BACHELOR THESIS

AI-Driven Game Development with Gemini 2.5 Pro

Can Prompting Alone Lead to a Fully Playable Game?

Author

Elena Popp

angestrebter akademischer Grad / aspired academic degree

Bachelor of Science (BSc)

Wien, 2025 / Vienna, 2025

Studienkennzahl lt. Studienblatt / Studyprogramm id according to studysheet :	UA 033 521
Fachrichtung / Specialisation:	Informatik – Medieninformatik
Betreuer / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Abstract

The rapid advancement of Large Language Models (LLMs) raises fundamental questions about the limits of AI in software development. This thesis investigates whether a complete and playable game can be developed using only natural language prompts with an advanced LLM, Gemini 2.5 Pro. To explore this, the study compares two opposing development methodologies without the use of a traditional game engine.

The first method, a top-down "Big Prompt" approach, attempted to generate the entire game from a single, comprehensive instruction. This approach failed, as the AI was unable to generate required game assets and produced a cascade of fundamental code errors. Resolving these errors required extensive manual intervention and further prompting, which defeated the initial hypothesis of a single-step, autonomous generation process and resulted in a non-playable prototype.

The second method, a bottom-up iterative approach, broke the development down into 585 small, manageable tasks. This strategy proved highly successful, leading to a fully functional and playable 3D brick-building game. A quantitative analysis of this process showed that the AI's performance is task-dependent: it excels at generating code for common, well-documented problems like user interfaces, but struggles with novel, project-specific logic that requires detailed, step-by-step explanation.

The study concludes that creating a playable game with prompts alone is possible, but only through a structured, iterative process. This positions the AI not as an autonomous developer, but as a powerful co-pilot that requires skilled, step-by-step guidance from a human expert.

Kurzfassung

Die rasante Weiterentwicklung von großen Sprachmodellen (LLMs) wirft grundlegende Fragen zu ihrer praktischen Anwendung in der Softwareentwicklung auf. Diese Bachelorarbeit untersucht, ob ein vollständiges und spielbares Spiel ausschließlich durch natürlichsprachige Anweisungen (Prompts) mit einer fortschrittlichen LLM, Gemini 2.5 Pro, entwickelt werden kann. Um dies zu untersuchen, vergleicht die Studie zwei gegensätzliche Entwicklungsmethoden ohne den Einsatz einer traditionellen Game-Engine.

Die erste Methode, ein Top-Down-"Big Prompt"-Ansatz, versuchte, das gesamte Spiel aus einer einzigen, umfassenden Anweisung zu generieren. Dieser Ansatz scheiterte, da die KI nicht in der Lage war, benötigte Spiel-Assets zu erstellen und eine Kaskade von grundlegenden Code-Fehlern produzierte. Die Behebung dieser Fehler erforderte erhebliche manuelle Eingriffe und weitere Prompts, was die ursprüngliche Hypothese einer autonomen, einstufigen Generierung widerlegte und zu einem nicht spielbaren Prototyp führte.

Die zweite Methode, ein iterativer Bottom-Up-Ansatz, zerlegte die Entwicklung in 585 kleine, handhabbare Aufgaben. Diese Strategie erwies sich als äußerst erfolgreich und führte zu einem voll funktionsfähigen und spielbaren 3D-Bauspiel. Eine quantitative Analyse dieses Prozesses zeigte, dass die Leistung der KI aufgabenabhängig ist: Sie brilliert bei der Erstellung von Code für gängige, gut dokumentierte Probleme wie Benutzeroberflächen, hat aber Schwierigkeiten mit neuartiger, projektspezifischer Logik, die eine detaillierte, schrittweise Erklärung erfordert.

Die Arbeit kommt zu dem Schluss, dass die Erstellung eines spielbaren Spiels allein durch Prompts möglich ist, aber nur mittels eines strukturierten, iterativen Prozesses. Dies positioniert die KI nicht als autonomen Entwickler, sondern als leistungsstarken Co-Piloten, der eine fachkundige, schrittweise Anleitung durch einen menschlichen Experten benötigt.

Contents

List of Figures

1 Introduction

Developing digital games is not only technically challenging but also creatively demanding, especially for beginners. Programming languages, design, and the use of specific tools are not easy to learn and require significant effort and expertise.

However, in recent years, the rapid development of artificial intelligence and large language models (LLMs) has begun to ease some of these challenges by supporting developers in both programming and learning processes. On November 30, 2022, OpenAI released the first version of ChatGPT for end users, and since then, the range of AI tools available on the market has expanded rapidly [1]. In response, Google released Gemini 1.0 on December 6, 2023 [2]. The most powerful version currently available is Gemini 2.5 Pro [1] [3].

These large language models enable users to solve tasks using natural language input. Whether it's research, image generation, or code generation, the possibilities seem nearly limitless [4]. These new opportunities raise fundamental questions about the practical limits of AI in software creation. This thesis therefore investigates whether a complete and playable game can be created using exclusively natural language prompts with a large language model like Gemini 2.5 Pro. To isolate the direct capabilities of the model, game engines like Unity or Godot are deliberately excluded.

To explore this, the study compares two opposing development methodologies. The core of the investigation lies in answering which prompting strategy is more effective: a "top-down" approach using a single, comprehensive "Big Prompt," or a "bottom-up" iterative process. The first approach tests the AI's ability to manage a large, complex task autonomously, while the second positions the AI as a co-pilot in a structured, step-by-step workflow. The goal is to evaluate these methods based on the development process, the quality and functionality of the generated code, and the final playability of the game. A key aspect of this evaluation also includes assessing the model's capability to generate all required game assets, such as images and sounds, a critical component for a truly automated development process.

This thesis investigates a relatively new application area of artificial intelligence and aims to clarify what users can realistically expect when using prompt-based development for games. It also identifies current limitations and highlights where human expertise is still essential.

To answer these questions, this thesis is structured as follows. Chapter 2 provides an overview of the current state of research on LLMs in game development and prompt-based engineering. Chapter 3 details the game concept and the specific design requirements that served as the basis

---

[1] As of April 2025

for the experiment. Chapter 4 documents the implementation process and the outcomes of both the "Big Prompt" and the iterative development approaches. Subsequently, Chapter 5 presents a quantitative analysis of the successful iterative process and discusses the comparative findings. Finally, Chapter 6 concludes the thesis by summarizing the key insights and providing an outlook on future work.

2 Related Work

The integration of Large Language Models (LLMs) into game development is a rapidly advancing field promising to reshape traditional workflows [5]. To contextualize the experiment of this thesis, this chapter reviews the current state of research at the intersection of AI and game creation. The evolution of LLMs has unlocked new possibilities for prompt-based development, a paradigm where natural language commands drive creation processes [4]. This chapter first provides an overview of how LLMs are currently applied in the industry and then focuses on the specific discipline of prompt-based development, outlining established strategies and the inherent challenges that directly motivate the experimental design of this work.

LLMs are unlocking new creative and technical territories for game mechanics, interactive content, and storytelling [6], [7]. Industry surveys confirm this trend: a large percentage of studios already rely on AI technologies and intend to expand their use, with many reporting significant productivity gains, for instance, 39% of studios report productivity increases of at least 20%, and 25% cite comparable cost savings [8], [9]. The support for game creation through LLMs manifests in two broad ways: *in-game*, where they can act as interactive characters to foster immersive play, and *out-of-game*, where they aid in design concepts, code generation, and content creation [10]. This rise of LLMs has initiated in what many researchers term "Video Game Development 3.0" [11], where the developer's role shifts from pure creation to one of curation and refinement of AI-generated content. This approach is crucial, as the collaboration between human expertise and AI capabilities defines the current state of AI-driven development [12], [13].

In graphics and animation, AI tools are increasingly used to enhance visual quality, from creating realistic character animations to procedurally generating terrain and improving lighting effects [14]. For storytelling and creative content, AI speeds up tasks such as level design and quest scripting, which is particularly beneficial for narrative-driven games [6]. A notable example is Game Retheming, where LLMs generate new game worlds and characters based on a high-level theme [10]. Further studies, such as the CrawLLM study, illustrate how an LLM can function as a central coordinator in the development process, orchestrating extensive portions of a game from a single theme [15]. However, this potential comes with significant challenges. A primary concern is "hallucinations," where LLMs generate false or illogical information. A particularly risky sub-category, "Package Hallucinations," involves the invention of non-existent software libraries, posing a security risk [16]–[18]. Other challenges include biases from training data, technical constraints like poor long-term planning, and legal risks surrounding content ownership [10], [19], [20].

The discipline of prompt-based game development focuses on transforming natural language prompts into game elements [5], [10]. This approach enables several key applications: players can dynamically shape text-based games, prompts can generate new design content like dialogue or items, or even introduce and adjust gameplay mechanics effortlessly. The reliability of these outcomes depends heavily on the prompting strategies. In the literature, a key distinction is made between single-task and multitask prompts, a topic that is fundamental to understanding the effectiveness of different development workflows and is central to this thesis [21]. Research has analyzed various strategies to improve output quality. These include the Base Prompt, which offers a basic instruction without any additional context, Documentation Prompting, which enriches the prompt with relevant technical documentation, In-Context Learning, which uses a few task-related examples to guide the model, and a Combined Approach, which mixes both documentation and in-context examples. It has been shown that providing examples (In-Context Learning) generally yields the highest functional correctness, while including documentation improves the verifiability of the generated code [22]. The practice of carefully crafting prompts, known as prompt engineering, is crucial, as even minor changes can significantly impact the output [21], [23].

Structured approaches like Chain-of-Thought (CoT) or frameworks like the ADIHQ guideline[2] aim to improve logical reasoning by breaking down problems into smaller steps [25]. The most effective approach, however, often proves to be an iterative, dialogue-based refinement where the user provides feedback [26]. Empirical assessments confirm that this conversational prompting significantly improves performance over fully automated approaches [23]. Advanced techniques like Retrieval-Augmented Generation (RAG) and Self-Refinement aim to automate this process [17], [25]. Moreover, the performance of LLMs can be substantially improved through targeted training techniques like Proximal Policy Optimization (PPO) or innovative approaches such as Prompt-FDC, an augmented prompting technique designed to generate code that adheres to safety-critical industry standards [22], [27].

Despite these advanced techniques, the quality of AI-generated code often requires human intervention to correct errors and ensure it is fully functional and secure [28], [29]. LLMs tend to produce systemic errors, which researchers have systematically categorized and studied. For example, AI-specific errors, often called "hallucinations," are classified into types such as "Intent Conflicting," where the code deviates from user requirements, and "Knowledge Conflicting," where it misuses APIs [18], [30]. The security of this code is a major concern, as models can produce insecure code in a high percentage of security-critical tasks. Research shows, however, that specific prompting techniques, such as adding a security-focused prefix or using Recursive Criticism and Improvement (RCI), can significantly reduce the creation of vulnerabilities [31]. This highlights that while both human and LLM-generated code contain flaws, their nature and severity can differ [32]. The automated assessment of code correctness is therefore critical, with tools using symbolic execution

---

[2] The ADIHQ framework, an acronym for the Spanish Análisis, Diseño, Implementación, Hacking y Quality Assurance (Analysis, Design, Implementation, Hacking, and Quality Assurance), outlines a structured prompting process. The authors conceptually describe these steps as Analyze, Design, Implement, Handle (corresponding to 'Hacking'), and Quality [24].

showing promise for a more detailed evaluation than simple similarity metrics [33].

In game development, these principles are used to create entire game experiences. LLMs have revitalized the field of Procedural Content Generation (PCG), allowing for the creation of game elements like quests, characters, and game mechanics through natural language [34]. The ChatPCG framework even demonstrates how LLMs can design reward functions for other AI agents [35]. However, creating binary assets such as images and 3D models is a separate technical challenge that requires different kinds of models. Research shows that the primary tools for visual asset creation are specialized architectures like Generative Adversarial Networks (GANs) and Diffusion Models [14]. This distinction highlights a central finding of this thesis: a fully prompt-driven development pipeline requires combining different, specialized tools. A post-game jam study further shows the practical difficulties, finding that AI-generated assets are often seen as lower quality and hard to customize [36]. The main challenge is coordinating these different tools, where an LLM could act as a central manager [15]. Finally, this entire process must also handle complex ethical issues, from addressing social biases in generated code [20] to the challenges of building and evaluating serious games with AI [19]. The development of AI-driven tools must therefore consider not just technical feasibility but also the vital human factors of trust and usability, especially in safety-critical domains [27], [37].

3 Design and Main Idea

3.1 Game Concept and Core Objective

To effectively test the capabilities of AI-driven code generation, a suitable project was required. The chosen concept for this thesis is a 3D brick-building game titled "Build Your Brick House," where the main goal for a player is to construct virtual structures with bricks, similar to classic toy sets. This game concept was selected for its variety of technical challenges that are well-suited for evaluating an LLM's ability to write functional code. Specifically, these challenges included the creation of a 3D world with a movable camera, the management of grid-based brick connections, and the implementation of a user interface for selecting different brick shapes and colours. A crucial rule for the experiment was to build everything from scratch, without relying on a traditional game engine like Unity or Godot. This "pure" approach ensures that the evaluation focuses exclusively on the code generation skills of the LLM itself.

3.2 User Experience and Interface Design

A key focus of the game design was to create an intuitive and universally accessible user experience, intended to run on multiple platforms, including Windows, macOS, and Android tablets. The controls were designed to be straightforward, adapted for both mouse-based PC interaction and touch input on tablets, targeting a broad audience from ages 6 to 99 with no prior experience required.

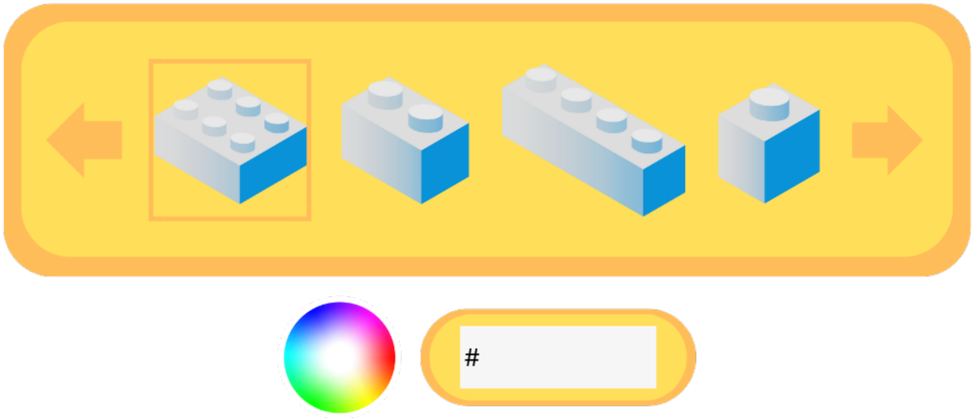


Figure 3.1: Conceptual design of the main user interface

To support this goal, the user interface (UI) was designed to be visually clear and simple, employing a consistent color scheme based on blue and yellow tones. The experience begins with a start screen that features a single ’start’ button, leading directly into the main game. As conceptualized in Figure 3.1, the main game UI provides distinct areas for brick selection in a horizontal, scrollable menu and color customization via a color wheel. A significant design choice was the use of internationally recognized icons and emojis for control elements like save, reset, and undo, as shown in Figure 3.2. This text-free approach makes the game immediately understandable for a global audience, including children who may not yet be able to read.

(a) Save button

(b) Undo button

(c) Reset button

Figure 3.2: Examples of the emoji-based UI buttons, designed for universal understanding without text

3.3 Core Gameplay Mechanics and Logic

The core of the game revolves around the interaction with and placement of virtual bricks. Players can choose from a library of classic brick shapes and special parts, customizing the color of each brick via a color wheel or direct hex code input. Bricks are placed onto a base plate via drag-and-drop and can be subsequently repositioned.

To ensure a realistic and satisfying building experience, a clear set of rules for connection logic and stability was defined. A fundamental requirement is stackability, meaning bricks can only be placed on a grid-based system, preventing floating structures. This is supported by a snapping mechanism that automatically aligns bricks to valid positions. To simulate a basic form of structural dependency, a rule of structural integrity was established: a supporting brick cannot be removed if other bricks are resting upon it. As an advanced stress-test for the AI, the design included the optional goal of structural collapse, where unstable constructions could topple over.

The gameplay is further supported by several technical features. The in-game lighting is designed to resemble natural sunlight, while the audio design includes calm background music and distinct sound effects for player actions. Finally, a save function automatically preserves progress, allowing the player to seamlessly continue their project upon restarting.

Fulfilling these design goals also required specifying a range of digital assets essential for the final product. The core 3D objects included various brick shapes and a base plate. For the user interface, graphical UI icons were also required. The auditory design encompassed calm background music, a click sound for player interactions, and a distinct sound for brick removal. Finally, the design called for a background image for the start screen, featuring scattered, colorful bricks.

3.4 Programming Language and Framework Selection

Choosing the right programming language and framework was a critical first step. To maintain the integrity of the experiment and ensure the process was strictly AI-driven, this decision was delegated to the Gemini 2.5 Pro model. The AI was given a detailed prompt, as shown in Figure 3.3, which outlined the project's core requirements: a standalone 3D game built without a full game engine, cross-platform compatibility, and the use of a language well-suited for AI-assisted development.

Prompt to Gemini 2.5 Pro

> I want to create a simple standalone 3D game without using any game engine like Unity or Godot. However, using lightweight frameworks and libraries is allowed.
>
> The game should run on Windows, macOS, and Android tablets.
>
> Please recommend two programming languages (e.g., JavaScript, Python, C++, or GDScript) that work well for AI- assisted development via prompting, especially with models like Gemini. The languages should be suitable for building such games using only frameworks (not full game engines), and the code should be easy to generate, understand, and modify through natural language prompts. The development workflow should also be compatible with Visual Studio Code on a MacBook Air.

Figure 3.3: The prompt used to select the project's programming language

The model's analysis presented two main options: JavaScript with web technologies like Three.js, or Python with multimedia libraries. It highlighted that JavaScript's ecosystem offered more mature frameworks for 3D graphics and a smoother path to cross-platform deployment. Based on this reasoning, the AI's final conclusion, shown in Figure 3.4, was a clear recommendation for JavaScript.

Reply from Gemini 2.5 Pro

Conclusion:

Both JavaScript (leveraging web technologies like WebGL/WebGPU via Three.js/Babylon.js and packaging tools) and Python (using libraries like Pyglet for desktop and Kivy for cross-platform including Android) are excellent choices that meet your requirements.

- Choose **JavaScript** if you prefer the web technology stack and want the potentially smoothest path to cross-platform deployment (especially web-to-mobile). The 3D frameworks are very mature.
- Choose **Python** if you prefer its syntax and ecosystem, value its exceptional AI-friendliness for rapid iteration, and are comfortable using a framework like Kivy for Android deployment.

Both languages will work well with Gemini for code generation/understanding and are well-supported in VS Code on your MacBook Air.

Figure 3.4: Gemini 2.5 Pro's recommendation of JavaScript for the project

To verify this choice, a follow-up prompt confirmed JavaScript's suitability for the specific asset types required (Figure 3.5). Consequently, this AI-driven decision was crucial for eliminating author bias and ensuring the study's objectivity.

Reply from Gemini 2.5 Pro

Overall Recommendation for this specific project:

JavaScript (with Three.js or Babylon.js)

Reasons:

1. **Superior GLB Handling:** The gITF/GLB ecosystem is most mature and seamlessly integrated within web-based 3D libraries like Three.js and Babylon.js. This is a key asset type for you.
2. **Web Audio API:** A powerful, standard API for all your audio needs.
3. **Flexible UI Options:** You can choose between simple HTML/CSS overlays (very easy for buttons) or more integrated in-canvas UI systems offered by the 3D libraries, both supporting PNGs.
4. **AI-Assisted Development:** The sheer volume of examples, tutorials, and open-source projects using Three.js/Babylon.js with GLB models, audio, and UI elements means Gemini will have a vast dataset to draw from for generating accurate and useful code snippets. Prompting for "load a GLB model in Three.js and make it clickable" or "play a WAV sound on button press using Web Audio API" is likely to yield excellent results.
5. **Cross-Platform Consistency:** The development workflow using web technologies packaged with Electron (Desktop) and Cordova/Capacitor (Android) provides a consistent codebase and environment. The underlying browser engines (Chromium) in these wrappers have excellent support for all your specified asset types.

Figure 3.5: AI confirmation of JavaScript's suitability for required asset types

4 Implementation

This chapter details the implementation phase of the project, documenting the two distinct prompting methodologies used to generate the game code. The first approach, the 'Big Prompt', attempted to generate the entire game from a single, comprehensive instruction set based on the design requirements from Chapter 3. The second, the 'Iterative Approach', built the game step-by-step through a series of targeted prompts.

4.1 The "Big Prompt" Approach: A Top-Down Experiment

The first method tested was a "top-down" approach, using one single, large, and comprehensive prompt. This experiment was designed to test the ultimate promise of advanced LLMs: the ability to act as an autonomous developer. The hypothesis was that the model, given all design requirements at once, could create a complete and playable game without any manual help or step-by-step guidance. The AI was provided with the full game design specification (see Appendix A), with the expectation of receiving a "copy, paste, and play" solution.

However, this initial strategy immediately failed due to a fundamental limitation: the AI's inability to generate binary files. When prompted for the complete game including assets, Gemini 2.5 Pro explicitly stated its constraint, as shown in Figure 4.1.

Reply from Gemini 2.5 Pro

```
...due to the limitations of this text-based format, I cannot directly
provide binary files like images and sounds.
```

Figure 4.1: The AI's explicit statement on its inability to generate binary files

This outcome immediately revealed a critical flaw in the strategy. As the model could not produce the required binary files (.png, .mp3, or .glb), a complete, asset-inclusive game generation was impossible. This first result already confirmed that a fully automated development pipeline, a key aspect of the initial research question, cannot be achieved with a purely text-based model. Consequently, the strategy was adjusted to request only the source code, shifting the focus to the AI's programming capabilities alone.

Reply from Gemini 2.5 Pro

Project Structure: `build-your-brick-house/`

```
|- assets/
|  |- (Optional:  textures, models if not procedural)
|- css/
|  |- style.css
|- js/
|  |- vendor/
|  |  |- three.module.min.js (Three.js library)
|  |  |- cannon-es.min.js (Cannon-es physics library)
|  |  |- OrbitControls.js (Three.js OrbitControls)
|  |- BrickUtils.js
|  |- Config.js
|  |- InputHandler.js
|  |- Physics.js
|  |- SaveLoad.js
|  |- UI.js
|  |- main.js
|- index.html
|- README.md (Instructions will go here conceptually)
```

Figure 4.2: Project structure proposed by the AI for the code-only "Big Prompt" attempt

Initially, the AI's proposal of a project structure seemed reasonable (Figure 4.2). This initial promise, however, quickly dissolved as the generated code proved to be fundamentally flawed. The development process immediately devolved into a cascade of errors, starting with setup and syntax issues like missing library files and invalid module imports, which rendered the project non-functional from the start (Figure 4.3).

Console Error

```
Failed to load resource:
the server responded with a status of 404 (Not Found)       cannon-es.min.js:1
Uncaught SyntaxError:

Cannot use import statement outside a module  three.module.min.js:6 Refused to

execute script from 'http://.../cannon-es.min.js'
because its MIME type ('text/html') is not executable,
and strict MIME type checking is enabled.   index.html:1 Uncaught SyntaxError:

Cannot use import statement outside a module       OrbitControls.js:1 Uncaught

ReferenceError:
THREE is not defined                                         InputHandler.js:7
  at InputHandler.js:7:17
  at InputHandler.js:389:3
...
```

Figure 4.3: Initial setup and syntax errors from the "Big Prompt" approach

Even after these initial problems were manually fixed, deeper logical flaws appeared, which started an inefficient "error-fixing loop." As documented in Figure 4.4, reporting a runtime error to the AI often resulted in a suggested fix that was also flawed, immediately creating a new, different error. This frustrating cycle was a practical demonstration of the challenges discussed in Chapter 2, such as "hallucinations" and a lack of long-term planning. It showed the AI's inability to understand the entire codebase at once, which made a systematic debugging process impossible and highlighted the severe limitations of this approach.

Prompt to Gemini 2.5 Pro

```
I have now corrected the import and dependency errors myself.  Now there is
the following error:  Physics.js:83 Uncaught TypeError:  world.overlapTest is

not a function
at Object.isPositionAboveOccupied (Physics.js:83:30)
at checkCanMove (InputHandler.js:204:25)
at handlePointerDownLogic (InputHandler.js:158:29)
at HTMLCanvasElement.onPointerDown (InputHandler.js:141:13)
```

Reply from Gemini 2.5 Pro

```
3.  Delete or Comment Out the overlapTest Code:  Find the block of code that
creates a checkBody and calls world.overlapTest.  It looks like this:

    // Inside isPositionAboveOccupied function:
    //   START OF BLOCK TO DELETE OR COMMENT OUT
    const halfExtents = new CANNON.Vec3(tolerance, tolerance, tolerance);
    const shape = new CANNON.Box(halfExtents);
    const checkBody = new CANNON.Body({ mass: 0 }); // Static body
    checkBody.addShape(shape);
    checkBody.position.copy(checkPosition);
    // This is the problematic line and related setup:
    const result = world.overlapTest(checkBody);
    //   END OF BLOCK TO DELETE OR COMMENT OUT
```

Console Error

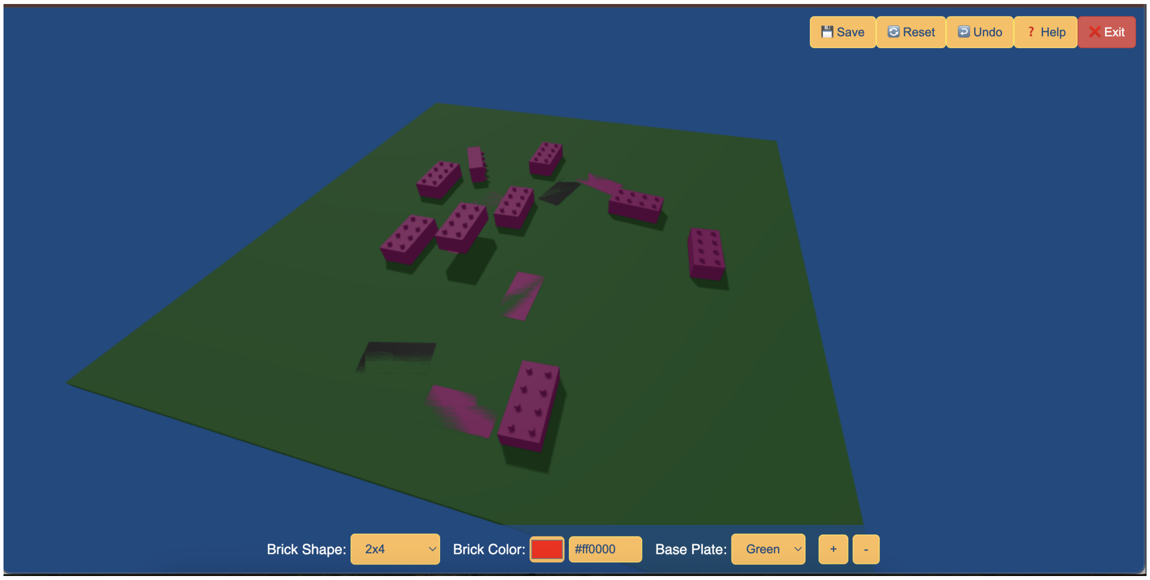
```
Uncaught ReferenceError:  rayStart is not defined              Physics.js:92
  at Object.isPositionAboveOccupied (Physics.js:92:9)
  at checkCanMove (InputHandler.js:204:25)
```

Figure 4.4: The inefficient "error-fixing loop": A runtime error (top), the AI's flawed solution (middle), and the resulting new error (bottom)

After numerous debugging cycles, which required significant manual intervention and further prompting, the "Big Prompt" approach eventually produced an application that could be started. However, this executable version was a fundamentally broken and unplayable prototype that failed to meet the core functional requirements of the game design, as shown in Figure 4.5.

While the application could render a basic scene, the essential gameplay mechanics were chaotic and unpredictable. The brick placement system, for example, did not function correctly; there was no reliable grid snapping, and attempts to stack bricks resulted in erratic behaviour, such as bricks flying apart or collapsing without reason. This made it impossible to construct any meaningful structure. Furthermore, the user interface was rudimentary, consisting of simple text elements instead of the designed graphical buttons.

In conclusion, the "Big Prompt" methodology failed entirely. It did not generate a "copy, paste, and play" solution but rather non-functional code that demanded an extensive and inefficient debugging process just to become executable. Even after this effort, the final prototype did not meet the design requirements and was functionally useless. This outcome provides a clear answer to one part of the thesis's central question: the "Big Prompt" strategy is not a viable path to creating a playable game.



Brick Shape: 2x4
Brick Color: #ff0000
Base Plate: Green

Figure 4.5: The final state of the prototype from the "Big Prompt" approach

4.2 The Iterative Approach: A Bottom-Up Process

The second method tested was an iterative, bottom-up approach. This was the direct opposite of the "Big Prompt" methodology. The idea here was to break down the large task of creating a game into many small, manageable steps. This should allow the LLM to be guided more effectively and, in turn, produce better and more functional code. Instead of giving the AI the entire game design at once, the development started with a single, simple feature. New features were then added one by one. The game design document from Chapter 3 was used as a general guide or roadmap, not as a strict set of instructions.

Each new feature was developed using a simple, repeated cycle. This workflow is central to the success of the iterative method: A short, clear prompt was written for a single, well-defined task, for example, "Create a movable camera" or "Make the bricks snap to the grid." The LLM then created a piece of code specifically for that one task. This new code was immediately added to the project and tested to see if it worked as expected. If an error occurred or the functionality was not perfect, this initiated a debugging sub-cycle. The error message or a description of the bug was given back to the AI in a new prompt (e.g., "The code throws a ‘TypeError‘ on line 5" or "The bricks are snapping with an offset of 0.5"). The AI’s suggested solution was then integrated and tested again. This loop was repeated until the feature worked correctly. The prompts themselves were intentionally kept short and precise. For more complex tasks involving mathematical calculations, such as the advanced placement logic, the prompts included concrete explanations and detailed examples to guide the AI, as shown in Figure 4.6. An interesting observation during the process was that the language of the prompt (German, English, or even a mix of both within a single prompt) had no discernible impact on the quality of the generated code. The AI handled multilingual input seamlessly.

Prompt to Gemini 2.5 Pro

It's still not working as intended. When I press the spacebar, the block I'm holding should jump to the next free Y level below its current position, but only if there is enough vertical space for the block to fit.

Important: A Y level is only considered valid if:

* There is enough space between the objects above and below it to fit the block
* Specifically: the space must be at least 2 * (block height + 0.2)

Example:

* I'm holding a block that is automatically placed at Y = 4
* If there is nothing at Y = 2, then the first spacebar press should move it to Y = 2
* If Y = 0 is empty, the next spacebar press should move it to Y = 0
* BUT: If there is already a block at Y = 0, then the next spacebar press should move it to Y = 1 (if that space is valid and fits)

Figure 4.6: Example of a detailed prompt for a complex feature

The development process was executed in a series of logical steps, starting with the most fundamental components and progressively adding complexity. The initial step was to establish the basic 3D world. A simple, focused prompt was used to generate the base code for the scene. As shown in Figure 4.7, the AI was tasked with creating a ground plane, setting up basic lighting, and implementing a fully movable camera with controls for zooming, rotating, and panning. This ensured that a functional, navigable environment existed before any gameplay logic was introduced. The AI successfully generated the base code for this initial request. The result was a simple but functional 3D scene with a ground plane and a fully controllable camera, as shown in Figure 4.8. This created a solid foundation for all subsequent development steps.

Prompt to Gemini 2.5 Pro

`Please provide the base code for a 3D game in JavaScript. The game should run on Windows, macOS, and Android tablets. For now, it should just display a ground plane when opened.`

- `* The camera must be movable:`
  - `* Zoom with the mouse wheel or pinch-zoom on touch (Android tablet)`
  - `* Rotate the camera by holding the right mouse button and dragging (or two-finger rotate on touch devices)`

Figure 4.7: The initial prompt of the iterative process for scene and camera setup

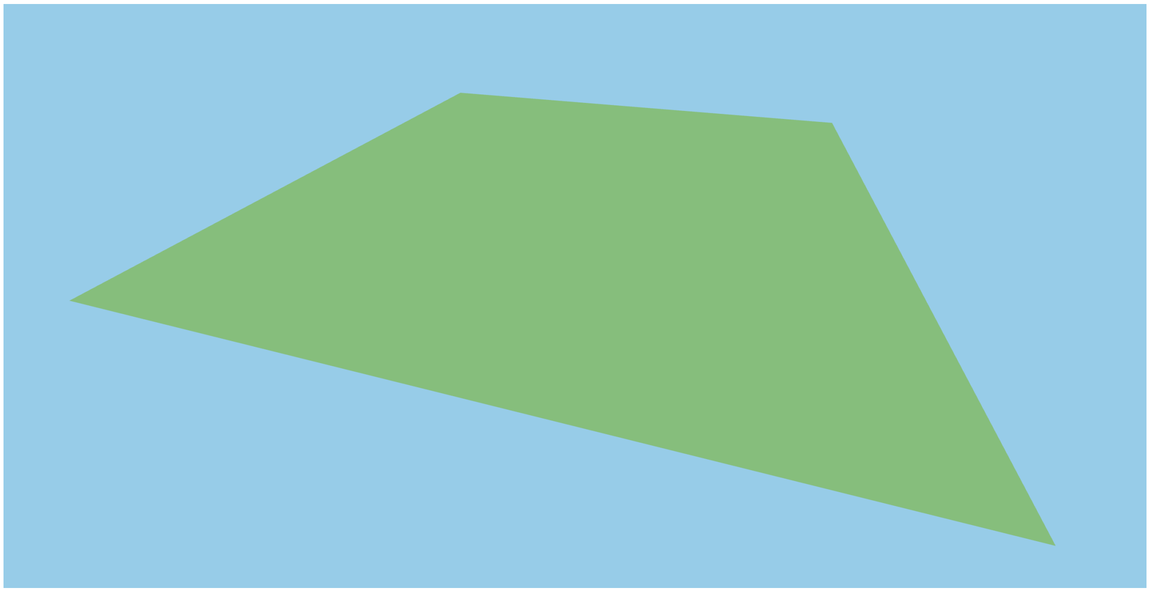


Figure 4.8: The visual result after the first prompt: A navigable 3D scene with a ground plane

With the scene in place, the development focused on the core gameplay mechanics. This complex stage was managed by breaking it down into thematic categories and granular sub-tasks. This problem-driven nature of the workflow allowed specific bugs, like early issues with a "Jitter Base Plate", to be isolated and addressed with targeted prompts before building more complex features like "Brick Placement on Grid".

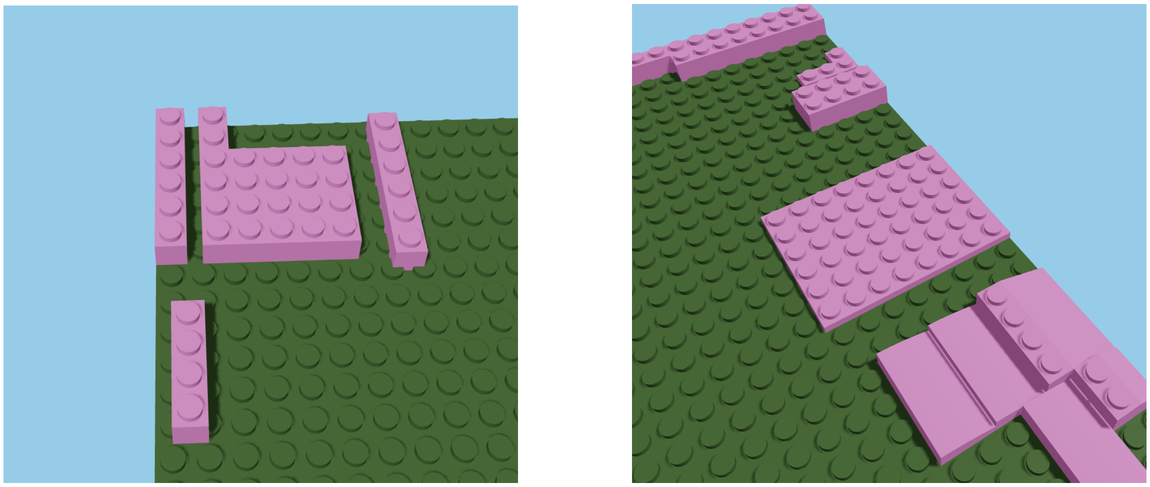
A concrete example of this refinement cycle was the implementation of the grid-snapping mechanism. The initial AI-generated code for this often resulted in a slight alignment offset. The specific follow-up prompt used to correct this is shown in Figure 4.9.

Prompt to Gemini 2.5 Pro

```
It's almost correct now, but the block is offset by 0.5 on both the X and
Z axis.  Please fix this alignment issue so that the snapping is perfectly
centered.
```

Figure 4.9: Example of a targeted prompt used for fine-tuning a feature

This example highlights a core strength of the iterative method: a problem could be isolated, described precisely, and corrected with a targeted follow-up prompt. The visual result of such a refinement cycle is shown in Figure 4.10. This fine-grained control was essential for achieving the required precision in the placement logic.



(a) Placement with incorrect offset

(b) Corrected placement after refinement

Figure 4.10: Visual comparison of the brick snapping logic before and after a targeted refinement prompt

During the implementation of the stacking logic, it became evident that fulfilling the "Structural Collapse" requirement from the design document with a full physics engine was impractical. A conscious design decision was therefore made to adapt the requirements. Instead of a complex physics simulation, the core principle of stability was implemented through a simpler, rule-based system. For example, a targeted prompt was used to enforce the rule that a brick cannot be removed if another brick is resting upon it. This pragmatic adjustment highlights a key advantage of the iterative approach: the ability to identify implementation issues and adapt the design accordingly, ensuring a robust and functional final product. The user interface was also constructed element by element, following the same iterative principle. In stark contrast to the text-only UI from the "Big Prompt" attempt, this approach focused on creating a graphical and interactive interface as specified in the design requirements. Since the AI could not generate 3D models, the required brick assets (in ‘.glb‘ format) were created manually. Subsequently, a series of targeted prompts was used to instruct the AI on how to load these assets and display them in a scrollable side menu. This process resulted in a visual and intuitive brick selection system.

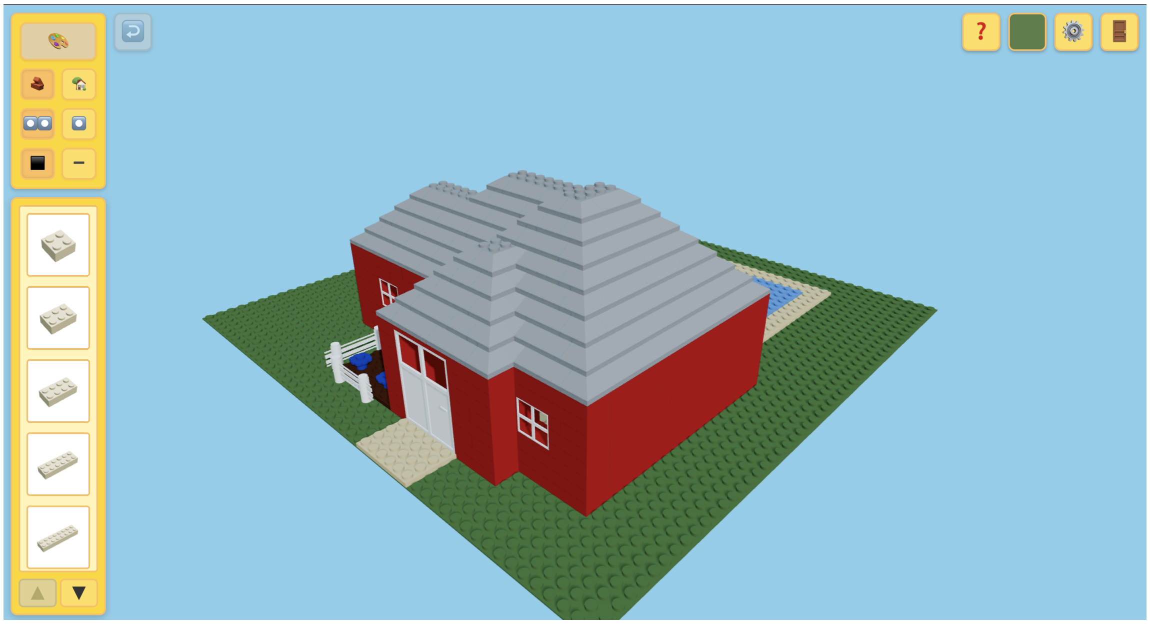


Figure 4.11: The final, functional game resulting from the iterative approach, showcasing a complex user-created structure and the full UI

The iterative, bottom-up methodology proved to be highly successful. This method resulted in a fully playable and feature-complete game that met the core requirements outlined in the design document. The final state of the application, featuring a complex user-created structure and the fully functional graphical interface, is shown in Figure 4.11. All key systems, including the 3D environment, camera controls, core brick placement and stacking logic, and the graphical user interface, were implemented and functional. The step-by-step process allowed for continuous testing and refinement, which prevented the kind of cascading errors that plagued the top-down approach. It is important to highlight the role of manual work in the iterative process. As established in the "Big Prompt" experiment, the AI was unable to generate binary assets. Consequently, all required 3D models for the game's bricks were created manually by the author using Blender. While all game logic was AI-generated, a central configuration file (‘config.js‘) was also maintained by hand. This file, shown in Figure 4.12, served as a central repository for static values, including the file paths to these self-made 3D models, as well as other gameplay parameters. This separation of logic and configuration allowed for easy adjustments without altering the AI-generated code and represents the only part of the codebase not created through prompting. In conclusion, this demonstrates that for complex software development tasks, guiding an LLM through a series of small, well-defined prompts is a far more robust and effective strategy than providing a single, monolithic instruction set.

```javascript
// -- Lighting --
export const LIGHT_CONFIG = {
  ambientIntensity: 1.0,
  directionalColor: 0xffffff,
  directionalIntensity: 1.5,
  directionalPosition: new THREE.Vector3(20, 15, 10),
  shadowMapSize: 2048,
  shadowCameraSize: 50,
  hemisphereSkyColor: 0xcceeff,
  hemisphereGroundColor: 0x777777,
  hemisphereIntensity: 1.7,
};

// -- Controls --
export const CONTROLS_CONFIG = {
  minDistance: 5,
  maxDistance: 100,
  minPolarAngle: 0.1,
  maxPolarAngle: Math.PI / 2 - 0.05,
  dampingFactor: 0.05,
};

// Fallback properties if dimensions cannot be derived from GLB
export const DEFAULT_OBJECT_PROPERTIES = {
  dimensions: { width: 1, height: 1, depth: 1 },
};

// -- Color Palette --
export const COLOR_PALETTE_SIZE = 10;
export const CLASSIC_DEFAULT_COLORS = [
  "#C91A09", // Bright Red
  "#0055BF", // Bright Blue
  "#F2CD37", // Bright Yellow
  "#237841", // Dark Green
  "#FFFFFF", // White
  "#FE8A18", // Bright Orange (LEGO Orange)
  "#A0A5A9", // Medium Stone Grey
  "#6C6E68", // Dark Stone Grey
  "#E4CD9E", // Brick Yellow (Tan)
  "#582A12", // Reddish Brown
];
export const DEFAULT_PALETTE_COLORS = CLASSIC_DEFAULT_COLORS.slice(
  0,
  COLOR_PALETTE_SIZE
);
export const DEFAULT_SELECTED_COLOR_INDEX = 0;

// -- Object Models --
export const OBJECT_MODELS = {
  brick_1x1: "models/Brick_1x1.glb",
  brick_1x2: "models/Brick_1x2.glb",
  brick_1x3: "models/Brick_1x3.glb",
```

Figure 4.12: A snippet of the manually maintained ‘config.js‘ file

5 Evaluation and Discussion

This chapter evaluates the outcomes of the two development methodologies and discusses their implications in the context of the research questions. As documented in Chapter 4, the "Big Prompt" approach failed to produce a playable game and resulted in a chaotic, unmanageable development process. Consequently, a detailed quantitative analysis of this approach was not feasible. The iterative approach, however, was successful and yielded a rich set of process data. The following quantitative analysis therefore focuses exclusively on the iterative development process to evaluate the effort, error rates, and complexity involved. The findings are then used to draw a comparative conclusion between the two approaches.

5.1 Quantitative Analysis of the Iterative Process

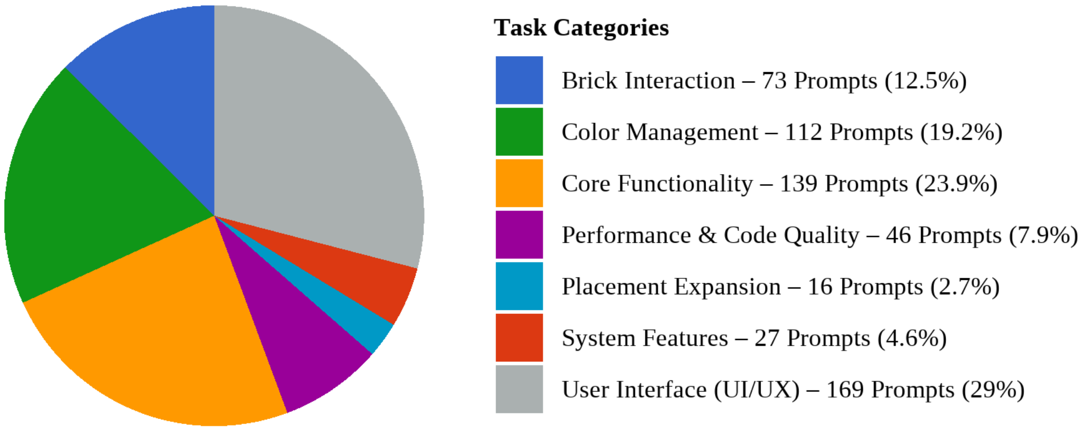


Figure 5.1: Distribution of the 585 prompts across the seven development categories

To understand the development effort involved in the iterative approach, every prompt issued to the AI and any resulting errors were systematically logged and categorized. This data provides the basis for the following analysis of the development workflow. During the iterative development of the game, a total of 585 prompts were issued to the AI. The development work was structured into seven main thematic categories, as outlined in Chapter 4. The distribution of prompts across these categories, illustrated in Figure 5.1, provides insight into where the most development effort was concentrated. The two largest segments, User Interface (UI/UX) at 29% and Core Functionality at 24%, together account for more than half of the total prompts.

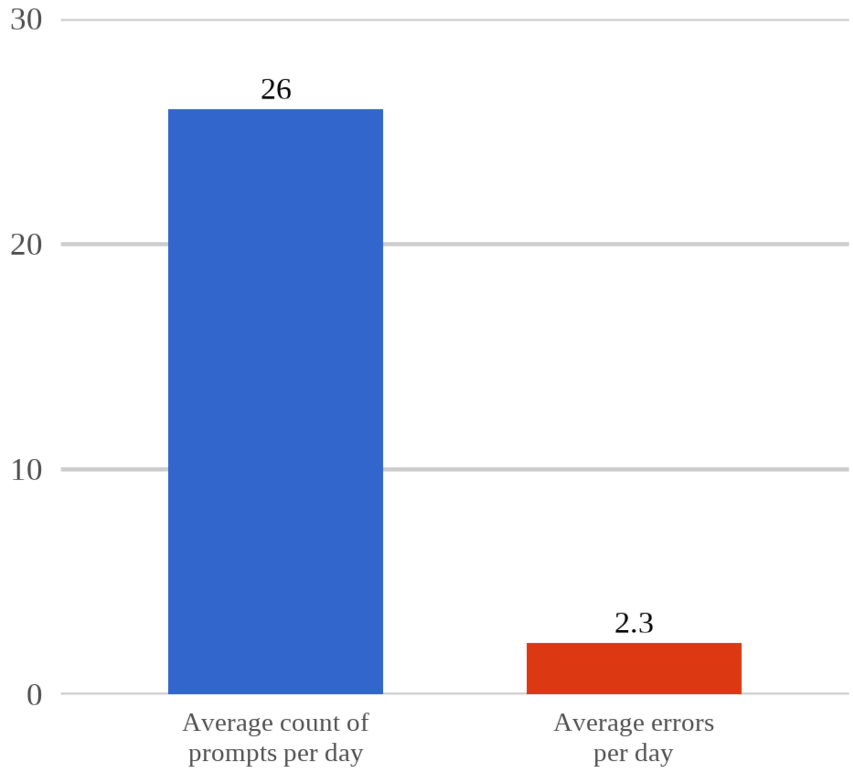


Figure 5.2: Average count of prompts and errors per day during the development process

This indicates that the primary development effort was focused on implementing the user-facing elements and the fundamental gameplay logic. Categories like *Colour Management* (19%) and *Brick Interaction* (12%) also required significant attention, while areas such as *Placement Expansion* (3%) and *System Features* (5%) were less prompt-intensive. On a daily basis, the development involved an average of 26 prompts and resulted in an average of 2.3 errors that required correction (Figure 5.2), highlighting the constant cycle of prompting, testing, and refining that characterized the iterative workflow.

While the overall number of prompts indicates the effort, the error rate per category reveals insights into the AI's reliability for different types of tasks. Figure 5.3 provides a direct comparison between the sum of prompts and the corresponding error rate for each development category. A key finding emerges from this data: there is no direct correlation between the number of prompts and the error rate. The User Interface (UI/UX) category, which required the most prompts (169), had one of the lowest error rates (5.33%). This suggests that the AI is highly reliable for generating code for well-defined, albeit numerous, UI components. In contrast, categories involving more complex logic, such as Placement Expansion (18.75% error rate) and System Features (18.52% error rate), showed significantly higher error rates despite requiring far fewer prompts. This indicates that while the AI can handle a large volume of simple tasks efficiently, it struggles more with tasks that require a deeper understanding of spatial relationships, game state, or intricate logic. The AI's efficiency appears to be task-dependent, excelling at repetitive or straightforward requests but showing weakness in areas demanding complex problem-solving.

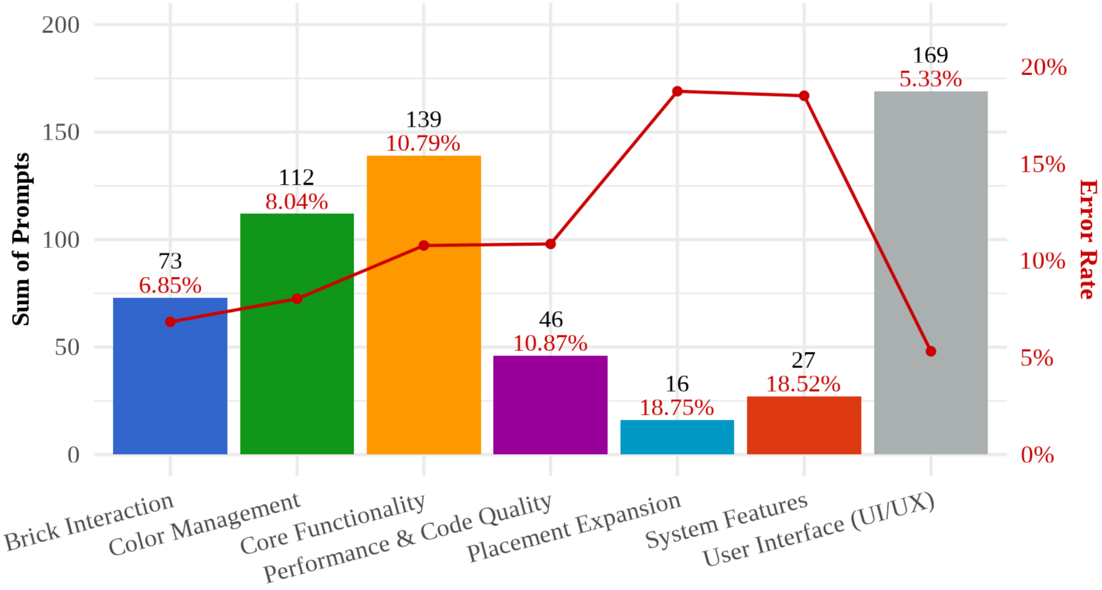


Figure 5.3: Comparison of the total number of prompts versus the error rate for each development category

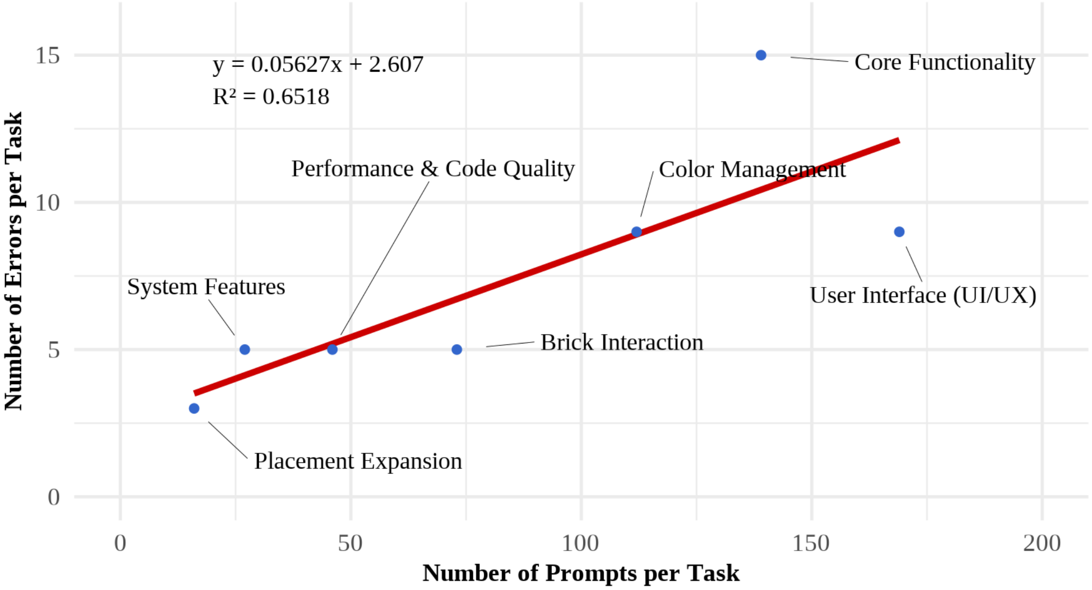


Figure 5.4: Correlation between the number of prompts per task and the resulting number of errors per task

To further investigate the relationship between development effort and error frequency, a correlation analysis was performed. Figure 5.4 plots the number of errors against the number of prompts for each of the seven development tasks. The analysis reveals a coefficient of determination (R^2) of 0.6518, indicating a moderately strong positive correlation between the scope of a task and the total number of errors encountered. However, when analyzing the process on a daily basis (Figure 5.5), a different picture emerges. The correlation between the number of prompts issued per day and the number of errors on that day is significantly weaker, with an R^2 value of only 0.3537. This suggests that the complexity of the specific tasks being worked on appears to be a more significant factor for error occurrence than the sheer volume of work done on any given day.

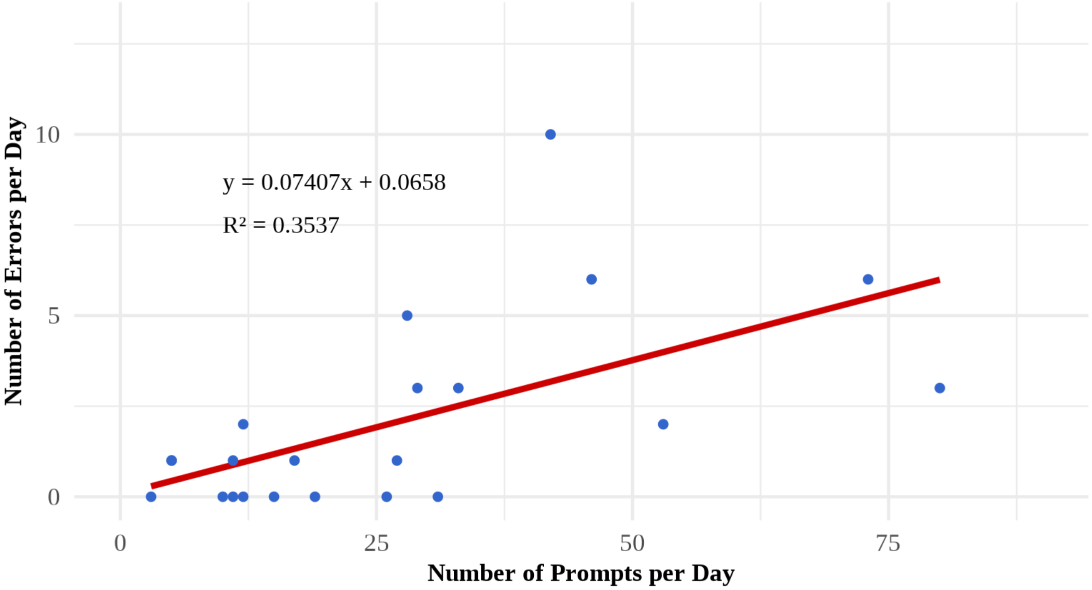


Figure 5.5: Correlation between the number of prompts per day and the number of errors on that day

5.2 Discussion and Method Comparison

The results from the two experimental approaches reveal a stark contrast in effectiveness and practicality. A direct comparison of the methodologies is summarized in Table 1. The "Big Prompt" approach resulted in a chaotic and unpredictable development process. The lack of intermediate steps meant there was no control over the generation process once the prompt was submitted. Errors were often fundamental and cascaded, making debugging a circular and inefficient task. In contrast, the iterative process was structured, measurable, and offered full control at every step. The ability to test and refine small code snippets individually was crucial for managing complexity and ensuring the project remained on a stable path.

The difference in the final product could not be more stark. The "Big Prompt" produced unusable, functionally broken code that, even after significant manual intervention, did not meet the core requirements of the game. The iterative approach, however, yielded high-quality, functional code snippets that could be successfully integrated into a fully playable game. This demonstrates that LLMs, in their current state, are not "black box" code generators capable of handling complex, interconnected requirements, but rather powerful tools that excel when guided with precision. These findings lead to clear answers to the central research questions of this thesis. This experiment demonstrates that a complete, playable game can be created using only natural language prompts, but *only* when employing a structured, iterative methodology. The belief that an AI can function as an autonomous game developer from a single high-level prompt is, based on this experiment, not supported. The iterative, step-by-step strategy is unequivocally more effective. It leverages the AI's strength in solving well-defined, single-task problems while keeping the human developer in full control of the overall process, thus fulfilling the role of a powerful and controllable co-pilot.

Feature	**Big Prompt Approach**	**Iterative Approach**
Process	Chaotic and unpredictable; lack of control.	Structured and measurable; full control at every step.
Code Quality	Unusable and riddled with cascading errors.	High-quality, functional code snippets.
Efficiency	Highly inefficient due to extensive, circular debugging.	Efficient and direct path to implementing features.
Result	A non-playable, functionally broken prototype.	A fully playable game meeting all core requirements.
AI's Role	A "black box" code generator producing an unmanageable output.	A powerful and controllable tool for step-by-step development.

Table 1: Comparison of the "Big Prompt" versus the "Iterative" approach.

These findings also shed light on the evolving role of the human developer in an AI-driven workflow. The traditional role of the programmer, who writes code line by line, recedes into the background. In its place, a new set of skills becomes paramount: the developer becomes an architect and a guide for the AI. In this experiment, the primary task was no longer typing code but the art of decomposing a complex end goal—a playable game—into hundreds of precise, logically sequenced, and unambiguous instructions. This requires a deep understanding of the overall system, strategic planning, and the ability to steer the AI while actively compensating for its weaknesses, such as a lack of long-term context. Thus, the value of human expertise shifts from pure code implementation to the domains of problem decomposition and process orchestration.

5.3 Limitations of the Study

This study has some limitations that are important for a correct interpretation of the results. First, the experiment was conducted using only one AI model: Gemini 2.5 Pro. The results reflect the capabilities of this specific model, and other Large Language Models, such as OpenAI's GPT-4 or Meta's Llama 3, might perform differently. Therefore, the findings of this thesis cannot be generalized to all AI models. Second, the human factor is a limitation. Only one person authored all prompts for the iterative approach. Since the phrasing of a prompt significantly influences the AI's response, another developer with a different style could experience a different development process. The study demonstrates a successful path, but it is not the only possible one. Third, the game genre is specific. The project was a 3D brick-building game, which requires significant logic for placement and interaction. While the results are highly relevant for this genre, they may not apply to other game genres, such as narrative-driven adventure games or fast-paced action games, which have different requirements. Finally, the success of the iterative approach was critically dependent on the quality of the prompting itself. A developer with less experience in "prompt engineering" might have achieved less successful results. The skill of the human "pilot" is therefore a crucial variable that was not isolated in this study and significantly impacts the outcome.

6 Conclusion and Future Work

This thesis explored whether a fully playable game can be developed using only natural language prompts with an AI, comparing a single "Big Prompt" method against an iterative, step-by-step approach. The experiment clearly demonstrated that the "Big Prompt" approach failed to produce a viable result, while a running application could be forced after extensive manual debugging, the game itself was broken, unplayable, and did not meet the core requirements of the design document. The process was inefficient, chaotic, and difficult to control.

In stark contrast, the iterative, bottom-up approach proved highly successful. This method made it possible to guide the AI effectively to build a complete and playable game that met all key design requirements. By breaking the development down into 585 small, manageable prompts, the process remained controlled and predictable. The analysis of this process revealed that the AI excels at generating code for well-defined tasks like user interfaces, where it made few errors, but struggled more with tasks requiring complex, novel logic.

In conclusion, this thesis successfully demonstrated that creating a playable game with prompts alone is possible, but critically, only when a structured, iterative strategy is employed. The experiment refutes the idea that an AI can function as an autonomous developer from a single, high-level prompt. The findings establish that the iterative, step-by-step strategy is unequivocally more effective, positioning the AI as a powerful co-pilot that requires skilled human guidance to navigate the creation of a complex and functional final product.

These findings open up several promising avenues for future research. A key area would be to test if next-generation AI models, such as the upcoming Gemini 3.0, can better handle the "Big Prompt" approach, which would be a test of their improved long-term planning skills. Furthermore, exploring hybrid methods could yield valuable insights, for example, one could use a medium-sized prompt to generate the basic game structure and then use the iterative method to add and refine features. The applicability of this successful iterative method could also be tested on other types of software projects beyond games, such as web applications or data analysis tools, to gauge its broader utility. Finally, future work could focus on creating better tools to support the iterative process.

A The "Big Prompt" Used in the Experiment

The following is the complete, unaltered text of the "Big Prompt" that was given to Gemini 2.5 Pro in the top-down experiment.

```
Please create the complete source code for a game based on the following
game design (see below). The programming language should be JavaScript.

Please provide me with:

The complete game code (in one file or multiple files, as you prefer).
The file/folder names to be used for all code files.
Instructions on how to convert and start the game as a standalone version
and use it on the platforms mentioned in the game design
(no app store required).
A list of all necessary extensions for Visual Studio Code (VSCode),
including all additional settings.
The game should be executable directly after copying/downloading the
code and assets - without further configuration or changes.

Please consider the following game design when creating the game:


Game Design

Introduction

The game is called "Build Your Brick House." It is a creative 3D brick-building
game that makes classic building bricks virtually accessible. The focus is on
intuitive and simple controls, realistic physics simulation, and cross-platform
usability.

All building elements in the game - including bricks, special parts, and the base
plate - are inspired by classic studded building systems. Each piece has realistic
studs and tubes to enable accurate stacking and interlocking behaviour, just like
in real-life brick sets.
```

The parts visually and functionally resemble those well-known interlocking brick systems. The appearance includes visible circular studs on the top and matching holes underneath. Bricks are modular and connect securely by aligning their studs with the underlying piece.

The overall visual style is playful and colourful, but the connection logic should follow realistic rules: Bricks can only snap if aligned properly, and stacking is limited by real physics and stability.

Platforms & Controls

Supported Platforms:
- Windows
- macOS
- Android Tablets

Controls:
- Mouse on PC
- Touch input on tablets

User Interface (UI)

Colour scheme:
- blue (#144a81) for the backgrounds
- yellow / white (#ffbd59, #ffde59, #ffffff) for the buttons

Game Menu:
- Save button and exit button
- Button to reset the game field (delete all bricks)
- Help button (?) with an overview of controls and key functions (e.g., rotate, delete, select)
- Back button to undo the last action

Game Field & Camera

Base Plate:
- Choice between a green or grey surface

Camera:
- Freely movable around the constructed object
- Zoom buttons

Controls:
- Rotate using the right mouse button
- Rotate with a two-finger gesture (touchscreen)

Building Bricks & Interaction

Brick Selection:
- Classic brick shapes (e.g., 2x4, 2x3)
- Special bricks: roofs, doors, windows, flowers, grass, thin/thick bricks
- Select the shape before choosing the colour

Colour Selection:
- Colour wheel or hex code input

Placement & Editing:
- Drag & drop bricks onto the plate
- Bricks can be moved by clicking (left mouse button) and dragging

Removal:
- Detach a brick from another by clicking and dragging (left mouse button) or by long pressing and dragging (touch)
- Either move it to another location or to the trash bin

Control & Functionality:
- All bricks must be stackable
- The top brick must be removed first to move the one below
  (realistic building method)

Physics:
- Realistic physics
- Overloaded or unevenly built structures will topple
  (e.g., a leaning tower falls over)

Lighting & Graphics

Lighting should resemble natural sunlight. It should not be too bright or reflective, so that the bricks remain clearly visible.

Save Function

The construction progress is automatically saved when closing the game. When the game is reopened, the project is automatically loaded, allowing the player to continue building.

Target Audience

All users interested in building with bricks. No prior knowledge is required; the interface is simple and intuitive. Suitable for ages 6-99, if they are familiar with using a PC or tablet.

References

[1] OpenAI. “Introducing ChatGPT.” (Nov. 30, 2022), [Online]. Available: `https://openai.com/index/chatgpt/` (visited on 04/06/2025).

[2] S. Pichai and D. Hassabis. “Introducing Gemini: Our largest and most capable AI model.” (Dec. 6, 2023), [Online]. Available: `https://blog.google/technology/ai/google-gemini-ai/` (visited on 04/06/2025).

[3] K. Kavukcuoglu. “Gemini 2.5: Our most intelligent AI model.” (Mar. 25, 2025), [Online]. Available: `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/` (visited on 04/06/2025).

[4] The AllBusiness.com Team. “The definition of large language model (LLM),” TIME. (Apr. 3, 2025), [Online]. Available: `https://time.com/collections/the-ai-dictionary-from-allbusiness-com/7273967/definition-of-large-language-model-llm/` (visited on 04/12/2025).

[5] P. Sweetser, *Large language models and video games: A preliminary scoping review*, 2024. DOI: `10.48550/arXiv.2403.02613`. arXiv: `2403.02613`.

[6] N. F. Bonn, “The impact of integrating artificial intelligence into the video games industry: A case study,” M.S. thesis, KTH Royal Institute of Technology, Stockholm, Sweden, Jun. 2023.

[7] N. Jain and M. Bhiyana, “The role of artificial intelligence in enhancing software engineering practices: A comprehensive analysis of current applications and future directions,” *International Journal of Contemporary Research in Multidisciplinary*, 2025, Autor:innen als "Dr." im Original. DOI: `10.5281/ZENODO.15700276`.

[8] Camellia. “KI in der spieleentwicklung: Ein tiefer einblick in die trends der spieleentwicklung im jahr 2025.” (Jan. 16, 2025), [Online]. Available: `https://www.meshy.ai/de/blog/ai-game-development` (visited on 04/14/2025).

[9] T. Kirwin, *AI x game dev 2024 – a16z games survey results*, LinkedIn-Post, Dec. 2024. [Online]. Available: `https://www.linkedin.com/posts/troykirwin_ai-x-game-dev-2024-a16z-games-activity-7275181860152877058-s9zM/` (visited on 04/15/2025).

[10] R. Gallotta, G. Todd, M. Zammit, *et al.*, “Large language models and games: A survey and roadmap,” *IEEE Transactions on Games*, pp. 1–18, 2024. DOI: `10.1109/TG.2024.3461510`. arXiv: `2402.18659`.

[11] J. Ratican and J. Hutson, “Video game development 3.0: AI-driven collaborative co-creation,” *Metaverse*, vol. 5, no. 2, p. 2904, 2024. DOI: `10.54517/m2904`.

[12] U. K. Durrani, M. Akpinar, M. Fatih Adak, A. Talha Kabakus, M. Maruf Öztürk, and M. Saleh, “A decade of progress: A systematic literature review on the integration of AI in software engineering phases and activities (2013-2023),” *IEEE Access*, vol. 12, pp. 171 185–171 204, 2024. DOI: `10.1109/ACCESS.2024.3488904`.

[13] C. Treude and R. G. Kula, *Interacting with AI reasoning models: Harnessing "thoughts" for AI-driven software engineering*, 2025. DOI: `10.48550/arXiv.2503.00483`. arXiv: `2503.00483`.

[14] R. Ribeiro, A. V. De Carvalho, and N. B. Rodrigues, "Image-based video game asset generation and evaluation using deep learning: A systematic review of methods and applications," *IEEE Transactions on Games*, pp. 1–10, 2024. DOI: `10.1109/TG.2024.3487054`.

[15] M. Zammit, A. Liapis, and G. N. Yannakakis, "CrawLLM: Theming games with large language models," in *2024 IEEE Conference on Games (CoG)*, Odense, Denmark: IEEE, 2024, pp. 1–2. DOI: `10.1109/CoG60054.2024.10645576`.

[16] J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, B. Viswanath, and M. Jadliwala, "We have a package for you! a comprehensive analysis of package hallucinations by code generating llms," *arXiv preprint arXiv:2406.10279*, 2024. [Online]. Available: `https://arxiv.org/abs/2406.10279`.

[17] A. Eghbali and M. Pradel, *De-hallucinator: Mitigating LLM hallucinations in code generation tasks via iterative grounding*, 2024. DOI: `10.48550/arXiv.2401.01701`. arXiv: `2401.01701`.

[18] F. Liu, Y. Liu, L. Shi, *et al.*, *Exploring and evaluating hallucinations in LLM-powered code generation*, 2024. DOI: `10.48550/arXiv.2404.00971`. arXiv: `2404.00971`.

[19] N. Humble, "Play my thesis: Opportunities and limitations of building serious games with AI," *International Conference on AI Research*, vol. 4, no. 1, pp. 165–174, 2024. DOI: `10.34190/icair.4.1.3130`.

[20] L. Ling, F. F. Rabbi, S. Wang, and J. Yang, "Bias unveiled: Investigating social bias in llm-generated code," *arXiv preprint arXiv:2411.10351*, 2024. DOI: `10.48550/ARXIV.2411.10351`. [Online]. Available: `https://arxiv.org/abs/2411.10351`.

[21] M. Gozzi and F. Di Maio, "Comparative analysis of prompt strategies for large language models: Single-task vs. multitask prompts," *Electronics*, vol. 13, no. 23, p. 4712, 2024. DOI: `10.3390/electronics13234712`.

[22] A. Al-Mashahedi and O. Ljung, "Robust code generation using large language models: Guiding and evaluating large language models for static verification," Master's thesis, Blekinge Tekniska Högskola, Jun. 2024.

[23] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati, *Prompt engineering or fine-tuning: An empirical assessment of LLMs for code*, 2025. DOI: `10.48550/arXiv.2310.10508`. arXiv: `2310.10508`.

[24] R. Cruz, J. Contreras, F. Guerrero, E. Rodriguez, C. Valdez, and C. Carrillo, *Prompt engineering and framework: Implementation to increase code reliability based guideline for LLMs*, 2025. DOI: `10.48550/arXiv.2506.10989`. arXiv: `2506.10989`.

[25] Y. Wang, Q. Zhao, D. Xu, and X. Liu, "Purpose enhanced reasoning through iterative prompting: Uncover latent robustness of ChatGPT on code comprehension," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, International Joint

Conferences on Artificial Intelligence Organization, 2024, pp. 6513–6521. DOI: `10.24963/ijcai.2024/720`.

[26] V.-A. Pădurean, P. Denny, A. Gotovos, and A. Singla, *Prompt programming: A platform for dialogue-based computational problem solving with generative AI models*, 2025. DOI: `10.48550/arXiv.2503.04267`. arXiv: `2503.04267`.

[27] M. Liu, J. Wang, T. Lin, Q. Ma, Z. Fang, and Y. Wu, "An empirical study of the code generation of safety-critical software using LLMs," *Applied Sciences*, vol. 14, no. 3, p. 1046, 2024. DOI: `10.3390/app14031046`.

[28] J. H. Klemmer, S. A. Horstmann, N. Patnaik, *et al.*, "Using AI assistants in software development: A qualitative study on security practices and concerns," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2726–2740. DOI: `10.1145/3658644.3690283`. arXiv: `2405.06371`.

[29] C. Tony, N. E. D. Ferreyra, M. Mutas, S. Dhiff, and R. Scandariato, *Prompting techniques for secure code generation: A systematic investigation*, 2025. DOI: `10.48550/arXiv.2407.07064`. arXiv: `2407.07064`.

[30] Z. Zhang, Y. Wang, C. Wang, J. Chen, and Z. Zheng, *LLM hallucinations in practical code generation: Phenomena, mechanism, and mitigation*, 2025. DOI: `10.48550/arXiv.2409.20550`. arXiv: `2409.20550`.

[31] M. Bruni, F. Gabrielli, M. Ghafari, and M. Kropp, *Benchmarking prompt engineering techniques for secure code generation with GPT models*, 2025. DOI: `10.48550/arXiv.2502.06039`. arXiv: `2502.06039`.

[32] S. A. Licorish, A. Bajpai, C. Arora, F. Wang, and K. Tantithamthavorn, *Comparing human and LLM generated code: The jury is still out!* 2025. DOI: `10.48550/arXiv.2501.16857`. arXiv: `2501.16857`.

[33] D. Cotroneo, A. Foggia, C. Improta, P. Liguori, and R. Natella, "Automating the correctness assessment of AI-generated code for security contexts," *Journal of Systems and Software*, vol. 216, p. 112 113, 2024. DOI: `10.1016/j.jss.2024.112113`. arXiv: `2310.18834`.

[34] M. Farrokhi Maleki and R. Zhao, "Procedural content generation in games: A survey with insights on emerging LLM integration," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 20, no. 1, pp. 167–178, 2024. DOI: `10.1609/aiide.v20i1.31877`.

[35] I.-C. Baek, T.-H. Park, J.-H. Noh, C.-M. Bae, and K.-J. Kim, *ChatPCG: Large language model-driven reward design for procedural content generation*, 2024. DOI: `10.48550/arXiv.2406.11875`. arXiv: `2406.11875`.

[36] S. Bezzina and A. Dingli, "My teammate is an ai: Evaluating generative ai in game asset creation through a post-gamejam study," 2025. DOI: `10.26503/dl.v2025i2.2438`. [Online]. Available: `https://dl.digra.org/index.php/dl/article/view/2438`.

[37] S. V. Gunturi, J. J. Blum, and T. S. Love, “An evaluation of prompt engineering strategies by college students in competitive programming tasks,” 2025.