Will Domm, John Saluta, Natalie Frank
December 10, 2024
Systems Programming

It’s Just Business: Final Report

Overview

It’s Just Business is a two-dimensional retro video game. We used bevy as a game engine, created game design art via Aseprite, and integrated art into a tilemap using Tiled. In addition to this, we used various crates that allowed for sprite movement, animation, user interaction, tilemap placement, and more. We used device-query (v2.1.0) for user input, rapier2d (v0.22.0) for physics, bevy (v0.14) for game functionality, bevy_ecs_tiled (v0.4) for Tiled integration, bevy_ecs_tilemap (v0.14) for tilemap methods, rand (v0.8.5) for random number generation, bevy_rapier2d (v0.27.0) for physics in bevy, bevy_tweening (v0.11.0) for animated character movement, proc_macro2 (v1.0) for proc_macro states and derive, quote (v1.0), and syn (v1.0). While game development in Rust is still growing and adapting (Bevy released a new version in the middle of our project), bevy is a powerful game engine that supports many different features and functionalities. We modeled our game after other successful games developed with bevy and thus achieved features that advanced our game. bevy’s bundles, textures, and app installation systems made possible a game with multiple sprites, a tilemap, and interactive game mechanics.

Project Description

It’s Just Business follows an old janitor, Cliff, who explores the corporate office after five p.m. The office is structured so that there are two conference rooms, a main cubicle area, a small break room, the boss’s office, a security office, and a closet. Each room is scattered with mess, such as drawing-scribbled papers, tipped-over chairs, clutter, and trash. It is Cliff’s job to clean the office and head home, only to do it again tomorrow.

But tonight, Cliff decides to explore the office. This office is, for a small time, his world. He waters the plants. He rights the chairs. Most importantly, Cliff imagines his life if he were more than a janitor, more than a mere corporate worker, but the boss himself. With the office empty, it is briefly Cliff’s, and Cliff is the boss.

There are clues hidden throughout the office. Find them all and Cliff can advance up the corporate ladder, climbing from janitor to worker to boss, all the way to CEO. With each advancement, the office changes to a new level. Cliff begins in a small corporate office; he must clean the office, and if he finds a critical item, he can leverage this to his advantage. He works for the office-workers; this office is their livelihood, and Cliff is prepared to climb above it. It’s not personal. After all, it’s just business.

Gameplay

The game begins on a title screen; users may select to *Play* or *Exit* the game. After the former selection, users will be landed in the retro office. Cliff begins at the entrance to the office at the top of the screen. Using keyboard keys such as the up, down, left, and right arrow, users can control Cliff's movements around the screen. Users should explore the office. They may enter different rooms, navigate around cubicles, and inspect the office.

Features that are not implemented yet may include Cliff fixing cluttered parts of the room. By navigating to a cluttered part of the room, Cliff will fix the mess and return the furniture or space to cleanliness. After the user as Cliff has finished cleaning all of the messes, he will be free to find the clue that advances him to the next level.

Goals

There were many goals that we accomplished and some stretch goals that we could have accomplished with more time.

We successfully completed a Game Design Document (GDD) that detailed the game's goals, mechanics, and features. We researched successful 2D game engines, set up a github repository, and implemented bevy. We used crates to receive mouse and keyboard input, implemented player movement, and used a physics crate. We created a player sprite, 2D game assets, and a tilemap; with these objects, we set up an app installation that uploads and renders the player. We set up a title screen; we could have added more comments or narration during the game.

We began to add collision detection between the player and objects but did not complete it. We did not add game mechanics like item collection or enemies; these were stretch goals. We designed the structure of these mechanics but did not implement them. We did not add visual or sound effects; if we had more time, we could have cued these at various events, movements, or other game mechanics. We did not add UI for victory or advancement points. We did not create multiple levels or transitions. We could have added these features with more time.

Challenges

We faced a few challenges during this project. Off of the bat, bevy is a constantly changing and adapting game engine. Thus, it goes through frequent version updates. During our project, bevy advanced from v0.14 to v0.15. Version changes make it hard to ensure generalizability across code. Unfortunately, this also means that sample code or online coding forums do not present usable code or functional methods. They provide structures or frameworks that we can emulate, but constantly adapting code to fit new versions can be a challenge.

Once we learned the basics of bevy, we learned how to create a basic game set up. This includes spawning the tilemap, a sprite, and a camera to center on the map. This involved using the Transform method. We initially had an issue with centering the tilemap and sprite as they were oriented to the center of the screen, and we could not deduce how to change the position of the tilemap. After a few days of hard work, we realized that we should center the camera on the map and transform the sprite to be at the top center of the tilemap.

Correct tilemap, camera, and sprite placement set the foundation for dealing with more complex features such as player movement, animation, and collision. We struggled with player movement as our initial goal. It was difficult because we had spawned the sprite, and we did not know how to integrate sprite spawning and using keyboard input to move it. While we were trying to achieve high modularity within our project, this may have been our handicap as creating multiple different files only served to confuse where our error was. Eventually, we gave up on trying to isolate features into different files and worked directly in main.

Discovering tweening was a game changer—*literally*. When we wrote tweening animations into our main code, we had large success with moving our sprite. Initially, the sprite would start at the bottom of the screen, but we found a way to size it correctly and place it at its starting position.

As always, there are struggles with creating a video game and all of its features, especially in a game engine that changes so frequently. Animation and collision detection continue to be challenges that we approach. We have found that documenting our code, separating code into clear methods, and using online resources can be a great way to overcome challenges.