PROYECTO FINAL: DISEÑO DE ARQUITECTURA – PRIMERA PARTE

En esta primera parte del trabajo se nos pide desarrollar un diseño de la arquitectura inicial con diagramas UML para un videojuego-simulador desarrollado en Unity.

Las escenas de usuario son breves descripciones de cómo un usuario o personaje interactúa con un sistema para lograr un objetivo específico.
Las historias de usuario suelen seguir la estructura de "Como [personaje] quiero [tal] para [tal]". De esta forma podemos buscar el perfil para el que desarrollamos cada cosa, cual es la intención del personaje y que es lo que están intentando conseguir, y por qué lo hacen.

Por lo tanto, vamos a realizar las distintas historias de usuario indicando una pequeña descripción de la escena, el proceso que pensamos que vamos a tener que seguir, así como lo que esperamos obtener con esa escena.

Además, por cada escena vamos a incluir una frase siguiendo la estructura mencionada para entenderlo con más claridad.

Posteriormente vamos a identificar el objetivo del juego, los requisitos y las restricciones de nuestra aplicación.

Para el diagrama UML debemos justificar la decisión de la arquitectura, y razonar el uso de los patrones implementados.

ESCENAS DE USUARIO

Escena de Usuario 1: Juego Inicial - Configuración y Gestión de Escenas
Actor: Jugador

1. **Descripción**: Al iniciar el juego, el GameManager carga la primera escena del juego y configura los parámetros iniciales (vidas, puntaje, etc.).

2. **Proceso**:
    - El jugador inicia el juego y se muestra la primera escena (por ejemplo, el menú de inicio).
    - El SceneManager se encarga de cargar la escena del juego seleccionada por el jugador.
    - El DataManager se asegura de que los datos de juego previos (si existen) estén disponibles, como la puntuación y la vida.
    - El SoundManager reproduce música de fondo y efectos sonoros.

3. **Resultado esperado**: El juego comienza en la escena correcta, con la música y efectos correspondientes, y los datos iniciales (como vidas y puntaje) se configuran adecuadamente.

Como jugador, quiero que el juego cargue la escena inicial y los parámetros correctos, para comenzar el juego de la manera adecuada y con la información correcta.

Escena de Usuario 2: Control de un Personaje en el Juego
Actor: Jugador

1. **Descripción**: El jugador controla un personaje (como un ciudadano o taxista) que se mueve en la ciudad.

2. **Proceso**:
    - El jugador usa los controles para mover al personaje en la escena.
    - El PersonController se encarga de interpretar las entradas del jugador (como movimiento y dirección).
    - Dependiendo del tipo de personaje (por ejemplo, Citizen o Taxi Driver), el personaje puede realizar diferentes acciones.

3. **Resultado esperado**: El personaje se mueve de acuerdo con los controles del jugador, y las acciones específicas de ese tipo de personaje están disponibles.

Como jugador, quiero controlar el movimiento de mi personaje dentro de la cuidad, para realizar las acciones del personaje y avanzar en el juego.

Escena de Usuario 3: Llamada de Taxi en la Ciudad
Actor: Ciudadano (controlado por la IA o por el jugador)

1. **Descripción**: Un ciudadano necesita un taxi y utiliza la aplicación de taxi para solicitar un viaje.

2. **Proceso**:
    - El ciudadano abre la TaxiApp para solicitar un taxi.
    - La aplicación asigna un Taxi cercano y el VehicleController controla el movimiento del taxi hacia el cliente.
    - Una vez que el taxi llega, el ciudadano sube al vehículo y se dirige a su destino.
3. **Resultado esperado**: El taxi llega al ciudadano, lo recoge y lo lleva a su destino de acuerdo con las reglas de tráfico y obstáculos presentes en la ciudad.

Como ciudadano, quiero solicitar un taxi mediante una aplicación, para que el taxi me recoja y me lleve a mi destino de manera eficiente.

Escena de Usuario 4: Interacción con la Policía en una Patrulla
Actor: Policía

1. **Descripción**: Un oficial de policía patrulla la ciudad en su vehículo y reacciona a situaciones específicas, como ciudadanos en peligro o vehículos que infringen las normas.

2. **Proceso**:
    - La clase PoliceOfficer usa el VehicleController para manejar el movimiento de su patrulla (un PoliceCar).
    - El policía patrulla siguiendo una ruta determinada y está atento a cualquier Event activado en la ciudad (como un ciudadano en peligro).
    - Si un evento ocurre, el AI Agent notifica al policía, quien interviene según el protocolo.

3. **Resultado esperado**: El policía responde al evento de manera adecuada, aumentando la seguridad de la ciudad y manteniendo el orden.

Como policía quiero patrullar la ciudad y reaccionar a determinadas situaciones, para mantener el orden y aumentar la seguridad.

Escena de Usuario 5: Generación y Manejo de Obstáculos en la Escena
Actor: Juego (controlado por la IA)

1. **Descripción**: Durante el juego, diversos obstáculos aparecen en la escena, como patrullas de policía, vallas o radares de velocidad.

2. **Proceso**:
    - El ObstacleFactory genera diferentes tipos de obstáculos (como PoliceCar, Fence, SpeedRadar) en posiciones aleatorias de la ciudad.
    - Los personajes y vehículos deben esquivar o interactuar con estos obstáculos, y algunos pueden afectar el puntaje o la vida del jugador.

3. **Resultado esperado**: Los obstáculos se generan de manera dinámica, creando desafíos para el jugador y aumentando la dificultad de la partida.

Como jugador, quiero que los obstáculos se generen de manera dinámica, para que el juego ofrezca nuevos desafíos y aumente la dificultad.

Escena de Usuario 6: Aumento del Nivel de Alerta
Actor: Jugador

1. **Descripción**: El jugador realiza una acción que aumenta el nivel de alerta, activando más patrullas y obstáculos.

2. **Proceso**:
    - El jugador realiza una acción sospechosa o infringe las reglas de la ciudad, lo que activa un evento en el AI Agent.

- El GameManager aumenta el nivel de alerta y se activan más patrullas de policía o PoliceCar en la escena.
- El jugador debe adaptarse para evitar ser detectado por los nuevos obstáculos y la policía.

3. **Resultado esperado**: El nivel de alerta aumenta, se incrementa el número de patrullas y obstáculos, y el jugador debe evadir la policía para continuar.

Como jugador, quiero que el nivel de alerta aumente cuando infrinja las reglas, para enfrentar más obstáculos y evadir más policías.

Escena de Usuario 7: Gestión de Puntuación y Vida
Actor: Jugador

1. **Descripción**: Durante el juego, el jugador gana puntos y puede perder vida dependiendo de sus acciones y decisiones.

2. **Proceso**:
   - El jugador completa acciones exitosas (como recoger pasajeros o evadir obstáculos) y acumula puntos mediante el Score en el DataManager.
   - Si el jugador falla, pierde vida a través de la clase Life en el DataManager.
   - Al finalizar el juego, el puntaje total se guarda y el jugador puede ver su puntuación final.

3. **Resultado esperado**: La puntuación y vida se actualizan correctamente, reflejando el rendimiento del jugador durante la partida.

Como usuario, quiero que mi puntuación y vida se actualicen durante el juego, para ver mi progreso y mis valores actuales.

Escena de Usuario 8: Cambio de Estado de Sonido en el Juego
Actor: Jugador

1. **Descripción**: El jugador puede ajustar o desactivar la música y los efectos de sonido en el juego.

2. **Proceso**:
   - El jugador accede al menú de opciones y selecciona los ajustes de sonido.
   - El SoundManager ajusta la música de fondo y los efectos de sonido según las preferencias del jugador.

3. **Resultado esperado**: La música y los efectos de sonido se adaptan a las preferencias del jugador, mejorando la experiencia de juego.

Como jugador, quiero poder ajustar la música y efectos de sonido, para que la experiencia sea mejor según mis preferencias.

OBJETIVO DEL JUEGO, REQUISITOS Y REESTRICCIONES

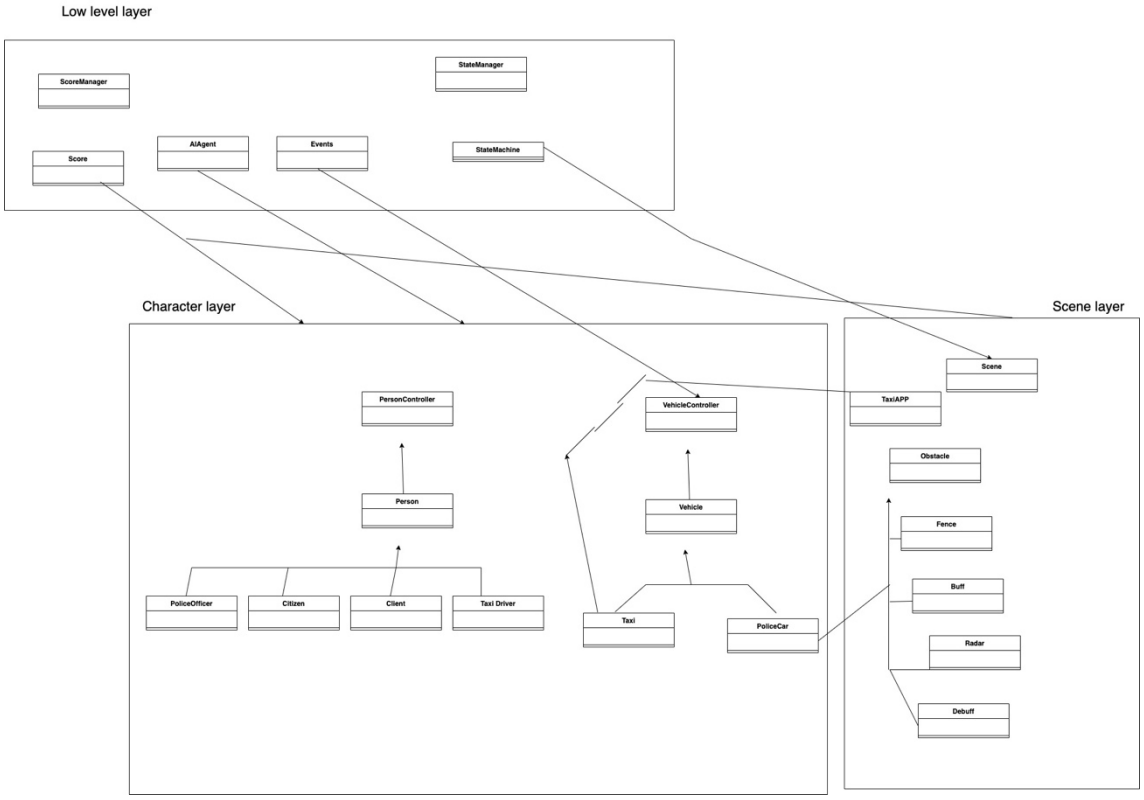
Aunque ya lo hemos definido en las escenas de usuario, vamos a hacer un pequeño resumen de todo lo indicado para tener mayor claridad y poder comprenderlo mejor.

El objetivo principal del juego es conducir un taxi en una ciudad caótica, recogiendo pasajeros y llevándolos a su destino. Durante el trayecto el jugador debe evitar los obstáculos y policías, así como aquellas cosas que reduzcan su vida. Tu objetivo es conseguir el mayor dinero posible con tus viajes, y convertirte en el mejor taxista de la ciudad.

Los requisitos de la simulación son sobre todo funcionales. Son prácticamente lo mismo que hemos mencionado antes, el taxi debe evitar obstáculos, el jugador debe poder controlar movimientos, deben aparecer obstáculos, si infringe alguna norma debe haber consecuencias, debe poder controlar el sonido...
Con respecto a los requisitos no tan relacionados con la funcionalidad podemos destacar que queremos que el juego sea funcional, poder jugar desde distintos lugares, que sea bonito...

Finalmente, con respecto a las restricciones, podemos mencionar la capacidad de vida del taxi, las reglas de tráfico, o las restricciones de espacio de la ciudad.

UML



Esta estructura de diagrama UML está dividida en tres capas principales: low level layer, character layer and scene character. Esta estructura está elegida para facilitar la organización de las responsabilidades en distintos niveles. Esta estructura modular además permite agregar, modificar o quitar funcionalidades en cada capa sin afectar las unas a las otras, por ejemplo, en la capa de personajes podemos añadir nuevas cosas sin tener que modificar la capa de bajo nivel. Además, nos permite la reutilización de código.

Con respecto a los patrones de diseño, tenemos que destacar varias cosas del código.
Tenemos una máquina de estados, que cambia dinámicamente el comportamiento de los agentes en función del estado actual.
También tenemos los controladores PersonController y VehicleController, que manejan las interacciones y acciones de los personajes y de los vehículos.
Al usar una organización en capas, se entiende el uso de un patrón de fachada. Esto permite que las interacciones entre capas sean más controladas y evita que personajes de una capa conozcan detalles específicos de otras.
Tenemos herencia y composición. Esto permite extender las funcionalidades sin tener que duplicar el código, permitiendo una organización jerárquica.