Language Driven Play: LLMs as Game Playing agents in Slay the Spire

Reva Agarwal, Joey Perrello, Sohail Syed

https://github.com/jperrello/MiniStS-Language-Driven-Play-Reimagined

Abstract

We experiment with Large Language Models (LLMs) to determine their suitability for evaluating procedurally generated content in games. To do so, we utilize a simplified implementation of *Slay the Spire* (MiniStS), a deck-building game requiring arithmetic reasoning and long-term planning. We extended the original research by using current LLM model providers with similar prompting strategies like a Reverse Chain-of-Thought (RCoT) LLM agent, a Monte Carlo Tree Search (MCTS) agent, and a None (no prompt) LLM agent. We also integrated new testing scenarios with the generated cards from the Pirates group. This offered insight into how effective LLMs are at playing with LLM generated cards. Our results demonstrate that the RCoT agent not only achieves high win rates on novel content but also significantly outperforms both heuristic scripts and zero-shot LLMs, validating the necessity of reasoning structures for general game playing.

1. Problem Description

The original paper addresses the difficulty of evaluating procedurally generated game rules. When a PCG system creates new content (e.g., a new card with unique mechanics), ensuring it is balanced typically requires human playtesting. Automated agents are needed, but traditional agents (like Reinforcement Learning) require costly retraining for every new rule.

Our work addresses this by implementing and evaluating a spectrum of **General Game Playing (GGP)** agents. We investigate whether an LLM equipped with reasoning capabilities (Reverse Chain-of-Thought and Chain-of-Thought) can serve as a superior "zero-shot" evaluator compared to three distinct baselines:

1. **Heuristic Script (NoneAgent):** Represents traditional, static game AI.
2. **Naive LLM (BasicAgent):** Represents raw model intuition without reasoning.
3. **Search (MCTS):** Represents simulation-based optimization. This comparative analysis allows us to quantify the specific value of "reasoning" against hard-coded rules and raw computational search.

2. Implementation Overview

Our project is built on the `MiniStS` framework, a headless Python simulation of *Slay the Spire*. We extended the architecture with the following components:

2.1 Grammar-Based Card Generation (GIGL):

We integrated the Pirate group's GIGL (Grammar-based Intelligent Game Logic) Context-Free Grammar system for procedurally generating novel cards. These cards are constructed from blueprints and balanced using a "Power Point" (PP) cost system. The GIGL system allows for systematic generation of cards with novel mechanics that were not present in the training data for LLMs, providing a robust test of generalization capability.

2.2 Agent Implementations

2.2.1 Baseline Agents:

Random: The Random Agent serves as the performance floor for all experiments. At each decision point, it randomly selects from all valid actions (playable cards or end turn).

MCTS Agent: A Monte Carlo Tree Search agent (`g3_files/agents/mcts_bot.py`) that uses UCT to simulate move sequences and determine play based on state value heuristics.

Backtrack (N step look ahead)

The Backtrack Agent uses exhaustive n-step lookahead search, evaluating all possible action sequences to depth n. For our experiments, we use depth=3, meaning the agent considers all possible sequences of 3 moves ahead. This approach is in the original paper.

2.2.2 LLM Agents:

None Agent (Zero shot LLM):
The None Agent implements the minimal prompting approach, asking the LLM to directly select an action index without any reasoning or explanation. This isolates the model's raw intuition for game playing.

RCoT Agent:

A Reverse Chain-of-Thought agent (`rcot_agent.py`) using OpenRouter's GPT models. This agent is instructed to output a strategic explanation after its decision, forcing the model to articulate its understanding of new card text and arithmetic constraints. Additionally, this agent along with CoT are designed to use structured outputs, which is a new output format of the Responses API. It returns responses in a json format rather than plain text, and is claimed to return better results. Following the reference paper, we do not include move history. Each decision is based solely on the current game state. This reduces token usage and prevents context window limitations.

CoT Agent:

A Chain-of-Thought agent (`g3_files/agents/cot_agent.py`) using OpenAI's GPT models. This agent is architected to output its reasoning before reaching a decision, forcing the model to consider important variables before coming to a conclusion. Ultimately, we decided to use RCoT over CoT for the LLM Agent because both agents had similar win rates but RCoT performed better in terms of player hp. CoT agent had an avg player hp of 32.2 while RCoT averaged 37.7 player hp in a control deck scenario. (see Figure 2) Based on these statistics, we using RCoT as our LLM Agent for further testing.

Testing Examples

Run Agent Comparison:
```
python evaluation/evaluate_bot.py 25 4
0 h rcot-gpt41 mcts none basic rndm
--time
```

Test Generated Cards/Scenarios:
```
python evaluation/evaluate_card_gen.py
10 4 20 h rcot-gpt41 --gigl-dir
GIGL/generated_cards
```

3. Results and Evaluation

We conducted extensive testing across 5 distinct scenarios, including standard decks like Bomb and Tolerate, and a random scenario populated by GIGL-generated cards. The resultant data recorded from these tests will be explored in the following sections. All experiments used the following parameters:

- Test count per agent per scenario: 25 simulations
- Starting player health: 80 HP
- Starting mana per turn: 3
- Enemy: Single enemy with 44 HP, alternating between attack (22 damage) and block (10 damage absorbed)
- Evaluation metrics: average final player health

3.1 Prompting Strategies

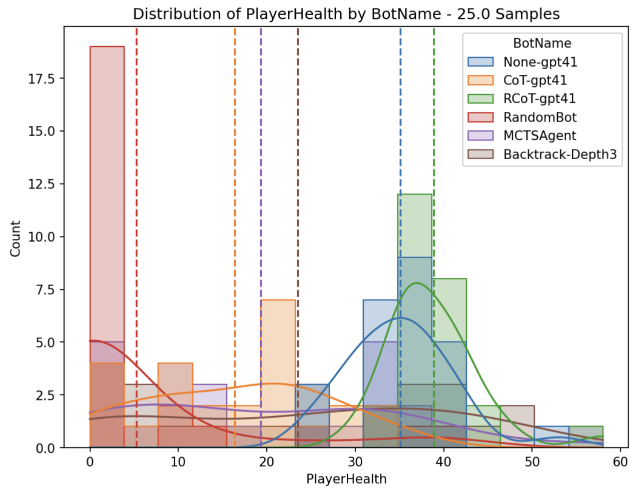


Figure 1: Player Health across 25 simulations of the started deck scenario against the Hobgoblin. Each LLM agent uses OpenAI GPT 4.1

After running the simulations we see that the most successful prompting strategy is now RCoT rather than CoT. Surprisingly None only had a few points lower hp on average than RCoT. We infer that RCoT is now the best prompting strategy because a lot of model providers now use “reasoning” which does a lot of thinking before it provides an answer.

3.2 Different Model Providers

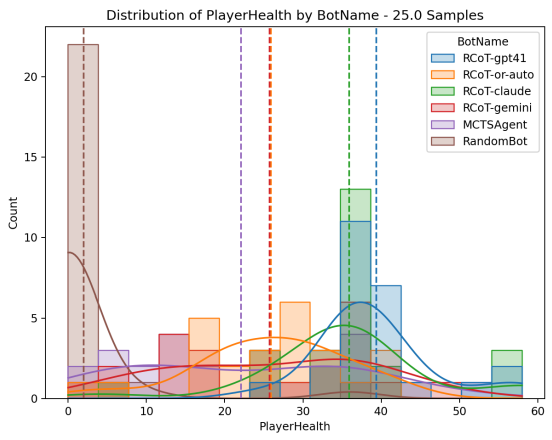


Figure 2: Each RCoT agent is a different model provider. This figure shows their average health across the starter deck scenario

In the original paper, the authors only used the OpenAI provided models. We wanted to expand on this by experimenting with different model providers via OpenRouter. The models we included were GPT 4.1, Claude Sonnet 4.5, OpenRouter auto, and Gemini 2.5, accompanied by baseline agents. GPT was the model that had the highest average health, to our surprise. Interestingly, OpenRouter auto claims to route requests to the best model provider but it got third place in our findings. We also experimented with free models, but the results were underwhelming.

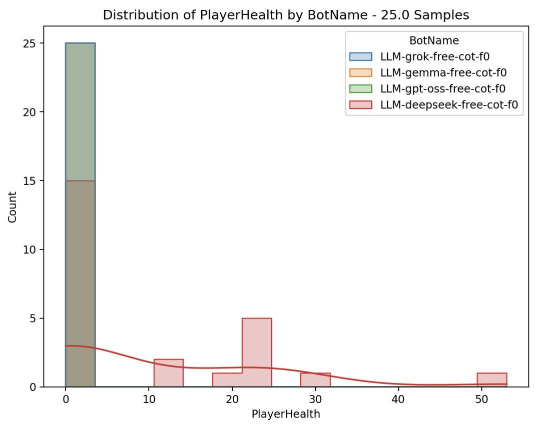


Figure 3: Free model providers

3.3 Tolerate Scenario

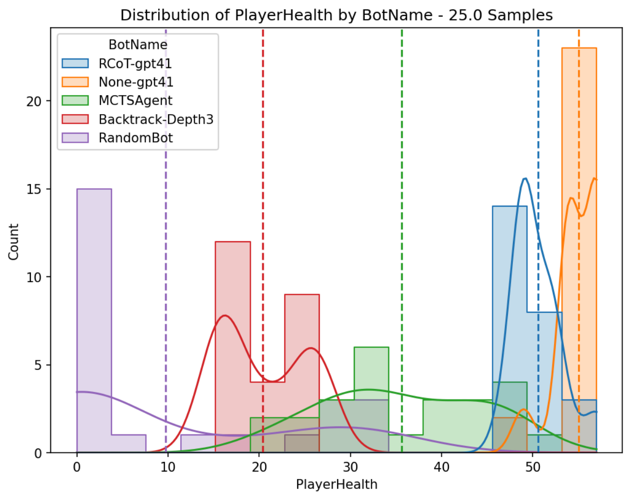


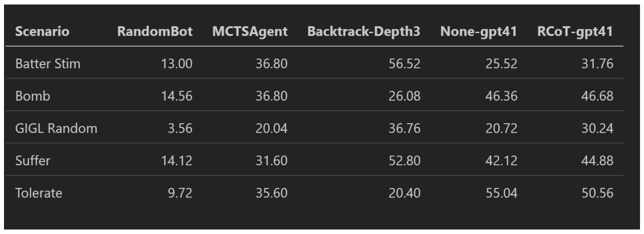
Figure 4: Avg player health in the tolerate scenario

Specifically, in the tolerate scenario, agents were required to play a high-cost power card that offered no immediate benefit but guaranteed long-term victory. This served as a test of strategic foresight.

MCTS Failure: The search-based agent struggled to see past the immediate health loss of playing the card, resulting in a lower average health of **35.6**. The simulation horizon was too short to realize the card's value.

RCoT Success: The LLM agent, utilizing RCoT, semantically understood the text "Gain Block every turn" as a winning condition. It consistently prioritized playing the card early, maintaining a superior average health of **50.6**. This demonstrates that LLMs can "read" strategies that are computationally expensive for search agents to "find."

3.4 Cross-Scenario Model Comparison



Scenario	RandomBot	MCTSAgent	Backtrack-Depth3	None-gpt41	RCoT-gpt41
Batter Stim	13.00	36.80	56.52	25.52	31.76
Bomb	14.56	36.80	26.08	46.36	46.68
GIGL Random	3.56	20.04	36.76	20.72	30.24
Suffer	14.12	31.60	52.80	42.12	44.88
Tolerate	9.72	35.60	20.40	55.04	50.56

Figure 2: Average Player Health for each agent in a given scenario. On our github in evaluation_results, you can find histograms for each scenario.

The most interesting scenario to draw from here is the GIGL Random scenario, which uses a random deck of generated cards from the pirate group. Surprisingly, RCoT performed better than every agent for every scenario except this one. Backtrack-Depth 3 resulted in the highest average health when testing LLM generated cards. This suggests that LLMs may not be the best agents at testing their own generated content

Conclusion

Our comprehensive evaluation highlights the distinct roles of different agent architectures in automated game testing. While simple heuristics (NoneAgent) are computationally efficient for regression testing of known content, they lack the adaptability required for evaluating new PCG mechanics.

The RCoT Agent emerges as the superior solution. By leveraging reverse Chain-of-Thought reasoning, it effectively "reads" new rules like a human player, offering strategic insights that pure search methods (MCTS) miss.

Our scenario experiments highlight the importance of monitoring your game playing agents and selecting the best agent for a given scenario. RCoT may have out-performed all other agents overall, but it still came short in specific instances.

References

Bahar Bateni and Jim Whitehead. 2024. Language-Driven Play: Large Language Models as Game-Playing Agents in Slay the Spire. In Proceedings of the 19th International Conference on the Foundations of Digital Games (FDG '24). Association for Computing Machinery, New York, NY, USA, Article 20, 1–10. https://doi.org/10.1145/3649921.3650013

Eger, M. 2025. Assignment 6: MCTS. CMPM 146 Course Materials, University of California, Santa Cruz. Accessed: December 9, 2025. https://yawgmoth.github.io/CMPM146/assignments/assignment6.html

Microsoft. 2025. How to use structured outputs with Azure OpenAI in Microsoft Foundry Models. Microsoft Learn. Accessed: December 9, 2025. https://learn.microsoft.com/en-us/azure/ai-foundry/openai/how-to/structured-outputs

OpenAI. 2025. Migrate to the Responses API. OpenAI Platform Documentation. Accessed: December 9, 2025. https://platform.openai.com/docs/guides/migrate-to-responses

OpenAI. 2024. Prompt Caching in the API. OpenAI Blog. Accessed: December 9, 2025. https://openai.com/index/api-prompt-caching/