**Instituto Tecnológico de Costa Rica**

**Escuela de Ingeniería en Computadores**

**Algoritmos y estructuras de datos I**

**CE1103**

**Profesor: Isaac Ramírez Herrera**

**Proyecto 1**

**TRON**

**Nombre: Fabiola Alston Ramos**

**Carné: 2024227262**

**Link a Repositorio en GitHub:**

https://github.com/fabiola-alston/Tron.git

## Tabla de contenidos

[Tabla de contenidos 2](#_Toc177156529)

[Introducción 3](#_Toc177156530)

[Breve descripción del problema 4](#_Toc177156531)

[Descripción de la solución 5](#_Toc177156532)

[Diseño general 9](#_Toc177156533)

## Introducción

Este proyecto tiene como objetivo desarrollar un juego inspirado en *TRON*, en el cual los jugadores controlan motos futuristas que dejan una estela destructiva a su paso. A través de este proyecto, se busca poner en práctica el uso de estructuras de datos lineales para gestionar las diferentes mecánicas del juego, como el movimiento de las motos utilizando listas enlazadas, y la interacción de las motos con ítems y poderes del mapa, utilizando pilas y colas.

El juego se desarrolla en un mapa dividido en una cuadrícula donde las motos pueden moverse en cuatro direcciones diferentes. Los jugadores deben evitar las estelas que dejan tanto su propia moto como las de los oponentes, ya que chocar con una de ellas resultará en la destrucción de la moto. A medida que avanzan, los jugadores también podrán recoger ítems que aumentarán sus habilidades, como mayor velocidad, combustible o incluso escudos temporales.

Además de la participación del jugador, el juego cuenta con la presencia de bots (los cuales, en este caso, se denominaron “CPUs”) controlados por la computadora, que simulan otros jugadores y siguen reglas similares. Estos bots añaden un nivel de desafío extra, ya que sus movimientos son impredecibles y pueden competir por los mismos recursos en el mapa.

El proyecto no solo pone a prueba la capacidad técnica del estudiante desarrollador para implementar estructuras de datos y algoritmos eficientes, sino también su creatividad en el diseño de las mecánicas de juego y su capacidad para resolver problemas de forma original.

## Breve descripción del problema

El proyecto consiste en desarrollar el clásico juego de TRON utilizando restricciones para ciertas partes, tales como utilizar listas enlazadas y pilas y colas. Las posiciones de las motos y sus colisiones se deben registrar con listas enlazadas. Los ítems y poderes son agregados a una cola y una pila respectivamente, dependiendo del orden en el que sean colectados, y su orden para ser ejecutados está definido por el orden que tiene la estructura de dato lineal a la que pertenecen. Se debe encontrar una manera de implementar todas las estructuras de datos lineales solicitadas de manera que el juego logre cumplir todos los requisitos de manera exitosa.

Se trabajó en Unity Game Engine para la resolución de este proyecto.

## 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.**

Lo anterior se implementó utilizando un “grid lógico”, es decir, el grid no existe como tal, si no que los objetos (las motos), se mueven una cantidad específica de posiciones a la vez. Se mueven en incrementos de 0.25, y así, el “grid” se forma uniformemente. De esta manera, las listas enlazadas de las posiciones guardan únicamente las posiciones que se estén utilizando de las motos, lo cual se puede calcular con simple aritmética.

Se pudo haber implementado utilizando una matriz gigante con todas las posiciones, pero se optó por no seguir este camino.

La lista enlazada causó varias limitaciones y complicaciones a la hora de trabajar con las posiciones globales para todas las motos. Por ejemplo, las motos no detectaban las posiciones de otros carros por un simple error de una variable estática.

Otro problema que se detectó fue a la hora de programar la estela, al principio, fue muy complicado el proceso de dibujar la estela utilizando únicamente listas enlazadas. Al final, se lograron guardar correctamente todas las posiciones.

La estela fue implementada para cada jugador utilizando Prefabs en forma de cuadro del color de cada carro. Estos prefabs son instanciados y borrados a como seas necesario.

1. **Las motos tienen los siguientes atributos:** 
   1. **Velocidad: valor aleatorio entre 1 y 10 que determina qué tan rápido una moto se mueve.**
   2. **Tamaño de la estela: valor que determina el largo de la estela. Inicialmente vale 3.**
   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.**
   4. **Ítems: cola de elementos que afectan permanentemente la moto.**
   5. **Poderes: pila de poderes que afectan temporalmente la moto**

Las motos todas tienen un valor de *speed*, *tailLength*, *gas*, y *itemsQueue* y *powerUpStack*. Cada una de estas hace lo que se especifica en los requerimientos. Lo ideal hubiera sido hacer una clase para todos estos atributos, sin embargo hubieron complicaciones con el programa Unity al principio.

Un gran problema que se encontró con este método fue que a la hora de manejar los datos de cada instancia específica de un CPU / Player, se hacía mucho desorden, ya que sólo hay una instancia de Player, pero hay varias instancias de CPUs al mismo tiempo. Esto llevó a un problema común en la programación denominado “spaghetti code” en inglés.

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

Esta solución no se logró implementar.

1. **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.**

El orden que mantienen los ítems y poderes para ser ejecutados se manejan con dos tipos de estructuras de datos lineales solicitadas: colas y pilas respectivamente. Para los ítems, se utilizó una cola, la cual ejecuta los ítems de más viejo a más nuevo. Los que lleven más tiempo dentro de la cola son los primeros en ser ejecutados. En cambio, para los poderes, se utilizó una pila, la cual ejecuta el “top” de la pila, es decir, el poder más recientemente agregado.

La funcionalidad de botón, sin embargo, no se logró implementar.

1. **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.**

La funcionalidad de aplicarse con un segundo de delay y la del combustible no se logró implementar.

1. **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.**

Como mencioné en el requisito #1, las motos utilizan listas enlazadas para saber en todo momento todas las posiciones ocupadas en el mapa por una moto. En el momento que la posición actual de una moto cruce con la posición de otra, esto causa la muerte inmediata de la moto.

Las motos en ningún momento se detienen, puesto que hay un Coroutine de Unity constante que permite que se muevan de manera continua y uniforme, utilizando el método de trasladarse un “unit” (unidad creada para las motos de 0.25 pixeles), en cierta cantidad de tiempo.

El jugador puede cambiar la dirección en la que se dirige su moto utilizando las teclas del teclado, esto gracias a una función que constantemente revisa los valores de X y Y hacia la que se traslada la moto. Los bots se dirigen en direcciones aleatorias. En el momento que choquen con pared, cambian de dirección.

Ambos tipos de objeto mueren al chocar con la estela de otro jugador. El jugador muere si choca con una pared.

Se encontró un problema con las motos CPU grave. Por algún motivo, tienden a correr el método Death(), de morir, en momentos aleatorios. Dada la restricción del tiempo , no se logró identificar la causa de este problema, por lo tanto se mantiene en el juego final.

1. **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.**

En el caso de este proyecto, la estela inicial de todas las motos es de *10 posiciones*, ya que 3 posiciones sería muy poco para el tamaño del grid. La lista enlazada, por ende es de 11 nodos, sin embargo la lógica es tal y como se pide. Todo lo solicitado en este requisito lo he explicado en otros requisitos.

El tamaño “previamente definido” que se solicita para el proyecto es un tanto complicado de explicar en términos de Unity, ya que, para este caso, el tamaño definido es el de la cámara básica de Unity, el cual tiene un máximo de 10 pixeles desde el origen hacia izquierda y derecha y 5 pixeles desde el origen hacia arriba o abajo.

Se perdió muchísimo tiempo del proyecto por el tema de como crear el grid de manera eficiente en Unity, y se logró identificar el problema muy tarde. Por ende, muchísimas funcionalidades del proyecto quedaron incompletas.

1. **En la red aparece ítems y poderes aleatoriamente, que pueden recoger el jugador. Los ítems incluyen:** 
   1. **Celda de combustible: incrementa el combustible de la moto. Cada celda tiene una capacidad aleatoria.**
   2. **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.**
   3. **Bombas: cuando un jugador toma una bomba, explota.**

**Los poderes incluyen:**

1. **Escudo: permite que la moto se haga invencible por un tiempo variable. Afecta visualmente la moto.**
2. **Hiper velocidad: aumenta la velocidad de la moto en un valor aleatorio y por un periodo aleatorio. Afecta visualmente la moto.**

Los ítems y poderes logran aparecer de manera aleatoria en el mapa utilizando un generador de posiciones al azar, que estén redondeadas a las restricciones del grid (todo debe de estar dentro de espacios de 0.25 Unity units). Existe un Objeto Vacío de Unity que renderiza todas las posiciones aleatorias, y luego existen Prefabs para cada tipo de ítem y poder. Estos Prefabs contienen los scripts necesarios que le dicen a cada ítem/poder que debe de hacer al ser colectado.

Dada la restricción del tiempo, no se logró programar exitosamente la funcionalidad de cada ítem/poder, sin embargo, aparecen correctamente en el mapa y se detectan exitosamente (y distintamente) al ser colectados por algún jugador. Se logra detectar qué objeto fue colectado y por quien.

1. **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.**

Cada bot logra todo lo que logra el jugador principal. Tienen sus propios atributos de *speed*, *tailLength,* *combustible*, etc. El comportamiento de los bots, como fue mencionado anteriormente, es totalmente aleatorio, y en el juego se encuentran 4 bots: Naranja, Amarillo, Verde y Rosado.

1. **El juego se programará en C# con interfaz gráfica en Windows Forms/MAUI/Unity.**

BINARIA: Se programó en C# en Unity. Todos los scripts utilizan C#.

1. **El estudiante debe implementar todas las estructuras de datos requeridas en el proyecto.**

BINARIA: Se utilizaron listas enlazadas, pilas y colas.

1. **Se evaluarán buenas prácticas de programación en el código, las cuales es responsabilidad del estudiante investigar y aplicar, esto incluye patrones de diseño.**

CRITERIO DEL PROFESOR.

## Diseño general