Nombre del juego:

Lane Escape X

Temática y Ambientación:

Nuestro juego es únicamente para PC con el sistema windows y se basará en una idea general donde programaremos dicho juego que tratará sobre el jugador principal el cual tendrá la posibilidad de esquivar obstáculos yendo a una velocidad inicial la cual al pasar el tiempo irá aumentando con su marcador de puntos para dejar un record, y los objetos bajarán más rápido, pensamos en darle una interfaz de 4 carriles y el carro se mantendrá subiendo mientras tiene que esquivar dichos obstáculos, a medida que transcurre el tiempo, la dificultad irá aumentando; ya que la velocidad del carro incrementará progresivamente y los obstáculos descenderán con mayor rapidez, lo que exigirá reflejos y decisiones rápidas.

Mecánica Principal:

El jugador controla un carro (la vocal A) que puede moverse entre cuatro carriles usando las teclas de dirección izquierda y derecha. Desde la parte superior de la pantalla, aparecen obstáculos (#) en carriles aleatorios que se desplazan hacia abajo. A medida que pasa el tiempo, la velocidad del carro y la frecuencia de aparición de obstáculos aumentan, haciendo que el juego sea cada vez más desafiante.

El objetivo es esquivar todos los obstáculos; si el carro choca, el juego termina y se muestra una animación de “Game Over” junto con la puntuación obtenida. El jugador gana puntos conforme se mantenga en juego sin chocar, y si supera su récord anterior, este se guarda automáticamente. Para simular animación, la pantalla se limpia y se vuelve a dibujar en cada frame, dando una sensación dinámica en tiempo real.

Idea General de la Jugabilidad:

El juego reta al jugador a reaccionar rápido para cambiar de carril y evitar obstáculos que aparecen cada vez más rápido, buscando superar su propio récord de puntos y viendo como el ambiente va acelerando haciéndolo más difícil.

Aplicación de los temas vistos en clase:

- **Variables y tipos de datos:**

  Usamos variables enteras **(`int`)** para guardar las posiciones del carro y los obstáculos en pantalla, así como para la velocidad y el puntaje. También utilizamos variables booleanas **(`bool`)** para controlar el estado del juego (por ejemplo, si está activo o si ya terminó). Todos estos datos se organizan usando estructuras **(`struct`)**, lo que nos permite manejar de forma más ordenada la información relacionada con el jugador y los obstáculos.
- **Estructuras condicionales:**

  Las condicionales **(`if`, `else`)** nos ayudan a controlar el movimiento del carro dentro de los límites de los carriles, a decidir cuándo y dónde generar nuevos obstáculos y, lo más importante, a detectar colisiones entre el carro y los objetos que bajan. Si hay una colisión, el juego se detiene.
- **Ciclos:**

  El juego corre dentro de un ciclo principal (**`while`**) que se ejecuta mientras el jugador no haya perdido. Dentro de ese ciclo usamos **(`for`)** para actualizar la posición de los obstáculos, dibujarlos en pantalla y manejar el tiempo de aparición de nuevos obstáculos.
- **Funciones:**

  Modularizar el código usando funciones específicas para tareas como mover el carro, dibujar los elementos en pantalla, detectar colisiones, generar obstáculos, etc. Esto hace que el código sea más limpio, fácil de entender y más sencillo de modificar si queremos hacer cambios.
- **Estructuras de datos:**

  Creamos estructuras **(`struct`)** para agrupar las variables del carro, los obstáculos

y el estado general del juego. Esto nos facilitó mucho la organización y el acceso a los datos.

- **Manejo de archivos:**

Implementamos la opción de guardar el récord más alto en un archivo externo. Cada vez que el jugador supera su mejor puntaje, este se actualiza automáticamente, permitiendo llevar un historial del mejor resultado.

Manejo de errores:

Evitamos errores controlando los límites del movimiento del carro para que no se salga de los carriles. También validamos las entradas del teclado para que no afecten la ejecución del juego.

Funciones:

se implementaron funciones modularizadas para garantizar un código limpio y mantenible. Funciones clave como `GenerarNumeroAleatorio()` (para la aparición de obstáculos), `MoverJugador()` (control de movimiento) y `HayColision()` (detección de impactos) fueron definidas en archivos separados (`Utils.cpp`, `Game.cpp`, `Obstacle.cpp`), siguiendo los principios de encapsulamiento y reutilización. Cada función incluye parámetros específicos (como coordenadas o rangos) y retorna valores que interactúan con la lógica principal del juego, demostrando así una estructura organizada y eficiente.

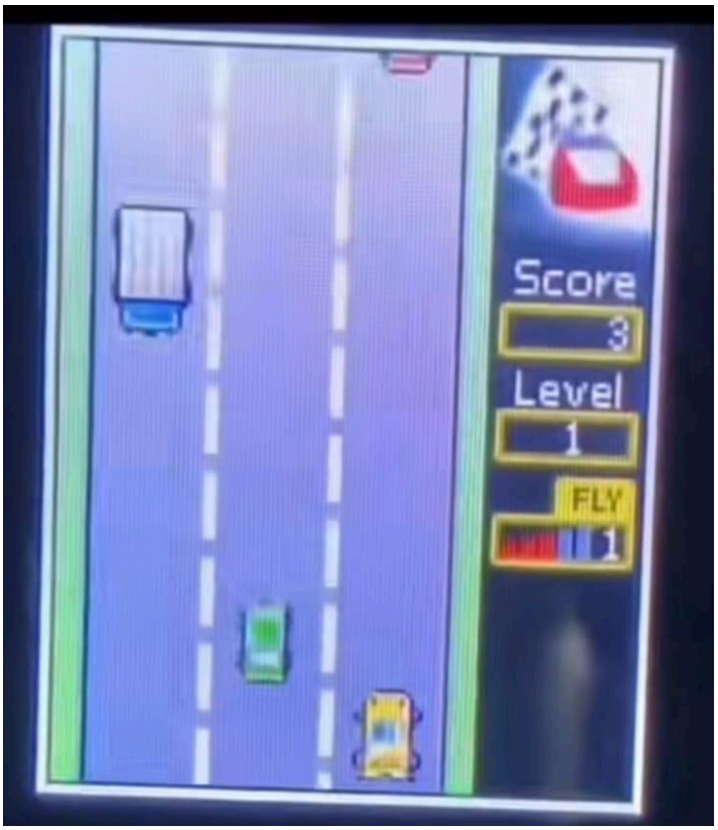
Buenas prácticas y técnicas :

Durante el desarrollo del juego aplicamos buenas prácticas de programación para asegurar que el código fuera entendible, organizado y fácil de mantener. Comentamos cada parte del código explicando su función, lo que facilita su comprensión tanto para nosotros como para cualquier persona que lo revise.

También utilizamos nombres significativos y descriptivos para las variables y funciones, lo cual ayuda a identificar rápidamente su propósito dentro del programa.

El código está dividido en funciones pequeñas y específicas, lo que permite una estructura modular y optimizada. Además, mantuvimos una indentación clara y consistente en todo el proyecto, lo que mejora la legibilidad y organización general del código.

Mockups / Imágenes:



Integrantes:
Franco Landaverde, Josué Alejandro – 00083825

González Recinos, Evelyn Sofía - 00181825

Banderas Ruiz, Andrés Antonio -00044125

Alvarado Landaverde, Cesar Melquisedec -00029425