Instituto Tecnológico de Costa Rica

Ingeniería en Computadores

Algoritmos y Estructuras de Datos I

Proyecto I : Tron

Grupo : 1

Profesor :

Leonardo Araya

Miembros de Grupo :
Jiacheng Tan He

Septiembre 8, 2024

Tabla de contenidos

Introducción

En este proyecto, se implementará una solución para el juego “Tron”, utilizando estructuras de datos lineales como listas, pilas y colas. Tron es un juego donde los jugadores controlan motocicletas de luz que dejan una estela destructiva a su paso. El objetivo de este proyecto es aplicar conceptos de estructuras de datos, algoritmos y patrones de diseño para crear una solución eficiente y funcional. Adicionalmente, se hará uso de diagramas UML para modelar la solución y se aplicarán buenas prácticas de programación.

Descripción del problema

El juego Tron consiste en motocicletas de luz que se mueven dentro de una malla o grid, controladas por los jugadores mediante el teclado. Cada motocicleta deja una estela destructiva y su objetivo es evitar colisiones tanto con las estelas de otros jugadores como con los bordes de la malla. Las motocicletas tienen atributos como velocidad, tamaño de la estela y combustible, los cuales influyen en su desempeño. Además, existen ítems y poderes que los jugadores pueden recoger para obtener ventajas temporales o permanentes. El reto principal del proyecto es implementar el comportamiento de las motocicletas y la interacción con los ítems utilizando estructuras de datos adecuadas.

Descripción de la solución

1. Las motos de luz se implementan como una lista enlazada simple. Cada moto deja una estela destructiva a su paso. El movimiento de las motos se puede asemejar al de una oruga. Cuando la moto se crea, inicialmente tendrá una estela de 3 posiciones.

La solución empleada para este problema se centró en primero definir una clase de estela que se encargaría de crear estelas. Dando también la capacidad de agregar estelas extras para también complementar el poder de aumentar estela. A su vez, luego se implementó la clase Moto la cual se encarga de tener todos los atributos necesarios como posición, vida, combustible, uso de ítems y poderes, movimientos, entre otros. Por otro lado, aunque la clase en sí no heredé ningún atributo de otra, esta si utiliza las funciones o métodos de otras para complementar la funcionalidad del código.

2. Las motos tienen los siguientes atributos:

• Velocidad: valor aleatorio entre 1 y 10 que determina qué tan rápido una moto se mueve
• Tamaño de la estela: valor que determina el largo de la estela. Inicialmente vale 3. 10

• Combustible: valor que determina cuánto combustible tiene la moto. Se consume automáticamente dependiendo de la velocidad de la moto a una tasa de 1 celda de combustible por cada 5 elementos de la malla recorridos. Es un valor de 0 a 100.
• Items: cola de elementos que afectan permanentemente la moto
• Poderes: pila de poderes que afectan temporalmente la moto

Los atributos de velocidad, estela y combustible se establecieron desde un inicio. A su vez, también se integraron los métodos encargados de su reducción de forma correcta. Estableciendo los valores así correspondiera y reduciéndolos como se indicaba, todos estos métodos se realizaron en la clase Moto para mantener el orden. Por otro lado, la cola de ítems y de poderes también se mantuvieron en la clase Moto en donde se integraron las funciones de recolección y almacenamiento de ítems y poderes a la vez que el uso de cada uno.

3. Cuando una moto se destruye los ítems y poderes que tenía aun sin usar, se colocan en el mapa en posiciones aleatorias.

Para la realización de la solución, se empleó el uso de un método en el cual este detectaba si la cola de ítems y la pila de poderes contenían algo. De ser este el caso, el método utilizaba el nombre para volver a colocar el ítem o poder dentro del mapa, posicionándolos en zonas aleatorias para que estos sean agarrados por otros bots o el jugador.

4. El jugador escoge cuándo ejecutar los poderes, los cuales se ejecutan en un orden definido por el jugador. En pantalla, el jugador podrá ver la pila de poderes. Presionando un botón puede ir moviendo el elemento del tope de la pila para dejar el poder que más le convenga de primero. Cuando presione el botón de aplicar el poder, se aplicará siempre el elemento del tope.

Para la creación de este método se decidió crear un método en el cual se tomaban dos pilas o stacks, el principal de recolección de poderes y uno secundario el cual serviría para poder navegar entre los poderes adquiridos. Cambiando del poder en el tope o cima para poder usar el deseado.

5. Los ítems se aplican en el orden de llegada automáticamente con un delay de 1 segundo entre la aplicación de uno y otro, aplicando prioritariamente las celdas de combustible. Si el combustible está lleno, la celda se vuelve a insertar en la cola sin aplicarse.

Para implementar la solución de este problema, se utilizaron los métodos de Enqueue y Dequeue, en donde, con el delay de un segundo, los ítems se van usando de forma procedural. A su vez, si se detectaba que el combustible se encontraba al 100%, el ítem de combustible sería devuelto a la cola.

6. Una moto se destruye al chocar con otro jugador (ambos mueren), cruzar una estela o quedarse sin combustible. Las motos nunca se detienen. El jugador únicamente puede cambiarlas de dirección.

La solución implementada ante este problema fue la de agregar un atributo booleano el cual se encarga de definir si el jugador o el bot están vivos. Al detectar una colisión, ya sea con un muro, moto, estela o consigo mismo, este provocaría la activación del método Morir() en donde el booleano sería puesto en falso, deteniendo el juego si todos los bots han muerto o si el jugador ha muerto.

7. Tal y como se indicó el mapa es un grid o malla de tamaño fijo. Se implementará mediante una lista enlazada en la que cada nodo posee 4 referencias a otros nodos, formando así la red. Cuando el juego inicia, se carga el mapa de un tamaño previamente definido. El jugador utiliza las flechas del teclado para mover la moto en el grid

El mapa se implementó mediante nodos dentro de un PictureBox, en donde se establecieron los nodos Norte, Sur, Este y Oeste para formar la red, este grid se creó con un tamaño de 104 x 87. Por otro lado, el movimiento del jugador se implementó mediante la detección de las flechas del teclado, en donde estas se encargan solamente de cambiar la dirección de la moto, ya que esta no se puede detener.

8. En la red aparecen ítems y poderes aleatoriamente, que pueden recoger el jugador.

Los ítems incluyen:

• Celda de combustible: incrementa el combustible de la moto. Cada celda tiene una capacidad aleatoria.

• Crecimiento de estela: incrementa el tamaño de la estela en un tamaño variable. Cada ítem tiene un valor aleatorio de 1 a 10 que determina cuánto va a incrementar la estela.

• Bombas: cuando un jugador toma una bomba, explota.

Los poderes incluyen:

• Escudo: permite que la moto se haga invencible por un tiempo variable. Afecta visualmente la moto.

• Hiper velocidad: aumenta la velocidad de la moto en un valor aleatorio y por un periodo aleatorio. Afecta visualmente la moto.

Para la implementación de cada consumible se realizó lo siguiente:

Combustible : Para este, se creó un método dentro de la clase Ítems en donde, con un valor aleatorio del 1 a 100, se agregaba combustible al tanque del jugador o bot.

Crecimiento de estela : Para este se tomó ventaja de la clase de NodoEstela en donde se realizó un método el cual facilita el incremento de estelas. Formando un método el cual aumentaba la estela en un valor aleatorio.

Bombas : Para la implementación de las bombas se tomó una ruta mas sencilla en donde la bomba, ya que esta explota al entrar en contacto, actúa como muro. Matando instantáneamente a cualquiera que entre en contacto con ella.

Escudo : En el caso del escudo, se implementó un método en la clase de moto en donde este afectaría al booleano de Viva de la moto en cuestión, anulando la verificación de colisiones por un tiempo aleatorio.

Hipervelocidad : En el caso de la hipervelocidad, se decidió modificar el multiplicador de cada moto dentro de la clase de Moto. Aumentando la velocidad en dos veces por un tiempo aleatorio.

9. Existen bots que simulan a otros jugadores. Todas las reglas anteriores aplican para los bots. Su comportamiento es aleatorio. Al menos 4 bots simultáneos deberán aparecer en el juego

Ante este problema, se implementó la solución de crear una clase Bot que heredó los atributos de la clase Moto. Por consiguiente estos se inicializaron en el código y se incluyen en una lista. Esto se debió a que, para realizar un movimiento automático y la recogida y uso automático de ítems y poderes, el uso de una lista para poder aplicar los métodos especializados de la clase Bot proveía de mayor eficiencia.

Diseño general : Diagrama UML