Jump Quest Postmortem

Marcel - Level design and key contributions to the first level
Vinz - Programming, project management, and overall development

September 2024

Contents

1 Project Overview

This project evolved from an experimental Twitch API exploration test into a practical solution for a competitive gaming community. The goal was to create an interactive game integrated with Twitch chat, designed to maintain viewer engagement during downtime in weekly tournaments.
Key aspects:

- Utilises Twitch API and chat functionalities
- Allows real-time audience participation through chat commands
- Enhances viewer experience during breaks between matches
- Demonstrates innovative use of streaming technology

The project showcases the ability to transform a prototype into a valuable tool that addresses a specific community need while exploring new possibilities in interactive streaming experiences.

2 Project Timeline

2.1 Experimental Phase

03/08/24 - 07/08/24

- Successfully read and write to Twitch API
- Message parser for commands
- Create Godot mono project
- Basic player jump physics

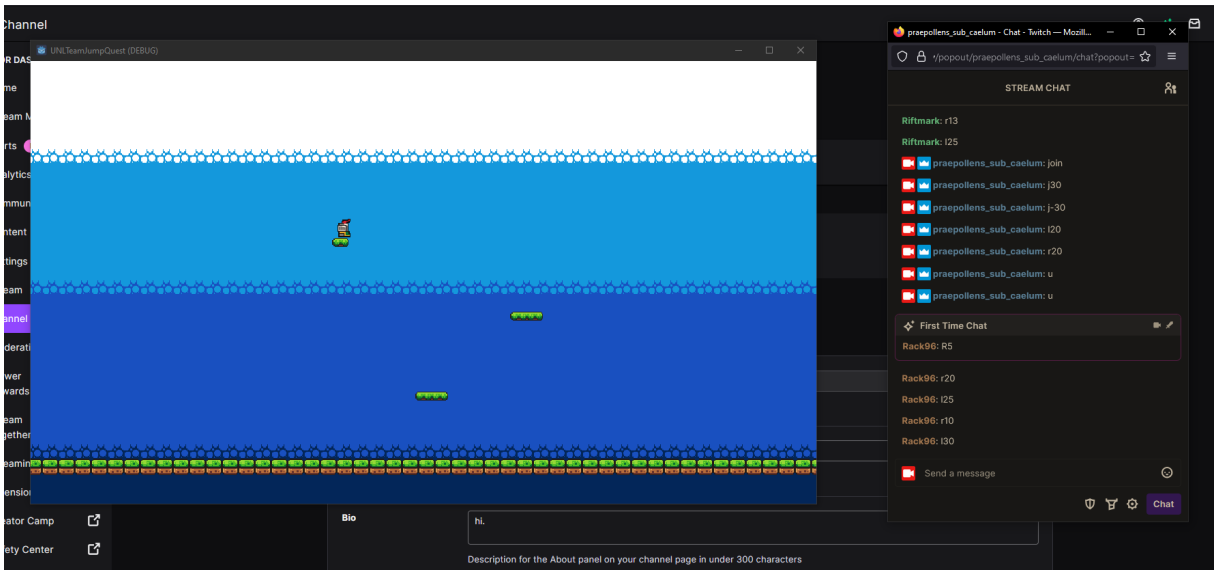

Figure 1: End of Experimental Phase

2.2 Development Phase

07/08/24 - 23/08/24

- Incorporate git version control
- Marcel joins, I teach him Godot and Github
- Debug tool for Marcel to spawn and delete character without Twitch chat
- Camera tool for debugging purposes
- Calculating highest distance to landing distance for faceplant feature
- Created 'Manage Teams' interface where you can create teams that you can import and export to .json. Base64 image encoder so you only need to share the .json file.
- Settings menu that you can connect to your Twitch Access Code and username that is encrypted.
- Create shader to change sprite colours and to make pixelated text more clearer.
- Menu UI Action Reactor system i.e., 'Floating Text Message' to show you pressed buttons.
- Chat box display, game timer, and team score display added
- Robust point scoring implemented. tl;dr, Store last 10 platforms, find average height, only get points when above height or same height as those platforms.

- Resetting of players, given players personal stats that are given to GameManager.cs
- End game screen to show players stats
- Add combo system

2.3 Client Feedback and Testing Phase

23/08/24 - 31/08/24

- Formatting 1 - 4 length strings for team abbreviation
- More readable where your player is at
- 'Easy Mode' feature so everyone can learn the controls and not die to a single player going to the top.
- Streamer QoL features like auto play, esc/pause, and other settings.

Jump Quest demo on YouTube: `https://www.youtube.com/watch?v=A2n6WIh7ekM`

3 Goals and Objectives

3.1 Initial Project Goals

- Explore Twitch API Integration: The project began as an experimental dive into Twitch API functionalities, aiming to create interactive viewer experiences.
- Develop a Chat-Integrated Game: Set out to create a game playable through Twitch chat commands, enhancing viewer participation.
- Address Community Needs: The goal evolved to solve a specific problem for a niche community - maintaining viewer engagement during tournament downtime.

3.2 Evolution of initial goals during development

The goal didn't really evolve and the main idea for the project stayed the same which was interactive chat based video game experience. The only changes that were made during development was on how to better utilise the livestreaming interactive chat. These changes include; using Twitch's integrated chat points system to impact the game, how to better adapt platformer gameplay controlled via text to be more engaging for the viewers, and how to give the streamer a lot more control of the outcome to the game ie. Instead of creating a specific gameplay loop he is able to build on the current one with different game modes' in other words an additional 'Easy Mode' setting another 'TillEveryoneDies' section. These game modes that stack upon each other will give the streamer a more bespoke game design loop that is able to dynamically change to fit the skill of the players.

4 What went right

4.1 Effective Task Documentation and Planning

Throughout the project, I maintained a clear backlog of tasks in a simple .txt file, which served as a sequential checklist for ongoing and future work. This method allowed me to map out immediate steps, prioritise new features, address bugs, and plan for testing efficiently. Each day, before ending development, I would document the next set of tasks to tackle, ensuring a smooth transition into the following day's work. This helped me stay organised but also allowed for more efficient time management and quicker focus when starting development sessions.

4.2 Gradually Building Around a Core Mechanic

When tackling a task, such as creating a point system for when a player jumps to a new platform, I would start by implementing the most basic version of the mechanic – in this case, awarding points when the player lands on a new platform. From there, I would iteratively enhance the system. For instance, I added a function to store the player's highest y position as a variable to determine when points should be awarded. Later, I introduced conditions to reward points for forward progression, even when the platform's y position remained the same.

This approach allowed me to refine the system step by step until I was satisfied with the results. After several days of testing, I ultimately decided that the player should store the last ten platforms in an array and only be awarded points if the new platform's y position exceeded the highest in the array or matched the current distance. However, reflecting on this process, I realised that gradually adding complexity to an initially simple system made it harder to refactor and incorporate new features later on. For example, when I needed to store dead players' stats for the 'Hall of Stats', I had to create an intermediary class to manage communication between two older systems – the UNL.Teams class, which stored team points, and the Player.cs class, which managed individual player points.

Nevertheless, building out mechanics in this way helped me stay motivated throughout development, as each small success provided the momentum to continue working on what was my first project.

4.3 Slowly Learning New Concepts

I approached problem-solving by first using my existing knowledge. I would break down complex tasks into simpler components, which made them easier to handle. If I was confident in my solution but saw inefficiencies, I would research how others typically solved similar problems. After reading through alternative methods, I assessed whether I could incorporate them into my current architecture and, if feasible, refactored my code to implement the improvements. If refactoring wasn't possible at that stage, I kept the new concepts in mind, documenting them for future use.

My focus wasn't on perfecting each system or class, as that would have hindered my overall progress. Instead, I viewed it as an opportunity to improve in future projects. Since the game's architecture wasn't designed holistically but rather developed piece by piece, I accepted that I couldn't incorporate every new idea seamlessly. This iterative learning process allowed me to strike a balance between immediate progress and long-term development growth.

5 What went wrong

5.1 Inefficient Task Documentation

Although, I was content with my task documentation there was much to be improved. At the start of the project, I didn't document my progress effectively, as I was merely experimenting with the Twitch API. This lack of initial structure led me to rely on a simple .txt file to track tasks. When I added Marcel to the team, I briefly tried using the Miro collaboration board. While its features were promising, particularly for Kanban-style task management, we didn't stick to using it. We only used its collaboration tools during one testing session, and found ourselves using Discord screen share to discuss ideas instead.

The slow performance of Miro, compared to the simplicity of a notepad file, also discouraged me from using it regularly. In hindsight, I should have prioritised a more structured task management system from the outset. For future projects, I plan to adopt Trello or a similar Kanban tool, which will give me a clearer overview of priorities, task duration, and deadlines, making the project more manageable and improving workflow.

5.2 Workflow Issues with Team Members

While Marcel was passionate about platform design, having created levels in the 'TeeWorlds' game as a child, he lacked experience with source control, game engines, and coding. I underestimated how challenging it would be to onboard him to the technical aspects of the project. Although I took time to teach him the engine basics and how to use source control via GitHub Desktop (while I used git CLI), I struggled to guide him through the GUI, as I wasn't familiar with it myself. This led to inefficiencies, as I repeatedly instructed him through screen share, slowing down our progress.

In retrospect, I should have familiarised myself with the GUI version of GitHub Desktop before trying to teach it. I also expected that once he learned the basics, he would continue to develop his skills independently. However, this wasn't the case, and I should have limited his tasks to what he was most comfortable with – working within the TileMap node and placing blocks inside the level. In future collaborations with similarly inexperienced team members, I'll focus on streamlining the onboarding process by teaching them only what they need to know and simplifying their tasks, ensuring they can contribute effectively without overwhelming them with unnecessary complexity.

5.3 Lack of Planning for System Architecture

Throughout the project, I often implemented systems without considering how they would interact with one another. To make these systems work together, I had to create junction classes, which complicated the development process. This was particularly problematic with my manager scripts, such as SceneManager.cs and GameManager.cs. I didn't give enough thought to the overall system architecture, which made it difficult later on when multiple scripts needing to exchange data.

One major issue I encountered was the difference between how the game engine version handled data and how the exported version did. The engine would return errors in the game engine environment that didn't appear in the exported version, which was confusing and led to me rewriting parts of the system just to fix these bugs. In future projects, I will dedicate more time, potentially a few days, to thoroughly plan out the system architecture, especially how scripts will communicate and handle data. I will also research programming patterns to ensure smoother integration and data flow across different components, as many of my issues stemmed from poor initial planning.

5.4 Bespoke Debug Logger

During the later stages of development, I encountered several bugs that were difficult to reproduce, leaving me unsure of their causes. Debugging was further complicated because I was using the C# version of the Godot engine, which isn't the primary version. Stepping through the code using the basic debugger often involved clicking through extensive engine code before reaching the sections I had written. This process was time-consuming and frustrating.

To improve this, I initially used the Output console to display log messages whenever certain events, such as execution, instantiation, or deletion, occurred. However, this became inefficient. I would delete these debug print statements after thinking the bug was resolved, only to realise I needed to reinsert them later. While working on other .NET projects, such as Blazor, I learned about debugging levels and logging

priorities—such as trace, debug, information, warning, error, and critical—which inspired me to develop a bespoke logger for future projects.

My idea was to create a static class called Log, which would store the namespaces of each class and its corresponding debug level. This way, I could log detailed information, such as Log.Trace("Player has been moved due to 'MovePlayerDuringGameStart()' function, self"), and tailor the level of detail to the specific debug needs of each script. This system would allow me to toggle the verbosity of logs without needing to delete or rewrite print statements, saving time and making debugging much more efficient. Although it would take longer to set up initially, this logger would be flexible, allowing me to easily apply it to individual files as needed and carry it over to future Godot and C# projects.

6 Summary

6.1 What I learnt and will use next time

- Using partial classes and extension methods to keep large classes more manageable. They will allow me to break up complex classes into smaller files, making the code easier to maintain.
- Adopting Trello for project management will help me visually organise tasks, set priorities, and track progress more effectively. This structured approach will replace my previous reliance on notepad files, ensuring a clearer workflow and better task management.
- Rather than only planning the immediate next step, I will focus on anticipating at least three steps ahead for system design in future projects. This will provide a more strategic overview, allowing me to better foresee potential challenges and streamline the development process over time.
- When on-boarding inexperienced team members, I will initially spend time planning how to effectively present information to them and simplify their tasks. This will be followed by a discussion on gradually increasing their responsibilities as they become more familiar with the project.
- In future projects, I will utilise a bespoke debug logger to track and manage errors more efficiently. This custom tool will allow me to set different debugging levels, making it easier to identify and resolve issues without repeatedly adding and removing print statements.

7 Conclusion

I started off playing around with the Twitch API then slowly creating a prototype which then developed into a bespoke software that was designed for a small community. I learned a lot of things about creating a project such as how I could efficiently improve my work flow which I always try to improve. The main aspect that I am proud of is that I continued this project to the end and I delivered a project that I was proud of. There is potential for me to continue working on the game design aspect of the project and make it an enjoyable single and multiplayer experience that I can ship off to the world. There are a lot of things that I could improve and refactor but I learnt a lot of lessons that I will take away when I do my next projects, but I am happy that this is my first project I put my heart and soul into and hopefully the momentum of this project will encourage me to make more.