Creating a 5000-line app for programming with algebra with Claude Code

Ken Kahn - home page

4 November 2025

Idea and history of the idea

Twenty years ago I joined the EU-funded REMATH ('Representing mathematics with digital media') project. Researchers from Greece, France, Italy, and the United Kingdom (where I've been for more than twenty years) explored new digital tools to help students explore mathematics. I lead the design and development of MoPiX. My idea was to create a programming language and associated programming environment to enable the creation of simulations, games, and animations solely by creating algebraic equations. Change of state was handled by explicit references to time. A user's program is just a function of time so it can run forwards and backwards easily.

MoPiX 1 was implemented in Adobe Flash. When it became clear that HTML5 would be better I implemented MoPiX 2. It was implemented in Java and automatically translated to JavaScript by the Google Web Toolkit.

Here is a paper we published about MoPiX.

MoPiX 3: Designed by me and Claude Code; Implemented by Claude Code

Claude Code for the web is available to me as a Pro subscriber (about $20/month). When I first make a request it spins up a virtual machine. It creates and manages files and does testing in this virtual machine. This way any bugs won't affect my files. Unlike prompts to the chat version of Claude, Claude Code can come up with multi-step plans and work on them one after the other.

I wanted to see what it was like to use Claude Code so I gave it access to the GitHub MoPiX 2 repository and asked it to make a modern version using JavaScript (and HTML and CSS). At first it created a folder with about two dozen files along with complex instructions for running it

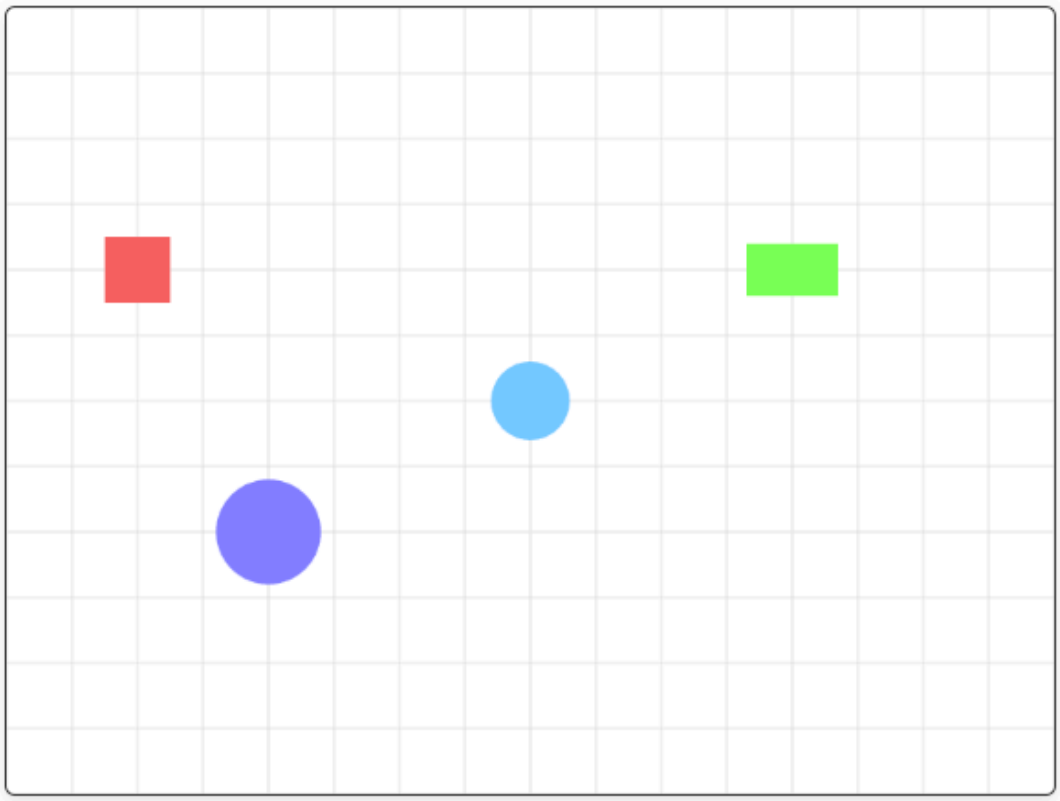
relying upon Node and TypeScript. I told it I wanted a single HTML file and it created this early version of the app:

MoPiX Modern - Enhanced Demo

Full-featured visual programming with trails, dragging, and persistence

Interactive Demo (Click, Drag, Draw Trails)

- **Click** objects to select them
- **Drag** selected objects to reposition
- **Toggle Pen** to enable trail drawing
- **Create** new objects with equations or static placement



Reset | Step Back | Play | Step Forward

Frame: 0

Speed: 30 fps | Clear Trails

Object Management

No object selected | Toggle Pen | Delete | Create Object

Save & Load

Save to Browser | Load from Browser | Export JSON | Import JSON

New Features Added

That was a good start. The objects move and spin and I can create new objects. But critically I can *not* create or edit equations. After a week we had this greatly enhanced fully functional app. While it took a week my typical interaction pattern was:

1. Request a change or changes
2. Read other things for 5 to 60 minutes
3. Skim the response and changes
4. Load the latest version
5. Test it for a few minutes
6. Repeat

I estimate I was working at most one fourth of the time on the new MoPiX. Claude also was active about the same amount of time. Three fourths of the time I was doing other things. I probably spent about 20 hours on this project.

The new version is over 5000 lines of code. Claude Code would update a branch on GitHub (along with detailed documentation of the changes) and I merged it into the master version over 80 times.

Here is the in-app documentation generated by Claude:

> 💡 **Available:**
>
> **Variables:** t (time), *user variables* (define in sidebar), *object name.property*
> **Constants:** pi, e
> **Properties:** x, y, width, height, rotation, red, green, blue, thicknessPen, redColorPen, greenColorPen, blueColorPen, opacityPen, *custom properties*
> **Input:** mouse.x, mouse.y, keyDown(key)
> **Previous Frame (t-1):** prev.x, prev.object name.property
> **Operators:** + - * / ^ (power), <, >, <=, >=, ==, !=
> **Math:** sin, cos, tan, abs, sqrt, ln, log, exp, floor, ceil, round, sign, fract
> **Logic:** if(cond, true, false), and, or, not
> **Collision:** overlaps(obj1, obj2)
> **Multi-arg:** dist, angle, min, max, clamp, lerp, random, mod, pow, atan2
> **Easing:** smoothstep(edge0, edge1, x), step(edge, x), ease(x, power), bounce(t), wrap(val, min, max)

Each item displays tool tips if the mouse hovers over it.

The programs are interpreted as equations where the only free parameter is *t* which is the time or the frame number. For example, to make an object named ‘ball’ move right at a speed of 5 units per tick you can enter

$$x = prev.x + 5$$

Which is shorthand for this equation:

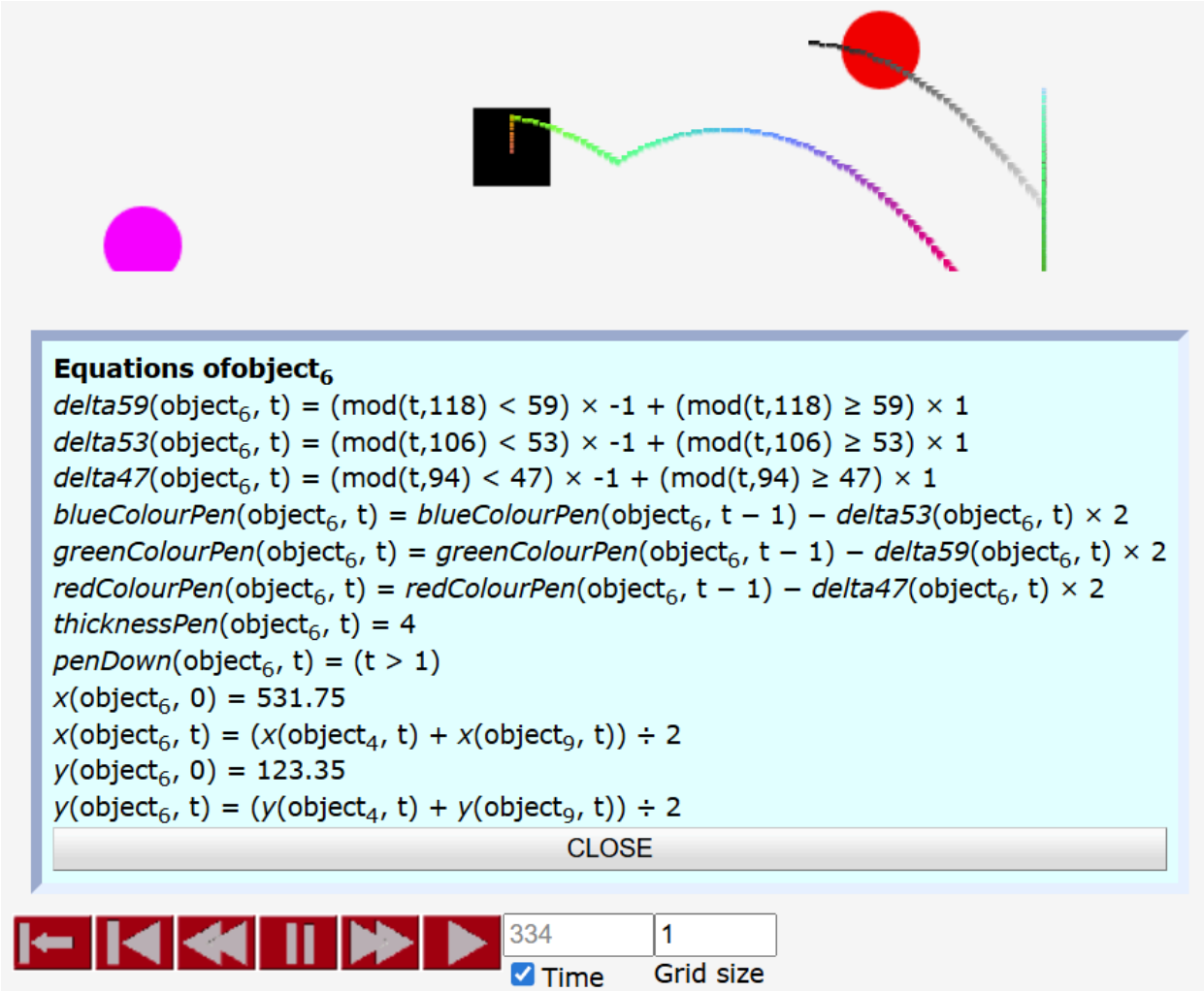
$$x_{ball,t} = x_{ball,t-1} + 5$$

Video demos on YouTube

- Sample objects demo - 52 seconds
- Four graphs demo - 47 seconds
- Average of two bouncing balls demo - 2 minutes 11 seconds
- Creating a clock demo - 2 minutes 50 seconds

The original MoPiX 2

This effort was much more than just translating or porting an old version of MoPiX to modern technology. Here is what the old version looks like:



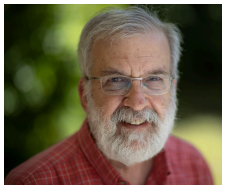
The new version has a great deal of new functionality.

The app

The app starts with sample objects. Documentation appears as needed. Try it out. If you create something neat send it to me at toontalk@gmail.com.

Here are two models I created that you import. A much better average of two bouncing balls than in the sample models and a clock.

About me



I've been researching AI, creativity, and education for fifty years. I started when working on my phd from the MIT AI lab. Most recently I wrote a book about how anyone can use chatbots to co-create apps, adventures, illustrated stories, and discussions.

The Learner's Apprentice: AI and the Amplification of Human Creativity

In the book I describe creating apps using the chat versions of LLMs. Here I explored how much more can be done with the code agent version. I didn't need to read or write any of the 5000 lines of code.

You can follow me on LinkedIn, Facebook, Threads, or BlueSky.