Informe Proyecto Final

Modelamiento de clases

David González Mazo

Departamento de Ingeniería Electrónica y
Telecomunicaciones
Universidad de Antioquia
Medellín
Abril 05 de 2022

Índice

1. Hollow Souls **2**
1.1. Mecánicas principales 2
1.2. Filosofía de diseño de niveles 2
1.3. Objetivos deseados 2

2. Vista general de las clases **3**
2.1. Fundamentales 3
2.2. Florituras 3

3. Descripción detallada de clases **4**
3.1. Clases Abstractas/logicas 4
3.2. Clases físicas/graficas 5

1. Hollow Souls

Hollow Souls sera un metroidvania inspirado en los iconicos juegos Dark Souls y Hollow Knight.
Un matroidvania es un subgénero de videojuego de acción-aventura basado en un concepto de plataformas no lineal. El acceso a parte del mundo se limita por puertas u otros mecanismos que solo se abrirán después que el jugador haya adquirido unos artículos especiales, herramientas, armas o habilidades en el juego. Por el escenario habrá repartidos distintos tipos de enemigos, armas, obstáculos y habilidades con las cuales el jugador podra acceder a lugares del mapa a los que antes no podía llegar por cuenta propio.

1.1. Mecánicas principales

Atacar, saltar y explorar serán los pilares fundamentales del juego. Todo estará pensando y construido desde estas tres mecánicas.
El personaje principal por defecto tendrá equipada una espada con la cual defenderse y para que el juego no se haga monótono existirán varios tipos de enemigos todos con patrones de ataque y velocidades distintas que supondrán un a superar. Las secciones de plataformas serán desafiantes y exigirán cierta pericia en los controles y por ultimo para incentivar la exploración se repartirán por todo el mapa lugares secretos/opcionales(en lo posible).

1.2. Filosofía de diseño de niveles

Los niveles estarán diseñados de tal manera que el juego no se haga muy repetitivo, Por ejemplo jamas habrán dos escenario seguidos que consistan de un pasillo cerrado repleto de enemigos.
Se irán introduciendo de manera progresiva las distintas mecánicas del juego con ello la dificultad ira variando en la medida que el jugador aprende y domina las distintas habilidades adquiridas.
Al inicio del juego también habrá un selector de dificultad que modificara los parámetros de los enemigos, como por ejemplo, aumentar el daño que hacen o aumentar sus barras de vida.
En un principio solo existirán 2 zonas (si da el tiempo agregaría una tercera) que consistirán en varios escenarios interconectados. En lo posible cada zona tendrá un jefe principal.
Los escenarios se cargaran por medio de matrices para hacer el proceso de diseño mas fácil.

1.3. Objetivos deseados

El objetivo principal es hacer que Hollow Souls sea fluido y divertido de jugar pero a la vez desafiante sin ser frustrante.

La duración estimada es de 10-20 minutos y se tratara que las distintas zonas sean lo mas cortas posibles.

2. Vista general de las clases

2.1. Fundamentales

Aquellas clases de las que no se podrá prescindir, son necesarias para el funcionamiento correcto del juego y para cumplir los requisitos mínimos.

* TheBox.
* Game.
* Map.
* Sprites.
* Motion.
* HealthBar.
* Player.
* Enemy.
* Block.
* Saw.
* Pendulum.

2.2. Florituras

Clases que harán de la experiencia de juego mucho mas amena y divertida. Solo se harán dichas clases en caso de que me quede tiempo para incorporarlas, aun que habrán algunas, como por ejemplo Dash, que sin ella, el juego seria demasiado simple y genérico, así que dentro de esta categoría, algunas tendrán mas prioridad que otras.

* Arrow.
* Bow.
* Dash.
* Potion.
* Rope.
* Camera.
* EquipInterface.
* DamageInterface.
* Score.
* InvertGravity.
* DoubleJump.
* StaminaBar.
* Bomb.

3. Descripción detallada de clases

A continuación se hará una descripción de las clases que tienen mayor prioridad.

3.1. Clases Abstractas/logicas

Comprenden todas las clases relacionadas con la lógica del juego y que no tienen una representación física dentro del mismo. Como por ejemplo una clases que contiene todos los elementos que interactúan dentro del juego y que a la vez tiene la capacidad de mostrar escenas en pantalla.

WhiteBox

Sera una interfaz grafica que sirva para probar todos las clases que se vayan creando y que permite cambiar sus distintos parámetros en tiempo de ejecucion.

Propiedades:
* scene: Scene donde estaran todos los elementos a probar.

Funciones e interacciones:
Tendrá diversas funciones y slots para poder ingresar parámetros e iniciar la simulaciones desde la interfaz grafica.

Game

Contiene todo lo relacionado con el juego. Heredara QGraphicsView y sera una clase que se encargue de contener todos los elementos del juego que interactúan y de mostrar las distintas escenas del juego en pantalla. Todas sus propiedades seran publicas, ya que todas las clases necesitaran tener acceso a ellas menos a mapa.

Propiedades:
* Mapa.
* timer.
* Player.
* Blocks.
* Enemies.

Nota: En esta fase de diseño solo veo necesario estos.

Funciones e interacciones:
Su principal interacción va ser con el Map, mostrara los distintos escenarios y pantallas que genere Map.

* show(Map): mostrara la escena o la pantalla correspondiente que estará en mapa.

Map

Se encarga de crear las distintas escenas del juego que luego se mostraran en pantalla, esto incluye el menu principal y los distintos escenario. Esta clase incluira QGraphicsScene.

Propiedades:
* QGraphicsScene *scene: se carga el nivel actual del juego.
* QGraphicsScene *mainMenu: escena donde se cargara el menú principal.
* QGraphicsScene *loading: escena de pantalla de carga, sera una escena con fondo negro. Esto se mostrara hasta que se garantice que se cargo scene correctamente.

Funciones e interacciones:
Entre Game y Map mantendrán el flujo de que escenas mostrar en pantalla. También interactuara con Hada para poder guardar las partidas.
* loadScene(char i): carga la escena etiquetada con i.
* loadMenu(): carga el menú principal.
* loadGame(Hada): carga una partida guardada.
* saveGame(): guarda la partida actual.

3.2. Clases físicas/graficas

Clases que tendrán propiedades graficas y físicas para poder aparecer en la escena y moverse.

Sprite

Clase padre que dará a sus hijos lo necesario para poder aparecer en escena y todos lo relacionado con manejo de sprites. esta clase heredara QObject y QGraphicsPixmapItem.

Propiedades:
* sprite: Sprite principal.
* frame: Frame que se muestra en pantalla.
* width: Ancho en pixeles.
* height: Altura en pixeles.

Funciones e interacciones:
* setSprite(QString name)
* setFrame(typeX, typeY): recorta el frame a mostrar en pantalla
* setSize(w, h): establece el tamaño en pixeles de los frames a recortar.

Motion

Clase padre que heredara la clase Sprite. Le dará sprites y movimiento a sus clases hijas.

Propiedades:
* masa. * r: Vector posición.
* v: Vector velocidad.
* a: Vector aceleración.

Funciones e interacciones:
* move().
* calculateAceleration(): calcula la aceleración dependiendo del tipo de movimiento.

HealthBar

Incluira QGraphicsRectItem y QGraphicsTextItem, consistirá en dos rectángulos, uno rojo y otro gris cuyos tamaños dependerán de la salud actual del personaje.

Propiedades:
* int health: cantidad de vida actual de personaje.
* QGraphicsRectItem bar: rectángulo cuyo ancho dependerá de cuanta salud le quede al personaje.
* QGraphicsTextItem textBar: texto que mostrara el por ciento de vida del personaje, este texto estará superpuesto sobre bar.
* healthMax: maxima cantidad de vida.
* QGraphicsRectItem barLose: rectángulo que dependerá de la vida perdida del personaje.

Funciones e interacciones:

Interactuara directamente con el player o con el enemigo, al momento de recibir daño o curar se actualizara la HealthBar * updateBarHealth: actualiza la barra de vida.
* updateBarLoseHealth: actualiza la barra de vida perdida.
* increase(int cure): Incrementa la vida y actualiza las barras.
* decrease(int damage): decrementa la vida y actualiza las barras.

Player

Heredara Motion. sera manejado por el usuario, tendrá el focus del teclado.

Propiedades:

* bool state: si esta vivo o muerto.
* HealthBar health: sera la barra de vida del personaje.
* Weapon weapon: arma equipada.
* int flechas: cantidad de flechas que tiene actualmente.
* Bool arco: si tiene arco.
* Bool dash: si tiene el dash.
* Bool invertGravity: si tiene invertir gravedad
* Bool inmu: si es inmune al daño o no.
* int potions: cantidad de pociones.

Funciones e interacciones:

Player interactuara con los enemigos, bloques, sierras, péndulos, habilidades, atacar con el arma equipada y disparar un arco. * dieAnimation: animación de muerte.
* cure(Potion): Usa una poción para curarse. esto modifica la barra de vida del personaje.
* attack(Weapon): ataca con el arma equipada.

* damage(Enemy, Block, Saw, Pendulum, arrow): recibe daño que disminuye su barra de vida y le da frames de invulnerabilidad.
* shoot(Bow): usa un arco para disparar una flecha.
* dash(Dash): realiza un dash y le da frames de invulnerabilidad contra ciertos objetos.
* doubleJump(DoubleJump):
* invert(InvertGravity): invierte la gravedad.

Enemy

Heredara la clase motion, sera una clase padre, de la cual sus hijas serán todos los distintos tipos de enemigos que tendrán sus propios patrones de movimiento y de ataque.

Propiedades:
* HealthBar health: la misma que el jugador pero mas pequeña y dicha posición se ira actualizando con la posición del mismo enemigo.
* bool state: si esta vivo o muerto.
* atk: daño que hace al atacar
* Bool inmu: Si es inmune al daño o no.

Funciones e interacciones:
Podra interactuar consigo mismo y con el player.
* dieAnimation: Animación que se activa al perder toda la vida.
* damage(): recibe daño.
* attack(Player): ataca al jugador cuando esta en el rango, los ataques de cada tipo de enemigo serán distintos.

Spike

Heredara la clase Enemy. pinchos inmóviles que le hacen daño a Player al contacto.

Propiedades:
No tienen ninguna propiedad particular.
Funciones e interacciones:
La única interacción sera con Player.

Block

Heredara motion. Clase padre de la cual heredaran los distintos tipos de bloques. Segun el tipo de bloque tendra movimiento.

Propiedades:
No tendrá propiedades aparte de las que hereda.

Funciones e interacciones:
Interactúa de manera pasiva con el jugador, con el enemigo y con las flechas. Destruye las flechas y supone un obstáculo tanto como para el jugador como para el enemigo.
* destroyArrow(arrow): destruye una flecha.

Bow

Heredara la clase Sprites. Servirá para dispara una flecha.

Propiedades:
* pot: Magnitud de un vector velocidad.
* angle: Angulo del vector velocidad inicial de la flecha a disparar.
* atk: daño que hace la flecha que dispara.

Funciones e interacciones:
El arco interactuara con el enemigo y el jugador, ya que ellos son los lo que lo usaran. También interactúa con las flecha, el arco es el que las dispara.
* shoot(arrow): Dispara una flecha.
* shoot(player, enemy): jugador o enemigo que dispara el arco.

Arrow

Heredara la clase motion. La flecha al ser disparada tendrá una velocidad inicial y una posición inicial dadas por el arco.

Propiedades:
* bool state.
* bool type: si es disparada por el enemigo o por el jugador.
* atk daño que hace al impacto.

Funciones e interacciones:
Interactua con Block, con Player y con Enemy.
* destroyAnimation: animación al destruirse contra un bloque.
* damage(personaje, enemigo): hará daño al jugador o enemigo dependiendo quien la disparo.

Potion

Heredara la clase Sprite.

Propiedades:
* pot: cantidad de vida que recupera al usarse.

Funciones e interacciones:
Interactúa con la HealthBar.
* animation(): animación que se activa al usarse.
* cure(healthBar): Cura la vida del Player/HealthBar.

Saw

Hereda de la clase enemigo. Habran cierras inmóviles, móviles y de distintos tamaños. Tendrá un movimiento circular uniforme sobre si misma que dará el efecto de ser una sierra cortante.

Propiedades:
* HitBox: se refinara la hitbox para que sea exactamente la de un circulo.
Tendrá las propiedades necesarias para simular el movimiento circular sobre su propio eje.

Funciones e interacciones:
Al ser hija de enemigo, interactúa con Player y con los Enemy que sean de un tipo diferente a Saw. Todas las funciones necesarias ya están en Enemy.

Pendulum

Hereda de la clase Block y la clase Motion. sera un bloque con movimiento.

Propiedades:
* HitBox: se refinara la hitbox para que sea exactamente la de un circulo.
En motion estará lo necesario para simular dicho movimiento.

Funciones e interacciones:
Mismas interacciones que la de un Block solo que su hitbox sera circular.

Dash

Hereda la clase Sprites. Habilidad del personaje, al usarla este se moverá rápidamente hacia una dirección ignorando los efectos de la gravedad por un breve periodo de tiempo.

Propiedades:

* pot: Aumento de velocidad del personaje al usarla.
* time: Duración del efecto.
* state: Si esta disponible para usarla. Cada x tiempo después de usarse volverá a estado true.

Funciones e interacciones:

* animation(): Animación al usarla.
* use(Player): Uso de dicha habilidad. El efecto terminara después de un breve periodo de tiempo o si choca contra un bloque.