Proyecto final: Desarrollo del videojuego – Segunda parte

Objetivo: Desarrollar un videojuego-simulador desarrollado en Unity.

Descripción: El objetivo del proyecto final es desarrollar un videojuego en Unity. Como segunda parte de este proyecto final, se deberá desarrollar el videojuego diseñado en la primera parte, resolver los problemas de diseño que surjan y discutir de manera crítica la solución adoptada en cuanto a la arquitectura de software.

Además de los requisitos de la parte 1, también se deben cumplir los requisitos de desarrollo.

Requisitos de desarrollo:

1. Se debe desarrollar en Unity 2022.3.44f1
2. El proyecto debe ser jugable sin *crashes* para poder ser probado.
3. El proyecto debe poder ser compilado y jugado en otros dispositivos (PC o Mac).
4. Se hará gran hincapié en la calidad y complejidad del código en el proyecto, el uso de programación orientada a objetos, cumplir con los principios SOLID y el uso de patrones de diseño.
5. Documentar todo el proceso de desarrollo, comparando el diseño inicial con el diseño final obtenido y exponer los problemas enfrentados a lo largo del desarrollo argumentando los cambios realizados. Para el diagrama UML final se seguirá la misma filosofía que para el inicial, incluir únicamente el diagrama de clases con los métodos más relevantes (No será necesario incluir todos los atributos ni todos los métodos).
6. Repositorio Github estructurado con commits y ramas donde se pueda trazar el desarrollo realizado.
7. En el videojuego se deben incluir componentes que sean *RigidBody* y que implementen algún mecanismo de físicas. (Ya sea colisiones elásticas, aplicación de fuerzas por ejemplo una fuerza de rotación aplicada a ruedas para impulsar un vehículo, lanzamiento de objetos que describa un tiro parabólico…)

Estos serán los punto a tener en cuenta a la hora de evaluar la practica final, ordenador de mayor a menos importancia.

Entrega del proyecto final:

La entrega del proyecto final se ha de realizar **antes del día 11/01/2025 - 23:59h**

- Archivos con la documentación, enlace del repositorio y diseño UML de la arquitectura del proyecto. Estos archivos deben estar en el directorio /doc
- Carpeta con el proyecto jugable compilado para PC o Mac.
- Carpeta con el proyecto entero de Unity, borrando las siguientes carpetas dentro del proyecto antes de enviarlo
    - Library
    - Temp
    - UserSettings
    - Logs
    - Obj
- Si se ha entrenado a un agente ML, se ha de incluir también en una carpeta:

- Proyecto de entrenamiento si es diferente al original.
- Archivo de configuración .yaml usado para entrenar a dicha IA.

- Un video de la jugabilidad (opcional).

Como realizar un ejecutable
File -> Build Settings -> Te sale la plataforma a la que quieres hacer -> Windows, Mac, Linux. Desde Mac se puede hacer de Windows, entonces ponemos Windows 64 bits.
Arrastramos las escenas que vamos a utilizar a Scenes in Build. Es importante que nos aseguremos que la primera escena que aparece en Scenes in Build, es la primera escena, rollo la de inicio o lo que sea.
Creamos una carpeta que se llame build. Le damos al botón de abajo al lado de build and run y seleccionamos la carpeta que hemos creado y se llama build.
Después le damos a build and run y ya carga todo. Los archivos no podemos abrirlos porque son binarios y solo los entiende la máquina.
Podemos ejecutar el programa, y se nos va a abrir como el programa. Se va a abrir la escena primera. Por eso es importante respetar el orden de la primera escena que metemos en Scenes in build.

Lo del sería sacar el ejecutable y después probarlo en otra máquina, para segurarnos que está bien, y que no haya ninguna dependencia ni nada.

DISTRIBUCIÓN TRABAJO
- movimiento del resto de vehículos
- movimiento de las personas npc
- multiijugador (dejarlo para después)

Isa:
- scene manager
- agente de IA (debe ser oponente del usuario => meter ia a los coches de policía)
- Data manager (controla score y money) (estará suscrito al taxi probablemente)
  -- money manager es el que controla lo del dinero, tiene que contemplar dinero negativo
- meter sonidos
- Control de pasajeros (persona) (que aparezca mensaje en pantalla para que el taxi vaya a recoger a la persona, que esté la persona esperando y cuando se sube al taxi, desaparece del mapa)
  -- definir (trayecto) destino del pasajero
  -- definir propinas (tienen en cuenta tiempo que tarda en llegar al destino, choques con objetos (los obstaculos te quitan vida y velocidad))

Sofía:
- Funcionalidad taxi
  -- controlar los inputs del usuario (InputHandler)

- obstáculos (crear en unity y programarlos)
- Player (hereda de taxi driver) (PLAYER VA A SER EL TAXI)
- police officer (police car) debe poder perseguir al taxi y porder moverse como el resto de vehículos
- police car (aplicar patron de facade)
- radar debe tener rayo laser y medir constantemente (evento: collider lanza un evento y devuelva la velocidad medida de un vehículo que choca con ese collider)
  -- choque con radar te quita dinero que ya tienes, no afecta al viaje