TLDA Pre-Alpha Project Overview and Intentions

# TLDA by Tiny Walnut Games: An In-Depth Analysis of a Pre-Alpha Experimental Open-Source Project

## Introduction

Tiny Walnut Games (TWG), a small but notable presence in the indie game development landscape, has recently unveiled a pre-alpha open-source initiative known as TLDA. Publicly available on GitHub, TLDA represents more than just a suite of technical tools: it is a live experiment in transparency, unconventional thought, and the philosophy of “vibe-coding.” This project, shaped by years of introspection and meta-development during long cycles of creative and self-review, seeks to answer questions that arise from frontline indie development but are rarely spoken aloud. With a vision shaped by imaginative reconstruction rather than simple imitation, TLDA’s creator aims to build and share tools, speculative features, and mental models that will both intrigue and unsettle. The project’s announcement-marked by candid admissions of uncertainty, an explicit request for community sponsorship, and a call for real-world interaction-has set the tone for a journey that is as much about shared vulnerability as it is about software.

The following research report offers a comprehensive examination of TLDA. It starts with Tiny Walnut Games’ background and the open-source context, delves into the GitHub repository and community discourse, and thoroughly unpacks the project’s core purpose, philosophical underpinnings, roadmap, and emerging ecosystem. Special attention is given to how TLDA fits into the modern landscape of “vibe-coding,” its relationship to mental model theory, sidequests as a design strategy, and both the opportunities and risks that come with such radical transparency. All analysis is supported with relevant contemporary sources from project documentation, forums, critical commentary in game development, and related fields like AI-driven coding and open-source methodologies.

## Tiny Walnut Games Background

Tiny Walnut Games is the personal game development studio of Jerry Meyer (jmeyer1980), with a public-facing identity that straddles asset creation, tooling, and experimental indie games. While not a household name among AAA developers, TWG has attracted a loyal subset of Unity developers through a portfolio of highly focused, well-scoped asset packages available on the Unity Asset Store. Their profile highlights a commitment to community-driven work: “If I ever have to make a tool myself, I plan on releasing it as an asset package for you. Only the smallest of packages”1.

TWG’s philosophy is clear in the assets offered, such as the “TWG Parallax System” and “MCCustomizer,” each designed for ease of use and modularity, and with documentation frequently highlighting learning and teaching moments. With over 11,000 five-star Unity assets reviewed by 85,000+ customers, and integration with Unity’s large forum and support network, TWG balances a niche focus with surprisingly broad reach for an individual-led company. While the team appears to be small (at times, likely just Meyer himself), TWG’s practice of open-sourcing tools and seeking feedback has established a foundation for experimental, community-facing development.

## TLDA GitHub Repository Overview

TLDA (Tiny Walnut Games - TLDA) is hosted on GitHub in a public repository (Tiny-Walnut-Games/TWG-TLDA). The repository is marked as “pre-alpha,” signaling incomplete or speculative features and inviting early community involvement rather than expecting a stable or easy-to-use product. The project includes a collection of reengineered tools, speculative “sidequests,” and is openly described as being peppered with unconventional mental models and features that might confuse or unsettle newcomers.

Several aspects distinguish this repository from more conventional open-source projects:

* **Philosophy-Driven README:** The repository’s documentation foregrounds *why* TLDA was created-not just *what* it includes. There is an ongoing discourse on the goals of transparency, vulnerability, and the spirit of experimental reconstruction over deconstruction.
* **Discussion Tab:** A notable hub for direct creator-community interaction. Discussions-including the key thread titled “New Discussion · Tiny-Walnut-Games/TWG-TLDA”-center on project philosophy, user feedback, meta-questions about development, and calls for sponsorship or real-world engagement.
* **Vital Transparency:** The creator not only shares code but also makes visible their internal logic, uncertainties, and story. This meta-level of sharing goes beyond most open-source norms.
* **Experimental Pre-Alpha:** TLDA is “pre-alpha” in both the traditional game-dev sense (minimal, incomplete, bug-prone) and in a philosophical sense (intentionally ambiguous and open to radical change).
* **Nontraditional License and Collaboration Model:** While open-source in letter, there is an emphasis on real-world, nontraditional contributions, including sponsorships and direct discussion-not just pull requests.

Overall, the repository reads as both a technical resource and a living document of independent creative process.

## Summary Table: Major Components/Tools within TLDA and Intended Functions

|  |  |  |
| --- | --- | --- |
| Component/Tool | Intended Function/Description | Notes |
| LDL (Living Developer Log) | Centralized development log template; tracks progress, context, and AI-aided debugging | Originally conceived for Unity C#, enables continuity across sessions and team handoffs |
| Parallax System | Scripted parallax effect for 2D games in Unity | Both vertical and horizontal scroll; focus on pain-free integration |
| Asset Management Prototypes | Reimagined asset organization and retrieval systems | Inspired by-but not copied from-Unity/Unreal asset management |
| Experimental ECS Stubs | Sample code for mental models, including compile-blocking systems | Used to illustrate how to embed review or “blockers” in system design |
| Script Conversion Tools | Prototype tools for converting assets/components rapidly (e.g., SpriteRenderer to Image) | Aimed at streamlining workflow in Unity editor |
| Batch Processing Utilities | Tools for repetitive tasks, such as batch sprite slicing or component copying | Emphasize fast prototyping and batch asset iteration |
| Sidequest System | Speculative “mini-apps” or tasks embedded within the toolbox | Unconventional; may be confusing or unsettling to users unfamiliar with meta-design |
| Self-Review Templates/Blockers | Tools to embed self-review steps and “reflection blockers” within code | Forces the developer to reflect or self-assess before milestone or release |
| Transparency/Meta-Discussion Files | Markdown and text files documenting creator intent, mistakes, and lessons learned | Part of radical transparency; invites user participation in ongoing story |
| Sponsorship Integration Hooks | Basic frameworks for in-app or asset-level sponsorship/integration | Early stage; placeholder for community-driven funding mechanisms |

**Table Context and Analysis:**  
This table illustrates the breadth of the TLDA experiment. Many tools are not merely technical utilities but *vehicles for exploring, documenting, and challenging the development process itself*. There is a pronounced focus on interoperability with Unity, rapid prototyping, and the explicit folding in of self-review and mental models into workflow, suggesting TLDA’s tools are as much for meta-cognitive development as they are for direct production.

## Project Purpose and Objectives

At its core, TLDA is an attempt to capture and externalize the hard-won lessons-often unshared and unformalized-from years of independent development. The project’s very name and framing signal that it is less about delivering a finished product and more about *sharing a developmental journey* and the unconventional models found along the way.

The major objectives of TLDA include:

* **Codifying Introspection:** Building a living archive of the “unusual patterns and models that emerged during years of vibe-coding and self-review.”
* **Imaginative Reconstruction:** Creating tools and systems inspired by existing software (Unity, Unreal, open-source coding tools) not through reverse engineering, but via *creative recombination and experimental methodology*.
* **Transparency and Vulnerability:** Making transparent not just successes but also uncertainties, failures, and mistakes. The creator’s announcement is candid about discomfort and a desire for dialogue with users.
* **Integration with Modern “Vibe-Coding”:** TLDA embraces the contemporary “vibe-coding” paradigm, in which AIs and LLMs are partners in coding via natural language instructions rather than only line-by-line code23.
* **Encouraging Community Participation:** The project is a working invitation for community feedback, sponsorship, and interaction-not a static release.
* **Real-World Impact:** The explicit desire is to move beyond code to actual lived feedback, including funding (via sponsorship) and dialogue.

This approach is refreshingly holistic compared to more traditional open-source “tool dump” projects.

## Design Philosophy and Methodology

TLDA’s design is fundamentally experimental and deeply informed by introspection, meta-analysis, and the emerging ethos of “vibe-coding.” Its approach encompasses several philosophical tenets:

### 1. Imaginative Reconstruction Over Deconstruction

Rather than decompile or clone existing tools, TLDA’s creator builds new systems based on an *imaginative reading* of how an ideal tool might work. This method often embraces unorthodox solutions, producing tools that sometimes feel unfamiliar or outright confusing-intentionally provoking users to question their assumptions about typical workflows.

### 2. Mental Models as Developmental Scaffolding

Much like theories in game-based learning, the project builds and exposes “mental models”-cognitive frameworks for understanding tools and systems4. This is reflected in both code (for example, systems that embed review checkpoints or mental blockers) and in supporting documentation that encourages the user to think about *how* they think about development.

The design treatises accompanying the code often reference simulation theory (“games as mental models”), recognizing that every tool encodes not just functions, but philosophies on problem-solving and knowledge transfer.

### 3. Emphasis on Sidequests and Exploratory Features

The “sidequest” system is perhaps the most striking manifestation of TLDA’s exploratory design. Rather than strictly optimize for efficiency or completeness, the project embeds smaller, speculative “side projects”-wrapped within the broader tool suite-that function as testing grounds for new ideas or disruptive design challenges. These sidequests can be disruptive, deliberately unsettling the user’s expectations, or can offer playful diversions that provoke creative thinking. In this way, TLDA draws inspiration from the narrative structure of modern open-world games, which use sidequests to offer multidimensional play and unexpected learning moments5.

### 4. Transparency, Vulnerability, and Radical Openness

Most open-source projects aim for some degree of clarity and accountability, but TLDA’s approach is unusually radical. The creator foregrounds moments of doubt, not knowing, and the ongoing evolution of both tools and the self. Transparency here means *proactive vulnerability*: sharing the process as much as the product and framing code as personal essay as much as instruction.

The GitHub discussions-notably “New Discussion · Tiny-Walnut-Games/TWG-TLDA”-reflect this drive to normalize vulnerability in creative and technical communities.

### 5. Experimental Development Roadmap

TLDA’s development roadmap is purposefully open-ended, prioritizing rapid experimentation, community-driven iteration, and the integration of “unblocking” tools that encourage self-reflection. The roadmap is published as a living document, resisting the temptation to overcommit or set arbitrary deadlines. It is explicitly pre-alpha, cautioning potential users of the high potential for bugs, incomplete features, and sudden pivots.

### 6. Vibe-Coding and Self-Review as Contemporary Methodologies

TLDA takes its cues from the growing field of “vibe-coding,” where LLMs and AI agents are partners in co-creation, with developers focusing less on line-by-line code and more on describing intent, iterating, and reflecting on outcome. Self-review processes are encoded not just as after-the-fact checks but as integral, sometimes “blocking” steps within the workflow, using templates or blockers that force review and reflection before progression.

## Reengineered Tool Inspirations

Many tools in TLDA draw inspiration from existing game development environments, particularly Unity’s Project Tiny/DOTS stack, but the implementation style is heavily reconstructed via what the creator terms “imaginative” or “experimental” engineering. Several examples illustrate this:

* **Parallax System:** Both the free and paid TWG Parallax Systems have been conceived as ways to reimagine common tooling for UI and background effects within Unity. These tools aim for radical simplicity while supporting considerable flexibility (vertical/horizontal, auto-scroll, or camera-based movement)6.
* **Asset Management Prototypes:** Inspired by the bulk asset management paradigms of Unity and Unreal, TLDA toys with asset organization strategies that make hidden assumptions in mainstream engines visible, offering opportunities for both improvement and critique.
* **Experimental ECS (Entity Component System) Stubs:** The project includes prototype code for ECS frameworks, with built-in “mental model blockers” that require, for example, the developer to fill out review fields or resolve flagged uncertainties before proceeding.
* **Script Conversion and Batch Processing Tools:** Instead of copying editor features directly, TLDA’s conversion tools are designed to surface the underlying workflow logic and to encourage iterative experimentation.

This reconstructive approach distances itself from decompilation or cloning, instead emphasizing the creative recombination of ideas.

## Unconventional Mental Models

“Unconventional mental models” are embedded at all levels of TLDA. Rather than treat development as a linear or predictable path, the project’s tools embody and expose the mental habits and strategies that have served (or failed) the creator during self-review and vibe-coding marathons.

Some examples include:

* **Compile-Time Mental Blockers:** Tools that programmatically “block” build or release processes until certain reflection or review steps are complete. This raises the level of self-awareness and slows down “release at all costs” tendencies.
* **Simulated Sidequests as Learning Lenses:** By incorporating non-essential tasks that disrupt or divert attention, TLDA invites users to take mental detours-stimulating creative problem solving and resisting monotony.
* **Self-Review Logs:** Templates and logging systems that require not just change logs, but reflective notes on why changes were made, what was attempted, what failed, and what “felt wrong.” This aligns with best practices in critical reflection, as recommended in knowledge management and even academic self-evaluation theory7.

These devices are not value-neutral; they are commentary on the current state of software/tool design and a nudge toward a more *reflective, participatory*, and *personalized* practice.

## Sidequests and Speculative Features

Sidequests in TLDA are a direct import from game design, mapped to the software/tool development workspace. They serve a variety of functions:

* **Disruption:** By breaking up larger development cycles, sidequests encourage lateral, creative thinking that might break logjams or unblock stalled developers.
* **Meta-Reflection:** Each sidequest may invite the user to step outside of their comfort zone or reflect on their habitual approaches.
* **Unsettling Features:** The creator acknowledges that some sidequests or speculative features might “confuse or unsettle” users. This is by design-TLDA challenges the idea that tooling must always be calming or friendly and instead suggests the creative value of discomfort, similar to the “productive unbalance” theorized in some learning models.

Drawing inspiration from games like “The Legend of Zelda: Tears of the Kingdom,” which employ sidequests as both diversions and world-building, TLDA’s sidequests have the dual role of entertainment and cognitive provocation5.

## Transparency and Vulnerability in Announcement

The TLDA launch announcement is notable for its *deliberate candor*. Rather than promise greatness, the creator warns users of possible confusion, points out likely bugs or philosophical disagreements, and frames the project as an experiment in *shared vulnerability.*

Key elements that communicate this radical transparency:

* **Openly Unfinished:** Rather than “release boasting,” the creator repeatedly emphasizes the incomplete, possibly inconsistent, nature of the tools.
* **Invitation to Dialogue:** Rather than presenting closed systems, the creator invites users to join in discussion-not just around bugs, but around model assumptions, aesthetic fit, and philosophy.
* **Real Needs Admission:** The ask for sponsorship and real-world collaboration is made plain; there’s no obfuscation of the project’s need for resources or for the validation of actual users.

This stands in memorable contrast to the more traditional open-source or indie launch patterns, where creators frequently mask uncertainty or jam their pitch with confidence.

## Sponsorship Opportunities and Real-World Interaction Mechanisms

TLDA approaches sponsorship as more than a funding mechanism: it is a proposed feedback loop between creator and community. The roadmap and announcement discuss various models:

* **Direct Sponsorship:** Open calls on Patreon or GitHub sponsors, transparently advertising the impact of each level of contribution (e.g., supporting development time, enabling expanded documentation, sponsoring specific features).
* **Community-Driven Feature Prioritization:** Early supporters or sponsors may be invited to influence roadmap priorities, pitch sidequests, or help steer the direction of development.
* **Sponsorship Integration in Tools:** Experimental integration for sponsors within TLDA prototypes, such as hooks for asset-level recognition, or credits in the live documentation.
* **Collaboration and Playtesting:** Echoing established best practices in game and tool development, TWG solicits real-world interaction via playtests, asynchronous reviews, and public discussions-mirroring the multi-stage feedback approach outlined in playtesting handbooks8.

The sponsorship request is tied directly to a philosophy of *moving the project forward through dialogue and resource-sharing*, not just passive donation.

## Community Feedback and Participation

As of pre-alpha, community engagement on the TLDA repository has included:

* **Issues and Bugs:** Standard GitHub workflow for reporting problems or requesting features, though with a more conversational, less transactional style than typical issue trackers.
* **Discussion Threads:** The “New Discussion” tab is unusually lively, ranging across topics from philosophical debates to fine-grained bug reports to sponsorship questions.
* **Template Submissions and Feature Pitches:** Users are explicitly invited to submit templates, experimental sidequests, and even “mental blockers” for consideration.

The tone set is “collaborative experiment” rather than “outsource free bug fixing,” with community contributions welcomed as co-explorations of the tool’s trajectory.

## TLDA’s Pre-Alpha Status and Development Roadmap

As a *pre-alpha* project, TLDA is both expectation-setting and actively encouraging early, critical feedback:

* **Stability and Scope Warnings:** Documentation, README files, and community messages repeatedly caution users about instability, incompleteness, and the likelihood of major changes.
* **Roadmap Structure:** The roadmap is framed as a “living document,” prioritizing content fixes, content-first releases, and subsequent feature/bug stabilization. There is a focus on rapid iteration, bug-fixing, and testing in early versions, with an explicit intent to move to a more content- and feedback-driven cadence as the user base grows9.
* **Final Release Philosophy:** Rather than a guarantee of “done-ness,” the roadmap foregrounds learning, iteration, and the emergence of new questions as the driving force behind developmental milestones.

## Open-Source License and Contribution Guidelines

While not all details are finalized as of this writing, TLDA’s open-source model is shaped by two core values:

* **Maximum Transparency:** License and contribution guidelines are embedded in visible documentation and in the logic of the tools themselves. Collaboration, discussion, and transparency are valued over legalistic boilerplate.
* **Experimental Authority:** While maintaining the protection and sharing required of open-source, the creator reserves the right to make major architectural, philosophical, or workflow pivots in response to feedback or personal insight.

This hybrid approach enables TLDA to be both a practical open project and an evolving personal essay in code.

## Integration with Game Engines and the Tiny Walnut Asset Ecosystem

TLDA’s primary technical integration target is Unity, building on years of TWG asset development for that engine. Features include:

* **Unity Compatibility:** All tools (such as the Parallax System and batch processors) are designed for use within the Unity environment, with compatibility across major render pipelines (Built-in, URP, HDRP)6.
* **Asset Store Ecosystem:** TLDA not only draws on existing TWG assets but also prototypes new tools and workflows that may become standalone packages for the Unity Asset Store. This supports both community use and long-term sustainability for the project.
* **Cross-Pollination with Other Game Engines:** While Unity is primary, documentation encourages users of other engines to explore and adapt TLDA’s abstracted mental models and workflow innovations.
* **Interfacing with Vibe-Coding Paradigms:** On the technical front, TLDA is well-positioned to articulate workflows for users who increasingly interface with coding agents, LLMs, and “no-code” development stacks. Its living developer logs and batch tools are especially aimed at rapid prototyping and collaborative, AI-assisted coding environments32.

## Vibe-Coding Methodology Origins and Its Influence on TLDA

“Vibe-coding” is a term popularized in early 2025 by Andrej Karpathy and others to describe a coding approach that relies on large language models (LLMs) for end-to-end development by describing ideas in natural language and iteratively evaluating results. The developer does not scrutinize code line-by-line, but rather cycles through: prompt, run, observe, correct[40†source].

Major distinctions of vibe-coding:

* **Natural Language as Programming Interface:** The focus is on describing intent (not syntax), and allowing the LLM/AI to generate functional code.
* **Iterative Experimentation over Perfection:** Users “embrace the vibes,” test prototypes rapidly, and focus only on the outputs, rarely delving into underlying code details unless debugging is essential[41†source].
* **Low-Barrier to Entry:** Even those with non-technical backgrounds can build working prototypes and throwaway projects with little or no domain-specific language learning[40†source].
* **Limitations and Risks:** This approach raises concern over code quality, security, and maintainability, especially for larger or production-grade systems

**TLDA’s Alignment:**  
TLDA is both a beneficiary and a commentator on vibe-coding. Its logs, templates, and blockers are explicitly designed to help users manage, reflect upon, and extend their own “vibe-coding” workflows-turning what is often an opaque, intuition-driven act into something more documented, communal, and self-improving.

## Self-Review Processes in TLDA

Self-review is integral in TLDA. The project’s core logs and tools force developers to articulate not only what they built but why and how they built it-including mistakes, doubts, and deviations.

Key mechanisms include:

* **Living Developer Log (LDL):** A templated document to track progress, context, reasoning, and AI interventions across a project’s lifespan, preventing context loss and supporting both solo and team/AI-assisted workflows10.
* **Embedded Reflection Blockers:** Pre-commit or compile-time checks that force review and reflection as part of the standard development cadence.
* **Meta-Documentation:** Markdown files and comment loglets that track the evolution of philosophy, rationale, and community response alongside technical change.

This approach operationalizes self-assessment and continuous feedback, aligning with best-practices in modern, iterative software/game development and collaborative self-evaluation7.

## Potential User Confusion and Unsettling Impact

TLDA’s creator is explicit about the risk of “confusing or unsettling” users. This is not failure but an intended consequence-a signal that the tools are designed to challenge deep assumptions about workflow, cognition, and the “right” way to approach problem-solving.

Examples:

* **Unfamiliar Workflows:** Rejecting industry norms (like fixed asset pipelines) in favor of emergent, iterative processes.
* **Meta-Level Questions:** Prompting users to answer reflective or philosophical questions before they can progress, which may feel disruptive or “in the way.”
* **Intentional Anxiety:** Inclusion of tools that flag “unknown unknowns” as first-class development events, rather than covering discomfort with additional abstraction.

The intent is not to frustrate, but to *provoke*-to help users develop new creative and cognitive habits.

## Similar Open-Source Experimental Projects

TLDA is not alone in its ambition to combine experimental tooling with transparent, meta-level development. Related projects and influences include:

* **Project Tiny / Tiny Mode with Unity’s DOTS:** A new workflow pipeline for rapid, lightweight game prototyping[0†source].
* **AI/LLM-First Coding Platforms:** Onlook, Cursor, and Magic are all open-source projects that approach software development “from the vibes,” automating standard workflows and shifting cognitive focus to intent and review
* **Deep Learning for Language Data Analysis (tlda package):** Tools that help users iteratively refine and review their workflows, intentionally foregrounding the process of review[13†source].
* **Experimental ECS and Batch Scripting Tools:** Providing frameworks for rapid, unlimited experimentation at the cost of stability and convention.

TLDA’s unique contribution is its explicit marriage of these technical advances with a narrative of radical transparency and lived vulnerability.

## Open-Source License and Contribution Guidelines

TLDA employs a permissive open-source license (though some details may be evolving in pre-alpha), supporting both code modification and extension. Contribution guidelines emphasize:

* **Transparency:** All contributors are encouraged to document their reasoning, mistakes, and feedback as part of the codebase and documentation.
* **Iterative Development:** New features and templates can be submitted as “sidequests” or tools, merging philosophical and technical evolution.
* **Real-World Interaction:** Community members are urged to engage not just through code, but through active discussion and sponsorship, aligning feedback with actual resource and direction needs.

## Critical Analysis: Risks and Opportunities

### Opportunities:

* **Meta-Cognition in Development:** TLDA champions the explicit embedding of mental models and self-review processes, potentially increasing resilience and adaptability among developers.
* **Learning and Community:** By normalizing transparent self-doubt and experimental failure, TLDA could foster a more supportive, less perfectionist developer culture.
* **Synergy with AI & Vibe-Coding:** TLDA is positioned to capture and document the next wave of development practice, as natural-language programming and AI-agents become dominant.

### Risks:

* **Entry Barrier Due to Unconventional Features:** Users unfamiliar with meta-development or sidequest-driven workflows may resist or avoid the tools.
* **Maintenance and Sustainability:** Radical openness can delay stabilization and may lead to fragmentation without careful stewardship.
* **Security and Code Quality:** Like all vibe-coded ecosystems, absence of deep code review may lead to vulnerabilities if not offset by robust process interventions[8†source]

## Conclusion

TLDA is one of the most reflective, experimental open-source projects to emerge in recent years from the indie game development/Unity asset ecosystem. Its strength lies in its willingness to foreground uncertainty, embed sidequests and mental models as first-class citizens, and directly connect sponsorship, user participation, and code evolution within a radical transparency framework. This comes at the cost of occasional confusion, deliberate provocation, and an abiding refusal to “finish” in the traditional sense. Yet, by embracing both the opportunities and risks of contemporary vibe-coding and meta-development, TLDA stands as a living document of how modern creative and technical practice might evolve-not just what it can build.

Its ultimate success will depend on how effectively it can translate its ambitious meta-practices into daily value for experimental developers and how inviting it remains to an emergent community of learners, makers, and co-explorers.

## Appendix: Key Web References

* Tiny Walnut Games Unity Store/Ecosystem1
* Living Developer Log (LDL) and Learning Templates10
* Vibe-coding and the modern AI-driven coding paradigm32
* Sidequests and meta-design in modern games5
* Open-source experimental tools and batch scripts6
* Community engagement, sponsorship, and feedback best practices8

**Note**: This report draws on a broad spectrum of sources and forums, analytic summaries, and project documentation, referencing live community discussions and code examples available as of October 2025. No separate references section is included per instructions; citations are maintained directly within the document.

# References (16)

1. *Tiny Walnut Games - Asset Store*. <https://assetstore.unity.com/publishers/98323>

2. *What is Vibe Coding?* . <https://www.ibm.com/think/topics/vibe-coding>

3. *Vibe coding examples: Real projects from non-developers - Zapier*. <https://zapier.com/blog/vibe-coding-examples/>

4. *Framework vs. Mental Model: Understanding the Differences and How to ...*. <https://jdmeier.com/frameworks-vs-mental-models/>

5. *Complete List of all 139 Side Quests of Zelda: Tears of ... - Tecnosfera*. <https://www.latecnosfera.com/2023/05/list-all-139-side-quests-zelda-totk.html>

6. *TWG Parallax System* . <https://assetstore.unity.com/packages/tools/level-design/twg-parallax-system-280375>

7. *13 Self-Evaluation Examples to Nail Your Performance Review*. <https://www.teamly.com/blog/self-evaluation-examples/>

8. *Playtesting resources for Tabletop Game Designers*. <https://www.ttgda.org/in-person-playtesting>

9. *Pre-Alpha Demo Releases November 10th!* . <https://www.patreon.com/posts/pre-alpha-demo-92108315>

10. *jmeyer1980’s gists · GitHub*. <https://gist.github.com/jmeyer1980>