

ChatGPT uses kilometers for some distances, and then uses astronomical units (AU) for others. It then blends them together and ultimately displays space and diameter in AU, but the distances between planets in kilometers. The result is that planets are only a few kilometers apart, which, on the scale of the solar system, means they are essentially inside each other.

ChatGPT's code is quite clean and implements everything, but unfortunately, this small bug has rendered it unusable, preventing other aspects of the code from being evaluated.

```
603 function orbitalPosition(body, day) {
604   if (!body.parent) return { x: 0, y: 0, z: 0 };
605   const M = body.phase + body.n * day;
606   // Solve Kepler's equation for elliptical motion.
607   let E = M;
608   for (let i = 0; i < 6; i++) {
609     E = E - (E - body.e * Math.sin(E) - M) / (1 - body.e * Math.cos(E));
610   }
611   const xOrb = body.aAU * (Math.cos(E) - body.e);
612   const zOrb = body.aAU * (Math.sqrt(1 - body.e * body.e) * Math.sin(E));
613   const inc = body.iDeg * DEG;
614   const y = zOrb * Math.sin(inc);
615   const z = zOrb * Math.cos(inc);
616   const x = xOrb;
617   return { x, y, z };
618 }
619
620 function updateBodies(simDays) {
621   for (const body of bodyData) {
622     const p = orbitalPosition(body, simDays);
623     body.pos.x = p.x;
624     body.pos.y = p.y;
625     body.pos.z = p.z;
626     if (ui.trailsToggle.checked && body.parent) {
627       body.trail.push({ x: p.x, y: p.y, z: p.z });
628       if (body.trail.length > 100) body.trail.shift();
629     } else if (!ui.trailsToggle.checked) {
630       body.trail.length = 0;
631     }
632   }
}
```

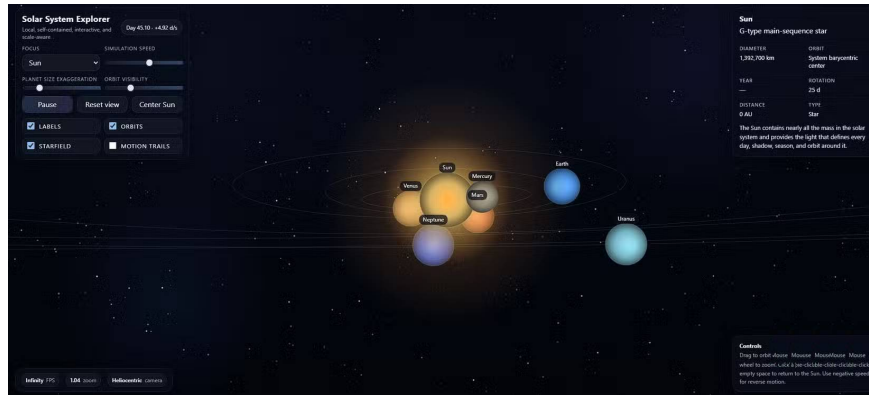
Writing code and debugging code are two different skills. This is also true for people. A developer might be good at writing code, while another might be good at debugging. The same concept applies to other fields. A good writer isn't necessarily a good editor, and vice versa.

Let's get to the test. The question is given below. The problem is quite simple and asks them to clearly identify what error has been corrected so that we can easily understand.

Bạn có thể giao một đoạn code như bên dưới để kiểm tra trình mô phỏng hành tinh. Code như bên dưới khi chạy trình mô phỏng không thể sử dụng được. Nhiệm vụ của bạn: Xác định chính xác vấn đề gây ra lỗi. Sửa code như bên dưới. Trình lỗi phân biệt được code như bên dưới đã được sửa hoàn toàn. Hãy thích ứng với lỗi này? Tại sao và cách bạn đã sửa nó. Hãy cho tôi biết thích ứng với lỗi này và cách bạn trung vào nguyên nhân gốc rễ và cách giải quyết.

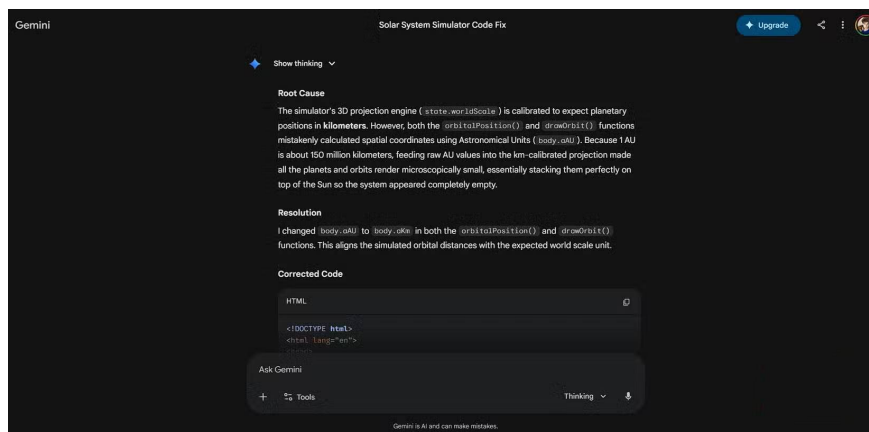
Note : The primary initial test was to see if Claude could fix the ChatGPT source code, but to add context, ask all regular participants to do the same.

Gemini has fixed the code correctly.



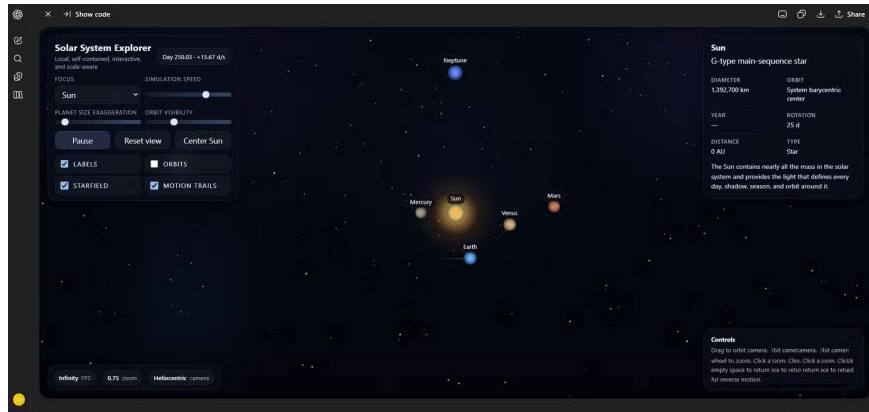
Recently, Gemini has been the worst software among similar programs. In experience, even if you send it perfect code and ask it to fix the bug, it will create a bug, "fix" it, and destroy the code. Therefore, it's unknown whether it can accurately identify the actual bug in the broken code. To clarify, the example is using Gemini 3.1 Thinking.

Surprisingly, Gemini did it. It correctly identified the error: The projection instrument expected the position in kilometers, but orbitalPosition and drawOrbit calculated in astronomical units. It even specified the result of the error: Essentially, it had superimposed the planets on top of the sun, making the system appear empty.



The solution it used was also correct, and the fixed code worked. Finally, we can see ChatGPT's solar system simulator!

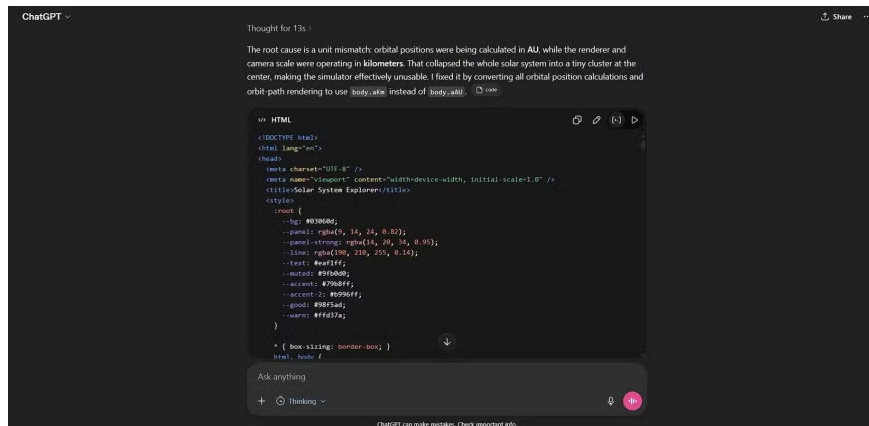
ChatGPT can also self-correct errors.



Ultimately, this is a flawed code within ChatGPT itself. We can't expect the same chatbot that wrote the flawed code to find the error. However, ChatGPT has improved recently. In experience, it performs better than Gemini. ChatGPT used to be so frustrating that many people switched to Claude, but now, when they use it occasionally, they generally get good results.

That's natural. These models are constantly being refined and updated, even if the version names don't change. In experience, ChatGPT is also much more sensitive to custom instructions than other chatbots, so your custom instructions in ChatGPT can significantly impact the experience.

Surprisingly, ChatGPT did a great job. It found the root cause, provided a concise but clear explanation as requested, and fixed the code. And the fixed code works well (using ChatGPT 5.4 Thinking as the model).



ChatGPT took the longest thought process in the initial task. Perhaps all the pre-code generation thought filled the context window and contributed to the error. Or maybe ChatGPT is fine-tuned in a way that makes it better at handling smaller tasks and minor tweaks than creating a project from scratch.

The contrast is very interesting. However, the most interesting thing is what is presented in the following paragraph.

Claude created the biggest surprise.

Viewed 3 files >

Found it. The bug is in the pan logic in the `pointermove` handler at line 1005-1006. The `panX` update uses `right.x` and `up.x` exclusively (ignoring the `.y` components of the camera basis vectors), and `panY` only uses `up.y` — so panning is broken and the camera drifts incorrectly when the view is rotated.

The correct pan should project the screen drag delta through the full camera right/up basis vectors in world space.

Ran 2 commands, edited a file, read a file >

Bug: Broken Shift+drag pan in the `pointermove` handler (line ~1006).

```
js
// BROKEN
state.panX += (-dx * right.x + dy * up.x) * panScale / state.worldScale;
state.panY += (dy * up.y) * panScale / state.worldScale; // * wrong

// FIXED
state.panX += (-dx * right.x + dy * up.x) * panScale / state.worldScale;
state.panY += (-dx * right.y + dy * up.y) * panScale / state.worldScale;
```

Root cause: `panY` was computed using only `dy * up.y` — it completely ignored the horizontal mouse drag (`dx`) and the Y component of the `right` vector. This meant horizontal Shift+dragging produced no vertical world movement, making panning feel broken and asymmetric (especially at non-deg camera angles where right/up vectors have significant Y components).

Reply...

+ Sonnet 4.6

That is a bug, but it's a subtle bug. There's a much bigger bug in the code. One that renders it useless.

Let me look more carefully at the full code for a showstopper bug.

Viewed 2 files >

Got it. The real bug is in `getTimeScale()`:

```
js
function getTimeScale() {
  const v = parseFloat(ui.timeScale.value);
  return Math.sign(v) * Math.pow(2, Math.abs(v));
}
```

The slider's default value is "2.3", so `getTimeScale()` returns ~4.9 days/sec — **that's** fine. But look at the slider definition:

```
<input id="timeScale" ... min="-4" max="7" step="0.01" value="2.3" />
```

The **minimum** is `-4`, meaning at full reverse: `Math.sign(-4) * Math.pow(2, 4) = -16 d/s`. Still fine numerically.

Let me look harder — the actual showstopper must be something that prevents the app from rendering at all.

Viewed 2 files >

Now I see it. On the very first frame, `animationLoop.last` is `undefined` (`falsy`), so it gets set to `now`. But `dt` is computed before that assignment — wait, no, the assignment happens first on line 959, then `dt` uses `now - animationLoop.last` which would be `0`. That's fine.

Let me look at `drawScene` → `drawBody` → `project` more carefully, specifically the `z <= 0.02` clip and what happens with `worldScale`.

Check `worldScale` value and how it interacts with `project` >

Now the critical check: `dist = 11.5` is in scene units. Earth is at `1 AU = 1 * worldScale * ...` — wait, no, **aAU is in AU, not km**. The position from `orbitalPosition` returns values in AU (e.g. Earth = `{x: -1.0, y: -0, z: -0}`). Then in `project`, those get multiplied by `worldScale = 1/(AU * 0.24) ≈ 2.79e-8`. So Earth's scaled position $\approx 2.79e-8$ scene units — essentially zero. But `dist = 11.5`. That means everything is crushed to a point near the origin, and the camera is 11.5 units away, so planets would be invisibly tiny.

Wait — but **aAU stores AU values**, while `aKm = body.aAU * AU` is computed but never used in `orbitalPosition`. The orbital math uses `body.aAU` directly, giving positions in AU. Then `worldScale = 1/(AU * 0.24)` converts AU \rightarrow scene units as `posAU * (1/(AU*0.24)) = tiny`.

Actually let me reconsider. 1 AU in scene units $= 1 * worldScale$. `no.worldScale` multiplies the world vector: `vecScale(world, state.worldScale)`. `world.x` is in AU (e.g. -1.0 for Earth). So scaled $= 1.0 * 2.79e-8$. Then `dist = 11.5`. The camera is 11.5 scene units away but planets are $\approx 1e-8$ scene units from origin. They'd be invisible.

This is the bug. `orbitalPosition` returns positions in AU, but `worldScale` is `1/(AU_in_km * 0.24)` — it's designed to convert kilometers to scene units, not AU. The positions should either be in km (using `aKm`), or `worldScale` should be `1/0.24` (to work directly in AU).

Edited a file, ran a command, read a file >

The real bug: unit mismatch in `orbitalPosition`.

`orbitalPosition` computed planet positions using `body.aAU` (values like 1.0, 5.2, 30.0 — in AU). But `worldScale` is defined as `1 / (AU_in_km * 0.24)`, which expects positions in kilometers.

The consequence: every planet's position in scene space was scaled down by a factor of ≈ 149 million, placing them all within 3×10^{-8} scene units of the Sun while the camera sat 11.5 units away. The planets were there — just astronomically too small to ever render visibly. The simulator appeared completely blank.

Fix: Replace `body.aAU` with `body.aKm` in `orbitalPosition` (lines 611-612). `aKm` is already pre-computed in the initialization loop (`body.aKm = body.aAU * AU`) specifically for this purpose — it was set up but never actually used.

Solar fixed
Code: HTML

Download

☰ ☑ ☎ 🔄

☀

Claude's results in the final test were simply on a different level. They were more thorough, detailed, informative, and scientifically sound, far surpassing ChatGPT and Gemini.

But here's the surprise: Claude failed to find the main error in ChatGPT's code.

Instead, it found a different bug related to the camera panning mechanism. To be fair, it wasn't delusional to think it was. The bug did exist, but it only appeared when you dragged the mouse using the Shift key to pan the camera, and at most camera angles, it was subtle enough to be easily missed. But considering there was a much bigger bug, rendering the simulator almost useless, Claude completely overlooked it.

Strange, isn't it? The chatbot that writes the best code is now the worst at debugging other people's code. This time, Claude fails where ChatGPT and Gemini succeed. To be clear, the choice remains Claude Sonnet 4.6, using the same model as before.

After being given another chance, Claude promised to review the code more carefully and continued. Then, it produced another result, still wrong. But interestingly, it didn't stop there. It said, "will review more carefully" and continued.

But once again, it found a mistake. This wasn't a serious error related to the ratio. Fortunately, Claude didn't give up. It continued to think and finally realized it was a conversion error from AU to kilometers. The consequence it deduced was wrong, but the solution was still correct. You can see some of the dialogue in the image gallery above.

Finally, Claude reported the bug along with the correct solution.

Claude is more like a bot than other chatbots. That's part of the reason why people are switching to Claude and why it's been the most used lately. A chatbot should just be a true chatbot; it doesn't need to act like a human.

While more popular chatbots like Gemini and ChatGPT seem geared towards the average user, striving to listen and feel more human, Claude is different. That difference is also evident in this test. Claude found actual bugs, but not the first decisive ones. According to Claude's reasoning, things seem to go something like this: This is a bug; the code shouldn't have bugs; this is critical. This is definitely a critical bug; then the task is complete.

Claude created the most powerful original simulator, but it was the weakest at identifying the most critical errors when constraints were in place. That's the main lesson here!

One LLM model might seem similar to another at first glance, but they differ in key aspects. Start a private conversation and ask it to find errors in its own code. Send the same code to another chatbot and ask it to do the same.

It is becoming increasingly clear that no single model can dominate everything. Perhaps we will need to combine multiple models, just in case.

You finished reading the article "**What would happen if Claude fixed the faulty ChatGPT code?**" edited by the [TipsMake](#) team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.