1

Proyecto Algoritmos y Estructura de Datos I TRON C#

Nombre: Jorge Alberto Marín Navarro

Carné: 2024174059

Curso: Algoritmos y Estructura de Datos I

Proyecto: Tron C#

Profesor: Leonardo Araya

I. Tabla de Contenidos

Documentación del Proyecto Algoritmos y Estructura de Datos I

II. INTRODUCTION

El Proyecto UTRON es un videojuego basado en el clásico juego de TRON, desarrollado en C# utilizando la tecnología de Windows Forms. El objetivo es que los jugadores controlen una moto que deja una estela en su recorrido, con la cual deben evitar colisionar, al tiempo que recolectan ítems y poderes para obtener ventajas temporales. El proyecto implementa bots que también se desplazan aleatoriamente por el campo de juego con lógica de colisiones y recolección.

El desarrollo de este juego implica resolver problemas relacionados con la generación de ítems y poderes, la detección de colisiones y la implementación de una interfaz gráfica amigable para el usuario, todo en un entorno orientado a objetos.

III. BREVES DESCRIPCIÓN DEL PROBLEMA

El juego tiene varios desafíos clave:

Movimiento y colisiones: La moto debe moverse de manera continua en un grid y dejar una estela. Las colisiones ocurren si la moto choca con su propia estela o con un bot.

Generación de ítems y poderes: Los ítems y poderes deben generarse de forma aleatoria en el grid, ser recogidos por la moto y aplicar sus efectos temporales.

Comportamiento de los bots: Los bots se mueven aleatoriamente, simulando otros jugadores, y deben tener la misma lógica de colisiones y recogida de ítems que el jugador.

Interfaz gráfica: El juego debe proporcionar una visualización clara de la moto, la estela, los bots, los ítems y poderes recogidos, además de mostrar el estado actual del jugador (combustible, velocidad, etc.).

IV. Descripción de la solución.

Descripción de la Solución

Requerimiento 1: Movimiento y colisiones

La clase Moto gestiona el movimiento de la moto y la creación de su estela. Utiliza un temporizador para actualizar la posición de la moto en intervalos regulares. Las colisiones se verifican comparando la posición de la moto con las posiciones de la estela o de los bots.

Alternativas Consideradas:

Usar una matriz para representar el campo de juego. Sin embargo, se decidió utilizar listas enlazadas para gestionar mejor la estela.

Limitaciones:

La velocidad de la moto puede afectar el rendimiento gráfico si se incrementa demasiado.

Requerimiento 2: Generación de ítems y poderes

Se implementaron las clases Item y Poder, que se generan en posiciones aleatorias del grid usando el método ObtenerPosicionAleatoria() de la clase Grid. Los ítems se gestionan como una cola (FIFO), y los poderes como una pila (LIFO), con sus respectivos efectos temporales.

Alternativas Consideradas:

Inicialmente, se pensó en generar ítems en lugares fijos, pero se descartó para hacer el juego más dinámico.

Requerimiento 3: Comportamiento de bots

Cada bot está representado por la clase Bot, que implementa un movimiento aleatorio en el grid. Los bots verifican colisiones con otros objetos en el juego de la misma manera que la moto del jugador.

Alternativas Consideradas:

Se consideró implementar un comportamiento más complejo para los bots, pero se priorizó un movimiento aleatorio por simplicidad.

Requerimiento 4: Interfaz gráfica

La interfaz se implementó utilizando Windows Forms. Los paneles gráficos muestran los ítems y poderes recogidos, así como el estado del jugador (combustible, velocidad, etc.). Se utiliza la clase Form1 para manejar la lógica visual del juego, incluyendo la actualización de la estela, los ítems, y los bots.

Limitaciones:

La interfaz puede volverse menos fluida si se aumenta el número de elementos en pantalla (estelas largas, múltiples bots, etc.).