Paradigmas Proyecto Final: Taxi Driver

Hecho por Isabel Morell y Sofía Negueruela

Enlace al repositorio de github: https://github.com/202215473/PyTP_ProyectoFinal

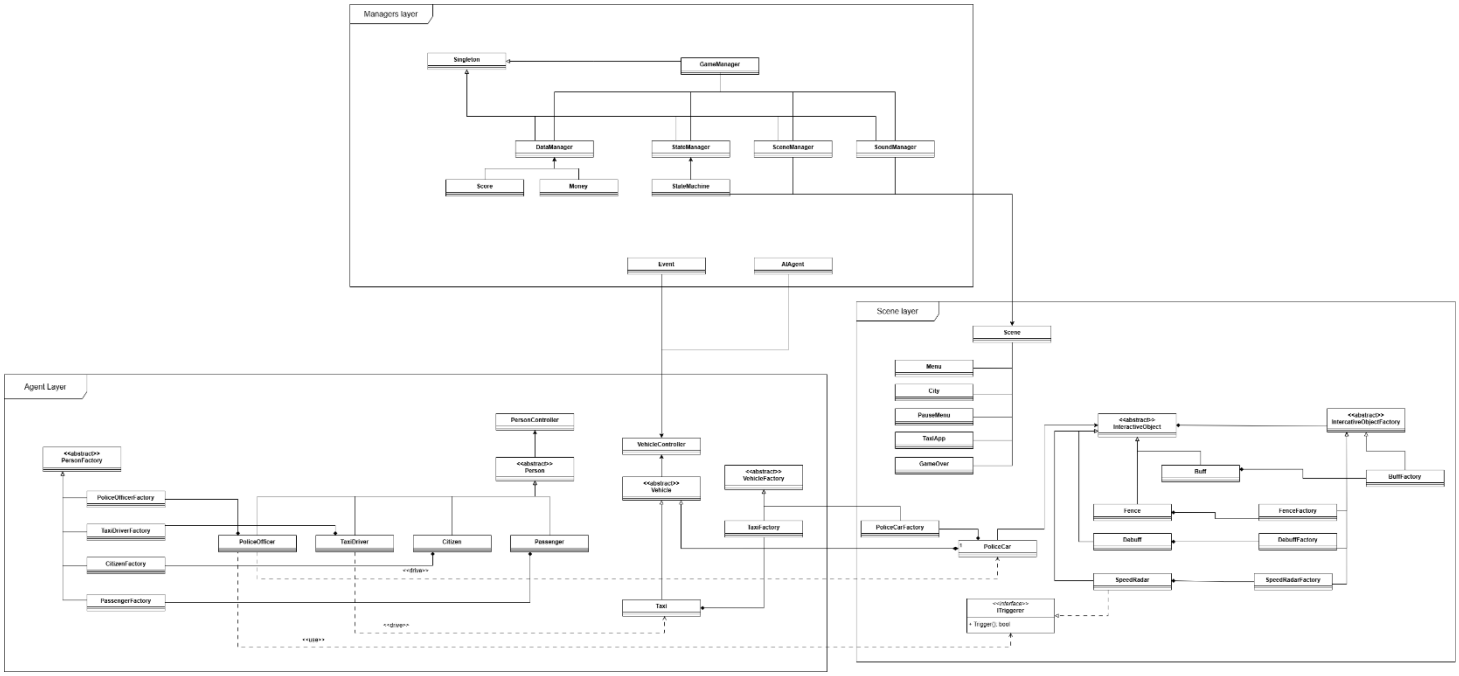
Introducción

Como proyecto final hemos decidido desarrollar el proyecto propuesto sobre el Taxi Driver. El juego va a consistir en un taxista que debe llevar a los pasajeros lo más rápido posible a su destino pero siempre yendo seguros y cómodos. Sin embargo, el taxi se enfrentará a diferentes obstáculos como la persecución de policías, vayas de obras o *debufs* que reducirán su velocidad. De la misma forma, el taxista también contará con *bufs* para recuperar la vida de su coche.

El objetivo del taxista será llevar al pasajero a su destino con la máxima comodidad posible para obtener el máximo dinero y poder comprar diferentes taxis y hacerles mejoras.

Arquitectura del juego

Hemos diseñado el siguiente diagrama UML que muestra la arquitectura que va a seguir el juego.



La arquitectura está dividida en tres capas: la capa de agentes, la capa de escena y la capa de más bajo nivel, la de los managers.

La capa de agentes contiene a las diferentes personas que van a interactuar en el juego. Las acciones de estas estarán controladas por el controlador de personas. Los personajes son los ciudadanos de la ciudad, los pasajeros del taxi, el taxista que será controlado por el jugador y los policías que conducirán los coches de policía. Todos serán creados por factorías siguiendo el patrón de diseño de la factoría con el objetivo de abstraer la creación de las personas de su funcionalidad y para seguir los principios SOLID.
Esta capa también incluirá los taxis que vayan a conducirse. Estos pueden ser uno o varios dependiendo de si se está jugando en modo un jugador o multijugador. Todos los vehículos serán creados por la factoría asociada a ese tipo de vehículo. Hemos elegido el patrón de la factoría para separar la creación de los objetos de su funcionalidad para cumplir el principio de única responsabilidad (Single Responsability Principle) y porque puede que se creen más de un taxi o más de un coche de policía.

La capa de escena incluye las diferentes escenas que se mostrarán durante el juego. Estos son elementos de Unity que no tendrán código asociado.
En esta capa también se incluyen los objetos interactivos del juego que son los objetos que podrán realizar algún tipo de efecto sobre el taxi, interactuando así con él. Cabe mencionar que el coche de policía no hereda de la clase InteractiveObject pero incluye su funcionalidad mediante el patrón *Decorator* y el SpeedRadar será lanzado por el policía.
Los objetos interactivos serán creados siguiendo el patrón de la factoría pues consideramos que así mantenemos la abstracción y seguimos los principios SOLID.

Por último, tenemos la capa de Managers que será la encargada de gestionar los elementos del juego y hacer que funcione correctamente. Para ello tendremos un GameManager que gestionará el resto de Managers y el juego a más bajo nivel. El resto de managers son: el DataManager, que se encargará de gestionar las puntuaciones de vida del taxi, del confort del pasajeros y del dinero del taxi; el StateManager, que controlará la máquina de estados y será el responsable de que una acción resulte en su consecuencia; el SceneManager se encargará del cambio de escenas debido un evento; y, por último, el SoundManager se hará cargo de los sonidos del juego. La división de tareas permitirá la claridad y encapsulación del código y la gestión de tareas por elementos externos al juego permite la abstarcción y encapsulación del juego.
Todos los *managers* heredarán de una clase Singleton pues solo queremos una instancia de cada manager para que no haya varios objetos que puedan realizar las tareas de gestión del juego mencionadas previamente.

Historias de usuario

Para comprender qué debe hacer el código exactamente, hemos emplea la técnica "Historias de usuario". Partimos de las siguientes premisas:

"Como jugador, quiero poder conducir un taxi y llevar a pasajeros rápidamente a su destino para ganar mucho dinero"

El objetivo principal del juego es permitir al jugador (taxista) trasladar a los pasajeros de un punto al otro del mapa (ciudad) lo más rápido posible, haciéndoles sentir seguros y cómodos durante el trayecto y sin perder puntos de vida.

Vamos a descomponer la historia en tareas. Estas engloban el desarrollo del proyecto en distintas fases que ayudan a crearlo de forma ordenada y por partes. Cuando se desarrolla un juego, es importante tener claro las tareas principales, aunque estas se puedan desglosar más adelante.

- Crear una interfaz de usuario para que el jugador pueda conducir el taxi.
- En esa interfaz, añadir elementos de la ciudad (personas, obstáculos...) descritos en la sección anterior, que sean visibles para el taxi.
- Crear funcionalidad de cada uno de dichos elementos y definir cómo interaccionan con el taxi.
- Implementar sistema de puntuación y vida para el jugador, que pueda observar cómo aumenta el dinero que tiene al recibir una propina de un viajero y, además, ver cuántos puntos de vida le quedan.

Los requisitos funcionales del proyecto engloban:

- Un menú principal, que permita iniciar el juego cuando desee el jugador.
- Un sistema de puntuaciones, tanto para la vida del jugador como para el dinero que obtiene.
- Jugadores CPU, es decir, que haya oponentes dentro del juego previamente programados con Inteligencia Artificial.
- El juego debe ofrecer la opción de multijugador en local.

Los no funcionales abarcan:

- Un *loop* de juego cerrado, es decir, que la partida acabe en algún momento (*win state*, *fail state*) y exista la opción de volver a jugar.
- Los componentes se crean y destruyen al inicio y final de cada partida respectivamente.

“Como pasajero, quiero que me lleven a mi destino de forma rápida y segura para llegar cuanto antes”

El objetivo del pasajero es tener un viaje seguro y cómodo, pero también lo más breve posible. Recordamos que en esta ciudad todos van siempre con prisa.

Vamos a descomponer la historia en tareas

- Crear una interfaz que muestre que el pasajero está dentro del taxi.
- Crear funcionalidad de cada uno de dichos elementos y definir cómo interaccionan con el taxi.
- Implementar sistema de satisfacción que sirva para determinar cuánta propina dar al taxista.

Los requisitos funcionales del proyecto engloban:

- El sistema de satisfacción del pasajero.
- Que haya otros ciudadanos que no sean pasajeros.

Los no funcionales abarcan:

- El viaje del pasajero tiene inicio y fin.

“Como policía, quiero que se respeten los límites de velocidad para que la ciudad sea un lugar seguro”

El objetivo del policía es reducir al máximo la posibilidad de accidentes. Para ello, han colocado radares por toda la ciudad, y cada coche de policía tiene un radar también.

Vamos a descomponer la historia en tareas

- Crear una interfaz que muestre los coches de policía y los radares de la ciudad.
- Crear funcionalidad de los radares y de los coches.

Los requisitos funcionales del proyecto engloban:

- El sistema de detenciones. Cuando un vehículo supere el límite de velocidad, será perseguido por la policía hasta ser detenido y llevado a la comisaría.
- Que el coche de policía y los radares seas visibles para otros vehículos.

Los no funcionales abarcan:

- El turno de patrulla del policía tiene inicio y fin.
- El radar solo mide la velocidad cuando está activado (los radares de la ciudad estarán activados siempre, los de los coches de policía estarán activados siempre que el policía esté patrullando.