Olga Cid

* **Creación del nivel y el escenario:**

Modo Landscape con las herramientas de Sculpt, Smooth, Noise, Flatten, Erosion y Ramp (para que al caer al agua se pueda volver arriba) para darle forma al escenario con montañas, cordilleras, altiplanos y una zona con agua.

Material múltiple con Layer Blend y herramienta Paint para el pintado del escenario.

* **Creación de los NPC:**

Creación desde cero de los enemigos usando las clases Character y AIController con mesh y animaciones descargadas del Marketplace.

Uso de NavMeshBoundsVolume para la limitación del rango de movimiento de cada enemigo.

Creación de Behaviour Tree y Blackboard de los enemigos para definir su patrón de comportamiento: patrulla por puntos definidos usando la clase TargetPoint,

Composición del Behaviour Tree con Sequences y Selectors, Service de propia creación para detectar al jugador, Decorator que comprueba si el jugador está dentro del radio, tareas creada para buscar la siguiente posición y moverse hacia el jugador, además de las tareas por defecto Move To y Wait. Además, el enemigo resta vida al jugador cada 2 segundos si este está lo suficientemente cerca usando una función de MyGameInstance.

* **Creación del Pawn:**

Creación del Blueprint de clase Character con dos componentes de tipo Camera hijas de la Mesh: una ajustada a la cabeza para la vista en primera persona y otra un poco alejada para la vista en tercera persona. Esta última va unida y es hija de un componente de tipo Spring Arm, que sirve de unión entre la Skeletal Mesh y la cámara.

Blueprint para el control del Pawn que permite el cambio entre ambas vistas, controla los inputs del jugador (desplazamiento, giro y salto)

* **Creación de blueprint: KeyCollect (Llave)**

Se compone de la Static Mesh de una llave y un capsule collision con propiedades de trigger.

Al sobreponerse con el IcaroBP, llama a la función “**AddKeys**” del Game Instance y a la función “**DestroyActor**” para desaparecer.

Luis Giraldo

* **Implementación de una instancia dinámica de material con un parámetro que se modifica en tiempo real:**

Para la planta nocturna hice su modelado, UVmapeo y vertex paint (para el movimiento del viento) en 3DS Max y texturizado simple con Photoshop.

Creé un material cuyo albedo es el que creé en Photoshop, y cuyo emisivo es la máscara emisiva que creé en Photoshop (todo negro, excepto la parte del bulbo, que estaba pintada) y la multipliqué por una constante, que acabó siendo un parámetro para poder modificarlo. Además, al worldPositionOffset le puse el SimpleGrassWind, y en el parámetro del peso del viento le puse el color del vértice, para que no se le aplique a la base de la planta. Los valores de intensidad y velocidad los puse como parámetros también.

Creé una instancia dinámica del material para poder modificar los parámetros. Después, creé el blueprint de la planta nocturna, cuyo único componente es la mesh de la planta. Este blueprint tiene como variable una referencia al material de la planta.

En el level Blueprint, al hacerse de día o de noche, por cada instancia del blueprint de la planta nocturna, se le actualiza el valor del parámetro Brillo según el timeline.

* **Implementación de un UMG con al menos dos variables a representar en tiempo real:**

Para esto, primero creé una Structure que fuese el inventario, con 2 variables de tipo integer, que son las que se representan en pantalla: la salud, de inicio 100, y el número de llaves, de inicio 0. También creé un enumeration con los posibles estados del juego: MainMnu, InGame y PauseMnu.

Por último, creé un Widget Blueprint para InGame, que muestra el nombre y el valor de las 2 variables asociadas.

* **Actualización de la UI: Health / Keys**

Hemos creado una función “Bind” para el texto que indica el número de llaves y otro para el número de salud restante. Ambas funciones llaman constantemente a las funciones “**GetKeys**” y “**GetHealth**” del Game Instance respectivamente.

Ariel Martínez Jacobson

* **Menú principal básico (UMG)**

Primero hemos creado los botones que formarán el menú principal y les hemos añadido una función al ser clicados. El jugador empezará en un punto en concreto del mapa (playerStart). Hasta que el jugador no pulse el botón de ‘Play’, no va a poder moverse ni hacer nada. El botón de Play “libera” al jugador y le da total libertad en el mundo de juego.

El botón de **Load** no funcionará si no se ha usado previamente el botón de Save. Si se ha guardado y el jugador le da al botón de Load, se carga la posición y rotación del personaje, la cantidad de llaves y vida que tenía previamente el personaje. También el estado de la puerta (abierta o cerrada) y no se spawnean las llaves que se han cogido previamente.

Luego pasamos al menú de pausa, que tendrá 3 botones:

El botón de **Main** **menú**, que nos redirigirá al menú principal, parando el juego totalmente y ocultando los botones del menú de pausa mientras enseña los botones del menú principal.

El botón de **Save**, que nos permitirá guardar nuestro progreso en el juego (vida actual del personaje, número de llaves que tiene encima, posición y rotación del mundo del personaje, estado de la puerta) por si queremos salir y retomar desde donde lo dejamos previamente.

El botón de **Resume**, que sencillamente reanudará el juego, ocultando el menú de pausa y dejándonos continuar el juego.

* **Blendspace**

Hemos hecho uso de una variable (**speed**) para hacer la interpolación entre animaciones de los enemigos, teniendo en cuenta sus velocidades actuales.

Estos enemigos interpolarán entre 2 animaciones: **idle**, que la ejecutarán cuando lleguen a uno de los puntos de patrulla y se paren (speed = 0), y **walk**, que la ejecutarán cuando se muevan (speed >= 25).

* **Blueprint de animación**

Creación del Blueprint de Animación de los enemigos con una máquina de estados que incluye 2 estados: Movement y Attack. Miramos en todo momento la distancia entre el NPC y el jugador y la velocidad del NPC.

En el estado de **Movement**, hacemos uso del Blendspace que hemos creado previamente y le pasamos como parámetro la velocidad del NPC. Gracias a esto, tenemos una transición fluida entre las animaciones de Idle y walk.

En el estado de **Attack**, simplemente ejecutamos la animación de ataque. Para llegar y salir de este estado, miramos en las transiciones si la distancia entre NPC y jugador es menor a 5 metros. Si es true, entra en el estado de Attack. Si es false, sale del estado de Attack o no entra.

Eric Reyes Guzmán

* **Creación funciones: Game Instance**

El Game Instance consta de 7 funciones que sirven para conseguir los comportamientos deseados para las variables previamente creadas en la “Structure” SInventory:

* **AddKeys:** Suma el valor del input “NumKeys” al número de llaves totales actualmente en el SInventory.
* **ConsumeKeys:** Comprueba si se dispone del número de llaves estipuladas por el input, de no ser así se realiza un return directamente. De tener un numero de llaves igual o mayor a las estipuladas, se resta ese valor a las llaves totales en el SInventory.
* **ConsumeHealth:** Comprueba si la vida es mayor que 0, después se comprueba si la restante que resultaría de restar el valor del input al actual sería mayor que 0. De ser mayor que 0, simplemente se le resta a la vida actual. Si el resultado fuese menor o igual a cero se iguala la vida a 0 y se llama a la función “**Defeated**” del IcaroBP, la cual llama a las funciones “**ResetKeys**” y “**ResetHealth**” y recarga la escena.
* **GetKeys:** Devuelve el número actual de llaves en el SInventory.
* **GetHealth:** Devuelve el número actual de salud en el SInventory.
* **ResetKeys:** Pone el número de llaves en el SInventory a 0.
* **ResetHealth:** Pone el número de salud en el SInventory a 0.
* **Creación de condición: Victoria y derrota**

Victoria: Al entrar en el Trigger que se sitúa alrededor de la fuente tras la puerta, se pone en “true” la variable booleana Win en el IcaroBP, lo que hace visible la palabra “Victory”, gracias a una función “Bind”, en la UI del jugador.

Derrota: Si la vida del jugador llega a 0, el método “**ConsumeHealth**” de la Game Instance llama a la función “**Defeated**” del IcaroBP, la cual llama a las funciones “**ResetKeys**” y “**ResetHealth**” y recarga la escena.

* **Creación de blueprint: DemonDoor (Puerta)**

Se compone de 2 Static Meshes, una con forma de puerta y otra con forma de altar, con sus respectivos collisions. El de la puerta lo bloquea todo y el del altar es un trigger con un text renderer.

Al comenzar el juego, se buscan todos los actores de tipo KeyCollect y se guarda la cantidad total en una variable que servirá para saber el número de llaves que se necesitan recoger para abrir la puerta.

Al entrar en el trigger del altar, se comprueba el número de llaves de las que se dispone, si no se dispone de suficientes, el altar muestra el número de llaves necesarias con el text renderer, si se dispone de las llaves necesarias, la puerta comenzara a bajar gracias a una TimeLine que modifica su localización Z relativa.