# Vibe Coding in Practice: Building a Driving Simulator Without Expert Programming Skills

MARGARIDA FORTES-FERREIRA, Leiden University, The Netherlands
MD SHADAB ALAM*, Eindhoven University of Technology, The Netherlands
PAVLO BAZILINSKYY, Eindhoven University of Technology, The Netherlands

The emergence of Large Language Models has introduced new opportunities in software development, particularly through a revolutionary paradigm known as vibe coding or 'coding by vibes,' in which developers express their software ideas in natural language and AI generates the code. This exploratory case study investigated the potential of vibe coding to support non-expert programmers. A participant without coding experience attempted to create a 3D driving simulator using the Cursor platform and Three.js. The iterative prompting process improved the simulation's functionality and visual quality. The results indicated that LLM can reduce barriers to creative development and expand access to computational tools. However, challenges remain: prompts often required refinements, output code can be logically flawed, and debugging demanded a foundational understanding of programming concepts. These findings highlight that while vibe coding increases accessibility, it does not completely eliminate the need for technical reasoning and understanding prompt engineering.

Additional Key Words and Phrases: Vibe Coding, Large Language Models (LLMs), Driving Simulator

**ACM Reference Format:**
Margarida Fortes-Ferreira, Md Shadab Alam, and Pavlo Bazilinskyy. 2018. Vibe Coding in Practice: Building a Driving Simulator Without Expert Programming Skills. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

The emergence of Artificial Intelligence (AI) technologies has significantly impacted individuals, organisations, and society in various fields, from education to computer science [8]. In the contemporary data-driven landscape, generative AI, particularly Large Language Models (LLMs), has emerged as a significant technological advancement [12]. Serving as a subset of AI that seeks to understand and generate human-like language, LLMs have expanded the scope of Natural Language Processing, enabling new functionalities in a wide range of applications, such as machine translation [23], text summarisation [9], question answering [24], and, since recently, code generation [22]. These models are trained in vast repositories of publicly available texts, including books, articles, and websites, enabling them to produce coherent responses and engage in complex linguistic tasks [18]. As a result, LLMs have fundamentally transformed how

*Corresponding Author

Authors' Contact Information: Margarida Fortes-Ferreira, m.raposo.fortes.ferreira@umail.leidenuniv.nl, Leiden University, Leiden, The Netherlands; Md Shadab Alam, m.s.alam@tue.nl, Eindhoven University of Technology, Eindhoven, The Netherlands; Pavlo Bazilinskyy, p.bazilinskyy@tue.nl, Eindhoven University of Technology, Eindhoven, The Netherlands.

individuals interact with digital systems, providing new opportunities for more intuitive, human-centred computing and inducing a paradigm shift in how developers approach their tasks [13].

The initial approaches to apply LLMs to code generation were demonstrated by models such as Code2vec, which was training on millions of methods extracted from the GitHub repositories [2]. These models underscored the effectiveness of capturing the code structure through machine learning techniques. Subsequent research addressed critical challenges associated with mapping natural language to programming syntax—such as the extensive vocabulary of variable names and abstract logic structures—through the advancement of enhanced models such as CodeBERT and Code2Seq [1, 10]. This progress laid the way for a new generation of large general-purpose models designed to manage a wide array of development tasks. Therefore, tools such as Codex [5], GitHub Copilot [7], Replit Ghostwriter [20], and Cursor [6] have emerged as specialised assistants crafted for the software development process and are commonly referred to as "AI Pair Programmers". Thesei systems present different design philosophies: Some function as general-purpose plugins for code editors, others are customised for specific platforms or businesses, and some introduce entirely new types of coding tools [16]. Although these tools offer significant advantages in terms of productivity, quality and efficiency, they also exhibit specific limitations inherent to their non-human nature, such as susceptibility to bias or reliability [13].

Building upon these trends, a novel paradigm in software development has emerged: Vibe coding. This concept was popularised by a tweet in February 2025 [14]. According to Maes [**?** ], vibe coding is an AI-assisted development approach in which the developer articulates their software idea using natural language and the AI automatically generates the corresponding code. Rather than manually writing code, developers express their ideas using natural language, often in the form of concise, high-level prompts, or informal descriptions, and the AI translates this input into executable code. Hence, this paradigm emphasises intent over implementation, thereby shifting the developer's role from coder to conceptual guide. Although early forms of this approach emerged nearly a decade ago utilising non-LLM AI [17], the current generation of tools has made vibe coding more practical and accessible. Developers now provide succinct high-level prompts, referred to as 'vibes', such as single-sentence feature requests, which AI interprets and transforms into functional code. This reduces the barriers to entry for software development and has the potential to democratise the field, making it more accessible to individuals without formal programming training.

According to Pajo [19], despite its promising benefits—increased accessibility, improved efficiency, and increased creative freedom—vibe coding also presents significant challenges. These challenges encompass concerns about the quality and maintainability of AI-generated code, potential security vulnerabilities, as well as changes in the developer skill set that may prioritise high-level design and communication over traditional coding expertise. As the adoption of vibe coding escalates, continuous research and critical evaluation will be imperative to ensure that this paradigm effectively integrates human creativity with the generative power of AI while adequately addressing its inherent risks.

Recent examples, such as *Slow Roads* [21], an endless procedurally generated driving game, can illustrate the creative potential of vibe coding. In particular, this game shows how natural language prompts can independently lead to the creation of interactive 3D environments, complete with vehicle dynamics and terrain generation, capabilities that would traditionally require considerable programming expertise. This increasing accessibility also presents new opportunities for sectors beyond entertainment. Specifically, driving simulators have long been recognised as valuable tools for human factors research on automated driving and traffic safety [3, 4, 11].

These developments underscore the creative opportunities and practical limitations of AI-assisted programming for simulation environments. While AI tools enable the development of interactive 3D experiences using natural language, there is limited research understanding of how accessible and effective these workflows are for non-experts, particularly when applied to complex systems requiring realistic physics and graphics, such as driving simulators.

### 1.1 Aim of study

The aim of this study was to explore how vibe coding, a novel AI-assisted natural language programming approach, can support non-expert developers in creating complex driving simulator games. The research question guiding this work was: *How can vibe coding support non-expert developers in creating complex driving simulator games?* We conducted an exploratory case study in which a participant with no prior coding experience attempted to build a 3D driving simulator using the Cursor platform. This study investigated how effectively prompt-driven development reduces barriers to entry in simulator creation while highlighting the challenges, advantages, and iterative strategies involved. Particular attention was paid to the clarity of communication with AI, the accuracy and usefulness of the generated code, and the overall user experience of the resulting application.

## 2 Method

The study tested the development of an interactive 3D driving simulator using the Cursor platform. Cursor incorporates LLMs into a code editor, enabling users to write and enhance code through natural language prompts. Furthermore, this platform promotes conversational workflows, allowing users to describe what they want the programme's functionalities to be, with the model generating or modifying code in response. Additionally, this platform aids debugging by giving recommendations and alternatives when faced with challenges, making it particularly beneficial for users with limited programming backgrounds.

The simulator was developed by one participant, a Master's student in psychology from a university in [redacted] without coding expertise. The participant employed an iterative, trial-and-error approach to guide the LLM using different natural language prompts, referred to as "vibes" to request features or modifications. For example, the questions included sentences such as 'Make the car more realistic' or 'I want to have a more realistic city'. The system responded with code suggestions, which the participant tested in real time and reviewed through follow-up prompts. When the initial outputs were incomplete or flawed, the participant refined the original prompt or requested corrections.

This project involved Three.js, a JavaScript library for creating and rendering 3D environments in the browser. The library was selected for its lightweight structure and compatibility with web-based rendering, making it a popular choice for interactive visual applications.

The development process was structured around a series of milestones, each focused on implementing a particular feature or functionality, such as city realism, road generation, camera control, or physics and function of cars. These milestones included the description, timeline, and status of each stage. Detailed prompt interactions and corresponding code corrections are provided in Appendix A. Several approaches were tested for many milestones, mainly when the initial attempts produced incomplete or faulty results.

The simulator was developed and tested on a MacBook Air (Retina, 13-inch, 2020) running macOS Monterey 12.7.6, with a 1.2 GHz Intel Core i7 quad-core processor, 8 GB 3733 MHz LPDDR4X RAM, and Intel Iris Plus Graphics (1536 MB). The development was carried out in the Chrome browser using the Cursor application.

## 3 Results

### 3.1 Prototypes of the Simulator

The participant created different interactive 3D driving simulator prototypes using the Cursor platform and Three.js. Each iteration represented a unique evolution of prompt-based workflow and exhibited differences in visual aesthetics, camera perspective, vehicle physics and function, and environmental complexity. These variations emerged from how the LLM interpreted and reacted to different natural language directives. Two representative versions are illustrated in Figures 1 and 2. The first shows a global perspective of a blue vehicle in a stylised urban setting. In contrast, the second shows a third-person perspective of a red vehicle on the road in a realistic city, including complex city elements, in a procedurally generated cityscape. These two prototypes illustrate two distinct trajectories of prompt-based development. One is visually more photorealistic yet static, while the other is functionally complex but less polished in its appearance.

Fig. 1. Prototype 1: a blue car on a stylised city with realistic grass, buildings, and trees.

Fig. 2. Prototype 2: a red car viewed from behind, navigating an AI-generated city with buildings and city elements.

Table 1 summarises the main features of the driving simulator prototype in both versions. Six core features were compared: camera perspective, road generation, vehicle functionality, vehicle physics, environment, and interactivity.

Table 1. Feature comparison between Prototypes 1 and 2 of simulator.

| **Feature** | **Figure 1** | **Figure 2** |
|---|---|---|
| Camera Perspective | - Global overview of the scenario | - Third-person perspective |
| Road Generation | - Consistent geometry across map | - Self-generated road network<br>- Includes intersections and traffic lights |
| Vehicle Functionality | - Static visual model<br>- No driving input or interactivity | - "WASD commands-based movement<br>- No collision detection |
| Vehicle Physics | - Include wheels, reflective properties<br>- Blue | - Red<br>- Simple with a cube shape |
| Environment | - Realistic materials on grass and buildings<br>- Static trees, no interaction | - Procedurally generated city<br>- Dynamic street elements and complex layout |
| Interactivity | - Fully static scene<br>- Primarily visual demonstration | - User-driven navigation |

Each version was generated using natural language prompts in Cursor, without direct code manipulation. The differences in output emerged from the way the prompts were interpreted, refined, or corrected in dialogue with the AI assistant. For example, prompts emphasising realism led to adding buildings and lights, while more open-ended prompts resulted in abstract landscapes. This highlights the limitations and advantages of LLM-assisted coding: achieving both interactivity and realism often requires multiple trial-and-error prompting.

### 3.2 Observations on Prompt Iteration

During the initial stages of the iterative process, the Cursor platform repeatedly delayed processing and responding to prompts. This system was slow to provide outputs or generate appropriate code suggestions, temporarily hindering the rapid iteration that was expected with vibe coding. As the tool, workflow, and participants' familiarity with Cursor developed, its interaction model improved in both responsiveness and efficiency. Thus, this evolution enabled for more seamless experimentation and development in the later stages. Furthermore, during all attempts to develop prototypes of a driving simulator with a single prompt were attempted, the prompt did not produce the desired result. Many core features—particularly those that needed the integration of multiple systems, such as city realism and functioning vehicle controls—were developed through an iterative process. The participant often provided follow-up prompts (as illustrated in Appendix). It took multiple iterations to achieve a usable result for each feature in diverse cases.

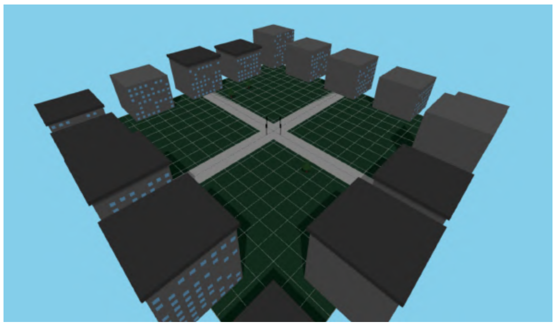

Fig. 3. In-process output, Appendix A.

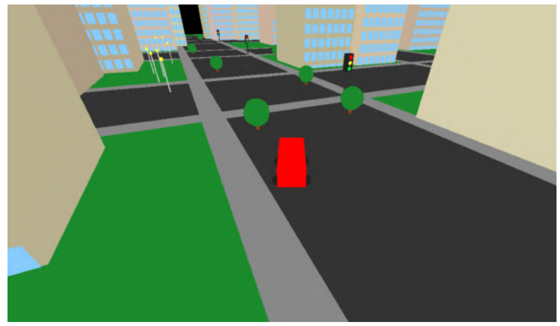

Fig. 4. In-process output, Appendix B.

In addition, prompts that combined technical goals with aesthetic goals generally produced more complex and less stable results. In contrast, simpler prompts typically resulted in clearer, but less intricate, outcomes. These observations highlight the importance of clarity, specificity, and iterative refinement when using LLMs for simulation development. The source code for the final state of the simulator (Prototype 2) and the history of commits are available at [redacted].

## 4 Discussion

This study aimed to understand how vibe coding can help non-expert programming users develop complex simulation environments, as assessed by the result of different prototypes. The analysis of the resulting prototypes indicate both the promise and the limitation of this revolutionary software development paradigm. Vibe coding significantly reduced the barrier to entry for the participant. The participant could design functional simulation scenarios without interfering or writing manual code, which shows the accessibility and potential of this paradigm. However, developing realistic and high-quality environments, such as a photorealistic 3D driving simulator with the Cursor Platform, proved challenging. The process required iterative prompt refinement, trial-and-error, and frequent clarification, ultimately falling short of the desired complexity. These findings indicate that while LLM-based platforms such as Cursor are accessible to the public, they still present challenges in interpreting vague or high-level creative objectives without substantial user guidance and by repeating the same prompts sequentially.

The precision of user prompts influenced the quality of AI-generated code: (1) simple, well-defined prompts resulted in more functional, though basic, outputs; and in contrast, (2) more abstract prompts led to incomplete or logically flawed code. These flaws frequently manifested as error messages in the website console, which required user intervention to identify issues and prompt the system to revise or correct the code. In many cases, the same instruction had to be reformulated two or three times before the system produced a satisfactory response, highlighting the trial-and-error nature of the interaction. These findings emphasise the importance of *prompt literacy*, which refers to the user's ability to express ideas in a way that AI can efficiently convert into executable code. Although vibe coding removes the requirement for coding expertise, it does not eliminate the need for technical reasoning. Users must still understand the system logic, dependencies, and domain-specific concepts. As a result, vibe coding redesigns the developer's role from a traditional or manual programmer to an iterative designer and prompt engineer. Therefore, mastery of prompt engineering is essential to unlocking this paradigm's pull potential. One could argue that a person with a background in Computer Science may be able to 'vibe code' a driving simulator suitable for human factors research. Parallels could be drawn with how a technically skilled X user Pieter Levels (@levelsio) 'vibe coded' a dog fighting game [15], while tweeting the process live. The game was 'complete' and fully functional, featuring realistic plane physics and immersive graphics, resulting in the project becoming viral and more than 100,000 people playing the game.

These insights have broader implications. Vibe coding has the potential to make software accessible to everyone, allowing individuals from non-technical fields—such as user experience design or social sciences—to prototype tools without relying on professional developers. This aligns with the goals of inclusive design and broadening participation in digital innovation.

Furthermore, the findings indicate the need to redesign traditional software development skills. As AI evolves, future frameworks may prioritise systems thinking, natural language communication, and collaborative problem-solving over traditional coding syntax. With continued improvement, platforms like Cursor could become not just tools for development but environments for learning, experimentation, and interdisciplinary collaboration.

## 5 Conclusion

This study proposes the potential of LLMs to help non-experts create 3D driving simulators using natural language prompts. Using the Cursor platform as a unified development environment and relying exclusively on AI-generated code, the research showed that iterative prompting and conversational refinement can produce functional prototypes without the need for traditional programming skills. The findings demonstrated both the potential and limitations of vibe coding. On the one hand, enabling the generation of complex interactive environments with minimal technical knowledge reduces barriers to creative development, enhances the accessibility of computational tools, and empowers non-technical users such as user experience (UX) designers or behavioural scientists to participate actively developing various scenarios. However, the process was challenging: prompts required numerous refinements, the code was syntactically correct, yet logically flawed, and effective debugging still necessitated an understanding of programming logic and structure. This creates an important reflection: Although LLMs can support the coding process, they do not eliminate the need for computational thinking and knowledge. *If programming becomes more abstracted through natural language systems, what types of coding literacy will future developers require?* Coding may transform from a hands-on technical endeavour to a high-level design conversation, but the skills to analyse systems, identify errors, and organise logic remain crucial.

## 6 Limitations and Future Work

This study was conducted with a single non-expert programming participant, who fulfilled the roles of developer and evaluator of the AI-assisted programming process. Although this approach allowed in-depth qualitative insights, the findings cannot be generalised to a broader population. Future research may aim to include a more diverse set of participants' characteristics with diverse levels of technical expertise to investigate the consistency, accessibility, and usability of the prompt-based development process across a broader user base. Furthermore, it remains crucial to include questionnaires and semi-structured interviews because these can further enhance the data by capturing subjective experiences, perceived ease of use, and overall satisfaction.

Furthermore, the development process was limited by the duration of two months, which constrained the complexity and refinement of a final driving simulator prototype. Within this short timeframe, only a subset of potential features could be implemented and more advanced functionalities were not possible to be explored. Therefore, future research may extend the timeline to investigate how prolonged iterative prompting influences the depth and quality of AI-generated applications.

Moreover, the development process relied exclusively on the Cursor platform. This platform was chosen because of its accessibility, cost and availability to the general public. However, it constitutes merely one instance of LLM-assisted programming. Future studies may compare between Cursor and other AI coding platforms, such as GitHub Copilot or OpenAI Codex, to evaluate differences in usability, code quality, error handling, and user trust across these tools.

Finally, while this study focused on the technical aspects of prompt-based development for the driving simulator using vibe coding, a logical next step would be to create a more realistic and controlled driving simulation to analyse human decision-making, attention and traffic safety. Thus, this study provides a crucial foundation for experimental studies to examine human behaviour in different driving contexts.

## Supplementary Material

A maintained version of code is available at https://github.com/Shaadalam9/vibe-simulator.

## References


[1] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400* (2018). doi:10.48550/arXiv.1808.01400

[2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29. doi:10.48550/arXiv.1803.09473

[3] Pavlo Bazilinskyy, Md Shadab Alam, and Roberto Merino-Martınez. 2025. Pedestrian crossing behaviour in front of electric vehicles emitting synthetic sounds: A virtual reality experiment. In *Proceedings of 54th International Congress Exposition on Noise Control Engineering (INTER-NOISE)*. São Paulo, Brazil.

[4] Pavlo Bazilinskyy, Lars Kooijman, Dimitra Dodou, and J. C. F. De Winter. 2020. Coupled simulator for research on the interaction between pedestrians and (automated) vehicles. In *Proceedings of Driving Simulation Conference (DSC)*. Antibes, France.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021). doi:10.48550/arXiv.2107.03374

[6] Cursor. 2025. The AI Code Editor. https://www.cursor.com/. Accessed: April 24, 2025.

[7] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734. doi:10.48550/arXiv.2206.15331

[8] Yogesh K Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, et al. 2023. Opinion Paper:"So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International journal of information management* 71 (2023), 102642. doi:10.1016/j.ijinfomgt.2023.102642

[9] Jiangnan Fang, Cheng-Tse Liu, Jieun Kim, Yash Bhedaru, Ethan Liu, Nikhil Singh, Nedim Lipka, Puneet Mathur, Nesreen K Ahmed, Franck Dernoncourt, et al. 2024. Multi-LLM Text Summarization. *arXiv preprint arXiv:2412.15487* (2024). doi:10.48550/arXiv.2412.15487

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020). doi:10.48550/arXiv.2002.08155

[11] Christian Gold, Daniel Damböck, Lutz Lorenz, and Klaus Bengler. 2013. "Take over!" How long does it take to get the driver back into the loop?. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 57. Sage Publications Sage CA: Los Angeles, CA, 1938–1942.

[12] Desta Haileselassie Hagos, Rick Battle, and Danda B Rawat. 2024. Recent advances in generative ai and large language models: Current status, challenges, and perspectives. *IEEE Transactions on Artificial Intelligence* (2024). doi:10.48550/arXiv.2407.14962

[13] Sajed Jalil. 2023. The Transformative Influence of Large Language Models on Software Development. *arXiv preprint arXiv:2311.16429* (2023). doi:10.48550/arXiv.2311.16429

[14] Andrej Karpathy. 2025. Post on X (formerly Twitter). https://x.com/karpathy/status/1886192184808149383. Accessed on February 2, 2025.

[15] Pieter Levels. 2025. fly.pieter.com. https://fly.pieter.com. Accessed: 2025-06-10.

[16] Stephane H Maes. 2025. The gotchas of ai coding and vibe coding. it's all about support and maintenance. doi:10.5281/zenodo.15343349

[17] Stephane H. Maes, Karan Singh Chhina, and Guillaume Dubuc. 2016. Natural language translation-based orchestration workflow generation. US Patent 11,120,217 B2, granted in 2021.

[18] Jesse G Meyer, Ryan J Urbanowicz, Patrick CN Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, et al. 2023. ChatGPT and large language models in academia: opportunities and challenges. *BioData mining* 16, 1 (2023), 20. doi:10.1186/s13040-023-00339-9

[19] P Pajo. 2025. Vibe Coding: Revolutionizing Software Development with AI-Generated Code. https://doi.org/10.13140/rg.2.2.36458.22727. doi:10.13140/RG.2.2.36458.22727 Accessed: 2025-06-07.

[20] Replit. 2025. Intro to Ghostwriter. https://replit.com/learn/intro-to-ghostwriter. Accessed: 2025-06-08.

[21] Slow Roads. 2025. Slow Roads: An Endless Driving Experience. https://slowroads.io. Accessed: 2025-06-10.

[22] Haoran Su, Jun Ai, Dan Yu, and Hong Zhang. 2023. An evaluation method for large language models' code generation capability. In *2023 10th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 831–838. doi:10.1109/DSA59317.2023.00118

[23] Maryana Tomenchuk and Kseniia Popovych. 2024. Large Language Models and Machine Translation. (2024). doi:10.52058/2695-1592-2024-11(42)-422-431

[24] Wenting Zhao, Ye Liu, Tong Niu, Yao Wan, Philip S Yu, Shafiq Joty, Yingbo Zhou, and Semih Yavuz. 2023. DIVKNOWQA: assessing the reasoning ability of llms via open-domain question answering over knowledge base and text. *arXiv preprint arXiv:2310.20170* (2023). doi:10.48550/arXiv.2310.20170

## Appendix A: Cursor Prompts Used During Development (Figure 1)

The following prompts were entered into Cursor (AI code assistant) during the development of the 3D driving simulator to iteratively build and refine the scene. Each prompt represents a specific design intention or correction attempt in natural language.

**P1:** `Create a basic Three.js scene with a ground plane, ambient light, and a perspective camera. Use orbit controls to inspect the scene for now.`
P2: `Use realistic acceleration and turning logic, not just simple position translation. The car should face the direction it is driving.`
**P3:** `I want to have a more realistic city.`
**P4:** `Create a realistic city, with buildings, trees, traffic lights, roads, and sidewalks for pedestrians.`
**P5:** `Can you create the city even more realistic?`

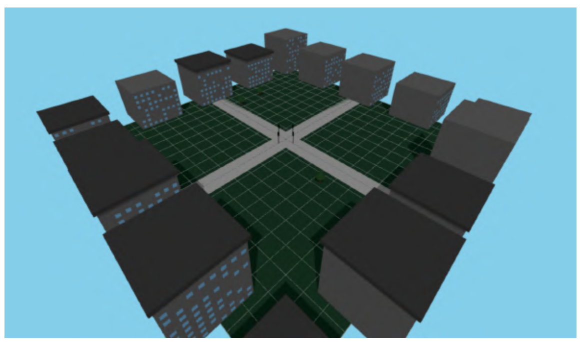

Fig. 5. Result of Prompt 5.

**P6:** `Can you create the city even more realistic?`

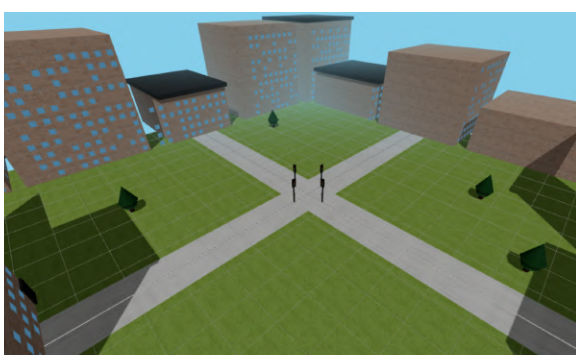

Fig. 6. Result of Prompt 6.

**P7:** `Build a realistic car, with wheels, windows, and five doors.`

**P8:** `Make the car even more realistic, with the right materials, and the color can be blue.`
**P9:** `Make the car more realistic.`
**P10:** `Make the car even more realistic! Focus on the materials first.`
**P11:** `Make the car a bit bigger in terms of dimensions, and I want to properly be able to see the doors of the car in a realistic way.`
**P12:** `Make the car look more realistic!`
**P13:** `Make the car with four doors that are visible and realistic.`
**P14:** `Make sure that I can see the interior of the car through the windows.`
**P15:** `Can you make the trees even more realistic?`
**P16:** `Use the right materials for the trees, to make them real.`
**P17:** `Can you create a "photorealistic" city? Use all the datasets or open APIs you need.`
**P18:** `Add integration with any of these APIs (would require API keys but more realistic).`
**P19:** `Enhance the building data processing?`
**P20:** `Make a "photorealistic" city, using Google or Apple Maps.`
**P21:** `Using the base you already have which is good, can you: include and create a 3D city scene with real map data (e.g., Leiden), legally and freely, using only open APIs and open-source tools?`

### Appendix B: Cursor Prompts Used During Development (Figure 2)

The following sequence illustrates an approximate set of natural language prompts issued to Cursor (AI assistant) during the development of the 3D driving simulator. These prompts were iteratively refined to improve scene complexity, realism, and interactivity.

**P1:** `Create a 3D city scene using Three.js, with roads, sidewalks, buildings, and trees.`

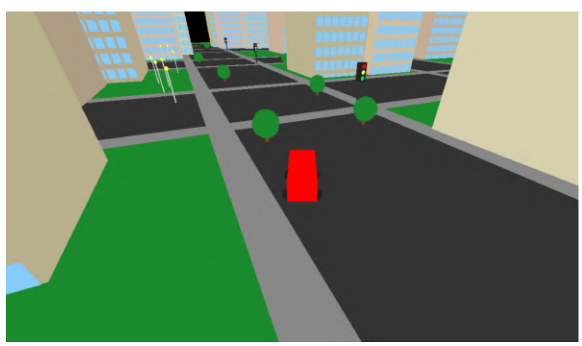

Fig. 7. Result of prompt 1.

**P2:** `Make the city look more realistic, add details like streetlights and traffic lights.`
**P3:** `Improve the materials and lighting to make the city more visually realistic.`
**P4:** `Add pedestrian elements, benches, and bus stops to the city.`
**P5:** `Add a realistic car to the scene.`

**P6:** `Make the car more realistic.`

**P7:** `Ensure the car faces the direction of travel, and moves using proper acceleration and turning logic.`

**P8:** `Add keyboard controls "WASD" to drive the car forward, backward, and turn left/right.`

Fig. 8. Prompt 8 result

**P9:** `Create realistic physics for braking and acceleration.`

**P11:** `Detect collisions with buildings or trees and stop the car.`

*Note: Some instructions had to be reformulated multiple times due to incomplete or flawed outputs, highlighting the trial-and-error nature of prompting and the importance of clarity and precision.*