
**Proyecto I TRON**

Adrian Pereira Ramirez

Ingeniería en Computadores

CE1103 Algoritmos y Estructuras de Datos I

Leonardo Araya Martinez

Cartago, Costa Rica

2024

# Tabla de contenidos

# 1. Introducción

Este proyecto consiste en el desarrollo de un juego estilo Tron, en el cual un jugador compite contra bots en una cuadrícula. El jugador y los bots dejan una estela detrás mientras se mueven, y colisionar con una estela o los bordes del mapa resulta en una explosión. El jugador puede recoger ítems y poderes durante el juego, como combustible, velocidad y escudos, para mejorar su rendimiento. El objetivo es sobrevivir más tiempo que los bots y evitar las colisiones. El proyecto está implementado en C# siguiendo el paradigma de programación orientada a objetos (OOP), aplicando patrones de diseño y una arquitectura de clases bien estructurada.

# 2. Breve Descripción del Problema

El objetivo principal del proyecto es diseñar e implementar un juego de estilo Tron en el que tanto el jugador como los bots se muevan por una cuadrícula, dejando una estela que los destruye si colisionan con ella o con los bordes. El jugador puede recolectar ítems que le proporcionan ventajas temporales como combustible, escudos o mayor velocidad. El desafío está en la implementación de un sistema de colisiones eficiente, movimientos aleatorios de los bots y una interacción correcta entre los jugadores y los poderes.

# 3. Descripción de la Solución

## 3.1 Movimiento de Jugadores y Bots

- Los jugadores y bots se mueven en la cuadrícula, dejando una estela detrás de ellos.
- Se implementó un movimiento aleatorio para los bots, con un porcentaje de probabilidad de cambiar de dirección.
- El jugador puede controlar su moto mediante las teclas del teclado.

## 3.2 Recolección de Ítems

- Tanto el jugador como los bots pueden recoger ítems distribuidos aleatoriamente en la cuadrícula.
- Los ítems incluyen combustible, crecimiento de estela, bombas, escudos e hiper velocidad.

- Los ítems afectan las propiedades del jugador, como el combustible que se recarga o la velocidad que aumenta temporalmente.

### 3.3 Colisiones

- Se implementó un sistema de colisiones que verifica si el jugador o los bots chocan con estelas, bordes o entre ellos.
- Si un jugador o bot colisiona, se elimina del juego mediante una explosión visual.

### 3.4 Estelas

- Los jugadores y bots dejan una estela mientras se mueven. La estela se maneja mediante una lista enlazada que crece cada vez que el jugador se desplaza.

## 4. Alternativas Consideradas

- **Movimiento de los Bots:**
  - Inicialmente se consideró que los bots siguieran patrones de movimiento determinísticos, pero se decidió implementar un movimiento aleatorio con probabilidades para hacer el juego más impredecible.
  - Se pensó en darles inteligencia para esquivar obstáculos, pero fue descartado para mantener la jugabilidad simple.
- **Colisiones:**
  - La detección de colisiones podría haberse implementado utilizando bounding boxes o rectángulos de colisión más avanzados. Se optó por una implementación directa que verifica la posición de cada nodo de la estela comparado con las coordenadas del jugador y los bots.

## 5. Limitaciones y Problemas Encontrados

### 5.1 Sincronización de Movimiento

- Fue un reto sincronizar el movimiento del jugador con los bots sin que hubiera desfases visuales o errores de cálculo.

### 5.2 Dibujo de la Estela

- Al principio, la estela del jugador se dibujaba mal alineada en las celdas. Se resolvió ajustando las coordenadas de la estela al tamaño exacto de la celda.

### 5.3 Alineación del Jugador

- El jugador no estaba centrado correctamente en las celdas del tablero. Se ajustó la lógica de dibujo para que el jugador esté centrado correctamente en una celda al moverse.

### 5.4 Activación de Poderes

- La activación de los poderes recogidos, como el crecimiento de la estela o el incremento de velocidad, no se logró de manera completa. Aunque los poderes se almacenan en el inventario del jugador, no se aplican correctamente en todos los casos, lo que impide su funcionamiento como se había previsto. Esto requerirá mejoras en la lógica de activación de poderes.

# 6. Diseño General

## 6.1 Diagrama UML

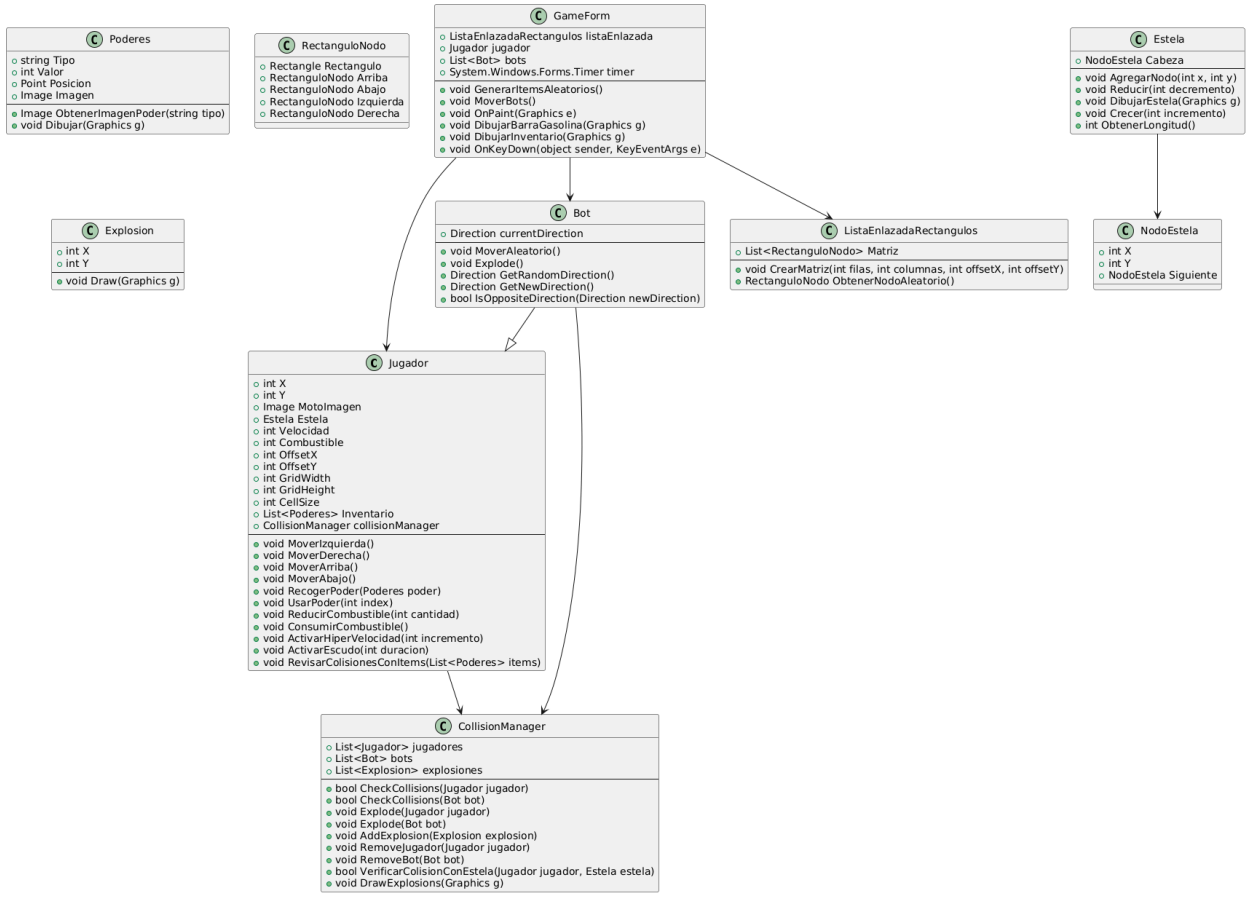


**Figura 1**: Diagrama UML del Proyecto Tron.

### 6.2 Explicación del Diseño

El diseño está organizado en varias clases principales, cada una con una responsabilidad bien definida:

- **Clase Jugador:** Representa al jugador que se mueve por la cuadrícula, recoge poderes y gestiona su combustible. Hereda de esta clase la clase `Bot`, que añade comportamiento aleatorio a los movimientos.
- **Clase Estela:** Se encarga de gestionar la estela que dejan el jugador y los bots. Utiliza una lista enlazada para representar las posiciones anteriores del jugador.
- **Clase Poderes:** Representa los ítems que el jugador o los bots pueden recoger. Incluye ítems como combustible, escudos y bombas.
- **CollisionManager:** Esta clase gestiona las colisiones entre los jugadores, bots y sus respectivas estelas. También maneja la lógica de explosiones y la eliminación de jugadores y bots.

# 7. Conclusiones

Este proyecto de juego estilo Tron fue un reto divertido y útil para practicar conceptos de programación orientada a objetos (OOP) en C#. El mayor desafío fue implementar un sistema eficiente de colisiones y movimientos, y asegurar la correcta interacción entre los jugadores, bots y los ítems del juego. Los principales aprendizajes fueron la importancia de una buena organización de las clases y el manejo adecuado de los recursos visuales en un entorno dinámico como un videojuego.