Team Clown Town Postmortem

The final deliverable for the course is for each team to do a postmortem of the team over the entire semester. This document should be filled out as a team and each member should contribute something. Below content from The Game Production Handbook Chapter 17: Postmortems.

The primary purpose of a postmortem is to learn what methods worked and what did not work during game development. These items are usually focused more on the production process—such as scheduling, planning, and implementing features—instead of on the actual design features of the game. Although successes and failures in the game are discussed in postmortems, they are usually tied to something that was done correctly or incorrectly in the production process.

Postmortems tend to be overlooked in the development process for a variety of reasons, such as: there is not enough time; it is not a high priority; or people are not interested in improving the process. (After all, the game was finished, right?) However, postmortems are a vital way to learn from mistakes and validate new ideas that improve the process. By not doing one, developers are overlooking the greatest source of concrete information they have on making things more efficient, less costly, and better on future projects.

In order to extract this knowledge from the team, postmortems should focus on answering these questions:

Did we achieve the game you set out to make in the proposal?

The purpose of this question is not to highlight where the team fell short of the goal, but rather to evaluate why the goal was not achieved, such as changing scope, shifting priorities, or limited time. Solutions to these impediments can be examined and implemented to prevent this in the next game.

Bounce:

We completed all our explicit design goals, but we wish we had the time to polish the core mechanic of jumping more and to add more levels to flesh out the game. We were still learning to balance working on a longer term project with other classes with more frequent deadlines.

Tower Wizard:

We had nearly everything done except for the upgrading towers mechanic and variable game speed setting. We planned and implemented many features for this game. However, where we wanted to implement features we skipped in the first sprint, this made the scope for

the second larger than we were able to handle during finals week. We had a little more enthusiasm during this second sprint, which is why we still got so much done, but didn't achieve all of our goals due to overscoping.

What went right? What went wrong?

This is an opportunity for the team to relay personal experiences regarding what worked and did not work on the project. By discussing both the positive and the negative, the team can carry over expertise to other projects, including which procedures to implement and which ones to avoid. Additionally, if a procedure did not work, solutions can be discussed for ways to make something different, which has the same desired results, work on the next project.

What went right?

We vibed. We're cool with each other. None of us outright hate each other nor had to go behind each other's backs to solve problems. There were mercifully few source control issues. There were no major, "oops we wiped/broke everything" moments. There were many somewhat annoying merges where hierarchy connections in scenes got broken between merges and had to be reconnected, but those never took very long. Both of our games benefited from a mutually understood identity. There were no memorable clashes over design decisions.

Collaborating effectively took a bit of trial and error but we did find a good "rhythm" for it. Having more discrete and defined tasks available for people to do was a big factor in collaborating better, although we weren't very consistent with that.

What went wrong?

Since other classes had more frequent and faster paced deadlines than the deliverables in this class, our work on these sprints was hampered by the intuition that we could just keep putting off these sprints until the week of. While this worked mostly, our games undoubtably could have been improved by more frequent meetings. COVID distancing also made meeting a little more difficult, but each of us working from our own home computer would have been the way we worked regardless of COVID. Of course, the varying degrees to which a modern super plague killing over 500,000 people in our country could also have affected us, if any of our close friends or family were affected, but none of us had this happen to us. In the future, finding more ways to connect with other members and to be transparent with our progress would help bring the team and game together more.

What are the lessons you learned?

The team examines all the information gathered from the previous questions asked and determines the core lessons learned during development. These lessons should focus more on big picture items, and less on small details. The details can be used as methods for implementing the lesson learned on the next project. For example, "communicate deadlines clearly to the team" might be one lesson learned, whereas the methods for communicating

these deadlines (email, status reports, and weekly meetings) provide the details on how to accomplish this. The regular postmortems featured in Game Developer Magazine provide some great examples of lessons learned.

Something we all took away from this was experience working on team projects. We have more “muscle memory” when it comes to working on group software/video game projects. The steps of getting into the repo, opening it up locally, working on it, dealing with merge conflicts takes less energy and thought the more you do it. Additionally, having to seek out groupmates in the beginning to work on a project was a very realistic situation. It did well to look into others and compare strengths and weaknesses, as well as synergy. We can be more confident when it comes to building teams and starting projects together.

In terms of the development cycle, we’ve learned more in the importance of iterating on and improving the core gameplay loop (as opposed to just fixing bugs and/or implementing new features). In the game Bounce, we touched back on the feel and how information was given to the player in the game. It shows that just adding content to a game is just as important as having a solid core.

We learned that we should have a single spot in which we gather our updates and progress on the game. We originally tried Trello and writing a thorough design document, but both these things were abandoned pretty quickly. Due to the fairly small scope of both our projects, it didn’t *feel* very necessary, even when it would have helped direct things better. The first game, Bounce, was held up at one or two points because certain groundwork parts weren’t finished first. All in all, this is an area we should improve on for future projects.

One recurring lesson that we taught frequently by our professor was understanding our limits, and implementing achievable goals within those limits. This is done through communicating with the team the workload and availability each member has, and then planning our work based off of that. Despite none of us being a team manager, we learned what one should be.