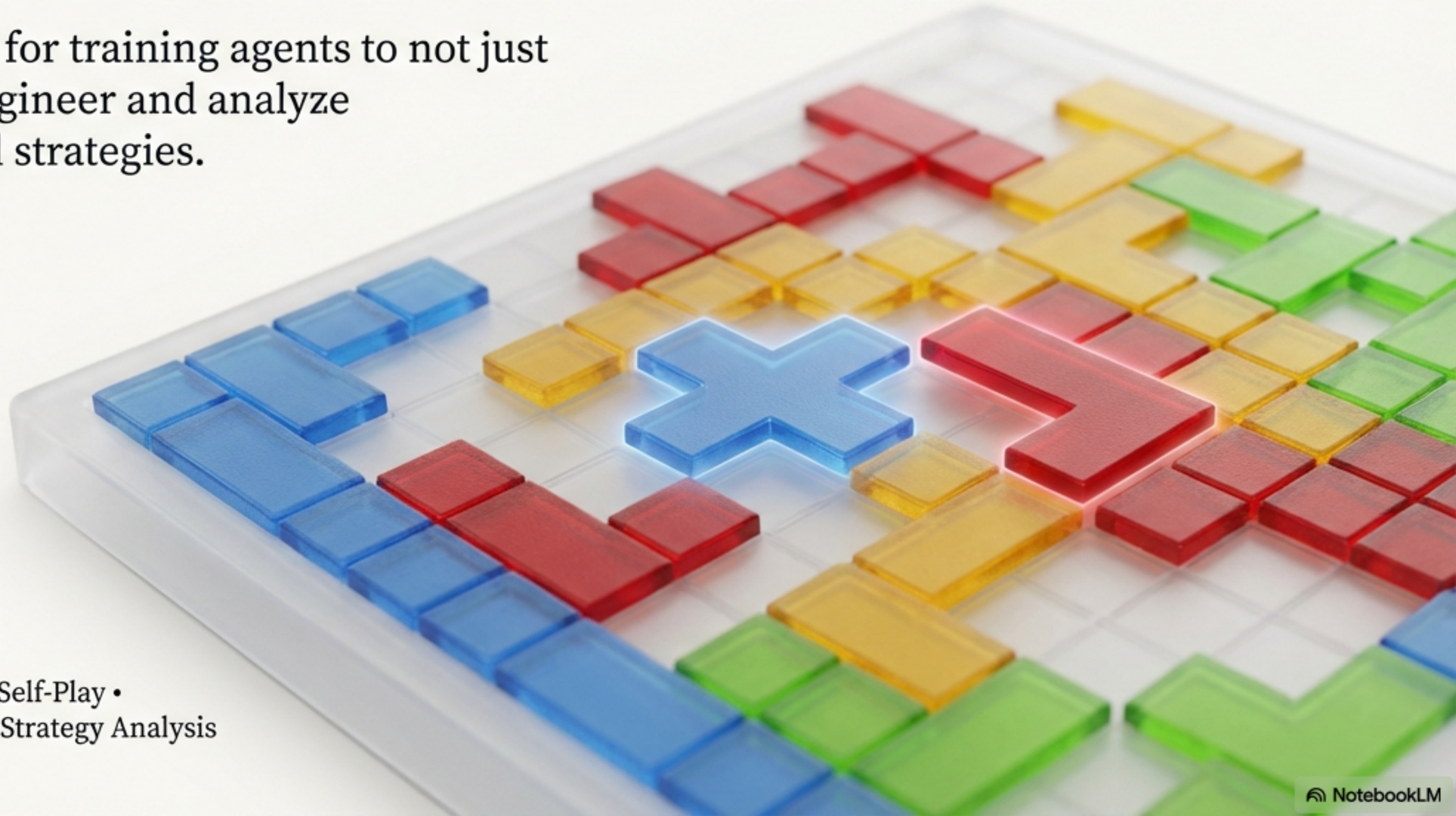
Blokus RL: Teaching an AI to Master Strategic Depth Through Self-Play

An end-to-end system for training agents to not just win, but to reverse-engineer and analyze complex, human-level strategies.



Reinforcement Learning • PPO • Self-Play •
Systems Design • Game Theory • Strategy Analysis

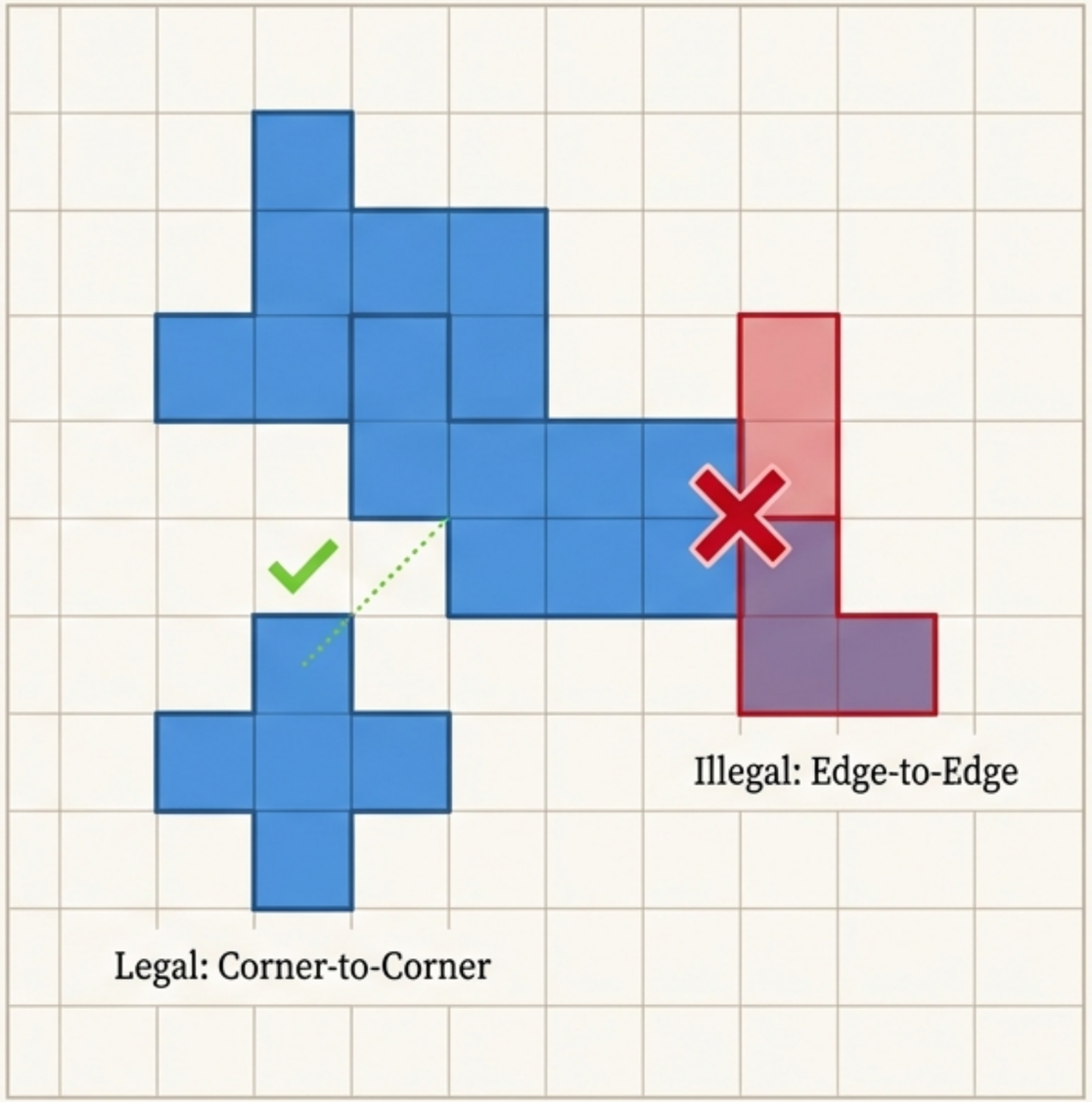
Blokus: A Deceptively Complex Arena for Long-Horizon Planning

Core Content

Objective: 4-player abstract strategy game on a 20x20 grid. Goal is to place the most squares from a personal set of polyomino tiles.

Core Rules:

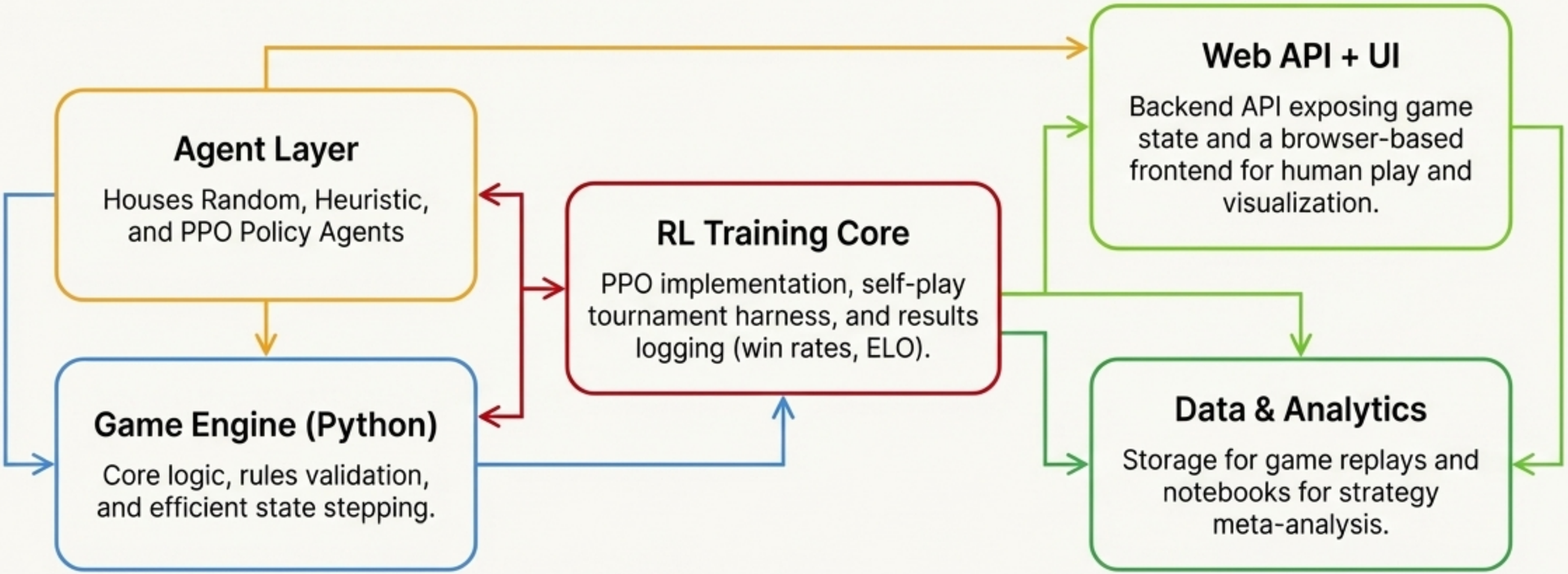
1. New pieces must touch a corner of one of your existing pieces.
2. Pieces of the same color can never touch edge-to-edge.



Why It's a Hard Problem

- **High Branching Factor:** A vast number of legal moves are available on any given turn.
- **Long-Horizon Planning:** Early moves have a compounding impact on the endgame, requiring foresight.
- **Rich Emergent Strategies:** The simple rules lead to complex, conflicting tactics like:
  - Rushing the center vs. fortifying a corner.
  - Aggressive blocking vs. peaceful expansion.
  - Using large, complex pieces early vs. saving them for later.

The Blueprint: An Integrated System for Strategy Discovery



Key Takeaway: The project was designed as a full-stack ‘strategy laboratory’—a complete ecosystem from game simulation to human-facing analysis.

The Foundation: A Correct and Performant Game Engine

Representation

Board: 20x20 integer grid / bitboard.
Pieces: Pre-computed shapes for all rotations and mirrored variants.

Checks: Fast functions for in-bounds, no-overlap, and rule compliance (corner-touch, edge-avoid).

Performance

Move Generation: Optimized routines to scan all remaining pieces and test all legal placements efficiently.

Goal: Engine is fast enough to support the high throughput required for thousands of RL training rollouts.

API for AI

`reset()` -> initial state.

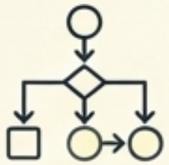
`step(action)` -> apply move, advance player, detect terminal state, and compute rewards.

```python
def step(self, action):
    # Unpack action: piece, orientation, (x, y)
    piece, orientation, position = action

    # 1. Validate the move
    if not is_legal(piece, position, self.board):
        return self.state, PENALTY, False # state, reward, done

    # 2. Apply the move to the board state
    self.board = apply_move(self.board, piece, position)
    self.remaining_pieces[self.current_player].remove(piece)

    # 3. Advance to the next player
    self.current_player = (self.current_player + 1) % 4

    # 4. Check for terminal state
    done = is_game_over(self.board)
    rewards = compute_rewards(self.board) if done else 0

    return self.state, rewards, done
```

A Spectrum of Agents: From Random Baselines to a PPO Master

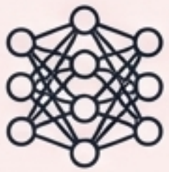
1. Random Agent

**Strategy*: Picks a legal move uniformly at random.
**Purpose*: Sanity-checks the environment and provides a zero-skill baseline.



2. Heuristic Agent

Strategy: Rule-based logic: prefers center moves, using large pieces early, creating future “hooks”.
**Purpose*: A strong baseline and a “teaching opponent” for early-stage RL training.



3. PPO RL Agent

Strategy: A neural policy and value function trained via self-play.
**Purpose*: To learn complex, non-obvious strategies directly from game outcomes without human priors.

4. Specialized Agents (Planned)

**Strategy*: Agents trained with biased reward shaping to encourage specific styles (e.g., “Center-Rush,” “Defensive”).
**Purpose*: Subjects for meta-analysis to understand the strengths of different playstyles.

Why PPO? Taming Instability in a Vast Action Space

The Problem with Vanilla Policy Gradients

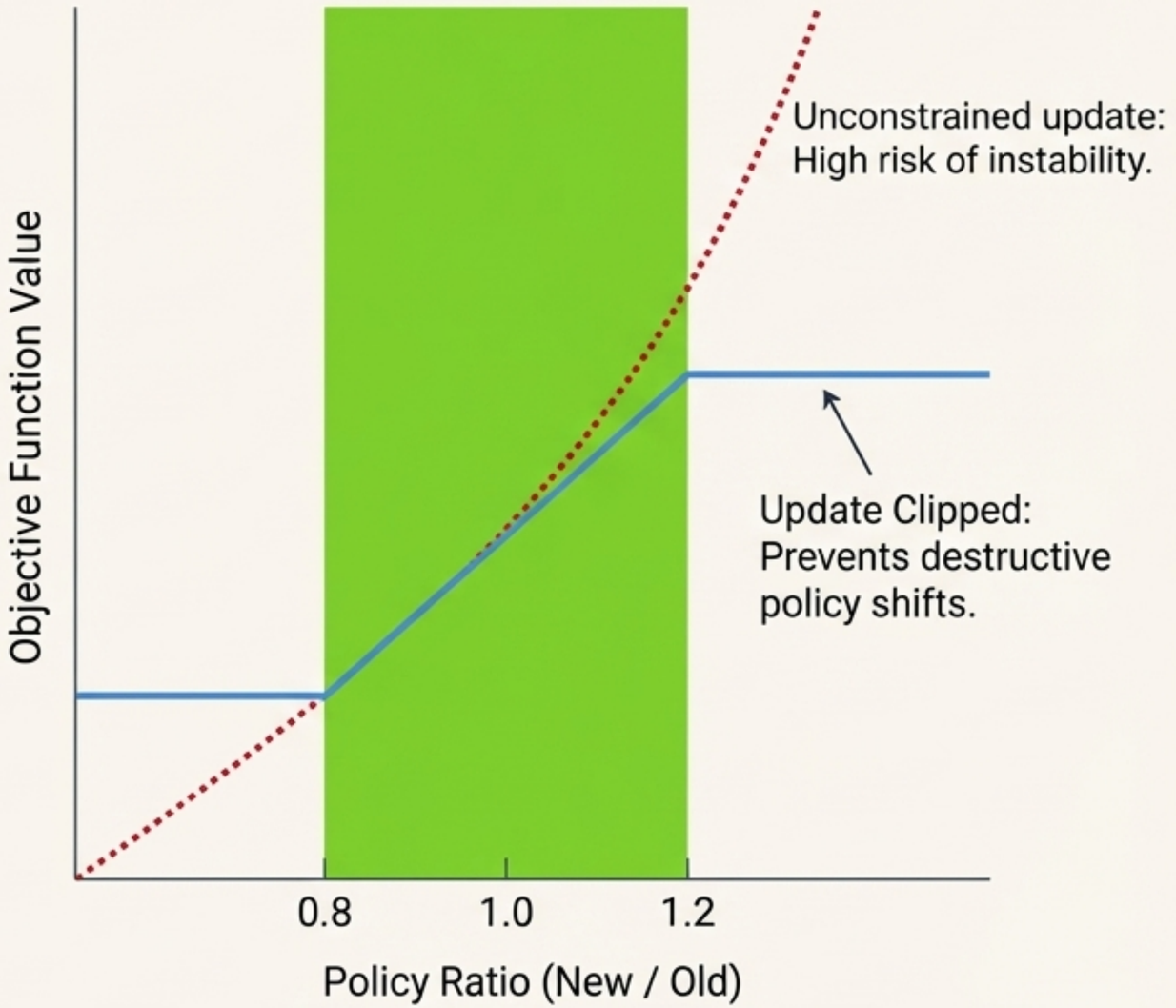
- Large, uncontrolled updates can catastrophically destroy a good policy.
- High variance in gradient estimates leads to inefficient and unstable training.

The PPO Solution: Conservative, Clipped Updates

- **Core Idea:** Update the policy conservatively,
- **Core Idea:** Update the policy conservatively, ensuring the new policy doesn't deviate too far from the old one in a single step.
- **Mechanism:** Uses a surrogate objective with a clipped probability ratio ($r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$). This ratio is clipped within a small range (e.g., [0.8, 1.2]) to prevent excessively large updates.

Benefits for Blokus

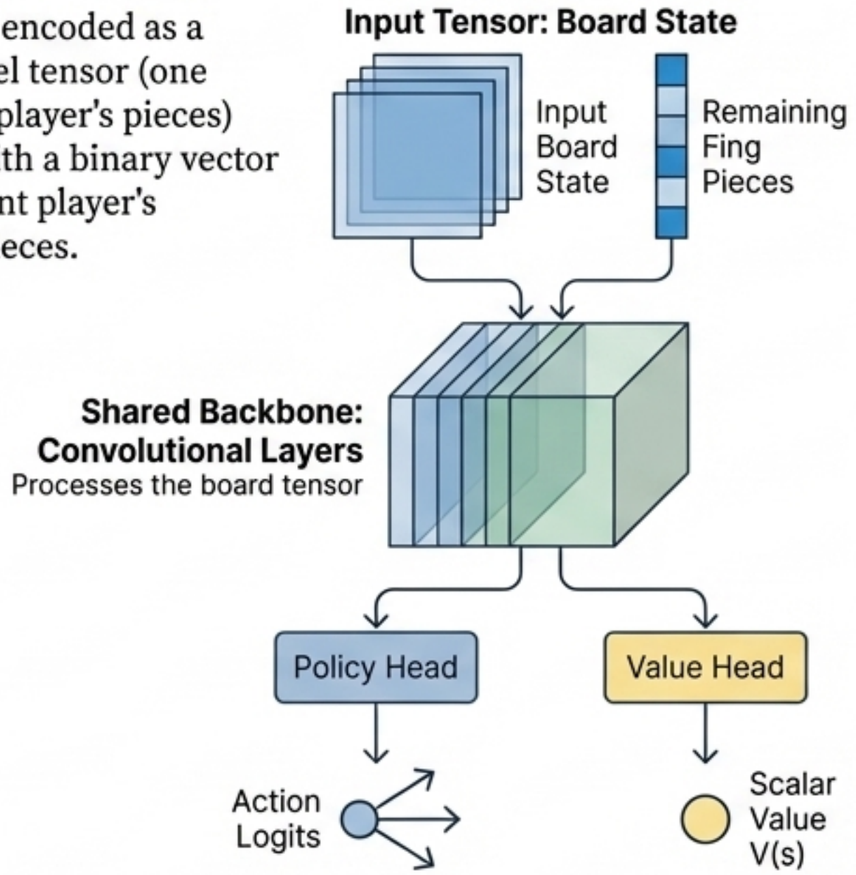
- **Stability:** Crucial for learning over a long self-play process.
- **Performance:** Achieves strong results in environments with large, discrete action spaces like Blokus.
- **Simplicity:** Relatively simple to implement and tune compared to other state-of-the-art methods.



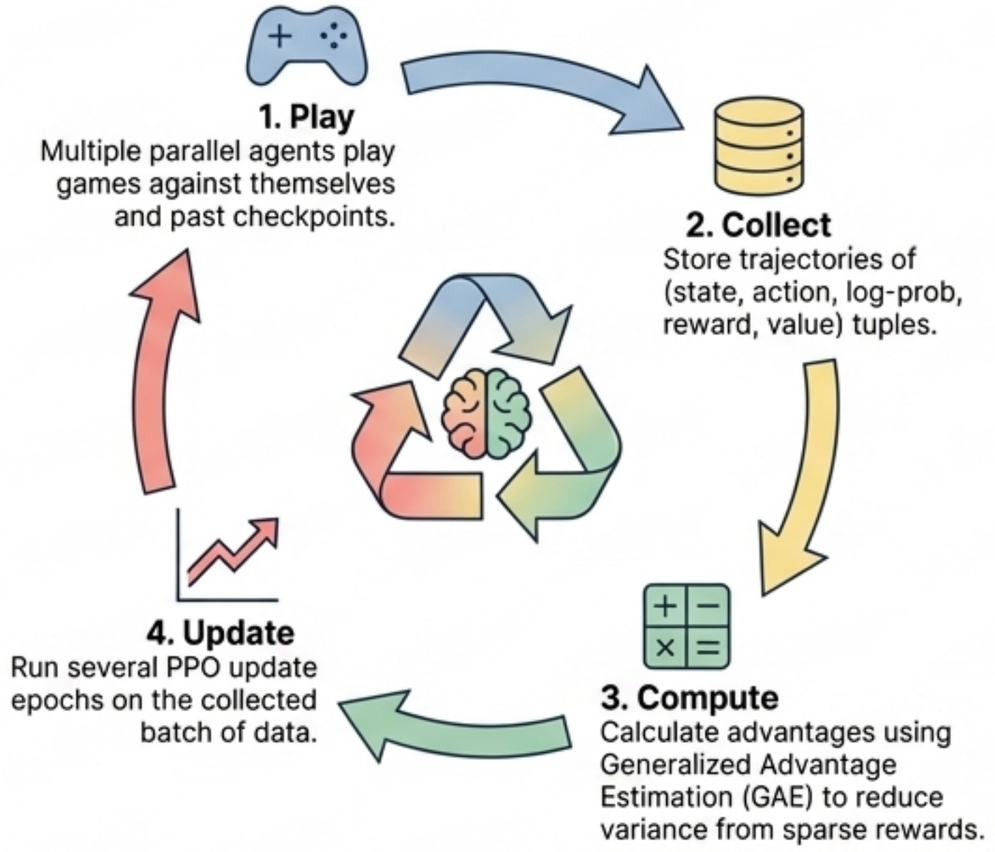
PPO Implementation: Architecture and the Self-Play Loop

State Encoding & Network

The board is encoded as a multi-channel tensor (one channel per player's pieces) combined with a binary vector for the current player's remaining pieces.

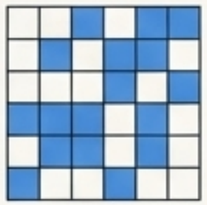


The Self-Play Training Loop



Measuring Strategy: Deconstructing ‘Good’ Play into Quantifiable Metrics

To understand *how* agents win, we need to move beyond win rates and measure the underlying strategic patterns.



Board Coverage

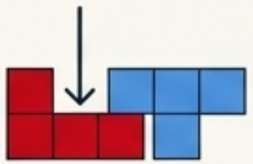
The percentage of the board filled by an agent at game end. A measure of efficiency.

Center Control

Number of squares occupied by an agent in the central 10x10 region by move 10. A measure of spatial dominance.

Piece Usage Profile

The average turn number on which large, complex pieces (e.g., 5-square pieces) are used. A measure of tempo and resource management.



Blocking Frequency

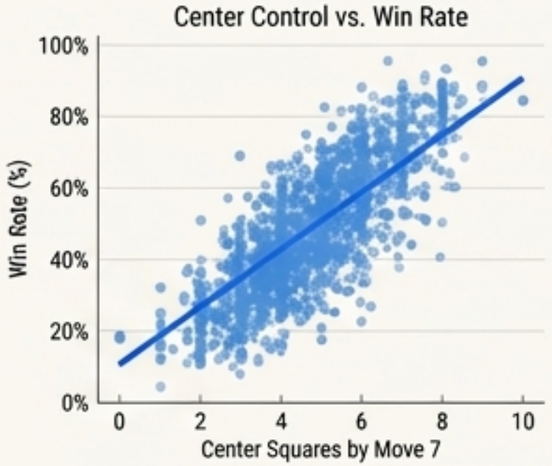
How often a move directly reduces an opponent’s total number of subsequent legal moves. A measure of aggressive interaction.

These metrics allow for correlating specific behaviors with winning outcomes and clustering agents into distinct playstyle "archetypes."

From Data to Discovery: Uncovering Winning Patterns in Blokus

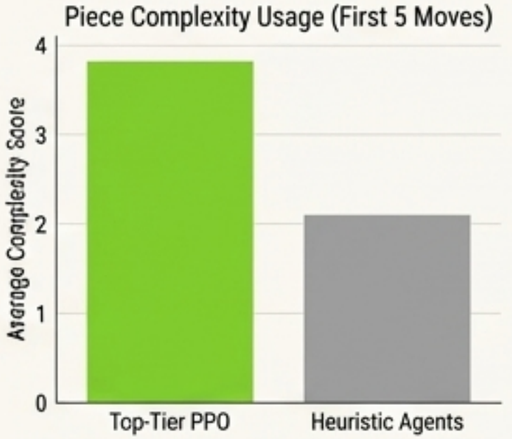
Winning Agents Aggressively Control the Center Early

Finding: Agents that establish a foothold in the central 6x6 region by move 7 win significantly more often.



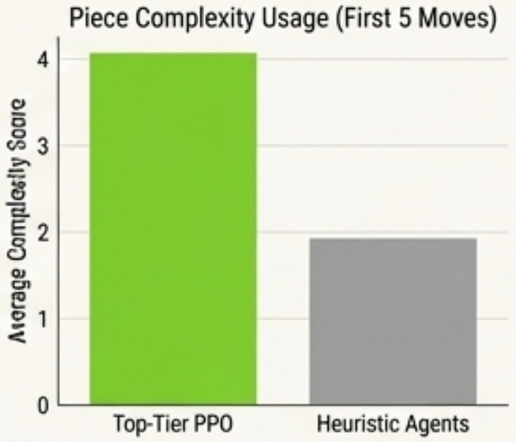
Top Agents Use High-Value Pieces to Open Up the Board

Finding: The most successful agents consistently use their 2–3 largest pieces within their first 5 moves.



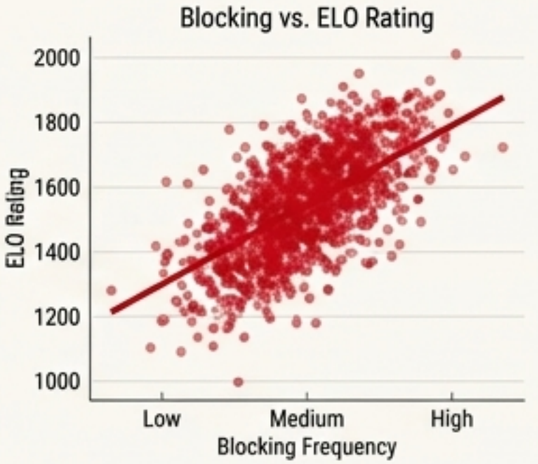
Top Agents Use High-Value Pieces to Open Up the Board

Finding: The most successful agents consistently use their 2-3 largest pieces within their first 5 moves.



Strategic Blocking Outweighs Pure Expansion

Finding: In agent vs. agent tournaments, policies with a higher 'Blocking Frequency' metric consistently achieve a higher ELO rating, even if their final board coverage is slightly lower.

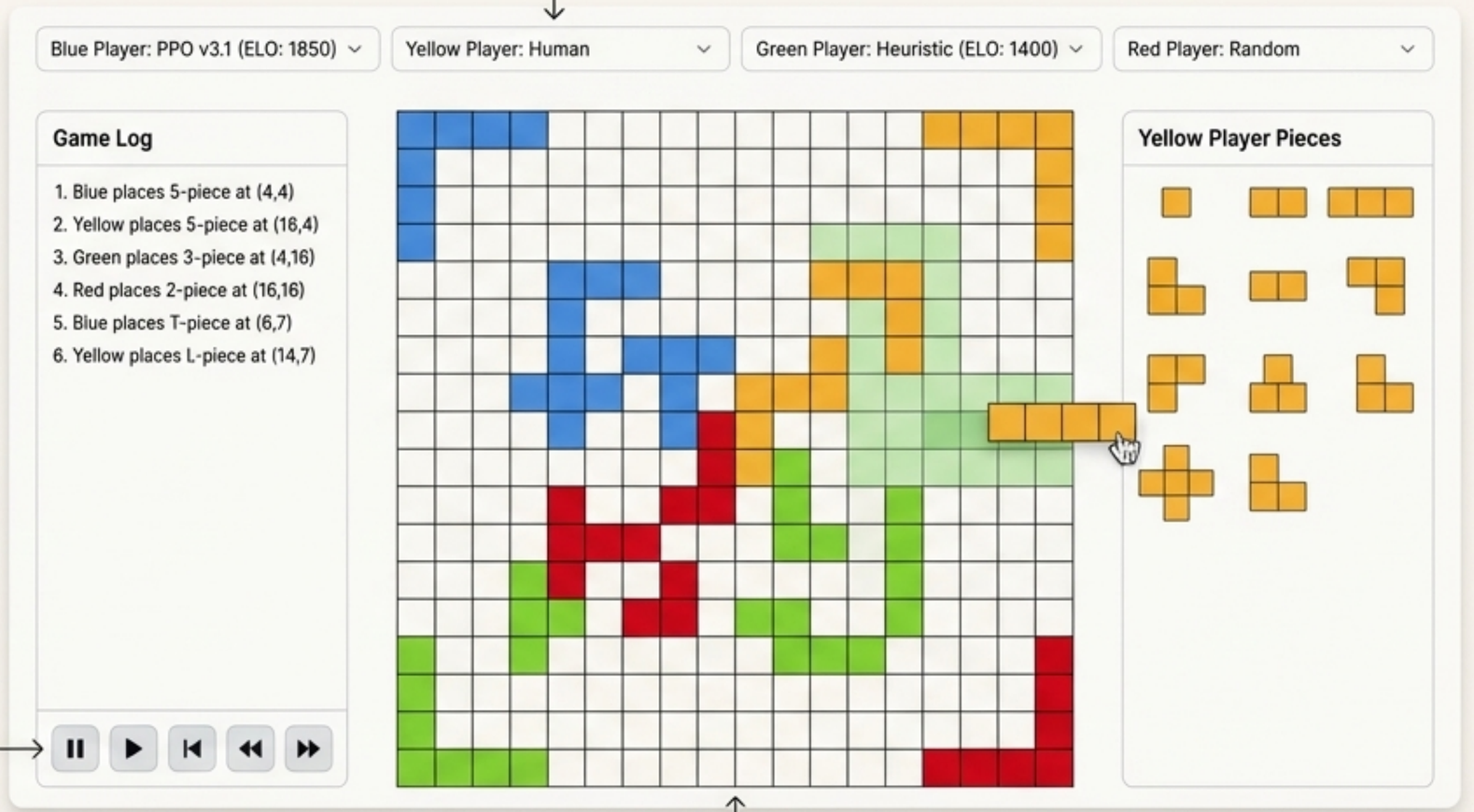


The Human Interface: A Web UI for Visualizing and Challenging the AI

Live Gameplay: Play as a human against any of the developed agents (Random, Heuristic, PPO).

Strategic Purpose

- Provides a direct, intuitive way to experience the difference in agent skill and style.
- Serves as a powerful debugging and analysis tool.
- Demonstrates the ability to build a complete, user-facing application around an ML system.



Game Replay & Analysis: Step through completed games move-by-move to observe an agent's strategy unfold.

Visual Move Validation: The UI provides real-time feedback, highlighting legal and illegal placements.

Technical Challenges and Engineered Solutions

Challenge	Solution
Huge, Variable Action Space	Implemented efficient action encoding and masking within the policy network to handle valid moves only.
Sparse & Delayed Rewards	Used a learned Value Function and Generalized Advantage Estimation (GAE) to propagate credit back from the final game score.
Multi-Player Credit Assignment	Designed a relative reward scheme (+1 for win, -1 for last place) to handle the 4-player dynamics.
High-Throughput for Self-Play	Optimized the game engine with pre-computed transforms and parallelized environment execution during training.

Core Competencies Demonstrated

Reinforcement Learning

- Proximal Policy Optimization (PPO)
- Self-Play Training Paradigms
- Reward Shaping & Advantage Estimation (GAE)
- Value Function & Policy Network Design

Systems Design & Backend

- High-Performance Game Engine Architecture
- Python Backend Optimization
- Designing APIs for ML Systems

Experimentation & Analytics

- Building robust logging and evaluation pipelines.
- Defining meaningful metrics for strategy analysis.
- Deriving actionable insights from experimental data.

Full-Stack Thinking

- Connecting a complex backend ML system to a human-facing web interface for introspection and interaction.

Roadmap: Evolving from a Project into a Research Platform

Short-Term (Next Steps)

- Build visual dashboards for real-time strategy metrics (center control, piece usage).
- Train more specialized agents via reward shaping to create a richer tournament ecosystem.

Medium-Term (New Features)

- Develop an interactive “Coach Mode” where the AI suggests moves to a human player and explains its reasoning.
- Implement game replay tools with annotation overlays to highlight key strategic moments.

Long-Term (Vision)

- Establish Blokus RL as an open research playground for interpretable RL, automated strategy discovery, and human-AI collaborative learning.