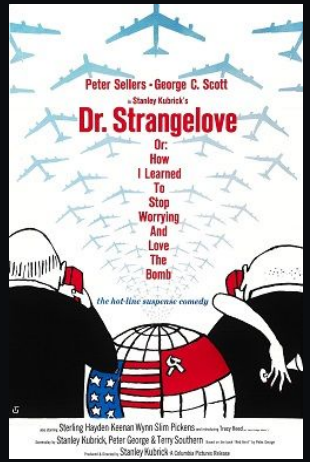
# Dr. Claudelove,

## or: How I Learned to Stop Worrying and Love the Git Worktree

*“Agents, you can't code in here,*
*this is the main worktree!”*




**Disclaimer**

- I am not a Git expert.
- I am not the most sophisticated Git user or an expert in worktrees and how they work / what you can do with them. (Apologies if the title is false advertising.)
- I simply heard they were useful for parallelizing agentic development and asked Claude to spin them up and rebase when needed.

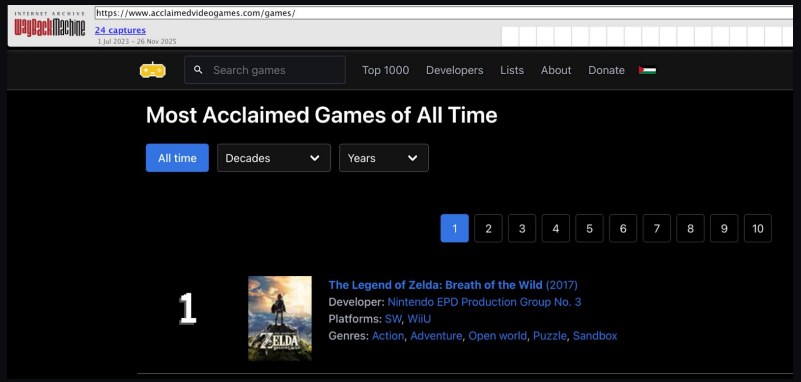
**The following is a “YOLO” experiment in pushing the limits of Claude Code for a personal project. And a deliberate challenge to MAXIMIZE, not minimize, token usage at Anthropic’s promotional rate before New Year’s.**

# Project Context

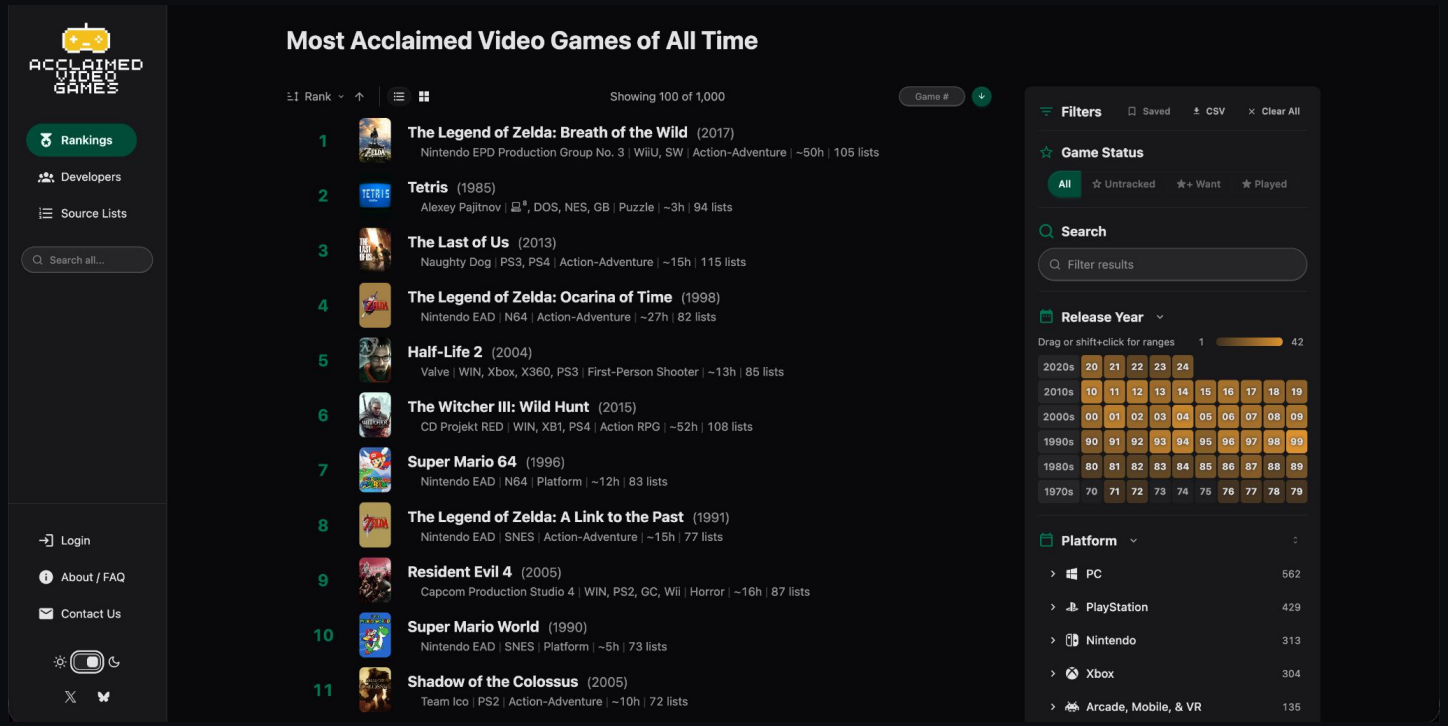
## Personal Project

- acclaimedvideogames.com is a site that aggregates >800 critic lists of the greatest video games. (This has better data for older retro games than something score-based like Metacritic.)
- In the past month and a half it has undergone several major refactors:
    - Migration from heavier frontend single-page app (SPA) on Vue.js to Django + HTMX + Alpine.js for Wayback Machine archivability and simplicity.
    - Completely redoing the layout of pretty much every page and flexibility of filtering the results. (Including custom year heatmap component etc.)
    - Expanding and generalizing the backend to support book lists in the future.


No archive :(




Archive! :)

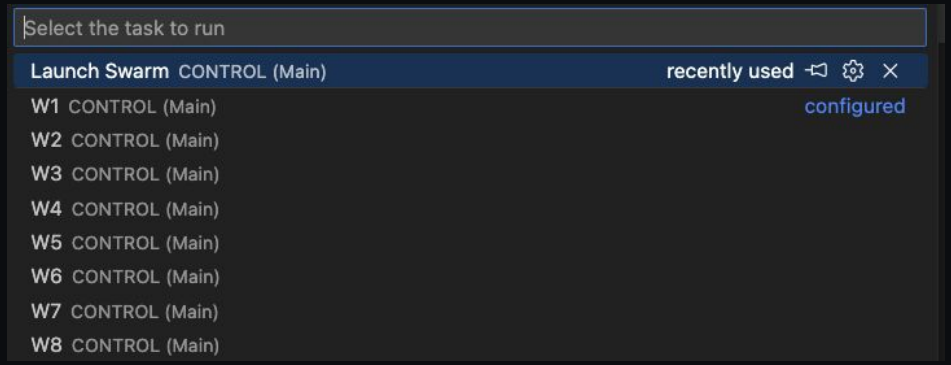



Final state :O
(Books changes are backend-only right now)

# Launching the Plane

## Takeoff

- The Script: Wrote a shell script to start eight Git worktrees simultaneously.
- The Command: Integrated as a VS Code workspace task to spin up eight parallel terminals of Claude Code in YOLO mode.
- The Goal: Deliberately doing “the crazy thing” to test massive agent parallelism.

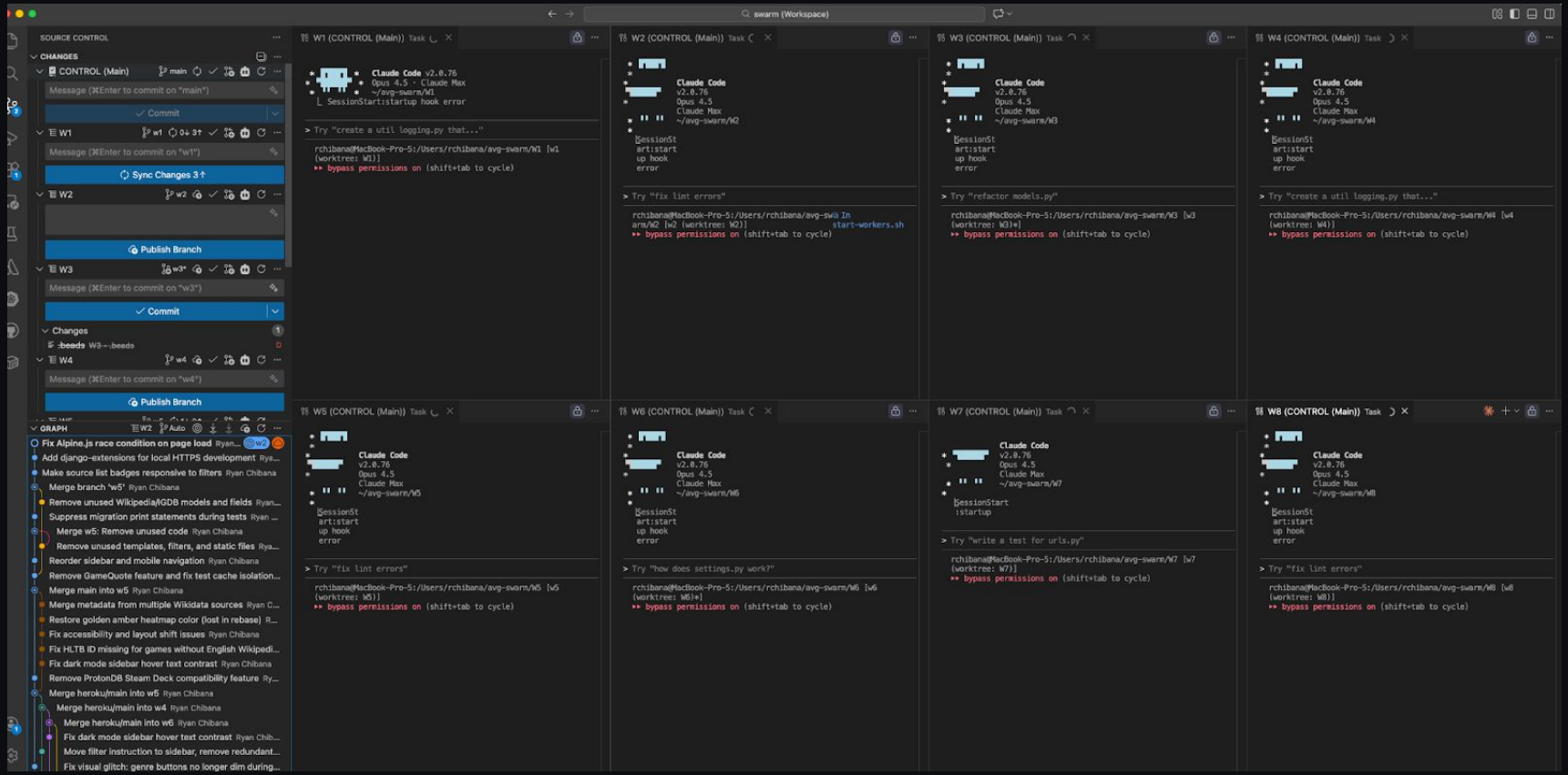



$ start-workers.sh

```
#!/bin/bash
# Opens VS Code workspace and starts Claude workers in git worktrees

SCRIPT_DIR="$(cd "$(dirname "$0")" && pwd)"
PARENT_DIR="$(dirname "$SCRIPT_DIR")"

# Open VS Code workspace
code "$PARENT_DIR/acclaimedgames-workers.code-workspace"

# Start each worker in its own Terminal with Claude
for i in {1..8}; do
  WORKER_DIR="$PARENT_DIR/acclaimedgames-worker-$i"
  if [ -d "$WORKER_DIR" ]; then
    osascript -e "
      tell application \"Terminal\"
        do script \"cd '$WORKER_DIR' && echo 'Worker $i ready.' && claude --dangerously-s
      end tell
    "
  else
    echo "Warning: $WORKER_DIR does not exist"
  fi
done

echo ""
echo "Workers started."
```

claude –dangerously-skip-permissions 😈

# The War Room: This Deserves Its Own Slide



# Managing the Madness

## Operating realities

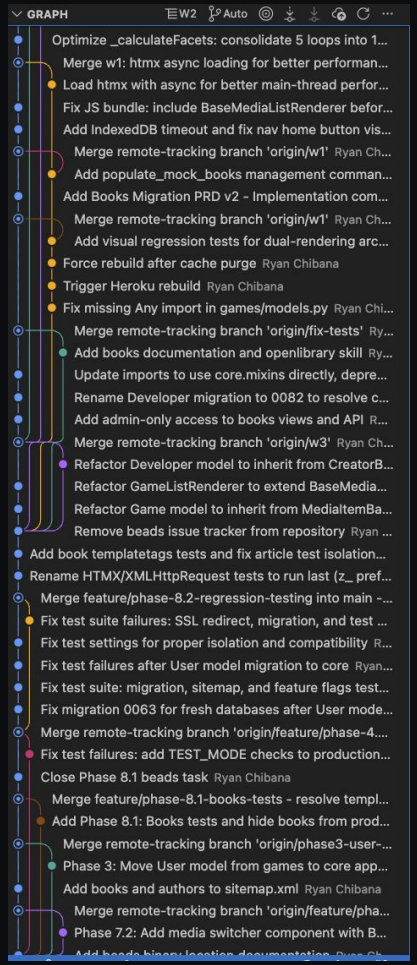
- Visual Monitoring: eight terminals looks insane, but monitoring Git state is essential.
- Coordination Failure: tried using “beads” for issue tracking; failed due to setup + manual tree management.
- The Pivot: fell back to a single Markdown PRD.
- Agent Interaction: agents claim tasks and mark “done” directly in the PRD.

```
### Phase 1: Core Infrastructure

#### 1.1 Create Core App Structure

| Task ID | Task | Files | Complexity | Risk | Dependencies |
|---------|------|-------|------------|------|--------------|
| 1.1.1 | Create core/ app skeleton | `core/__init__.py`, `core/apps.py` | S | Low | None |
| 1.1.2 | Add to INSTALLED_APPS | `acclaimedgames/settings.py` | S | Low | 1.1.1 |
| 1.1.3 | Create abstract MediaItemBase | `core/models.py` | M | Low | 1.1.1 |
| 1.1.4 | Create abstract CreatorBase | `core/models.py` | S | Low | 1.1.1 |
| 1.1.5 | Create abstract ExternalDataBase | `core/models.py` | S | Low | 1.1.1 |
| 1.1.6 | Create abstract UserTrackingBase | `core/models.py` | S | Low | 1.1.1 |

**Parallelization**: 1.1.3–1.1.6 can be done in parallel after 1.1.1–1.1.2

#### 1.2 Move Shared Utilities

| Task ID | Task | Files | Complexity | Risk | Dependencies |
|---------|------|-------|------------|------|--------------|
| 1.2.1 | Move RobustPaginationMixin to core | `core/mixins.py`, `games/mixins.py` | S | Low | 1.1.2 |
| 1.2.2 | Move HTMXPartialMixin to core | `core/mixins.py`, `games/mixins.py` | S | Low | 1.1.2 |
| 1.2.3 | Add backward-compat re-exports in games | `games/mixins.
```

# Mutually Assured Destruction: Drift and Deployment

## What it feels like

- The Drift: eight agents inevitably drift away from each other.
- Rebasing: you must tell agents to rebase regularly (hard at swarm scale).
- The Middle State: you hit moments where you don't understand anything that's happening and briefly consider starting over and throwing out tens of thousands of lines.
- Pushing Through: with intuition + delegation, you emerge into a stable, "very good" state.
- Every now and then, dedicating an agent to reviewing all the code produced and the PRD to determine incompatibilities / gaps helps a lot. I generated a v2 PRD based on this that simplified things immensely.

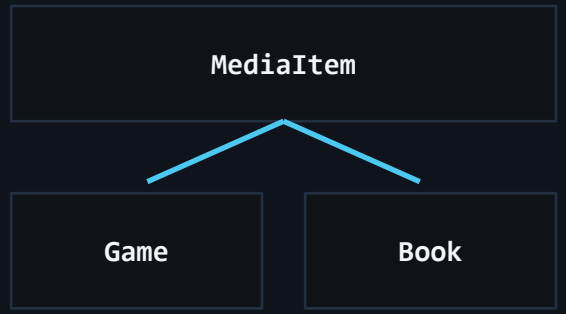


# From Games to Books

## The refactor

- Objective: generalize the data model of acclaimedvideogames.com to support book lists.
- Games now inherit from a general MediaItem class.
- Abstracted templates + pages into a core app with derived versions specific to the games and books apps.
- Scale: involved rewriting tens of thousands of lines of code.
- AI autonomy: agents designed aspects of the new model without it being in the original PRD.
- Outcome: the books refactor was completed and correct **within a couple of hours**.

Data model (mock)




Agents inferred what components could be broken out and which were specific to each app.

# Is 8× Faster Real?

≈4×

…faster than a single agent session (in practice)

## What actually happens

- Truth: it isn’t eight times faster than a single agent session — there is wasted work.
- Reality: it feels closer to ~4× for large refactors.
- Strategy: clear early and clear often; start new sessions frequently.
- Surprise: despite chaos, the final structure eventually landed on an organized and logically correct state.

# The Post-Atomic Age

## Can traditional 400-line PR reviews survive code generation at this scale?

### Implications

- Reality: these massive refactors involved zero traditional code review.
- New approach: shift review up a level — validate PRD / intent, then verify outcomes.
- Challenge: we don’t have the final answers yet, but the speed forces a rethink.
- Risk vs. speed: correctness wasn’t essential for a hobby project — adjusting this approach to corporate / Compliance levels of risk is the hard part.

# Riding the Bomb: From Coder to Fleet Manager

### What changed in my head

- I now see myself as a manager of a fleet of agents, learning to coordinate them effectively.
- The power of iterating across eight agents simultaneously is hard to grasp until you see it. (Okay, eight is too many. Try four just to try it.)
- If things break later, I have an entire team of agents ready to fix them.

# What Now?

## Discussion Questions (Open-Ended)

- What's the best "unit of work" for agents: PRD tasks, files, modules, or end-to-end features?
- If line-by-line review doesn't scale, what should review focus on: intent/PRD, architecture diffs, tests?
- Have you come across any good methods for coordinating swarms of coding agents?
- What are the must-have guardrails to ship safely: test strategy, observability, rollout plan, and automated checks?