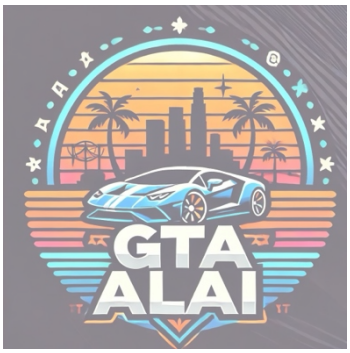
Proyecto Final: Paradigmas y técnicas de programación (GTA ALAI)



Miguel Ángel Huamani Salinas y Jorge Ibinarriaga Robles

Grupo B

Paradigmas y técnicas de programación
3º Grado en Ingeniería Matemática e Inteligencia Artificial

Índice

Link al github:
https://github.com/Ibinarriaga8/Proyecto-Final-Paradigmas

Introducción

Este proyecto final tiene como objetivo el desarrollo de un videojuego mediant eel programa de Unity (version de editor 2022.3.44f1), en el cual se aborda la creación de un mundo virtual interactivo y jugable con la implementación de diversas mecánicas de juego, tales como el manejo de un vehículo y la implementación de diversos sistemas físicos y dinámicos dentro del entorno del juego. El videojuego desarrollado tiene también como objetivo aplicar conceptos avanzados de programación orientada a objetos (POO), para crear una arquitectura de software flexible y modular, garantizando a su vez el cumplimiento de los principios SOLID, como la utilización de patrones de diseño para una mejor mantenibilidad y escalabilidad del código.

A lo largo de este proyecto, se ha trabajado para mejorar y afinar la arquitectura del software y la implementación de las funcionalidades necesarias para que el juego sea jugable y entretenido para los usuarios de este. Además, se ha dado especial atención a los aspectos de física, utilizando componentes como RigidBody para garantizar una experiencia realista en cuanto a la interacción de objetos y personajes en el mundo virtual.

Más adelante en este informe se explicará el funcionamiento y objetivo del juego, así como el proceso de desarrollo y la teoría aplicada a este.

Evolución y arquitectura del videojuego

Diseño inicial

En el diseño inicial del videojuego, se establecieron las historias de usuario que sirvieron como una estructura de la que partir, y cuales serían los diferentes personajes del juego. El jugador, en su rol de conductor de un Sedan, debía escapar de la policía, evitando ser capturado mientras se desplazaba por la ciudad. Además, el jugador debía gestionar su vida a través de la recolección de elementos de mejora (Bufs) que aumentaban su salud, al mismo tiempo que debía esquivar obstáculos que redujeran su velocidad (Debufs), lo que complicaba su huida. Por otro lado, la policía, controlada por un agente de inteligencia artificial, tenía la misión de perseguir al jugador y evitar obstáculos como los Bufs y los Debufs. También debía esquivar a los civiles que se encontraban en las calles, los cuales se movían de manera autónoma, sin interferir directamente en la persecución. De esta manera, el diseño inicial del videojuego desafiaba al jugador a mantener un equilibrio entre la evasión de la policía y la gestión de su salud y velocidad, mientras a su vez este interactuaba con un entorno en constante movimiento.

Diseño final: Explicación de GTA ALAI

El diseño final del juego consiste en un simulador de huida de la policía. Tú, el jugador, eres un delicuente que acaba de cometer un crimen, y tu mision es escapar de la policía.
El jugador tomará el control de un coche deportivo con la misión de superar niveles llenos de desafíos. A lo largo del recorrido, se deben enfrentar obstáculos físicos y enemigos inteligentes (los coches de policía), que perseguirán activamente al coche deportivo.

El coche del jugador cuenta con un sistema que permite maniobrarlo con precisión, ofreciendo funciones como frenado(**SPACE**), dirección (**FLECHAS**) y la posibilidad de reiniciar la posición en caso de quedarse atascado (**TECLA R**).

Lo primero que se observa al comenzar el juego es la siguiente pantalla de inicio junto con una melodía de fondo:

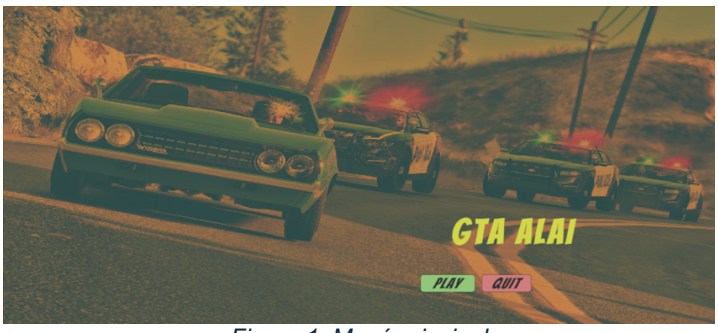


Figure 1. Menú principal

El jugador ganará la partida cuando haya superado los 5 niveles que existen. Cada nivel incrementará la dificultad del juego, generando cada vez más coches de policía que perseguirán a nuestro protagonista. Los niveles de dificultad vendrán indicados por un número de estrellas amarillas en la esquina superior derecha. Para superar los niveles, el jugador deberá conducir hasta un punto en el mapa que representará la huida de la ciudad y por tanto de la policía. Esta salida no se le indicará directamente al jugador, sino que vendrá determinada por la aparición de un helicoptero amigo que guiará al jugador, situándose encima de esta. Si el jugador tiene dificultad viendo dónde está situado el helicopter, el juego incluye sonidos, que aumentarán cuando el jugador más se acerque al helicóptero (sonido de hélices), y cuando más se la acerque un policía (sonido de sirenas).

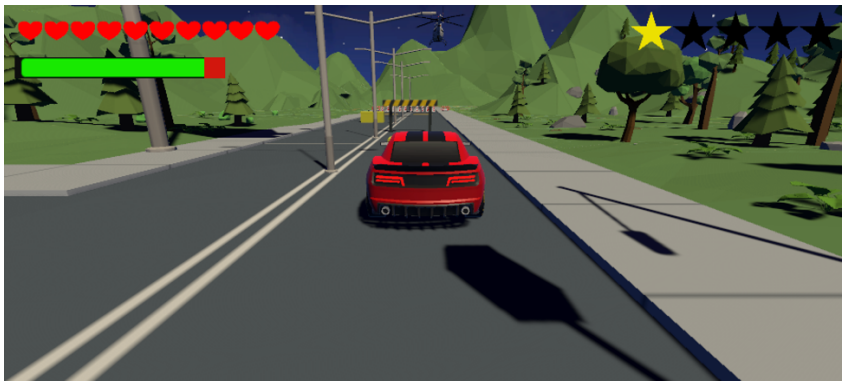


Figure 2. Nivel 1 del juego

Como se observa en la imagen, además de las estrellas de dificultad, en la esquina superior izquierda se muestran tanto las vidas restantes del jugador como una barra indicadora de la vida actual de este. El jugador inicialmente comienza con 10 vidas, y si este es chocado por un Policía o se choca con una Barrera, la vida actual, que comienza en 100, disminuye en -50 y -10 respectivamente.

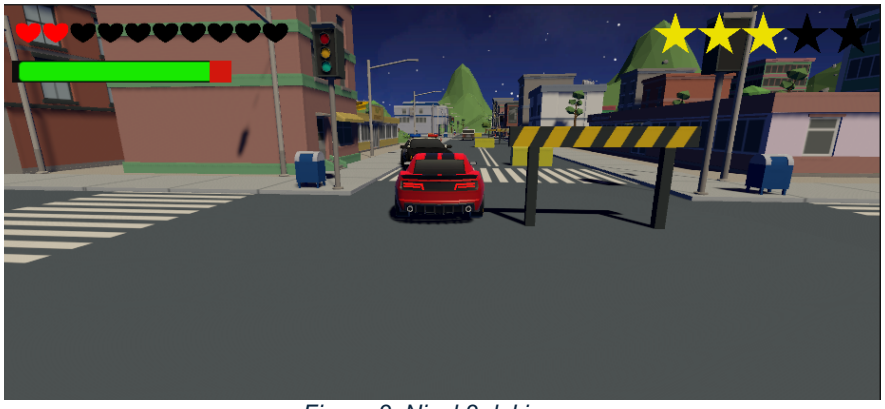


Figure 3. Nivel 3 del juego

Además de obstáculos como vallas, tambien aparecen en el mapa, unos cubos amarillos que representan los Speed Debuffers, realentizadores que si detectan que el jugador pasa a través de ellos, reducirán la velocidad de este en 25 unidades, volviéndolo una presa fácil de la policía.

Si logras superar el juego, completando los 5 niveles con éxito y sin perder todas las vidas, se mostrará la escena final mostrada a continuación en el lado izquierdo. En caso de que este pierda todas las vidas, se mostrará la escena alternativa mostrada a la derecha.

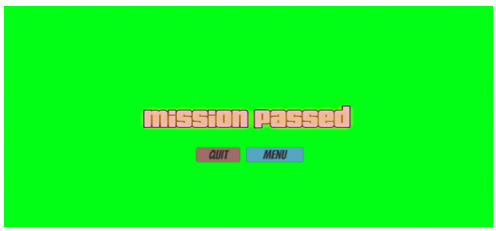


Figure 4. Escena final juego (LOSER)

Figure 5. Escena final juego (WINNER)

Proceso de desarrollo

Programación orientada a objetos (POO)

Principios SOLID: El código sigue de manera consistente los principios SOLID, lo que ayuda a mantener una arquitectura limpia, escalable y fácil de mantener. El **Single Responsibility Principle** lo aplicamos en RaceCar y CameraController, donde cada clase tiene una única responsabilidad (gestionar el coche y controlar la cámara, respectivamente). El **Open/Closed Principle** se cumple mediante la creación de clases abstractas como Obstacle y sus subclases PhysicalObstacle y TriggerObstacle, que pueden extenderse sin modificar el código existente. El **Liskov Substitution Principle** se respeta al usar clases derivadas de Obstacle, como Fence, que cumplen con la interfaz definida sin alterar el comportamiento esperado. El **Interface Segregation Principle** se observa al no sobrecargar las clases con métodos que no sean relevantes para ellas. Finalmente, el **Dependency Inversion Principle** se logra mediante el uso de la clase ObstacleFactory para crear obstáculos, permitiendo cambiar la implementación sin afectar el resto del código.

Patrones de Diseño: Hemos implementado varios patrones como el **Factory Method** que se utiliza con la clase ObstacleFactory y sus subclases (FenceFactory), permitiendo crear diferentes tipos de obstáculos (como Fence) de manera flexible sin alterar el código cliente. El **Strategy Pattern** se implementa en la clase PoliceCar, donde el comportamiento del coche (seguir al jugador) se delega a un agente de navegación (NavMeshAgent), lo que permite cambiar el comportamiento de la persecución sin modificar la clase PoliceCar. Además, el **Observer Pattern** se aplica a través de eventos en el sistema ya que clases como HeartManager, LifeBarManager y StarsManager, quedan se suscriben a eventos con el objetivo de ser notificados cuando estos toman lugar y poder modificar sus respectivos elementos del GUI.

Componentes y elementos integrados

Los elementos del juego están interconectados a través de las físicas que definen su comportamiento en el entorno. Los vehículos, tanto los de policía como los del jugador, están sujetos a la gravedad, fricción con el suelo, y dinámicas de aceleración y desaceleración, lo que afecta su control y respuesta en el terreno.

Las colisiones entre el coche deportivo y los obstáculos, dependerán del tipo de obstáculo. Las vallas (Fences) chocan directamente como un objeto fisico causando daño en el jugador mientras que los cubos amarillos(Speed Debuff) no chocan directamente con el objeto, sino que unicamente detectan cuando el jugador los atraviesa para disminuir de forma temporal la velocidad de este.

Resultados y conclusión

Dificultades y problemas (IMPORTANTE)

Inicialmente hubo problemas para poder trabajar de forma colaborativa a través de GitHub, ya que solo el creador del repositorio podía conectarse y subir cosas, a pesar de haber establecido al otro como contribuidor. Es por esta razón que al principio se han hecho commits únicamente desde el portátil del dueño del repositorio a pesar de haber trabajado ambos.

Finalmente, conseguimos trabajar de forma conjunta haciendo uso de GitHub Desktop, haciendo cada uno su parte del trabajo mediante el uso de ramas (Ibinarriaga y Huamani_Branch). Sin embargo, una última complicación nos ha surgido cuando hemos tratado de hacer merge con el main, por lo que **LA VERSIÓN DEFINITIVA DEL CÓDIGO EN EL GITHUB ESTÁ EN LA RAMA IBINARRIGA en *Final commit with final game*.**

Conclusión final

A pesar de haber enfrentado ciertas dificultades, este proyecto ha sido una valiosa experiencia de aprendizaje en el desarrollo de videojuegos y el trabajo colaborativo, donde hemos logrado hacer uso de lo visto en clase. Este proyecto también nos ha ayudado a desarollar nuestros conocimientos sobre programación orientada a objetos, diseño de juegos y manejo de eventos, consolidando habilidades esenciales para futuros desarrollos en el ámbito de la programación de videojuegos. Quedamos satisfechos con el resultado final de este, un juego entretenido, divertido y con una cierta dificultad que reta a aquellos usuarios que decidan probarlo.