

Roblox Survival Prototype

Examples from the project memory and skills that guided Claude Code and Roblox Studio

Craig Stevenson | mysorethumbs.com | Generative AI project 2 of 3

Overview

A multiplayer survival prototype built in about a week: a school field trip crashes on an island, and players choose between cooperating and turning on each other. Claude Code worked inside Roblox Studio through MCP. Earlier work had been done with Studio's built-in assistant, which shares no history with Claude Code, so I had Claude write a project memory file as the bridge, then kept it current as the game grew.

Instruction files	
CLAUDE.md	Onboarding and project memory: concept, every system and where it lives, bugs found and fixed, and known inconsistencies.
roblox-tools-gathering	A skill holding the build history for tools, gathering, caves, and map layout rules.
roblox-weather-system	A skill for weather, temperature, lightning, tornadoes, and the debug commands.
roblox-animals	A skill for campfire cooking and the animal pattern, one species at a time.
roblox-structures	A skill for shelters, ownership, combat against structures, and the placement preview.

Example 1: A project memory that says what is true and what is stale

The memory file separates intent from implementation, and records its own corrections so a later session does not trust a stale number.

```
Memory is the source of truth for intent, this file for implementation
state. ... Everything is server-authoritative; the client is a thin UI
and input layer.

Map: 400x400 studs, 16 chunks (corrected 2026-09-02: this had incorrectly
said 1600x1600 since the earliest onboarding pass, stale from before the
chunk size was tuned down).
```

It also states plainly that the build is a disposable mechanics prototype with no persistence, so nobody mistakes it for a shippable product.

Example 2: Skills keep the main file small

As the project grew, the memory file filled with long build histories. I had them moved into four topic skills that Claude loads only when working on that area. Each section in the main file keeps a short summary and a pointer.

```
Full build history moved to the roblox-tools-gathering skill (invoke it
when working on the Cave chunk template, gathering tools, or map-layout
rules) - summary: ChunkTemplates.CaveChunk was reworked from an open-air
lot into an enclosed cavern ...
```

This keeps every session's starting context small and cheap while the detail is still one step away.

Example 3: Bugs recorded with their root cause

Each bug entry says how it appeared, what the real cause was, how it was fixed, and how the fix was verified live.

Symptom	Root cause and fix
"No map generates"; the player falls to a default floor.	The map was fine. The character spawned before the map existed. Fixed by disabling automatic character loading and spawning each player only once the map is ready.

Symptom	Root cause and fix
Dying did nothing; the player never respawned.	Disabling auto-load also disabled Roblox's respawn timer. Fixed by hooking the Died event and loading a new character after a 5 second delay.
Falling through the map after the /newmap debug command.	The old map was destroyed under standing players. Fixed by reloading every character after regeneration.
Starvation and thirst damage seemed not to work.	Roblox's default health-regen script was silently cancelling the damage. Found only because precise rate measurements disagreed with the numbers. The script is now stripped on every spawn.

Example 4: Verification, and honesty about what was not verified

The instructions record how each system was checked live in Studio, and are explicit when something was not.

Live-verified: forced a death, confirmed the Humanoid state was Dead, waited past the delay, confirmed a genuinely new Character instance existed with a fresh Humanoid.

Not live-verified: an actual 20 percent Bear spawn never came up during testing, so the chase, leash and turn-to-face math are unverified against this mesh. Verify live the next time a Bear actually spawns.

One bug (an ore vein dropping wood) was caught by my own manual play rather than by Claude's automated pass, and the file says so. Another note warns that testing from the script console runs in an isolated context, so firing the real remote event is the only way to prove a full flow works.

Example 5: Design decisions I made, written as rules

- **Build one creature at a time.** Each new animal (birds, fish, wolves, and others on the list) mirrors the pattern the Rabbit established.
- **Raw food should work immediately; cooking should be a real upgrade.** This became the consumable design for berries and water.
- **Armor reduces all damage.** Damage from hunger, weather, lightning, tornadoes, and animals now routes through one function, so a single armor value applies everywhere. One exception is deliberate: a tornado throw at critical health is unsurvivable regardless of gear.
- **Cooperate or betray.** Shelters have owners and can be attacked and destroyed with partial refunds, the first real target for the betrayal pillar.

What I learned

- **Give each AI tool a shared memory.** A written bridge file let a second tool pick up work the first one started.
- **Split memory by topic.** Short summaries in the main file and long histories in skills kept sessions cheap.
- **Write the root cause, not just the fix.** Several bugs shared one underlying cause, and the notes let Claude spot the pattern.
- **Record what is unverified.** A clear "not tested yet" is more useful to the next session than a confident guess.
- **A prototype exists to answer a question.** Keeping it disposable let me test the cooperate-or-betray tension quickly.