~ P3 ~

Arquitectura - versión 1.0 - *20/02/2019*

Resumen	
Géneros: FPS, circle-strafing, arcade...	**Modos:** 1 jugador
Público objetivo: Todas las edades.	**Requisitos mínimos:** Prácticamente ninguno
Cantidades: - 1 personaje jugable - 1 arma mejorable - Enemigos - 1 mapa	**Hitos:** ***Hito 1*** - **15/03/19 -** ***Hito 2*** - **9/04/19 -** ***Hito 3*** - **17/05/19 -**

Descripción general

Para P3 hemos pensado hacer un juego en primera persona inspirado en Devil Daggers, un circle-strafing arcade. En principio el jugador aparece en una arena circular y enemigos spawnean para matarlo. El objetivo del juego es aguantar la mayor cantidad de tiempo posible.

La dificultad va en aumento a medida que pasa el tiempo, ya sea debido a un número mayor de enemigos o nuevos enemigos más peligrosos. El arma del jugador también se puede ir mejorando.

Autores

- Baratto, Diego DiegoBV dbaratto@ucm.es
- Barea, Juan JuBarea jubarea@ucm.es
- Cañellas, Lluis LluiCS lluiscan@ucm.es
- Mateos, Diego dimateos dimateos@ucm.es
- Rodríguez, Jorge jorgerodrigar jorger09@ucm.es
- Sanz, Gonzalo gonzsa04 gonzsa04@ucm.es

Diseño general

Juego FPS arcade de sobrevivir oleadas

- Jugador: dispara, salta…
- Mapa: interactivo? varia
- Enemigos: varios tipos...

El diseño desarrollado se encuentra en el gdd

Tabla de contenidos

1. Proyectos

1.1. Main

Inicia la aplicación y todo lo necesario, etc.

- La máquina de estados quizás en un proyecto aparte para poder reutilizarla
- Dependencias con todos los demás proyectos (directas o indirectas)

1.2. Entity, componentes y mensajes

Entidades y mensajes y la mayoría de componentes

- Dependencias con todos los demás proyectos menos main

1.3. Render e input

Interfaz sobre ogre y el componente de render

- Dependencias con OGRE 3D
- Quizás en algún momento separar el input

1.4. Física

Interfaz sobre physx y el componente de física

- Dependencias con nvidia Physx

1.5. Config y logs

Permite interactuar con jsons, logs centralizados, acceso a la configuración...

- Dependencias con la librería para JSONs

1.6. Sonido

Manager de sonido

- En principio dependencias con SDL_mixer
- En el futuro cambiarlo por Another FMOD SoundManager

2. Estructura de clases

2.1. Game

Inicia todos los recursos y motores. Inicia la máquina de estados.

2.2. GameState

Clase que contiene una lista de entidades a las que llama para que se actualicen. Una de ellas tendrá asociada un componente de input que gestiona la lógica de ese estado (hace las veces de GameManager). Desde un fichero se le asignan distintos tipos de entidades para dar lugar a varios tipos de estado (pausa, menú, juego, etc).

2.3. GameStateMachine

Apila GameStates y ejecuta la lógica del que esté en la cima. Permite desafiarlos etc.

2.4. Entity

Clase con una lista de componentes, con métodos para añadir y quitar nuevos componentes. Tiene un booleano “active” que determina si deben actualizarse sus componentes o no. No estamos seguros de si tendrá un transform. Desde un fichero se le asignan distintos componentes para dar lugar a varios tipos de entidades (player, enemigo, etc).

2.5. Component

Clase abstracta, padre del resto de componentes (update, render y handleEvents, etc).

2.6. Emitter y listener

Clases de las que heredan otros componentes para ser oyentes y/o remitentes unos de otros. Al crear una entidad se determinará cuáles de sus componentes son oyentes de cuáles.

2.7. Message

Clase base que solo contiene un enum para diferenciar tipos. Tendrá subclases que contengan más información si es necesario.

3. Pipeline de generación de contenido

Todo se configuraría desde JSON:

- Configurar los estados (lista de entidades).
    - Configurar cada entidad (lista de componentes).
        - Configurar cada componente (parámetros, etc).

4. Planificación

4.1. 1er Hito

Desarrollo de la arquitectura / motor de juego: principalmente los proyectos de main y máquina de estados, entidades y render. Y algo de config y logging.

4.2. 2do Hito

Acabar la arquitectura: proyectos de física y enlace con render, también proyecto de sonido.
Desarrollo del juego: enemigos, mapas, componentes, etc.

4.3. 3er Hito

Corrección de errores y pulido de arte. Quizás desarrollo de contenido extra.