." A spinning top stands on a disc labeled "non-transitive cyclic dimensions." Its vertical axis is the transitive dimension. Labels on the left mark, from bottom to top: agents trying to lose (the narrow tip at the bottom), an extremely non-transitive band (the widest part of the top, shaded, citing their Theorem 1), a region where non-transitivity gradually disappears, and the Nash of the game at the narrow top. A magnified circle shows strategies as nodes with many cyclic "beats" arrows. The "Game profile" plot on the right shows the length of the longest cycle (or Nash cluster size) as a function of transitive strength: it peaks in the middle band and falls toward both ends.

Czarnecki et al. (2020) propose that many real-world 2p0s games share this geometry: strategies vary along a transitive axis of skill and a cyclic axis, and the amount of non-transitivity is largest at intermediate skill and shrinks toward the top. Three results anchor the slide.

- **Long cycles exist.** Proposition 1 shows that Go contains a cycle of length at least 2^{1000} (p. 5).
- **Populations can guarantee progress.** The authors define *Nash clusters* by repeatedly peeling off the support of a Nash equilibrium of the remaining strategies. Theorem 3: if the training population contains a full Nash cluster C_i, then finding a policy π that beats every member of the population ($f(\pi, \pi_j) > 0$ for all π_j) guarantees that π lies in a strictly stronger cluster (p. 7). Proposition 3 gives a related convergence condition for fixed-memory fictitious play with a large enough population (p. 6).
- **Not universal.** In their experiments, games that are not "Games of Skill," such as the Disc game, cycle for every population size, while in a fully transitive game (their Elo game) naive self-play already converges (Table 1 caption, p. 8).

The authors connect this to AlphaStar directly: AlphaStar "explicitly attempts to cover the non-transitivities using exploiters" (p. 7). In the picture, exploiters try to widen the population's coverage of the current band of the top, and main agents push upward. Theorem 3's premise (a full Nash cluster in the population) is not something AlphaStar's paper verifies, so the connection is a motivation, not a guarantee.

3.7 Slide 18: Measuring the League With Nash

Two equilibrium-based measurements appear in the AlphaStar paper.

- **Nash distribution over the league.** Treat every league player as a pure strategy of a symmetric meta-game with the empirical payoff matrix and compute a Nash equilibrium. Over training, it "puts the most weight on recent players, suggesting that latest strategies largely dominate the previous ones, without much forgetting or cycling" (Fig. 4 caption).

- **Relative population performance (RPP).** For leagues A and B with payoff matrix P_{AB}, solve the meta-game between them for $(\mathbf{p}_A, \mathbf{p}_B)$ and set RPP $= \mathbf{p}_A^\top P_{AB}\, \mathbf{p}_B$ (Methods, p. 19). High RPP means A can form a mixture that exploits B while not being too exploitable by any mixture from B. Note the restriction: RPP measures relative strength under equilibrium play between two finite populations. It is not exploitability, which asks about the worst case over *all* strategies, and the two can differ greatly. (The paper's Fig. 3 caption says RPP "measures exploitability"; we keep the narrower reading.)

Extended Data Fig. 8 plots the Protoss-versus-Protoss payoff matrix; its caption reports that across the full payoff matrix there are about 3,000,000 rock-paper-scissors cycles (each edge requiring at least a 70% win rate) that involve at least one exploiter, and about 200 that involve only main agents. The main agents behave transitively.

3.8 Slide 19: AlphaStar's Results

Figure description. Figure 2A of Vinyals et al. (2019). Match making rating (MMR, 0 to 7,000) is plotted against the percentile of ranked human players on Battle.net, with badges for the leagues from Bronze to Grandmaster along the bottom. A star marks AlphaStar Supervised at about 3,700, in the Diamond range. AlphaStar Mid and AlphaStar Final sit at the far right, in Grandmaster; an inset magnifies the top percentile.

MMR is Blizzard's Elo-like matchmaking rating (Methods, p. 13). AlphaStar Final reached 6,275 MMR as Protoss, 6,048 as Terran, and 5,835 as Zerg: Grandmaster level for all three races, above 99.8% of ranked human players on the European server (p. 4; Methods p. 17).

The action-rate ablation and the pause question. In Fig. 3G, test Elo is 1540 at the action-rate limit actually used, 1145 at 10% of that limit, and 1392 with no limit at all. The authors' tentative explanation is that with more actions "the agent spends more effort refining micro-tactics than learning diverse strategies" (Fig. 3 caption). A game-theoretic reading: a constraint on the action space can make exploration easier by shrinking the space the learner has to cover, a theme from M6 abstraction.

4 Part III: OpenAI Five

4.1 Slide 21: OpenAI Five: Mostly Self-Play, at Very Large Scale

Figure description. Figure 11a of Berner et al. (2019): a Dota 2 game screen from game 1 of the OpenAI Five Finals against OG, with heroes fighting near a tower, the minimap in the lower left, and ability bars along the bottom. In the paper, the screenshot illustrates the bot's predictions just before a tower falls.

OpenAI Five (Berner et al. 2019, arXiv preprint) played Dota 2 with five heroes per team, restricted to 17 of the game's 117 heroes (p. 3). One policy, a recurrent network with a single 4096-unit LSTM and about 159 million parameters, controls each hero through replicas with identical parameters (pp. 3 to 4). Training used Proximal Policy Optimization with generalized advantage estimation (p. 5). Rollout workers "play the latest policy against itself for 80% of games, and play against older policies for 20% of games" (p. 5).

The opponent sampler, and why it resembles a bandit. Each past opponent $i = 1, \ldots, N$ has a quality score q_i and is sampled with probability $p_i \propto e^{q_i}$. Every 10 iterations the current agent joins the pool with quality equal to the current maximum. After a game against i, nothing changes if i won; if the current agent won,

$$q_i \leftarrow q_i - \frac{\eta}{N\, p_i}, \qquad \eta = 0.01 \tag{2}$$

(Appendix N, Eq. 14, p. 59). Two observations, which are our interpretation rather than the paper's: first, this is a softmax (exponential-weights) distribution, as in Hedge from Module 3. Second, dividing by p_i is the importance-weighting correction familiar from adversarial bandits (Exp3): opponents are observed only when sampled, so a loss to an *infrequently* sampled opponent receives a proportionally larger update. The effect is that opponents the current agent keeps beating fade out, and the pool's probability mass stays on opponents that still win games. Compared with PFSP (Eq. 1), both schemes down-weight opponents the learner already beats; PFSP does so with a fixed function of an estimated win rate, OpenAI Five with an online multiplicative update.

OpenAI Five was a single training run from June 30, 2018 to April 22, 2019. It trained for 180 days spread over 10 months of real time and used 770 ± 50 PFlops/s-days of compute (p. 8).

4.2 Slide 22: Team Spirit

Table, read as prose. Hero 1 earns a raw reward of 5 and the other four heroes earn 0, so the team mean is 1. With team spirit 0, hero 1 receives 5 and the others 0. With team spirit 0.3, hero 1 receives 3.8 and each other hero 0.3. With team spirit 1, every hero receives 1.

OpenAI Five transforms rewards twice (Appendix G). Across teams, "we ensure that all our rewards are zero-sum, by subtracting from each hero's reward the average of the enemies' rewards" (p. 43). Within a team, a team-spirit parameter $\tau \in [0, 1]$ mixes individual and shared reward:

$$r_i = (1 - \tau)\, \rho_i + \tau\, \bar{\rho}, \qquad \bar{\rho} = \tfrac{1}{5} \textstyle\sum_{j=1}^{5} \rho_j$$

(Eq. 12, p. 45). At $\tau = 0$ each hero is rewarded only for its own actions; at $\tau = 1$ every reward is split equally. The authors "care about optimizing for team spirit $\tau = 1$," but found that lower values reduce gradient variance early in training, and OpenAI Five annealed τ from 0.3 to 1.0 (Table 2, p. 31).

Game-theoretic reading and the pause question. With $\tau = 1$, the five heroes on one side play an identical-interest game: their payoffs coincide, so any profile that maximizes team reward is an equilibrium among teammates. The difficulty is credit assignment: a hero's gradient reflects everyone's actions, so the signal about its own contribution is noisy. With $\tau = 0$ the signal is sharp, but incentives are misaligned: a hero is paid for a kill even when letting a teammate take it would help the team more. Annealing τ upward trades the first problem for the second as skill grows. The Cicero paper adds a structural remark: Dota 2 "is two-team zero-sum but with unlimited information sharing between teammates, which makes the game equivalent to 2p0s" (FAIR et al. 2022, note 12). That is why OpenAI Five can treat the team as one player for equilibrium purposes.

4.3 Slide 23: Surgery

During the ten months, the code, the observation and action spaces, the network size, and the game version all changed (§3.3). Restarting after each change was too expensive, so the team

developed *surgery*: offline operations that transform the old policy π_θ into a new policy $\hat{\pi}_{\hat{\theta}}$ compatible with the new environment, chosen where possible so that "the new policy implements the same function from observed states to action probabilities" (p. 8), as in Net2Net-style function-preserving transformations. The team performed over twenty surgeries, about one every two weeks (pp. 2, 8); the move to Dota 2 version 7.21d happened eight days before the match against OG (p. 7).

To check what surgery cost, the team trained a second agent, "Rerun," from scratch with the final environment and architecture. Rerun took two months and 150 ± 5 PFlops/s-days, about 20% of OpenAI Five's resources, and reached over 98% win rate against the final OpenAI Five. Restarting after each of the twenty major surgeries would have taken an estimated 40 months instead of 10 (§4.2, p. 10). For this course, surgery is a warm start for a best-response oracle in a game whose rules drift, the situation of a deployed LLM agent whose tools change underneath it.

4.4 Slide 24: Results: World Champions, Then the Public

Figure description. Figure 3 of Berner et al. (2019): OpenAI Five's TrueSkill rating (0 to about 255) against training compute in petaflop/s-days (0 to about 800). The curve rises almost vertically past calibration opponents (random, hand-scripted, an amateur test team, a semi-pro test team, and a team of casters) within the first 50 petaflop/s-days, then climbs slowly. Green dots mark professional matches won, ending with OG, the world champions, at the far right.

On April 13, 2019, OpenAI Five defeated OG, the reigning Dota 2 world champions, 2 to 0 in a best-of-three (p. 8). From April 18 to 21, 2019, OpenAI ran OpenAI Five Arena, opening the bot to the public "to explore whether OpenAI Five could be consistently exploited by creative or out-of-distribution play." It played 3,193 teams in 7,257 games and won 99.4%; 29 teams beat it, for 42 losses in total (p. 8). Footnote 9 adds a caveat: abandoned games were counted as wins for the bot, 3,140 of its 7,215 wins.

A contemporaneous press account (Vice/Motherboard, April 2019, journalism and not peer-reviewed) quotes OpenAI's Greg Brockman: winning teams "take advantage of every single weakness in Five ... gradually build up resources, and most importantly, never engage Five in a fair fight." The same article reports that players noticed the bots "never try to deny towers" and are bad at split-pushing. We cite this only to illustrate the kind of play that beat the bot; the quantitative results come from the paper.

The lesson is the difference between two numbers. A 99.4% win rate is an average over whoever showed up. It says little about how the best response to the policy would fare, and self-play, which only ever trains against itself, leaves its blind spots exactly where no self-play opponent went.

5 Part IV: Strong Is Not Unexploitable

5.1 Slide 26: Two Ways to Measure a Policy

Definition 1 (average-case strength and exploitability)**.** Fix a finite 2p0s game with value v^* and payoff u_1 to the row player. For a row strategy π and a distribution $\mathcal{D}$ over column strategies, the *average-case strength* of π against $\mathcal{D}$ is $\mathbb{E}_{\sigma\sim\mathcal{D}}\, u_1(\pi, \sigma)$. The *exploitability* of π is

$$\mathrm{expl}(\pi) := v^* - \min_{\sigma} u_1(\pi, \sigma) \;\geq 0.$$

By the minimax theorem, $\mathrm{expl}(\pi) = 0$ exactly when π is a maxmin strategy, that is, an equilibrium strategy for the row player (M2). Elo, MMR, and Arena win rates estimate the first quantity, with $\mathcal{D}$ equal to the population of opponents actually met. CFR (M5) drives the second

quantity to zero, and in small games we can compute it exactly by solving for the best response. For Go, StarCraft, or Dota nobody can compute $\min_\sigma$ exactly, so the practical proxy is to *train* an approximate best response against a frozen copy of π and measure its win rate. AlphaStar's main exploiters do this during training; the KataGo study does it after training.

Pause and predict. "Wins 99% against every human it has played" is a statement about $\mathcal{D} =$ those humans. The strongest justified claim is a *lower* bound on average-case strength against that population, together with an upper bound on how much those particular opponents could exploit it. It gives no upper bound on $\text{expl}(\pi)$, because none of those opponents was a trained best response.

5.2 Slide 27: Adversarial Policies Beat Superhuman Go AIs

Figure description. Figure 1.1a of Wang et al. (2023): a full 19 by 19 Go board late in a game. The cyclic adversary plays white against KataGo (black, searching 10 million visits per move). A large loop-shaped group of black stones, each marked with a red cross, winds through the board; a triangle marks the point where white's next move captures the whole group.

Wang et al. (ICML 2023) attacked KataGo, which they chose because it was "the strongest publicly available Go AI system at the time of writing" (p. 4) and is trained by an AlphaZero-style process. They froze KataGo and trained an adversary by RL against it; inside the adversary's search, the victim's moves are predicted with the victim's own network (abstract, §3). Their findings, from the abstract and §1:

- The adversary wins more than 97% of games against KataGo at superhuman search settings, and more than 99% against KataGo without search, using less than 14% of the compute used to train KataGo.
- The attack transfers zero-shot to other superhuman Go AIs, and it is comprehensible enough for human experts to execute. An author who is a Go expert won 14 of 15 games against JBXKata005, a strong KataGo-based bot on the KGS server (Appendix, p. 43 of arXiv v4).
- "Despite being able to beat KataGo, our adversarial policies lose against even amateur Go players" (p. 1; Appendix J.1).
- "The core vulnerability uncovered by our attack persists even in KataGo agents adversarially trained to defend against our attack" (abstract).

The slide states the paper's Fig. 2.1 cycle in words: the adversary beats a superhuman program that beats professionals, who beat amateurs, who beat the adversary. Strength in Go is therefore not a single number even at the top, and self-play's strength holds on the distribution of positions self-play visits. The adversary wins by steering the game into positions no self-play game reached. In the language of the previous slide, KataGo's average-case strength is superhuman while its exploitability is large.

5.3 Slide 41: What Defends Against Exploitation (Backup Slide)

Table, read as prose. Five tools, where we met them, and where they appear today. Equilibrium computation with a certificate: CFR and exploitability bounds in M5; used in the Libratus and Pluribus blueprints, with abstraction from M6. Population training: fictitious play in M3 and PSRO and double oracle in M7; used in AlphaStar's league and OpenAI Five's 20% of games against past opponents. Dedicated exploiters: best-response oracles in M7; used as AlphaStar's main and league

exploiters. Red-teaming after training: monitoring and the adversary who read the paper in M8; used in OpenAI Five Arena and the KataGo adversaries. Safe exploitation: safe best response in M2; not used by these systems.

Every row buys worst-case robustness with compute or average-case strength. AlphaStar's league ran twelve learners for 44 days rather than one. Defense after the fact is also hard: in the KataGo study, adversarial training did not remove the core vulnerability. Whether safe best response (M2), which guarantees not to do worse than the equilibrium value while exploiting opponents, can be made to work at the scale of StarCraft is an open question we pose for discussion; we found no source that does it.

6 Part V: Cicero and Diplomacy

6.1 Slide 29: Beyond Two-Player Zero-Sum: What Is the Target?

In Diplomacy, seven players compete to control supply centers on a map; each turn's moves happen "simultaneously after nonbinding, private negotiations" (FAIR et al. 2022, p. 1067). Two facts from M2 explain why "compute an equilibrium and play it" stops being a recipe. General-sum games typically have many equilibria with different payoffs, and equilibria are not interchangeable: if I play my part of one equilibrium and you play yours of another, the result need not be an equilibrium and can be bad for both of us. Which equilibrium is reached depends on conventions, and in Diplomacy the conventions are those of the human players at the table.

The FAIR team reports exactly this failure. In dialogue-free Diplomacy, "a self-play algorithm that achieved superhuman performance in 2p0s versions of the game performed poorly in games with multiple human players owing to learning a policy inconsistent with the norms and expectations of potential human allies" (p. 1067). More generally, "self-play without human data is no longer guaranteed to find a policy that performs well with humans, even with infinite compute and model capacity." The new target is a strong best response to a realistic model of the other players, and the model must come from human data. piKL is the tool that provides it.

Pause and predict. Pluribus had six players but competed in a game with no communication and no possibility of cooperation that benefits two players at a third's expense in a binding way; each hand is close to "everyone against everyone," and Pluribus's authors chose empirical strength against humans as their goal. Diplomacy rewards forming and keeping alliances through conversation, so a policy that is strong against copies of itself but strange to humans loses the allies it needs.

6.2 Slide 30: Cicero's Architecture

Figure description. Five boxes in a row, left to right: "Board state, dialogue history"; "Strategic reasoning (piKL)"; "Intents: planned actions"; "Dialogue model (2.7B params)"; "Filters." A box "Action (orders)" hangs below the strategic-reasoning box. A return arrow from the filters to the first box is labeled "message sent; new messages recompute intents."

Cicero (FAIR et al. 2022) couples a strategic reasoning module with a dialogue model through *intents*: a set of planned actions for Cicero and its conversation partner for the current turn and possibly future turns. The dialogue model is a 2.7-billion-parameter Transformer encoder-decoder, further trained on human Diplomacy games and conditioned on intents, so that its messages are about the plan the planner chose (p. 1068). Generated messages pass through filters designed to limit messages that "are nonsensical, inconsistent with intents, or strategically poor" (Fig. 1

caption). During a turn's negotiation, intents are recomputed every time Cicero sends or receives a message, and at the end of the turn Cicero plays its most recently computed intent (p. 1068).

6.3 Slide 31: Intents in Action

Figure description. One of the three panels of Figure 3 of FAIR et al. (2022). The heading "Depicted intents" reads: England convoys an army to Belgium with the support of France and Germany while taking Norway in a manner friendly to Russia. Below it, a Diplomacy map of northern Europe shows these planned orders as arrows. Under the map, two generated messages: England to France, "Mind supporting Edi - Bel?"; and England to Germany, "Do you wanna support my convoy to Bel? With Italy going aggressive France will fall quickly and we can make gains off of both Russia and France."

Figure 3 of the paper shows the same game situation with three different sets of intents and, for each, the messages the dialogue model generates when conditioned on them (p. 1070). The slide shows one panel. The point is mechanical: change the planned actions and the messages change with them. This is how Cicero keeps its talk about a plan the planner actually chose.

6.4 Slide 32: piKL: Best-Respond, but Stay Close to the Humans

Imitation learning (behavioral cloning) predicts human moves well but is brittle and weaker than strong humans; self-play is strong but unlike humans. piKL (Jacob et al. 2022) interpolates. Each player i has an *anchor policy* τ_i learned from human data (no relation to OpenAI Five's team spirit τ) and an anchor strength $\lambda_i > 0$, and is assumed to maximize the regularized utility

$$U_i(\pi_i, \pi_{-i}) = u_i(\pi_i, \pi_{-i}) - \lambda_i \, D_{\mathrm{KL}}(\pi_i \| \tau_i) \tag{3}$$

(Jacob et al. 2022, §4.1, p. 5; FAIR et al. 2022, p. 1071). The case $\lambda_i = 0$ is the plain best response; the closed form below needs $\lambda_i > 0$.

Proposition 1 (the piKL best response)**.** Fix π_{-i} and write $u(a) = u_i(a, \pi_{-i})$ for the expected payoff of action a. Assume $\lambda_i > 0$ and restrict attention to actions with $\tau_i(a) > 0$ (any π_i that puts mass where $\tau_i(a) = 0$ has infinite KL divergence, so such actions never enter the maximizer). The unique maximizer of Eq. (3) over distributions π_i is

$$\pi_i^\star(a) = \frac{\tau_i(a)\, \exp\big(u(a)/\lambda_i\big)}{\sum_b \tau_i(b) \exp\big(u(b)/\lambda_i\big)}.$$

Derivation. The objective is $\sum_a \pi(a)u(a) - \lambda_i \sum_a \pi(a) \log\big(\pi(a)/\tau_i(a)\big)$, which is strictly concave in π on the simplex because negative entropy is strictly convex. Add a multiplier μ for $\sum_a \pi(a) = 1$. Setting the derivative in $\pi(a)$ to zero gives $u(a) - \lambda_i\big(\log(\pi(a)/\tau_i(a)) + 1\big) - \mu = 0$, so $\pi(a) = \tau_i(a) \exp(u(a)/\lambda_i) \cdot e^{-1-\mu/\lambda_i}$; the constant is fixed by normalization. Positivity of every such $\pi(a)$ shows the solution lies in the interior of the face where $\tau_i > 0$, so it is the global maximizer. □

The two limits match the slide. As $\lambda_i \to \infty$, $\pi_i^\star \to \tau_i$: the player is predicted to play like a human. As $\lambda_i \to 0$, $\pi_i^\star$ concentrates on the best responses among actions humans sometimes play. Plugging $\pi_i^\star$ back in gives the optimal value $\lambda_i \log \sum_a \tau_i(a) e^{u(a)/\lambda_i}$, a "soft maximum" of u weighted by the human prior.

From best response to equilibrium: piKL-Hedge. With several players, each player's best response depends on the others. piKL-Hedge runs a regret-minimizing (Hedge) learner for each player on the regularized utilities. Jacob et al. prove that each player accumulates regret of order $\log T$ with respect to these utilities in any game (p. 5; Appendix A, Proposition 1), and that in two-player zero-sum games the average policies converge to a Nash equilibrium of the regularized game (Appendix A, Proposition 2). The Cicero paper summarizes that "piKL provably converges to an equilibrium in the modified utility space" (p. 1071). For seven-player, general-sum Diplomacy the precise guarantee is weaker, and the slide names it: in general games the empirical distribution of the players' joint play under Hedge approaches the set of *coarse correlated equilibria* (CCE) of the regularized game, not a Nash equilibrium (Jacob et al., p. 4). This is a statement about time-averaged play, not about the sequence of policies. In either case the equilibrium is of the *modified* game, in which every player pays a price for straying from human play, so it is human-like by construction. This is the regret-minimization machinery of Module 3 applied to a regularized game. The same KL-to-a-reference penalty is familiar from RLHF and appears in Nash learning from human feedback (M3 agent thread).

How Cicero uses it. Cicero runs piKL to predict every player's policy for the current turn, conditioning the anchor policies on the dialogue. For its own move it chose $\arg\max_{a_i} \; u_i(a_i; \pi_{-i}) + \lambda \log \tau_i(a_i)$ with $\lambda = 3 \times 10^{-3}$, where τ_i is the dialogue-conditional imitation policy. This λ is smaller than the one used for predicting other players, so "the dialogue more strongly informed Cicero's expectations of how other players would coordinate while still allowing Cicero more leeway to deviate" (p. 1072). One observation of ours connects this rule to the proposition: since $\log \pi_i^\star(a) = \log \tau_i(a) + u(a)/\lambda + \text{const}$, the chosen action is exactly the *mode* of the piKL best response with anchor strength λ.

6.5 Slide 33: Talk Tied to Intent: Cheap Talk, and Cicero's Results

Figure description. The left panel of Figure 6 of FAIR et al. (2022), "Example of coordination: Cicero is Austria." Italy asks what Austria is thinking long term and whether Italy should go for Turkey or head west. Austria replies that Turkey went to Armenia, that Italy can go either way, but that if Turkey commits to Russia Italy could always "lepanto," and adds that a Lepanto into Turkey is very strong. Italy agrees and asks for help; Austria says "Of course, happy to do that!" and Italy answers "Fantastic!"

Cicero's link between words and actions was a design decision in two places. In training, the intent attached to each human message was predicted by a model trained on a "truthful" subset of the data, in which the speaker's dialogue was predicted not to be deceptive, because the actions a player actually took might not match an insincere message (p. 1069). In play, Cicero conditioned its dialogue on the action it intended to play that turn. The paper notes the tradeoff: this "maximizes Cicero's honesty and its ability to coordinate but risks leaking information that the recipient could use to exploit it" (p. 1070).

In the vocabulary of Module 7, Diplomacy negotiation is *cheap talk*: messages cost nothing and bind no one, so they carry information only if listeners have reason to believe them. Tying messages to the planner's real intent is a commitment the designers built into the agent; the game does not enforce it, and a different designer could remove it.

Cicero played 40 games of an anonymous online Diplomacy league (webDiplomacy.net) over 72 hours, sending 5,277 messages. It achieved more than double the average score of the human players and ranked in the top 10% of participants who played more than one game (p. 1067).

Bridge to LLM agents. Cicero is a language model driven by a game-theoretic planner. Current LLM agents usually invert the roles: the language model does the planning, and any structured planner is a tool it calls. In that arrangement nothing guarantees that an agent's statements match its actions, so checking that correspondence becomes a monitoring task of the kind studied in Module 8.

7 Part VI: Pluto, Other Recent Results, and Synthesis

7.1 Slide 35: Pluto (2026)

Figure description. The first six columns of the round-robin results table from the official 2026 CoG competition report: bot, games, wins, losses, win percentage, and bots beaten. Pluto is first with 2,844 wins and 40 losses in 2,884 games (98.61%), beating all 9 other bots. PurpleWave (70.66%, 8 bots beaten), Stardust (77.57%, 7), and BananaBrain (73.9%, 6) follow; the last row gives 14,425 games in total. Bots are ranked by bots beaten, which is why PurpleWave is second despite a lower win percentage than Stardust.

What was searched and what was found. The instructor had heard of "something for StarCraft lately called Pluto." A web search found a GitHub repository (`tscmoo/pluto`) hosting binary releases of a StarCraft: Brood War bot, news coverage, and, most importantly, the official 2026 IEEE Conference on Games (CoG) StarCraft AI Competition report by the organizer, David Churchill (Memorial University of Newfoundland). The report links a technical README written by Pluto's author, Vegard Mella. We treat the report and the README as primary sources. Neither is peer-reviewed; the report says the author "intends to write a future publication with more details."

The competition. The 2026 CoG competition ran in August 2026. Ten bots competed, playing 14,425 round-robin games on six maps; the top two by number of opponents beaten met in a best-of-7 final, which Pluto won 4 to 1 over PurpleWave. The organizer describes Pluto as "the first Broodwar bot ever entered into a competition that was made completely by reinforcement learning," submitted as a model-weights file and an executable, running on the standard CPU-only competition hardware (Churchill 2026).

The technique. From the README (sections "Model" and "Training signal"):

- **Network.** One policy network with 315 million parameters, int8-quantized for CPU inference, plays all three races. Unit tokens pass through a 6-layer transformer; the map is a 128×128 build-tile grid encoded by a convolutional network; a 4096-dimensional GRU and a slower transformer memory over a 128-token cache carry state across the game. "Remembering scouted information is left to the network," which is recurrent for that reason.
- **Action interface.** One action every 6 game frames, at most about 4 actions per second, issued through five masked heads: what to do; which unit type, tech, or ability; a coarse 8×8 location; a fine location within it; and a target unit. Fog of war applies as for any bot; there is no camera, so the model sees the whole explored map.
- **Training.** "Trained from scratch by pure self-play reinforcement learning (a PPO-style actor-critic): no replays, no imitation, no scripted behavior, no external bots." "Both sides of every training game are played by the same current weights": the README adds that older checkpoints and other opponents are never used. No search at play time.

- **Rewards.** A dense score equal to the mineral and gas cost of the player's units; a bonus for loosely following a build order sampled from a hand-written set; a small diversity bonus for building rarer units and tech; and ±1 for winning or losing. "These are all applied zero-sum." The model is told the label of the sampled opening but not its contents, and the README cautions that labels may not match behavior.
- **Adaptation in play.** In tournament play Pluto chooses its opening "with a bandit over the win/loss counts" it stores per opponent, so it adapts across a round robin. This is an online learning problem of exactly the Module 3 kind, used on top of a fixed policy.

7.2 Slide 36: Whom Do You Train Against? Three Answers

Table, read as prose. Rows compare AlphaStar, OpenAI Five, and Pluto. Human data: AlphaStar used 971,000 replays before RL; OpenAI Five and Pluto used none. Opponents: AlphaStar's league mixes self-play, PFSP, and exploiters; OpenAI Five plays its latest self 80% of the time and past versions 20%; Pluto plays only its latest self. Diversity comes from exploiters and the strategy statistic z in AlphaStar, from the past-opponent pool in OpenAI Five, and from sampled openings and a diversity bonus in Pluto. Adaptation during play: not reported for AlphaStar or OpenAI Five; a bandit over openings for Pluto. Source: *Nature*, arXiv preprint, and the README with the competition report.

One row of the earlier version moved here: action rates differ (at most 22 non-duplicate actions per 5 seconds for AlphaStar, an action every fourth frame for OpenAI Five, and one action per 6 frames for Pluto).

The puzzle. AlphaStar's ablations found that naive self-play has high Elo "but is more forgetful" (Fig. 3 caption), and the paper's league exists to counter cycling. Pluto reports success with pure self-play in a closely related game. The sources do not explain the difference, so the following are hypotheses for discussion, not findings. (1) Conditioning on a randomly sampled opening plus a diversity bonus may inject the strategic variety that a league otherwise supplies. (2) Brood War at the level of current bots may be less cyclic than StarCraft II at Grandmaster level, or the bot field may simply not have contained the counter-strategies a league would find. (3) The test-time bandit can patch some weaknesses by choosing a different opening against an opponent that exploits one. Testing these would require ablations that have not been published.

The ladder episode. In a public post on September 18, 2026, the organizer reported that "an anonymous person has hacked the bot to play in the new Starcraft: Remastered client and is using it to play against humans on the competitive ladder," and called this "quite disturbing." Press reports added further claims about that copy (for example, that it used a map hack); we could not verify them and draw no conclusions from them. The episode is relevant to Module 8 as an unauthorized deployment of a released agent, not as evidence about Pluto's strength against humans.

7.3 Notes Only: Hidden Information at Scale (Ataraxos and Obscuro)

These two systems had their own slide in an earlier version of the deck; it was cut for time. Two peer-reviewed 2025 to 2026 results return to the imperfect-information methods of Modules 4 to 6.

- **Ataraxos (Stratego).** Sokota et al. (*Nature*, published September 30, 2026; arXiv:2511.07312, November 2025) introduce general techniques for self-play RL and test-time search under hidden information. Stratego has over 10^{33} possible piece configurations. Ataraxos "defeated

the most decorated human Stratego player of all time by a large margin," which the authors describe as, to their knowledge, the first superhuman result in the game, "while consuming orders of magnitude less compute and data than previous efforts." The same techniques gave a superhuman AI for Barrage Stratego and state-of-the-art AIs for Hanabi and dou dizhu (abstract and main text).

- **Obscuro (Fog of War chess).** Zhang & Sandholm (ICLR 2026; arXiv:2506.01242) present the first superhuman AI for Fog of War chess, in which after every move each player observes only the squares onto which her pieces can legally move (rules, p. 13). It introduces search techniques for imperfect-information games that do not rely on common knowledge, and in experiments it was significantly stronger than the prior state-of-the-art AI and than human players, including the world's best (abstract).

Why they belong in this course: Sokota et al. explain that the most successful prior approaches use transformations based on public information whose cost "scales with the amount of hidden information, making them applicable only when this amount is small, as in Texas hold'em (in which there are 1,326 possible hands)." Both papers attack that limit, which is the limit of the subgame-solving methods from M4 to M6.

7.4 Slide 37: Synthesis

Table, read as prose. One table, eight systems, giving the core technique and the course concept behind it. Libratus: CFR blueprint and subgame solving; Nash and CFR (M5, M6). Pluribus: Monte Carlo CFR blueprint and limited-depth search; no equilibrium guarantee beyond 2p0s. AlphaZero: MCTS plus self-play; minimax and policy iteration (M4). AlphaStar: imitation, then a league with PFSP; fictitious play and PSRO (M3, M7). OpenAI Five: PPO, 80/20 self-play, team spirit; team games. KataGo attack: an RL best response to a frozen victim; exploitability. Cicero: piKL and intent-grounded dialogue; equilibrium selection and cheap talk. Pluto: pure self-play with an opening bandit; limits of self-play and bandits.

For the notes-only systems: MuZero searches inside a learned model (abstraction, M6), and Ataraxos and Obscuro push imperfect-information search past the limits of subgame solving (M4 to M6).

7.5 Slide 38: Three Things to Keep

1. Self-play is an equilibrium heuristic with a home turf. In 2p0s perfect-information games it acts like approximate policy iteration toward the minimax value, without a convergence guarantee for deep networks. Cycles, teams, and general-sum payoffs each break a different piece of that argument.
2. Leagues are PSRO with engineering: keep a population, choose opponents with a meta-strategy such as PFSP, add dedicated best-response oracles, and measure the result with a Nash distribution over the population.
3. Strength is not robustness, and equilibrium is not always the goal. A 99% win rate does not bound exploitability (OpenAI Five Arena, KataGo). With many players and language, the target becomes a best response to a realistic model of others (piKL).

7.6 Slide 39: Discussion Questions

1. **Leagues for LLM agents.** A possible mapping: the deployed agent is the main agent; a red team that attacks only the current deployment is a main exploiter; a red team that searches for prompts or tasks that break *every* archived version is a league exploiter. PFSP becomes the rule for allocating red-team effort to the attacks the current agent most often fails, and the meta-game payoff matrix records attack success rates of each attack family against each agent version.
2. **Is 99.4% good?** A protocol closer to exploitability fixes the agent, trains or searches for a best response against it with a budget comparable to a fraction of the agent's own training (as in the KataGo study), and reports that best response's win rate alongside Elo. The cost is a second training run per evaluated version.
3. **Grounded talk.** Requiring an LLM negotiator's messages to match its intended actions leaks its plan, as Cicero's authors note. A monitor could compare stated intentions with subsequent actions and flag divergence, which is an audit problem of the kind in Module 8.

7.7 Slide 40: What Comes Next

Lecture 9.2 covers agent benchmarks (SWE-bench, WebArena, GAIA) and evaluation methodology: cost, variance, contamination, and reproducibility. Today's Arena and KataGo results are the cautionary tale for that lecture: a benchmark score is an average over the test distribution, not a worst case. The syllabus reading is the Cicero paper; for the league, read the Methods section "Multi-agent learning" of Vinyals et al. (2019). For projects, both PFSP (Eq. 1) and the OpenAI Five sampler (Eq. 2) are a few lines of code on top of the PSRO loop from Module 7.

Appendix: Pseudocode for the Two Opponent Samplers

The OpenAI Five sampler is mentioned only in words on Slide 21; its update rule is Eq. (2) in the Part III notes.

> **PFSP sampler** (Vinyals et al. 2019). Input: learner A; pool $\mathcal{C}$; win-rate estimates $\hat{x}_C \approx \Pr[A \text{ beats } C]$; weighting f.
>
> 1. For each $C \in \mathcal{C}$, set $w_C \leftarrow f(\hat{x}_C)$.
> 2. Sample B with probability $w_B / \sum_C w_C$; play a game; update $\hat{x}_B$ with the result.
>
> With $f = f_{\text{hard}}$ and large p, this approaches training against $\arg\min_C \hat{x}_C$.
>
> **Quality-score sampler** (Berner et al. 2019, Appendix N). Input: pool of N past versions with qualities $q_1, \ldots, q_N$; rate $\eta = 0.01$.
>
> 1. Every 10 iterations, add the current agent with $q \leftarrow \max_i q_i$.
> 2. With probability 0.8 play the current agent against itself; otherwise sample past version i with $p_i = e^{q_i} / \sum_j e^{q_j}$.
> 3. If the current agent beats i, set $q_i \leftarrow q_i - \eta/(N p_i)$; otherwise leave q unchanged.

References

- Berner, C., Brockman, G., et al. (2019). "Dota 2 with Large Scale Deep Reinforcement Learning." arXiv:1912.06680. [preprint]

- Brown, N., and Sandholm, T. (2018). "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals." *Science* 359: 418-424. [peer-reviewed]
- Brown, N., and Sandholm, T. (2019). "Superhuman AI for multiplayer poker." *Science* 365: 885-890. [peer-reviewed]
- Churchill, D. (2026). "2026 CoG StarCraft AI Competition Report." https://davechurchill.ca/starcraft/cog/results/2026/report/. [competition report]
- Churchill, D. (2026). LinkedIn post on the StarCraft ladder, September 18, 2026. [organizer's public statement]
- DeepMind (January 24, 2019). "AlphaStar: Mastering the real-time strategy game StarCraft II." Lab blog post; source of the gameplay figure on Slide 12. [lab publication, not peer-reviewed]
- DeepMind (October 30, 2019). "AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning." Lab blog post; source of the figure on Slide 13. [lab publication, not peer-reviewed]
- Czarnecki, W. M., Gidel, G., Tracey, B., Tuyls, K., Omidshafiei, S., Balduzzi, D., and Jaderberg, M. (2020). "Real World Games Look Like Spinning Tops." *NeurIPS 2020*; arXiv:2004.09468. [peer-reviewed]
- Meta Fundamental AI Research Diplomacy Team (FAIR), Bakhtin, A., Brown, N., et al. (2022). "Human-level play in the game of Diplomacy by combining language models with strategic reasoning." *Science* 378: 1067-1074. [peer-reviewed]
- Jacob, A. P., Wu, D. J., Farina, G., Lerer, A., Hu, H., Bakhtin, A., Andreas, J., and Brown, N. (2022). "Modeling Strong and Human-Like Gameplay with KL-Regularized Search." *ICML 2022*, PMLR 162; arXiv:2112.07544. [peer-reviewed]
- Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., et al. (2017). "A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning." *NIPS 2017*; arXiv:1711.00832. [peer-reviewed]
- Mella, V. (2026). "Pluto: StarCraft: Brood War AI, CoG 2026 StarCraft AI Competition entry" (PLUTO_README.txt), linked from the competition report; releases at https://github.com/tscmoo/pluto. [author documentation, not peer-reviewed]
- Schrittwieser, J., Antonoglou, I., et al. (2020). "Mastering Atari, Go, chess and shogi by planning with a learned model." *Nature* 588: 604-609; arXiv:1911.08265. [peer-reviewed]
- Silver, D., Huang, A., et al. (2016). "Mastering the game of Go with deep neural networks and tree search." *Nature* 529: 484-489. [peer-reviewed]
- Silver, D., Schrittwieser, J., et al. (2017). "Mastering the game of Go without human knowledge." *Nature* 550: 354-359; author manuscript used. [peer-reviewed]
- Silver, D., Hubert, T., et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362: 1140-1144; preprint arXiv:1712.01815 used. [peer-reviewed]
- Sokota, S., Vinitsky, E., Hu, H., Fan, Z., Kolter, J. Z., and Farina, G. (2026). "Scalable decision-making for games of imperfect information." *Nature*, published September 30, 2026, doi:10.1038/s41586-026-11036-y; arXiv:2511.07312. [peer-reviewed]
- Vice / Motherboard (April 2019). "'We Will Win:' OpenAI's 'Dota 2' Team Is Crushing Humans Online, But Players Are Not Giving Up." [journalism]
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature* 575: 350-354; author manuscript used. [peer-reviewed]
- Wang, T. T., Gleave, A., Tseng, T., Pelrine, K., et al. (2023). "Adversarial Policies Beat Superhuman Go AIs." *ICML 2023*, PMLR 202; arXiv:2211.00241v4. [peer-reviewed]
- Zhang, B. H., and Sandholm, T. (2026). "General search techniques without common knowledge for imperfect-information games, and application to superhuman Fog of War chess." *ICLR 2026*; arXiv:2506.01242. [peer-reviewed]