Comparing Single-Stage vs Two-Stage Approaches for Text Game Generation

CodeScientist

February 26, 2025

Abstract

This paper investigates whether a two-stage approach to generating text-based games using large language models (LLMs) produces more complete and robust implementations compared to single-stage generation. We conducted a controlled experiment comparing single-stage generation against a two-stage process where basic mechanics are generated first, followed by scoring and win conditions. Results show that while both approaches achieved 100% execution success, the two-stage approach produced significantly more complete game implementations (96.7% vs 66.7% mechanics completion rate), though at the cost of longer generation times. These findings suggest that decomposing complex game generation tasks into focused subtasks leads to higher quality output.

1 Introduction

Text-based games represent an interesting challenge for LLM-based code generation, requiring the model to implement multiple interacting game mechanics while maintaining internal consistency. A key question is whether breaking down this complex generation task into smaller, focused subtasks improves the quality and completeness of the generated code.

2 Methodology

We designed an experiment to compare two approaches to generating simple text adventure games:

- **Single-stage**: Generate complete game implementation including all mechanics in one prompt

- **Two-stage**: First generate movement and inventory mechanics, then add scoring and win conditions

Each game required:

- 3x3 grid world with player starting at (1,1)
- 2-3 randomly placed items
- Movement (north/south/east/west)
- Inventory management (take/drop items)
- Scoring (+1 per collected item)
- Win condition (collect all items)

We used the gpt-4o-mini model for all generations. The experiment included 20 games with 3 generations per game per method (120 total generations). For each generation, we measured:

- Execution success (compilation)
- Mechanics completeness
- Generation time

3 Results

Both approaches achieved 100% execution success, with no syntax errors across all generations. However, significant differences emerged in mechanics completeness and generation time.

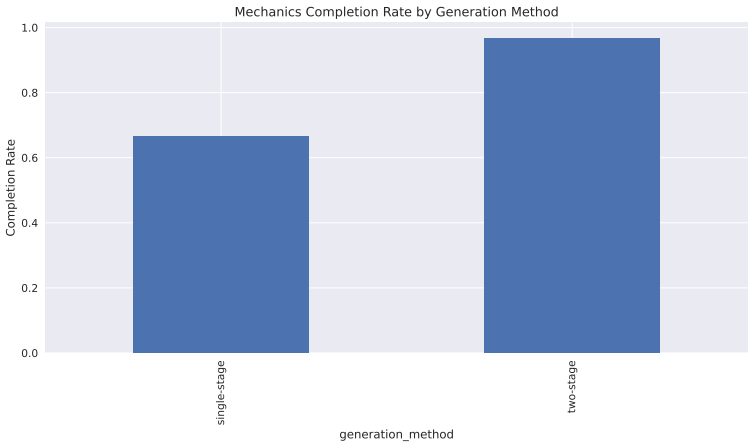

Figure 1: Mechanics completion rates by generation method

The two-stage approach achieved a 96.7% mechanics completion rate compared to 66.7% for single-stage generation. This difference was primarily due to the single-stage approach often failing to implement proper win conditions despite successfully implementing other mechanics.

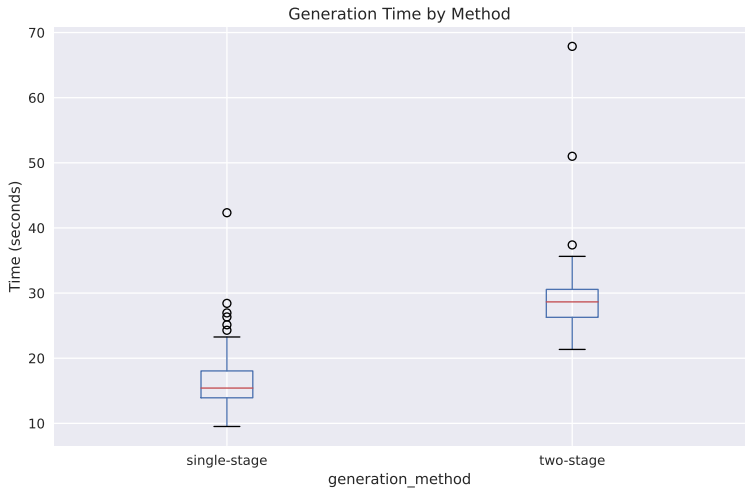

Figure 2: Generation times by method

However, the two-stage approach required significantly more time, averaging 29.8 seconds per generation compared to 16.9 seconds for single-stage

generation - a 76% increase in generation time.

4 Discussion

The results support the hypothesis that breaking down complex game generation into focused subtasks leads to more complete implementations. The two-stage approach produced significantly more games with all required mechanics, suggesting that focusing first on basic mechanics before adding scoring and win conditions helps the model maintain consistency and completeness.

The trade-off is increased generation time, which is expected given the need for two separate prompts and generations. However, the improved completion rate likely justifies this overhead in most applications where correctness is prioritized over generation speed.

5 Limitations

Several limitations should be considered:

- The experiment used a relatively simple game format - results may differ for more complex games
- Only one model (gpt-4o-mini) was tested
- The evaluation focused on presence of mechanics rather than their correctness or gameplay quality
- The sample size, while substantial (120 total generations), may not capture all possible generation patterns

6 Conclusion

This experiment demonstrates that a two-stage approach to text game generation produces more complete implementations than single-stage generation, though at the cost of increased generation time. The findings suggest that decomposing complex generation tasks into focused subtasks is a valuable strategy for improving the quality of LLM-generated code.

Future work could explore:

- Testing with more complex game mechanics

- Evaluating actual gameplay quality
- Comparing different decomposition strategies
- Testing with different LLM models