MonoManiac - An Advanced Monopoly Agent for Social Reasoning Benchmark

Federico Baldan [*1] **Larry Liu** [*1] **Runlin He** [*1]

Abstract

Large Language Models (LLMs) have demonstrated impressive capabilities in different areas, such as math, coding, and writing. However, traditional benchmarks fail to capture the combination of probabilistic decision-making, negotiation, and social reasoning needed to recreate real-world scenarios. For instance, this limitation can be observed when playing a game of chess with the most recent LLM agents: after a few moves, the agent will start moving pieces off the board. Utilizing the game of Monopoly as a mix of planning, negotiation, and social reasoning, we created MonoManiac, an advanced Monopoly agent designed to address these issues and improve the reasoning architecture. Through a simulated Monopoly environment, we developed a Strategic and Social Reasoning Benchmark, allowing us to test current state-of-the-art models and assess key points of failure in social reasoning environments. Across several simulated games with different LLMs, our approach outperforms state-of-the-art LLM agents and highlights improvements in distinctive strategic reasoning, reasoning speed, and evaluation context.

1. Introduction

In recent years, AI has achieved remarkable progress across different domains and has become increasingly integrated into everyday life at an unprecedented scale. Large Language Models(LLMs), in particular, represent in many aspects the future of human–machine interaction, positioning them at the frontline of cutting edge Ai research. Despite unthinkable improvements, the main question remains crystal clear: how can we evolve technology in order to make systems better compatible with real-life human needs and requirements? While reinforcement learning methods have demonstrated impressive results in terms of reasoning and decision-making processes, multi agent negotiation tasks represent a particularly tough obstacle to overcome. When considering real-world professional negotiation scenario, success is not only determined by a clear understanding of the goal, but also, and mainly, by perceiving intentions, strategies, ideologies, and peculiarities of each "agent". Basically, planning a strategy accordingly to the evolving context. Moral of the story: adding bluffing or tricks makes this incredibly hard even for state-of-the-art models.

MonoManiac, an agent for social reasoning benchmarking, aims to investigate the dynamics and evaluation errors of LLMs in these environments, using the board game Monopoly as a case study.

To illustrate the challenge, it is useful to consider the question: "Why do LLMs perform so poorly at chess, for example?" Someone might expect that for an advanced model capable of solving complex mathematical and coding problems, it would be like taking candy from a baby. Surprisingly enough, after a few rounds, we can easily find ourselves playing with pieces outside the chessboard, tearing down our Elo rating. The reasons are simple: first, LLMs lack of robust systematic generalization, performing purely in state understanding and tracking; second, they are optimized for model-based planning, aiming more to next token prediction or to generate realistic moves rather than aiming to win. Similarly, in the game of Monopoly, multiple players are allowed to cooperate with each other for multiple rounds, although the ultimate goal is to win over the opponent. Consequently, this creates the perfect environment to asses LLMs' behaviors in multi-agent social strategic scenarios. Our methodology will be based on first evaluating, through a series of Strategic and Social Reasoning Benchmarks, the performance of the LLMs involved in the simulations. By understanding the current deficiencies, we then implement MonoManiac, an agent specifically designed to address these gaps. Through this work, we highlight how targeted simple changes to decision-making principles can yield to a completely different performance in socal reasoning tasks. Our full implementation, including the simulator, agent framework, and evaluation pipeline, is publicly available at: https://github.com/Lars1505/monopoly.

[1]Department of Computer Science, University of Washington, Seattle, WA. Correspondence to: Federico Baldan <federico@uw.edu>, Larry Liu <larryl23@uw.edu>, Runlin He <rh74@gmail.com>.

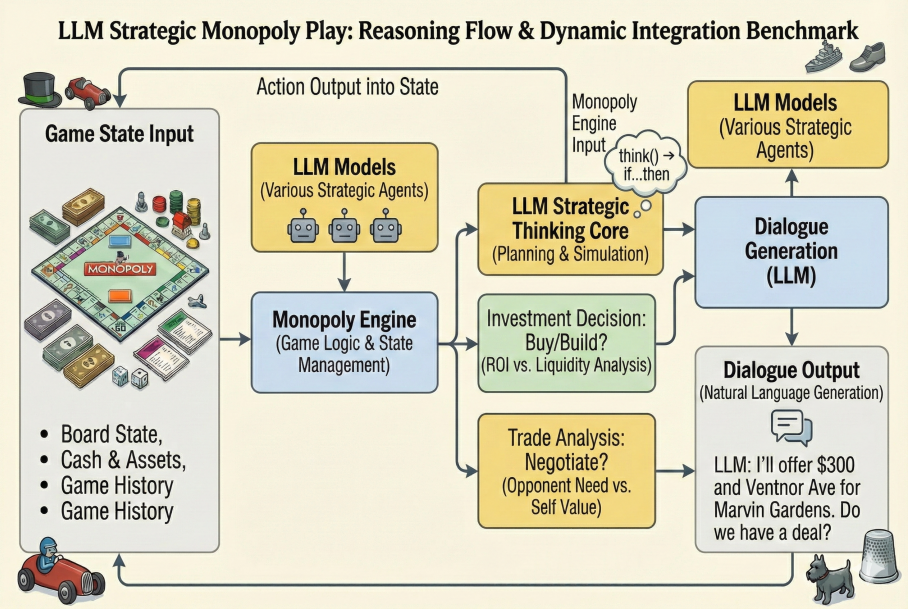

Figure 1. Strategic reasoning pipeline for Monopoly using LLMs.

1.1. The game of Monopoly

Monopoly is a multi-player strategy board game typically involving four or more participants. All players begin at the same starting position and roll two dice each turn to advance around the board, composed by a combination of purchasable properties and other mixed event spaces(e.g. going to jail, pay a tax, chance cards) . When landing on a property, if not already owned by a player, someone can buy it through the money of the starting capital. Properties owned by other players require rent payment upon landing. A key game mechanic is the ability to develop properties: when a player owns all properties in a color group (a monopoly), they can construct houses and hotels to substantially increase the rent collected from other players. To give an idea of the importance of this dynamics, certain properties can increase ten times the initial rent fee if a hotel is built upon it. Each turn, players may freely communicate with each other in order to negotiate trades, crate all sorts of strategic alliances, with the ultimate objective of acquiring property monopolies to maximize revenue generation. Tradable assets include owned properties, cash, and special cards (e.g., Get Out of Jail Free), and players may also negotiate future commitments or conditional agreements to secure strategically valuable properties. Player behavior might vary in strategy risk tolerance: some adopt aggressive strategies involving early trades and alliances, while others maintain a more conservative approach to resource allocation. Victory is achieved either by eliminating all opponents (bankruptcy) or, in timed or rounds limited games, by accumulating the highest total net worth. This is calculated as the sum of cash, property values, and development investments. Official rules are taken from (Hasbro, Inc., 2008).

1.2. Key Challenges in the Game Environment

Prior work has demonstrated that two-player zero-sum games can be mastered by AI agents through reinforcement learning algorithms (Silver et al., 2018). Monopoly does not involve particular challenges in terms of state evolution planning and advancement: unlike deterministic strategy games, Monopoly introduces stochastic elements through dice rolls that determine board advancement. However, similar to the game of Diplomacy(, FAIR), a substantial portion of the success lies on natural language negotiation and communication. Despite LLMs can process straightforward decision-making scenarios, they struggle heavily with behaviors involving strategic deceptions, bluffing, or manipulating opponents toward specific outcomes. A second critical challenge involves trade evaluation. Since trades are fundamental to Monopoly strategy, players must evaluate whether proposed exchanges provide net benefit without excessively strengthening opponents. Differently from humans, LLMs don't have references to evaluating trade value relative to long-term strategic objectives. Consequently, they won't be able to recognize subjective benefits of the trade based on strategy that they want to purse, such as selling a property that will allow another player have an entire set. A related challenge is risk calibration, which indicates the ability of a model to determine when aggressive resource allocation is strategically justified. For instance, early-game strategy typically favors acquiring any available property regardless the strategy, as the evolving board state creates numerous potential paths to monopoly forma-

tion. Moreover, optimal early-game play often requires aggressive property acquisition that lead players to remain with a small amount of cash. However, LLMs tend toward overly conservative financial management, failing to recognize this strategic principle and refuse important early stage fundamental transactions in favor of maintaining what they perceive as safe capital reserves.

2. Related Work

2.1. Reinforcement Learning for LLM Reasoning

Reinforcement Learning (RL) in Large Language Models has evolved beyond alignment tasks via RLHF (Jaques et al., 2019) to directly enhance reasoning and mathematical capabilities. Recent advancements, such as DeepSeek-V3.2 (DeepSeek-AI et al., 2025) and the Gemini family (Team et al., 2025), demonstrate that self-supervised learning and RL with Verifiable Rewards (RLVR) can effectively unlock chain-of-thought reasoning using rule-based signals. However, these approaches often rely heavily on the model's ability to maintain long contexts and extensive memory history. MonoManiac addresses this dependency by integrating a natural language-based engine with a board simulator, enabling the autonomous generation of reasoning and negotiation challenges without human supervision.

2.2. LLMs in Multi-player Game

Games serve as critical testbeds for evaluating LLM capabilities (Xie et al., 2025). While recent research has utilized multi-player cooperative games such as Overcooked to study human-AI collaboration (Sarkar et al., 2023), and frameworks like ViGal (Xie et al., 2025) have shown that arcade games can enhance mathematical reasoning, these approaches often focus on singular modalities. In contrast, MonoManiac integrates these aspects through a multi-agent self-play framework. Our approach enables LLMs and heuristic agents to share a board environment and negotiation history, utilizing fully self-supervised learning to master diverse scenarios. Crucially, MonoManiac demonstrates the transfer of capabilities from social gaming to rigorous reasoning and calculation tasks, achieved without access to future states during training.

2.3. Strategic Negotiation in Multi-Agent Systems

Beyond individual reasoning, effective deployment of LLMs requires the ability to navigate complex social dynamics and strategic negotiation. Prior research in multi-agent systems has explored negotiation primarily through game-theoretic models or structured communication protocols, often lacking the nuance of natural language. While recent works like Cicero (, FAIR) have demonstrated success in combining language with strategic planning in games like *Diplomacy*, they typically rely on massive datasets of human game play logs.

3. Engine Simulator

We developed a high-fidelity, text-based Monopoly simulator in Python that serves as the evaluation environment for both heuristic rule-based agents and LLM-driven agents. The simulator allows for the direct comparison of rigid, algorithmic decision-making against the flexible, natural-language reasoning of state-of-the-art language models. The engine manages the full game state, enforces the official rules, and handles the communication protocol between agents.

3.1. Game Basics and Architecture

The simulation environment is built upon a modular architecture consisting of the following core components:

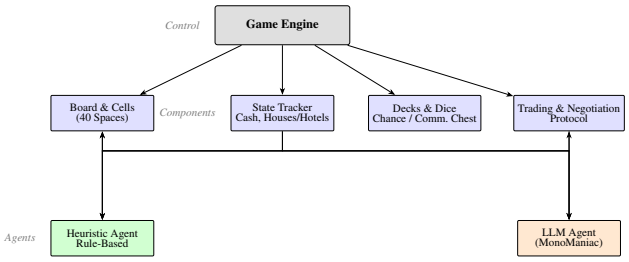

Figure 2. Modular architecture of the Monopoly simulator. The game engine manages core components (board and cells, global state and finite resources, stochastic decks and dice, and the trading & negotiation protocol) that interface with both heuristic rule-based agents and the LLM-based MonoManiac agent. Bidirectional arrows indicate interactive communication channels.

Decks and Probability: The *Chance* and *Community Chest* decks are implemented as shufflable lists of action strings. When a player lands on a draw cell, an effect is triggered immediately, ranging from simple position updates (e.g., "Advance to Go") to complex state changes (e.g., "Get Out of Jail Free" ownership). The `Dice` class handles stochastic movement, tracking the "doubles" count to enforce the rule sending players to jail after three consecutive doubles.

State Tracking: The global game state tracks resource scarcity (finite count of 32 houses and 12 hotels). Each `Player` instance tracks individual metrics: cash balance, current board position, list of owned `Property` objects, and jail status (including days spent and possession of pardon cards).

3.2. Decision Making Architectures

The engine supports two distinct agent types that interact with the same board API but utilize fundamentally different reasoning backends.

3.2.1. Heuristic Rule-Based Agents

The baseline agents use deterministic logic defined in the `Player` class, designed to mimic rational, risk-averse human play.

Buying Strategy: The agent decides to purchase a property based on a liquidity check. It proceeds only if:

$$C_{prop} \leq M_{current} \quad \text{and} \quad (M_{current} - C_{prop}) \geq T_{safety} \tag{1}$$

where C_{prop} is the property cost, $M_{current}$ is the player's cash, and T_{safety} is a configurable "unspendable cash" buffer.

Improvement Logic: The `improve_properties()` method iteratively builds houses using a "cheapest-first" heuristic. Improvements occur only when:

1. A full color monopoly is owned.
2. No properties in the group are mortgaged.
3. The "even build" rule is satisfied (difference in house counts ≤ 1).

Crisis Management: When a player cannot meet a financial obligation (rent or tax), the `raise_money()` method triggers a specific liquidation hierarchy: selling hotels back to houses, selling houses to cash, and finally mortgaging properties sorted from highest to lowest mortgage value.

3.2.2. LLM Agents (MonoManiac)

The `LLMPlayer` subclass replaces heuristic checks with natural language prompts. The agent's decision-making is driven by a text-based context window that is dynamically constructed at every decision point.

Context Construction: The `_build_full_context()` method aggregates the game state into a structured prompt:

- **Self State:** Net worth, cash, current position, and property portfolio.
- **Global State:** A compressed representation of the board, grouping properties by color and explicitly noting ownership (e.g., `Green: Pacific:Bob|NC:Charly`).
- **Action Space:** A strictly formatted set of valid options.

Strategic Batching: To optimize token usage and reasoning coherence, the agent utilizes a "Batched Strategy" prompt every 3 turns. This prompts the model to plan high-level moves, such as trading and improvements, simultaneously:

```
TURN STRATEGY - Provide answers for ALL
    sections:
A) TRADE PROPOSAL (Available: LLM1)
   Respond with: TRADE_PROPOSE:<target>:<
       give>:<receive>:<cash>
B) PROPERTY IMPROVEMENTS
   Respond with: IMPROVE:<prop1>,...
```

3.3. Trading and Negotiation Protocol

The trading engine is the most complex component, facilitating asset exchange.

3.3.1. Heuristic Trading

Rule-based agents use set-theoretic logic to identify "mutually beneficial" trades. An agent signals a desire to sell a property if it is the sole owner in a color group dominated by another player. Conversely, it seeks to buy properties that complete its own potential monopolies. A trade is executed only if the `get_price_difference()` method determines the value exchange is within a relative tolerance threshold (e.g., $\pm 20\%$).

3.3.2. LLM Natural Language Negotiation

LLM agents engage in a multi-turn, structured negotiation protocol managed by the `ActionParser`. The negotiation flow follows a *Propose* $\rightarrow$ *Counter* $\rightarrow$ *Resolution* cycle, allowing agents to haggle over value.

Proposal Structure: Agents generate structured strings such as `TRADE_PROPOSE:LLM1:Park Place:Boardwalk:200`. This indicates the proposer gives Park Place and $200 to receive Boardwalk.

Counter-Offers: The recipient can accept, reject, or counter. The following excerpt from our simulation logs (`game_1_LLM2_chat_history.txt`) demonstrates an agent recognizing a strategic opportunity and initiating a complex trade to acquire a monopoly piece:

```
USER: TURN STRATEGY...
A) TRADE PROPOSAL
   Your properties: C1 St. Charles, D2
       Tennessee, F2 Ventnor...

ASSISTANT: A) TRADE_PROPOSE:LLM1:R4 Short
    Line:F2 Ventnor:50
```

In this instance, the agent (LLM2) proactively offers a railroad (`R4`) plus cash to acquire `F2 Ventnor Avenue`, demonstrating an understanding of asset utility beyond simple face value. The simulator parses these intent strings,

validates ownership and funds, and executes the transaction atomically if accepted.

4. Results

Our experiments revealed significant behavioral difference between the Large Language Model (LLM) agents (Llema3.2(Touvron et al., 2023), gpt-oss (OpenAI et al., 2025)) and the rule-based baseline agents (Bob, Charly). Previous work (Laban et al., 2025) have demonstrated the current LLMs does not have the capability to manage the memory optimization in such long turns game and may lose mathematical logic during the game. The results highlight specific strengths in negotiation fluency but critical failures in long-term strategic planning and asset valuation.

4.1. Differences in Model Behaviors

The most distinct difference observed was the strategy of the agents. While LLM agents demonstrated superior capability in initiating and negotiating complex trades, they consistently failed to account for cash flow velocity, whereas rule-based agents prioritized rapid development.

4.1.1. Trader vs. Builder

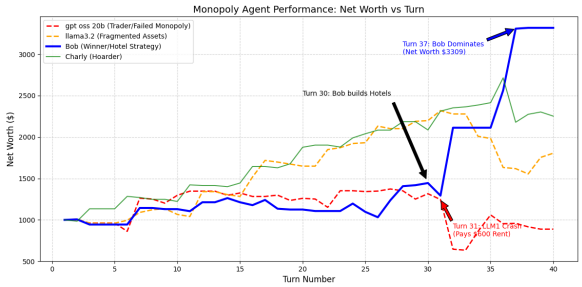

Figure 3. Monopoly net property vs. turns for gpt-oss 20b model and llama 3.2 with rule based models

The LLM agents focus being as "Traders," focusing on asset accumulation and portfolio shuffling (monopoly). In Game 1, LLM1 successfully negotiated a monopoly on the high-value Dark Blue set (Park Place and Boardwalk) by Turn 33. However, it failed to maintain sufficient liquidity to develop these properties. In contrast, the rule-based agent (Bob) acted as a "Builder." Bob secured a lower-value Light Blue monopoly and immediately began housing development in Turn 13, achieving Hotel status by Turn 30. The LLM's failure to calculate future rent liabilities led to its insolvency in Turn 31, directly caused by rent payments to the rule-based agent

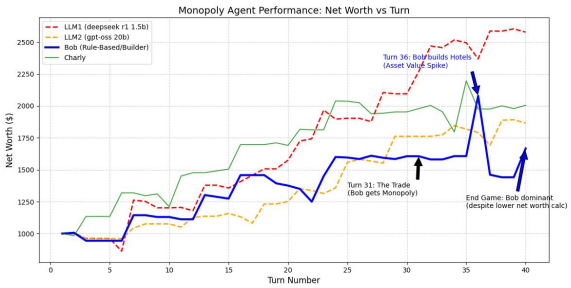

Figure 4. Monopoly net property vs. turns for gpt-oss 20b model and deepseek r1 1.5b model with rule based models

4.1.2. Strategic Blindness

The LLM agents did not disturb the opponents monopoly. In Game 2, Turn 31, LLM1 acquired *Vermont Avenue*, the final piece required for Bob's monopoly. Rather than holding this property to block Bob's victory, LLM1 immediately traded it to Bob in exchange for a low-value Utility (*Waterworks*) and $50. This trade effectively handed the game to the rule-based agent, who built hotels five turns later, demonstrating that while LLMs understand the mechanics of a trade, they struggle to evaluate the consequences of a trade on the game state.

4.1.3. Break and Repair

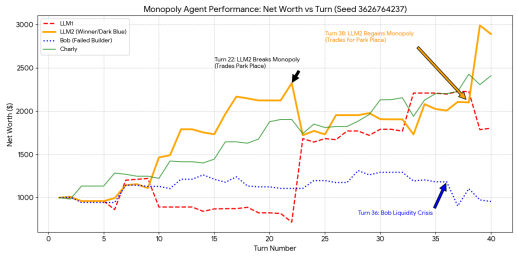

Figure 5. The LLM breaks the monopoly and regain the monopoly

LLM2 initially secured a natural monopoly 5 on the Dark Blue set by Turn 20, acquiring Boardwalk to complement its earlier purchase of Park Place. In a standard strategic framework, this position would trigger immediate housing development. However, LLM2 exhibited a critical failure in long-term planning by voluntarily breaking this monopoly just two turns later. In Turn 22, LLM2 traded Park Place and $100 to LLM1 in exchange for Baltic Avenue, a low-value asset. This decision effectively neutralized LLM2's competitive advantage, suggesting that the model prioritized immediate transaction completion or perceived "fairness" over the retention of a win condition.

4.2. Recursive Trading Patterns

An interesting phenomenon observed exclusively in LLM-to-LLM interactions was "Recursive Trading," where specific assets were exchanged back and forth multiple times with no clear strategic progression. In Game 1 (3), the Utility properties (*Electric Company* and *Waterworks*) entered a high-velocity circulation loop:

- **Turn 15:** LLM1 traded both Utilities plus cash to LLM2 to acquire properties.
- **Turn 16:** Immediately following the previous trade, LLM1 traded high-value assets (Short Line and New York Avenue) to re-acquire the same Utilities.
- **Turn 24:** LLM2 initiated a trade giving up colored properties (Ventnor and Tennessee) to acquire the Utilities back once again.

Also in the other game, we also find this recursive and unreasonable trading.

- **Turn 8:** (The First Swap): LLM2 traded away the high-value H1 Park Place and D2 Tennessee to LLM1 in exchange for the low-value U2 Waterworks.
- **Turn 11:** (The Reversal): Three turns later, LLM2 traded U2 Waterworks back to LLM1 to regain H1 Park Place.
- **Turn 14:** (The Consolidation): LLM2 traded H1 Park Place away again to LLM1 in exchange for U1 Electric Company and U2 Waterworks.

This circular behavior suggests that the LLMs may prioritize deal itself as a local reward, treating assets as highly liquid currency rather than long-term rent generators, which prevented the LLMs from establishing the stable capital base required to win.

4.3. Agent Errors Pattern Analysis

Through manual inspection of reasoning traces across multiple LLM agents, we identified several recurring behavioral patterns that consistently lead to suboptimal decisions and eventual loss bu the LLM. These errors are not simple state-evaluation mistakes, as they often highlight deeper strategic blind spots in how models align actions with long-term objectives. Our analysis is based on a human assessment of chain-of-thought reasoning and decision patterns observed during the game play. To conduct that we read all the written thinking processes of each turn for every evaluation and understand the criticalities.

The most significant failure mode involves a **misunderstanding of the strategic importance of completing and preserving property monopolies**. Securing a full color group enables house and hotel construction, which represents the primary mechanism for generating exponential rent and winning the game in the majority of the cases. LLMs repeatedly misjudge this priority across two key phases. During the acquisition phase, agents fail to prioritize properties that would complete their own sets, often focusing instead on nominal property value or maintaining too conservative cash portfolios. Even more critically, during the trading phase, agents frequently trade away properties that would complete an opponent's monopoly, even though having full information to infer the strategic consequence. In contrast, the rule-based agents (Bob and Charly) are already programmed to pursue and protect color group completion, enabling them to consistently capitalize on these LLM oversights. Interestingly, LLMs don't recognize almost never this patter.

Other that property set management, LLM agents exhibit a **limited ability to leverage not trivial, rule dependent strategies**. The main example is the "jail strategy", a well known mid/late game tactic where players intentionally remain in jail to minimize exposure to opponent rent while still collecting income from their own properties. The complete absence of this behavior in LLM decision making highlights an important limitation, which is the difficulty in recognizing counterintuitive or emergent advantages that come form from rule analysis and elaboration.

A third recurring pattern involves the **systematic overvaluation of high-cost properties**. LLM agents tend to prioritize acquiring expensive tiles (e.g., the Dark Blue or Green groups), implicitly conflating price with strategic value. Our results show, however, that and effective early-game play relies on acquiring multiple cheaper properties that can be monopolized and developed quickly.

5. MonoManiac Agent

5.1. Reinforcement Learning Self-Play Architecture

We extend the MonoManiac framework to enable multi-agent reinforcement learning through self-play, integrating the Monopoly engine with Tinker's distributed RL training infrastructure. The architecture maintains the existing game engine unchanged while replacing external LLM API calls with a learnable policy that improves through experience.

5.1.1. Coordinator Pattern and Threading Model

The integration employs a `DispatchingMonopolyCoordinator` that manages the synchronous game loop in a separate thread while synchronizing with Tinker's asynchronous RL environment via queue-based communication. Each coordinator

instance encapsulates a single 4-player game, with all players controlled by the same policy (shared weights), enabling true self-play where the agent competes against progressively stronger versions of itself.

Thread Architecture: The coordinator spawns two daemon threads:

- **Game Thread:** Executes the standard Monopoly game loop, calling `player.make_a_move()` sequentially for each player. When an `LLMPlayer` requires a decision, it invokes `TinkerChat.send_message()`, which blocks until the RL policy generates an action.
- **Dispatch Thread:** Continuously reads observations from a shared `request_queue` (populated by all players) and routes them to per-player queues that the RL environment consumes asynchronously.

$$\text{Game} \xrightarrow{\text{observation}} \text{request queue} \xrightarrow{\text{dispatch}} \text{player queues} \rightarrow \tag{2}$$

$$\xrightarrow{\text{RL env}} \text{response\_queue} \xrightarrow{\text{action}} \text{Game} \tag{3}$$

This design allows the synchronous game engine to remain unmodified while interfacing with Tinker's async RL framework. The blocking `send_message()` call ensures that game state advances only after receiving a policy-generated action, maintaining turn-based semantics.

5.1.2. Reward Structure and Dense Feedback

The reward signal is computed from net worth changes, providing dense feedback at each decision point rather than sparse end-game rewards. For player i at step t, the reward is:

$$r_i^{(t)} = \text{NW}_i^{(t)} - \text{NW}_i^{(t-1)} \tag{4}$$

where $\text{NW}_i^{(t)} = M_i^{(t)} + \sum_{p \in P_i} V_p^{(t)}$ represents the player's net worth (cash M plus property values V). This formulation encourages incremental wealth accumulation and penalizes poor decisions immediately, accelerating learning compared to terminal-only rewards. Net worth is calculated using the existing `Player.net_worth()` method, which accounts for mortgaged properties at reduced value.

5.1.3. Self-Play Training Configuration

The training setup generates multiple parallel game instances per batch, with each game running independently in its own coordinator thread. With batch size $B = 128$ and 4 players per game, each batch contains $B/4 = 32$ concurrent games. The dataset configuration targets approximately 128 total games across 4 batches, with each game limited to 10 turns (configurable via `SimulationSettings.n_moves`) to balance training efficiency with game completion.

Environment Wrapper: Each `MonopolyEnv` instance wraps a coordinator and represents a single player's perspective. The environment implements Tinker's `Env` interface:

- `initial_observation()`: Waits for the first decision point and returns the rendered game state.
- `step(action)`: Parses the action string, sends it to the coordinator, and awaits the next observation with computed reward.
- `get_observation()`: Constructs a `ModelInput` using the configured renderer (e.g., `llama3`) from the raw game state string.

The renderer converts the game's natural language prompts (identical to those used in the original LLM setup) into token sequences compatible with the policy model's expected format, ensuring that the RL agent receives the same contextual information as the original MonoManiac agents.

5.1.4. Policy Learning and Model Updates

The training loop follows Tinker's standard RL pipeline: collect rollouts from parallel game instances, compute advantages using the collected trajectories, and update the policy via importance sampling or PPO. All four players in each game share the same policy weights, creating a symmetric self-play scenario where the policy learns to play optimally against itself. As training progresses, the policy encounters increasingly sophisticated opponents (itself at later training stages), driving continuous improvement in strategic play, negotiation, and resource management.

6. Future Work

While MonoManiac demonstrates strong improvements in structured decision-making and negotiation-driven gameplay, several limitations in both the agent design and evaluation framework highlight promising avenues for future research.

Enhancing Long-Horizon Reasoning. Across simulations, LLM agents frequently failed to account for long-term cash flow constraints, delayed rent risk, and the nonlinear payoff structure of property improvements. Future work should incorporate explicit long-horizon value estimation, either through learned value functions, hybrid symbolic–LLM planning modules, or trajectory-level reward shaping (e.g., penalizing avoidable bankruptcies or rewarding liquidity-aware strategies).

Dynamic Risk Calibration and Uncertainty Modeling. LLMs consistently exhibited overly conservative financial behavior in the early game and overly aggressive trades in the mid game. A fruitful direction is to introduce explicit uncertainty modeling into the decision pipeline, such as probabilistic roll-out estimates or Monte Carlo tree sampling guided by LLM priors. This may allow agents to internalize risk–reward trade-offs absent from purely text-based reasoning.

Learning Trade Valuation from Self-Play. A major performance bottleneck lies in the agent's inability to evaluate asymmetric trades involving monopolies or high-leverage properties. Future work may integrate a differentiable trade evaluator trained on large-scale self-play data, enabling LLMs to anchor negotiation proposals around calibrated utility estimates rather than heuristic or narrative justification.

Reducing Context-Length Sensitivity. The current architecture constructs large textual state descriptions, making performance sensitive to formatting and model context windows. Future iterations should explore:

- structured, schema-based state embeddings,
- compressed representations with symbolic abstractions,
- and retrieval-augmented state decomposition.

These approaches may reduce hallucination, prevent state drift, and improve the reliability of long games.

Adversarial and Deceptive Negotiation. Although LLMs can generate coherent negotiation proposals, they lack stable strategies for deception, bluffing, or coalition building. Extending the benchmark with adversarial training settings—e.g., hidden goals, asymmetric information, or multi-agent alliances—would enable more realistic evaluations of social reasoning capabilities.

Scalable Multi-Agent Evaluation. Current experiments focus on small-scale games with limited agent diversity. Future work should evaluate MonoManiac in:

- large populations of heterogeneous agents,
- mixed LLM–human gameplay,
- and meta-game tournaments assessing robustness to strategy drift.

This would allow for a more rigorous study of emergent cooperation, collusion resistance, and negotiation stability.

Acknowledgements

This project is completed as a requirement for Natasha Jaque's course *CSE 599J1 Social Reinforcement Learning*. This project represents an equal collaborative effort among all team members. To support transparency and reproducibility, all code and project materials are publicly available in our GitHub repository for the benefit of the scientific community.

References

DeepSeek-AI et al. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

(FAIR), M. F. A. R. D. T. et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022. doi: 10.1126/science.ade9097. URL https://www.science.org/doi/abs/10.1126/science.ade9097.

Hasbro, Inc. Monopoly: Official rules. https://www.hasbro.com/common/instruct/00009.pdf, 2008. Accessed: December 7, 2024.

Jaques, N., Ghandeharioun, A., Shen, J. H., Ferguson, C., Lapedriza, A., Jones, N., Gu, S., and Picard, R. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, 2019. URL https://arxiv.org/abs/1907.00456.

Laban, P., Hayashi, H., Zhou, Y., and Neville, J. Llms get lost in multi-turn conversation, 2025. URL https://arxiv.org/abs/2505.06120.

OpenAI, :, Agarwal, S., Ahmad, L., Ai, J., Altman, S., Applebaum, A., Arbus, E., Arora, R. K., Bai, Y., Baker, B., Bao, H., Barak, B., Bennett, A., Bertao, T., Brett, N., Brevdo, E., Brockman, G., Bubeck, S., Chang, C., Chen, K., Chen, M., Cheung, E., Clark, A., Cook, D., Dukhan, M., Dvorak, C., Fives, K., Fomenko, V., Garipov, T., Georgiev, K., Glaese, M., Gogineni, T., Goucher, A., Gross, L., Guzman, K. G., Hallman, J., Hehir, J., Heidecke, J., Helyar, A., Hu, H., Huet, R., Huh, J., Jain, S., Johnson, Z., Koch, C., Kofman, I., Kundel, D., Kwon, J., Kyrylov, V., Le, E. Y., Leclerc, G., Lennon, J. P., Lessans, S., Lezcano-Casado, M., Li, Y., Li, Z., Lin, J., Liss, J., Lily, Liu, Liu, J., Lu, K., Lu, C., Martinovic, Z., McCallum, L., McGrath, J., McKinney, S., McLaughlin, A., Mei, S., Mostovoy, S., Mu, T., Myles, G., Neitz, A., Nichol, A., Pachocki, J., Paino, A., Palmie, D., Pantuliano, A., Parascandolo, G., Park, J., Pathak, L., Paz, C., Peran, L., Pimenov, D., Pokrass, M., Proehl, E., Qiu, H., Raila, G., Raso, F., Ren, H., Richardson, K., Robinson,

D., Rotsted, B., Salman, H., Sanjeev, S., Schwarzer, M., Sculley, D., Sikchi, H., Simon, K., Singhal, K., Song, Y., Stuckey, D., Sun, Z., Tillet, P., Toizer, S., Tsimpourlas, F., Vyas, N., Wallace, E., Wang, X., Wang, M., Watkins, O., Weil, K., Wendling, A., Whinnery, K., Whitney, C., Wong, H., Yang, L., Yang, Y., Yasunaga, M., Ying, K., Zaremba, W., Zhan, W., Zhang, C., Zhang, B., Zhang, E., and Zhao, S. gpt-oss-120b gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Sarkar, B., Shih, A., and Sadigh, D. Diverse conventions for human-ai collaboration, 2023. URL https://arxiv.org/abs/2310.15414.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Team, G. et al. Gemini: A family of highly capable multimodal models, 2025. URL https://arxiv.org/abs/2312.11805.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Xie, Y., Ma, Y., Lan, S., Yuille, A., Xiao, J., and Wei, C. Play to generalize: Learning to reason through game play, 2025. URL https://arxiv.org/abs/2506.08011.

Contribution Statement

Exploratory analysis
Runlin led data analysis to identify patterns in game states and visualization. Federico helped with the analysis, running certain solution to confirm the results.
Engine Implementation
Larry developed the core Monopoly game environment and rule enforcement. Runlin implemented the specific logic for agent negotiation protocols within the engine.
Sampling
Larry, Runlin and Federico all ran tests and evaluated the results. Runlin did a more detailed analysis using different model classes and quantified the result with plots.
Reinforcement Learning
Larry setup the MonoManic agent RL self-play using tinker and still has multiple training runs in-progress.
Writing
All authors contributed to the writing and editing of the manuscript. Federico led the organization of the paper and presentation structure.
Other Contributions
Federico conducted the literature review with a focus on the conceptual and project-level motivations. Larry and Runlin contributed to the implementation-focused portion of the literature review.