Guiding Light

Design Journey

The decisions I made, the walls I hit, and the choices I changed my mind about.

Why this document exists

Guiding Light's design documents describe what the prototype is. This document is about how I got there. Every decision in those documents was made under real constraints, with real wrong turns, and several of them only landed after I had built and discarded something else.

I'm writing this because I think the design decisions in a prototype are more interesting when you can see what was considered and rejected, not just what shipped.

The starting idea

I wanted to build a 2D platformer that argued one specific thing: visibility should not be free. In most platformers, the player sees the world by default. That assumption seemed worth overturning.

The first version of the mechanic I sketched was a flashlight with a finite battery. Then I reframed it as a torch with oil. The shift mattered: 'oil' carries emotional weight that 'battery' doesn't. Oil suggests a precious resource, a glowing flame, a thing the player would feel anxious about burning. The mechanic didn't change. The framing changed how I designed everything around it.

The mechanic decisions that took multiple tries

Hold-to-light vs toggle

I initially built the torch as a toggle — press Z to light, press again to extinguish. It felt clean in the editor. But when I tested, I realized I was forgetting the torch was on. I'd be navigating in light, focused on the platforming, burning oil passively without choosing to.

Switching to hold-to-light made every second of light an active decision. The mechanic didn't change; the player's relationship to it did. That distinction — between passive and active resource consumption — became something I now think about for any meter or timer system.

Continuous oil across levels

This was the riskiest decision. Most platformers reset resources between levels. I designed Guiding Light to carry oil through all three levels with no checkpoint refills.

I almost backed out of this. It's harsh. A player who burns oil recklessly in Level 1 makes Level 2 nearly impossible. I worried it would feel unfair. But the alternative — resetting oil each level — made the economy local rather than systemic. The decision to spend felt small, because the consequence didn't reach far.

I kept continuous oil. After playtest, the player feedback I cared about was someone saying, 'I knew I shouldn't have used so much light in Level 1.' That's the design landing. They felt the consequence of an earlier choice. That sentence justified the harshness.

Auto-refill on death

Early in development, dying refilled the player's oil to full. This was a comfort decision — it made the prototype more forgiving. But I noticed in self-playtest that I started using death as a recovery mechanic. I'd burn oil freely, die deliberately, and reset with full oil.

That removed the oil economy entirely. I changed death to preserve the player's oil at scene-entry state. Dying still resets the scene, but the resource budget is real. The discipline of the economy required this. Comfort was the wrong design instinct.

Level decisions I reversed

Level 2's structure

My first sketch of Level 2 had the player walking right-to-left, with the EndLevel near the start position. When I looked at it after a day, I realized I'd labeled it confusingly — the 'level end' marker was ambiguous, and I hadn't decided whether the lower path or the upper path was the main route.

I rewrote Level 2 as a loop: lower path right, climb up, upper path back to EndLevel. The same physical space, but the player traverses it twice — once learning stalactite timing, once testing it as a wave pattern. The loop structure came out of having to clarify ambiguity in my sketch. The clarification was the design.

The Level 3 platform-when-dark mechanic

I considered cutting this twice. The idea that monsters become solid platforms in darkness was central to my pitch for Level 3, but it required teaching counterintuitive behavior. I worried players wouldn't discover it.

The first time I considered cutting it, I argued myself out by promising to design the teaching moment carefully. The second time, I considered adding a text prompt ('Monsters become solid in darkness') as a safety net.

I rejected the text prompt. My locked design principle was 'level design teaches the mechanics, not text.' Adding text would have undermined that principle in the one moment it mattered most. Instead, I designed the Twist beat as a forced encounter — a monster in the only path forward, wide enough that any dark-jump attempt would land on it. The discovery would happen through necessity, not narration.

Playtest validated this worked. Players discovered the mechanic within 3-5 attempts. Some took longer. But every player eventually understood it through play, which was the design intent. Adding text would have stolen that moment from them.

Things I built and then changed

The camera

I spent more time on camera behavior than I expected. Initially I built a lock-on camera that always centered the player. Then I switched to a held-start camera that doesn't move until the player crosses center, then follows bi-directionally. That felt right for Level 1.

Level 2 has an upper path that's vertically separated from the lower path. I assumed I'd need Y-axis camera follow. I built it as a two-axis lock-on system. Tested it. The camera bobbed during jumps in a way that fought the player's ability to memorize layouts during the brief torch-on window.

Instead of redesigning the camera, I redesigned the level. I pulled the upper path closer to the lower path so both fit in the camera's existing X-only frame. The mechanic I'd built (Y-axis follow) became dead code. The fix wasn't more code; it was less geometry. I should have tested geometry adjustments before building camera systems.

This is the iteration I think about most. I almost shipped a complex camera I didn't need. The lesson was scope discipline: test the simpler fix first.

The death sequence

I designed a 'player vanishes for a brief pause, then scene reloads' death. The pause was 0.3 seconds. I tested at 0.0 (too jarring), 0.3 (clean), and 1.0 (too slow).

I also tested whether to play a death sound. Original design didn't have one. I added one after playtest revealed that without audio feedback, sudden deaths felt confusing — players asked 'wait, did I die?' A short death sting confirmed the moment and gave it weight.

Small design decision, but it taught me that audio is a feedback system layer, not a polish layer. The death sound exists because without it, the moment of death was ambiguous. Removing it would remove information the player needs.

Things I cut

Two-intensity light

I sketched a version of the torch with two brightness levels — dim (slow oil drain) and bright (fast drain). The idea: give the player more granular control over their resource.

I cut it before building. Two brightness levels would give the player a permanent middle-ground option, eliminating the discrete decision moments that make the binary system feel tense. The mechanic's emotional power was in the commitment of either/or, not in the granular control of more/less.

Save/load system

I considered adding session-persistent saves. Cut it before building. A 15-minute prototype is experienced in one sitting. Saves would have added build complexity without strengthening the design argument.

This was a scope discipline call. Every feature that doesn't strengthen the design argument is a feature that dilutes it.

Animation

I planned a full animation pass — walk cycles, jump arcs, idle states. Cut it after building the static sprite version and playtesting. The mechanic landed without animation. Adding animation would have added time without changing the design argument.

This was the cut I almost regretted. Static sprites look like 'prototype.' Animated sprites look like 'game.' But I had to commit to the prototype reading honestly. Adding animation polish would have changed what the prototype was. The static version is what the prototype is.

Patterns I noticed in my own design thinking

I want to add things more than I want to cut things

Multiple times during development, I proposed adding a feature (text prompts, save system, difficulty modes, intro flashing text after multiple deaths). Each time, I had to argue myself back to the simpler design. The instinct to add is strong; the discipline of cutting is harder.

By the end of the build, I had a rough rule: every feature has to either strengthen the design argument or it's noise. The prototype has fewer features than my first sketch by design.

I trust my paper sketches more than my Unity instincts

Twice, I tried to skip the paper-sketch step for a level (Level 2 first try, Level 3) and started building in Unity directly. Both times, I hit structural problems mid-build that would have been visible on paper in five minutes.

The lesson: paper design isn't a formality. It's the cheapest place to catch structural problems. Skipping it costs build time that's harder to undo than redrawing a sketch.

I value playtest evidence over abstract reasoning

Several decisions changed because of playtest, not analysis. The oil drain rate. The stalactite warning duration. The monster speed. I could have argued for any of those values from first principles. But the playtest data overruled my arguments every time.

This was the part of game design I hadn't fully internalized before this prototype: you don't design tuning values. You design ranges, then test, then commit. The committed value is data, not preference.

If I were starting over

I'd skip the camera Y-axis exploration entirely and design Level 2's geometry to fit a single-axis camera from the start. I'd lock the torch input scheme (hold-to-light) before building anything else, instead of switching mid-development. I'd commit to no auto-refill on death from day one, instead of building a comfort version and then removing it.

But each of those wrong turns taught me something I now use. The camera detour taught me to test the cheapest fix first. The toggle-vs-hold detour taught me to distinguish active from passive consumption. The auto-refill detour taught me that comfort can be the wrong design instinct.

If I had started over with all those lessons, the prototype would have been faster. But I would not have known why those lessons were lessons. The wrong turns were the price of the understanding.

The shipped prototype

Three levels. One design argument. ~3 weeks of work, mostly solo. Static art, no animation, no save system, no difficulty modes — every absence was a choice.

The prototype is winnable in 12-15 minutes by a player who understands the rationing mechanic. It's not winnable by someone who plays it like a normal platformer. That distinction is the prototype landing as designed.

It is not a complete game. It is an argument made playable.

On collaboration

Programming was done in Unity 6.3 with C# scripting assistance from Claude (Anthropic's AI). Every design decision — what to build, what to cut, what to tune, how to teach mechanics, how to structure levels, what playtest feedback to act on — was mine. The dialectic was collaborative; the design accountability is solo.

—

Pratik Jadhav. Solo prototype, ~3 weeks. Unity 6.3, WebGL.