Cynthia Li
Daniel Lee
Jorge Aparicio

Unit 2 Write Up

For Unit 2, our group managed to complete the two 2-d games with a total of 7 components for the first game and 4 for the second game. While our original intention was to balance the components between the two games, we found it much easier to focus our energy and resources onto the first game, and have a bit more fun for our second game. For our first game, we decided to recreate "Flappy Bird" with a more personal twist and thus, "Flappy Pigeon" was born. The first component we hit was the four different graphical building blocks, where we used filled and non-filled rectangles to create obstacles, textures to create the pigeon and backgrounds, and finally text to display scores and other various menu items (1). We have separate title, play, and game over screens, each with different inputs, fulfilling the menu/modal UI component of the unit (2). Focusing on the pigeon specifically, we implemented the sprite and paper doll animation with atlases component, where the pigeon flapping its wings counts as the sprite, and the paper doll animation as the object in the pigeons mouth changing each time the user starts a new game (3). As for game feel, the pigeon falls realistically due to gravity, and it's vertical movement feels similar to that of the original Flappy Bird's (4). Looking at the background and foreground, our pigeon remains in the same x position while the background tilemap continuously scrolls to provide the effect that the pigeon is moving (5). In conjunction with this component, we implemented the infinite tile map and PCG component. While we did not adhere to the traditional definition of an infinite tile map, we essentially added to the background and foreground obstacles as the player continues playing the game. In order to accomplish this, similarly to a sprite sheet, our code reads pieces of a texture and adds them to the game world. From this we procedurally generate our background, where the buildings, clouds, and obstacles randomly switch when added to the screen (6). As for our final component, we added music. To emulate a pigeon in NYC, we added dim city noises to the background, pigeon cooing sounds when the game starts, and pigeon flapping sounds during the actual gameplay (7).

For our second game, we decided to stray from our game 1 concept, and produce a more visual-novel style game. Named "Finding Nemo - The Afterstory", game 2 focuses on presenting the potential sequel to Pixar's "Finding Nemo". All 4 components implemented in this game overlap with "Flappy Pigeon". The first component we re-used were graphical building blocks, as we implemented filled and unfilled rectangles to create textboxes, filled the textboxes with text, and included textures in developing the background as well as sprites for characters (1). The second component was game feel, which we argue exists when we consider the core of the game. By attempting to mimic a visual novel, we focused more on the story line (albeit this is a game engine class) and upgraded what we learned from our unit 1 adventure games. The use of background and sprites to create a 2-d environment, we'd argue, enhances the dialogue aspect of the game, which is a foundational component of content-based games (2). The third component was menus, turn taking, and modal UI. Separate title, play, and end game screens, as well as pickable dialogue options all contribute to this component (3). The final component we used again was music (4). In order to create an underwater vibe, we just added simple music playing in the background while the user runs through the game. Originally, we had wanted to play different music for different scenes, but decided it wasn't necessary to hit the component for the game, and plan on implementing it in the future if desirable.

Playtesting the game allowed us to really see what others as players would want to play and how we as creators could realistically make certain changes possible. Playtesting also helped us to find errors and bugs and various approaches to how a user who is completely unfamiliar with our game would go about the game. For example, in our “Flappy Pigeon”, if the user simply does not flap at all, the score can just infinitely increase without ever having to run into an obstacle. As players of other games, we realized that a set of instructions at the beginning of the screen is extremely important, as without them, it is not intuitive to guess respective controls for specific games. We also were able to have a visual representation of more cleverly implemented components as well as live reactions to our created games, both of which are great forms of feedback for our unit 3 games. Ultimately, seeing others play our game and enjoy them was not only rewarding, but also enlightening in terms of future changes we could make to our projects.

There were an abundance of major takeaways from completing this unit. From start to finish, our team definitely had amazing teamwork. By planning out the work week-by-week, our group didn’t really have any time crunch issues. We always had a game plan of what needed to be done, met an average of 3-4 hours a week to work on the unit synchronously, and worked asynchronously on tasks we couldn’t finish together. Daniel worked hard to contribute to the coding, but Jorge and Cynthia definitely pulled the reins in figuring out a lot of the coding kinks and leading the team to a very structured game engine. Each members’ responsibilities were clearly defined, and overall, we learned a lot from each other in different aspects of the unit. The above average coordination, communication between each other, and attendance of each member created a harmonious working environment, one that any of us wouldn’t hesitate to work in again. The only minor setback we could have improved on was meeting more than once a week. Needless to say, given each of our busy schedules, we did make most of the time we did have with each other. In terms of the actual unit itself, we ended up focusing heavily on our first game, “Flappy Pigeon”, completing a total of 7 components for the game. This was not even intentional, but rather due to having a planned structure for a game, which naturally led to the completion of multiple components. Having our first game be game engine focused in turn allowed us to take a more content-based approach for our second game, allowing us to explore more fun ideas in the 2d-game world. Working on a visual-novel-esque theme for our second game truly highlighted the difficulties of incorporating elaborate content with actual code, a really eye opening point in terms of how complex the games we play are. Reusing the engine code from the first game was difficult, as features oriented toward games with collision, physics, etc do not easily identify with a visual-novel-esque game. This was to be expected however, as we wanted to produce two differently styled games. While the coding was time consuming overall (as it should be), dealing with collision and JSON was perhaps our least favorites. We actually didn’t implement a high level of collision (not even restitution), but were reminded of how arduous the task is. The JSON file was just meticulous and way too time consuming, leading us to cut our original script for the second story game in half. The last bit of coding that was tricky to handle was the music portion. We had originally intended to separate music handling into its own rs library file, but ended up putting the code for it in each main file instead of as part of the engine.