TRABAJO FINAL PARADIGMAS Y TÉCNICAS DE PROGRAMACIÓN

TAXI DRIVER

Blanca López-Jorrín Pérez

Victoria Guillén de la Torre

Victoria Guillén de la Torre
Blanca López-Jorrín Pérez

ÍNDICE

Victoria Guillén de la Torre
Blanca López-Jorrín Pérez

1. INTRODUCCIÓN

Trabajo final para la asignatura.
ENLACE: https://github.com/victoriaguillen/Taxi-Driver

2. ENVIRONMENT

La interacción con el entorno y la gestión de las carreteras fue un requisito fundamental en el desarrollo del juego. Para llevar a cabo esta gestión de manera eficiente y estructurada, se diseñaron dos clases principales: RoadTile y RoadObject.

- **RoadTile**: Esta clase define todo lo relacionado con un segmento individual de la carretera. Sus principales funciones son:
    - Generación de una posición aleatoria en la acera del segmento, utilizada para la colocación de pasajeros.
    - Almacenamiento de información sobre segmentos vecinos, esencial para la navegación en el entorno.
- **RoadObject**: Esta clase gestiona el conjunto completo de RoadTile y soporta la navegación del taxi. Sus funciones principales son:
    - Agregar vecinos a cada RoadTile, identificando los segmentos contiguos que representan posibles rutas para el taxi.
    - Determinar la mejor ruta entre dos puntos (por ejemplo, entre la posición actual del taxi y el punto de recogida de un pasajero). Esto se logra mediante un algoritmo de búsqueda basado en un enfoque BFS (Breadth-First Search).
    - Proveer dos métodos auxiliares clave:
        - GetRoadTileAtPosition(position): Devuelve el RoadTile correspondiente a una posición específica.
        - GetRandomTile(): Selecciona un RoadTile aleatorio del conjunto existente. Este método se utiliza para generar tanto obstáculos como pasajeros.

En conjunto, estas clases establecen una estructura sólida para gestionar la navegación y la interacción en el entorno del juego.

3. TAXI

El objeto taxi podemos desglosarlo en sus componentes internos (estado del taxi o los pasajeros que este lleva), su interacción con el entorno (mecanismo de detección de pasajeros, inicio y fin de la ruta), la interacción con otros componentes (control de velocidad y el control de su vida).

TaxiPrefab

3.1 Estado y Componentes Internos

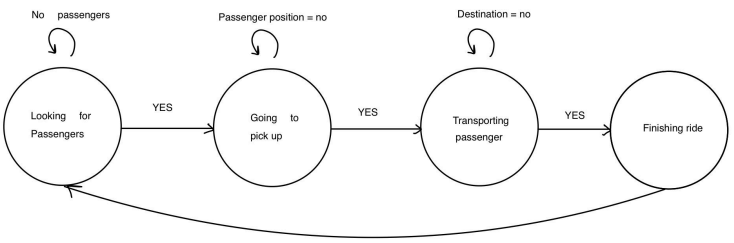
- **Rutas y pasajeros**

El taxi puede buscar y seguir rutas hacia pasajeros cercanos (FindClosestPassenger), transportarlos a sus destinos y gestionar la finalización del viaje.

Utiliza un sistema de tiles (RoadTile) para calcular y seguir rutas, asegurando que el taxi se desplace de manera lógica por el mapa.

- **Estado del taxi**

El flujo de control del taxi se gestiona utilizando un patrón de máquina de estados. Este enfoque es especialmente útil dado que existen múltiples condiciones que determinan el estado del taxi en cada momento. Al implementar un sistema de estados, conseguimos que el código sea más estructurado, intuitivo y fácil de mantener. Gracias a esta organización, el taxi puede cambiar de estado de manera secuencial, desde la búsqueda de pasajeros hasta la finalización del viaje, permitiendo una transición clara y controlada entre las distintas fases del comportamiento del taxi.



Máquina de estados del taxi

1. **LookingForPassanger**: En este estado, el taxi busca el pasajero más cercano. Utiliza la distancia euclidiana para determinar cuál es el pasajero más cercano en el mapa. El taxi sigue buscando hasta encontrar uno que esté dentro de un rango de distancia determinado (el umbral de proximidad). Pasa al siguiente estado cuando encuentra en el entorno de juego un pasajero al que recoger.
2. **GoingToPickUp**: El taxi comienza a desplazarse hacia la posición del pasajero que ha sido identificado en el paso anterior. En este estado, se calcula la ruta más eficiente para llegar hasta el pasajero utilizando un enfoque BFS y, durante el trayecto, se actualiza la posición del taxi en el mapa. Cuando el taxi llega a la posición del pasajero y lo recoge, se pasa al siguiente estado.
3. **TransportingPassenger**: El taxi transporta el pasajero hacia su destino. Cuando el taxi llega a la posición de destino del pasajero. Se pasará al siguiente estado cuando el taxi llegue a la posición de destino del pasajero .
4. **FinishingRide**: Este estado se activa cuando el taxi ha llegado al destino del pasajero y sirve para reiniciar la situación. Se desactiva el pasajero, se le paga al jugador, y se le muestra un mensaje de confirmación del éxito del viaje. Una vez que las acciones de finalización del viaje se han completado, se vuelve al estado inicial y se vuelve a llevar a cabo el proceso completo.

3.2 Interacción con el Entorno

- **Sistema de Navegación y Localización de Pasajeros**: Con el objetivo de facilitar la localización de los pasajeros, se añadió un componente visual de efectos behavioural denominado "halo" en cada RoadTile. Este halo se activa para indicarle al Taxi el camino a seguir para llegar al pasajero más cercano. Además, para mejorar aún más la visibilidad, se implementó el mismo componente en los pasajeros, lo que facilita su localización dentro del espacio del juego.
- **Detección de Pasajeros**: El taxi localiza al pasajero más cercano utilizando la distancia euclidiana, siempre que el pasajero esté activo.
- **Inicio de Viaje**: Cuando el taxi encuentra un pasajero, se genera una ruta desde su posición actual hasta el destino del pasajero. Durante el viaje, el pasajero se desactiva visualmente.
- **Finalización del Viaje**: Al llegar al destino, el pasajero se reactiva en su nueva posición, y el taxi recibe un pago por parte del banco.

3.3. Integración con Otros Componentes

- **Control de Velocidad**:

El taxi utiliza un controlador de velocidad (SpeedControler) que permite modificar su velocidad temporalmente debido a colisiones con los obstáculos. El taxi cuenta con un componente que actualiza su velocidad en tiempo real (UpdateTaxiSpeed), mostrando este valor al jugador en la interfaz.

- **Vida del Taxi**:

A través de la clase TaxiLifeController, se gestiona un sistema de salud que disminuye en caso de colisiones. La salud se refleja en una barra de progreso (Slider)..

4. POLICE CAR

El desarrollo del coche de policía utiliza IA con NavMeshAgent, eventos personalizados y corrutinas para simular persecuciones dinámicas, gestionar multas y coordinarse con el banco del juego de manera eficiente.

PoliceCar prefab

4.1 Funcionamiento general

El coche de policía hereda de la clase Vehicle, que a su vez deriva de MonoBehaviour. En la clase se han implementado las siguientes funcionalidades clave:

- **Detección de violaciones de velocidad:**

La clase está suscrita al evento OnSpeedViolationDetected, el cual se dispara desde un SpeedManager cuando se detecta una infracción de velocidad en un taxi.

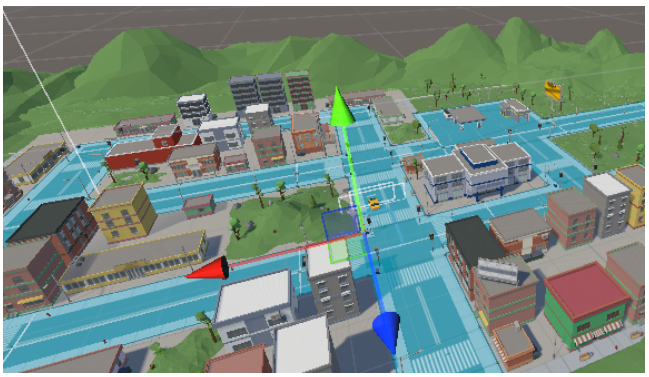
Cuando esto ocurre, el coche de policía inicia la persecución del taxi infractor estableciendo su posición como destino a través de un NavMeshAgent.

- **Sistema de persecución:**

Durante la persecución, el coche ajusta dinámicamente su destino para seguir al taxi en caso de que este se mueva. También rota suavemente hacia la dirección del objetivo, simulando un movimiento más realista gracias al uso de quaternions.

Si la distancia entre el coche y el taxi es menor a un umbral definido, la persecución se detiene.

Para realizar esta persecución, como dijimos anteriormente utiliza un NavMeshAgent. Para que esto funcionara de manera correcta, tuvimos que crear una leyer a parte de 'Default', llamada 'Road'. A esta layer pertenecen todos aquellos objetos que son children of 'Road'. De esta forma definimos las posiciones válidas sobre las que el coche de policía puede moverse.



NavMesh Surface

- **Eventos personalizados y penalización:**

Se utiliza un evento de tipo Action<Taxi> llamado OnTaxiCaught para notificar a otros componentes del juego cuando el coche de policía alcanza al taxi. Esto permite una separación clara de responsabilidades entre los sistemas de interacción (como el banco) y la lógica del coche de policía.

4.2 Restauración del estado inicial:

Tras detener la persecución, se emplea una corrutina (IEnumerator) para esperar un tiempo definido antes de que el coche vuelva a su posición inicial. Este enfoque asíncrono permite evitar bloqueos en el flujo del juego y mejorar la experiencia visual.

4.3 Integración con el juego

En el GameManager, se manejan las acciones asociadas a la captura del taxi. Si el taxi tiene fondos suficientes, se deduce el monto correspondiente como multa a través del sistema de banco Bank. Si no, el juego termina. Este sistema permite gestionar el flujo económico y las consecuencias del comportamiento de los taxis.

5. RADAR

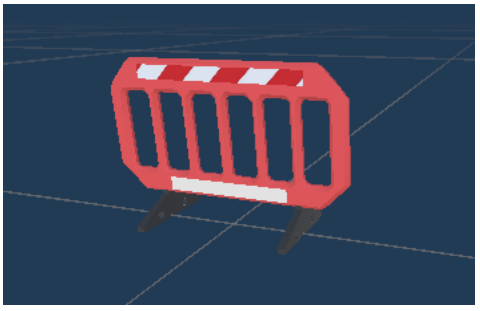
La clase Radar es responsable de detectar taxis dentro de un área definida y verificar si exceden el límite de velocidad permitido. Utiliza un radio de detección configurable para identificar taxis cercanos y calcula su velocidad en tiempo real a partir de su componente Rigidbody. Si un taxi supera el límite de velocidad establecido, la clase dispara un evento personalizado, OnSpeedViolationDetected, que notifica a otros componentes, como el PoliceCar, para que inicien acciones de respuesta, como una persecución. Esto permite integrar un sistema dinámico de monitoreo y penalización en el entorno de juego.

RadarPrefab

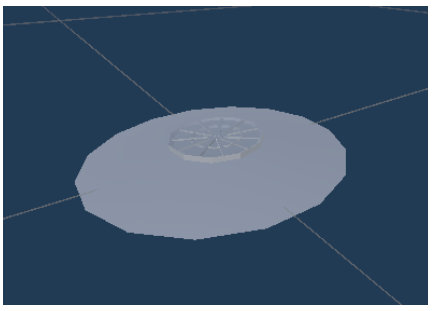
Hemos decidido colocar tres radares estratégicamente en distintas zonas del mapa. Inicialmente, el jugador desconoce sus ubicaciones, lo que lo obliga a conducir con precaución. Sin embargo, con el tiempo y la experiencia, el jugador podrá identificar sus posiciones y limitar su velocidad únicamente en esos tramos específicos. Este enfoque busca reflejar la realidad, ya que es un comportamiento común.

6. OBSTACLES

El juego cuneta con dos obstáculos, que afectan de forma diferente al taxi. El fenceObstacle y el WeakenerObstacle. Podrímos considerar obstáculos también al resto de los elementos de la ciudad, pues el taxi al chocar contra ellos pierde vida. Sin embargo, estos tienen dicha clase asociada.



FenceObstaclePrefab



WeakenerObstacle Prefab

6. 1 Tipos de Obstáculos:

Los obstáculos son creados como objetos de tipo Obstacle (Weakener) o su subclase SolidObstacle (Fence).

El Obstacle tiene características como typeOfObstacle (tipo del obstáculo), negLivePoints (puntos de vida negativos que puede afectar al taxi), multiplyFactor (un factor que puede modificar el comportamiento del taxi), secAcffectedHighSpeed (tiempo en segundos que afecta al taxi si está a alta velocidad), y priceToDestroy (el precio que cuesta destruir el obstáculo).

6.2 Generación de Obstáculos:

Los obstáculos se generan en posiciones válidas dentro del mapa (RoadObject).

Cada vez que se cumple el intervalo de tiempo (definido por spawnInterval), un nuevo obstáculo elegido aleatoriamente entre los dos disponibles se genera aleatoriamente en una de las posiciones disponibles. Este intervalo de tiempo va cambiando a lo largo que avanza el juego, para así incrementar su dificultad.

Empleamos el patrón Factory para centralizar y simplificar la creación de objetos de tipo Obstacle. Al emplear este patrón, se encapsula la lógica de creación de objetos, lo que permite generar obstáculos de manera flexible y reutilizable, independientemente de su tipo específico.

6.3 Interacción con el Taxi:

El obstáculo puede tener un impacto directo sobre el taxi, dependiendo de su tipo y propiedades.

- Un Weakener reduce la velocidad del taxi durante un tiempo determinado (definido por secAcffectedHighSpeed) y afectarlo con el multiplyFactor. Este tipo de obstáculos debilitaría al taxi.
- Una Fence tambien reduce la velocidad de la misma forma que el anterior, pero además reduce la vida del taxi tras la colisión y dificulta el paso de este.

6.4 Destrucción de Obstáculos:

Los obstáculos pueden ser destruidos por el jugador si tiene suficiente dinero en el banco, representado por el Bank.

El jugador interactúa con un obstáculo haciendo clic sobre él, lo cual invoca el evento OnObstacleClicked. Este evento verifica si el jugador tiene suficiente saldo para destruir el obstáculo. Si tiene el dinero, el obstáculo es destruido y eliminado del juego. Si no tiene suficiente dinero, se muestra un mensaje informando al jugador.

Cuando un obstáculo es destruido, también se libera la posición del RoadTile, permitiendo que otro obstáculo pueda ocupar ese espacio, pues como es lógico dos obstáculos no pueden estar en la misma posición.

7. PASSENGERS

El sistema de gestión de pasajeros en el juego está compuesto por dos clases principales: PassengerPool y Passenger. Estas clases trabajan juntas para generar, manejar y controlar el comportamiento de los pasajeros que serán transportados por los taxis. A continuación, se explica el funcionamiento de cada una de estas clases y cómo interactúan entre sí.

Por un lado, tenemos la clase *Passenger*, que define tanto el comportamiento como las características de cada pasajero individual dentro del juego. Esta clase permite abstraer toda la lógica y los atributos relacionados con los pasajeros del resto del sistema, facilitando su gestión y ofreciendo un acceso directo a sus parámetros de manera sencilla y ordenada.

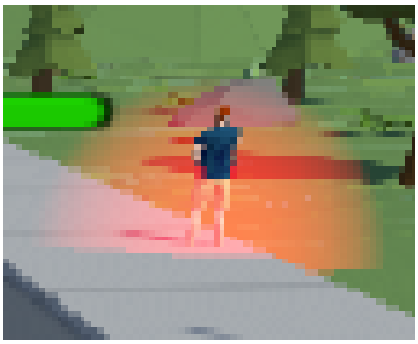
Por otro lado, estaría la clase PassengerPool, que se encarga de la generación aleatoria de pasajeros y se apoya en la clase RoadObject para lograrlo. Algunas de las consideraciones clave que se tuvieron en cuenta durante su implementación son:

1. **Número limitado de pasajeros**: No puede haber más de un tipo de pasajero activo al mismo tiempo en el tablero.
2. **Generación aleatoria de posiciones**: Tanto las posiciones de origen como de destino de los pasajeros son generadas aleatoriamente entre las posiciones posibles en el mapa.
3. **Generación ininterrumpida**: Los pasajeros se generan de forma continua, cumpliendo con las restricciones de tiempo definidas por el *spawnTimer* y asegurando que se respeten las limitaciones anteriores.

Durante el diseño de la lógica de generación aleatoria de posiciones para los pasajeros, nos enfrentamos a varios desafíos. El primero de ellos fue garantizar que las posiciones generadas fueran accesibles desde la carretera, ya que, de no ser así, el juego perdería sentido. Para ello, la clase RoadObject jugó un papel crucial al definir las "tiles" válidas donde los pasajeros podían aparecer.

Sin embargo, este enfoque no fue suficiente. También era necesario asegurarnos de que los pasajeros no se generaran en cualquier parte de una "tile", sino específicamente en las aceras, para que tuvieran un comportamiento realista dentro del entorno urbano. Para esto, optamos por utilizar BoxColliders en cada prefab de carretera y activando el Trigger, ya que su funcionalidad era únicamente definir rangos. De esta manera, nos aseguramos de que los pasajeros solo pudieran aparecer sobre la acera dejando libre el asfalto para la circulación de vehículos.

Finalmente, otro reto que tuvimos que considerar fue la regeneración de los pasajeros después de que hubieran sido transportados. Una vez que un pasajero es dejado en su destino, es esencial reiniciar su estado para que pueda ser reutilizado correctamente en futuras generaciones, asegurando la continuidad y eficiencia del sistema.



Passenger Esperando a ser Recogido

8. BANK

La actividad monetaria del Taxi se gestiona a través de la clase Bank. Esta clase se encarga tanto de los depósitos, que ocurren cuando el Taxi transporta correctamente a un pasajero a su destino, como de las retiradas, que se producen cuando el jugador paga una multa o decide utilizar el dinero del banco para destruir un obstáculo.

Para mantener al usuario informado sobre su saldo en todo momento, la clase Bank utiliza un componente TextMeshProUGUI, que actualiza dinámicamente la interfaz de usuario con el balance actual del jugador.

9. GAME CONTROLLER

El Game Controller es el componente principal encargado de gestionar la lógica y el flujo general del juego, coordinando sistemas como la generación de obstáculos, el manejo de eventos globales y el estado del juego .Actúa como un intermediario que conecta diferentes managers, supervisa las condiciones de victoria o derrota, y asegura que todo funcione en conjunto para ofrecer una experiencia de juego coherente.

10. CAMERAS

El juego cuenta con dos cámaras:

10.1 Main Camera

Esta es la cámara principal del juego, cuenta con un CameraController el cual se encarga de hacer que la cámara siempre siga al taxi y que esta mire en su dirección.

10.2 MiniMap Camera

Esta segunda cámara tiene como objetivo representar un mapa de la ciudad para que el jugador pueda ver que está sucediendo. Se colocó estática a gran altura y rotada hacia abajo. Su vista se colocó en un pequeño rectángulo abajo a la derecha, simulando un mapa.

MiniMap Camera

11. NOTICE MANAGER

Por último, se implementó un sistema de notificaciones en pantalla para informar al jugador sobre eventos relevantes del juego. Este sistema se basa en el script *NoticeManager*, que utiliza la interfaz *INoticeManager* para garantizar una arquitectura flexible y extensible. Para facilitar la comunicación entre los diferentes componentes del juego, se creó el evento NoticeEvents.OnNotice, al cual *NoticeManager* está suscrito. Esto permite que otros sistemas envíen mensajes de notificación al jugador de manera centralizada y ordenada. Los mensajes tienen una duración limitada configurada por el usuario, tras la cual desaparecen automáticamente de la pantalla para evitar saturación visual.