Rampant Spark

Examples from the instructions that guide Claude Code on a Unity cyberpunk arcade dodger

Craig Stevenson | mysorethumbs.com | Generative AI project 1 of 3

Overview

I designed Rampant Spark and directed Claude Code, connected to Unity through MCP, to do the engineering. Because Claude has no memory between sessions, the project runs on written instructions: a project memory file, a roadmap, a design brief, and a read-only balance-review agent. The excerpts below show how those files steer the work. They are condensed and lightly redacted.

Instruction files	
Project brief	The design intent: a casual, approachable cyberpunk arcade dodger for people new to the genre, played with one finger.
CLAUDE.md	Project memory: game structure, rules, vocabulary, and the session ritual Claude follows every time.
ROADMAP.md	Milestones, a "Next up" list, parked ideas, open questions, and a dated session log written at the end of every session.
balance-reviewer	A read-only reviewer agent that analyzes balance across 30 levels and returns a ranked report with numeric proposals.

Example 1: A session ritual that survives memory loss

Every session starts cold. The ritual makes the roadmap, not the chat history, the source of truth.

```
Session ritual
1. Start: read ROADMAP.md ("Next up" + "Session log"), then git status.
2. Work on one milestone item at a time; commit in small logical commits.
3. End: tick finished items in ROADMAP.md, write a dated Session log
entry (what changed, what is broken, what is next), and commit.
```

The roadmap also records design decisions and rejected ideas, so a later session does not reopen a question I already answered.

Example 2: A reviewer agent that cannot change anything

The balance-reviewer is defined in a small markdown file with a name, a description that says when to use it, a restricted tool list, and a cheaper model. It never edits files; it proposes changes and I apply them.

```
---
name: balance-reviewer
description: Reviews game balance ... Use proactively when tuning enemies,
rosters, hazards, power-ups, health or chests ... Read-only; returns a
ranked report with numeric proposals.
tools: Read, Grep, Glob, Bash
model: sonnet
---
You are read-only. Never edit or write files, never run Unity, builds or
tests. Use Bash only for read-only commands. Propose changes; the user
applies them.
```

Choosing a smaller model and a narrow tool list keeps the reviewer fast and inexpensive to run often, which matters on a limited usage plan.

Example 3: Giving the agent my design goals and facts it must not get wrong

The agent is told who the players are and what fair means, then given the facts Claude previously got wrong.

```
The testers are mostly children and family, so the bar is approachable
and fair, not challenging. A death should feel like the player's mistake,
never the game's.

There is no avatar speed in the default input. Spark follows the finger
directly with zero smoothing, so "enemy speed versus player speed" is
meaningless. Compare enemy speeds and telegraph times against reaction
time, the hit radius, and the field size.
```

Design rules are stated as constraints, so proposals stay inside the game I am making:

```
Difficulty scales by enemy COUNT, spawn rate and speed, not by new
archetypes. No monetization and no meta-progression in the MVP. Red means
danger. Do not propose new enemy archetypes; if a problem can only be
solved that way, raise it as an open question for the user.
```

Example 4: A shared vocabulary

The codebase used the word "wall" for several unrelated things, which caused confused reports. The agent now has to use fixed terms.

- **Pinball shapes:** the solid wedge, bar and diamond obstacles that World 3 enemies bounce off.
- **Arena squeeze** and **corner cuts:** modifiers that close in the arena, not obstacles.
- **Ring barrier:** the level 30 boss gate, made of 10 segments; the player needs only one gap, not to destroy it all.

Example 5: Evidence rules and honest reporting

The agent must prefer real playtest data over its own arithmetic, and must say how sure it is.

```
Treat real data as ground truth over your own arithmetic. Report sample
sizes (runs and testers per level) and say when a level has too few
attempts to conclude anything. Anything drawn from the synthetic sample
logs must be labelled "synthetic, not evidence".

Each finding: severity, the claim, evidence with file:line for every
number, and a confidence label: Computed, Inferred, or Needs playtest.
```

It also has to say what code cannot tell it (how the game feels, finger occlusion, readability on a phone) and put those items on a playtest checklist instead of guessing. If a value cannot be found, the instruction is to say "not found" and never guess.

Example 6: Narrative authored in Twine, driven by tags

I write the story in Twine. Claude built the Unity side that reads the passages, and tags in the text drive typewriter timing, effects, audio and camera. The roadmap records each narrative item as it ships. One example from the log is a pause tag I can type directly into a passage:

```
keep me company{pause:2} or are we

{pause:seconds} freezes the typewriter for an authored number of seconds
mid-line. Rich-text tags do not count toward the position, and a
tap-to-skip during typing skips a pending pause too.
```

What I learned

- **Write down the decisions, not just the tasks.** The roadmap's rejected ideas and open questions saved as much time as its to-do list.
- **Constrain the reviewer.** Read-only tools, a cheaper model, and a fixed output format made the balance agent safe to run often.
- **Teach the agent the facts it keeps getting wrong.** Each correction became a line in the instructions, so it was not repeated.

- **Keep the design vision with the designer.** The agent proposes numbers; I decide what the game should feel like.