**Placeholder Productions Individual Post Mortems**

Kenneth Probeck

Placeholder Productions

Level Maker / Tool Builder / Interface

“Rocket Jump,” as a whole, has taught us of Placeholder Productions many things dealing with creating a game from the bottom up.  On my part, we accomplished many goals, such as creating 3 levels for the player to challenge, which include four to five panes each.  The interface of our game was also a success, both in being user-friendly and working correctly.  Another aspect that turned out in our team’s favor was the Map tool, which allowed for us, and for possible users, to create their own levels for “Rocket Jump” with ease.

However, not all of our aspects did work out.  One part of our game that had to be left out was having boss battles.  I felt that a game such as this could have definitely had great potential for boss battles, however, this part of the game had to be cut because of time constraints.  We also did not have time to add the support of a gamepad, which would have been a nice aspect to add to our game.  Lastly, a very unfortunate part of the game that I would have enjoyed to create is the animation of enemies.  Our main character, Axel, was able to be animated, however, the enemies for our game were unable to have the same fate.

From this project, I have learned to be much more diligent with my time, rather than pushing certain goals off until a certain time.  I feel that the milestones definitely influenced this, and for the future, if given a project without set milestone dates and times, I feel that I should create my own schedule to stay on track and to not fall behind on anything within my project.  I have also learned how to work well in groups, which is something I did not have too much experience with beforehand.

Some aspects of the project that I would have handled differently is how much I worked in the coding of the game and how aspects interacted with each other.  I feel that I should have been more up-to-date and helpful with my group with the game functions, however, I did enjoy creating most of the interface for our game.  Another aspect I would have done differently is how our characters in our game, being Axel and the enemies, turned out.  I would have enjoyed to look more into pixel art and created more complex character art, however, I feel that time constraints and my lack of an artistic background resulted in our art that we currently have in the game.

**Matthew Turczmanovicz**

**Placeholder Productions**

**Character and Object Design/Implementation**

Creating a game was a very interesting experience for me, and (I’d assume) for the entire group. My main tasks for the game was to create and work with Axel, enemies, and other object implementations. Axel works perfectly, and along with Ken’s animations and Simon’s rocket jump game mechanic, we have created an incredibly fun character to play as. The enemies are pretty decent as well, but nothing extraordinary. However, they stalk the player and attack him, which I am happy with. My biggest task however, would become working on the game’s collision detection.

Quite surprisingly, collision detection took us a very long time to complete. Honestly, it took way longer than I believe it should have. I suppose that I take a game’s collision detection for granted because I surely did not expect for it to take as long as it did. From first few milestones we had accomplished firing a rocket and having it collide with a wall and explode. However, working with Axel jumping and flying around proved to be my most difficult task. In the end, I had a very rudimentary system of collision detection working. However, Robert had come to my aid and started working on a very complex collision detection system; one that would ultimately work with Simon’s help.

Unfortunately we did not get to have animations outside of Axel function, and we could not get that would-have-been awesome boss fight up and running. In hindsight we can look at all the goals that we wish we had completed, but I would prefer to look at the things we did get down. I worked on creating the first few classes and methods that would eventually grow into bigger and better things. The enemies are pretty basic, but they get the job done. I brought in the various pieces of content that would be in the game, and I created the ultra helpful GameVariables class. Finally, Axel works and collision detection works pretty dang well for the most part. I am pleased with the work we have done. However, I definitely wish that my coding skills were better than they are, because I felt terrible that I was unable to properly complete collision detection.

I learned that game development is certainly not the easiest thing, but then again, I did not expect it to be easy either. I did learn that I definitely should not underestimate the challenges that I face. Case in point, I certainly did not think collision detection would take as long as it did, and I definitely did not think two other group members were going to have to help me with it. I also learned the values of commenting code. Unfortunately, some methods were not commented that well and I had no idea what was happening sometimes. This definitely hurt my ability to help out with some existing methods. Furthermore, I learned that milestones are very important when it comes to projects like this. Projects need to be split up over time, and groups need to have clear, well-defined goals for each milestone. This kind of strategy will help with any project.

As far as what I would differently; I certainly wished that I expected collision detection to be a challenge. With that in mind, I would have started thinking of solutions for it earlier, and I would have started working on it earlier. I think we would have changed our systems of rectangles and other objects as well, since relying on the rectangle’s Intersect method is not the best idea. I did not really think of collision detection when I created the Axel, enemy, and terrain rectangles. I thought the Intersect method would work fine, but Robert’s final collision detection method goes way beyond simply checking if two rectangles are intersecting. Personally, I wish I had done some work with the game during Spring Break, as another week of working on the game would have done wonders. Maybe I could have added that boss fight. Overall, I am proud of the game, and what we, as a team, managed to create in the past few weeks. It’s been one of my most unique experiences in college thus far.

**Name: Robert Bailey**

**Team: Placeholder Productions**

**Role: Leader/Coder**

Game Idea:

The goal was to create a 2D platformer, with at least two enemies, that was three levels long (each level being four to five screens in length).  The player would navigate the levels using normal left, right, and jump movements.  However, the player would also be able to rocket jump, similar to *Quake* or *Team Fortress 2* to reach higher points on the map to progress.

What went right?

The game is in two dimensions, has two enemies, and has three levels (two are five screens long, one is four).  There is a graphical map editor that can be used, and which generates map files for use in the game.  The player can navigate the levels using the rocket jump in addition to standard movement controls.  The algorithms behind collision detection and rocket jumping work, the game does not crash, and enemies move on set platforms.

The team did not fight throughout the production of the game, and there were no issues getting people to work on their tasks.  The game was completed with a day to spare, with an animated main character, the enemy types looking different beyond a recolor, and (Despite the group name) no placeholder assets in the game.

We had complete control over how the game’s development progressed, but had our status checked with milestones.  We met the first and second milestone on time, as well as the fourth.  We were free to come up with our own algorithms and levels, without needing to appear like any other game.   Despite having to start from scratch, we created a playable game in the allowed time.

On the paperwork side, the team stayed on schedule, with everyone submitting write ups, presentations, and code when needed.

What went wrong?

Luckily, we were able to avoid group fighting and had quality communication.  However, we still encountered errors with the group working.  This mainly came from learning how to use an online based version control tool (Bitbucket with the SourceTree interface) at the same time as the game progressed.  I managed most of the merging and cleaning conflicts until the last two weeks, which did lead to other groupmates being confused as to how Bitbucket and SourceTree worked.  Merging conflicts became the norm as people added and commented in the same classes as others.  Our main class, Game1, became particularly bloated as production continued.  Adding assets became a nightmare, as it lead to a host of merge conflicts down to the FileAbsoluteText and other underlying elements of VisualStudio and Monogame.  These type of conflicts had to be resolved manually, or the offending code would have to be rewritten after the main branch got updated, essentially forcing that coder to redo their work.  These types of delays slowed production of the game, and created bottlenecks, specifically around collision detection and map generation functions.

These bottlenecks were also not purely the fault of merge conflicts.  When picking tasks, I had a habit of assuming a task would be faster than it ended up being.  While the map maker took me a week to finish, the map reader took closer to three.  A similar problem happened with collision detection near the end of the project, as testing the game had to wait until it was playable.  Collision detection was expected to take two weeks at max, and ended up taking over a month.  This was particularly prevalent with Matt and I - we always managed to volunteer for deceptively difficult algorithmic functions, and stalled production.

Those types of delays caused the game to fall behind the original schedule.  Milestone 1 was completed a week before the due date, as our schedule stated.  Milestone 2 was completed on time, when we originally planned it to be a week ahead.  Milestone 3 didn’t meet the original goal, as the lack of collision detection meant no level could be finished.  We recovered near the end, but the delays did cause the team to hold additional meetings to make up for lost time.

In addition, the project started about a week late, due to the team learning to work with Bitbucket and SourceTree.  No member had extensive experience with version control software, so the learning experience was required.  However, this had the upside of no code getting permanently lost or erased during production, even if there were resubmissions required.

Enemies also do not feature animation.  This is due to them having entirely robotic exoskeletons, without joints because their scientists, not programmers, ran out of time.

Overall, the game suffered from the team underestimating the difficulty of creating core game functions, and having to learn new software during the production period.

**Conclusion**

What we learned

The entire team ended up learning how to use Bitbucket and SourceTree, as well as learned how to fix merge conflicts by the end of the production period.  This will undoubtedly be beneficial moving to the future, and help avoid the initial problem of learning new software while developing a game.

We also learned how to code in a team environment, and how to divide a workload.  We learned how to read code we didn’t write, and how to analyze bugs on the fly.  We learned how to test games, and how to log bugs for future fixes.

I learned how to do collision detection and analyzation, how to manage a group and team, and how to how to turn text into a graphical, changing image.  I learned a lot about what I can do in any given week, and how to better predict the time investment an individual feature or algorithm would need.  I’d never really worked well in groups, and preferred to do everything myself, but I learned how to work with others and trust other people to finish their parts.

What I would do differently

I would schedule a weekly meeting that was longer than ninety minutes.  Our meeting time was 2-3:30 on Wednesdays, which included setup time.  This meant that we only worked out of class for 90 minutes as a team.  While this worked for milestone 1 and 2, as the game progressed we didn’t have the time during meetings to address the problems.  This required meetings on Saturday that would last until the problems were fixed (two to five hours), which wasn’t ideal.  If I did it over, I would find a time for two hour long weekly meetings.

I think we adapted well to the changing schedule, so that wasn’t an issue.  I would, however, adjust the expectations for each milestone.  I think each member should have had an algorithm (reading maps in, collision detection, rocket jump, and enemy behavior), and completed that before moving on to other aspects.  Initially, Kenny and I developed the map tool and then started work on the game, which meant that we only had Simon on collision detection and Matt on enemy behavior until Milestone 2 had started.  I think this slowed the development timing, and caused unneeded stress.  Instead, if the algorithms were all worked on from the beginning, production could have gone at a more natural pace, as opposed to members jumping between algorithms and classes getting edited by multiple people at once.

In addition, I would want to have a more clear plan of what happened when a member finished their task.  As it stood, people would just wait when they finished a task, and there was no clear progression.  Having a procedure for continuing work would have helped avoid that issue.

I did like having just one code manager, but I think everyone should have learned about Bitbucket and SourceTree features around the same time, so that no one member had to teach others how to use the tool.  This would also help every member as we move to separate groups.

Simon Voorhees

Placeholder Productions

I think a lot of things went right for the team. We were able to meet consistently and made all of our main goals. The actual gameplay took a bit of work, but it worked well in the end, and I had time to help with collision detection and other features. Our team worked well together, and we divided labor well.

The things that went wrong are more about the way we timed things, our priorities, and communication. Collision detection should have been one of the first things we worked on instead of the map tool, and we should have assigned it one person from the get go. Two people worked on the map tool and reader which was finished pretty quickly. If we started with a clear plan for how we were going to do collision detection it would have been easier. Two people were essentially racing to see who could do collision detection first. They didn’t do this because they were competitive, but we needed it done before we could finish the game, and we were running out of time. I made a mistake early on when I set up the architecture. Something wasn’t communicated between Matt and I, so there was some redundancy at first. Matt made some objects in classes that were supposed to be passed down from parent classes. We should have communicated about what we were doing earlier. I thought it was clear that we were going to add all movement to a vector2 and then add that to the position rectangle, so you could easily stop movement. Matt thought that we were just adding to the rectangle. This caused us to waste time later on because we were fixing problems we wouldn’t have had if we were on the same page.

I think this was a very helpful project. I needed experience working with a team. I learned to clearly agree on and explain systems. I also learned to work on parts of the game based on how important they are to the game. If I was going to do it over again I would work on collision detection once we had a moving player and a basic map, and I would sit down with the whole team and have a discussion over the specifics of the game systems so everyone would be on the same page.