From Layoff to Innovation: GenAI Game Tool Showcasing Full SDLC

Krishnamohan Malladi

km_malladi@yahoo.com

Abstract

In May 2024, after an unexpected layoff, I found myself with time—something rare in our fast-paced industry. Rather than dwell on the unknown, I turned to an idea that had long been in the back of my mind: building a game that could serve both a technical purpose and a social one. What if I could create something that supported my community, kept me technically sharp, and explored the power of AI?

This paper is the story of that journey—from concept to a fully functioning, GenAI-powered word game designed for the Telugu-speaking community. Built entirely by me, the project covers the complete software development lifecycle (SDLC), blending quality assurance principles with emerging AI technologies like prompt engineering, OpenAI APIs, and dynamic content generation.

More than just a technical exercise, this experience taught me what it means to lead with quality, think like a full-stack developer, and use modern tools to build culturally meaningful and scalable solutions. I hope it inspires QA professionals and technologists to see themselves not just as testers or coders, but as innovators capable of creating impact beyond their day jobs.

Biography

Krishna is a seasoned IT Quality Assurance professional with a rich background working across industries—from nimble startups to large-scale global enterprises. With a strong passion for delivering high-quality software, Krishna has deep expertise in both functional and non-functional testing, test management, and automation.

Committed to continuous growth, Krishna stays ahead of the curve by regularly upgrading technical skills and exploring emerging technologies such as AI/ML in QA. After pursuing a postgraduate course in AI/ML, Krishna applied this knowledge to an innovative side project—developing a GenAI-powered educational game designed for the South Indian Telugu-speaking community.

Outside the professional realm, Krishna is an avid traveler and storyteller. A recent adventure includes hiking Mt. Kilimanjaro. Krishna also writes short stories in Telugu and enjoys peaceful walks in nature.

1. Introduction

I've been in the software industry for many years, with a strong foundation in quality assurance. While testing has been my core area, I've always been curious about the responsibilities and perspectives of other roles—developer, DevOps, production support, and even sales. As I explored these areas, I found myself constantly thinking like a QA professional—questioning assumptions, validating outcomes, and pushing for the best in every phase. In a way, I developed what I call a "split personality," switching hats to understand the full 360 degrees of software development. That desire deepened even further after I completed a postgraduate course in AI/ML, fueling my motivation to experience the end-to-end lifecycle hands-on.

After being laid off, I decided to use that unexpected free time to pursue a personal project I had long envisioned—building an online word game for the Telugu-speaking community across all age groups. This project became more than just a way to give back to my community; it allowed me to apply the knowledge I gained from my AI/ML coursework and, more importantly, immerse myself in every phase of product development—from ideation and coding to testing, architecture, and user feedback.

2. Problem Statement

Telugu is spoken by approximately 96 million people worldwide, making it the fourth most spoken language in India and ranking 16th globally. Notably, Telugu has been identified as the fastest-growing language in the United States, experiencing an 86% increase in speakers between 2010 and 2017.

Despite this growth, many Telugu speakers, especially within the diaspora, are gradually losing touch with the language. Factors contributing to this decline include demanding work schedules, the predominance of English in professional settings, increased mobile device usage, and multicultural family environments where Telugu may not be the primary language.

While various organizations in the U.S. and Telugu state governments in India have initiated programs to promote the language, there remains a need for more engaging and intuitive methods to encourage its use. Developing an entertainment-based game presents an innovative approach to reconnect individuals, particularly younger generations, with the Telugu language in a fun and interactive manner.

3. Brief Introduction of the Game

Link to the game:

Telugu Word Game - Registration

I won't go into every detail, but here's a quick overview of the game. The welcome page includes a registration form where users can optionally enter their details to personalize their experience. If they prefer not to share information, they can skip registration and jump straight into the game.

Once users click "Start Game," various categories are displayed, allowing them to choose and begin playing. The game was built with two primary goals:

1. **Mental Stimulation** – Guessing each letter of a word in just four attempts challenges the mind. While most words may be familiar, the process of guessing within limited tries enhances focus and builds resilience.

2. **Enjoyment**:- It's designed to be a fun, engaging way to spend time.

For younger users, there is a special **"Kids Click Here"** link on the welcome page that lets them access the game directly without entering any details. The main objective is to encourage Telugu language learning in children—by playing daily, they can pick up at least one new word in a fun and interactive way.

4. The Journey:

As mentioned earlier, my primary goal in building this game was to experience the end-to-end software development process firsthand. Although my core expertise lies in software testing, this journey allowed me to wear many hats—product owner, developer, designer, tester, and even early-stage marketer.

One key realization throughout this process was that my QA instincts remained active no matter which role I was playing. Whether I was building a proof of concept or developing a new feature, I constantly tested my work against the requirements. This continuous feedback loop helped me catch issues early and iterate quickly, even when I was working solo.

The idea began with identifying a socially impactful opportunity: promoting the Telugu language through a fun, engaging tool. I transformed that idea into a business case and began validating it. I reached out to a few Telugu cultural association organizers and pitched the concept. Their response was overwhelmingly positive—they saw potential and asked for a demo version. They expressed interest in presenting it to their leadership teams, with the intention of possibly using it for public events such as online competitions and community engagement.

That early encouragement gave me a huge motivational boost and confirmed that I was onto something valuable—both culturally and educationally.

A.Tackling Technical Challenges in Multi Script Game Design: A Story of Resilience

I chose JavaScript to build the game, using HTML and CSS for page structure and styling. As I transformed my idea into working features, I began to encounter limitations—especially given the linguistic complexity of Telugu. I'll outline how I tackled each challenge, one step at a time.

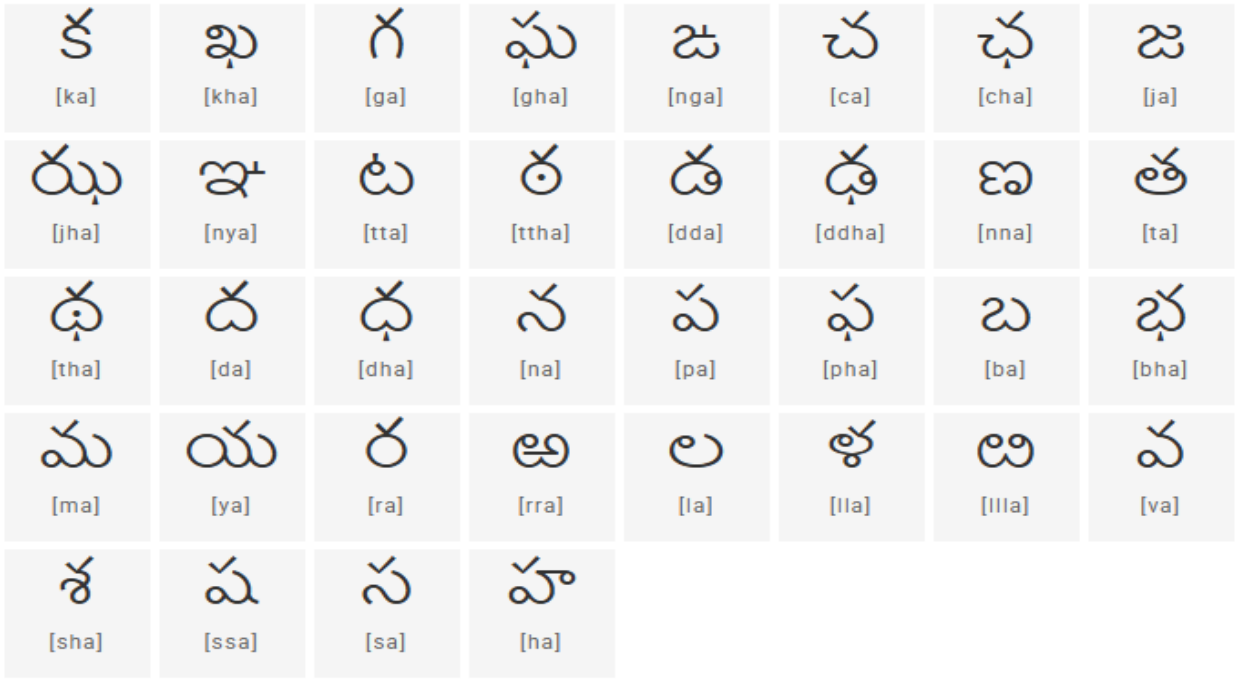
The game features a virtual keyboard for Telugu letters. Unlike English, which has 26 letters, Telugu has 52 commonly used characters (historically 56, though 4 are now obsolete). Among these, 16 are vowels—2 of which are rarely used and considered *independent vowels*—and 36 are consonants. The complexity increases with the addition of 15 *dependent vowels*, which are not standalone characters but diacritic marks that modify consonants. They can appear above, below, before, or after a consonant, creating a combinatorial script structure.

Telugu Language Letter Structure: Independent Vowels, Consonants, and Dependent Forms

Independent Vowels:

Dependent Vowels:

Consonants:



Initially, I attempted to create a full virtual keyboard including all characters, but it quickly became impractical due to space and usability constraints.Initially, I explored building a full keyboard with all Telugu letters, but I then pivoted to a more focused design: including only consonants and independent vowels, as these form the structural base of most Telugu words. Dependent vowels were intentionally left for players to infer, a challenge familiar to both learners and native speakers.

Additionally, the virtual Telugu keyboard is fully responsive, automatically adjusting to fit desktop, mobile, or tablet screens, ensuring a seamless experience across devices

For example, if the word was “మాయాబజార్” (Maya Bazaar) from the movie category, the base consonants are మ, య, బ, జ, and ర. My early idea was to use JavaScript libraries to dynamically insert dependent vowels when users guessed the correct consonants. However, after testing several options, I realized these libraries either lacked the flexibility I needed or failed with complex words.

I turned to ChatGPT, which suggested trying regular expressions to match patterns dynamically. While this helped in some cases, it was not a comprehensive solution. After multiple iterations, experiments, and considerable frustration, I almost abandoned the idea. Still, the challenge remains an opportunity: I believe this problem can be solved programmatically with a fresh approach, and I’m committed to revisiting it in the next development phase.

B. Developer’s Hat: Solving Script Complexity with Smart Data Structures

As we all know, JavaScript Map objects are powerful data structures that allow us to store key-value pairs with preserved insertion order. Unlike regular objects, Maps support keys of any data type—objects, functions, or primitives—making them ideal for dynamic scenarios.

To handle Telugu word decomposition and display behavior, I leveraged Map to store letters and their expected visual representations:

export const teluguWords : సినిమాలు: { word: "మాయాబజార్"}

Category :సినిమాలు (Movie) , Word : ‘మాయాబజార్’ (Movie name).

export const keyValuePairs = { మ: "మా", య: "యా", బ: "బ", జ: "జా", ర: "ర్" }

How it works:
When the user selects a letter—say ‘మ’—if it exists in the target word, “మా” appears in the appropriate game slot. This solution worked well in early tests, and I initially felt a sense of breakthrough.

However, my QA instincts kicked in.

Repeated Letters - Different scenario :

In this case, the letter ‘క’ appears twice—once as “కా” and again as “కి”. Using a list as the value solved part of the problem, but not all. The logic for handling dynamic key-value substitution started failing in loops during game play.

export const teluguWords : ప్రదేశాలు : (word: "కాకినాడ"}

Category :ప్రదేశాలు(Places) , Word: "కాకినాడ" (city name)

export const keyValuePairs = word: { క: ["కా", "కి"], న: "నా", డ: "డ" },

Refining the Solution

After several failed iterations and debugging cycles, I explored two main options:

1. **Reveal All Instances**: Automatically display all occurrences of a guess letter.
2. **Controlled Reveal**: Reveal one instance, then remove it from the list so other positions still need to be guessed.

After testing both, I chose **Option 2**—removing the revealed letter from the list after display. This added a layer of difficulty while preserving user interaction.

The final solution worked consistently across test cases, including (1) words with unique letters and (2) words containing repeated letters. Feedback from family and friends supported this approach, noting that it struck a good balance between challenge and playability

Before pushing code to GitHub, I applied standard linting rules using VS Code to ensure the code adhered to clean coding principles and best practices. This step helped maintain consistency and readability as the project evolved. I often created feature branches for new functionalities, merging them into the main branch once complete and then deleting the branch — mirroring structured workflows in professional SDLC. Beyond technical benefits, this process kept me accountable. Each commit and merge acted as a visible milestone, documenting my progress and motivating me to keep going, particularly when I was going through challenging phases of development. Seeing the cumulative changes over time gave me the confidence that I was steadily moving toward the final product.

C. Architecture Hat: Integrating GenAI for Scalable Word Generation

Initially, I relied on a static file to feed the words for the game. However, as the game evolved—especially with the complexity of Telugu script—it became clear that manually curating and maintaining these key:value pairs was not a scalable or sustainable solution.

This led me to explore GenAI and prompt engineering.

My idea was simple:
When a user selects a category (e.g., “Movies”), the application would use OpenAI to fetch relevant Telugu movie titles from the internet and dynamically construct the required key:value pairs for gameplay.

To test this theory, I built a **proof of concept (PoC)**:

- I used the OpenAI API to send a category-specific prompt (e.g., *“Give me five popular Telugu movie titles”*).
- The API returned relevant results, formatted in clean and usable text.
- I then parsed the response into a JavaScript object as a **key:value pair** for the game engine (e.g., `movies: ["మాయాబజార్", "బాహుబలి", "సీతారామం"]`).

- This validated my hypothesis: it was technically feasible, fast, and highly relevant.

I plan to implement full OpenAI integration in the coming months. This shift to dynamic word generation offers several benefits:

- **Reduced manual effort and maintenance**
- **Real-time content freshness**
- **Scalability** for expanding categories
- **Alignment with modern GenAI-driven architectures**

Ultimately, the PoC gave me the confidence that the approach is both technically viable and valuable in delivering a scalable, intelligent game experience

To ensure resilience, I also maintained a backup static data file with a limited set of words. This acts as a fallback when there are network issues or latency. Additionally, the static file can be periodically refreshed with new words from OpenAI, reducing the need for manual intervention.

Originally, all logic was client-side. But as complexity grew, I refactored the system into a **client-server architecture**:

- **Client Side**: Handles the user interface using HTML, CSS, and JavaScript.
- **Server Side**: Communicates with the OpenAI API, performs prompt processing, and fetches dynamic content from the internet.

This separation improved performance, security, and scalability while laying a foundation for future enhancements—such as user authentication, analytics, and leaderboard functionality.

D.Security Hat: Safeguarding Code and Logic

Initially, I implemented the entire game logic on the **client side**. However, I soon realized a potential risk: what if someone cloned the GitHub repository and manipulated the code?

To address this, I explored JavaScript obfuscation. JavaScript obfuscators help **protect code and intellectual property** by making the code difficult to read or reverse-engineer. While obfuscating HTML and CSS isn't very effective, I successfully applied obfuscation to all critical JavaScript and data files.

Although I later planned to split the application into **client and server components**, I still chose to obfuscate the client-side JavaScript to offer an additional layer of security—especially during early-stage deployments

E. DevOps Hat: Iteration Through Continuous Integration

Throughout the development process, I continuously tested the application in real time. I used the **"Go Live"** extension in **Visual Studio Code**, which allowed me to view changes instantly in the browser

without needing a manual refresh. This helped ensure immediate visual and functional feedback during development.

In parallel, I regularly committed updates to **GitHub**, leveraging **GitHub Pages** to host the application for free. This enabled quick deployment of the latest version, which I shared with family and friends for testing and feedback. Their hands-on interaction with the game served as an informal end-to-end testing loop, identifying issues and offering suggestions early in the cycle.

Key enhancements implemented based on user feedback include:

- Managing score history
- Revealing only one letter per correct guess
- Adding a user help section
- Ensuring cross-device compatibility

While formal test automation (unit, component, or E2E) wasn't yet implemented, this iterative and feedback-driven approach helped shape a more polished, intuitive, and user-friendly product. In future iterations, I plan to add automated test scripts to further strengthen the quality pipeline and support CI/CD practices

5. Summary

What began as a personal experiment turned into a deeply fulfilling and technically rich experience. I spent several weeks building this game incrementally—testing, coding, refining, and adapting through every phase of development. One volunteer from a non-profit Telugu organization was impressed with the outcome and expressed interest in presenting it to their leadership team, with the goal of sharing it publicly. That kind of validation meant everything—it confirmed this wasn't just a side project, but a solution with real potential for community engagement.

This journey took me well beyond my QA roots. I played the roles of product owner, developer, architect, and tester—all while maintaining the mindset of quality. Along the way, I gained practical experience with client-server architecture, prompt engineering, dynamic content workflows, and JavaScript data modeling—all while building something personal, purposeful, and functional.

Though the project was rooted in promoting the Telugu language, the technical framework I developed—category-based content, script-aware data mapping, AI-driven content generation—is applicable to any language-learning or cognitive training tool. The architecture can scale, and the patterns are domain-agnostic.

6. Future Roadmap

I have already integrated the kids' and adults' versions into a single platform to cater to all age groups. By the time this paper is published in October, I am confident that AI capabilities—such as dynamic content generation and smarter user interactions—will be fully incorporated. I also plan to host the game on a public domain, making it accessible to a wider audience and more impactful for the community. Looking

ahead, I intend to implement automated end-to-end testing using tools like **Cypress** or **Playwright**. My goal is to integrate these tests into the development pipeline so that, upon each code push to GitHub, the automation scripts run immediately to validate the application's stability. This approach not only supports continuous testing but also aligns well with modern **CI/CD practices**, ensuring quality and reliability as the product evolves.

7. Lessons Learned (Post-Mortem)

As I embarked on this solo development journey, I encountered several technical and design challenges. Notably, existing JavaScript libraries lacked the flexibility needed to support Telugu script complexity—particularly with issues like handling repeated letters in a word and integrating dependent vowels. My initial reliance on static data also proved limiting in scalability and user experience.

However, this experience taught me a powerful lesson: rather than settling for limitations, deeper research and experimentation often lead to viable solutions. With today's advancements in GenAI, prompt engineering, and modern web development tools, even complex problems can be approached creatively.

I also learned the value of **continuous user feedback**. Early testing with family and friends helped me identify usability gaps and refine features incrementally—far more efficient than fixing issues at the end.

Finally, stepping outside of my comfort zone was one of the most transformative aspects. Wearing multiple hats—from analyst to developer, tester, and even DevOps—allowed me to appreciate the depth and breadth of the software development lifecycle (SDLC). This holistic perspective reinforced my QA mindset and strengthened my belief that quality is a shared responsibility across all roles.

8. Acknowledgements

I extend my heartfelt thanks to my family and friends for their continuous feedback and encouragement. Special thanks to the Telugu community organizations that showed interest and support. I also want to acknowledge the role of ChatGPT and GenAI in helping accelerate this project and making experimentation easier and more insightful.

9. Closing Thoughts

This project reminded me that being a QA professional doesn't mean staying in a testing box. It means questioning systems, anticipating failure points, and thinking deeply about user experience. These skills translated seamlessly into every phase of development. In many ways, my QA mindset was my compass—guiding design decisions, debugging challenges, and validating ideas before they became features.

More importantly, this experience reaffirmed a truth I hope to share: that curiosity and resilience can turn any career pause into an opportunity for reinvention. With emerging tools like GenAI, solo developers, QA engineers, and technologists at any level can now build, iterate, and deploy meaningful products at unprecedented speed.

I hope this paper encourages others to explore, create, and lead—not just in their job titles, but in their thinking.

References

About Telugu Language:

https://en.wikipedia.org/wiki/Telugu_language

About Telugu Letters:
https://www.brownpundits.com/2024/09/11/is-this-language-or-music/?utm_source=chatgpt.com

Graphical view of Vowels and Consonants:

https://www.easyteluguTyping.com/telugu/letters

JavaScrip Maps:

https://www.w3schools.com/js/js_maps.asp

Prompt Engineering concepts:

https://www.geeksforgeeks.org/what-is-an-ai-prompt-engineering/

JavaScript Obfuscator:

https://www.javascript-obfuscator.com/