CODENAME: SNOOPY

Design Journey Document

Pratik Jadhav — Senior Game Designer

15 June 2026 – 5 July 2026 | 20 Days

Why I Built This

I have been designing F2P mobile games for 7.5 years across 9 shipped titles. I am good at retention hooks, monetisation gates, and engagement loops. I am bored of them. Codename: Snoopy started as a deliberate exercise in designing the opposite of what I know: a game with no IAP, no daily login, no FOMO mechanic — just a player, a city, and a decision they cannot take back.

I wanted to build something that could sit in a portfolio as evidence that I understand how systems create emotion — not just how systems create retention. That is a harder thing to demonstrate. This was my attempt.

Week 1: Building the Foundation (15–21 June)

What went right

The core architecture decisions made in week one proved to be correct. One scene. CSV-driven dialogue. JSON persistence. NavMesh roads only. A game clock with a single public time scale variable that everything else derived from. These decisions cost nothing to make in week one and would have cost enormous rework to change in week two. I have shipped enough games to know that foundation decisions compound — getting them right early is the most valuable thing you can do.

The first wrong turn: Jac's movement

Jac was originally designed to walk to road stops, get in his car, drive to the destination, get out, and walk to the building. This added NavMesh agents on Jac himself, complex walk-to-point coroutines, and a dozen edge cases around animation states and collision. After two days of debugging, I made the call: Jac has no NavMesh agent. His car drives. He teleports. The player never sees Jac walk — only the car moves, and Jac appears inside the building when it arrives. This decision removed approximately 300 lines of fragile code and introduced zero visible change to the player experience. It was the right call and I should have made it on day one.

The opening sequence

The title screen, loading screen, prologue, and camera opening sequence were built in sequence and each handed off to the next via event callbacks. This felt elegant in design but created a brittle chain: a failure in any one step could silently prevent everything downstream from firing. In retrospect, I would add explicit debug logging to every handoff point from day one. During the prototype I spent significant time hunting for which link in the chain had broken during regressions.

Week 2: The Systems Build (22–28 June)

The save system: what I got wrong first

The first version of the save system saved data in a custom path I had redirected during development. When the LocalLow folder did not exist on the machine, the save silently failed. The fix was Directory.CreateDirectory() — two lines — but finding the problem took longer because the failure was silent. Lesson: every save operation needs a success confirmation log. Every load operation needs a fallback log. Silent failure in persistence is the most invisible bug category in game development.

The tracker system: two wrong approaches

The tracker went through three designs before landing on the final version. First design: a cylinder trigger that automatically started the timer when Jac entered it. Cut because it removed player choice at the critical moment. Second design: a detection range that blocked all player movement while the timer ran. Cut because disabling controls during the approach phase — before the player had committed — felt punishing rather than tense. Final design: ring appears on proximity, button appears when Jac enters inner range, player taps to begin timer, movement disabled only during the fill. Each iteration was an improvement in player agency.

The NPC system: scope creep I caught

At one point during the NPC conversation system build, I had designed Collie as an active character who could respond to every notebook entry the player brought him, spinning each piece of intel against Jac. This would have required a separate dialogue tree for Collie with 69 response entries keyed to conversation IDs, a Collie location system tied to the time clock, and a separate UI flow distinct from the building pie menu. I cut the entire Collie conversation system in week two. The design was correct — it would have been the best feature in the game — but it was not achievable in the remaining time without compromising the quality of the systems that were already built. Knowing when to cut is a design skill. I am proud of this cut.

Week 3: Polish, Bugs, and the Story Problem (29 June – 5 July)

The story changed twice

The original story had Jac as fully guilty — he ordered the bombing, Collie was simply using Bon to remove him. The epilogue was a double-cross: Collie sent the cops after Bon after getting what he wanted. This ending worked mechanically but had no emotional weight — eliminating a genuine bomber and then getting betrayed by the person who hired you is a heist-movie twist, not a moral reckoning.

The second version made Jac fully innocent — Collie had framed him completely. This required explaining how Collie had placed Roven on Jac's private yacht, planted documents in Jac's internal files, and coached 23 independent witnesses across 7 locations — all without anyone noticing. The plot collapsed under its own weight.

The final version resolved both problems: Jac is genuinely connected to Roven and Aldren Holdings — but as a willing participant in what he believed was a legitimate property development deal. Collie introduced Roven to Jac, structured the operations under Jac's name, and used Jac's genuine business activities as the frame. Jac is not innocent in the sense

of being uninvolved — he is innocent in the sense of not knowing what he was part of. The NPCs tell the truth. The evidence is real. And Bon kills a man who was trying to find the same answers Bon was looking for. The story required no dialogue rewrites — only a shift in what the epilogue reveals.

The UI blocking bug

Intermittent UI blocking — buttons stop responding for no apparent reason — appeared three times during the final week. Each investigation eventually traced to the same root: a boolean flag or an event subscription that was set during one game state and not cleared during a transition to another. The pattern: PieMenu's isOpen flag stuck true after manual Inspector editing during Play. BonMovement disabled by PieMenu and never re-enabled because HideMenu() was bypassed. Risk button container reactivated by DialogueSystem overwriting NPCInteractionManager's state. The fix in each case was the same: clear flags on every state transition, not just the expected ones. I added explicit state-reset calls to every system that had a boolean mode flag.

The energy system: why it was cut

The energy bar was built, tuned, and then removed from active gameplay. The design was correct in isolation: deplete over 18 game hours, restore with 6 hours of sleep, sleep during Jac's downtime. The problem was sequencing. The mechanic only creates interesting strategic decisions once the player knows Jac's schedule well enough to plan sleep around it. In a first-time 7-day prototype, the player is still learning the schedule during the exact window the energy system would punish them for missing. The bar remains in the scene as a deactivated object — visible to portfolio reviewers as an intended mechanic, invisible to players as a constraint they cannot yet engage with meaningfully.

The zoom system: three iterations

Continuous zoom with Lerp smoothing created jerk on stop — the camera overshot the target ortho size and snapped back. Switching to MoveTowards eliminated overshoot but with zoomSmoothSpeed at 650 moved so fast it was effectively instant. Removing smoothing entirely (direct snap) eliminated jerk but felt mechanical. The final solution: five discrete zoom levels with MoveTowards animation between them. One scroll tick per level. The player cannot land between levels so there is no jerk — the camera always settles at a defined state. This was the right solution and I should have considered discrete steps from the beginning rather than trying to make continuous zoom feel smooth.

On Working With AI

This prototype was built with Claude (Anthropic) as a coding assistant. All game design decisions — systems architecture, mechanical design, tuning values, story, NPC dialogue, world design, cut decisions — were made by me. Claude wrote and rewrote C# scripts based on my specifications.

Working with an AI coding assistant across a 20-day build session has a specific failure mode: incremental changes across many sessions can cause drift between what is in the code and what the designer remembers designing. Scripts get rewritten to fix one bug and break another system in the process. Variables are renamed. Methods are restructured. The AI does not always flag what changed — it shows you the result without the diff.

The discipline that made this workable was insisting on full scripts rather than partial edits. Every change produced the entire file. This made regressions visible — if a full script removed a method that existed in the previous version, I would catch it in the Unity console rather than discovering it six sessions later. It also meant I could read every script and understand what it did, because I had seen every version of it.

The AI assistant also challenged design decisions — sometimes correctly, sometimes not. When it raised a legitimate concern (the energy system's timing problem, the plot hole in Jac's full innocence), the challenge improved the design. When it raised concerns about decisions I had already tested and confirmed (story direction, mechanic scope), it wasted time. Learning to distinguish between useful pushback and redundant friction is a skill I developed across this build that I did not expect to need.

What This Build Taught Me

- **Scope cuts are design decisions.** The Collie conversation system was a better feature than anything I shipped. Cutting it was the right call. I can explain why both things are true at the same time.
- **Mechanics only work in context.** The energy system is correct game design for a player who has played for three days. It is the wrong game design for a player on day one. The same mechanic has different value depending on when the player encounters it.
- **Story is a system.** The epilogue twist only works because the NPC dialogue system was designed to make Jac look guilty from every angle. The narrative and the mechanics were designed together, and they reinforce each other. In F2P I design mechanics and leave the story to writers. This build showed me what I miss by doing that.
- **Silent failures are the most expensive bugs.** Every silent failure in this build — save system, event chains, flag states — cost more debugging time than any error that threw a console message. Explicit confirmation logs are not optional.
- **I design feelings, not mechanics.** Every system in this game exists to produce one specific feeling at one specific moment: the player looks at their notebook, looks at the clock, and decides to act. Everything else is in service of that moment. That is what I want to keep building toward.