https://github.com/IVoidi/tron

Proyecto 1 Tron

Rodrigo Arce Bastos

Leonardo Araya Martinez

Abstract—Tron es un juego multijugador de carreras de motos de luz. La idea es implementar distintas estructuras de datos lineales y programación orientada a objetos, haciendo uso de Unity y C. Se logró entender el uso de estructuras de datos como pilas y colas, así como listas enlazadas circulares con enlaces múltiples. Se elaboró un diagrama UML del proyecto, para mejor entendimiento del sistema.

Keywords— *Estructuras de datos lineales, programación orientada a objetos, C#, Unity*

Contents

1 Introduccion **1**

2 Descripcion del problema **1**

3 Solucion **1**

3.1 Descripcion general de la solucion 1

3.2 Retos del programa 1

4 Diseño general **2**

References **2**

1. Introduccion

Tron es un juego de carreras de motos de luz. La idea era programar este juego con estructuras de datos lineales como listas enlazadas, y utilizando programación orientada a objetos. El juego sigue un modelo parecido al clásico juego de la Serpiente, donde la motocicleta solo puede moverse en 4 direcciones posibles y esta deja una estela destructiva a su paso que puede crecer. Este juego trae distintos y objetos y poderes que dan un empuje al jugador, como super velocidad, escudos, bombas destructivas, aumento de gasolina y aumento de estela. El juego, a diferencia de la Serpiente, es multijugador. Hay cuatro robots simultáneos jugando con el jugador.

2. Descripcion del problema

A continuación, se van a listar los requerimientos del juego. Empecemos por el mapa. Se nos dice que el mapa debe ser una lista enlazada de nodos, los cuales tienen 4 enlaces: arriba, abajo, izquierda y derecha. Estas, a su vez, son las únicas direcciones posibles en las que se puede mover la motocicleta futurista (por ejemplo, no puede realizar un movimiento diagonal). A su vez, la motocicleta también debe implementarse como una lista enlazada, donde la cabeza es la motocicleta en sí y el resto de elementos son las partes de la estela. Hablando un poco más del mapa, se requiere que estos tengan algunos items y poderes repartidos a lo largo del mapa en posiciones aleatorias, y estos dan poderes como hiper velocidad y escudo, y hay algunos items que dan ventajas como aumentar en uno el tamaño de estela, la gasolina y la capacidad de posicionar bombas en el mapa. Se menciona que, al ser destruida una motocicleta, esta debe esparcir sus items en posiciones aleatorias del mapa. Por último, se menciona que es un juego multijugador donde, aparte del jugador, hay 4 robots simultaneos con las mismas características.

3. Solucion

3.1. Descripcion general de la solucion

Para implementar el mapa, se decidió realizar un espacio toroidal. Esto significa que cuando una motocicleta toca un borde, esta aparece del otro lado del mapa. En código, esto significa que los nodos que están a los bordes tienen las referencias al otro lado del mapa. Por ejemplo, si hay un nodo en (0, 0), su referencia a la derecha seria (1, 0), su referencia a la izquierda sería (30, 0), su referencia a arriba sería (0, 1) y su referencia hacia abajo sería (0, 12). Se decidió implementar la lista enlazada como una matriz, para así poder acceder al sistema de coordenadas de manera inmediata, y no tener que iterar la lista enlazada entera hasta encontrar una coordenada. La clase nodo contiene algunos parametros adicionales que hacen que este sea el objeto más importante del juego: esObstaculo, esCabeza, item, poder. Esto nos permite saber la naturaleza del nodo. Item y poder toman el valor de null inicialmente, y luego si algún objeto cae en ese nodo, estos toman un valor Poder o Item, dependiendo de qué es. A su vez, si en el nodo actual se encuentra la cabeza, esCabeza se vuelve verdadero, lo mismo pasa con la estela, si es verdadero, el valor de esObstaculo va a tomar verdadero. El valor de esObstaculo es multiuso, ya que también toma el valor de verdadero con la explosión de la bomba.

Hablemos del objeto Motocicleta. Este objeto es una lista enlazada de nodos, donde la cabeza es un nodo solitario y la estela es una lista enlazada de nodos, los cuales toman el valor de esObstaculo como verdadero, y se van actualizando con las conexiones de nodos adyacentes. Cada objeto motocicleta comparten el mismo espacio, osea, en cada frame el mismo espacio se va actualizando y cada nodo va tomando un valor diferente. Esto significa que todas las colisiones se hacen mediante este espacio y no utilizando los objetos de Unity. A su vez, cada motocicleta tiene un vector direccion, el cual va a definir el siguiente movimiento del jugador, ya que este no se detiene. Cada motocicleta tiene su velocidad (1 - 10) y su combustible (0 - 100) que si llega a 0, el jugador se autodestruye. El jugador tiene un contador de cuántos items y poderes tiene, individualmente. Los items funcionan como una cola y los poderes funcionan como una pila, las cuales se implementaron con listas enlazadas para mayor facilidad. La motocicleta tiene una propiedad esJugador la cual toma el valor de verdadero si es la motocicleta del jugador. Si no, toma falso. Esto activaría las funciones de esquive y movimiento automático de las motocicletas de bots. Los bots están programados para utilizar los items apenas los recogen. A su vez, cada bot va a definir su movimiento en base a la ubicacion de algunos elementos, osea, van a dirigirse a un elemento o item asignado. En el mapa, hay 5 elementos iniciales y que simplemente cambian su ubicación aleatoriamente cuando alguna motocicleta los recoge. Eso si, cuando se destruye una motocicleta con items, estos se agregan en el mapa, osea, pueden haber 2 o 3 de cada dependiendo de la motocicleta, solo que estos nuevos items se derstruyen al tocarlos. Hablando de los items, esta es una cola prioritaria, lo que significa que los items de gasolina van a priorizarse de primeros. En el caso de la pila, estos poderes se pueden cambiar dependiendo de la necesidad del usuario. Inicialmente se prioriza la velocidad.

3.2. Retos del programa

El mayor reto fue implementar el mapa como una lista enlazada. Al ser una lista con enlaces cuadruples, y además, circular, era difícil hacerlo únicamente con un nodo central. Así que se optó por hacerlo en una matriz, ya que de esta manera era mucho más sencillo acceder directamente a un nodo utilizando la posición del jugador, sino, se tendría que haber recorrido uno por uno los nodos hasta encontrar el nodo requerido.

Convertirlo en una matriz también facilitó la implementación del sistema de coordenadas de unity con el del espacio. Para esto, se crearon funciones que permitieran utilizar las funciones de unity

como indices directamente. Como unity toma coordenadas negativas, entonces era totalmente necesaria esta funcion.

Aún con todos estos arreglos, las coordenadas en el eje y estaban "volteadas", esto es porque, como es una matriz, entonces la parte de arriba son los valores menores y conforme va "bajando" en la matriz, estos valores van aumentando. Para solucionar esto, se soltearon los indices a la hora de agregar los nodos. Ya con estos errores solucionados, se pudieron implementar las colisiones y los items de manera sencilla y eficaz.

4. Diseño general

A continuación, se presenta el diseño general del programa, en UML. Los diagramas se encuentran en la última página.

References

[1] *PGFPlots - A LaTeX package to create plots.* [Online]. Available: https://pgfplots.sourceforge.net/.

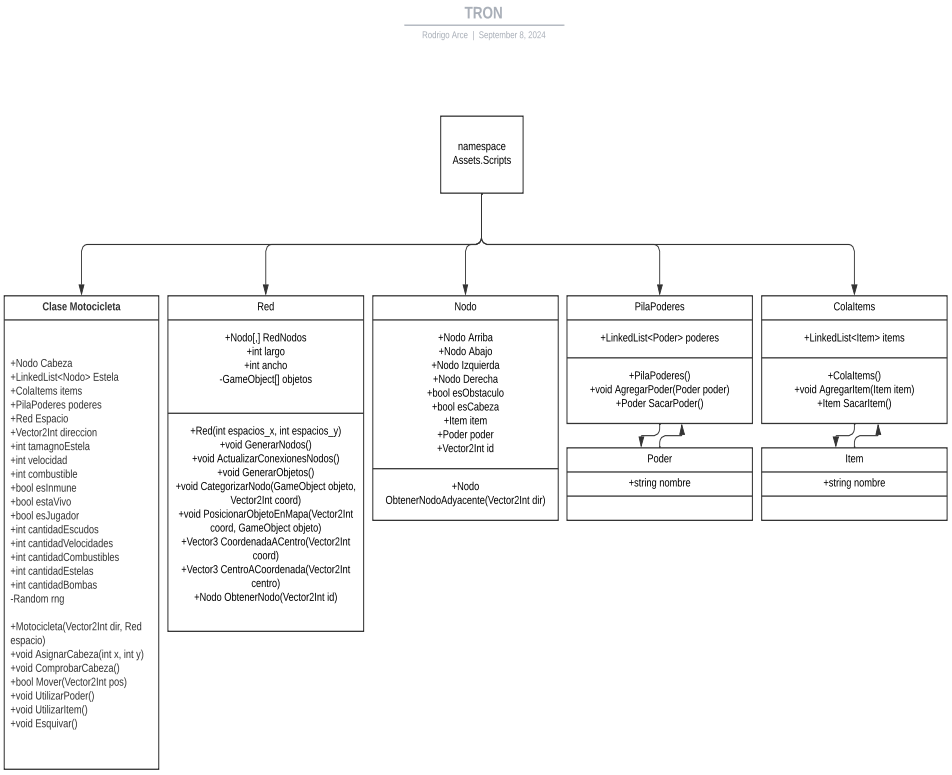

Figure 1. Funcionamiento interno del TRON. Obtenido con [1].

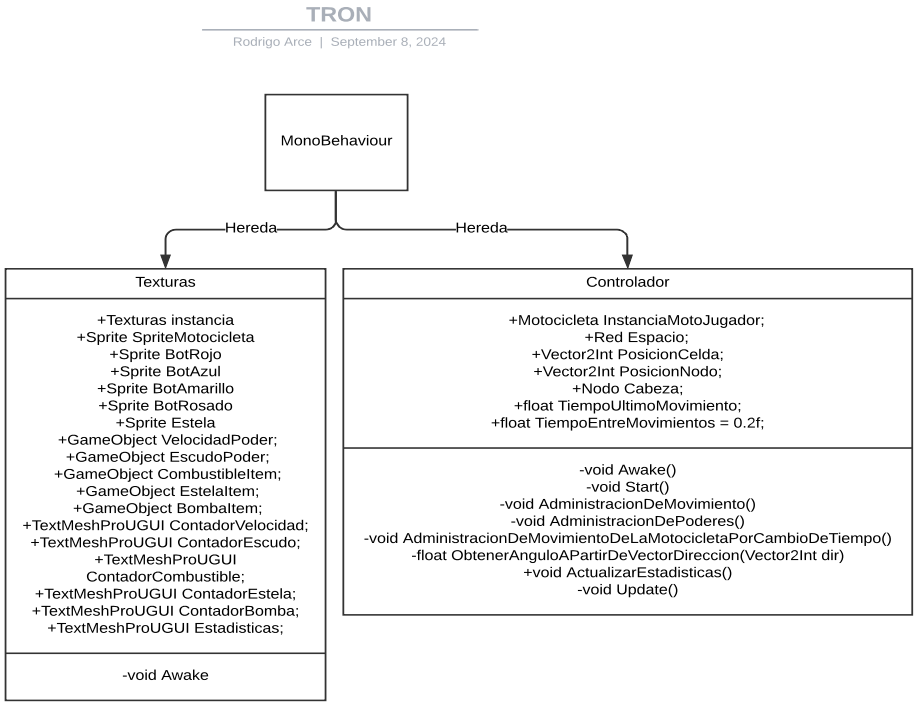

Figure 2. Funcionamiento de Unity. Obtenido con [1].