
# Proyecto I : TRON

Escuela de Ingeniería en Computadores

Algoritmos y Estructuras de Datos I (CE-1103)

Nombre: Tamara Vanessa Cajiao Molina

Carnet: 2024143333

Profesor: Leonardo Araya Martinez

II Semestre 2024

Tabla de Contenidos

**Introducción**

En este proyecto se nos invita a tomar en práctica las estructuras de datos lineales aprendidas a lo largo del primer trayecto del curso por medio de la recreación del juego clásico “Tron”. El cual consiste en la manipulación de una motocicleta que deja una estela destructiva a su paso, con el objetivo de eliminar a los enemigos, que son otras motocicletas con estelas destructivas. Sin embargo, este proyecto nos pide implementar cosas adicionales como ítems y poderes que afectan a la moto, por ejemplo combustible que si se acaba, la moto muere, hipervelocidad, entre otros.

La complejidad del proyecto radica en la implementación de un sistema que simula de manera correcta las reglas del juego. Las estelas destructivas de las motos estarán modeladas mediante listas enlazadas simples, mientras que los ítems recogidos durante el juego son manejados por una cola y los poderes por una pila. Además, se busca fomentar la creatividad en la resolución de problemas mediante el uso de patrones de diseño, diseño de la interfaz gráfica y diagramas UML.

**Descripción del Problema**

El problema del proyecto gira en torno a la implementación eficiente de diversas estructuras de datos lineales para simular el comportamiento del juego. En primer lugar, cada estela de luz destructiva está representada mediante una lista enlazada simple, donde cada nodo de la lista corresponde a una parte de la estela destructiva que la moto va dejando conforme avanza. La estela crece y debe actualizarse dinámicamente, eliminando nodos y añadiendo nuevos nodos conforme la moto se desplaza.

Además, se implementa una cola para gestionar los ítems que la moto recoge durante el juego. Cada vez que un ítem es recogido, se inserta en la cola y se aplica automáticamente en el orden en que fue adquirido. Esto asegura que los ítems afecten a la moto de manera secuencial, garantiza un manejo ordenado de los ítems y permite una gestión eficiente de los elementos en tiempo de ejecución.

Por otro lado, los poderes que afectan temporalmente a las motos se manejan mediante una pila, donde el jugador puede decidir el orden en que los poderes se aplican. La naturaleza de LIFO (Last In, First Out) de la pila permite al jugador reorganizar los poderes según la estrategia que desee implementar en cada momento del juego. Esta estructura facilita el control sobre el uso de los poderes, ya que el jugador siempre puede activar el poder que esté en la cima de la pila, lo que añade una capa de toma de decisiones estratégicas en tiempo real.

**Descripción de la Solución**

**Requerimiento 001 - Moto de Luz y estela destructiva**

La moto de luz es un sprite con su respectiva imagen y varios scripts adjuntos a ella, principalmente “Moto Script” el cual tiene varias funciones. Primero en cada frame se asegura de actualizar la posición continua de la moto, y su rotación dependiendo en la tecla que presione el jugador en el teclado, esto lo hace por medio del manejo de vectores. Además se asegura de disminuir el combustible con respecto al movimiento de la moto, con el combustible con un valor máximo de 10, y el costo del movimiento es de 2 por el tiempo transcurrido, ya que la moto nunca para de moverse. Por último, pero más importante, se encarga del manejo de la estela de luz destructiva, tanto en la interfaz de juego como en la estructura de datos lineal, la cual para este caso en particular, es una lista enlazada simple.

La lista enlazada es un atributo de la clase “Moto Script” creada en otro script la cual contiene como atributos instancias de otro script llamado “Nodo”, esto con el objetivo de manejar de forma eficiente y lógica una lista simplemente enlazada. Se utiliza una instancia de esta lista simplemente enlazada como atributo de la moto del jugador, que se usa para manejar la estela destructiva. Para esto se utiliza un contador y un atributo que maneja el valor inicial de la estela (el cual es 3), el contador es principalmente para limitar la creacion y eliminacion de los cuadrados de color que forman la estela visual e internamente. Se maneja de forma que si el contador es igual al valor de la estela, entonces va a eliminar el primer valor insertado a la lista enlazada, disminuye en 1 el contador, y seguidamente crea otro cuadrado brillante, de este modo se simula de forma exitosa que la estela en una línea continua que sigue a la moto en su recorrido.

Debido a que el requerimiento en este caso indica explícitamente que la estela se debe implementar como una lista enlazada simple, no hubo alternativas consideradas al respecto. Aprovechando las opciones de unity, se consideró y se aplicó la implementación del movimiento de la moto a través del espacio tridimensional brindado por unity; de esta forma, se hace más sencillo revisar la posición de la moto en todo momento a la hora de implementar la estela. Gracias a la herramienta utilizada para la implementación del proyecto, no hubo muchas limitaciones además del escaso conocimiento y manejo de la herramienta por parte de mi persona.

Los problemas encontrados a lo largo de la ejecución de este requerimiento fueron principalmente aspectos relacionados a confusiones con el posicionamiento de la moto en el espacio tridimensional, el instanciamiento de los datos en la lista enlazada de la estela y la construcción de esta lista.

**Requerimiento 002 - Atributos de la Moto**

La moto del jugador tiene varios atributos, entre ellos se encuentran los exigidos en el proyecto, como la velocidad, manejada con números enteros, con un valor inicial de 1. El valor de la estela, el cual también es manejado por números enteros con un valor inicial de 3. También el atributo del combustible, el cual se maneja con números flotantes, decreciendo conforme avanza la moto en el espacio, comienza con un valor inicial y máximo de 100. Items es un atributo el cual es una instancia de una cola que maneja datos de forma que siempre “saca” el primer elemento insertado en la cola cuando se requiere. Al igual que los Poderes, es una pila en la cual se insertan los poderes que recoge el jugador a lo largo del juego. Estos últimos dos atributos fueron los que tuvieron más dificultad al implementarse, pues es a partir del manejo de estructuras de datos.

Con respecto a estos últimos dos atributos, estos son instancias de clases de una cola y una pila respectivamente, fueron implementadas del siguiente modo: Contienen atributos que son instancias de la clase Nodo el cual se encarga de contener el dato a insertar, en la correspondiente pila o cola. Estos atributos se encargan de mantener control de los datos insertados en la estructura de datos; en el caso de la pila, solamente se le da seguimiento al último dato insertado, esto por su naturaleza LIFO (Last in First Out) como estructura de datos la cual permite únicamente al último dato insertado en la pila, ser el primero en salir de ella, y así sucesivamente hasta que la pila no tenga mas datos. Por otro lado, la cola si mantiene registro del primer dato insertado en ella y al igual que la pila, le da seguimiento al último dato insertado en ella. Esto porque, al ser de naturaleza FIFO (First in First Out), necesita saber cuál fue el primer elemento insertado en la cola para, cuando se le indique por medio del método correspondiente, poder sacar el primer elemento insertado en la cola.

Un problema encontrado al implementar estas estructuras de datos es, que al insertar elementos en estos, los datos no eran visibles de forma simple, pues había que también implementar un método que mostrara en la consola los elementos que se encontraban en la estructura de datos en el momento. Otro problema al implementar las pilas y colas, fue que el tipo de dato que se insertaba en el nodo, ya que al comienzo se inició con un dato tipo “T” y se insertaban las instancias gráficas de los ítems y poderes a las pilas y colas; sin embargo, esto traía un problema, q

**Requerimiento 004 - Pila de Poderes - Selección de poderes**

Como ya se mencionó anteriormente, la pila de poderes fue creada a partir de una pila creada desde cero, con sus respectivos atributos y métodos de estructura de datos correspondiente. Sus atributos son instancias de la clase Nodo, se encargan de mantener el registro del primer y último elementos insertados en la pila, pues la clase Nodo se dedica a mantener el dato guardado y al tenerlo como instancia, esta nos permite manejar el dato como sea necesario. Se inserta un dato de tipo string en la pila de poderes si la moto colisiona con un “coleccionable” de tipo podes como lo es la hiper velocidad o el escudo. Una vez insertadas en la pila, el jugador puede decidir cambiar el orden de los poderes una única vez por medio del teclado.

En cuanto a la hipervelocidad, una vez obtenida por el jugador, esta determina un valor aleatorio el cual se le va a aumentar a la velocidad actual. La velocidad inicial de la moto del jugador tiene un valor de 1, la hipervelocidad decide un valor entero aleatorio entre 1 y 10 y la moto acelera y se mueve a dicha velocidad por una cantidad de tiempo predeterminada. Con respecto al escudo, este tiene el propósito de hacer “invencible” al jugador por un tiempo limitado, haciendo que al colisionar con los bordes del mapa, la moto no se destruya, al igual que cuando colisiona con una bomba, con los bots o sus estelas de luz.

El principal problema encontrado en esta sección, fue la forma de implementar el cambio de orden en la pila de poderes, pues al presionar la respectiva tecla del teclado para cambiar el orden, si se encontraba un poder de hipervelocidad, esta a la segunda vez de cambiar orden, cambiaba el poder a invencibilidad. Una alternativa considerada para este problema fue implementar los poderes de forma instantánea al momento en el que el jugador los recoge, de este modo el jugador está decidiendo cual implementar, al momento de recolectarlo, sin tiempo de espera.

**Requerimiento 005 - Cola de Items**

La cola de ítems se implementa de forma muy similar a la pila de poderes, esta ingresa elementos a la cola en el momento en el que el jugador colisiona con una celda que contenga un objeto de tipo item previamente determinado en la creación del mapa. Al insertarse, la cola mantiene el registro del primer elemento insertado y al sacarlo de la cola, al implementar el ítem de forma inmediata, esta saca el primer elemento insertado y asigna como primer elemento al siguiente del previamente expulsado.

**Requerimiento 006 - Destrucción de Moto**

La implementación de la destrucción de la moto del jugador fue implementada mediante las herramientas que Unity nos facilita, mediante el “Box collider 2D”, este es la clave para que el sprite de la moto del jugador detecte colisiones con cualquier otro “collider”. Para ello, se le asignó tambien “Box collider 2D” a los bordes del mapa, de este modo, mediante la función “OnTriggerEnter2D” en el script “Collisionn”, si la moto del jugador se topa con algún borde del mapa, este sabe que debe destruirse inmediatamente.

Las colisiones para detectar si la moto recolecta algún “coleccionable” se implementa de forma similar, de modo que cada celda contiene su propio “Circle collider 2D” de modo que si la moto entra en el espacio de este, deba revisar si la celda a la que pertenece ese collider contiene algún “hijo” y si lo hace, debe de revisar si es un ítem o un poder y agregarlo a su respectiva pila o cola.

Y por último, se realiza de igual forma para las colisiones de los bots y sus estelas, cada instancia de “cubo” (pedazo) de estela contiene una “Box Collider 2D” al igual que la moto del bot como tal. El “collider” de la moto del bot detecta colisiones con los bordes del mapa al igual que la moto del jugador, del mismo modo detecta colisión con la propia moto del jugador, y en tal caso, ambos se destruyen. En cuanto a las estelas de los bots y el jugador, estas funcionan del mismo modo, el jugador y los bots detectan si colisionaron con un “Box Collider 2D” y actúan como corresponde. Al momento de destrucción de cualquiera de estos, la moto se destruye junto con su estela de luz.

**Requerimiento 007 - Implementación del Mapa**

El mapa está formado por un conjunto de 81 celdas, las cuales juntas forman la red del mapa por la cual el jugador y los bots se deberían de desplazar. Cada una de estas celdas tiene sus celdas vecinas definidas, solamente en direcciones arriba, abajo, derecha e izquierda. Los ítems y poderes se generan al comienzo del juego y sus instancias se generan como hijas de una celda aleatoria del grid del mapa. La moto al detectar colisiones, detecta si se encuentra con el área de colisión de cada celda, revisa si la celda tiene un hijo y si lo tiene revisa el nombre y determina si colisionó con un ítem o con un poder y lo agrega a la estructura de datos disponible.

El principal problema que fue encontrado durante el desarrollo de este requerimiento, fue el movimiento de la moto sobre el grid del mapa. La alternativa aplicada para este problema fue recurrir a las herramientas brindadas por unity y ser capaz de mover la moto del jugador por encima del espacio tridimensional que unity sugiere al implementar el movimiento de un sprite u objeto.

**Requerimiento 008 - Ítems y Poderes**

Como ya fue mencionado anteriormente, los ítems y los poderes fueron implementados de la forma en que se sugería en los requerimientos, en tipo de estructuras de datos de colas y pilas respectivamente. En cuanto a los elementos que a lo largo del juego se pueden insertar en las estructuras de datos, se clasifican en elementos que se aplican a la moto de forma inmediata como los ítems: La bomba, la cual hace al jugador destruirse al instante en caso de colisión con esta. El crecimiento de estela, la cual al momento de colisión con este ítem se aumenta el valor de la estela de luz (inicialmente 3) en un valor aleatorio del 3 al 10, y crece de acuerdo al valor generado. El aumento de combustible: Este funciona de manera similar al crecimiento de la estela, al colisionar con este, se aumenta en un valor de 10 el combustible de la moto del jugador, el cual tiene un valor inicial de 100 y va disminuyendo conforme a la distancia recorrida por la moto.

Además en las pilas se insertan los poderes: La Hiper Velocidad, la cual permite a la moto del jugador aumentar la velocidad en un valor aleatorio por un tiempo aleatorio; y el escudo, el cual debería darle cierta “invencibilidad” a la moto del jugador por cierta cantidad predeterminada de tiempo.

**Requerimiento 009 - Implementación de Bots**

La clase MotoBot hereda de la clase MotoScript pues los bots tienen un funcionamiento muy similar al del jugador y se pueden “reutilizar” algunos de los funcionamientos y métodos de la clase MotoScript del jugador. Así como unos métodos se pueden “reutilizar” otros se deben modificar para que funcionen acorde a lo que se requiere, por ejemplo el movimiento automático de los bots. A diferencia del jugador, los bots no se mueven con dirección especificada por el jugador por medio de flechas, estos se mueven aleatoriamente, iniciando a partir de una posición previamente determinada. Un problema que se desarrolló en este proceso de movimiento automático, es que las motos únicamente realizan este cambio de dirección aleatorio una sola vez, cuando deberían de decidir si cambiar de posición cada tres segundos.

Así como la clase principal de los bots hereda de la clase principal del jugador, la clase de colisión de los bots también hereda de la clase de colisión del jugador, de este modo los bots son capaces determinar si colisionan con un borde del mapa y morir automáticamente (y borrar su respectiva estela) o si colisionan con la celda en la que se encuentra un ítem o un poder. En ese caso, a diferencia del jugador, los bots no podrían recolectarlos, por lo que si colisionan con estos, no hacen nada.

Diagrama de clases UML: