Ian Seidman
A01028650

Carpeta Procesos

Desde el inicio del bloque hemos visto varios conceptos relacionados con el diseño de videojuegos, al igual que elaborado un proyecto de videojuegos propio. En este documento explicaré los conceptos que hemos visto, cómo los implementamos en nuestro proyecto, y mi participación en el mismo.

Primero, comenzamos viendo cómo elaborar figuras utilizando un documento .obj que delinea los vértices y las caras del objeto. Esto es algo difícil de hacer manualmente, particularmente para objetos más complejos, pero un triángulo y un cubo se pueden elaborar sin mucha complicación utilizando lógica geométrica.

Este concepto lo utilizamos en nuestro proyecto para un coleccionable, ya que es una figura geométrica algo diferente a cualquier otra pero no tan compleja que no se pueda hacer utilizando esta técnica.

De la misma manera aprendimos sobre las transformaciones que se pueden aplicar a un objeto en Unity, incluyendo trasladar, escalar y rotar. Estos conceptos básicos los aplicamos a la animación de un objeto a través del Animator y luego el Animation Timeline, donde puedes organizar las animaciones de varios objetos en una escena para sincronizarlo.

Para nuestro proyecto, las animaciones que se van a usar son extensas, pero yo personalmente elaboré la animación del enemigo con martillo manualmente, donde el enemigo mueve su martillo hacia atrás y luego lo aplasta directamente en frente.

Otro concepto que aprendimos en la clase es la teselación, la técnica que se utiliza para tomar figuras y sus vértices y convertirlas en estructuras más aplicables para el proceso de rendering. Vimos cómo se aplica la teselación utilizando un ejemplo de un triángulo con un script, pero esto se puede aplicar a figuras mucho más complejas.

También aprendimos sobre el modelado de los objetos y rigging utilizando Blender, donde importamos una figura (de una persona) y tuvimos que crear su rig automáticamente, pero arreglar todos los errores que surgieron de la automatización. Luego tuvimos que agrupar los huesos correctamente, finalmente importando a Unity para aplicar las animaciones. También vimos cómo utilizar la herramienta de Mixamo, que ayuda a simplificar el proceso aún más ya que puedes subir el fbx de tu personaje y aplicarle una variedad de animaciones fácilmente.

Nuestro proyecto utiliza este concepto directamente con el personaje principal, que descargamos en línea e importamos a Mixamo para aplicar las animaciones que necesitamos, aunque el modelado manual también planeamos aplicar para algún modelo más básico como un enemigo o una de las pistolas.

Fuera de los conceptos vistos en clase, para el proyecto he tenido la oportunidad de desarrollar varias otras cosas por mi cuenta. Comenzando con la cámara en tercera persona utilizando Cinemachine, tuve que aplicar transformaciones a la cámara para que fuera aceptable con nuestro personaje y el género de juego.

Al igual tuve que desarrollar aspectos de nuestro UI, particularmente el menú de settings y el menú de pausa. Estos los implementé utilizando scripts básicos conectados con los botones y sliders de Unity, visualizando todo con un Canvas. Los settings en particular incluyen la habilidad de cambiar el volumen del juego con un variable universal de Unity, y la capacidad de cambiar la sensibilidad del mouse (conectando de regreso a la cámara de Cinemachine).

También desarrollé los primeros tres enemigos del juego utilizando el NavMeshAgent de Unity en una clase genérica de "Enemy", con modificaciones particulares de los valores para cada enemigo. Los enemigos tienen tres modos básicos, *patrol*, *chase*, y *attack*, lo que determina a que estados entrar y qué hacer en cada uno puede cambiar dependiendo en variables como ViewRange y AttackRange, al igual que utilizando overloaded functions.

El primer enemigo por ejemplo, es de tipo "Roomba" y solamente se mueve alrededor de manera aleatoria, ya que su ViewRange es de 0 y nunca va a encontrar al jugador para salir de modo patrol. El segundo enemigo de tipo "Chaser" persigue al jugador hasta chocar contra él y causarle daño de esa manera. Esto a través de un AttackRange de 0, por lo tanto siempre se mantiene en modalidad de *chase*. Finalmente el enemigo de tipo martillo tiene un AttackRange bajo (aproximadamente 2) y al entrar al rango del jugador llama la animación de ataque donde mueve el martillo hacia atrás y luego rápidamente hacia adelante.

Tuve la oportunidad de desarrollar la interfaz gráfica básica del juego, incluyendo las mecánicas de vida del jugador y la visualización de su vida actual y la cantidad de balas que tiene cada pistola. Esto lo elaboré utilizando un sistema de canvas en Unity con scripts para modificar o leer los valores de vida y balas, representado con sliders y objetos de texto.

Diseñe el minimapa del juego utilizando un elemento de UI en el mismo canvas del interfaz de vida, representado con un RenderTexture de una cámara. Esta cámara se sitúa directamente sobre el jugador y utilizando un script lo sigue en el eje de X y Z. Una esfera morada está situada directamente sobre el jugador (solamente visible para la cámara de minimapa) para representar la ubicación del jugador de manera obvia e inmediata.

Elaboré un pequeño particle system de una explosión modificando los variables de velocidad, figura y rotación de las partículas (utilizando una textura del asset store de explosión) para aplicarlo al jugador y a los enemigos. La explosión se crea en la ubicación del personaje o los enemigos al morir y se destruye al finalizar el tiempo de animación deseado.

Para darle más énfasis a los golpes recibidos y a los enemigos destruidos, implementé la funcionalidad de "Hitstop" o un congelado completo al recibir daño o matar a un enemigo. Esto lo hice con un script que modifica el timescale del juego para detenerlo por completo y después de un tiempo determinado lo vuelve a resumir. Esto es posible utilizando coroutines que hacen un yield a realtime, en lugar de utilizar el tiempo del juego ya que este está congelado.

Para darle aún más poder a los golpes, también implementé un screen shake de la cámara cuando el hitstop se llama. Esto lo apliqué utilizando la funcionalidad de offset de la cámara de Cinemachine, agregando offset de valores aleatorios durante la coroutine, regresando la cámara a su ubicación original (almacenada en un variable) al terminar.