Proyecto 1 TRON

Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

Algoritmos y Estructuras de Datos II (CE 1103)

II Semestre 2024

Profesor: Leonardo Andrés Araya Martínez

Estudiante: Saddy Guzman Rojas

Carnet: 2023088184

Repositorio Git:
https://github.com/SaddyGR3/ProyectoTron6.git

TEC | Tecnológico de Costa Rica

Índice

1.Introducción 2

2. Tron:Legacy 2

3. Las Motos de luz 3

3.1 Pero porque compiten? 3

3.2 En que consiste las carreras de motos de luz? 3

4. Proyecto Tron 4

4.1 Inicio del proyecto 4

4.2 Desarrollo de la Matriz 5

5.Clase Nodo y Matriz..... 6

6. Clase Moto y Clases hija Jugador y Enemigos..... 7

6.1. Movimiento de la moto..... 8

6.2. Herencia y Polimorfismo..... 9

7. Objetos.....10

7.1 Items.....10

7.2 Poderes .....……..…11

7.3 Manejo de colas de los items.....11

8.0 El forms1 o programa.....12

8.1 Alternativas,problemas y limitaciones.....13

9.0 Diagrama UML.....14

1.Introducción:

En este proyecto se desarrollará un videojuego inspirado en la película **Tron** utilizando el lenguaje **C#** y herramientas como **Visual Studio** o **Unity**. El objetivo principal es implementar estructuras de datos en la lógica del programa, lo que permitirá profundizar en su funcionamiento y mejorar las habilidades en el diseño y desarrollo de Programas.

2.Tron: Legacy

Tron es una película de ciencia ficción originalmente lanzada en 1982, pero hablaremos principalmente de su secuela Tron: Legacy lanzada en 2010, esta rinde homenaje a la cinta original y explora lo mismo que esta, solo que va un poco más allá. Trata sobre Sam Flynn, el hijo de un legendario desarrollador de videojuegos que desapareció hace años, Sam que creció sin su padre, descubre una pista que lo lleva a una antigua sala de juegos de Arcade donde su padre trabajaba. Allí es transportado al mundo digital conocido como “The Grid” el universo virtual donde su padre ha estado atrapado por décadas. Una vez dentro Sam se encuentra con su padre, quien fue exiliado por CLU que es como una versión digital de si mismo creado para perfeccionar el sistema. Sin embargo, CLU se corrompió en algún punto y se vuelve un dictador que busca eliminar cualquier imperfección o defecto para lograr un sistema perfecto. Sam busca derrotar a CLU, liberar a su padre y encontrar una manera de escapar de “The Grid” antes de que CLU logre invadir el mundo real.

3.Las Motos de Luz en Tron: Legacy

Las motos de luz presentes en la película son Motocicletas futuristas utilizados en una de las competiciones más emblemáticas del mundo digital “The Grid” aparecen en varias secuencias de acción dentro de este mundo digital y además de ser un componente visual icónico de Tron, son una parte central del entretenimiento y la estructura de poder en este mundo virtual. Estas motos dejan tras de si una estela de luz solida que actúa como un tipo de barrera mortal.

3.1¿Pero porque compiten?

En “The Grid” las competiciones son una forma de control y entretenimiento. Bajo el control de CLU, los programas (entidades digitales) son forzados a competir en juegos mortales como una forma de mantenerse bajo control y entretener a las masas. En el fondo CLU utiliza estas competiciones como medio para eliminar programas que considera defectuoso o que se rebelan contra él.

En la película vemos como Sam es arrestado y obligado a competir en un tipo de coliseo digital en unos desafíos peligrosos, siendo uno de estos las Carreras de motos de luz.

3.2¿En qué consiste las Carreras de motos de luz?

Es un juego de estrategia y velocidad en donde los competidores deben maniobrar sus vehículos en una arena cerrada mientras sus motos dejan una estela de luz solida tras de sí. Esta estela actúa como barrera impenetrable para los oponentes y el objetivo es hacer que los competidores choquen con estas barreras o contra las paredes de la arena provocando su desintegración.

En la película la Estela de Luz no desaparece así que los competidores necesitan buenos reflejos y una estrategia solida debido a que la arena es un campo cerrado y entre más avance la competición menos espacio para maniobrar va a haber.

4.Proyecto Tron

Ya contextualizado el Proyecto, la idea es desarrollar un videojuego que plasme la idea de las carreras de motos luz de la película, utilizando lenguaje C# y Estructuras de Datos. La arena va a ser una malla o Matriz De nodos bidimensional, utilizando listas enlazadas y además habrán objetos o poderes esparcidos por la malla que podrán usar tanto el jugador como las motos enemigas y con este se utilizarán Pilas o colas para manejar la lógica.

4.1 Inicio del Proyecto

Como cualquier lenguaje de programación que recién se empieza a aprender, lo mas complicado es el inicio, comprender la estructura de este, la sintaxis etc.

Se comenzo el desarrollo del proyecto primero comprendiendo sobre sintaxis de las clases en C#, como se definen, como se inicializan y como se tratan sus métodos. Para esto se hizo uso del repositorio de GitHub del curso CE-1103

Definición de Clase

Las clases son *plantillas que definen la estructura y comportamiento de los objetos.* Por ejemplo, la clase Televisor en C# se puede definir de la siguiente manera:

```
class Televisor
{
    // Atributos
    string marca;
    int pulgadas;
    bool encendido;

    // Métodos
    void Encender()
    {
        encendido = true;
    }

    void Apagar()
    {
        encendido = false;
    }
}
```

También se vio sobre los principios de POO como la abstracción, polimorfismo, encapsulamiento, herencia y lo más importante en esta primera parte, los punteros.

Un puntero es una variable que almacena la dirección en memoria de un dato. Siendo este entonces una variable cuyo valor es la dirección de memoria de otra variable.

4.2 Desarrollo de la matriz o “The Grid”

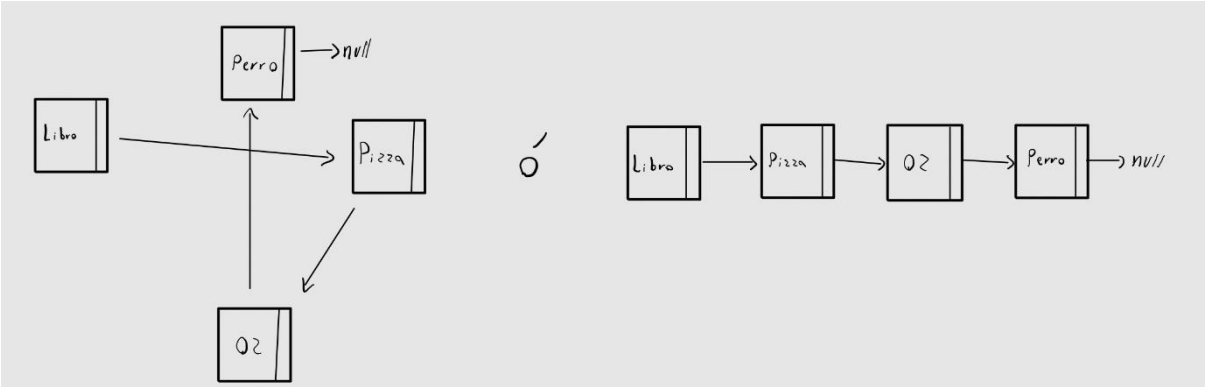
Lo primero fue comprender los nodos, que es un nodo, como se implementa en código, como se utiliza este en listas enlazadas.

Cada elemento de una lista es un Nodo y cada nodo posee dos partes.

-El valor: es el elemento importante para el programador, es el dato que almacena el nodo.

-La referencia: Es el dato que actúa como elemento para encadenar la lista, es un puntero al dato siguiente o al dato anterior.

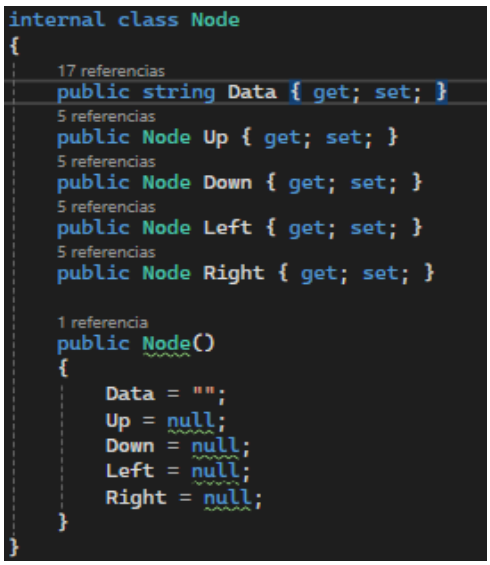
Una forma mas sencilla de verlo es con cajas, una lista es una fila de cajas separadas, podrían estar en orden o desorden físicamente, pero cada caja puede almacenar algo, ya sea un libro, comida o un perro y además cada caja tiene una flecha que indica cual es la caja siguiente.



Cada caja o nodo posee un pequeño espacio dedicado para el dato de donde está el nodo siguiente, ósea la referencia o puntero. Este es el que se muestra con una línea al lado derecho de cada caja. El ultimo nodo apunta a null o su referencia es igual a null lo que significa que no existe nodo siguiente. Es útil para comprobar cuando se llego al final de una lista o a saber si una lista es vacía.

5.Clase Nodo y Matriz

En código para hacer la matriz primero se desarrolló la clase Nodo



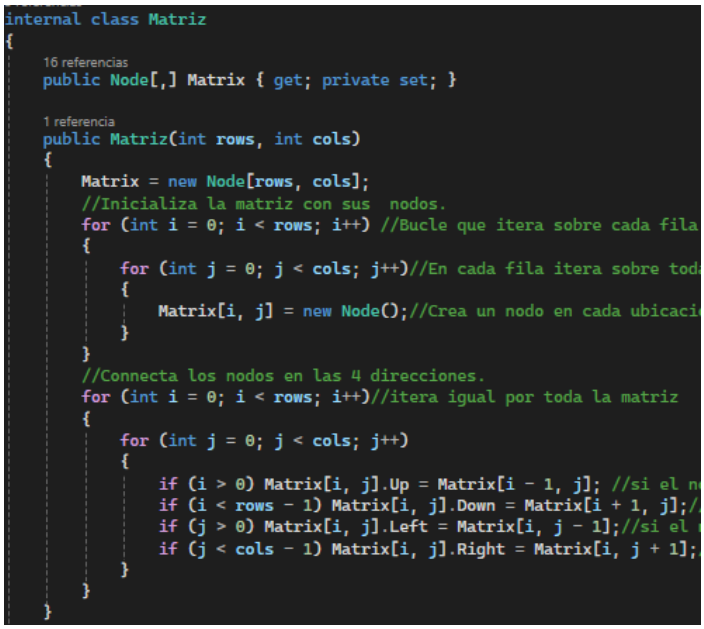
```
internal class Node
{
    17 referencias
    public string Data { get; set; }
    5 referencias
    public Node Up { get; set; }
    5 referencias
    public Node Down { get; set; }
    5 referencias
    public Node Left { get; set; }
    5 referencias
    public Node Right { get; set; }

    1 referencia
    public Node()
    {
        Data = "";
        Up = null;
        Down = null;
        Left = null;
        Right = null;
    }
}
```

Una clase Nodo con su respectiva variable Data que indica que es el dato que almacena.

Luego 4 variables de tipo Nodo(ósea si mismo) que se usaran para indicar las referencias de los nodos Arriba, abajo, izquierda y derecho, esto debido a que el juego es una matriz en 2D.

En un inicio cada referencia apunta a null, lo que significa que al crear una instancia nodo, se le deberán asignar sus referencias ya sea a si mismo si la lista esta vacía o al siguiente, anterior, arriba o abajo, según se requiera.

Para la matriz se creó una clase matriz



```
internal class Matriz
{
    16 referencias
    public Node[,] Matrix { get; private set; }

    1 referencia
    public Matriz(int rows, int cols)
    {
        Matrix = new Node[rows, cols];
        //Inicializa la matriz con sus  nodos.
        for (int i = 0; i < rows; i++) //Bucle que itera sobre cada fila
        {
            for (int j = 0; j < cols; j++)//En cada fila itera sobre toda
            {
                Matrix[i, j] = new Node();//Crea un nodo en cada ubicaci
            }
        }
        //Connecta los nodos en las 4 direcciones.
        for (int i = 0; i < rows; i++)//itera igual por toda la matriz
        {
            for (int j = 0; j < cols; j++)
            {
                if (i > 0) Matrix[i, j].Up = Matrix[i - 1, j]; //si el n
                if (i < rows - 1) Matrix[i, j].Down = Matrix[i + 1, j];//
                if (j > 0) Matrix[i, j].Left = Matrix[i, j - 1];//si el
                if (j < cols - 1) Matrix[i, j].Right = Matrix[i, j + 1];
            }
        }
    }
```

Acá se define la variable de un Nodo bidimensional Node[,] es un arreglo para objetos tipo Node.

Al inicializar la matriz recibimos como entrada el tamaño deseado ejemplo un 30x30. Primero se define a Matrix para crear la matriz bidimensional de tamaño 30x30. Pero aun no se han creado nodos, este solo sería el contenedor. Inicialmente la matriz posee referencias nulas en todas sus celdas ya que los objetos Nodo aun no existen.

Para la creación de Nodos se utilizan 2 bucles iterativos, donde se van creando Nodos en cada posición y asignándoles valores i y j, como normalmente se tendría en una matriz de Python. Solo se están creando instancias de nodos en cada posición [i , j] de la matriz.

Ya con los nodos de la matriz tendríamos la malla lista, pero son solo cajas sin flechas o referencias, aún no están conectados entre sí. Para esto se vuelve a iterar por toda la matriz y a cada nodo le va asignando referencias según su posición, como vemos en el código los ifs del bucle de asignación de referencias verifica si hay el nodo está en algún

borde la matriz, por ejemplo, si el nodo no esta en la primera fila entones tiene un nodo arriba. Si no esta en la ultima fila entonces tiene un nodo abajo. Y a partir de estos if asigna la respectiva referencia utilizando la notación Matrix[i, j] donde cada nodo tiene una identidad de pares ordenados para ubicarse en la matriz.

Se considero utilizar otros métodos para crear la matriz, como por ejemplo utilizando lista circular doblemente enlazada. Si necesitamos una matriz 30x30, simplemente creamos una lista de 900 nodos y cada 30 nodos “simulamos” que la lista se va para abajo. Al llegar a 30 nodos, al nodo 31 se le asigna como Nodo31.Arriba = Head. Y mientras recorre del 31 al 60 va asignando como Nodo.Arriba a los nodos del 1 al 30 en orden. Y a estos como Nodo.abajo a los nodos del 31 al 60. Al llegar al 60 la referencia del Nodo 31.Arriba que se guardo pasa a ser ahora del Nodo.arriba del Nodo 61 y asi iría hasta completar los 900 nodos. Esta idea no se implemento por falta de tiempo y sobre todo por errores o confusiones al intentar hacer la lógica de movimiento de la moto.

6.Clase Moto y Clases hija Jugador y Enemigos.

```
14 referencias
internal class Bike
{
    protected Node currentPosition;
    public int velocidad;
    public int combustible;
    public LinkedList<Node> estela;
    public int tamañoestela;
    public double intervalTime;
    public ItemQueue itemQueue = new ItemQueue();
    private bool isInvulnerable = false;
    protected string currentDirection; //Para el movimient


    2 referencias
    public Bike(Node initialPosition)
    {
        currentPosition = initialPosition; //Inicializa la
        currentPosition.Data = "Bike"; //El nodo actual al
        estela = new LinkedList<Node>(); //Inicializa la l
        tamañoestela = 3;
        combustible = 100;
        velocidad = new Random().Next(1, 11); //la velocid
        intervalTime = ConvertSpeedToInterval(velocidad);/
        currentDirection = "right"; // Dirección inicial p
    }
```

Se creo una Clase Bike que va a ser la principal o padre, acá se definen variables importantes como la de posición actual para saber en que nodo se encuentra la moto en ese momento, ya sea jugador o enemigo. También se asignan variables como el de velocidad, combustible, tamaño de la estela que sigue la moto, Si la moto tiene invulnerabilidad o dirección actual que guarda la dirección para que la moto se mueva constantemente y solo gire cuando se hace un cambio en la dirección.

En el código del Forms1 se crean las instancias de motos donde se les asigna una posición generalmente aleatoria, se define entonces la posición actual de la moto creada como el parámetro o ubicación que recibió de entrada al crearse.
También tendríamos la estela definida para usarla después en el código para hacer la estela de las motos, que será una lista enlazada. La velocidad que se decide en un intervalo aleatorio entre 1 y 10 y en un método este intervalo es convertido a nodos/segundo. Y por ultimo una dirección actual con un valor inicial cualquiera generalmente para evitar errores. Como se mostro en la parte de la matriz los nodos solo almacenaran un dato de tipo Sting para toda la lógica.

```
1 referencia
private double ConvertSpeedToInterval(int speed)
{
    speed = Math.Clamp(speed, 1, 10);// Limita la
    return 250 + (10 - speed) * 25;//Intervalo ba
}
```

Método de conversión de intervalos de velocidad a nodos/s. El método math.clamp restringe el valor al intervalo establecido, si es menor a 1 lo ajusta a 1 y si es mayor a 10 lo ajusta a 10.

6.1 Movimiento de la Moto.

```
protected void Move(Node newPosition)
{
    if (newPosition != null)
    {
        // Verificar colisiones
        if (HasCollision(newPosition))
        {
            HandleCollision(newPosition);
            return;
        }

        //Si no hay colisión, continuar movien
        estela.AddFirst(currentPosition); //Ag
        if (estela.Count > tamañoestela)
        {
            Node lastNode = estela.Last.Value;
            lastNode.Data = ""; //Limpia el da
            estela.RemoveLast(); //Elimina el
        }

        currentPosition.Data = "Trail"; //Marc
        currentPosition = newPosition; //Mover
        currentPosition.Data = GetBikeData();

        // Consumir combustible
        combustible -= velocidad / 5;

        if (combustible <= 0)
        {;
            this.Destroy();  //Combustible ago
        }
    }
}
```

Para el movimiento de la moto se hizo un método Move que se llama cada vez que se mueve una moto ya sea el jugador mediante las teclas WASD o los enemigos mediante un random. Se llama al método por ejemplo con un:

Move(currentPosition.Up);

Donde entonces la nueva posición seria prácticamente el nodo que este arriba de la moto en ese momento.

Primero verifica la colisión, que si en ese nodo al que se quiere mover no existe un Dato Moto cualquiera u Objeto bomba.

Si no hay colisión se puede mover, la posición actual pasa a agregarse al inicio de la estela y el ultimo Nodo de la estela pasa a ser vacío, ahora la posición actual para a ser de la estela y la de la moto pasa a ser la siguiente. Lo que quiere decir que la estela se mueve siempre detrás del jugador.

Al recorrer cada nodo se pierde 5 de combustible y cuando este llego a 0 destruye la moto.

6.2 Herencia y Polimorfismo.

Las Motos Hijas tanto la del jugador como la de los enemigos se heredan de la clase padre.

```
internal class Jugador : Bike
{
    private LinkedListNode<Item> currentItem;
    private ItemQueue itemQueue;
    3 referencias
    public bool isDestroyed { get; private set; }

    1 referencia
    public Jugador(Node initialPosition) : base(initialPosition)
    {
        currentPosition.Data = "Jugador"; //Cambia el dato que a
        itemQueue = new ItemQueue();
        isDestroyed = false;
    }
    1 referencia
    public void MoveUp()
    {
        Move(currentPosition.Up);
    }
    1 referencia
    public void MoveDown()
    {
        Move(currentPosition.Down);
    }
    1 referencia
    public void MoveLeft()
    {
        Move(currentPosition.Left);
    }
    1 referencia
    public void MoveRight()
    {
        Move(currentPosition.Right);
    }
    8 referencias
    public override void Destroy()
    {
```

Donde podemos ver 2 variables para el uso de objetos y la lista enlazada de ítems. En la que se profundizara después.

La lógica de destrucción se manejará mediante una variable bool en ambos casos sea enemigo o jugador.

Al apretar WASD se llama a estos métodos que a su vez llaman a el método Move de la clase padre.

La clase enemigo se inicializa idénticamente al jugador, lo que cambia son sus métodos, Como el de movimiento aleatorio que se maneja con una lógica de Switch y casos.

```
public void MoveRandom()
{
    Random random = new Random();
    List<int> possibleDirections = new List<int> { 0, 1, 2, 3 }; // 0 = Up, 1 =

    switch (currentDirection)
    {
        case "up":
            possibleDirections.Remove(1);
            break;
        case "down":
            possibleDirections.Remove(0);
            break;
        case "left":
            possibleDirections.Remove(3);
            break;
        case "right":
            possibleDirections.Remove(2);
            break;
    }

    int direction = possibleDirections[random.Next(possibleDirections.Count)];

    switch (direction)
    {
        case 0:
            Move(currentPosition.Up);
            currentDirection = "up";
            break;
        case 1:
            Move(currentPosition.Down);
            currentDirection = "down";
            break;
        case 2:
            Move(currentPosition.Left);
            currentDirection = "left";
            break;
        case 3:
            Move(currentPosition.Right);
            currentDirection = "right";
            break;
```

El método MoveRandom toma la dirección actual y restringe las posibles direcciones para evitar que pueda moverse en dirección contraria de golpe, por ejemplo, si va para arriba, sea incapaz de moverse hacia abajo. Se asigna una lista de posibles direcciones y según la posición actual elimina una. Luego tira un dado random que toma un valor de los restantes en la lista y usa este para elegir la dirección siguiente a la que se va a mover.

7. Los objetos

Primero se desarrolló una clase item y una Poder y a partir de esta derivar para obtener en ítems el combustible, incrementar y bomba.

7.1 Items

```
11 referencias
internal abstract class Item //Clase abstracta para los items
{
    6 referencias
    public abstract void Aplicar(Moto moto);

}
1 referencia
internal class Combustible : Item
{
    4 referencias
    public override void Aplicar(Moto moto)
    {
        if (moto.combustible < 100)
        {
            moto.combustible = Math.Min(moto.combustible + 10, 100);
        }
        else
        {
            // Reenfilar el combustible si está lleno
            moto.itemQueue.Enqueue(this);
        }
    }
}
0 referencias
internal class Incrementar : Item
{
    4 referencias
    public override void Aplicar(Moto moto)
    {
        Random random = new Random();
        int aumentar = random.Next(2, 6); // Incremento aleatorio ent
        moto.tamañoestela += aumentar;
    }
}
```

```
internal class Bomba : Item
{
    4 referencias
    public override void Aplicar(Moto moto)
    {
        //Verificar si la moto es invulnerab
        if (!moto.Invulnerable()) //revisar
        {
            moto.Destruir();
        }
```

Tratar a los objetos como clases ayudo a manejar la lógica de agregarlos a una cola para usarlos según se vayan recogiendo en el juego. Además de ayudar a la organización del código.

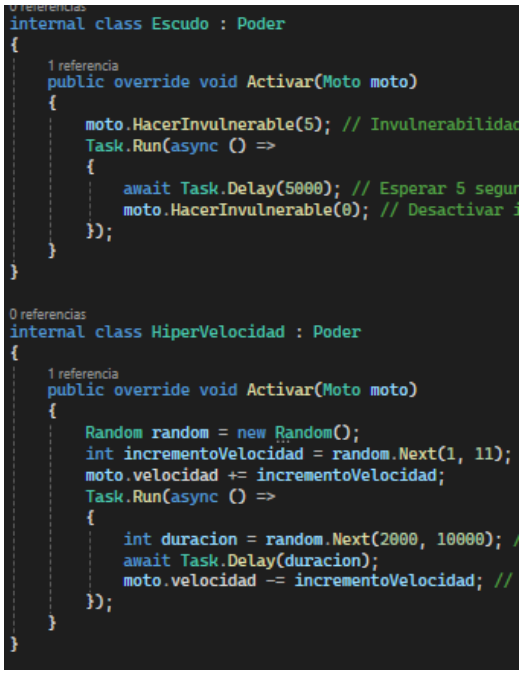
Cuando cualquier moto ya sea jugador o enemigo toma una celda de combustible esta se aplica directamente sumando 10 si este no esta al máximo, pero si esta al máximo lo vuelve a mandar a la cola donde se aplicara apenas tenga menos de 100 de combustible.

El incrementar aumenta el tamaño de la estela en un intervalo de 1 a 5 nodos, solo aumenta la variable de tamaño de estela de la moto que lo agarre.

Por ultimo la bomba es simplemente un objeto que aparece en la matriz y cualquier moto que no este con el poder invulnerable activo y la toque, se destruye.

7.2 Poderes

Para los poderes se divide en 2 subclases de la clase poder.



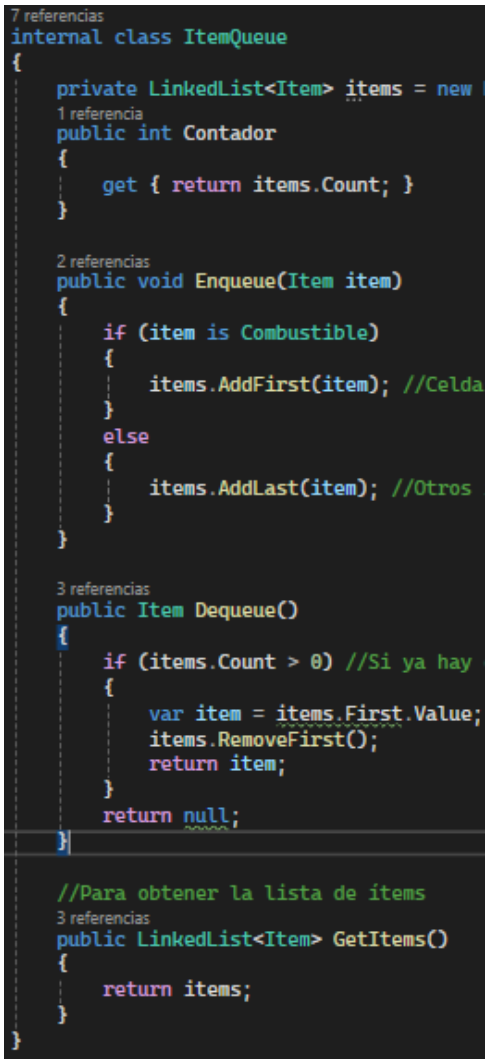
```
internal class Escudo : Poder
{
    1 referencia
    public override void Activar(Moto moto)
    {
        moto.HacerInvulnerable(5); // Invulnerabilidad
        Task.Run(async () =>
        {
            await Task.Delay(5000); // Esperar 5 segur
            moto.HacerInvulnerable(0); // Desactivar i
        });
    }
}

0 referencias
internal class HiperVelocidad : Poder
{
    1 referencia
    public override void Activar(Moto moto)
    {
        Random random = new Random();
        int incrementoVelocidad = random.Next(1, 11);
        moto.velocidad += incrementoVelocidad;
        Task.Run(async () =>
        {
            int duracion = random.Next(2000, 10000);
            await Task.Delay(duracion);
            moto.velocidad -= incrementoVelocidad; //
        });
    }
}
```

Escudo, apenas la active el jugador o cuando la agarre un enemigo, otorga 5 segundos de invulnerabilidad. Este se ejecuta en un hilo separado Task.Run que permitirá que el código principal no se bloquee.

Con Task.delay se espera de manera asincrónica durante 5 segundo sin bloquear el hilo principal. Al inicio la variable invulnerable de la moto pasa a true y luego del delay de 5s vuelve a false para seguir siendo afectado por las colisiones con normalidad.

Hipervelocidad al activarse aumenta la velocidad en un intervalo de 1 a 10, idénticos a los intervalos de velocidad base de las motos. Suma a la variable de velocidad de la moto y después de una Tarea asincrónica con un delay vuelve a su velocidad normal. La duración de la hipervelocidad también es aleatoria de 2 a 10 segundos.

7.3 Manejo de Colas de los ítems



```
7 referencias
internal class ItemQueue
{
    private LinkedList<Item> items = new
    1 referencia
    public int Contador
    {
        get { return items.Count; }
    }

    2 referencias
    public void Enqueue(Item item)
    {
        if (item is Combustible)
        {
            items.AddFirst(item); //Celda
        }
        else
        {
            items.AddLast(item); //Otros
        }
    }

    3 referencias
    public Item Dequeue()
    {
        if (items.Count > 0) //Si ya hay
        {
            var item = items.First.Value;
            items.RemoveFirst();
            return item;
        }
        return null;
    }

    //Para obtener la lista de ítems
    3 referencias
    public LinkedList<Item> GetItems()
    {
        return items;
    }
}
```

Para el manejo de estas se creo una clase ItemQueu donde se manejarán. Primero se crea un atributo que almacene la lista enlazada de objetos de tipo Item.

Se crea una propiedad para devolver el número de ítems en cola.

Y en métodos estaría el ya conoció Enqueu para poner al final de la cola el item que se recoja y Dequeu para eliminar el primer elemento en la lista ósea apenas se usa,dándole prioridad siempre al combustible.

8.0 El forms1 o Programa

El codigo del forms o programa agrega el uso de lista enlazada en enemigos y uso de los timers para manejar diferentes logicas del juego que lo necesitan.

El constructor del forms1 es donde se configura la interfaz grafica, se inclute el area donde se mostrara el tablero del juego y ademas etiquetas informativas en la parte superior, tambien crea al jugador en el centro ed la matriz y posicionara a los enemigos en nodos aleatorios,cada enemigo con su respectivo temporizador para controlar su movimiento basado en su velocidad. El jugador se controla con las teclas WASD pero este solo cambia su direccion,el jugador siempre esta en movimiento.

Para dibujar el tablero se utiliza el metodo Form1 Paint para dibujar la matriz del juego,el juegador,los enemigos,las estelas que dejan y los items y poderes.

La clase Forms1 seria la interfaz principal del juego,donde se gestiona todo el enterno grafico y la interaccion con el usuario. El codigo define metodos y variables para representar primeramente un panel, donde se ve una matriz de tamaño fijo de 30x30 y un panel superior donde esta la informacion,siendo labels y imágenes incertadas para los poderes del jugador.

En el panel superior se representan varias etiquetas Label que muestran informacion actualizada sobre el estado del jugador y los enemigos,como velocidad,combustible y poderes. A la derecha estarian imágenes cargadas desde la carpeta resoucers que tambiense utilizan para representar items visualmente en la matriz.

Primero se implemento una variable y metodos de temporizador para controlar el movimiento de los enemigos,del jugador y el uso de objetos. Como anteriormente se explico tanto los jugadores como los enemigos utilizan un temporizador para actualizar su posicion basado en un intervalo de velocidad calculado según su velocidad,para obtener una medida de “Nodos por segundo”.
Ahora se crea un bucle para crear los 3 enemigos en posiciones donde no queden nunca encima de algun item,del jugador o de otro enemigo.

```
//Creación de 3 motos enemigas con posiciones únicas
for (int i = 0; i < 3; i++)
{
    Nodo enemyStartNode;
    do
    {
        enemyStartNode = linkedList.GetNode(new Random().Next(0, GridSize), new Random().Next(0, GridSize));
    } while (enemyStartNode == jugador.RPosActual() || enemigos.Any(e => e.RPosActual() == enemyStartNode));

    var enemigo = new Enemigos(enemyStartNode);
    enemigos.Add(enemigo);

    //Configura el temporizador para cada enemigo con su respectiva velocidad
    var timerEnemigo = new System.Windows.Forms.Timer();
    timerEnemigo.Interval = IntervaloVelocidad(enemigo.velocidad);
    timerEnemigo.Tick += (sender, e) => Movimientoenemigo_Tick(enemigo);
    timerEnemigo.Start();

    timersEnemigos.Add(timerEnemigo);
}
```

Luego se van añadiendo a una lista de enemigos y se les asigna su velocidad respectiva a cada moto, se inicia el timer y se agrega a la lista correspondiente.

El timer del movimiento del jugador se maneja con un codigo identico al del enemigo.

Para la interaccion y controles, el jugador se mueve con las teclas W,A,S,D cambiando su direccion actual y siempre evitando giros abruptos de 180 grados. Esto se logra con el metodo MovimientoJugador_Tick, que hace que el jugador siempre este en movimiento a su ultima direccion elegida,al presionar una tecla cambia una variable que se pone “en cola” para ser el proximo movimiento a seguir hasta que se cambie nuevamente.

El metodo respawnItems “genera” los items en posiciones aleatorias de la matriz utilizando el metodo de nodos disponibles que recorre toda la matriz para ver cuales estan ocupado, se puede cambiar la cantidad de items generados. La asignacion de los items es mediante un switch con la logica (i % 5) que retorna entre 0 y 4 según el valor de i. A partir de ahí asigna al nodo.data un string según el objeto que se vaya a guardar

8.1 Alternativas,problemas y limitaciones

Se penso en muchas alternativas respecto al movimiento tanto enemigo como jugador,como que los enemigos tuviera un movimiento constante pero cada cierto tiempo cambiaran de direccion, el uso de 2 timers independientes para lograr esto se complico bastante asi que al final se descarto la idea y a su vez se adopto la opcion de dejar un movimiento completamente aleatorio pero que los enemigos eviten su propia estela,con el fin de que no se destruyeran al empezar la partida. Ahora se destruyen al chocar contra otra estela, contra el jugador o al quedarse sin combustible, siendo esta ultima la mas comun.

Con el movimiento del jugador hubieron muchisimos problemas porque guardaba los cambios de direccion sin primero cambiar de direccion,lo que provocaba que chocara con su propia estela y se destruyera sin sentido, o que podia girar abruptamente 180 grados y chocar con su estela. Se corrigio poniendo condiciones y verificaciones para que no pudiera cambiar la variable de direccion sin cambiar de direccion y verificando que no sea direccion contraria.

Y con los objetos,que agarrarlos aplicara su logica fue lo mas desafiante,se probo con metodos y variables en clases independientes o en la misma clase moto,no funciono,luego se adopto usarlos como clases bajo una clase padre con un abstract void de aplica para que al crear el objeto y llamarse con aplicar,aplicara los cambios,como agregar combustible o explotar y destruir la moto en caso de la bomba. Se hizo la clase items,poderes,pila y colaprioridad para manejar ambos,cuando la pos actual tenga un nodo.dato igual a un objeto,según el objeto que sea se agrega a la pila o la cola para ser usado.

9. Diagrama UML