Postmortem

Brandon Marshall - Technical Developer, Systems Architect

The GOOD

Overall team morale remained relatively high throughout development, which encouraged collaboration, communication, and problem-solving. Although not every team member was initially enthusiastic about creating a side-scrolling game, considerable effort was made to transform the project into something the team could be proud to showcase in a professional portfolio.

From a technical perspective, several gameplay systems were successfully developed and integrated into the project. These included player weapons, projectiles, pickups, player shields, autosave functionality, destructible objects, and fracture systems. Significant effort was also invested in designing supporting systems that connected these features. One example was the integration between the health system and the ECG interface, allowing player health changes to be reflected visually through both percentage values and the ECG display. This provided clearer player feedback while helping reinforce the game's visual identity.

Another successful implementation was the pickup and destructible framework. These systems were designed with flexibility in mind, allowing pickups to either spawn randomly or be placed manually by the level design team. The master destructible system also enabled designers to override default settings and create customized interactions without requiring additional programming support. This improved workflow efficiency while reducing technical bottlenecks.

The project also provided valuable experience in Unreal Engine systems integration. Multiple gameplay systems were required to communicate with one another, including health, UI, VFX, pickups, projectiles, and destruction systems. Successfully connecting these systems helped strengthen understanding of gameplay architecture and reinforced the importance of scalable design practices.

One of the most significant areas of growth was version control. The project provided extensive hands-on experience using Git and Git Bash within a collaborative development environment. Managing branches, resolving merge conflicts, maintaining repository organization, and adapting workflows based on project needs contributed greatly to expanding practical knowledge of version control practices commonly used in the game industry.

The BAD

While the role of Technical Director provided valuable leadership opportunities, several areas of improvement became apparent throughout development.

One notable issue was the implementation of certain gameplay systems, particularly within BP_Player. While the blueprint functioned correctly, it accumulated a large amount of logic and responsibility over the course of development. Several systems could have been separated into components, interfaces, or dedicated blueprints to improve modularity and maintainability. Breaking these systems into smaller, more focused modules would have reduced complexity and made future development easier. Even separating major systems into dedicated event graphs or actor components would have improved organization and scalability.

Repository management also presented challenges throughout development. While merge conflicts are a normal part of collaborative software development, the frequency of conflicts indicated underlying workflow issues. This highlighted weaknesses in repository structure and branching strategy that were not fully identified early enough in production. Recognizing these issues led to discussions with an industry professional regarding repository organization and version control best practices. As a result, the team transitioned toward a single project structure supported by feature branches, significantly improving repository stability.

Project prioritization occasionally became inconsistent. In some cases, priorities shifted due to communication breakdowns, differing opinions on feature importance, or assumptions regarding expertise. While these situations were generally resolved, they occasionally slowed development and created unnecessary friction. Establishing clearer prioritization processes and defining project goals more explicitly at the start of development would help mitigate these issues in future projects.

The team's organizational structure also created some challenges. Although decisions were often made democratically, the existence of a formal hierarchy occasionally

introduced communication barriers and perceived authority gaps. Future projects may benefit from a more collaborative structure that emphasizes specialization and expertise rather than rigid role-based authority. Encouraging open discussion and constructive feedback regardless of role would likely improve decision-making and team cohesion.

Communication was another area where improvements could be made. Text-based communication can often be interpreted differently than intended, leading to misunderstandings despite positive intentions. Increasing the frequency of face-to-face discussions, voice calls, or team meetings would help reduce ambiguity and improve clarity. Direct communication often resolves issues far more effectively than extended text conversations.

Finally, greater consideration should be given to team members with relevant prior experience. Future projects would benefit from actively identifying areas of expertise within the team and consulting those individuals when making decisions related to their specialties. Creating opportunities for experienced team members to share knowledge can improve decision quality while strengthening overall team performance.

The UGLY

One of the most significant issues uncovered during development involved project structure and version control workflow.

Initially, separate project folders were created for individual programmers, each containing a side-scroller variant where features could be developed independently before being migrated into the primary project. While the approach appeared practical in theory, it introduced substantial technical complications. Because each project variant contained assets, folders, and systems with identical names and structures, migration frequently resulted in asset replacement conflicts and increased integration complexity. Rather than simplifying development, the approach ultimately created additional maintenance overhead and increased the risk of introducing errors during feature integration.

The branching strategy also contributed to repository management issues. Branches were organized around individual programmers rather than specific features or tasks. As a result, branches often remained active for extended periods, accumulating large numbers of changes before being merged back into the main development branch. This led to stale branches, larger merge conflicts, and increased difficulty integrating completed work.

Following discussions with an industry professional, it became clear that both the project structure and branching strategy could be significantly improved. A feature-branch

workflow combined with a single shared project would have reduced integration issues, encouraged smaller and more frequent merges, and minimized the risk associated with long-lived branches. This lesson became one of the most important technical takeaways from the project, as it demonstrated how repository organization and workflow decisions can directly impact development efficiency and team productivity.

Key Lessons Learned

The project provided several valuable lessons that extend beyond programming and into broader software development and team management practices.

Build for scalability from the beginning.

Many systems functioned correctly but became increasingly difficult to maintain as additional features were added. Designing systems with modularity in mind from the start using components, interfaces, and clearly defined responsibilities can significantly reduce technical debt later in development.

Version control workflows directly impact productivity.

The project demonstrated that repository structure and branching strategies are just as important as the code itself. Long-lived developer branches and multiple project variants increased merge complexity and integration challenges. Feature branches, smaller commits, and more frequent merges would have reduced conflicts and improved overall workflow efficiency.

Integrate systems early and often.

Many gameplay features depended on multiple systems communicating with one another, including UI, health, VFX, pickups, weapons, and save systems. Regular integration and testing helped identify issues earlier and prevented larger problems from emerging later in development.

Communication should prioritize clarity over convenience.

Text-based communication can easily lead to misunderstandings because tone and intent are often left open to interpretation. More frequent voice calls, meetings, and face-to-face discussions would have reduced ambiguity and improved team alignment throughout development.

Technical decisions should be informed by relevant experience.

Projects benefit when team members with prior experience in a specific area are consulted before major decisions are made. Leveraging existing knowledge can prevent avoidable mistakes, reduce development time, and improve the quality of final solutions.

Technical debt accumulates quickly.

Short-term solutions are sometimes necessary, but temporary implementations often remain in the project longer than intended. Regular reviews of systems and architecture can help identify areas that require refactoring before they become larger maintenance challenges.

Documentation reduces dependency on individuals.

Several workflows, systems, and technical decisions relied heavily on verbal communication or individual knowledge. Maintaining clear documentation for systems, repository workflows, and development processes would improve onboarding, reduce confusion, and help ensure project continuity.

Cross-discipline collaboration strengthens development.

Some of the project's strongest outcomes resulted from collaboration between programming, level design, UI, VFX, and other disciplines. Creating systems that empower

non-programmers to implement content independently improved development efficiency and allowed team members to contribute more effectively within their areas of expertise.

Leadership involves adaptation as much as direction.

The Technical Director role highlighted that leadership extends beyond technical implementation. Identifying workflow problems, seeking outside industry feedback, adapting processes, and supporting team members proved just as important as solving technical challenges.

Success is measured by growth as well as the final product.

While not every system, workflow, or decision was optimal, the project provided significant growth in Unreal Engine development, gameplay programming, system integration, team collaboration, project management, and version control practices. The lessons learned throughout development will provide a stronger foundation for future projects and professional work within the game industry.