Agents of Change: Self-Evolving LLM Agents for Strategic Planning

Nikolas Belle*, **Dakota Barnes***, **Alfonso Amayuelas,**
Ivan Bercovich, **Xin Eric Wang**, **William Wang**
University of California, Santa Barbara
{nbelle, dakotabarnes, amayuelas}@ucsb.edu,
ibercovich@gmail.com, ericxwang@ucsb.edu, william@cs.ucsb.edu

Abstract

Recent advances in LLMs have enabled their use as autonomous agents across a range of tasks, yet they continue to struggle with formulating and adhering to coherent long-term strategies. In this paper, we investigate whether LLM agents can self-improve when placed in environments that explicitly challenge their strategic planning abilities. Using the board game *Settlers of Catan*, accessed through the open-source Catanatron framework, we benchmark a progression of LLM-based agents, from a simple game-playing agent to systems capable of autonomously rewriting their own prompts and their player agent's code. We introduce a multi-agent architecture in which specialized roles (Analyzer, Researcher, Coder, and Player) collaborate to iteratively analyze gameplay, research new strategies, and modify the agent's logic or prompt. By comparing manually crafted agents to those evolved entirely by LLMs, we evaluate how effectively these systems can diagnose failure and adapt over time. Our results show that self-evolving agents, particularly when powered by models like Claude 3.7 and GPT-4o, outperform static baselines by autonomously adopting their strategies, passing along sample behavior to game-playing agents, and demonstrating adaptive reasoning over multiple iterations.

1 Introduction

Large Language Models (LLMs) have achieved impressive results across a wide range of language and reasoning tasks. However, when it comes to long-term planning and strategic decision-making, current LLMs still face serious limitations [1]. LLMs are inherently trained for local coherence in text generation rather than long-horizon utility maximization [19]. This makes it challenging for an LLM agent to exhibit strategic, long-term planning [28, 4] in multi-step domains like board games. *Settlers of Catan* (Catan) is a prime example of a multi-agent strategy game requiring players to plan resource management, expansion, and negotiation over many turns. Traditional game AI methods (e.g. search or reinforcement learning) have achieved superhuman performance in perfect-information games such as Chess [15] and Go [24], yet Catan poses unique challenges with its stochastic dice rolls and partial observability. This gap motivates exploring how LLM-based agents can autonomously improve their long-horizon strategic planning in such environments.

We investigate how LLM agents can self-modify and evolve to enhance strategic planning over long horizons. Using the open-source *Catanatron* framework [5], we design four agent architectures with progressively greater self-improvement abilities: **(1)** a *BaseAgent* that maps an unstructured game state description directly to an action, **(2)** a *StructuredAgent* that receives a representation of the game state, available actions, and a basic strategy in natural language for better parsing and guidance, **(3)** a

*Equal contribution

Preprint. Under review.

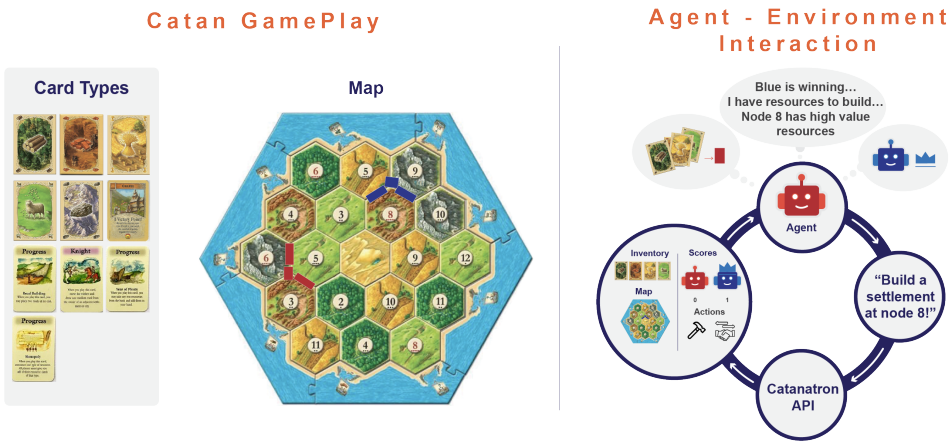


Figure 1: **Overview of Catan gameplay and LLM-agent interaction**. **Left**: *Settlers of Catan* – Players take turns to gather, trade, and spend resources to build on a modular board in a stochastic, partially observable strategy game. The objective is to reach 10 victory points by constructing settlements, roads, and cities [3, 2]. **Right**: Our LLM-based framework interacts with the Catanatron API, leveraging game state information and strategic reasoning to decide actions. Through repeated play and self-modification, agents evolve more coherent long-term strategies [25, 16, 17, 13, 33].

PromptEvolver where an Evolver agent and a Player agent interact for up to 10 iterations to refine, test, and evaluate a prompt for an LLM to play Catan, and **(4)** a *AgentEvolver* comprising of Evolver, Analyzer, Researcher, Coder, and Player roles, which can autonomously rewrite its game-playing code between games. Approaches (3) and (4) draw inspiration from frameworks like AutoGPT that coordinate multiple specialist AI roles [34] and from LLM self-reflection techniques [23], but we apply these ideas to a competitive, partially observable game environment.

We evaluate these agents against *Catanatron*'s strongest heuristic-based bot (an alpha–beta search AI) in head-to-head games. Each game-playing agent (using different models such as GPT-4o, Claude 3.7, or Mistral as backends) plays a series of games against the AlphaBeta bot, and we track relevant metrics such as average victory points, number of settlements/cities, largest army, and other development indicators. Our experiments reveal whether our multi-agent architecture can create agents that achieve higher scores or more effective long-term plans (e.g. building cities earlier, securing Largest Army) compared to simpler LLM agents and to the fixed strategy of the AlphaBeta bot. Additionally, they showcase how LLMs from different providers strategize in developing successful game-playing agents, how they use tools to diagnose and fix the core issues leading to poor performance, and implement strategies into the game-playing agents they design.

Main Contributions. We highlight the following key contributions from our work:

- **LLM Self-Evolving Agent Framework:** We propose a novel framework that enables *autonomous prompt and code evolution* in game-playing agents. This framework allows agents to iteratively self-improve in a long-horizon complex board game with no human intervention, refining their strategies and rewriting their own decision-making code. This extends the role of LLMs from passive solvers to active, self-improving designers.
- **Benchmarking strategic reasoning and planning in a complex game – *Settlers of Catan*:** Using Catanatron [5], we conduct extensive experiments with LLM agents in head-to-head Catan games. We benchmark their performance against a strong search-based bot. Agents with iterative self-improvement demonstrate more coherent strategies and higher average points than those without, narrowing the gap with the expert baseline.
- **Empirical evidence of performance gains from evolution**: Through extensive experiments, we show that agents capable of prompt and code evolution achieve consistently higher performance than static baselines. The PromptEvolver, in particular, outperforms fixed agents across key metrics, and its gains are amplified when paired with stronger base models, seen in Claude 3.7's 95% improvement from the BaseAgent (Table 1).

- **Insights from Evolved Strategies:** Through qualitative analysis of game logs and agent-generated code revisions, we examine how the LLM's evolved strategies differ between SOTA LLMs and from human-crafted agents. Our findings reveal how LLM agents learn to prioritize long-horizon objectives (e.g. balanced growth vs. opportunistic plays), adapt to past failures, and implement novel tactics through self-directed refinement.

2 Related Work

Game-Playing AI and Strategy Games Games have long served as benchmarks for AI research [11, 6, 18]. While significant progress has been made in perfect-information games like Chess and Go [22, 24], strategic board games such as *Settlers of Catan*, *Diplomacy* [8] or *Civilization* [21] introduce elements of randomness, partial observability, and multi-agent interaction, posing unique challenges to an AI system [27]. Previous works approached Catan using a specialized neural network architecture to handle its mixed data types, enabling an RL agent to outperform traditional rule-based bots [12]. In contrast, our approach leverages LLMs' natural language understanding to navigate Catan's complexities, focusing on autonomous strategy refinement without relying on extensive training data.

LLM-Based Autonomous Agents The concept of employing LLMs as autonomous agents has gained traction, particularly through frameworks that leverage environment feedback to drive reasoning and action [35, 32]. Sophisticated frameworks like CAMEL [14] and AutoGen [31] facilitating multi-agent collaboration through role-based interactions. These systems assign specific roles to LLM instances, such as task proposer, solver, or critic, enabling complex problem-solving through dialogue and coordination. Our methodology extends this paradigm by introducing a self-evolving multi-agent system where LLMs assume roles like Analyzer, Researcher, Coder, and Player. This architecture allows the agent to autonomously analyze performance, conduct research, modify its codebase, and iteratively enhance its strategy for gameplay.

Self-Improvement and Policy Evolution Recent studies have explored mechanisms for LLMs to self-improve without gradient-based training [26, 7]. Reflexion [23] introduced a framework where LLM agents use verbal feedback to refine their decision-making processes, maintaining an episodic memory of successes and failures. Similarly, PromptBreeder [10] or PromptAgent [30] treats design as an evolutionary process, allowing LLMs to generate and evaluate variations to optimize performance. AlphaEvolve [20] presents an evolutionary coding agent to tackle open scientific problems and algorithm improvement. Voyager [29] demonstrated an LLM-powered agent in Minecraft that iteratively writes and refines code to acquire new skills autonomously. Our work builds upon these concepts by integrating both and code evolution within a competitive, multi-agent environment, enabling the LLM agent to adapt its strategies based on performance feedback over successive game iterations.

While prior research has laid the foundations for LLM-based agents and self-improvement mechanisms, our study uniquely combines these elements within the context of a complex strategy game. By embedding an LLM game-playing agent within a multi-agent system capable of autonomous and code evolution, we investigate the extent to which LLMs can overcome inherent limitations in long-term strategic planning. This approach not only evaluates the agent's performance against a strong heuristic-based bot but also provides insights into the emergent behaviors and strategies developed through self-directed refinement.

3 Background

Settlers of Catan as a Strategic Benchmark *Settlers of Catan* is a 3–4 player board game where players collect and trade resources to build settlements and roads, racing to earn 10 victory points on a modular island map. The game emphasizes resource management, planning, and negotiation, with mechanics like the robber (which blocks resources) adding tactical depth. Catan is known for its balance of luck and skill. **Victory** goes to the first player to reach **10 points**, earned by building and upgrading settlements into cities, buying development cards, and achieving goals like the longest road or largest army. Each settlement is worth 1 point, each city 2, and some development cards grant hidden points or knight bonuses. **Every turn** starts with a dice roll that produces resources for

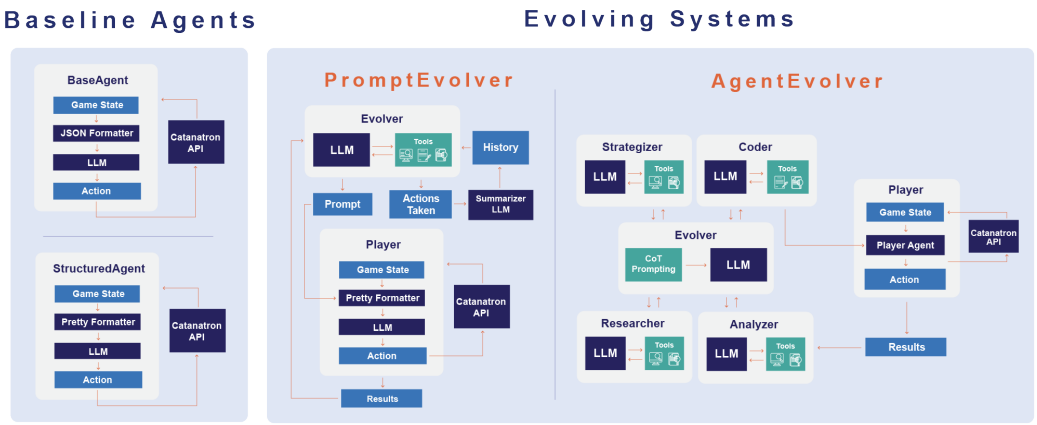


Figure 2: **Diagrams of the LLM-based Agent Architectures**. Baseline agents use LLMs to map Catan game states to actions by direct prompting (*BaseAgent*) or structured formatting (*StructuredAgent*). *PromptEvolver* adds a multi-agent loop where prompts are iteratively refined via analysis and summarization. *AgentEvolver* enables autonomous code evolution, with the Analyzer, Researcher, Coder, and Strategizer agents collaboratively redesigning player logic from gameplay feedback.

players with adjacent settlements. The active player may then trade and build. If a 7 is rolled, the robber is activated, blocking a tile and stealing a resource. Players must plan expansions, balance upgrades, and trade strategically to manage luck. This need for adaptation and foresight makes Catan a strong benchmark for evaluating strategic reasoning in agents.

The Catanatron Framework We use the open-source Python-based simulator **Catanatron** as our evaluation environment. Designed for automated gameplay of *Settlers of Catan*, Catanatron offers a programmatic interface for integrating custom agents and supports rapid simulation at scale. It faithfully implements the game's rules and dynamics, capturing key strategic elements such as resource management, trade negotiation via structured proposals, and randomness introduced by dice rolls. Each game consists of multiple players competing to reach ten victory points, with players interacting through well-defined game states that include current resources, board positions, available actions, and observable opponent statuses. Games typically span 40 to 100 turns, allowing for extended observation of agents' long-term planning capabilities. We benchmark our LLM-driven agents against **AlphaBeta**, the best-performing heuristic agent provided through the API which uses a depth-2 alpha-beta pruning algorithm with heuristic evaluation to select actions.

4 Method

To investigate the capabilities of LLMs in autonomously developing strategic planning abilities, we make them interact with the complex strategic board game: *Settlers of Catan*. We systematically evaluate different agent architectures, each leveraging LLMs in distinct roles: player, refiner, and creator. All agents share the common objective of winning the game. By doing so, we aim to analyze how these agents adapt and evolve their strategic reasoning through iterative self-improvement, particularly in comparison to traditional heuristic-based approaches.

4.1 Agent Architectures

We develop a series of LLM-based agents using the LangChain and LangGraph frameworks. LangChain facilitates structured interactions among multiple LLMs, managing iterative conversations, chaining, and message passing between agents. LangGraph complements this by offering visual and computational tools for modeling agent interactions as directed graphs, enabling transparent orchestration and robust management of complex agent workflows.

BaseAgent Our baseline LLM agent, named the *BaseAgent*, receives raw structured input describing the complete current game state from the Catanatron API, including available actions at each turn. The LLM directly outputs a selected action without additional engineering or textual refinements. This baseline measures the inherent strategic reasoning capability of an LLM when minimally guided and provides foundational performance metrics against the AlphaBeta agent.

StructuredAgent To establish a more informed performance baseline, we introduce the *StructuredAgent*. This agent utilizes human-developed prompts, based on the public GitHub repository [9]. These structured prompts explicitly guide the LLM in interpreting the state representation, highlighting key considerations such as resource prioritization, trade heuristics, and long-term strategic planning. By incorporating human expertise into the design, we establish a practical performance baseline for a single-agent, static-LLM.

PromptEvolver Next, the *PromptEvolver* is a multi-agent system designed to iteratively enhance the player's based on experiential feedback and strategic analysis. This architecture consists of the following specialized agents:

- **Evolver Agent**: Provided with access to game results, evolution history, and tools to search the web, view local files, and edit the Player Agent's prompt.
- **Player Agent**: Utilizes the evolved prompt to choose an action on a turn-by-turn basis.

We implement the *PromptEvolver* to evaluate whether LLM-based multi-agent systems can autonomously surpass human-level prompt engineering to elicit strategic planning in LLMs. The iterative refinement loop enables the system to learn strategically effective prompts based on previous game outcomes, analyses, and external knowledge. This helps us assess the capacity of LLMs to learn prompts that provide plan-following guidance to other LLM agents, despite their inherent struggle with following long-term plans themselves.

AgentEvolver Finally, the *AgentEvolver* represents a multi-agent framework that is autonomously capable of modifying or rewriting the player agent's underlying decision-making logic, starting with a blank template. This AgentEvolver architecture consists of the following specialized agents:

- **Evolver Agent**: The central coordinator. After each game, it digests Analyzer reports, then orchestrates a new iteration by querying the appropriate supporting agents.
- **Analyzer Agent**: Invoked after every game, and can be called by Evolver. It evaluates the player's gameplay, identifies weaknesses, and summarizes areas for improvement.
- **Researcher Agent**: Handles domain-specific queries about the Catanatron codebase or broader Catan strategy, utilizing both local file access and web search via the Tavily API.
- **Strategizer Agent**: Suggests high-level gameplay strategies or critiques past strategic choices. It uses both current and historical data as reference, as well as the web.
- **Coder Agent**: Translates proposed changes into concrete code modifications. It receives the latest player code and outputs a new version, along with a summary of changes made.
- **Player Agent**: The iteratively refined agent that plays the game.

We deploy the *AgentEvolver* architecture to investigate the extent to which LLM-based systems can autonomously design and implement complex strategic behaviors from first principles. Each of these agents operates within a persistent message-passing framework, maintaining memory across iterations to accumulate experience in their respective domains. This design allows, for example, the Analyzer to become more proficient at diagnosing recurring failures, or the Coder to learn stylistic patterns that align with successful strategies. By empowering the Evolver agent to independently analyze gameplay, conduct research, build strategy, and generate executable code, we aim to assess the capability of LLMs not just to refine but fundamentally create strategically competent agents. This evaluation provides insights into the potential of LLMs for advanced autonomous development and optimization of complex reasoning processes.

5 Experimental Setup

To rigorously evaluate the performance of our LLM-based agents, we conduct controlled experiments using the open-source *Catanatron* environment. The data was collected on a MacBook Pro 2019 16GB, and a MacBook M1 Max 2021 32GB over a total time of 60 hours. Our evaluation focuses on measuring each agent's strategic competence in competitive 2-player Catan games by comparing them head to head against the strongest built-in heuristic bot: *AlphaBeta*.

Language Models All LLM-based agents were instantiated using one of the following models: GPT-4o (via OpenAI API), Claude 3.7 (via AWS Anthropic Bedrock API), and Mistral-large-latest (via Mistral AI API). These models were chosen for their advanced reasoning capabilities, availability, and diversity in architecture. There was no limit to input and output size.

Test Procedure We evaluate each of our four agents (*BaseAgent*, *StructuredAgent*, *PromptEvolver*, *AgentEvolver*) against the AlphaBeta bot in a series of head-to-head games. Each game is played under identical conditions to ensure fair comparison. The specific setup is as follows:

- **Baseline Agents (BaseAgent & StructuredAgent):** Each agent plays 10 full games against AlphaBeta. Random seeds are fixed for initial board state generation to ensure consistency across runs.
- **PromptEvolver:** This multi-agent system iteratively evolves its prompt across 10 evolution cycles with the mini-map. In each evolution, the evolver gathers information to update the prompt, and the updated player plays 5 games against AlphaBeta. We return the average victory points across games per turn. A 10 game trial is then performed with the highest scoring prompt using the full-map.
- **AgentEvolver:** Similar to the *PromptEvolver* setup, this system undergoes 10 full-code evolution cycles using the mini-map. Each cycle includes the Evolver decision loop ending with writing new code, followed by 10 games played using the newly generated agent. We report the average victory points across 10 games for the highest scoring generated agent (Table 3).

All experiments track average victory points (VP), win rate, average number of settlements, cities, number of longest road achievements, largest army achievements, and victory points from development cards, turn count, and logs from each agent used in the respective architectures.

Evaluation We compare the best generated player from the PromptEvolver and the StructuredAgent to the BaseAgent as a baseline since they all utilize an LLM to make decisions. The best generated player from the AgentEvolver is compared against the RandomAgent as it is evolved from a blank file, without any explicit instruction to use an LLM as the decision making unit.

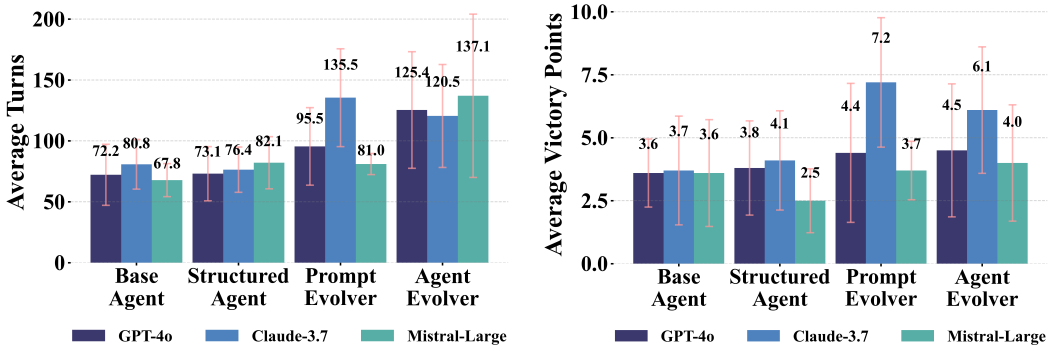
Randomness Control All games were initialized using randomized board layouts and due to inherent randomness in Catan (dice rolls, opponent behavior), we average results over multiple games to ensure statistical robustness. We also note that while AlphaBeta's behavior is deterministic, LLM-based agents may produce slight variability even with temperature control. We use a temperature value of 0.7 for our experiments.

6 Results

This section presents the empirical results of our experiments, evaluating how effectively various LLM-driven agent architectures performed against the heuristic-based AlphaBeta baseline in the *Settlers of Catan* game. The results are gathered with all agents against AlphaBeta across 10 games on the default map. We showcase the best performing generated agents from the PromptEvolver and AgentEvolver from their 10 evolutions.

Table 1: Agent performance across models (avg score and avg turns).

Agent	GPT-4o		Claude 3.7		Mistral Large	
	Score	Turns	Score	Turns	Score	Turns
BaseAgent	3.60	72.20	3.70	80.80	3.60	67.80
StructuredAgent	3.80 ↑ 6%	73.10	4.10 ↑ 11%	76.40	2.50 ↓ 31%	82.10
PromptEvolver	4.40 ↑ 22%	95.50	7.20 ↑ 95%	135.50	3.70 ↑ 3%	81.00
RandomPlayer	2.34	68.12	2.34	68.12	2.34	68.12
AgentEvolver	2.73 ↑ 36%	70.00	2.80 ↑ 40%	70.50	2.67 ↑ 34%	68.03



(a) Average Game Length

(b) Average Victory Points

Figure 3: **Performance comparison across LLM-agent architectures and model backends**. **Left:**Figure 3a. Average number of turns taken to complete the game across architectures. Higher values such as Claude-3.7 PromptEvolver indicate a more competitive game against the AlphaBeta opponent. **Right**: Figure 3b Average Victory Points (VP) achieved, showing that self-improving agents (PromptEvolver and AgentEvolver) outperform baseline agents, especially when paired with more capable LLMs like GPT-4o and Claude 3.7.

7 Discussion

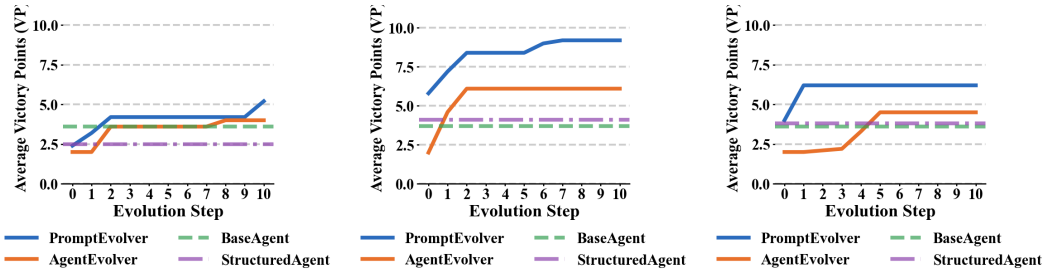
PromptEvolver Evolution Analysis The PromptEvolver framework demonstrated considerable variability in effectiveness among the different language models (Claude 3.7, GPT-4o, Mistral Large). In general, the system successfully improved agent performance by autonomously iterating on strategic prompts, although the degree of success heavily depended on the model used. Claude 3.7 exhibited the most significant strategic advancements, systematically developing detailed strategic prompts that outlined clear short-term and long-term plans, including precise settlement placement, resource prioritization, development card usage, and robust response strategies against opponent actions. This approach resulted in substantial performance gains (up to a 95% improvement), notably surpassing the baselines. GPT-4o achieved moderate performance gains through incremental enhancements primarily centered around improving mid-game strategies, robber placement, and targeted trading tactics. However, the improvements, while steady, were less aggressive and lacked the comprehensive strategic vision developed by Claude. Mistral Large showed the least effectiveness, achieving minimal performance gains due to superficial revisions and a limited strategic horizon. Mistral primarily focused on reactionary updates based on immediate past performance without deeper strategic planning or addressing fundamental gameplay weaknesses, evident by the low percent change in tokens across iterations in Table 2. Consequently, performance metrics saw marginal improvements only.

A key limitation observed across the PromptEvolver was its dependence on the inherent strategic reasoning capabilities of the underlying LLM. Models with less sophisticated strategic reasoning (such as Mistral) faced significant challenges in generating meaningful prompt refinements. Additionally, the associated randomness and complexity of the game made it difficult for the PromptEvolver to

Table 2: PromptEvolver Evaluation Metrics Across All Iterations

Model	% Improvement	Length (Tokens)	% Change (Tokens)
GPT-4o	22.2%	440	53.4%
Claude-3.7	94.6%	754	52.7%
Mistral-Large	2.7%	337	13.9%

recognize the Player Agents reaction to the new prompt. Despite these limitations, the PromptEvolver demonstrated significant strengths. Notably, the models were able to identity through self learning what kind of input would be best for their own decision process. The approach effectively demonstrated the capacity of LLM-driven agents to improve not only their gameplay strategy, but also the structure of their own prompts.



(a) Mistral Large (b) Claude 3.7 (c) GPT 4o

Figure 4: **Evolution performance of LLM agents over time**. Average Victory Points (VP) across evolution steps for each model (Mistral Large, Claude 3.7, GPT-4o). PromptEvolver consistently improves with more iterations, especially when paired with stronger models like Claude 3.7 and GPT-4o. AgentEvolver shows variable performance gains, while static baselines (BaseAgent, StructuredAgent) remain flat.

AgentEvolver Evolution Analysis The AgentEvolver architecture showed promising yet mixed results in its capability to autonomously design and iteratively enhance player agents through direct code generation and refinement. The results highlighted significant variations in strategic depth and implementation robustness among the tested models (Claude 3.7, GPT-4o, Mistral Large). As shown in Table 3, Claude 3.7 emerged as the top-performing model, showing pronounced strategic evolution with a 40.0% improvement. It consistently implemented effective adaptations such as improved settlement and road planning, strategic use of development cards, and context-sensitive resource management. Claude's helper agents (Analyzer, Researcher, Strategizer, Coder) were notably proficient at providing sophisticated strategies, accurate analytical feedback, and reliable code implementations, resulting in steady incremental performance improvements. GPT-4o provided modest strategic and implementation advancements, excelling primarily in technical debugging and reliable code refinements. Its evolutionary process was stable but conservative, focusing more on resolving immediate functional issues rather than pursuing ambitious strategic enhancements. As a result, GPT-4o delivered consistent but limited improvements relative to Claude. Mistral Large exhibited significant difficulties with strategic coherence and adaptability, typically repeating basic strategies without adequately responding to dynamic game conditions. Its helper agents frequently produced superficial analysis and unstable implementations, substantially limiting evolutionary improvements and overall performance gains.

Table 3: AgentEvolver Evaluation Metrics Across All Iterations

Model	% Improvement	Length (Tokens)	% Change (Tokens)	% No Errors
GPT-4o	36.5%	434	27.4%	35%
Claude-3.7	40.0%	598	53.5%	50.0%
Mistral-Large	33.5%	431	5.6%	50.0%

However, the AgentEvolver system struggled to surpass simpler LLM agents and the AlphaBeta. While individual agents demonstrated potential, the absence of a robust mechanism for long-term memory and advanced strategy integration limited consistent strategic refinement, ultimately reducing competitive effectiveness of the Player relative to both static LLM-based agents and the more strategically consistent AlphaBeta heuristic. Nevertheless, the project constitutes a significant success in demonstrating the feasibility of constructing a fully autonomous framework guided by a large language model, capable of generating and refining executable code entirely from scratch. Notably, the system operated without prior access to Catanatron documentation and still succeeded in discovering, adapting to, and leveraging the game's API and strategic mechanics through self-guided exploration and debugging. This highlights the model's capacity to autonomously learn complex domain-specific logic and programming interfaces without human intervention. Moreover, the Player Agent achieved consistent improvements over the Random baseline and lays a promising foundation for applying such autonomous, evolution-driven LLM architectures to other strategic environments and code synthesis tasks.

8 Limitations

While our approach demonstrates the potential of self-improving LLM agents, several limitations should be noted. First, the system is computationally expensive: each evolution cycle involves multiple agent roles, prompt/code generation, and full-game simulations, which limits scalability and rapid experimentation. Second, our evaluation is restricted to Settlers of Catan and it remains unclear how well the framework generalizes to other strategic or real-world environments. Additionally, performance is strongly tied to the base model's capabilities. Less capable LLMs, such as Mistral, struggled to meaningfully improve, highlighting a bottleneck in the underlying reasoning abilities. We also evaluate primarily against a fixed heuristic-based bot (AlphaBeta) and do not benchmark against learning-based baselines such as reinforcement learning agents.

9 Ethical Statement & Broader Impact

We study self-improving LLM agents in a closed, low-risk environment (Settlers of Catan), where agents iteratively revise prompts and code to enhance strategic play. To ensure safety, all generated code is sandboxed and manually reviewed, with no access to external systems. We avoid anthropomorphic framing, clarifying that self-improvement reflects architectural processes, not autonomy. Our framework emphasizes transparency and interpretability, producing auditable artifacts unlike black-box methods. We view this work as a step toward safer, more accountable AI systems and will open-source our code with clear safeguards and usage guidelines.

10 Conclusion

This work demonstrates the viability and promise of self-evolving LLM agents in complex, long-horizon decision-making tasks. By embedding language models within multi-agent systems capable of autonomously refining prompts and rewriting executable code, we show that LLMs can meaningfully improve their strategic behavior in a competitive game environment. Our experiments in Settlers of Catan reveal that agents using iterative evolution consistently outperform static baselines, adopting more coherent and long-term strategies over time. Beyond performance gains, our analysis highlights distinct behavioral patterns across LLMs in how they interpret failure, integrate feedback, and adapt strategy. These findings suggest a growing potential for LLMs not just as players, but as designers of themselves. Future work will explore extensions to multi-agent negotiation, broader generalization across games, and tighter integration of symbolic and neural reasoning for even more sophisticated autonomous improvement.

References

[1] Mohamed Aghzal, Erion Plaku, and Ziyu Yao. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning, 2025.

[2] Catan Collector. How to Identify Your Version of Catan. https://catancollector.com/catan-links/how-to-identify-your-version-of-catan. Accessed: 2025-05-15.

[3] Catan Fusion. 3 Royal Weddings in Catan. http://catanfusion.com/index.php/blog/entry-3-royal-weddings-in-catan. Accessed: 2025-05-15.

[4] Yanan Chen, Ali Pesaranghader, Tanmana Sadhu, and Dong Hoon Yi. Can we rely on llm agents to draft long-horizon plans? let's take travelplanner as an example. *arXiv preprint arXiv:2408.06318*, 2024.

[5] B. Collazo. Catanatron: Settlers of catan bot simulator and strong ai player. https://github.com/bcollazo/catanatron, 2025.

[6] Anthony Costarelli, Mat Allen, Roman Hauksson, Grace Sodunke, Suhas Hariharan, Carlson Cheng, Wenjie Li, Joshua Clymer, and Arjun Yadav. Gamebench: Evaluating strategic reasoning abilities of llm agents. *arXiv preprint arXiv:2406.06613*, 2024.

[7] Xiangjue Dong, Maria Teleki, and James Caverlee. A survey on llm inference-time self-improvement. *arXiv preprint arXiv:2412.14352*, 2024.

[8] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[9] Same Farrar. Catanatron-llm. https://github.com/samefarrar/catanatron_llm, 2024. Accessed: 2025-05-16.

[10] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

[11] Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Large language models and games: A survey and roadmap. *IEEE Transactions on Games*, 2024.

[12] Quentin Gendre and Tomoyuki Kaneko. Playing catan with cross-dimensional neural network. In *Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23–27, 2020, Proceedings, Part II 27*, pages 580–592. Springer, 2020.

[13] Hilmy Abiyyu A. Robot icons created by Hilmy Abiyyu A. - Flaticon. https://www.flaticon.com/free-icons/robot. Accessed: 2025-05-15.

[14] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023.

[15] Daniel Monroe and Philip A. Chalmers. Mastering chess with a transformer model, 2024.

[16] murmur. Conversation icons created by murmur - Flaticon. https://www.flaticon.com/free-icons/conversation. Accessed: 2025-05-15.

[17] murmur. Thought bubble icons created by murmur - Flaticon. https://www.flaticon.com/free-icons/thought-bubble. Accessed: 2025-05-15.

[18] Muhammad Umair Nasir, Steven James, and Julian Togelius. Gametraversalbenchmark: Evaluating planning abilities of large language models through traversing 2d game maps. *arXiv preprint arXiv:2410.07765*, 2024.

[19] Siddharth Nayak, Adelmo Morrison Orozco, Marina Ten Have, Vittal Thirumalai, Jackson Zhang, Darren Chen, Aditya Kapoor, Eric Robinson, Karthik Gopalakrishnan, James Harrison, Brian Ichter, Anuj Mahajan, and Hamsa Balakrishnan. Llamar: Long-horizon planning for multi-agent robots in partially observable environments, 2025.

[20] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. Technical report, Google DeepMind, 2025. White paper.

[21] Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, et al. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents. *arXiv preprint arXiv:2401.10568*, 2024.

[22] John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, et al. Mastering board games by external and internal planning with language models. *arXiv preprint arXiv:2412.12119*, 2024.

[23] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[24] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[25] Smashicons. Trade icons created by Smashicons - Flaticon. `https://www.flaticon.com/free-icons/trade`. Accessed: 2025-05-15.

[26] Yuda Song, Hanlin Zhang, Carson Eisenach, Sham Kakade, Dean Foster, and Udaya Ghai. Mind the gap: Examining the self-improvement capabilities of large language models. *arXiv preprint arXiv:2412.02674*, 2024.

[27] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. In *Advances in computer games*, pages 21–32. Springer, 2009.

[28] Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo, and Subbarao Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark), 2023.

[29] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.

[30] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[31] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.

[32] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

[33] yaicon. Build icons created by yaicon - Flaticon. `https://www.flaticon.com/free-icons/build`. Accessed: 2025-05-15.

[34] Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional opinions, 2023.

[35] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

A Performance metrics for each agent with various models

Agent	Win Rate	Avg VP	Avg Turns	Avg Settles	Avg Cities	Avg Roads	Avg Army	Avg Dev VP
BaseAgent	0.00	3.60	72.20	1.80	0.50	0.00	0.00	0.80
StructuredAgent	0.00	3.80	73.10	2.00	0.30	0.00	0.20	0.80
PromptEvolver	0.10	4.40	95.50	2.10	0.60	0.00	0.30	0.50
RandomPlayer	0.00	2.34	68.12	2.00	0.03	0.00	0.04	0.20
AgentEvolver	0.00	4.50	125.4	1.70	0.90	0.00	0.20	0.60

Table 4: Detailed performance metrics across agent types for GPT-4o.

Agent	Win Rate	Avg VP	Avg Turns	Avg Settles	Avg Cities	Avg Roads	Avg Army	Avg Dev VP
BaseAgent	0.00	3.70	80.80	2.10	0.30	0.00	0.20	0.60
StructuredAgent	0.00	4.10	76.40	1.80	0.20	0.00	0.40	1.10
PromptEvolver	0.10	7.20	135.5	0.90	1.30	0.10	0.80	1.90
RandomPlayer	0.00	2.34	68.12	2.00	0.03	0.00	0.04	0.20
AgentEvolver	0.10	6.10	120.5	1.80	0.70	0.20	0.50	1.50

Table 5: Detailed performance metrics across agent types for Claude 3.7.

Agent	Win Rate	Avg VP	Avg Turns	Avg Settles	Avg Cities	Avg Roads	Avg Army	Avg Dev VP
BaseAgent	0.00	3.60	67.80	2.50	0.10	0.00	0.20	0.50
StructuredAgent	0.00	2.50	82.10	2.30	0.00	0.10	0.00	0.00
PromptEvolver	0.00	3.70	81.00	1.90	0.30	0.30	0.10	0.40
RandomPlayer	0.00	2.34	68.12	2.00	0.03	0.00	0.04	0.20
AgentEvolver	0.10	4.00	137.1	0.60	1.60	0.10	0.00	0.00

Table 6: Detailed performance metrics across agent types for Mistral Large.

B Prompt Example

```
You are playing Settlers of Catan. Your task is to analyze the game state and choose the best action from
 the available options.

Rules:
1. Think through your decision step by step, analyzing the game state, resources, and available actions
2. Your aim is to WIN. That means 10 victory points.
3. Put your final chosen action inside a box like \boxed{5}
4. Your final answer must be a single integer corresponding to the action number
5. If you want to create or update your strategic plan, put it in <plan> tags like:
  <plan>Build roads toward port, then build settlement at node 13, then focus on city upgrades</plan>
6. Analyze the recent resource changes to understand what resources you're collecting effectively
7. Think about the next 2-3 turns, not just the immediate action

Board Understanding Guide:
- The RESOURCE & NODE GRID shows hexagonal tiles with their coordinates, resources, and dice numbers
- The nodes connected to each tile are listed below each tile
- 🔍 marks the robber's location, blocking resource production on that hex
- Settlements/cities and their production are listed in the BUILDINGS section
- Understanding the connectivity between nodes is crucial for road building strategy
- Ports allow trading resources at better rates (2:1 or 3:1)

TURN 1 | Player: BLUE | Action: BUILD_INITIAL_SETTLEMENT

📊 GAME STATUS:
 Longest Road: None (0 segments)
 Largest Army: None (0 knights)
 Robber Location: (0, -1, 1)

🗺️ CATAN RESOURCE & NODE GRID:

🏜️- (0, -1, 1) 🔍            🌾9 (1, -1, 0)
nodes: 2,3,9,10,11,12       nodes: 1,2,6,7,8,9

⛏️4 (-1, 0, 1)            🐑6 (0, 0, 0)            🧱3 (1, 0, -1)
nodes: 3,4,12,13,14,15      nodes: 0,1,2,3,4,5         nodes: 0,1,6,20,22,23

🌾8 (-1, 1, 0)            🌲5 (0, 1, -1)
nodes: 4,5,15,16,17,18      nodes: 0,5,16,19,20,21


BUILDINGS:
- 🔴 RED: 🏠 at node 5 (produces: 🐑6[★★★], 🌾8[★★★], 🌲5[★★])

ROADS:
- 🔴 RED: (4, 5), (5, 4)

PORTS:

👥 PLAYERS:

RED Player:
 Victory Points: 1/10
 Resources: 0 cards (hidden)
 Development Cards: None
 Buildings:
  🏠 Settlements: 1
  🏙️ Cities: 0
  🛣️ Roads: 1 built (Longest: 0)

👉 BLUE Player:
 Victory Points: 0/10
 Resources: None
 Development Cards: None
 Buildings:
  🏠 Settlements: 0
  🏙️ Cities: 0
  🛣️ Roads: 0 built (Longest: 0)

🏦 Bank:
 Development Cards: 25 remaining
 Resources available for trade with bank or ports

🎮 AVAILABLE ACTIONS:

Build Settlement Options:
 [0] Build a settlement at node 1 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [1] Build a settlement at node 2 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [2] Build a settlement at node 3 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [3] Build a settlement at node 6 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [4] Build a settlement at node 7 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [5] Build a settlement at node 8 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [6] Build a settlement at node 9 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [7] Build a settlement at node 10 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [8] Build a settlement at node 11 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [9] Build a settlement at node 12 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [10] Build a settlement at node 13 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [11] Build a settlement at node 14 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [12] Build a settlement at node 15 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [13] Build a settlement at node 17 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [14] Build a settlement at node 18 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [15] Build a settlement at node 19 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [16] Build a settlement at node 20 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [17] Build a settlement at node 21 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [18] Build a settlement at node 22 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)
 [19] Build a settlement at node 23 (Cost: 1 🌲WOOD, 1 🧱BRICK, 1 🌾WHEAT, 1 🐑SHEEP)

❓ Select action by number (0-19):


Based on this information, which action number do you choose? Think step by step about your
options, then put the final action number in a box like \boxed{1}.
```

Figure 5: Sample StructuredAgent LLM prompt.