I Proyecto – TRON

Institución: Instituto Tecnológico de Costa Rica **Fecha: 0**8/09/2024.

Profesor: Leonardo Araya.

Curso: Algoritmos y estructuras de datos I (CE 1103).

Estudiante: Justin Solano Calderón. **Carné:** 2024109807.

Resumen: El proyecto "TRON" desarrolla un juego de carreras de motos de luz, centrado en la aplicación de estructuras de datos y programación orientada a objetos. Se utilizan listas enlazadas, pilas y colas para modelar el comportamiento dinámico de las motos, como su movimiento, estelas destructivas, y manejo de poderes y ítems. El juego incluye características como un grid que simula el entorno de juego, bots con comportamientos aleatorios, y una interfaz visual para facilitar la interacción del jugador. El proyecto resalta la importancia de una implementación eficiente de estructuras de datos para manejar eventos dinámicos, optimizar el rendimiento y mejorar la experiencia del usuario. A lo largo del desarrollo, se enfrentaron desafíos significativos, como la gestión de colisiones y la aleatoriedad de los bots, que fueron abordados con soluciones técnicas específicas.

Palabras clave: Listas enlazadas, Pilas y Colas, Diagrama UML, Grid, Bots en videojuegos.

Introducción:

El presente proyecto, titulado **TRON**, tiene como objetivo principal la implementación de un juego de carreras de motos de luz. En este juego, cada jugador controla una moto que deja una

estela destructiva tras de sí, con la misión de evitar colisionar con otras motos y sus estelas, mientras intenta eliminar a los adversarios.

El desarrollo de este proyecto está enfocado en aplicar estructuras de datos lineales, tales como listas, pilas y colas, para modelar el comportamiento de las motos como los elementos interactivos del juego. Además, el uso de patrones de diseño y la creación de diagramas UML permiten un enfoque orientado a objetos que facilita la implementación y escalabilidad del juego.

Este proyecto se desarrolla como parte del curso Algoritmos y Estructuras de Datos I (CE 1103) del Instituto Tecnológico de Costa Rica.

Descripción del problema:

El problema principal que este proyecto busca resolver es la simulación de un juego de carreras de motos de luz en un entorno de malla o "grid", donde múltiples jugadores, tanto humanos como bots, compiten en un espacio limitado. Cada jugador maneja una moto que se mueve en cuatro direcciones, dejando una estela destructiva que elimina a cualquier moto que la cruce. Las motos nunca se detienen y el jugador solo tiene control sobre la dirección de la moto.

El juego presenta varios desafíos que deben ser resueltos mediante estructuras de datos lineales. Por ejemplo, la moto y su estela se implementan como una lista enlazada simple, donde la longitud de la estela puede cambiar dependiendo de varios factores, como la velocidad y los ítems recogidos. Los jugadores también deben gestionar una pila de poderes, los cuales pueden aplicar de manera estratégica, y una cola de ítems que afectan permanentemente el comportamiento de la moto.

Adicionalmente, el juego debe manejar una serie de eventos que complican la jugabilidad, tales como la generación aleatoria de ítems y poderes en el mapa, y la aparición de bots que simulan a otros jugadores, aumentando el nivel de dificultad. El reto consiste en implementar las estructuras de datos de manera eficiente para gestionar tanto las motos y sus atributos (velocidad, combustible, estela) como los eventos dinámicos del juego, garantizando que la experiencia sea fluida y desafiante.

Descripción de la solución:

- Implementación de las motos y sus atributos:

  La clase Moto se diseñó con varios atributos que controlan aspectos clave del comportamiento en el juego: velocidad, combustible, estado de la moto (“viva” o destruida), items, poderes y la estela. Para el movimiento de las motos, se utilizó un sistema de coordenadas basado en la suma y resta de valores, lo que permite que las motos se desplacen en las direcciones permitidas (arriba, abajo, izquierda y derecha) dentro del grid.

  Se incorporó un sistema de teletransporte para garantizar que las motos no salgan del grid. Si una moto alcanza el borde, automáticamente aparece en el lado opuesto de la malla. Adicionalmente, se verifica constantemente si la moto entra en colisión con su propia estela, otra moto o su estela, lo que provoca su destrucción. Al destruirse, los nodos de la moto y su estela se restablecen a su estado original y se eliminan del grid. El movimiento de las motos se controla mediante las teclas presionadas, que son comparadas para ejecutar las acciones correspondientes.

    - Problemas enfrentados: Uno de los mayores retos fue gestionar la implementación de todos los requerimientos sin que colisionaran entre sí. El uso de otros métodos o comandos en C# podría mejorar la eficiencia de la solución. Además, el control y movimiento de las motos dentro del grid requirió una optimización adicional para evitar conflictos en el procesamiento.

- Implementación del grid:

  El grid o malla se diseñó como una matriz de nodos, donde cada nodo tiene información sobre sus nodos adyacentes (arriba, abajo, izquierda y derecha). Para acceder y actualizar los nodos, se implementó un método que permite identificar y modificar los nodos adyacentes, facilitando el movimiento de las motos dentro del grid.

Además, se diseñó un sistema que recolorea los nodos en tiempo real. Cuando una moto pasa por un nodo, este se pinta para representar visualmente su estela. Cuando la estela desaparece o la moto es destruida, el nodo vuelve a su color original, asegurando una representación visual clara de la malla y los elementos del juego.

  - Problemas enfrentados: La principal dificultad fue asegurarse de que cada nodo pudiera identificar correctamente a sus adyacentes y responder adecuadamente a las interacciones. Garantizar que el sistema de actualización de nodos fuera preciso y eficiente también fue un desafío importante.

- Gestión de items y poderes:

  Los items en el juego, como las celdas de combustible o aumentos de velocidad, afectan permanentemente las características de la moto una vez que son recogidos. Cuando una moto recoge un item, sus atributos se actualizan inmediatamente. Los poderes, en cambio, solo afectan temporalmente a las motos y se gestionan mediante un sistema de timers que controlan la duración de su efecto.

  Por ejemplo, los poderes de hiper velocidad o escudo solo están activos durante un tiempo limitado. El sistema de temporizadores fue implementado para asegurarse de que los efectos sean revertidos después de cierto tiempo, restaurando las características originales de la moto una vez que el tiempo de uso del poder ha expirado.

  - Problemas enfrentados: Uno de los retos fue la implementación visual de los items y poderes, y asegurarse de que los poderes afectaran a las motos temporalmente y de forma precisa, sin generar conflictos con las modificaciones permanentes que introducen los items.

- Implementación de los bots:

  Los bots en el juego heredan los comportamientos de la clase Moto, lo que les permite moverse y actuar de manera similar a los jugadores controlados manualmente. Sin embargo, su comportamiento fue randomizado para que se movieran de manera

impredecible, además de una seríe de medidas que garantizan que estos no se eliminarán accidentalmente.

Para garantizar un comportamiento variado y desafiante, se generaron acciones aleatorias para los bots, tanto en términos de movimiento como en la utilización de los poderes. Esto añade un nivel de complejidad al juego, ya que los bots simulan el comportamiento de jugadores reales, pero con un componente aleatorio que incrementa el desafío para los jugadores humanos.

- Problemas enfrentados: El reto principal fue crear un comportamiento aleatorio para los bots que cumpliera con las reglas del juego sin ser predecible. Se requirió ajustar el balance entre la aleatoriedad y la coherencia en sus acciones para evitar que fueran demasiado erráticos o ineficaces.

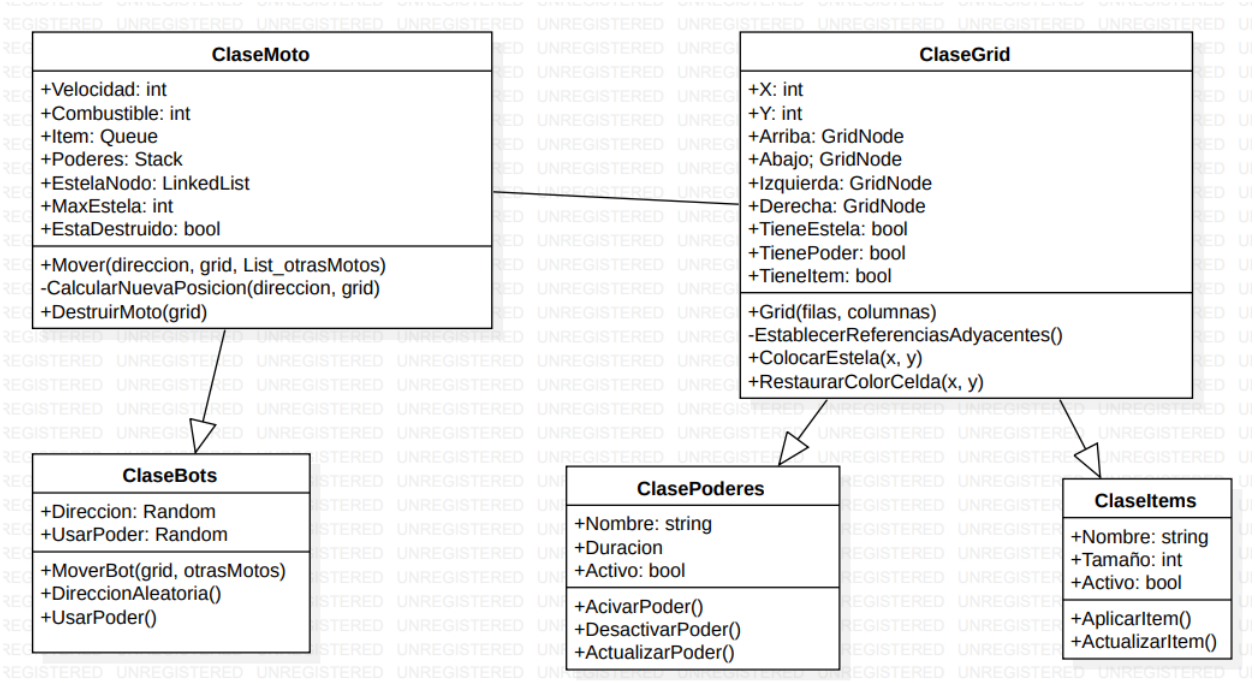
- Interfaz visual:

La interfaz visual del juego fue diseñada para proporcionar una representación clara del estado de las motos y el juego en general. Se optó por utilizar una barra de progreso para mostrar el nivel de combustible de las motos, lo que facilita a los jugadores entender cuánta energía les queda para seguir en la partida.

En el lateral izquierdo se encuentra información para el uso del jugador, como que el movimiento se controla con las flechas del teclado, su color por defecto.

Diseño general:

- Diagrama UML de clases:



- Explicación del diseño:

  El diagrama UML del proyecto muestra la estructura del juego TRON, representando las clases principales y sus interacciones. A continuación, se detalla la función de cada clase y los atributos y métodos más relevantes.

  - Clase Moto

  La clase Moto es el núcleo del juego, que representa las motos controladas por el jugador o los bots. Esta clase contiene varios atributos esenciales para la funcionalidad del juego:

  Velocidad: Un entero que define la rapidez con la que se mueve la moto.
  Combustible: Representa el nivel de energía de la moto, que decrece con el movimiento.
  Item: Una cola (queue) que almacena los items recogidos, los cuales afectan permanentemente las características de la moto.
  Poderes: Una pila (stack) que almacena los poderes recogidos, que pueden ser activados temporalmente.
  EstelaNodo: Una lista enlazada que simula la estela destructiva que sigue a la moto.
  MaxEstela: Define el tamaño máximo que puede alcanzar la estela.
  EstaDestruido: Un booleano que indica si la moto ha sido destruida.
  Los métodos principales incluyen:

Mover(direccion, grid, List_otrasMotos): Gestiona el movimiento de la moto en función de la dirección y comprueba posibles colisiones con otras motos o estelas.
CalcularNuevaPosicion(direccion, grid): Método privado que calcula la nueva posición de la moto en el grid.
DestruirMoto(grid): Reinicia la moto y elimina su estela cuando esta se destruye.

- Clase Grid

La clase Grid representa la malla sobre la cual se mueven las motos. Cada celda del grid es un nodo que contiene referencias a sus nodos adyacentes (arriba, abajo, izquierda, derecha), así como estados booleanos que indican si en ese nodo hay una estela, un poder o un item.

X, Y: Coordenadas de la celda en el grid.
Arriba, Abajo, Izquierda, Derecha: Referencias a los nodos adyacentes.
TieneEstela, TienePoder, TieneItem: Refencia que indica si la celda contiene una estela, poder o item.
Los métodos principales son:

Grid(filas, columnas): Constructor que inicializa el grid con un número de filas y columnas.
EstablecerReferenciasAdyacentes(): Configura las referencias a los nodos adyacentes para cada celda del grid.
ColocarEstela(x, y): Método que coloca una estela en una posición específica del grid.
RestaurarColorCelda(x, y): Restaura el color de una celda cuando se elimina una estela o se destruye una moto.

- Clase Bots

Los Bots heredan comportamientos de la clase Moto y se mueven de manera aleatoria por el grid, simulando jugadores controlados por inteligencia artificial. Los atributos principales de esta clase son:

Direccion: Generada aleatoriamente para decidir hacia dónde se moverá el bot.
UsarPoder: Un valor aleatorio que determina si el bot activará un poder.
Métodos clave:

MoverBot(grid, otrasMotos): Gestiona el movimiento del bot, verificando colisiones con otras motos y estelas.
DireccionAleatoria(): Genera una nueva dirección de movimiento aleatoria.
UsarPoder(): Activa un poder de manera aleatoria.

- Clase Poderes

Los Poderes son objetos que las motos pueden recoger para obtener efectos temporales.
Entre los atributos más importantes están:

Nombre: El nombre del poder, como "Escudo" o "Hiper Velocidad".
Duracion: Tiempo durante el cual el poder estará activo.
Activo: Un booleano que indica si el poder está activo.
Los métodos relevantes son:

ActivarPoder(): Activa el poder y aplica su efecto temporal sobre la moto.
DesactivarPoder(): Desactiva el poder y restablece las características de la moto.
ActualizarPoder(): Actualiza el estado del poder, controlando su duración.

- Clase Items

Los Items afectan de manera permanente los atributos de la moto cuando son recogidos.
Los atributos clave de esta clase son:

Nombre: El nombre del item, como "Celda de Combustible" o "Aumento de Velocidad".
Tamaño: Indica el tamaño o cantidad del item, por ejemplo, cuánto aumenta el combustible o la estela.
Activo: Un booleano que indica si el item está siendo aplicado.

Métodos:

AplicarItem(): Aplica el efecto del item a la moto.
ActualizarItem(): Actualiza el estado del item, modificando los atributos correspondientes de la moto.

Conclusiones:

El desarrollo del juego TRON como proyecto académico hizo posible aplicar de manera práctica conceptos fundamentales de algoritmos y estructuras de datos, así como habilidades de programación orientada a objetos. A través de la implementación de listas, pilas y colas, se resolvieron los desafíos del manejo de objetos dinámicos en el juego. Además, el uso de patrones de diseño y la creación de diagramas UML facilitan la comprensión del sistema usado como solución. Las dificultades enfrentadas, como la gestión eficiente de colisiones y la creación de comportamientos aleatorios para los bots, fueron resueltas mediante una optimización cuidadosa del código y del diseño del sistema. Este proyecto no solo proporciona una experiencia de juego atractiva, sino que también sirve como un aprendizaje valioso en la aplicación de estructuras de datos y diseño de software en un entorno donde es factible no solo en un escenario academico.

Referencias:

1. Danisable, “Programación en C# - Curso completo,” YouTube, 2023. [Enlace: https://www.youtube.com/watch?v=Mr2n06yvk28&list=PLAzlSdU-KYwVRRO6P9fn1LcoXSITIIev0].

2. MooICT, “C# Tutorial | Learn C# in One Video,” YouTube, 2023. [Enlace: https://www.youtube.com/watch?v=TzaCn1ZPalI&t=2s].

3. Desarrollo Web, “Manual de C#,” 2023. [Enlace: https://desarrolloweb.com/manuales/manual-c-sharp].

4. GitHub, “C# Language Specification,” GitHub, 2023. [Enlace: https://github.com/dotnet/csharplang].

5. T. C. Smith, “Game Development in C#: Basics of 2D Game Programming,” 2023. [Enlace: https://www.tutorialspoint.com/csharp/csharp_game_development.htm].