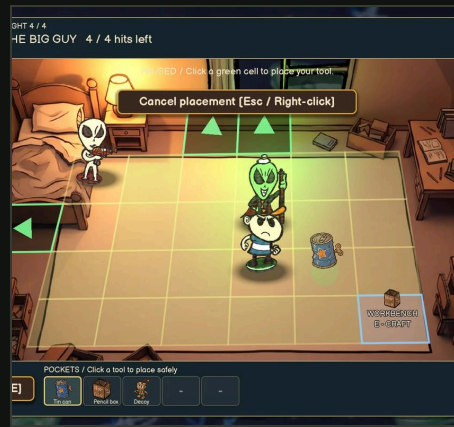


Christian D'Albano

Gameplay engineer and game designer. Four shipped games in three engines, designed and programmed by me, most of the art drawn by me. This deck is about the rules underneath them: what each game asks the player to do, how I found out it was wrong, and what I changed.



ThunderBashers, Unity, live on Wavedash with online rooms



I Won't Be Abducted, Godot, live on Wavedash



Undead Signal, Unity, live on Wavedash



Spare Knight, Phaser, live on itch.io; desktop release in progress

Five things I believe about games, each learned by getting it wrong first.

4

shipped games, all playable in a browser

Readability is a mechanic.

When a rule feels wrong it is often a legibility problem wearing a rules problem's clothes. ThunderBashers' core loop was fixed with a colour, not a rule.

The player's kit stays constant. The world moves.

In I Won't Be Abducted every difficulty number lives in a resource file and the boy never changes. Tune with numbers until numbers stop helping, then change one rule.

3

engines shipped: Unity, Godot, Phaser

Rules before code, and keep the dead drafts.

Undead Signal spent five days as written rules, three of them thrown away. The graveyard is what proves the final ruleset was chosen, not stumbled into.

Decide with a look test, not a spreadsheet.

A painted knight beside a 3D one under the real light ended a two-month engine argument in one evening. Write the decision down so it stays decided.

300

seeded runs per balance configuration

Balance is a measurement problem.

A bot that plays three hundred seeded runs finds the truth about an economy faster than a hundred feel passes, and it leaves numbers I can defend later.

5

days of rules before the first line of Undead Signal code

THE ENGINEERING FLOOR

I am a software engineer by trade: production Vue and analytics tooling at Mouseflow, Python data pipelines at Cencora. That is why every design here shipped. I build the system, watch people play it, and fix the right thing.

Spinning is your only verb. If the ball catches you not spinning, you are out.

Robots in a circular pit bash a storm-charged ball at each other. Spin strikes it, parries it, and shoves a rival in close. Five bot temperaments, each a data profile for reaction time, aim error, and nerve, fill a three-arena campaign. Built in five days after switching engines mid-jam, then rebuilt in stages without ever breaking the shipped game.

Engine Unity 6, C#, URP 2D, WebGL

Jam Very Serious Juniper Dev Game Jam, theme Spin To Win

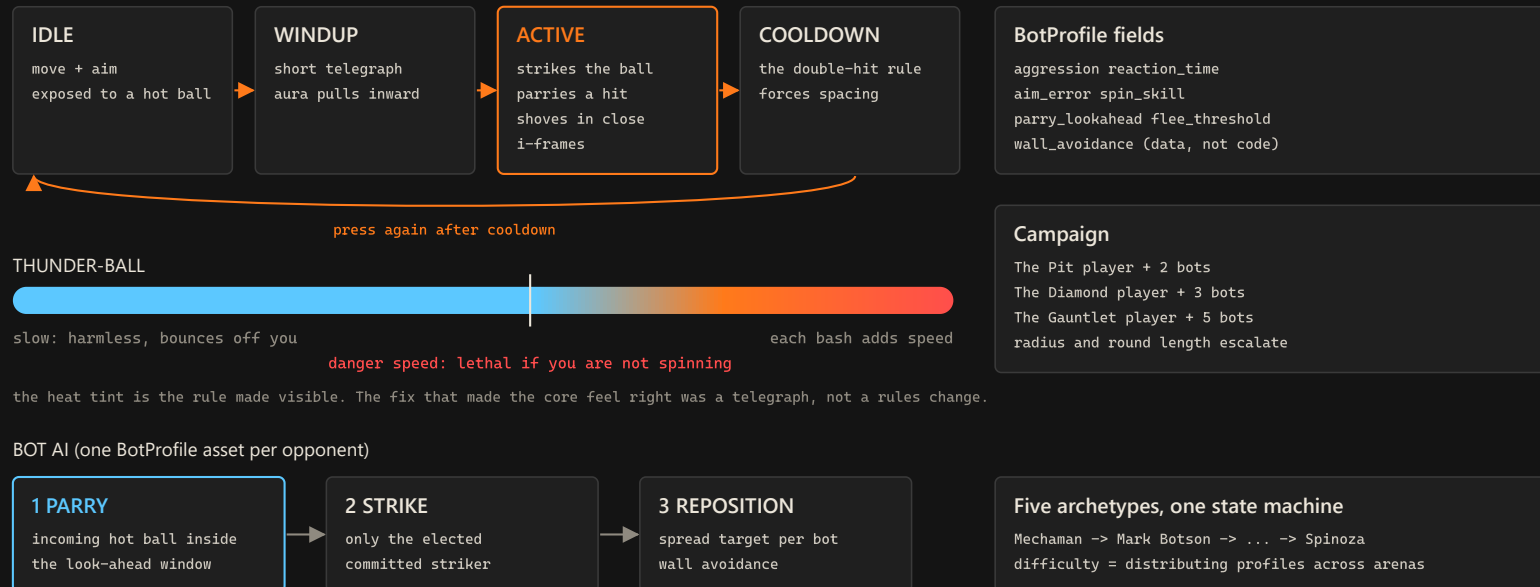
Role Design, direction, programming, bot AI, the jam's hand-drawn art

Shipped June 2026 jam build, v1.1 and online rooms September 2026

Play wavedash.com/games/thunder-bashers

ThunderBashers: one verb, three layers

spin state machine / ball lethality / bot priority



Unity 6 / C# / URP 2D. One elected striker per frame stops the midfield pile-up. Splitting offense from defense lets a bot be beatable without going passive.

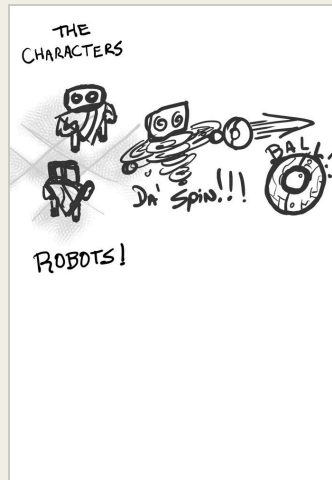
Systems map: the spin state machine, the ball's heat-gated lethality, the bot priority order, and the data fields behind every opponent.

The mechanic that fought back

For three days the core rule felt wrong, and every evening the temptation was to tune it again. When I stopped chasing symptoms it turned out to be five stacked problems: any-touch lethality, an over-corrected speed gate, a dead-zone speed band, one value gating both a bot's offense and its defense, and a ball that looked red while the kill check read it as safe after a bounce.

The fix that made it feel right was not a rules change. A heat tint on the ball, cool blue when slow and harmless, red and crackling when lethal, made the threat state visible. The rule had been fine. Nobody could see it.

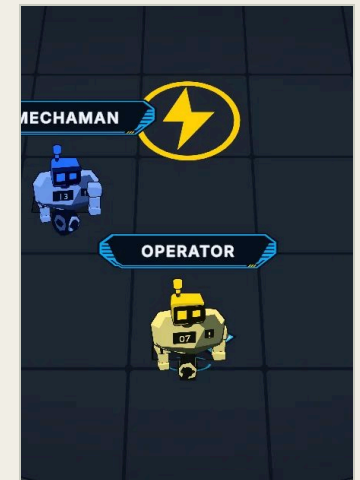
The bots are the game, so each is a profile over a PARRY, STRIKE, REPOSITION priority machine. One elected striker per frame stopped the pile-up. Splitting offense from defense let a bot be beatable without going passive. Parry rates run from roughly 25% for the rookie to 95% for the veteran, so difficulty reads as temperament rather than a slider.



First sketch, my hand: the robots, the spin, the ball. The whole game was on it.



Jam build: the ball glowing red-hot, the telegraph that says it is lethal.



v1.1 on Wavedash: one flat-shaded style for the whole match frame. The v1.1 update rebuilt the frame in stages without breaking the shipped game.

WHAT I'D KEEP

- / Root-cause, don't symptom-chase. Each fix that 'didn't work' was fixing a real but different layer.
- / Fix what the player can see before you change what the game does.
- / Guard every overhaul stage with a flag; the shipped game never stops being shippable. Online rooms shipped on the same rule, with the offline sim untouched underneath.

The player's kit never changes. Every bit of difficulty lives in the world.

A boy refuses to leave his room and defends it from a nightly alien invasion. The aliens are board-game pieces. The giant hands placing them are his. The final boss is his Dad opening the door, and beating the monster and the father getting through are the same event. The room was never the shelter. The play was.

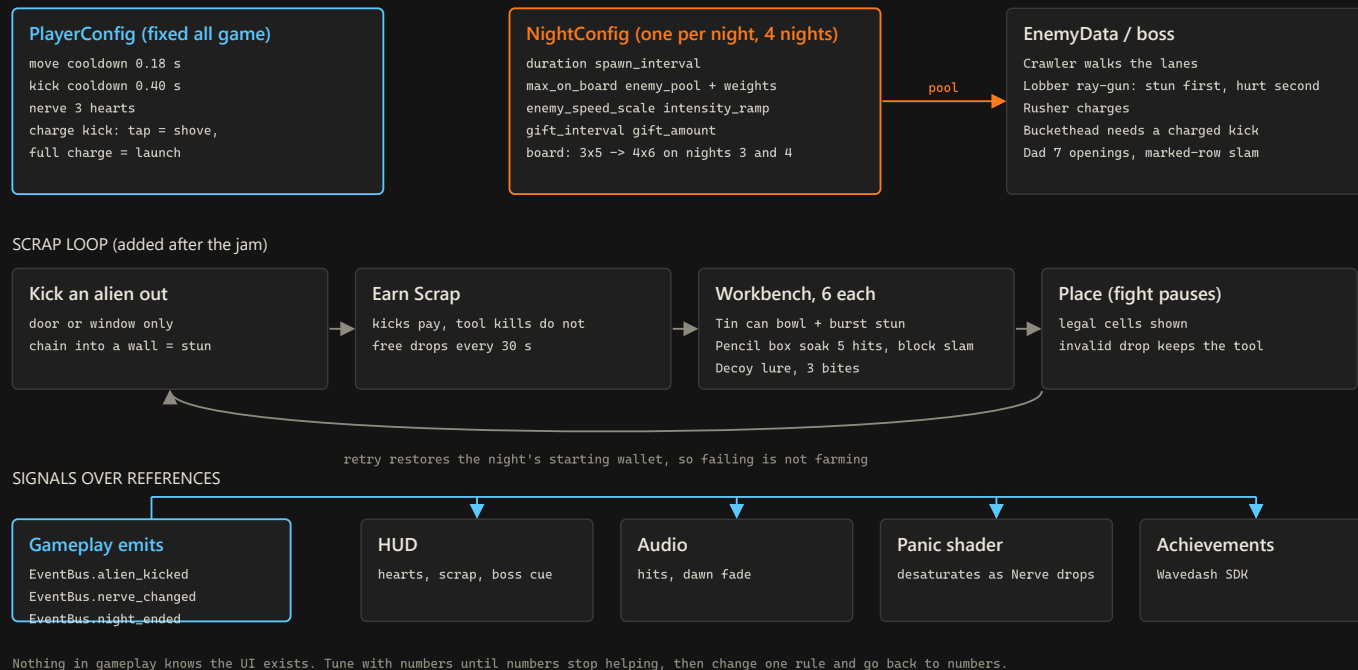
Engine Godot 4.6, GDScript Jam Wavedash Spring Jam 26, theme Shelter, 63 hours

Role Design, programming, direction, hand-drawn standees

Shipped June 2026, expanded September 2026 Play wavedash.com/games/i-won-t-be-abducted

I Won't Be Abducted: the player never changes, the world does

Godot 4.6 resources (.tres) carry the difficulty curve; the kit stays constant



Systems map: the fixed player config, the per-night dials, the enemy pool, the Scrap loop, and the event bus that keeps gameplay ignorant of the UI.

Write the do-not-build list first

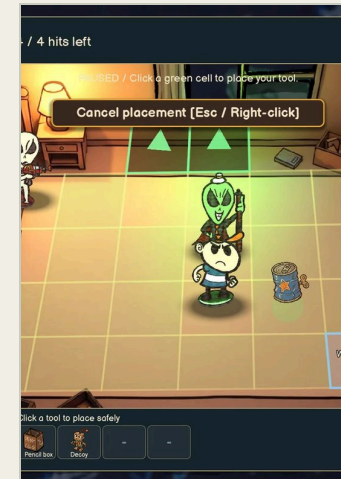
Before any code I wrote a three-column scope table: must ship, stretch, and do not build. The third column did the most work. The biggest cut was a shop between nights, which would have meant pricing every upgrade and pacing every income with no time to test either. It became a free draft of one card from three, and Scrap kept exactly one sink.

Twice no dial produced the right feel and the fix was a rule. Kicks that hit a wall now stun the whole chain, so every kick does something. The ray-gun's first hit stuns instead of hurting, so its punishment takes two mistakes.

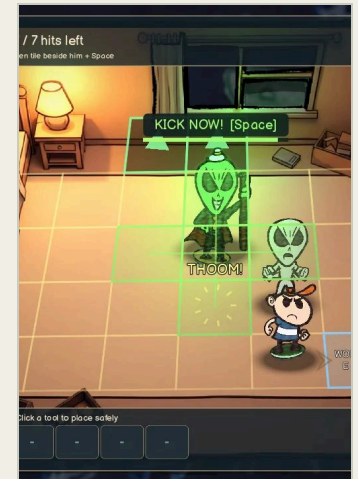
The September update was about what players could not read: a placement flow that passed its tests and ate clicks in the real interface, a boss that was too easy and then impossible to understand until his instructions moved onto the board above his head, and an economy that gave tools away.



Papercraft standees, my hand: the boy and the alien pieces. No walk cycles; all motion is code.



September build: the fight pauses, legal cells glow, and a blocked placement keeps the tool.



The boss's KICK NOW cue, countdown, and highlighted cells. Instructions where you are already looking.

WHAT I'D KEEP

- / Tune with numbers until the numbers stop helping, then change one rule and go back to numbers.
- / A test that calls the function is not a test of the click. Regression checks exercise the pointer.
- / Every system exists to protect one quiet beat at the end. Cut toward it.

The horde deals itself poker hands. Every move you have takes their hand apart.

You are a radio engineer restoring the signal in overrun locations. Whatever you do not remove from the table survives to form the horde's best five-card hand, and that hand's threat comes out of your health. Threat is one equation: the sum of five zombie ranks times a poker-hand multiplier. So you have exactly two levers. Drop the sum, or break the structure.

Engine Unity 6, C#, WebGL Jam Wavedash jam, theme Bad Connection

Role Rules, systems, balance, direction, zombie portraits

Shipped July 2026, updated September 2026 Play wavedash.com/games/undead-signal

Undead Signal: the whole game is one equation

THREAT = (sum of five zombie ranks) x (poker-hand multiplier)

$$\text{THREAT} = \sum \text{rank} \times \text{hand}$$

rank is also HP / suits are strains / High Card x1 ... Five of a Kind x18

Lever A: drop the sum

kill a zombie: its rank leaves the table
shrink a zombie: damage < rank lowers it

a shrunk 9 that becomes a 6 can complete
three 6s and RAISE the threat

Lever B: break the structure

scatter a pair, split a straight
the multiplier collapses first

the horde picks its best five cards
every act, so read the board

Every verb has a price

Discover: loot, but it feeds the horde pool
Loud weapon: Noise = extra horde draws
Weapons tap for 2 acts, then wear out
Failed kill: bite-back for remaining rank

RUN LOOP

Deal

act 1: 2 cards
act 2: 3
act 3: 4

Act

spend actions
kill / shrink / dig

Exit showdown

best-5 hand forms
THREAT hits Wellness

Safehouse

pick 1 of 3 Rig upgrades
+20 max Wellness

Next location

horde scales ~1.6x
4 locations + boss

Balance lab: a bot plays 300 seeded runs per configuration

v1 19% win (5% if it dared to Discover) -> gear density 0.15 to 0.30 + weapon-in-top-N rule -> 31% -> durability + bite-back -> 37% win, weapons cycle
gear density turned out to be two dials in one: more gear also means fewer zombies in the deck, and the second effect dominated

Unity 6 / C# / WebGL / Wavedash SDK (10 achievements, 2 leaderboards, cloud saves). Rules were drafted for five days before the build; three dead drafts are archived.

Systems map, drawn from the rules: the threat equation, the two levers, the price of every verb, the run loop.

Killing three drafts, then measuring the fourth

The first drafts were derivative: a zombie card battler, then a Balatro with zombies. Both functioned on paper and both were boring, because Balatro exists. The unlock was inverting it. The enemy assembles the scoring hand and every verb you own subtracts from it. Poker knowledge players already carry becomes threat literacy instead of a scoring manual.

Rank doubles as health. Hit a zombie for less than its rank and it shrinks, and a shrunk 9 that becomes a 6 can complete three of a kind and raise the threat you were lowering. Nobody designed that. It fell out of two rules colliding, and it is the rule players quote back to me.

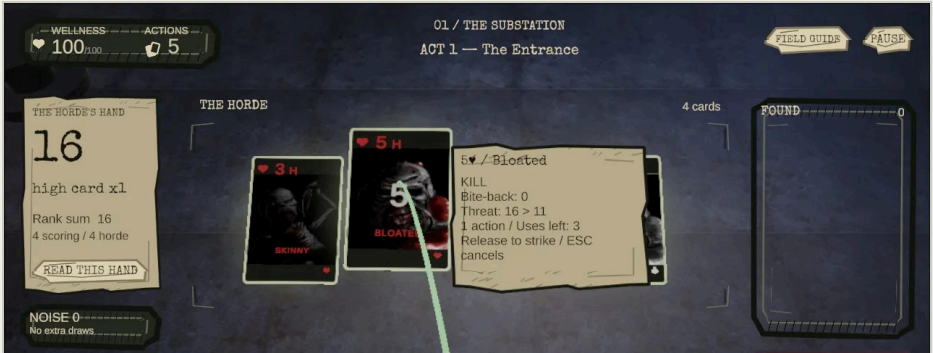
THE BALANCE LAB: A BOT PLAYS 300 SEEDED RUNS PER CONFIGURATION

PASS	WHAT THE BOT FOUND	WHAT CHANGED	RESULT
v1	19% win rate. 5% if the bot used the loot verb naively. Whole runs passed with no usable weapon before the first exit.	Gear density up; a rule guaranteeing a weapon near the top of the deck.	31% win. First location clear 52% to 84%. Deaths moved to the late-game wall.
v2	Playtesters: hands full at all times, loot stopped mattering.	Weapon durability and bite-back: a failed kill costs you the remaining rank.	37.3% win, weapons cycle. Target band 30 to 40%.

Gear density turned out to be two dials in one: more gear also means fewer zombies in the deck, and the second effect dominated. By hand I would have tuned it wrong forever.



July build: a five-zombie hand on the table, threat reading sum 19 times 2.



September build, the Survivor's Table: drag a weapon onto a card and the forecast appears beside the target.

WHAT I'D KEEP

- / Spend the design days before the build days; the archive of dead drafts is the receipt.
- / When a rule produces depth you did not author, the ruleset is telling you it works.
- / Write the target down before you measure, and re-run the report when someone asks.

Attacks never miss. The tension lives in your timing, not in the dice.

An outdated knight protects the last human he has in a world where the machines decided organic life is inefficient. Combat runs on a 6×6 isometric grid with speed setting the order. In a fifteen-minute run, a 75% roll that misses twice reads as theft, so the dice came out and randomness moved to the node map, where variance makes replays instead of resentment.

Engine Phaser 3, JavaScript, open source

Jam Gamedev.js Jam 2026, theme Machines, 13 days

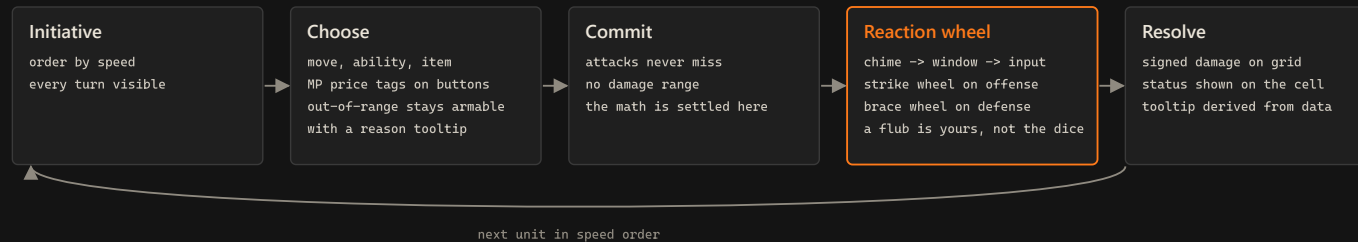
Role Direction, programming, painted characters, illustrations and interface; Nicholas Drabb on sound, voice and lore

Shipped April 2026, UI overhaul July 2026 Play chrisdbgames.itch.io/spare-knight

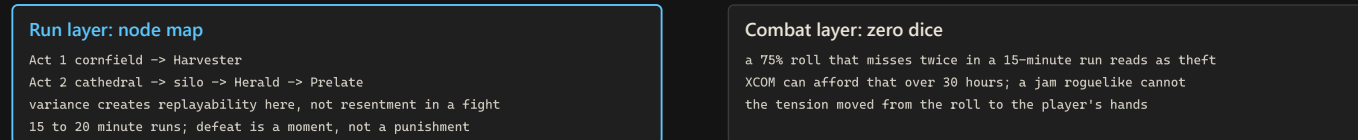
Next Electron desktop release aimed at Steam, in progress

Spare Knight: deterministic math, human timing

6x6 isometric grid / speed initiative / no attack RNG / reaction wheels



WHERE THE RANDOMNESS LIVES



INTERFACE AS A TOKEN SYSTEM (July overhaul, live on itch.io)



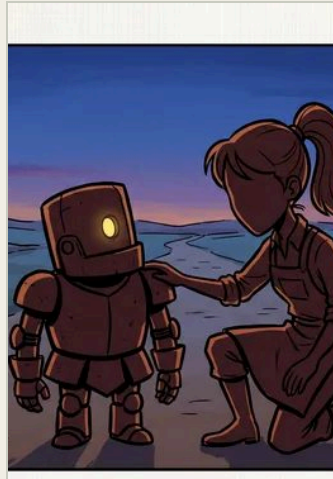
Phaser 3 / Vite. The live build is the July UI overhaul. A desktop beta with resumable runs and a black-and-paper interface is in progress and not yet published.

Deterministic math, human execution

Offense and defense both run through reaction wheels: hear a chime, commit on the beat. The math of an exchange is settled the moment you choose. Whether you flubbed the timing is on you. Defeat is a moment, not a punishment: short runs, instant restart, and a fiction that treats falling as part of being obsolete.

Players said the interface was hard to read, so I rebuilt it the way I would build a web app: named type roles, panel tokens, one shared tooltip every scene consumes, statuses rendered on the grid with tooltips generated from the mechanics data so the text cannot drift from the rules. That build is the one live today.

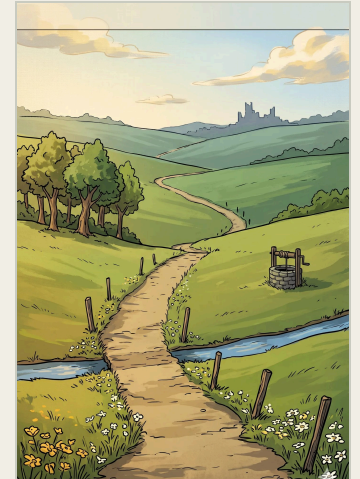
Five months after the jam I parked a better-engineered Unity rebuild after one evening's look test: the painted knight won on everything a player looks at. The decision is written down as an architecture record, and the Phaser build is heading to Steam.



Event illustration, painted in Krita: Wren whispers to the knight.



Act 1 combat on the isometric grid: Cog, the machines, and the painted interface.



The Act 1 node map, where the run's randomness lives.

WHAT I'D KEEP

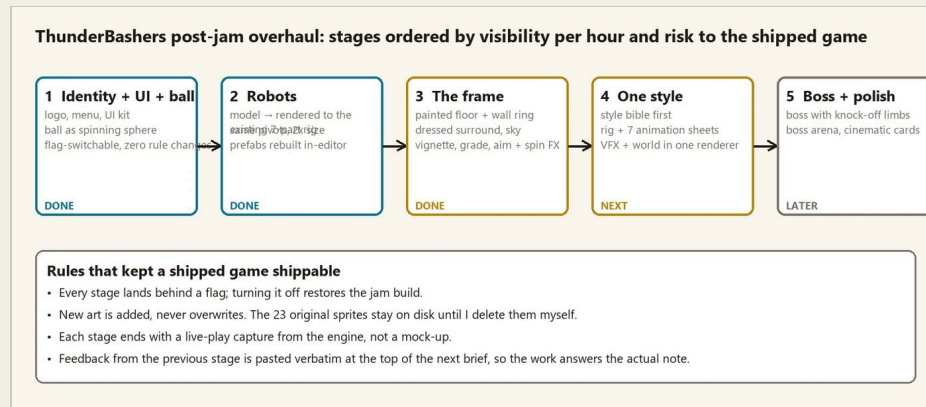
- / Lock the mechanical core in week one; the wheels were prototyped before any content existed.
- / Deterministic combat only feels fair when the player can see the determinism.
- / Scope the art to the camera, not to ambition.

Two instruments I now bring to every project.

The balance lab

Every episode derives from a config and a seed, so any number in any report can be regenerated exactly. A win rate belongs to a game and an agent together, never to the game alone, so every report names the bot and calls its rate a floor on competent play.

At three hundred runs a win rate near 35% carries roughly a five-point interval, so the verdict compares the interval to the target band, not the point. And the target goes in the config before the first run. That is the whole method. It balanced a shipped roguelike.

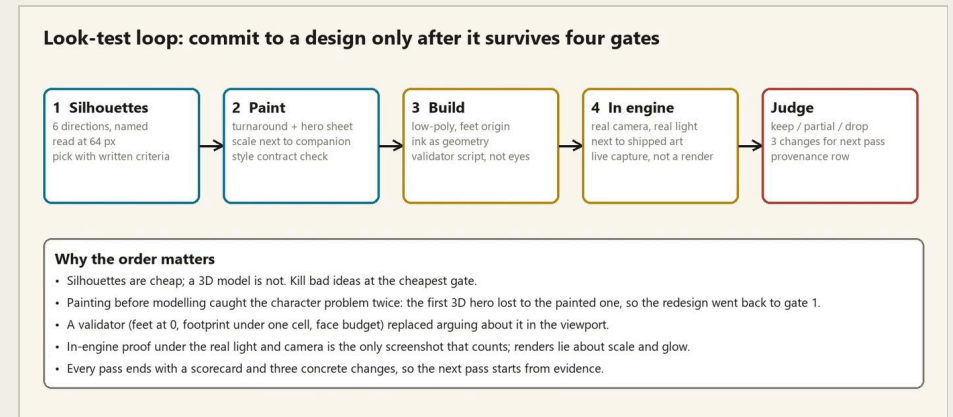


The staged overhaul plan for ThunderBashers, ordered by visibility per hour and gameplay risk. Drawn from the decision log.

The four-gate look test

Six silhouettes, one pick, judged at 64 pixels against written criteria. Then paint the winner beside its neighbour. Then build it and validate with a script. Then a live capture in the engine, or it did not happen.

Kill ideas at the cheapest gate. Three times this year a 3D character that looked good on a turntable lost to a painted one under the real light, and each time the loss was obvious in the engine and invisible in the render.



The look-test loop: silhouettes, paint, build, in-engine, judge. Drawn from the method, not a render.

Is this item worth its gold right now? A League of Legends question turned into a live tool.

Gold efficiency is how League players already argue about builds: what a stat line would cost bought raw against what the item costs. The numbers shift every patch and nobody recomputes them by hand. BuildValue does: per-item efficiency for every item in the game, six items side by side with a stat-by-stat breakdown, a full-build optimizer, and quick comparisons for a decision mid-draft.

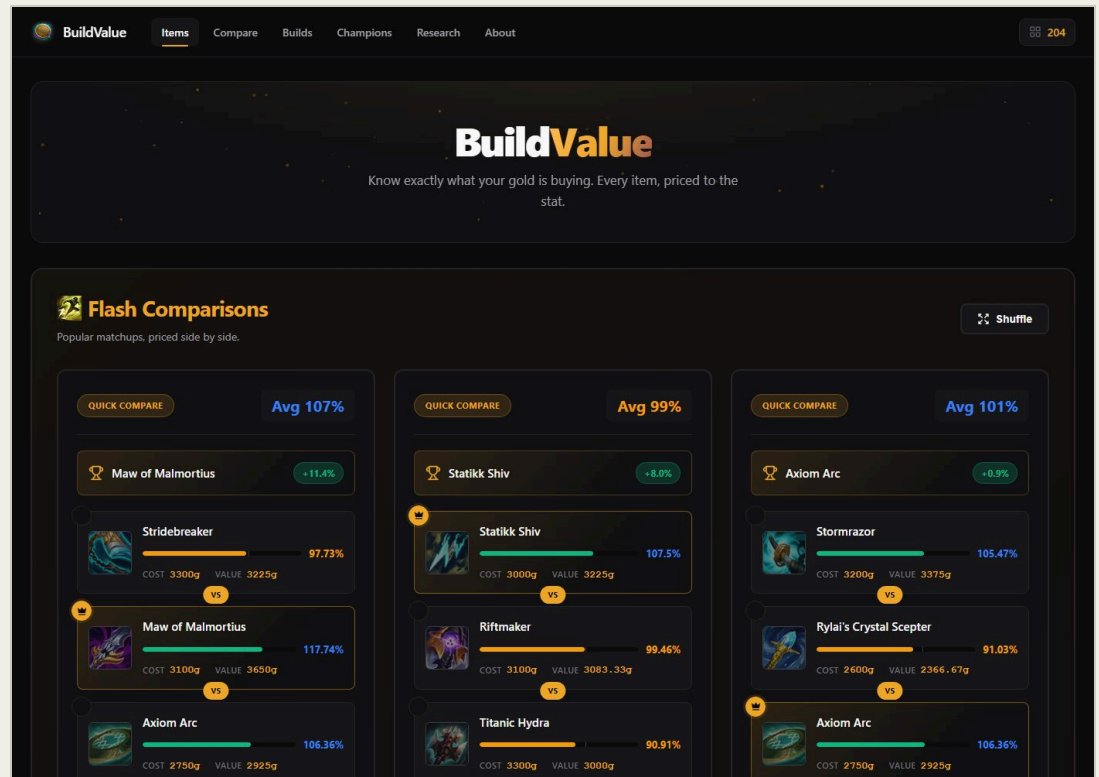
The design work was choosing which comparisons match how a decision is actually made, and making the tool self-maintaining so it stays true without me. No usage analytics are logged, so I describe it by what it can do, not by user counts.

Stack Vue 3, FastAPI, MongoDB, scheduled ETL

Data Riot's DDragon, synced weekly across patches

Live buildvalue.chrisdalbano.com

Note Not endorsed by Riot Games; League and Riot are their trademarks



BuildValue: item efficiency and comparison for League of Legends.

The engineering that makes the design checkable.

ROLES

Software Engineer · Mouseflow, remote Aug 2023 to present

Built the analytics dashboard's Vue and Chart.js data-visualization components and led its jQuery to Vue 3 migration. Owned the integrations platform end to end.

Senior Data Operations Engineer · Cencora Jul 2025 to present
(Intrinsiq), Orlando

Python tooling that monitors high-volume data pipelines on Azure Databricks and SQL Server; CI/CD through Azure DevOps.

Junior Frontend Developer · Baptist Health South Jun 2022 to Aug 2023
Florida, Miami

Patient-portal interfaces in React and TypeScript under WCAG requirements, on Java Spring Boot APIs.

EDUCATION

Bachelor of Applied Science (B.A.S.), Computing Technology and Software Development

Valencia College, 2025

A.S., Computer Programming and Analysis

Valencia College, 2023

SKILLS

Design. Systems and rules design, balancing and simulation harnesses, data-driven difficulty, playtest-led iteration, narrative structure, UI readability

Engines. Unity 6 (C#), Godot 4 (GDScript), Phaser 3 (JavaScript); Netcode for GameObjects; Wavedash SDK (achievements, leaderboards, cloud saves, lobbies)

Engineering. TypeScript, JavaScript, Python, C#, SQL; Vue 3, React; Django, FastAPI; Docker, Azure DevOps, Git

Art and tools. Krita and Wacom (painted characters, illustrations, interface), Blender for look tests

The games, the write-ups, and where to find me.

UNDEAD SIGNAL

Play: wavedash.com/games/undead-signal. Write-up: chrisdalbano.com/notes/undead-signal-postmortem. Achievements, leaderboards and cloud saves through the Wavedash SDK.

SPARE KNIGHT

Play: chrisdbgames.itch.io/spare-knight. Open source at github.com/Chrisdalbano/SpareKnight-webgame. Write-ups: the combat postmortem and the UI devlog.

THUNDERBASHERS

Play: wavedash.com/games/thunder-bashers, solo campaign and online rooms. Write-ups: the five-day postmortem and the overhaul story at chrisdalbano.com/game-design.

DIAGRAMS

Every systems map and process diagram in this deck is drawn by script from the rules and decisions, so it can be regenerated when a rule changes.

I WON'T BE ABDUCTED

Play: wavedash.com/games/i-won-t-be-abducted. Write-ups: the 63-hour postmortem, Tuning fear by numbers, and the September update.

METHOD

The balance lab and the four-gate look test are described in full on the site, with the numbers and the scorecards.

Every number in this deck comes from my own logs and re-runs on request. The full write-ups, with more screenshots, are at chrisdalbano.com/game-design.

chrisdalbano12@gmail.com

chrisdalbano.com/game-design

wavedash.com/@chrisdalbano

chrisdbgames.itch.io