

# The AI Commander: An Exploratory Study of LLM-Unity Communication for Resource- Constrained Game Developers in 2D Top-Down Shooters

S3716152

Haoduo Zhuang(Alex)

Studio 4 (Prof Res Project)

MAGI S4 2025

# Abstract

Independent game developers face significant resource constraints in creating Non-Player Character (NPC) AI with tactical complexity and unpredictability. The limitations of traditional AI, such as the predictable patterns of behavior trees, curtail a game's replay value, while the direct integration of Large Language Models (LLMs) is hampered by prohibitive real-time performance, network latency, and API costs.

This paper documents an exploratory study into a potential architectural approach: establishing a low-cost communication pipeline between the Unity game engine and an external LLM (DeepSeek). Through a practice-based research methodology involving functional prototyping and systematic observation, this study reports on the design and preliminary performance of a hierarchical model where the LLM provides high-level tactical commands while traditional state machines handle real-time execution.

Key observations include: the communication pipeline exhibited basic technical functionality and notable cost-effectiveness, with 454 test API calls totaling only 0.67 CNY. Furthermore, preliminary tests in a simplified 9-grid environment suggest the model's capacity to facilitate emergent tactical behaviors that were neither explicitly scripted nor specified in the prompts.

This study does not present a validated solution but documents an exploratory process and its preliminary findings. It reports on observed phenomena that suggest potential pathways for independent developers interested in exploring low-cost LLM integration possibilities in 2D top-down shooter development.

## Keywords

AI Commander, Exploratory Study, LLM-Unity Communication, Hybrid AI Architecture, Hierarchical Architecture, Game AI, Indie Game Development, Prompt Engineering

# Introduction

Developing smart, adaptive NPC AI remains a significant challenge in game development, especially for independent creators operating under tight constraints of time, money, and specialized skills. Conventional AI systems, such as behavior trees and finite state machines, while powerful, often produce deterministic and predictable NPC behavior that can diminish a game's long-term replayability once players learn the patterns (Yannakakis & Togelius, 2018). This creates a persistent gap between a developer's design ambitions and their practical production capabilities.

The advent of Large Language Models (LLMs) presents a transformative opportunity, offering unprecedented capabilities in complex reasoning and planning (Gallotta et al., 2024). However, as initial industry explorations and our own prototyping experiments have shown, directly applying LLMs for real-time control of every NPC introduces significant challenges. These include high network latency from API calls, unsustainable operational costs due to per-token pricing, and a lack of precise behavioral controllability (Wang et al., 2024a). For most independent projects, direct, granular LLM integration is neither feasible nor realistic. This paper documents an exploratory study into an alternative approach, centered on the following research question: How can a communication pipeline between the Unity engine and an external LLM be established and operated in a low-cost manner for a 2D top-down shooter, and what are the initial behavioral phenomena observed?

This study involved the construction of a prototype, termed the "AI Commander," which employs a hierarchical architecture. An LLM serves as a high-level strategic reasoning layer, periodically analyzing an abstracted battlefield state to issue tactical commands. These commands are then received and executed by traditional, locally-run state machines. The goal of this exploration was not to produce a finished, validated system, but to investigate the practical feasibility of such a pipeline and report on its potential to facilitate more dynamic AI behaviors for resource-constrained developers.

## Background

Deterministic systems have historically predominated the field of game AI. The industry has generally accepted Finite State Machines and Behavior Trees as standards, providing developers with a structured method for scripting NPC logic. Although behaviors are reliable, their predictability is the main weakness. Yannakakis and Togelius (2018) argue that players today possess a deep understanding of game mechanics and are quick to spot and use repetitive patterns from scripted AI. This results in less challenge and reduces replayability, which is a huge problem, especially for indie developers that rely on highly engaging gameplay to compete. The development process for these systems is similarly demanding; it requires specialized skills to design, implement, and debug complex logic trees that can quickly become unwieldy and challenging to maintain as games grow in complexity.

The proliferation of LLMs indicated a revolutionary change, affording the prospect of authentically adaptive and intelligent NPCs possessing intricate reasoning and planning capabilities. However, initial endeavors to integrate directly soon brought to light three insurmountable impediments for developers facing resource limitations. Firstly, network latency: the round-trip time for an API call to an external LLM may extend from hundreds of milliseconds to multiple seconds, a delay that is untenable in a real-time game such as 2D top-down shooter where reactions of a split-second nature are paramount. Second, operational cost: LLMs are priced per token. If an NPC were to query the LLM at a high rate, the resulting expenses would make the game economically unviable. Third, controllability: LLMs are susceptible to producing irrelevant, inconsistent, or nonsensical output ("hallucinations"), which makes guaranteeing stable and reliable NPC behavior difficult without substantial validation and filtering mechanisms. These impediments have substantially restricted the capacity of most independent game creators to harness the transformative capabilities of LLMs.

# Literature Review

A substantial amount of academic and industry research is investigating methods for incorporating LLMs into game environments, indicating a notable preference for hybrid architectures to address the previously mentioned issues. The review herein examines the principal literature informing the AI Commander framework, concentrating on the limitations of traditional AI, the rise of hybrid models, and the specific technical methodologies that facilitate them.

Predictability is a key weakness in traditional game AI. The comprehensive survey of Behavior Trees presented by Iovino et al. (2022) underscores both their modularity and readability. However, it recognizes that deterministic logic may prevent unexpected behaviors. Exploring LLM integration is primarily motivated by the gap between scripted logic and emergent intelligence. The potential for LLMs to fill this gap is vast, as mapped by broad surveys from Wang et al. (2024) and Gallotta et al. (2024), which cover applications from procedural content generation to dynamic narrative systems. These studies indicate that the enhancement of NPC intelligence represents a significant area of exploration in the field of LLM research within the gaming industry.

Current advanced research indicates a preference for hierarchical or hybrid models. The RL-GPT framework offers a robust conceptual model, wherein the LLM serves as a "slow agent" for complex planning and a more streamlined model handles low-level execution (Liu et al., 2024). This "slow-fast" approach effectively resolves the latency problem through the reservation of the computationally demanding LLM for decisions of a non-time-critical nature. This principle has been corroborated within the complex real-time strategy (RTS) genre. TacticCraft effectively utilized an LLM for the translation of high-level natural language commands into precise in-game tactics for StarCraft II (Jeon et al., 2025), whereas SwarmBrain implemented an LLM as a central coordinator for multi-agent maneuvers within the same game (Zheng et al., 2024). These studies corroborate the capability of an LLM to function as a strategic overseer. A

significant difference pertains to the operational method; systems such as Tencent's PORTAL framework pre-compile strategies generated by LLMs into behavior trees offline (Xu et al., 2024), thereby attaining elevated performance levels at the expense of real-time adaptability.

Therefore, the present review highlights a discernible lacuna in the research: while various studies demonstrate the power of hierarchical AI or the use of offline compilation, there is a lack of transparently documented case studies on a real-time, online communication pipeline designed specifically to be accessible and cost-effective for the resource-constrained independent developer. This research aims to provide an initial exploration into that gap.

## Methodology: An Exploratory Study Through Prototyping

The architectural design of the prototype is guided by two key principles that structure this pipeline:

### **Hierarchical Separation: The Pipeline's Structure**

Hierarchical Separation is the guiding architectural principle. The system is divided into two distinct layers:

**The Strategic Reasoning Layer:** This is the domain of the LLM. It is responsible for tasks that require context, inference, and planning. It processes an abstracted, high-level representation of the game state and outputs a strategic intent. It operates asynchronously and is not bound by the real-time constraints of the game loop.

**The Tactical Execution Layer:** This layer is composed of traditional, locally-run AI components, such as state machines. Its sole responsibility is to receive concrete commands and execute them reliably and immediately. It is highly performant and deterministic.

This separation is essential for addressing the fundamental difficulties associated with LLMs. The framework significantly curtails API calls by invoking the strategic layer periodically (e.g., every 5 seconds), thereby diminishing both latency impact and expenditure.

### **The Command Pattern: The Pipeline's Language**

The bridge between these two layers is implemented using the Command Pattern (Nystrom, 2014). This design pattern is crucial as it defines the "language" of the pipeline, decoupling the strategic request (from the LLM) from its tactical implementation (by the NPCs). In the AI Commander framework:

The LLM takes on the roles of the Client and Invoker. It determines the command for execution and instantiates a Command object.

The AI Command class (Figure 1) is the Command object itself. It is a data-centric object that encapsulates all information required for an action (e.g., `command_type: "move", unit_id: 3, target_zone: 5`). The LLM will provide only this structured JSON object as output.

The Command Processor service receives this Command object.

Individual NPC agents and their respective state machines serve as the Receivers, namely the objects that enact the action.

This pattern provides a significant degree of flexibility. The strategic layer need not possess any knowledge of the execution of a "move" command; This decouples the high-level reasoning from the low-level implementation, making the system highly modular and extensible.

## Core AI Commander Structure

AICommandModels

Code block

```
1  public class AICommand { public string command_type;
    public int target_id; public Dictionary<string,
    object> parameters; /* Or specific parameter
    classes */ }
2  public class CommandParameterBase { }
3  public class RenameCommandParams :
    CommandParameterBase { public string new_name; }
4  public class MoveCommandParams : CommandParameterBase
    { public Vector3 position; }
5  (Add other parameter classes as needed)
6  public class EnemyRuntimeData { public int id; public
    string currentName; public GameObject
    enemyGameObject; /* Other relevant runtime data */
    }
```

*Figure 1: The AICommand data structure, a key component of the Command Pattern. This object acts as the standardized 'language' within the communication pipeline, encapsulating a command request from the LLM into a structured format that the Unity execution layer can understand and process.*

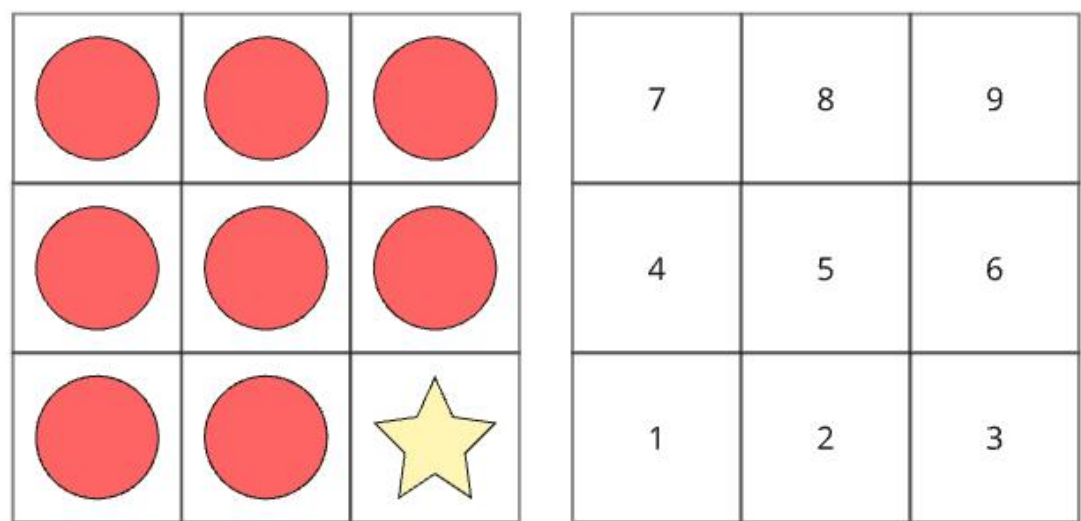


# Exploration and Observations

The exploratory process involved implementing the prototype and observing its performance within a controlled test environment. The observations focused on the pipeline's technical functionality, economic cost, and the behavioral characteristics that emerged.

## Evaluation Environment: The 9-Grid Testbed

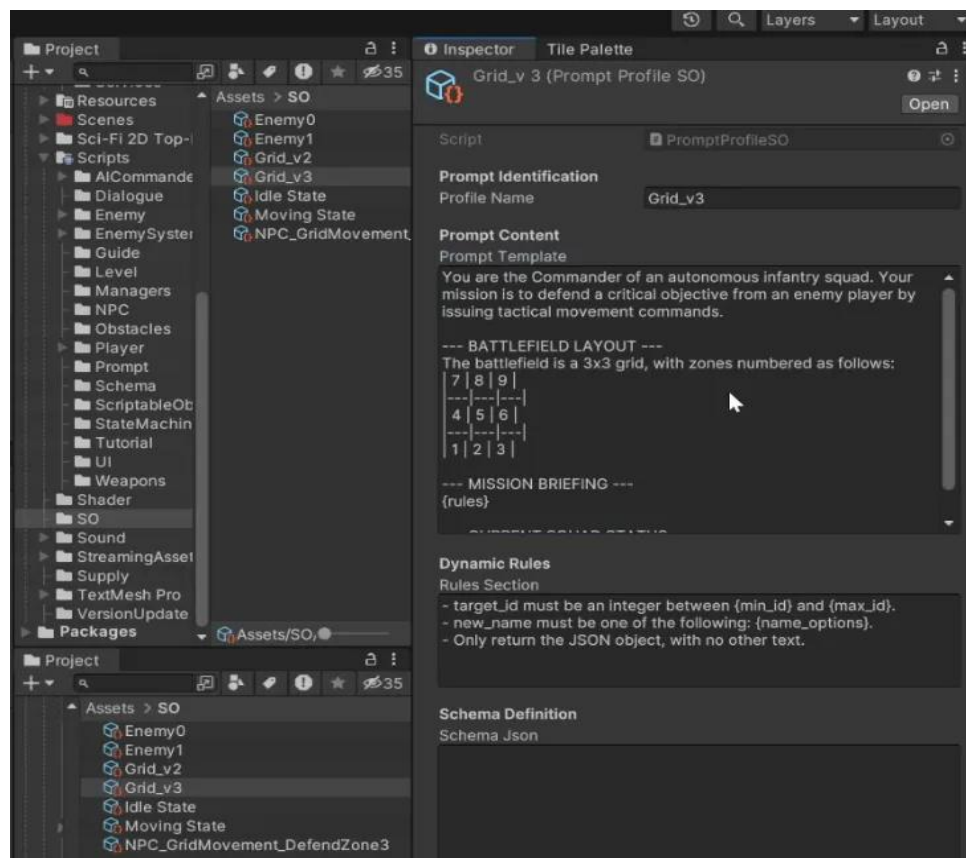
The framework's core logic was validated using a specifically designed 9-Grid Testbed (Figure 2). This environment simplifies the complexities inherent in a continuous, two-dimensional space by representing it as a discrete 3x3 grid. This abstraction is vital for enabling the LLM to more easily process spatial reasoning problems, facilitating a more targeted evaluation of the AI Commander's tactical decision-making capabilities. The evaluation protocol was designed to gather quantitative data, such as API latency and cost, as well as qualitative data concerning the nature and effectiveness of the generated commands.



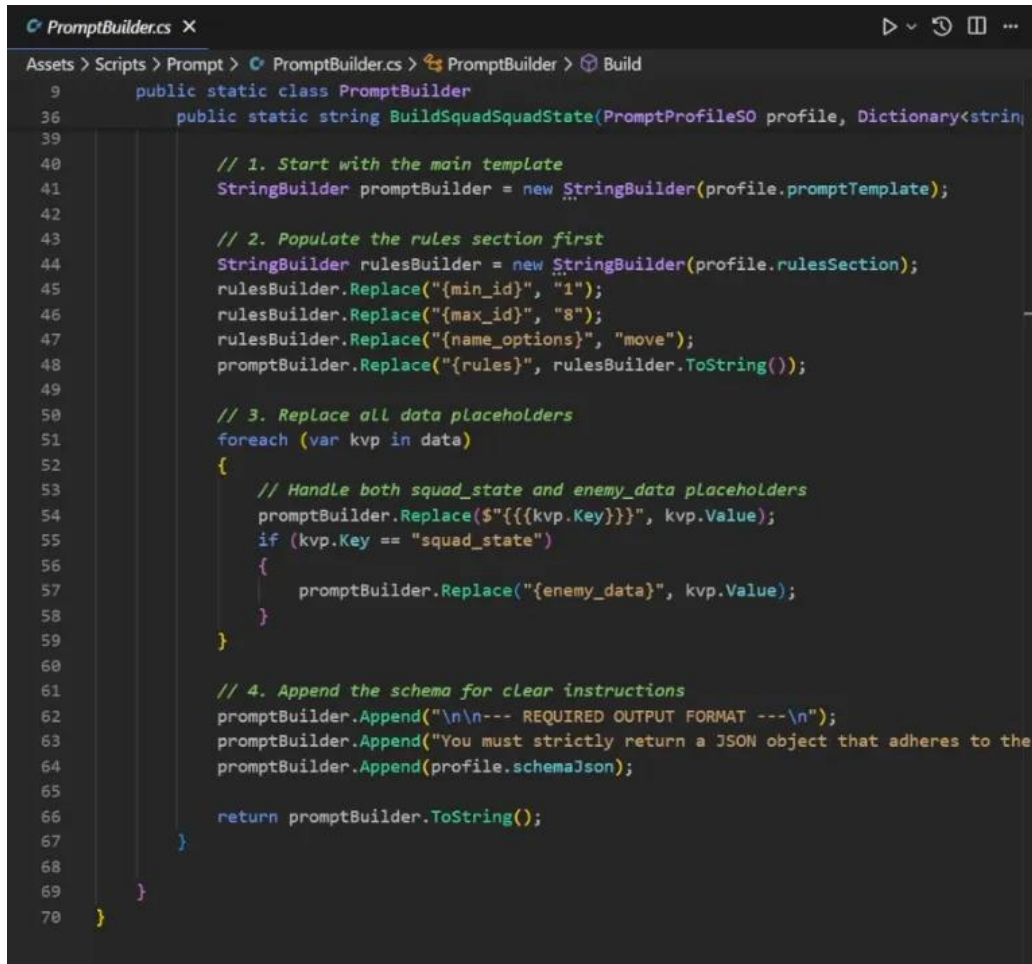
*Figure 2: Design of the 9-grid testbed. This environment discretizes the continuous game world into nine distinct zones, abstracting the complex problem of spatial reasoning into a structured, text-friendly format suitable for processing by the LLM.*

## Pipeline Implementation: Prompt Engineering

The pipeline is fundamentally based on its Prompt Engineering System. It leverages Unity's ScriptableObject to facilitate the generation of highly configurable prompt templates (Figure 3). Subsequently, a dedicated PromptBuilder class injects real-time battlefield data—for instance, squad status and enemy locations—into these templates (Figure 4). This modular methodology conceptualizes prompt construction as a type of "natural language programming," facilitating rapid iteration and experimentation with diverse strategic instructions for the LLM.



*Figure 3: The PromptProfileSO ScriptableObject in Unity. This component serves as a configurable template, defining the static structure and rules of the prompt sent to the LLM.*



```

9      public static class PromptBuilder
36      public static string BuildSquadSquadState(PromptProfileSO profile, Dictionary<string, string> data)
39
40      // 1. Start with the main template
41      StringBuilder promptBuilder = new StringBuilder(profile.promptTemplate);
42
43      // 2. Populate the rules section first
44      StringBuilder rulesBuilder = new StringBuilder(profile.rulesSection);
45      rulesBuilder.Replace("{min_id}", "1");
46      rulesBuilder.Replace("{max_id}", "8");
47      rulesBuilder.Replace("{name_options}", "move");
48      promptBuilder.Replace("{rules}", rulesBuilder.ToString());
49
50      // 3. Replace all data placeholders
51      foreach (var kvp in data)
52      {
53          // Handle both squad_state and enemy_data placeholders
54          promptBuilder.Replace($"{{{kvp.Key}}}", kvp.Value);
55          if (kvp.Key == "squad_state")
56          {
57              promptBuilder.Replace("{enemy_data}", kvp.Value);
58          }
59      }
60
61      // 4. Append the schema for clear instructions
62      promptBuilder.Append("\n\n--- REQUIRED OUTPUT FORMAT ---\n");
63      promptBuilder.Append("You must strictly return a JSON object that adheres to the");
64      promptBuilder.Append(profile.schemaJson);
65
66      return promptBuilder.ToString();
67  }
68
69  }
70  }

```

*Figure 4: The core logic of the PromptBuilder class. This class dynamically injects real-time game state data into the template from Figure 3, completing the 'natural language programming' step of the communication pipeline before sending the prompt to the API.*

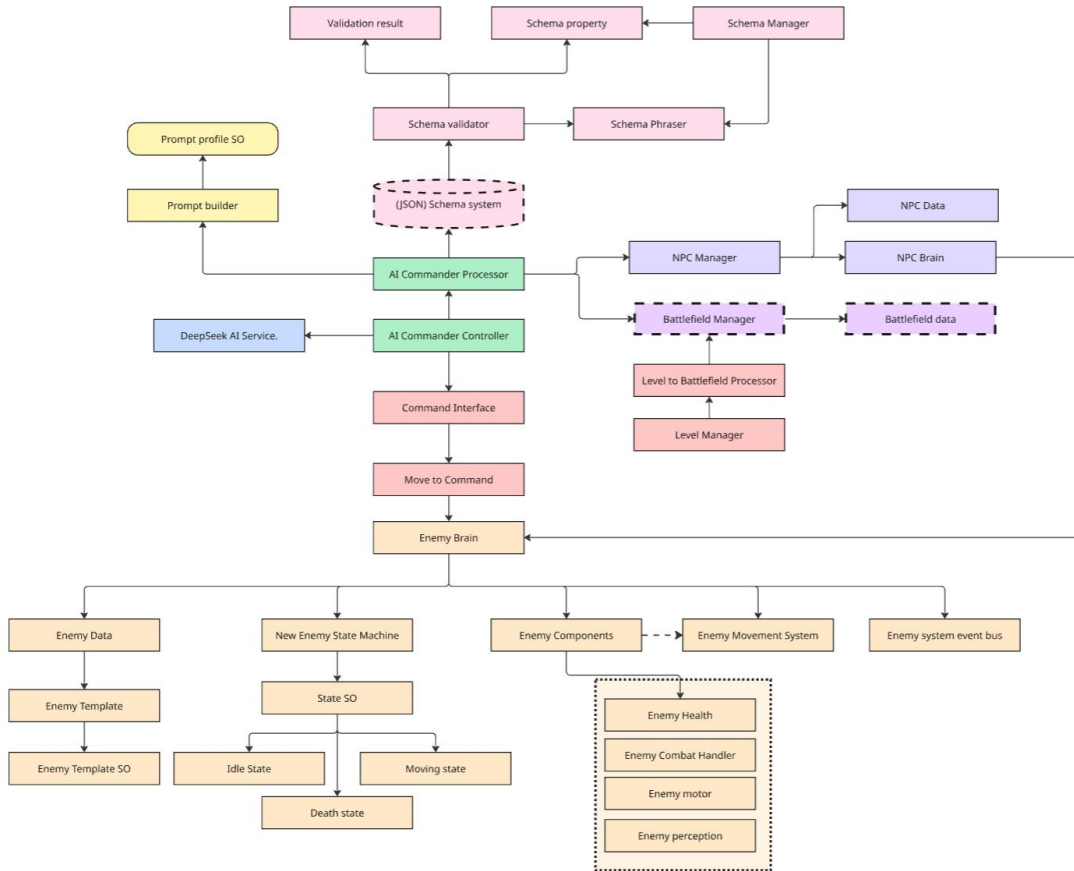
## Key Observations

This exploration yielded several key observations.

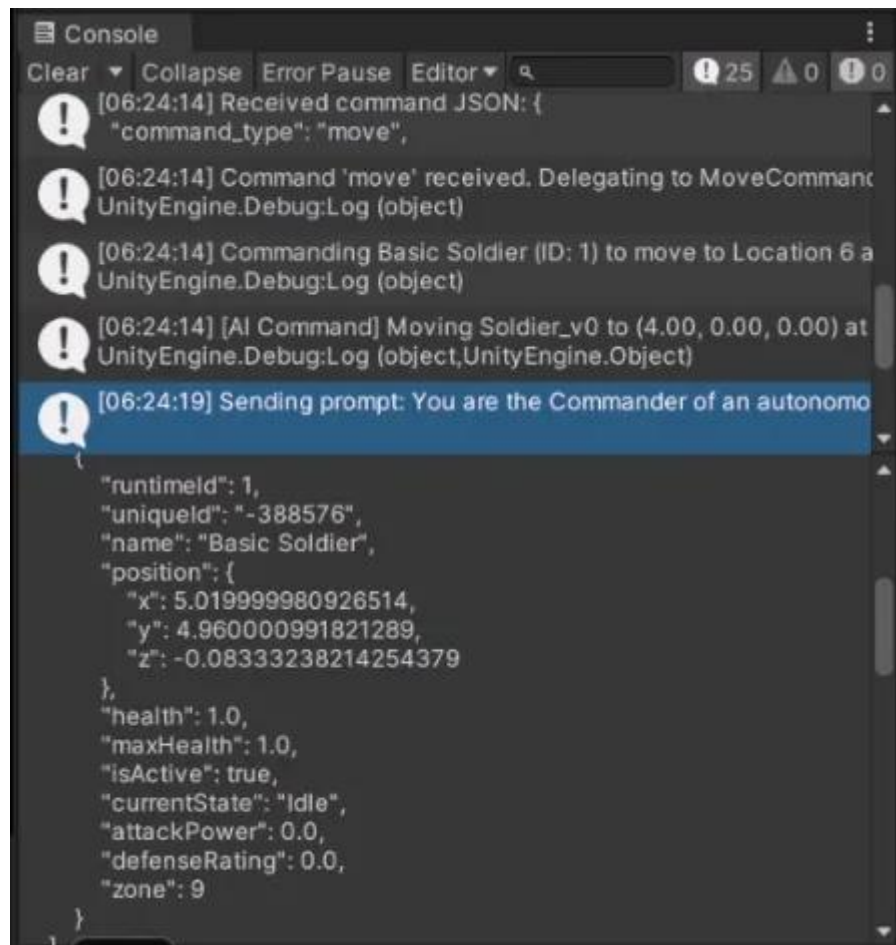
### Observation 1: The pipeline is technically functional.

The study confirmed the technical viability of the proposed LLM-Unity communication pipeline(Figure 5). The implemented prototype consistently executed its core operational loop: abstracting the real-time game state into a structured format(JSON), dispatching it to the external LLM via an API call, and

successfully receiving and parsing a structured JSON command in return. The entire process was observable through both console logs (Figure 6) and a purpose-built diagnostic UI (Figure 7).

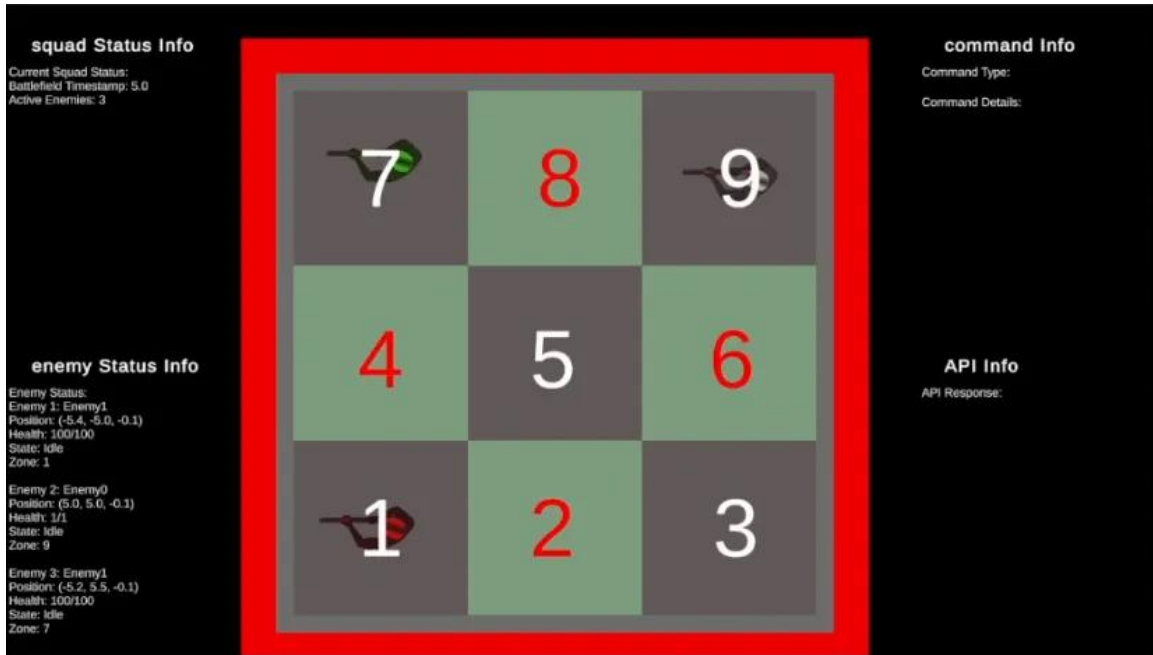


*Figure 5: A diagram of the complete communication pipeline architecture. It illustrates the cyclical data flow of the case study: from the Battlefield Manager capturing game state, through the Prompt Builder and DeepSeekAIService to the LLM, and the resulting AICommand flowing back through the AICommanderController to the individual EnemyBrain for execution.*



```
Console
Clear Collapse Error Pause Editor 25 0 0
[06:24:14] Received command JSON: {
  "command_type": "move",
}
[06:24:14] Command 'move' received. Delegating to MoveCommand
UnityEngine.Debug:Log (object)
[06:24:14] Commanding Basic Soldier (ID: 1) to move to Location 6 a
UnityEngine.Debug:Log (object)
[06:24:14] [AI Command] Moving Soldier_v0 to (4.00, 0.00, 0.00) at
UnityEngine.Debug:Log (object,UnityEngine.Object)
[06:24:19] Sending prompt: You are the Commander of an autonomo
{
  "runtimeId": 1,
  "uniqueId": "-388576",
  "name": "Basic Soldier",
  "position": {
    "x": 5.019999980926514,
    "y": 4.960000991821289,
    "z": -0.08333238214254379
  },
  "health": 1.0,
  "maxHealth": 1.0,
  "isActive": true,
  "currentState": "Idle",
  "attackPower": 0.0,
  "defenseRating": 0.0,
  "zone": 9
}
```

*Figure 6: A Unity console log demonstrating one complete cycle of the data flow: a prompt containing battlefield state is sent to the LLM, and the received JSON command is then delegated to the appropriate handler.*



*Figure 7: The custom in-game diagnostic UI. This tool provided real-time monitoring of the pipeline's state, including squad status, enemy info, and the latest API response, ensuring a transparent and repeatable evaluation process for the case study.*

**Observation 2: The operational cost is remarkably low.**

The entire testing period, which comprised 454 API calls and the processing of 431,106 tokens, incurred a total operational cost of only 0.67 CNY (Figure 10). This finding is critical as it demonstrates that leveraging an LLM for periodic, high-level strategic commands—rather than continuous, low-level control—is an economically viable approach for developers with limited budgets.

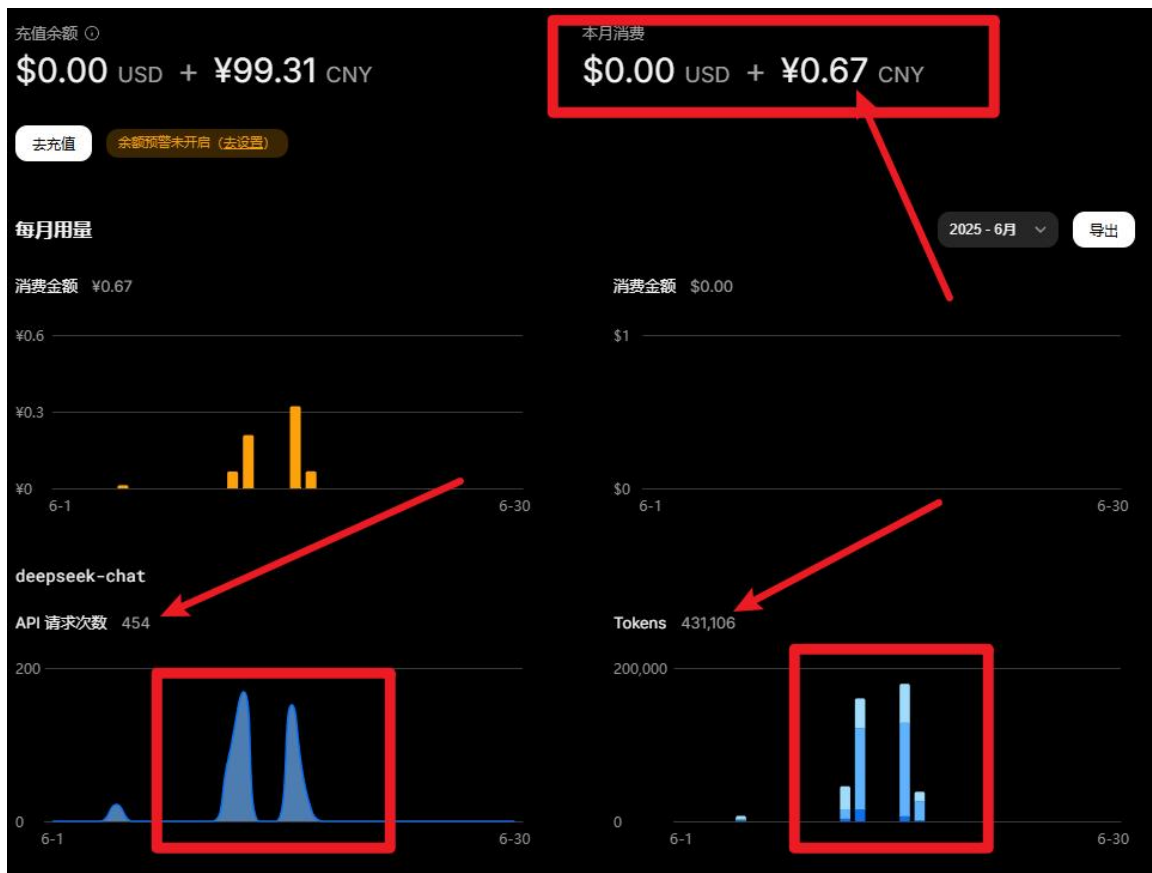


Figure 8: The DeepSeek API usage dashboard for the testing period, showing a total of 454 API calls with an accumulated cost of only 0.67 CNY.

### Observation 3: Unpredictability&Unscripted behaviors

The most intriguing finding is the LLM's ability to generate behaviors not explicitly defined in the prompt rules. While the prototype exhibits predictable logical responses—such as defending Area 3 (Figure 9) and encircling Area 7 (Figure 10)—it also repeatedly issues “Hold” commands (likely a conservative strategy when uncertain about the player's precise location). While the game's execution layer supported these commands, they were neither pre-programmed into the prototype nor explicitly specified in the prompt. This demonstrates that the AI commander system possesses fundamental contextual reasoning capabilities beyond its explicit instructions, holding potential to generate unpredictable AI behaviors for players.



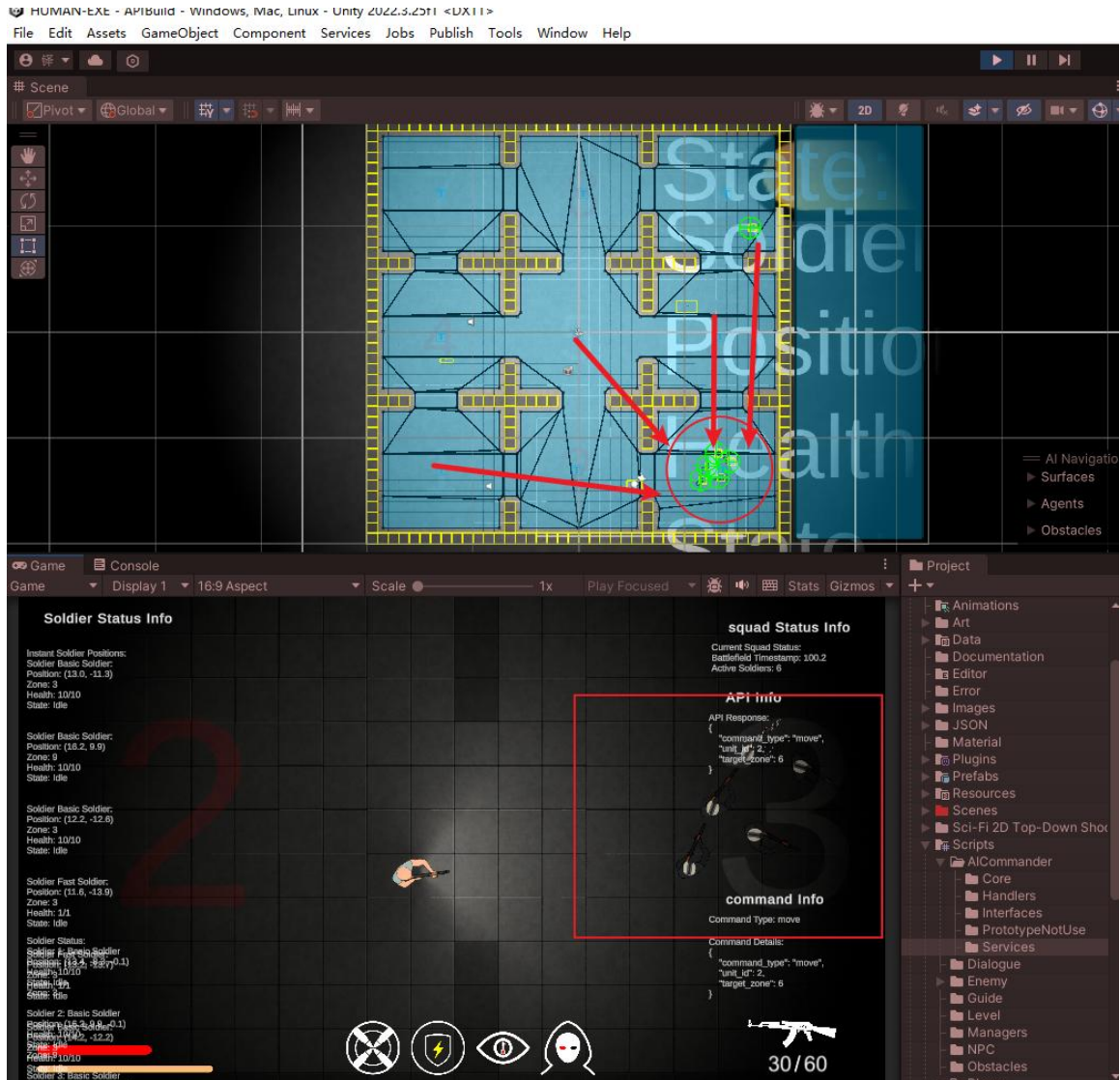


Figure 9: An instance of observed context-aware defense. Following the loss of a soldier while the player's location was unknown, the LLM generated a command to consolidate soldiers around the Area3, a logical but non-scripted conservative tactic.



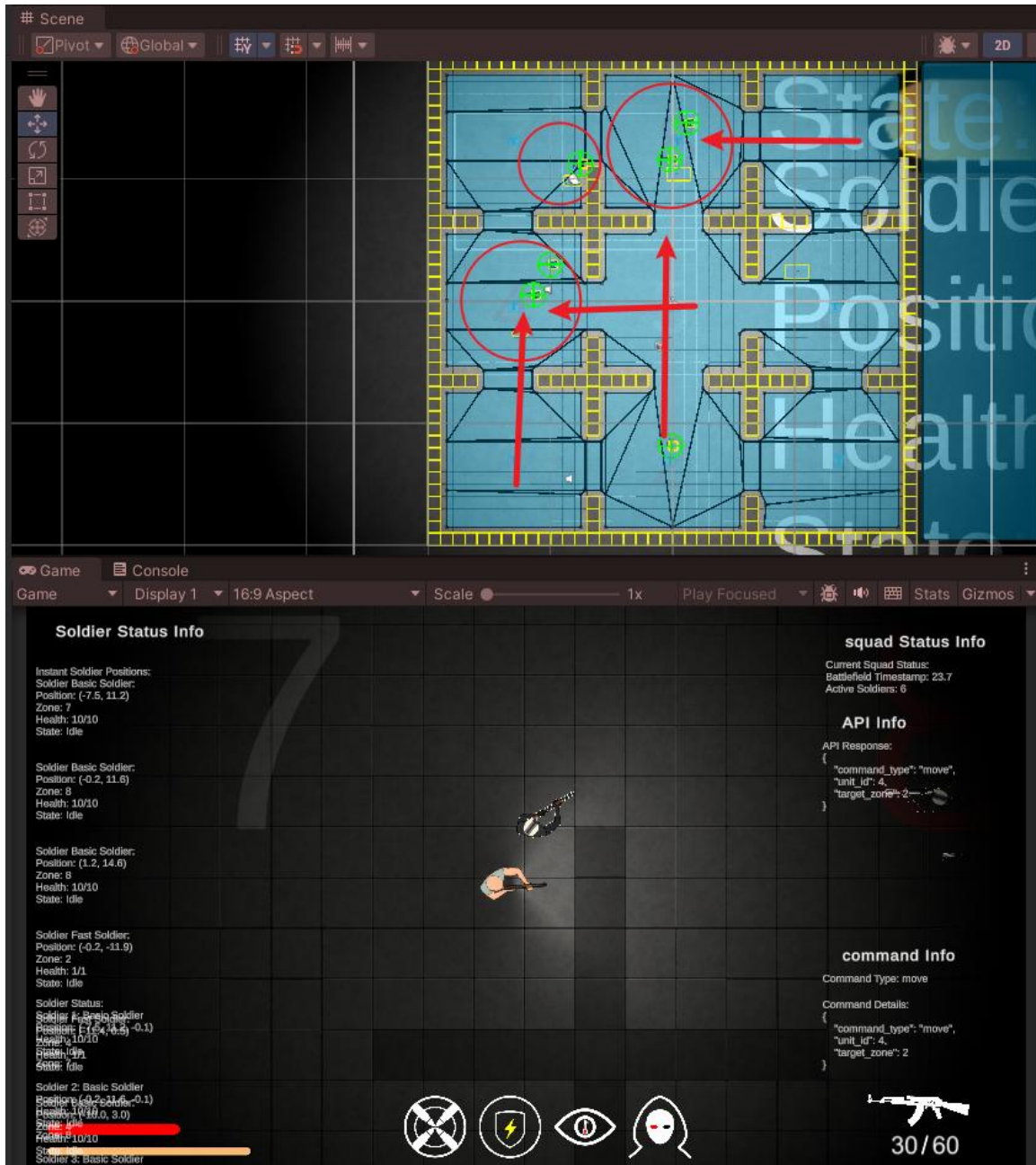


Figure 10: Observed encircling tactics executed by the LLM. When the AI commander system infers the player's position, it generates a "Move" command for each soldier unit, instructing them to execute an encircling tactic on Area 7.

## 8. Discussion

The observations from this exploratory study suggest that establishing a low-cost communication pipeline between Unity and an LLM is a promising approach for resource-constrained developers. The core architectural model appears technically sound and, most importantly, economically accessible.

The potential for emergent behavior, exemplified by the unsolicited "hold" command, is the most significant conceptual takeaway. It points towards a paradigm where developers design not just AI behaviors, but AI systems capable of generating their own behaviors. This directly addresses the weakness of predictability in traditional scripted AI. This study, therefore, serves as an initial data point suggesting that intelligent architecture, rather than raw computational power or complex scripting, might be a more effective instrument for indie developers seeking to innovate in AI design.

### **Limitations and Future Directions**

It is crucial to frame these observations within the study's significant limitations. This was an exploration, not a conclusive validation. It is acknowledged that the prototype's implementation was, at times, imperfect and encountered significant technical challenges, a common reality in exploratory prototyping. Furthermore, the test environment was highly simplified.

A critical limitation is the absence of a direct comparative analysis against a traditionally scripted AI. Quantifying the "unpredictability" or "intelligence" of the observed behaviors relative to a non-LLM baseline was beyond the scope of this study but remains the most vital next step for future research. Further work is also required to test the pipeline's robustness and latency under the stress of a more complex, continuous game environment and at a larger scale.

## 9. Conclusion

This paper has documented an exploratory study of the "AI Commander," a conceptual model for LLM-Unity communication tailored for resource-constrained developers. Through the process of building and observing a prototype, this study has reported on the practical feasibility of a low-cost, hierarchical communication pipeline.

The key observations —technical functionality, low operational cost, and the emergence of unscripted behaviors —suggest that this architectural pattern holds potential. It contributes not a definitive solution, but a transparently documented starting point and a set of encouraging preliminary data. For independent developers, the journey towards truly dynamic game AI may not require massive computational resources, but rather, clever and accessible architectures that facilitate a creative dialogue between game engines and the emergent power of Large Language Models.

## 10. Reference List:

1. Wang, L, Zhang, X, Lee, K, Yang, T, Liu, C, La, M, Yu, H & Cui, L (2024), *A Survey on Large Language Model-Based Game Agents*, arXiv preprint arXiv:2402.01589.  
Available at: <https://arxiv.org/abs/2404.02039>
2. Yannakakis, GN & Togelius, J (2018), *Artificial Intelligence and Games*, Springer, New York, NY.  
Available at: <https://www.gameaibook.org/book.pdf>
3. Gallotta, R, Todd, G, Zammit, M, Earle, S, Liapis, A, Togelius, J & Yannakakis, GN (2024), *Large Language Models and Games: A Survey and Roadmap*, arXiv preprint arXiv:2402.18659.  
Available at: <https://arxiv.org/abs/2402.18659>

4. Iovino, M, Scukins, E, Styrud, J, Ögren, P & Smith, C (2022), 'A survey of Behavior Trees in robotics and AI', *Robotics and Autonomous Systems*, vol. 154, p. 104096.  
doi:<https://www.sciencedirect.com/science/article/pii/S0921889022000513>
5. Liu, S, Yuan, H, Hu, M, Li, Y, Chen, Y, Liu, S, Lu, Z & Jia, J (2024), *RL-GPT: Integrating Reinforcement Learning and Code-as-policy*, arXiv preprint arXiv:2402.19299. Available at: <https://arxiv.org/abs/2402.19299>
6. Jeon, S, Kim, S, Oh, S, Kim, S, Kim, J, Lee, J & Kim, B (2024), *TacticCraft: Natural Language-Driven Tactical Adaptation for StarCraft II*, arXiv preprint arXiv:2507.15618. Available at: <https://arxiv.org/abs/2507.15618>
7. Zheng, H, Zhou, B, Yu, K, Yao, Y, Chen, Z, Wang, S, Wu, Y, Tang, Z & Chen, D (2024), *SwarmBrain: Embodied agent for real-time strategy game StarCraft II via large language models*, arXiv preprint arXiv:2401.17749. Available at: <https://arxiv.org/abs/2401.17749>
8. Xu, Z, Wang, X, Li, S, Yu, T, Wang, L, Fu, Q & Yang, W (2024), *Agents Play Thousands of 3D Video Games*, arXiv preprint arXiv:2503.13356. Available at: <https://arxiv.org/abs/2503.13356>
9. Nystrom, R (2014), *Game Programming Patterns*, Genever Benning. Available at: <https://gameprogrammingpatterns.com/>.
10. Wang, J, Wang, K, Lin, S, Wu, R, Xu, B, Jiang, L, Zhao, S, Zhu, R, Liu, H, Hu, Z, Fan, Z, Li, L, Lyu, T & Fan, C (2025), *Digital Player: Evaluating Large Language Models based Human-like Agent in Games*, arXiv preprint arXiv:2502.20807. Available at: <https://arxiv.org/abs/2502.20807>
11. Park, JS, O'Brien, JC, Cai, CJ, Morris, MR, Liang, P & Bernstein, MS (2023), 'Generative Agents: Interactive Simulacra of Human Behavior', *UIST '23: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1-22.  
doi: <https://dl.acm.org/doi/10.1145/3586183.3606763>
12. Sun, Z, Qiu, L, Wang, H, Zhu, W, Chen, W, & Yu, Y (2025), *AVA: Attentive VLM Agent for Mastering StarCraft II*, arXiv preprint arXiv:2503.05383. Available at: <https://arxiv.org/abs/2503.05383>

13. Todd, G, Earle, S, Nasir, MU, Green, MC & Togelius, J (2023), '*Level Generation Through Large Language Models*', arXiv preprint arXiv Available at: [:https://arxiv.org/abs/2302.05817](https://arxiv.org/abs/2302.05817)
14. He, K, Zhang, S, Shu, Y, Yang, Z, Zhang, A, Yu, B & Wang, L (2023), *Large Language Models Play StarCraft II: Benchmarks and A Chain of Summarization Approach*, arXiv preprint arXiv:2312.11865. Available at: <https://arxiv.org/abs/2312.11865>
15. Sawayama, Y, Morimoto, S & Nishino, J (2024), *Swarm-based real-time tactical planning for StarCraft II with Large Language Models*, arXiv preprint arXiv:2407.01859. Available at: <https://arxiv.org/abs/2401.17749>
16. del-Moral-Sanz, P, Fernandez-Esteche, M, Conde-Sousa, E, Fernandez-Blanco, E & Fdez-Riverola, F (2024), *A New Approach to Solving SMAC Task: Generating Decision Tree Code from Large Language Models*, arXiv preprint arXiv:2401.16024. Available at: <https://arxiv.org/html/2410.16024v1>
17. Wang, L, Zhang, X & Lee, K (2025), *Large Language Model Strategic Reasoning Evaluation through Behavioral Game Theory*, arXiv preprint arXiv:2502.20432. Available at: <https://arxiv.org/abs/2502.20432>
18. Chen, H & Liu, Y (2023), '*Can Large Language Models Play Text Games Well? Current State-of-the-Art and Open Questions*', arXiv preprint arXiv:2304.02868. Available at: <https://arxiv.org/abs/2304.02868>
19. Zhou, Z, Zhou, K, & Wang, Z (2024), *A Survey on Large Language Model based Autonomous Agents*, arXiv preprint arXiv:2308.11432. Available at: <https://arxiv.org/abs/2308.11432>
20. Zhang, Y, Gao, Y, Li, Z, Liu, B, Wang, Z, & Wang, Z (2024), *Atari-GPT: Benchmarking Multimodal Large Language Models as Low-Level Policies in Atari Games*, arXiv preprint arXiv:2408.15950. Available at: <https://arxiv.org/abs/2408.15950>
21. Wu, Y, Cai, S, Liu, Z, Chen, Z, Zhang, Y, Chen, H, Li, J, & Wang, L (2024), *BattleAgent: Multi-modal Dynamic Emulation on Historical Battles*

- to Complement Historical Analysis*, arXiv preprint arXiv:2404.15532.  
Available at: <https://arxiv.org/abs/2404.15532>
22. Zhu, D, He, Z, Wang, Z, Yang, B, Zhang, J, Liu, Z, Li, J & Zhang, M (2024), *Imgame-Bench: How Good are LLMs at Playing Games?*, arXiv preprint arXiv:2505.15146. Available at: <https://arxiv.org/abs/2505.15146>
23. Symflower Team (2024), 'LLM cost management: how to reduce LLM spending', *Symflower* Available at: <https://symflower.com/en/company/blog/2024/managing-llm-costs/>
24. AI Multiple Research Team (2024), 'LLM Pricing: Top 15+ Providers Compared in 2024', *AI Multiple*, blog post, viewed 20 August 2025, Available at: <https://research.aimultiple.com/llm-pricing/>
25. Walker, J (2024), 'How I Built an LLM-Based Game from Scratch', Available at: <https://towardsdatascience.com/how-i-built-an-llm-based-game-from-scratch-86ac55ec7a10/>
26. NVIDIA Technical Blog (2023), 'An Easy Introduction to LLM Reasoning, AI Agents, and Test Time Scaling', *NVIDIA Technical Blog*, blog post, viewed 20 August 2025, Available at: <https://developer.nvidia.com/blog/an-easy-introduction-to-llm-reasoning-ai-agents-and-test-time-scaling/>
27. Meta AI, FAIR, CMU, & UCSD (2022), 'CICERO: An AI agent that negotiates, persuades, and cooperates with people', Available at: <https://ai.meta.com/blog/cicero-ai-negotiates-persuades-and-cooperates-with-people/>