Proceedings of CONF-CDS 2025 Symposium: Data Visualization Methods for Evaluatio
DOI: 10.54254/2755-2721/2025.PO25155

Comparative Analysis of Different Generative Artificial Intelligence Tools in Game Code Development

Taihan Liu

International School, Beijing University of Posts and Telecommunications, Beijing, China
liutaihan@bupt.cn

Abstract. Generative Artificial Intelligence (GenAI) is revolutionizing procedural content generation (PCG) in computer games by enabling automated creation of diversified, adaptive, and complex game elements . The prospects and challenges of GenAI in video games have emerged as significant digital discourse topics, encompassing both cutting-edge technological applications and broad impacts on popular culture.This study conducts a comparative analysis of application differences among GenAI models in game code development. Addressing the research gap concerning free mainstream large models (Grok3, DeepSeek, Doubao, O3-mini, ChatGPT-3.5), we establish a multidimensional evaluation framework to reveal performance disparities in development efficiency, logical consistency, and creative support. Empirical results demonstrate substantial variations in code generation completeness across models under identical prompts (with Grok3 achieving optimal performance and GPT-3.5 yielding suboptimal outcomes).The core finding reveals that natural language interaction empowers non-programmers to create games without coding expertise, thereby democratizing game development. This breakthrough provides theoretical and technical pathways for human-AI collaboration paradigms and lowers barriers to digital content creation.

Keywords: Gen AI, Generative artificial intelligence, Artificial Intelligence, Computer Games, Comparison of LLM

1. Introduction

After two decades of development abroad, the game industry has grown into a high-tech, profitable, and quickly increasing business. In China, online gaming, in particular, represent an expanding profit development area within the business [1]. The production of games happens to be one of the industries that is most affected by GenAI [2].

Game creation has been greatly impacted by the rapid evolution of generative AI in recent years. Goodfellow et al. introduced Generative Adversarial Networks (GANs) in 2014 [3]. The basis for image generation is laid by GANs, which are made up of a discriminator and a generator that train antagonistically to create high-quality images. Variational Autoencoders (VAEs) were then proposed in 2013 [4]. VAEs were extensively used in image generation, data dimensionality reduction, and anomaly detection after surpassing the limits of conventional autoencoders in producing fresh data by integrating probabilistic models and variational inference. The Transformer architecture was first presented by Vaswani et al. in 2017 [5]. Because of its ability to handle long-sequence data well, its self-attention mechanism has advanced natural language processing and is now a fundamental part of many generative AI models. Diffusion model theory was expanded upon by Ho et al. in 2020 [6], improving the caliber and variety of images produced and offering a fresh development in generative AI.

Large language models (LLMs) and other generative AI models have advanced significantly in recent years. These models can produce text, including programming code, and include OpenAI's GPT series, Google's Gemini, and Meta's LLaMA [7,8]. Python is becoming the main language for AI-driven code generation in fields including web development, data analysis, and machine learning due to its readability and simplicity [9,10].

With the breakthrough development of AI technology, its application in the game industry is gradually becoming the core driving force to promote industry reform. Generative artificial intelligence (GenAI) has a particularly profound impact on game programming, and is reshaping the way code is created and optimized. It can automate code generation itself, and also assist programmers in intelligent code completion and give optimization suggestions. When building complex game systems, GenAI can help generate code structures or pseudo codes of core logic according to high-level design specifications, providing programmers with a high-level implementation blueprint. At the same time, GenAI also plays an important role in other aspects of game development: in terms of planning, GenAI can assist in text generation and editing; In the art field, GenAI can be applied to the automatic generation of audio/video and images; In the testing phase, GenAI can promote the automated testing process

This paper aims to study the assistance provided by GenAI tools in game code development. By employing multi-dimensional evaluation metrics to score different models, it explores their differences. The goal is to provide reference for others seeking to use GenAI to aid code development and to offer non-technical individuals an approach to realize their game design ideas.

The study primarily compares the following free models: grok3, deepseek, Doubao, O3mini, and ChatGPT3.5. Each model was given identical prompts (provided in both Chinese and English to mitigate language bias) to generate code, which was then evaluated based on actual runtime performance. The AI was tasked with generating code for two games: the relatively simple classic Snake Game and the more complex board game Chinese Chess.

2. Experiment introduction

2.1. Model introduction

The primary models discussed in this paper comprise the following: Grok3, a model which specializes in efficient code generation and logical reasoning that delivers exceptional workflow completeness in game development; DeepSeek, a model that excels in data structure processing and multi-turn interaction which is tailored for step-by-step guided programming scenarios; Doubao, a lightweight model whose robust interface generation capabilities are noteworthy yet whose logical consistency requires refinement; O3mini, a freely accessible model that balances performance while supporting innovative extensions which proves effective for prototyping complex systems; ChatGPT-3.5, OpenAI’s conversational model which demonstrates competent foundational code-generation abilities but exhibits stability limitations in advanced task execution; and Claude 3.5, a modular architecture-focused model that prioritizes code maintainability which operates under commercial licensing.

2.2. Method introduction

This study adopts a systematic five-phase methodology. First, functional specifications of the target game were rigorously defined and simultaneously established as evaluation criteria to assess the code completeness generated by GENAI systems. Subsequently, bilingual prompt templates (Chinese and English) were designed to standardize input conditions across models. These prompts were then iteratively fed into six GENAI platforms (Grok3, DeepSeek, etc.) to generate executable game code. All outputs underwent uniform compilation and testing within the CodeVS integrated development environment under controlled runtime conditions. Finally, comparative analysis of operational discrepancies – including error rates, feature realization fidelity, and code efficiency – was conducted to derive performance benchmarks and technical limitations across models.

3. Experimental process

3.1. Use iterative guided dialogues with a single GENAI model to generate the Snake Game code

The purpose of this experimental segment lies in investigating whether, as a complete novice to coding, one can successfully achieve the creation of functional game code through step-by-step interaction with the GENAI tool.

In software engineering, requirements analysis refers to all the work to be done when describing the purpose, scope, definition and function of a new system when building a new or changing an existing computer system. System requirements analysis is a very important link in the software life cycle. Good requirements analysis can clearly define the user's abstract requirements as the specific documents of software development, which is the cornerstone of early development. Only standardized requirements analysis documents can design good games [2].

The model employed in this segment is Deepseek. Initially, during the experiment, I openly admitted to Deepseek that I possessed no programming knowledge and expressed my hope that it could guide me on how to develop a Snake game solely with the aid of a compiler. Deepseek's response was thorough, presenting the code in modular form for output and addressing any inquiries about compiler usage promptly and comprehensively.

3.2. Use five GenAI models to generate the code of Snake Game according to the prompt words

The prompt words selected this time are to generate a greedy snake game. The interface includes the difficulty selection interface at the beginning of the game, the game running interface, the pause interface, and the death interface. Functions include control object movement, death detection, difficulty setting, etc. The prompt words used in this part are shown in Table 1:

Table 1. Chinese and English prompts for the Snake Game

Module	Submodule	Prompt(Chinese)	Prompt(English)
Game core mechanism	Snake movement	通过方向键控制移动方向（上/下/左/右）	control the movement direction (up/down/left/right) through the direction keys
	Food production	随机位置刷新食物，碰撞检测逻辑	refresh food at random position, collision detection logic
	Growth system	吃到食物后蛇身长度+1	snake length+1 after eating food
	Collision detection	边界碰撞与自碰检测触发游戏结束	boundary collision and self collision detection trigger the end of the game
Interface system	Start menu	显示难度选项（简单/普通/困难）和操作指引	display difficulty options (simple/ordinary/difficult) and operation instructions
	Pause interface	透明遮罩层覆盖，保留当前游戏画面	p key to enter the semi transparent mask layer coverage, and keep the current game screen
	End interface	显示最终得分，提供重启/退出选项	display the final score and provide restart/exit options
	Real time data display	屏幕左上角显示当前得分和难度	the current score and difficulty are displayed in the upper left corner of the screen

Difficulty system	Speed rating	简单(慢)
		普通(中)
		困难(快)
			simple(slow)
			ordinary (medium)
			difficult (fast)

	Score coefficient	不同难度下每个食物的基础得分不同	the basic score of each food is different under different difficulties
	Dynamic configuration	通过字典结构管理难度参数	manage difficulty parameters through dictionary structure
Other contents		开发一款具备完整游戏流程的贪吃蛇游戏，包含多级难度、交互界面和扩展功能 Python版：使用 Pygame 库实现图形渲染和事件处理	Please develop a snake game with complete game process, including multi-level difficulty, interactive interface and extension functions, provide complete code, and write clear comments after each line of code Python version: use Pygame library to realize graphics rendering and event processing You can add innovative functions and tell me your innovation points

Among them, the English version is robust on the basis of the Chinese version, while emphasizing the realization of the pause interface, and giving GenAI the right to add innovative functions on its own.

3.3. Use five GenAI models to generate the code of Chinese Chess Game according to the prompt words

The prompt in this part is to generate a small chess game that can support human-computer and two person combat according to Chinese chess rules, and add background music and special fonts. The prompt words in this part are as shown in Table 2:

Table 2. Chinese and English prompts for the Chinese Chess Game

Module	Submodule	Prompt(Chinese)	Prompt(English)
Other contents		我想让你根据我上传的文件里中国象棋的规则帮我开发一个中国象棋Python游戏， 现在请你用python写代码完成游戏功能的实现，输出完整代码，并在每一行代码后面写清楚注释功能要求如下：	I want you to help me develop a Chinese chess Python game according to the rules of Chinese chess in the file I uploaded, Now please write code in python to complete the implementation of the game function, output the complete code, and write clear comments after each line of code The functional requirements are as follows:
Startup interface	Title	中央显示行楷"中国象棋"标题	The title of "Chinese Chess" in running script is displayed in the center (Xingkai. ttf file is used for font)
	Mode selection	人机对战/双人对战按钮	There are two buttons below to enter the game: man-machine battle and two person battle
	Music control button	界面右上角一个开启关闭音乐的按钮(按钮字体小一些，显示文字为“曲”)	There is a button to open and close music in the upper right corner of the interface (the button font is smaller, and the display text is "Song")
	Background Music	背景音乐使用resources/bg_music.mp3	Use resources/bg_music.mp3 for background music
Chessboard interface	Chessboard drawing	根据中国象棋规则文件设计棋盘，由9纵线10横线构成，其中4-6列的两侧为3*3米字格	The chessboard is designed according to the Chinese chess rules file, which is composed of 9 vertical lines and 10 horizontal lines. There are 3 * 3m character grids on both sides of 4-6 columns.
	Chessboard color	棋盘颜色可以由你自行发挥	The color of the chessboard can be played by you.
	Chess font	棋子字体用行楷	The chessboard font is Xingkai (Chinese Semi-Cursive-Regular Style)
	Chess layout	红方在下半，黑方在上半	the red side in the lower half and the black side in the upper half
Human-vs-Machine Mode	Difficulty selection interface	点击后弹出初级/中级/大师选项	Click to pop up the difficulty selection interface (elementary/intermediate/master)

	Game Panel	棋盘和棋子在屏幕中间 右侧有切换难度，退出，悔棋三个按钮 界面右上角一个开启关闭音乐的按钮保持不变	Chessboard and pieces are in the middle of the screen There are three buttons on the right-- Difficulty switch， Exit and Regret. A button for turning on and off music in the upper right corner of the interface remains unchanged
	Difficulty switch button	点击切换难度会进入难度选择界面，选择难度之后保持和之前对局一样的进度。	Click the switch difficulty to enter the difficulty selection interface, and keep the same progress as the previous game after selecting the difficulty.
	Exit button	点击退出显示三个按钮：退出至主界面，退出游戏，取消。并实现对应的功能。	Click Exit to display three buttons: Exit to the main interface, exit the game, and cancel. And realize the corresponding functions.
	Regret button	点击悔棋按钮会复原电脑走的一步和玩家走的一步棋，支持多次悔棋。	Click the repentance button to restore the computer's move and the player's move. Multiple repentance is supported.
Human-vs-Human Mode	Game Panel	棋盘和棋子在屏幕中间 右侧有退出按钮，退出按钮上下侧均有悔棋按钮 界面右上角一个开启关闭音乐的按钮保持不变	Chessboard and pieces are in the middle of the screen There is an Exit button on the right side, and there are Regret buttons on the upper and lower sides of the Exit button A button for turning on and off music in the upper right corner of the interface remains unchanged
	Exit button	点击退出显示三个按钮：退出至主界面，退出游戏，取消。并实现对应的功能。	Click Exit to display three buttons: Exit to the main interface, exit the game, and cancel. And realize the corresponding functions.
	Regret button	悔棋上下两个按钮分别代表黑方悔棋和红方悔棋，点击悔棋按钮会复原刚落子玩家的一步棋。	The upper and lower buttons of repentance chess represent black repentance chess and red repentance chess respectively. Clicking repentance chess button will restore a move of the player who just finished playing.

4. Introduction to evaluation dimension

The scoring framework comprises five dimensions: Functional score, Code quality score, Performance score, Maintainability score, and Innovation score. The composite score is calculated as:

Functional score × 40% + Code quality score × 20% + Performance score × 15% + Maintainability score × 15% + Innovation score × 10%.

An example comparison table is provided below

4.1. Functional score

This part of the score mainly includes the integrity of the game interface, the basic functions of the game, the processing power of abnormal conditions, and the extended functions of the game. The integrity of the game interface is based on whether the game interface can be displayed normally as required, and the score of the game interface can be switched normally. The basic functions of the game are based on whether the basic functions of the game, such as winning or losing judgment, scoring judgment, etc., achieve scoring. The processing power of abnormal conditions is based on whether the game can handle abnormal conditions such as incorrect input and continuous input. The game extension function is based on whether the game can fully realize the preset scores of other systems.

4.2. Code quality score

This part of the score mainly includes EP8 normative, naming specification, architectural rationality, unit test coverage, and code extensibility.The EP8 specification scores based on the compliance of Python coding specifications such as checking code indentation, blank lines, and line length.The naming specification is based on verifying whether the naming of variables/functions/classes conforms to the principles of semantics and unity.The rationality of the architecture is divided according to the evaluation module, the clarity, the singleness of class responsibilities, and the control level of coupling are scored.The unit test coverage rate verifies the integrity score based on the measurement of the coverage ratio and boundary conditions of the key functional unit tests.Code extensibility is based on whether large-scale refactoring is required when checking for new functions, and the rationality score of the reserved interface

4.3. Performance score

This part of the score mainly includes frame rate stability, memory management ability, collision detection efficiency, and load time optimization ability.The frame rate stability is scored according to the frame rate fluctuation amplitude and the lowest frame retention ability in different scenarios of the test.Memory management capabilities are scored based on monitoring the risk of memory leaks and the effectiveness of resource loading/releasing strategies.The collision detection efficiency is scored based on the performance consumption of the verified physical system and the efficiency of multi-object collision processing.The loading time optimization ability is based on the evaluation of scene switching or resource loading time-consuming and asynchronous loading strategy rationality score

4.4. Maintainability score

The scoring in this part mainly includes the degree of modularity, the degree of modularity, and the quality of annotations.The degree of modularity is scored according to the degree of decoupling of the evaluation function and the possibility of module reuse.The degree of configuration concentration is based on the degree of unified management of the inspection parameter configuration and the convenience of modification score.Annotation quality scores based on measuring code annotation coverage, description accuracy, and timeliness of updates

4.5. Innovation score

The scoring in this part mainly includes unique algorithm design ability, interaction design ability, expansion mechanism ability, and artistic style innovation ability.The unique algorithm design ability is based on the evaluation of the innovation and performance of the core algorithm to optimize the breakthrough score.The interaction design ability is scored based on the originality of the human-computer interaction model and the improvement of the user experience.The ability of the expansion mechanism is scored according to the design of system extensibility and the possibility of MOD support.The ability of artistic style innovation is based on the verification of the differentiated expression of the game theme by the visual and sound design.

5. Result analysis

5.1. Iterative guidance of dialogue results using a single GENAI model

After nearly a hundred dialogues with an AI and extensive testing, the final code successfully implements all predefined game functionalities. Upon execution, the game displays a main interface with the title "Snake Game" centered at the top, followed by difficulty level options and a prompt instructing players to press the spacebar after selecting a difficulty. The snake's movement speed dynamically adjusts based on the chosen difficulty level.

During gameplay, the application runs smoothly without lag. All core functions operate as expected: The scoring system increments correctly; Game pauses immediately when pressing ESC; Resuming gameplay by pressing ESC again functions properly. The game-over condition triggers upon collision with walls or the snake's own body. The game-over screen displays the final score and provides options to quit (Q) or restart (R).

5.2. Running results of Snake Game code generated by different models

5.2.1. Comparison of running results of Greedy Snake Game code generated from Chinese prompt words

After comprehensive evaluations from various sources, the comparison table of code ratings generated based on the same Chinese prompt word is as follows(Table 3):

Table 3. Chinese prompt comparison table about Snake Game

GENAI Comparison Table						
Evaluation dimension	Functional score (40%)	Code quality score (20%)	Performance score (15%)	Maintainability score (15%)	Innovation score(10%)	Comprehensive score
deekseek	36	18	13	13	5	85
Doubao	28	12	10	10	7	67
chatGPT3.5	34	17	12	14	9	86
grok3	38	19	14	14	10	95
O3mini	32	16	11	12	8	79

For the game generated by the Deepseek model in this link, there are the following results. The text interface implementation was incomplete, with rendered fonts displaying primarily garbled characters interspersed with minimal English text. Core gameplay functionality was partially achieved: scoring through fruit consumption functioned correctly, but wall collision detection remained unimplemented. Other interface logic operated as expected. The solution demonstrated concise code volume, acceptable performance metrics, moderate maintainability, and no exceptional innovations.

For the game generated by the Doubao model in this link, there are the following results. Significant deficiencies were observed in text interface execution, with font rendering producing predominantly garbled output. Gameplay implementation exhibited critical flaws: an excessively large game map compromised control responsiveness. Unhandled edge cases persisted, notably directional input conflicts (e.g., right-key input during left movement causing erroneous direction changes) that triggered premature game termination. The solution featured substantial code volume, suboptimal performance, poor maintainability, and no distinguishing innovations.

For the game generated by the ChatGPT 3.5 model in this link, there are the following results. Minor text rendering imperfections were noted, with output displaying predominantly English characters alongside limited garbled text. Game functionality was substantially complete, fulfilling all prompt-specified requirements. The solution exhibited moderate code volume, standard performance benchmarks, adequate maintainability, and no significant innovations.

For the game generated by the Grok3 model in this link, there are the following results. Text interface execution was successful, with fonts rendering Simplified Chinese characters correctly. Game functionality demonstrated comprehensive implementation of prompt requirements. The solution featured minimal code volume, stable performance characteristics, good maintainability, though no groundbreaking innovations were present.

For the game generated by the O3mini model in this link, there are the following results. Slight text rendering deficiencies occurred, presenting primarily English output with intermittent garbled characters. Core gameplay functionality met essential requirements. The solution utilized moderate code volume, delivered standard performance, showed acceptable maintainability, and introduced no notable innovations.

The code implementations and visual presentations of ChatGPT 3.5 and O3mini exhibited striking similarities.

5.2.2. Comparison of running results of Greedy Snake Game code generated from English prompt words

After comprehensive evaluations from various sources, the comparison table of code ratings generated based on the same English prompt word is as follows(Table 4):

Table 4. English prompt comparison table about Snake Game

GENAI Comparison Table						
Evaluation dimension	Functional score (40%)	Code quality score (20%)	Performance score (15%)	Maintainability score (15%)	Innovation score(10%)	Comprehensi ve score
deekseek	26	10	8	13	3	60
Doubao	34	15	12	11	4	76
chatGPT3.5	10	10	5	5	0	30
grok3	38	19	14	14	5	90
O3mini	38	18	14	13	8	91

For the game generated by the Deepseek model in this link, there are the following results. While the text interface executed properly with correctly rendered English fonts, gameplay functionality remained incomplete. Critical failures included unresponsive snake movement controls and abnormal fruit relocation when unconsumed. The game automatically terminated after brief operation despite unimplemented end conditions. Other interface logic functioned as designed. This implementation exhibited moderate code volume, acceptable performance metrics, adequate maintainability, and no notable innovations.

For the game generated by the Doubao model in this link, there are the following results. Severe text interface deficiencies were observed, with all rendered English characters overlapping at screen center, severely compromising readability. Gameplay implementation suffered from an excessively scaled map and poor controllability. Unresolved edge cases permitted conflicting directional inputs (e.g., accepting right-key commands during leftward movement), triggering premature termination.

The solution featured limited code volume, suboptimal performance, poor maintainability, and no distinguishing innovations.

For the game generated by the ChatGPT 3.5 model in this link, there are the following results. Critical interface components were missing, including start, pause, and game-over screens. Execution immediately entered gameplay and terminated upon wall collisions or fruit consumption. The solution exhibited minimal code volume, unsatisfactory performance, poor maintainability, and no observable innovations.

For the game generated by the Grok3 model in this link, there are the following results. The text interface implementation was satisfactory, with fonts rendering legible English characters correctly. Core gameplay functionality was comprehensively implemented according to prompt specifications. The solution demonstrated minimal code volume, stable performance characteristics, and good maintainability, though no significant innovations were observed.

For the game generated by the O3mini model in this link, there are the following results. Text interface execution was successful with proper English font rendering. Core gameplay largely fulfilled prompt requirements, though directional input conflicts persisted during movement (accepting opposite-direction commands without collision warnings). Noteworthy was the innovative fruit scoring mechanism: blue fruits awarded ten points while red fruits awarded one point. This implementation utilized substantial code volume, delivered standard performance, and demonstrated good maintainability.

Both Doubao and DeepSeek generated non-English responses throughout the dialogue phases (excluding code segments), potentially creating accessibility barriers for English-speaking developers.

5.2.3. Comparison of running results of Greedy Chinese Chess Game code generated from Chinese prompt words

After comprehensive evaluations from various sources, the comparison table of code ratings generated based on the same Chinese prompt word is as follows(Table 5):

Table 5. Chinese prompt comparison table about Chinese Chess Game

GENAI Comparison Table						
Evaluation dimension	Functional score (40%)	Code quality score (20%)	Performance score (15%)	Maintainability score (15%)	Innovation score(10%)	Comprehensi ve score
deekseek	26	10	8	13	3	60
Doubao	34	15	12	11	4	76
chatGPT3.5	10	10	5	5	0	30
grok3	38	19	14	14	5	90
O3mini	38	18	14	13	8	91

For the game generated by the Deepseek model in this link, there are the following results.Generated only a window framework without background music. Failed basic gameplay validation tests.

For the game generated by the Doubao model in this link, there are the following results. Implemented complete interface flow with functional music and classically colored board. Deficiencies included: non-standard board dimensions preventing valid piece movement (pieces were decorative only). Solution comprised minimal code (300 lines), exhibited average performance, moderate maintainability, and no significant innovations.

For the game generated by the ChatGPT 3.5 model in this link, there are the following results. Produced solely a text-based board pattern displayed in the compiler terminal. Failed validation tests.

For the game generated by the Grok3 model in this link, there are the following results. Achieved complete interface flow with functional music and font rendering. Core functionality included valid piece movement and capture mechanics. Deficiencies: minor board rendering inaccuracies, flawed pawn movement logic, unimplemented check detection, identical difficulty levels in Human-vs-Machine Mode, and reversed undo move buttons for Red/Black in Human-vs-Human Mode. Solution contained substantial code (550 lines), demonstrated stable performance, and higher maintainability. Notable innovation: highlighting valid moves during piece selection.

For the game generated by the O3mini model in this link, there are the following results. Implemented a complete game interface flow with authentic background music and classically styled board rendering. The solution faithfully replicated standard chessboard dimensions while demonstrating correct piece movement logic, valid capture mechanics, properly functioning check resolution, and accurately implemented win/loss conditions. Deficiencies included mandatory full game termination upon completion, undifferentiated difficulty levels in Human-vs-Machine Mode, and inverted undo move controls for Red/Black players in Human-vs-Human Mode. The implementation comprised substantial code (700 lines), maintained stable performance metrics, and exhibited high maintainability.

5.2.4. Comparison of running results of Greedy Chinese Chess Game code generated from English prompt words

After comprehensive evaluations from various sources, the comparison table of code ratings generated based on the same English prompt word is as follows (Table 6):

Table 6. English prompt comparison table about Chinese Chess Game

GENAI Comparison Table						
Evaluation dimension	Functional score (40%)	Code quality score (20%)	Performance score (15%)	Maintainability score (15%)	Innovation score(10%)	Comprehensive score
deekseek	10	5	5	10	3	33
Doubao	25	15	10	10	5	65
chatGPT3.5	10	5	5	5	0	25
grok3	20	10	10	10	5	55
O3mini	20	10	10	12	0	52

For the game generated by the Deepseek and ChatGPT 3.5 model in this link, there are the following results. Both implementations failed validation tests, exhibiting deficiencies consistent with the Chinese version evaluation.

For the game generated by the Doubao model in this link, there are the following results. Implemented an English-language main interface with functional music controls, where both Human-vs-Machine Mode and Human-vs-Human Mode buttons successfully transitioned to the gameplay screen. However, the gameplay interface rendered only a non-standard chessboard with exit, undo move, and music controls, while exhibiting critical absence of playable chess pieces that prevented valid gameplay. The solution comprised minimal code (237 lines), demonstrated moderate maintainability, and introduced no discernible innovations.

For the game generated by the Grok3 model in this link, there are the following results. Implemented only a Chinese-language main interface with functional background music. Critical failure occurred when selecting either Human-vs-Machine Mode or Human-vs-Human Mode, triggering immediate program termination. The implementation comprised substantial code volume (622 lines) with moderate maintainability and no observable innovations.

For the game generated by the O3mini model in this link, there are the following results. Developed solely an English-language main interface featuring non-interruptible background music. Both Human-vs-Machine Mode and Human-vs-Human Mode buttons remained unresponsive. The solution utilized moderate code volume (485 lines), exhibited higher maintainability, though no distinguishing innovations were present.

6. Discussion

The experimental results demonstrate significant performance disparities among the models in the code generation task.

Specifically, the Grok3 and O3-mini models exhibit distinct advantages. Among the second-tier models, Doubao and DeepSeek display differentiated characteristics: the former excels in control flow generation tasks, while the latter shows superior capability in data structure manipulation. Notably, GPT-3.5 underperformed relatively in this task, with a significantly higher failure rate compared to other models. Claude was also tested using identical prompts, generating modularized Python files with more complete logic, yielding code with enhanced maintainability and functionality. However, as a commercially licensed model, Claude was excluded from this comparative analysis.

In cross-linguistic comparisons, all models exhibited increased error rates under English prompt conditions. Manual verification suggests this phenomenon may stem from semantic inaccuracies in English prompts, leading to misinterpretation of intended requirements.

Regarding game code generation using GENAI models, the incremental requirement introduction approach (gradually adding new requirements) significantly outperformed the single-prompt method (providing all requirements simultaneously).

7. Conclusions

This research empirically analyzes the performance disparities of multiple large language models in 2D game code generation tasks through systematic prompt engineering experiments. Key findings reveal:

Significant Model Capability Stratification: Modern instruction-tuned models (Grok-3, O3-mini) demonstrate superior performance in modular construction and code executability, with their code generation quality effectively lowering implementation barriers for non-professional developers.

Validation of Democratized Development Feasibility: Optimized model selection combined with progressive prompting strategies enables users without programming expertise to accomplish basic game development, confirming viable pathways for AI-driven democratization of content creation.

Non-negligible Generational Gaps: Early-generation models (e.g., GPT-3.5) fail to meet contemporary complex development requirements due to high logical defect rates.

This research empirically analyzes the performance disparities of multiple large language models in 2D game code generation tasks through systematic prompt engineering experiments. Key findings

reveal:

Significant Model Capability Stratification: Modern instruction-tuned models (Grok-3, O3-mini) demonstrate superior performance in modular construction and code executability, with their code generation quality effectively lowering implementation barriers for non-professional developers.

Validation of Democratized Development Feasibility: Optimized model selection combined with progressive prompting strategies enables users without programming expertise to accomplish basic game development, confirming viable pathways for AI-driven democratization of content creation.

Non-negligible Generational Gaps: Early-generation models (e.g., GPT-3.5) fail to meet contemporary complex development requirements due to high logical defect rates.

References

[1] Hu, J. (2007). Research and application of artificial intelligence in game development [Master's thesis, University of Electronic Science and Technology of China]. CNKI.

[2] Ko, H. K., Park, G., Jeon, H., Jo, J., Kim, J., & Seo, J. (2023, March). Large-scale text-to-image generation models for visual artists' creative works. In Proceedings of the 28th International Conference on Intelligent User Interfaces (pp. 919-933)

[3] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarialnets [C]//Advances in neural information processing systems. 2014: 2672-2680.

[4] Kingma D P, Welling M. Auto-encoding variational bayes [J]. arXiv preprint arXiv: 1312.6114, 2013.

[5] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need [J]. Advances in neural information processing systems, 2017, 30.

[6] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models [J]. Advances in neural information processing systems, 2020, 33: 6840-6851.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need, " in Proc. Adv. Neural Inf. Process. Syst., vol. 30, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Long Beach, CA, USA. Red Hook, NY, USA : Curran Associates, Dec. 2017, pp. 1–11.

[8] B. Idrisov and T. Schlippe, "Program code generation with generative AIs, " Algorithms, vol. 17, no. 2, p. 62, Feb. 2024.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pretraining, " OpenAI, Tech. Rep., 2019. Accessed: Apr. 11, 2025. [Online]. Available:

[10] D. Palla and A. Slaby, "Evaluation of Generative AI Models in Python Code Generation: A Comparative Study, " in IEEE Access, vol. 13, pp. 65334-65347, 2025.