Vibe Coding

On the power and danger of programming with AI

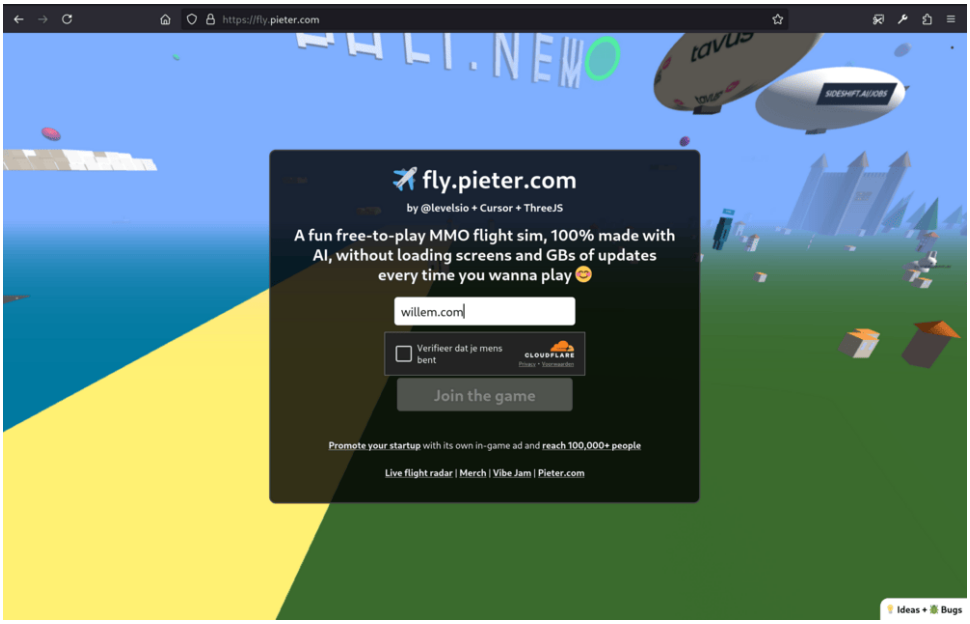
Willem L. Middelkoop
Apr. 15, 2025

Grossing over $48K a month, the flight simulator game by Pieter Levels spurred a whole flurry of innovation. Without detailed knowledge of 3D game engine technology, he 'vibe coded' his game using AI. Critics of his work pointed at security and scalability issues, while proponents lauded the amazing result. What can we learn from this?

Vibe Coding

Vibe coding is an AI-driven programming method where you describe a problem in a few sentences as a prompt for a language model tuned for coding. The model generates software, shifting the programmer's role to guiding, testing, and refining the output. Advocates claim it enables novice coders to create programs without extensive training or software engineering skills.

Pieter Levels created the game fly.pieter.com. On social media he kept an open loop of his efforts: from his initial idea all the way to publication. He attracted sponsors for in game ads and he sells premium upgrades (like special planes) in his game, grossing thousands of dollars each month.

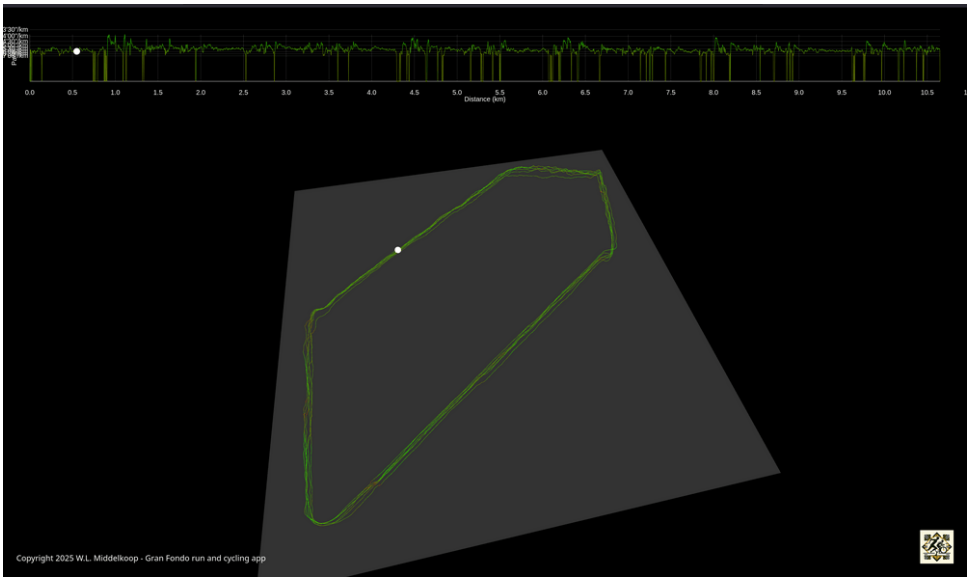


AI Generated Flight Simulator: fly.pieter.com

His success story raised questions about the future of game programming, or even programming in general. If one guy with a laptop can create a smash hit like this then what is the need for big budget studios? Or, asked inversely, what would happen to big budget studios if they would adopt a similar 'vibe coding' approach to production?

Trying it myself

Having some experience with developing a game myself (Snake 97, 40+ million downloads), Pieter's story inspired me to look into the 3D technology he used for his game: ThreeJS. Building a game is hard, doing it in 3D is even harder. The mathematics underpinning a world in three dimensions add a whole extra challenge. In many ways it is unlike ordinary programs or simple two dimensional games. Although I have done some experimentation before, I found the learning curve of 3D mathematics quite steep. This is where the AI can help.



3D visualisation of workout data

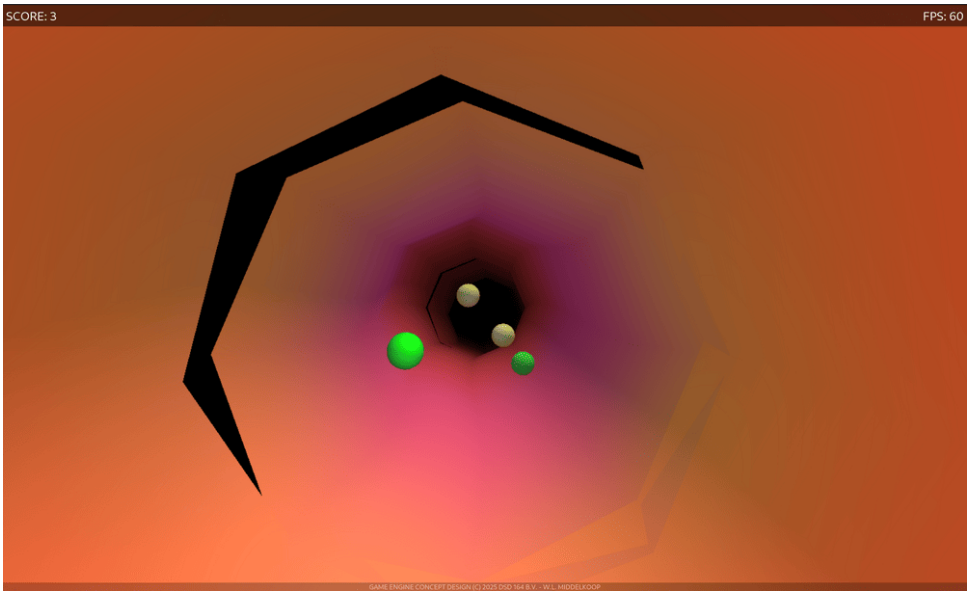
For the Gran Fondo app I used the AI to generate a three dimensional presentation of a recorded running activity. I formulated my goal in plain English and the AI-model generated the code needed to use ThreeJS to display my workout on a rotating pane. The initial results were not good, but I understood enough of the generated results to continue asking for specific refinements. Some hours later I had something I liked, but moreover I felt as if I have learned a great deal about the viability of my initial concept. The AI enabled me to learn faster.

Game Concept artwork from 2013: HDL particles in the blood stream

I decided to explore another idea that I had parked for some time: a game about bad and good cholesterol particles in a person's bloodstream. With aspects of an endless runner-style game, I figured this to be a good fit for some more experimenting as these

games feature computer generated levels. Building an automated level generator seems like something the AI would be able to help me with.

Leveraging the newfound 3D capabilities, I figured that the game should move into the arteries instead of lateral movement common for classic 2D games. This introduces challenges like visual clipping, distorted graphics due to the camera's position being outside of the 3D scene. Another challenge is increased complexity of the collision detection of in-game elements, a requirement for gameplay mechanics. The generated code from the AI model grew and grew with each iteration.



3D Game Engine in action, note the graphical glitches in the artery's wall

After some 800 lines of code I started to experience the limitations of the AI-model as it started to make errors and bugs that it had solved in earlier iterations. Every new iteration introduced new problems and the model kept forgetting refinements and fixes it had applied earlier. This is due to the limits on context capabilities (input tokens) that are imported for the large language models to work. Simply put: it can't remember enough to continue to iterate on the whole thing at once.

Productivity paradox: when operating the machine costs you more effort than the value of its output

I stopped developing the game engine using AI after some days of intensely experimenting. I reached a point were I was putting in more effort on managing the AI-model than effort directed at the actual game engine. I figured it would simply be better to get myself some books on the technologies, read them, become more proficient myself and then build the game engine by hand. When things become complex, our human brains seem to be able to focus on specific parts of a complex tasks while respecting the larger whole as it is.

On the foolishness of ”Natural Language Programming”

In addition to limits on the context that an AI model can comprehend, I think there is the problem of ambiguity. I recommend you read the words by Edsger Wybe Dijkstra, reflecting on the foolishness of ”natural language programming” in his EWD667. Unambiguous description, free of muddled thought, forms the bedrock of reliable code. Yet our native tongue leaves too much room for all kinds of nonsense and errors. Instead we should master our words, think sharply and specify boldly.

A formal language, like a programming language or a mathematical model or an exhaustive in and output specification, could be an amazingly effective tool for ruling out ambiguity and errors. Simply put: you'll get the best output when you give the best input.

Prompting the AI with code

To test if the AI would give me better code, I decided to change my language of asking. Instead of describing what I wanted in natural language, I uploaded a piece of code that I wrote earlier, with some very specific characteristics:

- the code came from a production system, I knew it worked well
- the code contained various error handling statements, adding to its robustness
- the code contained no syntax errors and used logically named variables and functions
- the code had a very specific scope, with clearly defined interfaces to the outside world

In addition to the code I specifically asked the AI to change one particular aspect of the code in question. It had to remove the usage of a certain external program (the NodeJS module 'lwip') and replace it by another external program (ImageMagick). Because both lwip and ImageMagick are open source code, the AI model is well aware of these programs. I gave it the perfect recipe for its cooking.

The code I got from the AI was a 99% perfect drop in replacement for my own code. The generated code had one bug that caused some parameters not to be escaped properly, leading to a crash. I could manually spot the error and fix it by hand, as I was fully comfortable and knowledgeable within the context of this specific piece of software. After some more testing I decided to take the AI-generated code in production as the new library performed much better than the one it replaced.

Conclusion

Vibe coding with AI enables fast prototyping and learning, as seen in Pieter Levels' success, but falters with complex projects requiring precision. It raises a question: does AI expand our potential or limit us to its constraints? Mastery blends human thinking with AI's power. Reflect on craftsmanship: when do tools sharpen your skill, and when might they blunt it?