Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

Documentación I Proyecto

Algoritmos y Estructuras de Datos I

Santiago Robles Obando

Carné: 2022207100

Grupo 1

II SEMESTRE

2024

Introducción

La presente documentación se realiza sobre el proyecto de software de un videojuego llamado Tron. El principal objetivo de este proyecto es el de implementar estructuras de datos lineales de manera creativa para resolver problemas complejos. La implementación de estructuras de datos lineales como Listas enlazadas, stacks y queues para resolver los mecanismos del videojuego. El proyecto se realizará utilizando el lenguaje de programación de C# con interfaz gráfica de Windows Forms. Usando el IDLE de Visual Studio.

En esta documentación se presentaran los diagramas UML de modelado sobre los objetos utilizados en este proyecto, así como una explicación de los algoritmos desarrollados y distintos patrones de diseño para resolver cada uno de los requerimientos.

Link del Repositorio de GitHub del Proyecto: https://github.com/santiroob/Tron-.git

Contenido

Breve descripción del problema

El juego trata de una malla sobre la cual navegan motos en 4 direcciones, y cada jugador compite por eliminar al resto y sobrevivir sin entrar en contacto con las estelas destructivas de las motos de su alrededor. Además, existirán distintos poderes e ítems que los jugadores podrán utilizar para adquirir ventajas en la partida. Se utilizará programación orientada a objetos.y estructuras de datos lineales para resolver los problemas.

Lista de Requerimientos:

001	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 moto funciona con una lista enlazada que tiene un nodo principal Head, de tipo Moto, y el resto de nodos que se añaden son del tipo Trail, agregando nodos de forma iterada 3 veces para cuando se crea la moto.

002	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.
	• **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

Se crea un atributo de tipo int para la velocidad, el tamaño y el combustible, y se crean atributos de tipo Queue y Stack (Clases creadas aparte) para los ítems y los poderes.

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

Por medio de una función que obtiene nodos aleatorios que estén vacíos se colocan los poderes e ítems que tenia la moto una vez destruida, en el método Destroy.

004	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.

Esto se realiza creando el stack con una double-ended list, de forma que se almacene el poder que está al final del stack, y además añadirle un método al stack que permita que se haga un Pop y el objeto que se recibe se le asigne como last del stack. Es importante que este requerimiento cambia la lógica de stack asimilándola a un queue.

005	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.

Debido a que no hay diferencia entre ítems que sean combustible (al ser valores aleatorios), se agrega una condición donde si el objeto es de combustible se asigne como nuevo first del queue. De caso distinto se trabaja como un enqueue normal. Generando una priority queue. Al utilizarse automáticamente los ítems, no se le da mucho uso a los queue debido a que no es tan sencillo obtener una cantidad de ítems en cortos intervalos de tiempo.

006	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.

Las motos al navegar por el mapa tienen un atributo que obtiene (en caso de que el nodo siguiente no sea vacío), el objeto que se encuentre en este. De esta forma determina si hay un Trail, o una moto para llamar al método destroy. En caso de que el atributo de combustible sea cero también se llama a este método.

007	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*.

Como lo indica el requerimiento, se modela una lista doblemente enlazada bidimensional, donde los nodos tienen un atributo hacia arriba, abajo, izquierda y derecha. De esta forma se crea el grid por el que navegan las motos que utilizan este tipo de nodo.

008	En la red aparece 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 item 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.	

Celda de combustible: Utilizando un valor de la función random.Next, se obtiene el valor y se le suma al atributo de combustible.

Crecimiento de estela: De forma homóloga se obtiene el valor y se ajusta al atributo TrailSize, Se llama a una función de actualizar estela que añade y dibuja los nuevos nodos de la lista enlazada de la moto.

Bomba: Llama al método destroy cuando es utilizada.

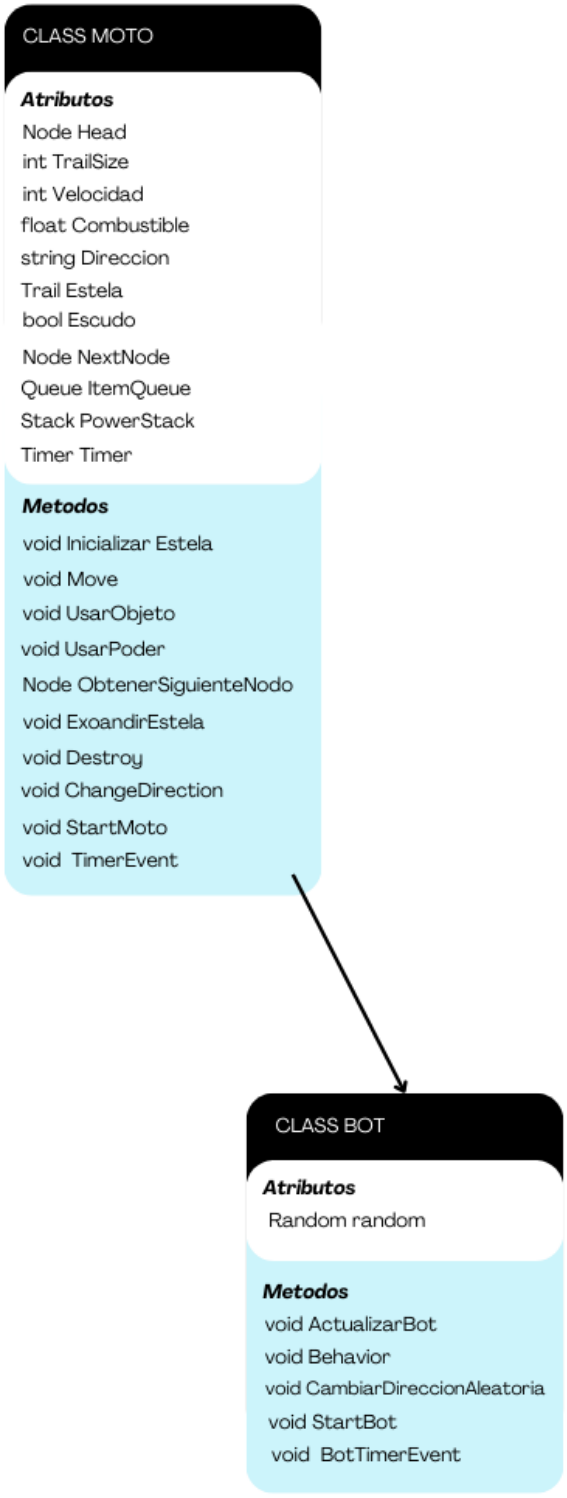
Escudo: Las motos poseen un atributo bool de escudo, el cual se le asigna True, por el periodo de tiempo aleatorio cuando se usa este poder

Hiper Velocidad: Se obtiene un valor random y se le suma al atributo de velocidad. Luego regresa al valor original.

009	Existen *bots* que simulan otros jugadores. Todas las reglas anteriores aplican para los *bots*. Su comportamiento es aleatorio. Al menos 4 bots simultáneos deberán aparecer en el juego	20

Se crea una clase bot, que hereda de la calse Moto, y se le crean métodos específicos para manejar los eventos de forma aleatoria, como cambio de direcciones o si agarrar o no un poder o item. Sin embargo que el comportamiento sea aleatoreo es una limitación ya que no tienen como tal una inteligencia y no representan un reto para los jugadores.

Diagramas UML



CLASS NODE

Atributos

int x
int y
Node Up
Node Down
Node Right
Node Left
object Occupied

CLASS ITEM

Atributos

Random random
int Itemtype
Item Next

CLASS POWER

Atributos

Random random
int Powertype
Power Next

CLASS TRAIL

Atributos

Node current
Trail Next

CLASS MAP

Atributos

Node Head
int Rows
int Cols
Random random

Metodos

void BuildGrid
void GenerateItemsRandomly
List<Node> GetAllNodes
Node GetRandomEmptyNode
void PrintGrid
Node GetNodeAt
void PrintAdjacentNode

CLASS Queue

Atributos

Item Front
Item Next
item Rear
int Count

Metodos

bool isEmpty
void enqueueItem
Item Dequeue

CLASS Stack

Atributos

Power Head
Power Bottom
int size

Metodos

bool isEmpty
void Push
void Pop
void Switch