Tecnológico de Costa Rica

Algoritmos y Estructuras de Datos I

Proyecto 1: Tron

Profesor: Leonardo Araya Martínez

Estudiante: Luis Castro Montenegro

Carné: 203083843

Fecha: 08/09/2024

Tabla de Contenidos

1. Introducción

El presente documento corresponde a la documentación del primer proyecto programado del curso Algoritmos y Estructuras de Datos I, el cual tiene como objetivo general implementar una solución a un problema mediante la utilización de estructuras de datos lineales. Se busca que el estudiante entienda e implemente listas, pilas y colas para que mediante el desarrollo de algoritmos pueda dar una solución al problema planteado. Todas las estructuras deben ser implementadas por el estudiante por lo que se debe contar con mecanismos para su manipulación. También se busca introducir conceptos de diagramas de UML y la aplicación de patrones de diseño para modelar una solución.

2. Descripción del Problema

Mediante la implementación de estructuras de datos lineales se busca que el estudiante diseñe e implemente un juego inspirado en Tron, el cual se basa en la carrera de motos de luz. Se deberá tener un mapa, similar a una matriz, en donde las motos podrán navegar en 4 posibles direcciones. El jugador controla una de las motos, las cuales dejan una estela destructiva de luz a su paso que se comporta como la cola de una serpiente en el juego Snake. Se deberá contar con otras motos las cuales serán controladas por el programa, al cruzar una estele dejada por otra o ella misma se deberá de destruir.

A continuación, se presentan los requerimientos definidos para la elaboración del proyecto.

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.

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.
    - 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.
    - Ítems: cola de elementos que afectan permanentemente a la moto.
    - Poderes: pila de poderes que afectan temporalmente a la moto.

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

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.

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.

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.

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.

8. 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 estala: 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 de tiempo aleatorio. Afecta visualmente la moto

9. 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.

10. El juego se programará en C# con interfaz gráfica en Windows Forms/MAUI/Unity, abierto a otras opciones.

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

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

3. Descripción de la Solución

En esta sección se explicará cómo se implementaron los distintos requerimientos especificados en la sección Descripción del Problema. En forma general el proyecto se desarrolló en Visual Studio 2022 utilizando el lenguaje de programación C#. Para la creación de la interfaz se eligió una opción diferente a las propuestas en los requerimientos, en este caso seleccionó el framework MonoGame el cual permite desarrollar juegos mediante código. Se utilizó GitHub como manejador de código, el cual esta disponible en el siguiente enlace: https://github.com/cluis11/Tron.git.

Se utilizará la misma numeración de cada requerimiento descrito en la sección anterior para indicar como se implementó, alternativas consideradas, limitaciones, problemas encontrados y cualquier otro aspecto que se considere relevante.

1. Para este requerimiento se crearon dos clases, una llamada *PlayerNode* la cual funciona como el nodo de la lista enlazada *Player*. Los nodos contienen una referencia al siguiente y un *MapNode*, otra clase creada para funcionar como el nodo del mapa del requerimiento 7, el cual permite asignar la cabeza del jugador o sus estelas a una posición en el mapa.

   Al iniciar el juego la lista enlazada *Player* solo muestra la cabeza, pero conforme hace los primeros movimientos se van creado los *PlayerNode* asociados a las estelas hasta que tenga 3, es decir la lista en total tiene 4 nodos incluyendo la cabeza. En la clase *Player* existe un método para agregar más estelas según sea necesario, así otros que serán descritos en otros requerimientos.

   Se implemento el patrón de diseño de método de fábrica para crear la instancia del jugador ya que este se define en el mapa, pero por como MonoGame funciona aun no se tienen todos los atributos necesarios, por lo que se utilizó un método constructor vacío y una función estática a nivel de clase que retorna una instancia de esta para poder crear al jugador en el momento que se tienen todos los atributos necesarios para su creación.

   Para poder dibujar un elemento en la interfaz gráfica de MonoGame se necesita de una textura, la cual es básicamente la imagen que se quiere dibujar, por esta razón se creo una clase llamada *Sprite* la cual tiene como atributo una textura y la posición donde se va a dibujar. La clase *PlayerNode* hereda de esta clase para contar con estos elementos. La razón de crear una clase por aparte fue para encapsular estos atributos y una función que asegura que la imagen se muestre con las dimensiones correctas para todos los elementos que se dibujaran en el mapa, esto se detallará más en el requerimiento 7 en esta sección.

2. Las motos, o en este caso la clase *Player*, cuenta con un atributo de velocidad, tamaño de la estela, combustible, ítems y poderes los cuales se detallarán a continuación:

   - Velocidad: cada jugador en el mapa tiene una velocidad distinta que se asigna de forma aleatoria, esta corresponde al tiempo en el que el programa actualiza la posición del jugador. Se cuenta con un arreglo de 10 elementos, que contiene los distintos tiempos de

actualización, mediante el uso de la clase *random* se genera un entero entre 0 y 9 que funciona como un índice en el arreglo de velocidades para asignarle un valor al jugador.

Originalmente se contemplo implementar la velocidad como número de espacios que el jugador se movía en un turno, sin embargo, este enfoque dificultaba la capacidad de obtener objetos en el mapa o evadir los bordes.

- Tamaño de la estela: mediante un entero de valor 3 que se asigna en el constructor de la clase se determina el valor inicial del tamaño de la estela. En cada movimiento que realiza el jugador se llama a una función que valida si el valor de la variable estelas es diferente de 0, en caso de serlo crea un nuevo nodo lo coloca al final de la lista y asigna el *MapNode* de la posición anterior del último elemento de la lista enlazada y resta 1 al valor de la variable estelas.

- Combustible: Se tiene una variable llamada fuel la cual al inicializar al jugador se le asigna 100 como valor, este se reduce en una unidad por cada 5 movimientos que la moto realiza por lo que se utiliza otra variable que sirve como contador. A esta última se le suma 1 por cada movimiento que el jugador realiza y tras completar cada movimiento se llama a una función que valida si el contador ya llego a 5, en caso de que si se reduce 1 a la variable fuel y se reinicia el contador. Si fuel llega a 0 se llama a la función encargada de hacer a la moto explotar.

- Ítems: Se implementó una cola de prioridad mediante listas enlazadas, en donde el atributo prioridad va de mayor a menor, para que cada vez que el jugador se mueve a un nodo del mapa que contiene un ítem este se agrega a la cola, eliminándolo del mapa. Los ítems se detallarán más en el requerimiento 8.

- Poderes: Se implementó una pila, mediante listas enlazadas, la cual agrega un elemento al tope cada vez que el jugador avanza a un *MapNode* que contiene un poder, eliminándolo del mapa. En las pilas no se puede reorganizar el orden de sus elementos, pero el jugador si es capaz de elegir que poder aplicar en caso de tener más de uno. Para lograr este comportamiento si el jugador quiere aplicar un poder que no sea el tope crea una nueva pila donde el poder seleccionado se encuentra en el tope. Los poderes se detallarán más en el requerimiento 8.

3. Al momento que una moto se destruye la clase *Map* que es la lista enlazada de *MapNode* antes de asignar al jugador o enemigos como nulos, para que ya no se muestren en el mapa, accede a la pila de poderes y cola de ítems. Los va sacando uno a uno y mediante la clase *Random* se genera un valor para columna y fila donde se van a asignar estos ítems y poderes a un *MapNode* para que regresen al mapa. *Map* cuenta con un método booleano que le permite validar si un nodo en una fila y columna específica ya tiene un elemento asignado por lo que este método usa un bucle hasta que los números generados correspondan a un nodo vacío.

4. Los poderes de un jugador se muestran en la interfaz gráfica en el mismo orden de la pila de izquierda a derecha, con las teclas Q y E el jugador se puede mover en estas mismas direcciones para seleccionar el poder que quiere usar presionando espacio. Internamente lo que se hace para crear la nueva pila es agregar los elementos a una cola de prioridad, donde a cada elemento que no es el seleccionado se le da una prioridad mayor y al seleccionado se le da la prioridad menor. Una vez hecho esto se crea una nueva pila agregando el primer elemento de la cola hasta que queda vació, el poder seleccionado siempre será el último elemento en la cola y por ende el primero en la pila. El jugador no se da cuenta de esto el solo selecciona el poder que quiere aplicar ya que visualmente no se reorganiza la pila solo se elimina el elemento seleccionado y se acomodan los demás en el orden correspondiente.

5. En la clase *Player* se cuenta con una función que se ejecuta tras transcurrir 1 segundo la cual valida si hay elementos en la cola de ítems y elimina el primer elemento de esta, almacenándolo en una variable. Una vez que ya no está en la cola se utiliza el método del ítem para aplicar su efecto. Los ítems se detallarán más en el requerimiento 8.

6. En MonoGame existe un método llamado Update el cual se encarga de la lógica del juego, la clase *Player* tiene su propio método de igual nombre el cual se llama desde la clase *Map* la cual contiene la instancia de *Player*. Este método se llama por cada frame que transcurre en el juego. Dentro del método del jugador para manejar la lógica se cuenta con una función que se encarga de determinar la dirección del jugador según se presionen las flechas del teclado. Se cuenta con otra función que se ejecuta únicamente cuando haya transcurrido el tiempo asignado en la variable asociada a la velocidad la cual mueve al jugador constantemente en el mapa con la dirección asignada.

7. Para poder crear el mapa mediante una lista enlazada se utilizaron dos clases llamadas *MapNode* y *Map*. La primera corresponde al nodo el cual cuenta con una referencia a cuatro variables de tipo *MapNode* para cada una de las 4 posibles direcciones, también se le asigna un valor para la fila y columna que le corresponde en el mapa, asimilando a una matriz, y una variable de tipo *Sprite* que corresponde a los diferentes elementos que se mostraran en el mapa. Los ítems, poderes y nodos del jugador heredan de *Sprite* para facilitar el almacenamiento, y dibujado, de los elementos en el mapa.

   La clase *Map* se define con un número fijo de filas y columnas que corresponde a la cantidad de nodos que tendrá de izquierda a derecha y de arriba hacia abajo, además de una variable que referencia al nodo en la posición (0,0). Para la creación de los nodos se utiliza un for donde se itera según el número de filas y columnas. Se utilizo una variable para referenciar el primer elemento de cada fila, el elemento actual, el elemento a la izquierda del actual y el elemento arriba del actual. Lo anterior para poder interconectar todos los nodos como una matriz.

   Al inició del juego se crean ítems y poderes los cuales se asignan a un nodo del mapa generando números aleatorios para la fila y columna y validando que dicho nodo esta libre. Lo mismo

pasa para el primer nodo de la lista enlazada del jugador, el cual mediante el uso de las flechas se mueve entre los nodos del mapa y se le cambia su posición según las variables de fila y columna de dicho nodo.

8.

9. Para la creación de los bots, o enemigos, se creó una clase llamada *Enemy* la cual hereda de *Player* por lo que cuenta con todas sus funcionalidades. Se tuvo que sobrescribir los métodos para actualizar su posición y manejar la dirección de forma que no requieran entradas del usuario para variar su movimiento y evadir obstáculos como bombas, otros enemigos o el jugador y sus estelas.

10. Para la creación del juego no se seleccionó ninguna de las opciones propuestas ya que existía la opción de elegir otra si así lo deseaba el estudiante. En mi caso se selecciono MonoGame por ser un framework para la creación de videojuegos el cual para los requerimientos de este proyecto significaba una menor curva de aprendizaje y mayor simplicidad en el desarrollo, permitiéndome enfocarme en el uso de las estructuras de datos lineales.

    MonoGame cuenta con una clase llamada *Game1* la cual cuanta con 4 métodos para el funcionamiento del juego. El primero se llama Initialize donde el desarrollador debería de instanciar las diferentes entidades que utilizara en el juego. El método LoadContent que es para cargar todo lo relacionado a los elementos visuales que se mostraran en la interfaz, cuenta con un gestor de contenido que permite agregar imágenes, sonidos y fuentes al proyecto. El método Draw es el encargo de dibujar los distintos elementos en la interfaz, se ejecuta cada frame. Finalmente, se tiene el método Update que se encarga de la lógica del juego, lo recomendado es no agregar ninguna lógica fuera de este método.

11. En el proyecto se implementó una lista enlazada simple para el jugador y los enemigos, una lista enlazada para la creación del mapa mediante nodos, una pila para los poderes del jugador y una cola para los ítems que el jugador recoge. Cada una de estas fue implementada por el estudiante para cumplir con los requerimientos del proyecto basando en el contenido del curso.

12. Como parte de la implementación se utilizó la herencia y el método de fábrica.

4. Diseño General

A continuación se presenta el diseño general de la solución implementada mediante un diagrama de UML.