Final Project: Game Design Document

My game was made in JavaScript using the Phaser3 framework, including physics from MatterJS. The game is a platformer with a start, game over, and win menu, a UI scene, and three gameplay levels. To stitch these classes together, I used an event manager, character and enemy controllers, and state machines. To create my levels and sprites, I used Tiled and TexturePacker, and to bundle the game for deployment I used Parcel. The goal of the game is to collect 10 coins to proceed to the next level without losing all your health. Once the player completes all three levels, they win. If a player loses all their health, they will need to restart from the beginning if they want to try again.

First, I will explain my scenes and their transitions, as shown in my scene transition diagram below. The first scene is a start menu that explains the controls and game objectives. Once a player presses the “play” button, the Level1 and UI scenes both load. The UI scene persists until another menu scene, such as Game Over or Win, is reached. If the player goes back to Start the UI scene is loaded again. While playing levels 1-3, the UI scene controls transitions between the levels and menus as it keeps track of the coin collection and health mechanics. Collecting all 10 coins in a level triggers a transition to the next level, and if the player is on level 3 when this happens, the Win scene is triggered. Losing all player health at any time triggers the Game Over scene. Both the Game Over and Win menu scenes allow the player to return to the Start scene with a “play again” button.

Diagram

Description automatically generated

The bulk of play time is spent in the three level scenes. The tilemaps for these levels were created using Tiled which exports a Tiled file as well as JSON data that can be read by Phaser. In Tiled, I used two sprite sheets, one for ground tiles and one for obstacles. Currently the only obstacle is spikes, but this could be easily extended. In Phaser, I load the same tileset image from public/assets that was used in Tiled which allows the drawn tiles to be matched to the level’s Tiled JSON data. This workflow also allows me to load a second layer from Tiled used for obstacles. The obstacle layer is used for two main purposes: first, it contains spawn points for the player, enemies, coins, and health packs; second, it contains the collision box for obstacles (currently only the spikes). Along with the Tiled level assets, I used TexturePacker to create sprite sheets for the player and enemies. Like Tiled, TexturePacker exports JSON data that I use to create the sprite animations.

Interactions take place between the enemies and ground, and the player and the ground, enemies, collectables, and obstacles. All interactions are handled through MatterJS body collisions. The enemy-ground and player-ground interactions simply prevent falling through the floor or walking through terrain tile. The player is also only allowed to jump after colliding with the ground. Originally, the idea that the player could collide with a wall and then jump was a bug; however, I thought it made sense that a monkey could slide down or jump off the wall, so this became a feature. Player-collectable collisions emit an event using the shared event manager to update the UI which keeps track of coin collection and the health display. Player-enemy collisions have two possible outcomes. First, if the player collides with a bee or frog and the player sprite is above the enemy sprite, the player will defeat the enemy which squishes the sprite and destroys it. If the enemy is not above the bee or frog or if the enemy is a saw, the player will be bounced back in the direction the enemy was moving when the collision occurred and will be damaged, again emitting an event for UI.

The player and each enemy have controller scripts whose primary job is to control its state machine, movement, and animations. Animations are created using Phaser3’s local animation creator, reducing the need to flood the project with global animations. Enemies have simple move and idle animations, while the player has idle, move, jump, and die. The enemies originally had death animations, which I added to their sprite sheets; however, the sprite squishing tween that happens when they are stomped made it too hard to see the dead animations so they were removed. I am not an artist, so I was limited in what animations were available. If I could add animations, I would add one for the monkey to play while in contact with a wall to show that they are intentionally wall sliding or wall jumping. The player and enemy state machines closely follow the animations and entity capabilities, including idle, movement, jumping, hitting, stomping, etc. Each sprite starts in the idle state and each controller dictates the possible states, transitions, and transition triggers. All state machines are finite and are never set outside of the controller. As mentioned earlier, the event system is used to omit events when something outside of the controller needs to trigger a state change. Finally, enemy AI is predetermined and is limited to moving back and forth horizontally. I originally implemented a randomized movement mechanic; however, this caused the enemies to fall off elevated platforms which looked unpolished.

I learned a tremendous amount from this project. I prefer strongly typed languages and have always been put off by JavaScript. Lately, I have been learning TypeScript in my free time and this project was my first attempt at programming in TypeScript which was a great experience. I enjoyed the Phaser3 and MatterJS frameworks, and felt they were similar to Unity which I am very familiar with. I may not use Phaser3 again, but I will definitely use some of the other tools such as MatterJS, Parcel, Tiled, and TexturePacker for other projects. Some of the bugs I was able to overcome during development include: MatterJS default sprite colliders were not the appropriate size and manually changing them is not obvious or simple (fixed after a couple hours of hunting documentation and forums), randomized enemy movement feels sloppy to the player (fixed by setting fixed movement), triggering game over on level 2 and restarting to level 1 would cause the UI to send the player to level 3 after they finished level 1 (fixed by adding a reset game event emitted from Game Over and Win to UI), state machine transitions can stack in odd ways (fixed by adding a mutex to state machine setState function and by being very careful about not going back to the Idle state just because it felt like the default thing to do).

This project presented many challenges, some of which I was not able to overcome. The biggest issue is that the Parcel template that I was so excited about originally, and that made development so easy with its built-in development server, did not originally work in a production web server. I could run the game without issue on the development server, and I followed all the instructions to build the production files correctly as far as I could tell; however, neither Tesla nor my home server would host the files properly. After two days of troubleshooting, I finally found that the html file needed a fully qualified location for the JS file and I was able to host my project successfully. I am very new to JavaScript and did not have this issue with my other JS projects in their class so that was an incredibly difficulty bug to fix. The other project issues were relatively minor, and I wish I could animate wall sliding and wall jumping. There is one major gameplay bug that I was not able to address in time. This bug will damage the player if they are falling too fast when stomping on an enemy. The bug triggers any time the player falls from too high down onto an enemy but cannot be triggered by simply jumping on an enemy like normal. I suspect the player-enemy height check is not calculated quickly enough when the collision occurs, so the player controller thinks it is not above the enemy. If I were to extend this project I would fix the height stomp bug, add a wall slide and jump animation, add more obstacles, add more levels, and vary the level layouts.