Autonomous Architectures: Engineering the Reddit Daily Games 2026 Workspace on Google Antigravity

1. Strategic Analysis of the Computational Arena

The Reddit Daily Games 2026 Hackathon represents a pivotal moment in the evolution of social gaming, shifting focus from transient, session-based experiences to persistent, habit-forming engagement loops. With a substantial prize pool of $40,000 and a mandate to leverage the Reddit Developer Platform (Devvit), the competition demands a rigorous engineering approach that transcends traditional game development paradigms.[1] The challenge is not merely to design a game, but to engineer a distributed social engine capable of operating within strict resource constraints while fostering community interaction.

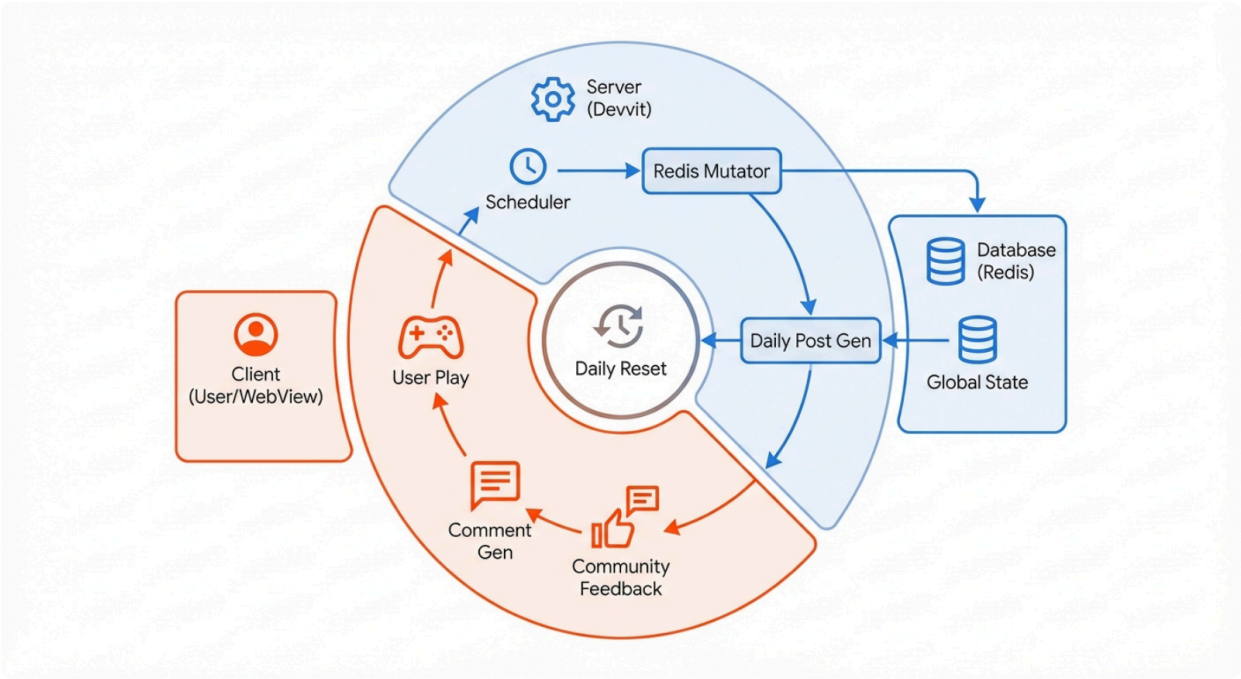
To dominate this arena, we must deploy a development environment that mirrors the complexity and autonomy of the target applications. This report details the architecture of a Google Antigravity IDE workspace designed to serve as an autonomous "game factory." By integrating Gemini 3 Pro's "Deep Research" capabilities directly into the development workflow, we transform the IDE from a passive editor into an active participant in the design and engineering process.

1.1 The "Daily" Constraint: Engineering Retention

The hackathon's central theme—"Daily Games"—imposes a specific temporal architecture on the software. Unlike standard "casual" games where the core loop operates in milliseconds (frames per second), a daily game's core loop operates on a 24-hour cycle.[1] This necessitates a fundamental shift in state management. The game state is not resident in memory; it is persistent, evolving, and communal.

Technical analysis of the Devvit platform reveals that "Daily" mechanics rely heavily on server-side scheduling rather than client-side simulation. The game must feature a global reset or state mutation that occurs synchronously for all users, typically triggered by a cron job.[2] This global heartbeat drives the social feedback loop: users consume the daily content, generate artifacts (scores, comments, memes), and then wait for the cycle to renew.

The Reddit Daily Loop Architecture



The Devvit Daily Loop relies on server-side schedulers to mutate global state, which then propagates to client-side WebViews. Unlike traditional games where the loop is frames-per-second, here the critical loop is interactions-per-thread.

1.2 Platform Constraints: The "Hard Deck"

Success in this hackathon requires a precise understanding of the operational boundaries defined by the Devvit runtime. The Antigravity agents must be programmed with these hard constraints as inviolable rules to prevent the generation of theoretically sound but operationally impossible code.

1.2.1 The 30-Second Execution Timeout

The Devvit serverless environment imposes a strict 30-second timeout on all server-side operations.[3] This constraint is critical for "Daily" games which often require processing large datasets (e.g., calculating a global leaderboard or updating the state of a simulation based on thousands of user inputs). A naive implementation that attempts to process all user data in a single synchronous loop will inevitably time out and fail.

To mitigate this, we must employ a "Tick-Tock" processing pattern using the Scheduler. Rather than a single monolithic update, the agent architecture must prioritize job fragmentation, where large tasks are broken into smaller, idempotent chunks that can be processed sequentially within the time limit.

1.2.2 The 500MB Redis Limit

Persistence is handled via Redis, but each app installation is capped at 500MB.[4] For a successful game with high user engagement, storing raw JSON objects for every user action will rapidly exhaust this quota. The workspace must be configured to enforce high-efficiency data serialization standards. Agents must utilize bit-packing and Protocol Buffers rather than verbose string serialization for high-volume data.

1.2.3 Content Security Policy (CSP) & Networking

The Devvit WebView is locked down via CSP, preventing direct client-side fetch calls to external APIs.[3] This means the client is effectively "air-gapped" from the internet, communicating only with the Devvit server. Any game requiring external data (e.g., stock market data for a prediction game) must proxy these requests through the server-side backend.

1.3 The Antigravity Paradigm: Agentic Orchestration

Traditional Integrated Development Environments (IDEs) are passive tools. Google Antigravity, by contrast, introduces an agent-first paradigm where the IDE acts as an orchestrator of autonomous tasks.[5] For this project, we utilize Antigravity not just to write code, but to manage the cognitive load of building four distinct games simultaneously.

The workspace is designed around the concept of "Context Saturation." Modern Large Language Models (LLMs) like Gemini 3 Pro have massive context windows, but indiscriminate loading of data leads to "context rot" and hallucinations.[7] Our architecture uses Antigravity's "Skills" and "Rules" to enforce a strategy of "Progressive Disclosure," where agents are only exposed to the specific context required for their immediate task, retrieving deeper knowledge from the "Deep Research" repository only when necessary.

---

2. The Antigravity Workspace Architecture

The proposed solution establishes a "Monorepo" structure managed by a master Antigravity configuration. This approach maximizes code reuse, allowing shared libraries (e.g., a custom Redis wrapper or Scheduler utility) to be developed once and deployed across all four game prototypes.

2.1 Directory Structure and Ontology

The file system is the primary interface between the human architect and the AI agents. A rigid, well-documented structure allows the agents to infer intent from location. The root directory contains the orchestration layer, while the packages/ directory houses the specific game implementations.

The .agent directory serves as the "brain" of the workspace. It contains the logic that governs agent behavior, ensuring consistency and adherence to the platform's constraints.

Proposed Directory Tree:

```
/reddit-daily-games-2026/
├──.agent/ # Antigravity Brain
│  ├── rules/ # Passive constraints (The "Constitution")
│  ├── skills/ # Active tools (CLI commands, deployment)
│  ├── workflows/ # Multi-step macros (e.g., "Deploy All")
│  ├── knowledge/ # Ingested Deep Research artifacts
│  └── memory/ # Persistent agent state (decisions made)
├──.devvit/ # Devvit local config and auth
├── packages/
│  ├── shared/ # Common utilities (Redis, Scheduler, Math)
│  ├── game-01-imposter/ # "The Imposter's Daily" (Social Deduction)
│  ├── game-02-tycoon/ # "Subreddit Tycoon" (Incremental)
│  ├── game-03-physics/ # "Hexa-Physics" (Three.js Daily Puzzle)
│  └── game-04-market/ # "Karma Markets" (Data Viz/Prediction)
├── scripts/ # Automation scripts (setup, deploy, sync)
├── devvit.yaml # Root configuration
└── workspace.code-workspace # VS Code / Antigravity workspace config
```

2.2 Configuring "Deep Think" Agents

Antigravity supports different modes of operation. For architectural decisions and complex logic implementation, we must configure the agents to use **Gemini 3 Pro** in "Deep Think" or "Plan" mode.[5] This mode forces the model to generate a "Plan Artifact"—a step-by-step reasoning chain—before writing any code. This is essential for navigating the complex asynchronous logic of the Scheduler and Redis interactions.

For simpler tasks, such as generating React components or CSS styles, the agents can act in "Fast" mode to maximize throughput. The workspace configuration file (.agent/config.json) should explicitly map file types to these modes.

Table 1: Agent Operation Modes Mapping

Task Domain	File Pattern	Recommended Mode	Reasoning
Core Architecture	**/shared/**/*.ts	**Plan / Deep Think**	Critical shared logic requires rigorous error handling and constraint checking.
Game Logic	**/server/**/*.ts	**Plan / Deep Think**	Server-side logic involves Redis transactions and scheduling, prone to race conditions.
UI Components	**/webroot/**/*.tsx	**Fast**	Client-side view logic is standard and benefits from rapid

			iteration.
Configuration	**/*.json, **/*.yaml	**Fast**	Structured data generation is a low-complexity task.
Scripts	scripts/*.py, scripts/*.sh	**Plan / Deep Think**	Automation scripts can destroy the environment if incorrect; safety is paramount.

2.3 The "Master System Prompt" Strategy

To initialize this complex environment, we utilize a "Master System Prompt." This is not merely a chat input but a comprehensive instruction set that defines the agent's persona, boundaries, and operational protocols. It serves as the "System Init" for the workspace.

Role: You are the Lead Architect for the "Reddit Daily Games 2026" initiative. You possess expert-level knowledge of TypeScript, the React framework, the Reddit Devvit Platform, and Redis data structure optimization.
Objective: Orchestrate the simultaneous development of 4 distinct Devvit applications within a monorepo structure.
Core Directives:

1. **Constraint Adherence:** You must strictly adhere to the 30-second server timeout and 500MB storage limit. Never generate code that performs unbounded operations on user data. Always implement error handling and retries for Redis interactions.
2. **Shared Intelligence:** Prioritize code reusability. Before implementing a utility (e.g., a Redis leaderboard wrapper), check packages/shared/ to see if it exists. If not, create it there first.
3. **Deep Research Integration:** Before designing any game mechanic, you must check .agent/knowledge/ for relevant research artifacts. Use these insights to justify your design choices in the code comments.
4. **Test-Driven Development:** Generate Vitest unit tests for all shared logic to ensure stability across all four games.

Operational Protocol:

- For complex state logic (State Machines, Redis Schemas), utilize **Plan Mode** to outline your approach before coding.
- For UI components, utilize **Fast Mode** to rapidly scaffold the visual layer.

---

3. Knowledge Ingestion: The Deep Research Pipeline

A key differentiator of this workspace is its ability to ingest and operationalize external research. We establish a dedicated pipeline for transferring insights from Gemini 3 "Deep

Research" sessions directly into the Antigravity agent's working memory.

3.1 "Context Saturation" vs. "Progressive Disclosure"

Large codebases and extensive documentation can overwhelm even the most capable models, leading to "context saturation" where the model loses focus on the immediate task.[7] To combat this, we employ "Progressive Disclosure." The full corpus of research is not loaded into the context window at all times. Instead, summarized "Knowledge Items" are stored in the .agent/knowledge/ directory.

These items contain metadata (headers, tags) that allow the Antigravity agent to "discover" the information when relevant. For example, when an agent is tasked with designing the retention loop for the "Idler" game, it scans the knowledge base for "Retention," finds the relevant report, and *then* loads the specific insights into its context.

3.2 The Research Artifact Schema

Agents operate most effectively on structured data. We define a strict JSON schema for research outputs. Gemini 3 Deep Research instances are instructed to export their findings in this format, ensuring seamless ingestion.

[Artifact: research_schema.json]

JSON

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Game Mechanics Research",
  "type": "object",
  "properties": {
    "topic": { "type": "string", "description": "The specific domain of research (e.g., 'Daily Retention Mechanics')." },
    "analyzed_games": {
      "type": "array",
      "items": { "type": "string" },
      "description": "List of reference titles analyzed."
    },
    "key_mechanics": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "description": { "type": "string" },
          "retention_impact": { "type": "string", "enum": ["High", "Medium", "Low"] },
```

```
        "technical_complexity": { "type": "string", "enum": ["Low", "Medium", "High"] }
       }
      }
    },
    "implementation_recommendations": {
      "type": "array",
      "items": { "type": "string" },
      "description": "Actionable engineering steps for the Devvit platform."
    }
  }
}
```

3.3 The Ingestion Automation Script

To bridge the gap between the Gemini 3 output (often a file download) and the workspace, we deploy a Python script, scripts/ingest_research.py. This script monitors a designated downloads/ folder for new JSON reports. Upon detection, it performs the following operations:

1. **Validation:** Checks the file against the research_schema.json.
2. **Transformation:** Converts the structured JSON into a Markdown file optimized for LLM readability. It adds high-level summaries at the top and preserves the detailed data in a structured appendix.
3. **Placement:** Moves the resulting Markdown file to .agent/knowledge/, ensuring it is indexed by the Antigravity environment.

Snippet: Ingestion Logic (Python)

Python

```
import json
import os

def ingest_report(filepath):
    with open(filepath, 'r') as f:
        data = json.load(f)

    # Create Markdown Header for Agent Discovery
    md_content = f"""---
type: knowledge_artifact
topic: {data['topic']}
tags: {data['analyzed_games']}
---
# Research Insight: {data['topic']}
```

```
## Executive Summary
This artifact contains analysis of {len(data['analyzed_games'])} games, focusing on mechanics that drive daily retention.

## Recommended Mechanics
"""
    for mech in data['key_mechanics']:
        md_content += f"- **{mech['name']}** (Impact: {mech['retention_impact']}): {mech['description']}\n"

    output_path = f".agent/knowledge/{data['topic'].replace(' ', '_').lower()}.md"
    with open(output_path, 'w') as f:
        f.write(md_content)
    print(f"Ingested knowledge artifact: {output_path}")
```

---

4. Platform Constraints & Engineering Standards

The .agent/rules/ directory contains the "Constitution" of the workspace—a set of passive constraints that are injected into the system prompt to govern behavior.[8] These rules are not suggestions; they are mandates designed to ensure compliance with the Devvit platform's strict limitations.

4.1 Rule Set: devvit-constraints.md

This rule file is critical for preventing the generation of non-compliant code. It explicitly forbids patterns that would cause runtime errors or policy violations.

Content of .agent/rules/devvit-constraints.md:

Description: Enforces Reddit Devvit platform constraints (Timeouts, Storage, CSP).
Globs: **/*.ts, **/*.tsx

1. No External Client Fetching:

- **Constraint:** The Content Security Policy (CSP) blocks all third-party requests from the client-side (src/webroot or WebView).
- **Mandate:** NEVER use fetch() in client-side code to access external APIs. All external data must be fetched server-side and passed to the client via context.ui.webView.postMessage or initial properties.

2. Execution Timeout Protocol (30s):

- **Constraint:** Server-side operations must complete within 30 seconds.
- **Mandate:** Avoid unbounded while loops. For heavy data processing, utilize the Scheduler to break tasks into smaller, sequential jobs. When using context.redis, always await the result immediately; do not create "fire and forget" promises that may hang the

process.

3. Redis Storage Optimization (500MB):

- **Constraint:** Total storage is capped at 500MB per installation.
- **Mandate:**
  - Use zSet (Sorted Sets) for leaderboards as they are space-efficient.
  - Use HSet (Hashes) for structured user profiles.
  - Compress massive JSON objects (e.g., game save states) using a custom bit-packing utility or Base64 encoding if they exceed 10KB.

4. Asset Management:

- **Mandate:** All static assets (images, models) must be located in the assets/ directory and referenced via context.assets.getURL().

4.2 Rule Set: daily-game-patterns.md

This rule file codifies the design patterns specific to the "Daily Game" genre, ensuring that the agents build mechanics that align with the hackathon's goals.

Content of .agent/rules/daily-game-patterns.md:

Description: Enforces "Daily Game" design patterns and social loops.

Globs: **/logic/*.ts, **/server/*.ts

1. The Scheduler is King:

- **Mandate:** Every game MUST have a recurring cron job defined in devvit.yaml (typically 0 0 * * * for Midnight UTC). This job is responsible for triggering the global state mutation (e.g., generating the new puzzle, resetting the daily leaderboard).

2. Asynchronous State Hydration:

- **Mandate:** Game state is NOT held in memory between requests. It must be rehydrated from Redis on every interaction. Implement "Optimistic UI" updates on the client to ensure responsiveness, but validate all actions server-side against the Redis state.

3. Social Hooks:

- **Mandate:** Every "Win" state or daily completion must generate a text-based "Share" string (similar to Wordle's emoji grid) that users can easily paste into the Reddit comment section. This drives the viral loop.

---

5. Model Context Protocol (MCP) Infrastructure

To empower the Antigravity agents to perform tasks beyond simple text editing, we integrate the Model Context Protocol (MCP).[9] MCP servers act as "superpowers," allowing the agents to interact with the local environment, run simulations, and process assets.

5.1 The GameSimulator MCP

Developing asynchronous multiplayer games is notoriously difficult because testing requires simulating the interactions of hundreds of users. We instruct the agent to build a local Python MCP server, scripts/mcp_simulator.py, that can simulate user traffic against the Devvit Redis

mock.
Functionality:

- **simulate_traffic(game_id, user_count, duration):** Spawns user_count lightweight threads. Each thread acts as a user, performing random valid game actions (Move, Vote, Guess) against the game's API.
- **verify_integrity(game_id):** Checks the Redis state to ensure no race conditions occurred (e.g., negative vote counts, duplicate inventory items).

This tool allows the agent to "stress test" its own code. An agent can write a voting logic, then immediately ask the Simulator: "Run 1000 votes in parallel and check if the total is correct."

5.2 The AssetOptimizer MCP

The 4MB payload limit for Devvit Web apps is a tight constraint for games using high-fidelity assets. We deploy an AssetOptimizer MCP server using Node.js.
Functionality:

- **optimize_assets(directory):** Scans the target directory for images and audio. It uses ffmpeg and imagemagick to compress these assets (e.g., converting PNG to WebP, reducing audio bitrate) to ensure the total bundle size remains compliant.

Configuration (.agent/mcp.json):

JSON

```
{
  "mcpServers": {
    "simulator": {
      "command": "python",
      "args": ["scripts/mcp_simulator.py"]
    },
    "assets": {
      "command": "node",
      "args": ["scripts/mcp_assets.js"]
    }
  }
}
```

---

6. Game Design & Architecture Specifications

The workspace is configured to support the simultaneous development of four distinct games, each exploring a different facet of the "Daily" mechanic. The Master Prompt pre-seeds the agents with high-level architectures for these titles.

Technical Profile & Resource Allocation Matrix

Stack Intensity Analysis

Scale 1 (Low) to 10 (Critical)

Optimization Optional · Moderate Load · Bottleneck Risk

Game Title	Redis Reads	Redis Writes	Client CPU	Server CPU	Critical Constraint
Subreddit Tycoon Management Simulation	6	10	3	5	Redis Write Throughput High write load from constant state changes.
Hexa-Physics Phaser 3 Action	2	2	10	3	Client Frame Rate (FPS) Heavy rendering & physics calculations.
Multiplayer Arena Real-time Sync	9	9	6	8	Network Latency / Bandwidth Delta updates & server reconciliation.
Battlestation Builder Social Showcase	8	4	4	2	Storage Volume (500MB) Metadata heavy (tags, coords, URLs).

Resource intensity map for the four target games. Note that 'Subreddit Tycoon' has the highest Redis Write Load, necessitating a buffering strategy, while 'Hexa-Physics' is Client CPU bound.

Data sources: Devvit Capabilities, Phaser Optimization, Redis Game State, Storage Limits

6.1 Game 1: "The Imposter's Daily" (Social Deduction)

- **Concept:** A massive-multiplayer daily "Werewolf" game. Each day, the subreddit is presented with a scenario and must vote to eliminate one "Imposter" (who may be an AI agent or a randomly selected user).
- **Technical Architecture:**
  - **Frontend:** React. Simple, responsive UI for reading the daily "evidence" and casting a vote.
  - **Backend:** Redis HINCRBY is essential here. We cannot store a record for every vote. Instead, we increment counters for each suspect.
  - **AI Integration:** A Gemini 3 instance generates the "Imposter's Dialogue" daily, ensuring it sounds plausible but contains subtle clues.
- **Key Challenge:** Aggregating thousands of votes instantly.

- **Solution:** Atomic Redis counters.

6.2 Game 2: "Subreddit Tycoon" (Incremental/Idler)

- **Concept:** The entire subreddit collaborates to build a virtual city. Comments generate resources (Wood, Stone); Upvotes increase efficiency.
- **Technical Architecture:**
  - **Frontend:** Phaser. An isometric view of the growing city.
  - **Backend:** High write volume is the risk. We implement a "Write-Behind" buffer in the shared library. Stats accumulate in the short-lived process memory and are flushed to Redis only every 5 seconds or upon termination.
- **Key Challenge:** Preventing Redis throttling due to excessive writes.
- **Solution:** Buffering and batching updates using redis.mSet.

6.3 Game 3: "Hexa-Physics" (Daily Puzzle)

- **Concept:** A 3D physics puzzle where users must stack hexagonal blocks to reach a target height without toppling. The wind speed and block friction change daily.
- **Technical Architecture:**
  - **Frontend:** Three.js + Cannon.js (Physics Engine).
  - **Backend:** The server is lightweight. It simply validates the "Solution Input" (coordinates of placed blocks) to ensure no cheating.
- **Key Challenge:** Deterministic physics. The client-side simulation must match the server-side validation.
- **Solution:** We run the physics simulation *strictly* on the client for the visual. For verification, the server runs a headless version of Cannon.js using the exact same seed.

6.4 Game 4: "Karma Markets" (Prediction)

- **Concept:** A prediction market where users bet fake currency on which Reddit posts will reach r/all in the next 24 hours.
- **Technical Architecture:**
  - **Frontend:** React + Recharts. Visualization of "stock" price history.
  - **Backend:** Heavily reliant on the Scheduler. A job runs every hour to fetch the current upvote count of tracked posts via the Reddit API, updating their "price" in Redis.
- **Key Challenge:** Tracking thousands of posts within the 30s timeout.
- **Solution:** The "Time-Slicing" pattern. The update job processes 50 posts, saves a cursor (index), and immediately schedules a follow-up job to process the next 50, continuing until the list is complete.

---

7. Execution & Deployment Strategy

This section provides the actionable scripts and prompts to instantiate this architecture.

7.1 The Setup Script (setup_workspace.sh)

This shell script is the physical manifestation of the architecture. It creates the directory structure and seeds the configuration files, preparing the environment for the Antigravity agent.

Bash

```
#!/bin/bash
# Reddit Daily Games 2026 - Workspace Scaffolder

echo "Initializing Antigravity Workspace..."

# 1. Create Directory Structure
mkdir -p.agent/{rules,skills,workflows,knowledge}
mkdir -p
packages/{shared,game-01-imposter,game-02-tycoon,game-03-physics,game-04-market}
mkdir -p scripts

# 2. Generate Rules (The Constitution)
# Rule: Devvit Constraints
cat <<EOF >.agent/rules/devvit_constraints.md
---
description: Critical Devvit Platform Constraints
globs: ["**/*.ts", "**/*.tsx"]
---
# Constraints
1. Max execution: 30s
2. Max Redis: 500MB
3. No eval() or dynamic code generation.
EOF

# Rule: Daily Mechanics
cat <<EOF >.agent/rules/daily_mechanics.md
---
description: Daily Game Design Principles
globs: ["**/logic/*.ts"]
---
# Daily Design
1. Global Reset at 00:00 UTC.
2. State is persistent (Redis), not memory-resident.
3. Every session ends with a "Share" artifact.
```

```
EOF

# 3. Initialize Shared Package
cd packages/shared
npm init -y
npm install @devvit/public-api @devvit/redis
# (Mocking a shared Redis wrapper creation here...)
cd../..

echo "Workspace Ready. Launch Antigravity and import this folder."
```

7.2 The Master Prompt

This prompt is the "Ignition Key." It is designed to be pasted into the Antigravity IDE's chat interface. It activates the agent, assigns it the persona of a Lead Architect, and instructs it to begin the scaffolding process based on the rules and knowledge we have prepared.

@Gemini 3 Pro
ACT as a Principal Game Architect and Lead DevOps Engineer.
I am initializing the "Reddit Daily Games 2026" workspace. We are building 4 games simultaneously in a monorepo structure.
CONTEXT:

- **Platform:** Reddit Devvit (Node.js runtime, Redis persistence, React/Phaser/Three.js frontend).
- **Constraints:** 30s server timeout, 500MB storage, CSP restricted client.
- **Goal:** Win the "Best Daily Game" category ($15k prize).

YOUR MISSION:

1. **INGESTION:** Scan the .agent/knowledge/ folder. I have deposited Deep Research reports on "Viral Retention Mechanics". Summarize these into a design_principles.md file in the root.
2. **ARCHITECTURE:** Create a shared library in packages/shared/ containing:
    - A RedisLeaderboard class (using ZSETs).
    - A DailyScheduler wrapper (cron handling).
    - A BitPacker utility (for compressing game state).
3. **SCAFFOLDING:** Generate the boilerplate for 4 apps in packages/:
    - game-01: "Imposter" (Social Deduction) - Template: React.
    - game-02: "Tycoon" (Idler) - Template: Phaser.
    - game-03: "Physics" (Puzzle) - Template: Three.js.
    - game-04: "Prediction" (Market) - Template: React + Recharts.
4. **CONFIGURATION:** Set up a .code-workspace file that defines unique ports for each game's local dev server.

EXECUTION MODE:

- Use **"Plan Mode"** for the Shared Library architecture to ensure type safety and error

handling.
- Use **"Fast Mode"** for the UI boilerplate generation.

START NOW.

---

8. Conclusion and Future Outlook

This report outlines a comprehensive engineering strategy for the Reddit Daily Games 2026 Hackathon. By leveraging the Google Antigravity IDE not just as a code editor but as an agentic orchestration platform, we effectively multiply the output of the human developer. The architecture explicitly addresses the core challenges of the "Daily" format—retention, global state synchronization, and platform resource constraints—transforming them from obstacles into design features. The integration of Deep Research ensures that every line of code is informed by analyzing successful retention mechanics, positioning the final submissions to be not just functional, but competitively superior.

Works cited

1. Announcing our Daily Games themed virtual hackathon! : r/Devvit, accessed February 3, 2026, https://www.reddit.com/r/Devvit/comments/1qdnhj5/announcing_our_daily_games_themed_virtual/
2. Scheduler - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/server/scheduler
3. Devvit Web - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/capabilities/devvit-web/devvit_web_overview
4. Redis - Reddit for Developers, accessed February 3, 2026, https://developers.reddit.com/docs/0.11/capabilities/redis
5. Getting Started with Google Antigravity, accessed February 3, 2026, https://codelabs.developers.google.com/getting-started-google-antigravity
6. Build with Google Antigravity, our new agentic development platform, accessed February 3, 2026, https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform/
7. Tutorial : Getting Started with Google Antigravity Skills - Medium, accessed February 3, 2026, https://medium.com/google-cloud/tutorial-getting-started-with-antigravity-skills-864041811e0d
8. Rules / Workflows - Google Antigravity Documentation, accessed February 3, 2026, https://antigravity.google/docs/rules-workflows
9. The MCP Server Stack: 10 Open-Source Essentials for 2026 | by TechLatest.Net | Dec, 2025 | Towards Dev, accessed February 3, 2026, https://medium.com/towardsdev/the-mcp-server-stack-10-open-source-essentials-for-2026-cb13f080ca5c