Instituto Tecnológico de Costa Rica


Ingeniería en computadores

Segundo semestre 2024

Fecha: 8 de septiembre de 2024

Algoritmos y estructuras de datos 1

Grupo #2

Primer proyecto

Estudiante:

Pablo Esteban Rivera Redondo

Carnet: 2023395989

# Introducción

En la presente documentación se expondrá la descripción del problema y de la solución que fue encontrada para este primer proyecto programado del curso, se explicara la solución presentada, así como alternativas consideradas y problemas encontrados durante el desarrollo del proyecto.

El objetivo del proyecto es evaluar los primeros temas del curso de algoritmos y estructuras de datos 1, como programación orientada a objetos y el uso de estructuras de datos lineales como lo son las listas enlazadas, colas y pilas.

# Contenido

## Descripción del problema

El proyecto consiste en la de la elaboración de un videojuego inspirado en el clásico juego de arcade conocido como tron, al que mas tarde se le dedicaron algunas producciones en la pantalla grande.

El juego cuenta con varios elementos, como la moto del jugador, los enemigos, los ítems y poderes, además del mapa en el cual se desarrolla el juego. El jugador debe mover la moto en un grid, este grid está formado por nodos, los cuales tienen 4 referencias hacia las diferentes direcciones, la moto del jugador es una lista enlazada en donde el head representa la moto y los demás nodos su estela, seria lo mismo para la moto enemiga, el juego consiste en destruir las motos enemigas haciéndolas chocar con la estela.

## Descripción de la solución

### Implementación de la clase moto:

Para la implementación de esta clase se establecieron atributos como velocidad, tamaño de la estela, combustible, ítems y poderes, los cuales se solicitaron en los requerimientos del proyecto, pero ¿Cómo funcionan en nuestra implementación?

La velocidad es una variable de tipo entero, la cual determina la velocidad de las motos, esta se selecciona de manera aleatoria entre 1 y 10, de la manera en la que estaba indicado en los requerimientos dejaba a entender que todas las motos debían tener una velocidad diferente, lo cual en el caso de esta solución no fue posible, ya que independientemente de la velocidad de la moto la visualización de esto iba a estar relacionado a la velocidad de actualización del formulario, entonces de la manera en la que lo implemente es que se elige un numero aleatorio para todas las motos y luego la velocidad del formulario se iba a afectar solo por esta velocidad, entonces técnicamente las motos si tienen velocidad aleatoria pero no individual.

El tamaño de la estela es una variable de tipo entera, la cual se establece a la hora de instanciar ya sea una moto o un enemigo, como esta en los requerimientos inicialmente es de 3 espacios.

El combustible es también una variable de tipo entera, esta inicializada en la clase en 100, en el método mover de la clase moto existe una variable que funciona como un contador, cuando el contador es igual o mayor a 5 (esto porque note que a diferentes velocidades se comportaba distinto) el combustible se reduce en una unidad.

Ítems es una cola, en la cual se van a almacenar los ítems que se generan de manera aleatoria en el grid, mas adelante profundizaremos en los ítems como tal, utilice un timer para el cooldown de un segundo entre la aplicación de un item y otro, este timer llamara a la función aplicarsiguiente(); que es por así decirlo el dequeue de mi código. Cuando una moto recoge un item llama a la función agregaritem(); el cual es el enqueue de mi código, aunque es

verdaderamente difícil recoger mas de un item en menos de un segundo, ya que los ítems tienen un intervalo de aparición de 5 segundos, sin embargo, funciona.

```
public virtual void Mover(List<Moto> motos)
    // Mueve la moto en la dirección actual
    // Añadir la nueva posición al frente de la lista de la estela
    Estela.AddFirst(nuevaPosicion);
    nuevaPosicion.BackColor = Color.Red; // Color de la moto

    // Actualizar el Head
    Head = Estela.First;

    // Eliminar el último nodo solo si la estela es más larga que el tamaño permitido
    while (Estela.Count > TamañoEstela)
    {
        LinkedListNode<Casilla> nodoAntiguo = Estela.Last;
        Estela.RemoveLast();
        nodoAntiguo.Value.BackColor = Color.Black; // Restaurar el color anterior
    }

    // Cambiar el color de la estela (los nodos que no son la moto)
    foreach (var nodo in Estela)
    {
        if (nodo != Head.Value)
        {
            nodo.BackColor = Color.Yellow; // Color de la estela
        }
    }
```

Este es un extracto de la función mover, para mostrar como es que se comporta el movimiento de la estela y la moto, con una función llamada  obtenernuevaposicion(); calcula la variable nuevaposicion, la cual será utilizada para actualizar el head de la lista enlazada y mover la estela con ella.

### Creación y dibujado del grid:

Para definir la matriz se utiliza un nuevo nodo (filas, columnas), nodo es una clase dentro del código, el cual tiene como atributos las cuatro referencias a las direcciones, se itera a través de las casillas de la matriz, si la casilla esta al borde de la matriz, es decir si es la primera o ultima fila o columna esta se establece como una casilla pared, de lo contrario como una casilla libre, de esta manera creando el espacio del juego.

```
matriz[i, j] = new Nodo(casilla);

if (i > 0)
{
    matriz[i, j].Arriba = matriz[i - 1, j];
    matriz[i - 1, j].Abajo = matriz[i, j];
}

if (j > 0)
{
    matriz[i, j].Izquierda = matriz[i, j - 1];
    matriz[i, j - 1].Derecha = matriz[i, j];
}
```

En esta imagen podemos ver como es que cada nodo se vincula con el nodo adyacente, si no está en la primera fila el nodo se conecta con el de arriba y el de abajo y si no está en la primera columna se conecta con el nodo de la derecha y el de la izquierda.

### Bots (enemigos):

Elabore la clase enemigos heredando de la clase principal moto, y sobre escribiendo el metdo de mover para que en vez de que se muevan con la presión de las teclas, hicieran un cambio de dirección cada 3 segundos, para esto fue necesario utilizar un timer para manejar este tiempo.

```
2 references
private void CambiarDireccionAleatoria()
{
    Random random = new Random();
    Direction nuevaDireccion;

    do
    {
        nuevaDireccion = (Direction)random.Next(0, 4); // Elegir una dirección aleatoria
    }
    while (EsDireccionOpuesta(nuevaDireccion)); // Asegurar que no sea la dirección opuesta

    direccionActual = nuevaDireccion;
}

1 reference
private bool EsDireccionOpuesta(Direction nuevaDireccion)
{
    return (direccionActual == Direction.Up && nuevaDireccion == Direction.Down) ||
           (direccionActual == Direction.Down && nuevaDireccion == Direction.Up) ||
           (direccionActual == Direction.Left && nuevaDireccion == Direction.Right) ||
           (direccionActual == Direction.Right && nuevaDireccion == Direction.Left);
}
```

Acá podemos ver como se hace el cambio de dirección de manera aleatoria, claro, velando por que la dirección nueva no sea la contraria, ya que iría en contra de la lógica propuesta para el juego.

### Ítems y generación aleatoria:

La clase ítems hereda de la clase casilla, esto para poder dibujar el ítem en una casilla del grid de manera aleatoria más adelante, la clase tiene atributos como X, Y que son para guardar las coordenadas del ítem al momento de generarse, también tiene un atributo de tipo bool, usado, este representara si el ítem ya fue usado y de esa manera eliminarlo del grid.

Ahora, de la clase ítem heredan otras tres clases, celda de combustible, crecimiento de estela y bomba, cada una de las clases sobrescribe el método aplicar de la clase, según el objetivo de cada ítem, la celda de combustible rellena la capacidad de combustible de la moto de manera aleatoria entre 10 y 40 unidades, lo mismo con el crecimiento de la estela, hace que aumente el atributo de la estela de la moto de manera aleatorio de 2 a 10 unidades.

Para la generación de los ítems en el grid se crea una lista de los tipos de ítems que pueden aparecer y luego se genera una posición aleatoria del grid para colocarlo, esto evitando los bordes.

```
public partial class Form1 : Form
    private void GenerarItemAleatorio(object sender, EventArgs e)
        List<Item> itemsDisponibles = new List<Item>
        {
            new CeldaCombustible(this),
            new CrecimientoEstela(this),
            new Bomba(this)
        };
        // Selecciona una posición aleatoria en el grid
        int x = random.Next(1, gridSize - 1); // Evita los bordes
        int y = random.Next(1, gridSize - 1);
        // Selecciona un item aleatorio
        Item itemAleatorio = itemsDisponibles[random.Next(itemsDisponibles.Count)];
        // Coloca el item en el grid y añade a la cola
        ColocarItemEnGrid(x, y, itemAleatorio);
        items.Enqueue(itemAleatorio);
    }
    1 reference
    private void ColocarItemEnGrid(int x, int y, Item item)
    {
        // Primero limpia la casilla donde se va a colocar el item, para asegurarse de que no esté oculta
        if (matriz[x, y].Casilla is Libre)
        {
            item.X = x;
            item.Y = y;
            item.Color = GetColorParaItem(item);
            matriz[x, y].Casilla.BackColor = item.Color;
            // Coloca el item en la posición (x, y) del grid
            matriz[x, y].Casilla = item;
            item.Width = 10;
            item.Height = 10;
            item.Left = x * 10;
            item.Top = y * 10;
            this.Controls.Add(item);
            this.Invalidate();
        }
```

En esta parte del código me he topado con un problema, y es que la moto se comporta de maneras extrañas e inesperadas a la hora de recoger un ítem, por alguna razón que no logre identificar, la moto al recoger un ítem se teletransporta a otra casilla, tuve la sospecha que es por como se colocaban los ítems de manera aleatoria, ya que a pesar de teletransportarse sigue respetando los limites del grid, pero de igual forma no pude llegar a una solución.

### Colisiones:

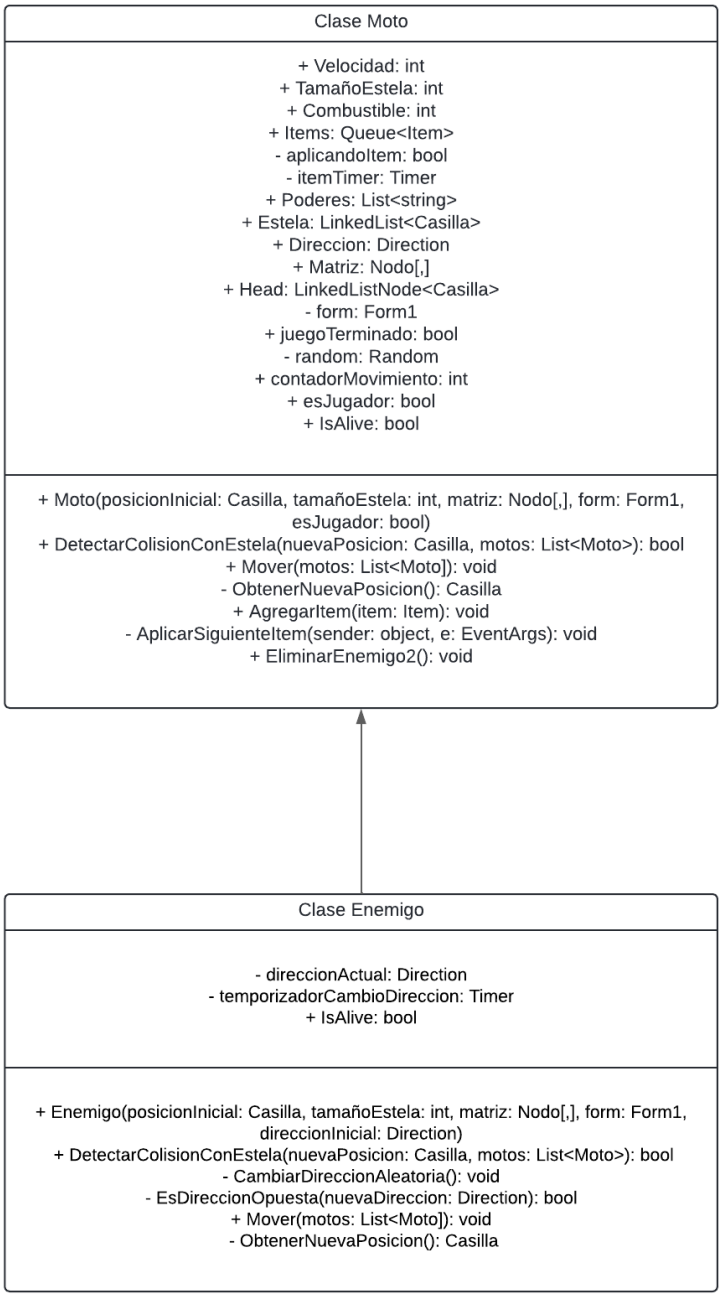
Las colisiones se evalúan, por así decirlo, dentro del método mover, si es una colisión con la pared simplemente verifica si la nueva posición es una pared y detiene el juego en el caso de la moto principal o elimina el enemigo en ese caso, para las colisiones entre las motos es algo más complejo.

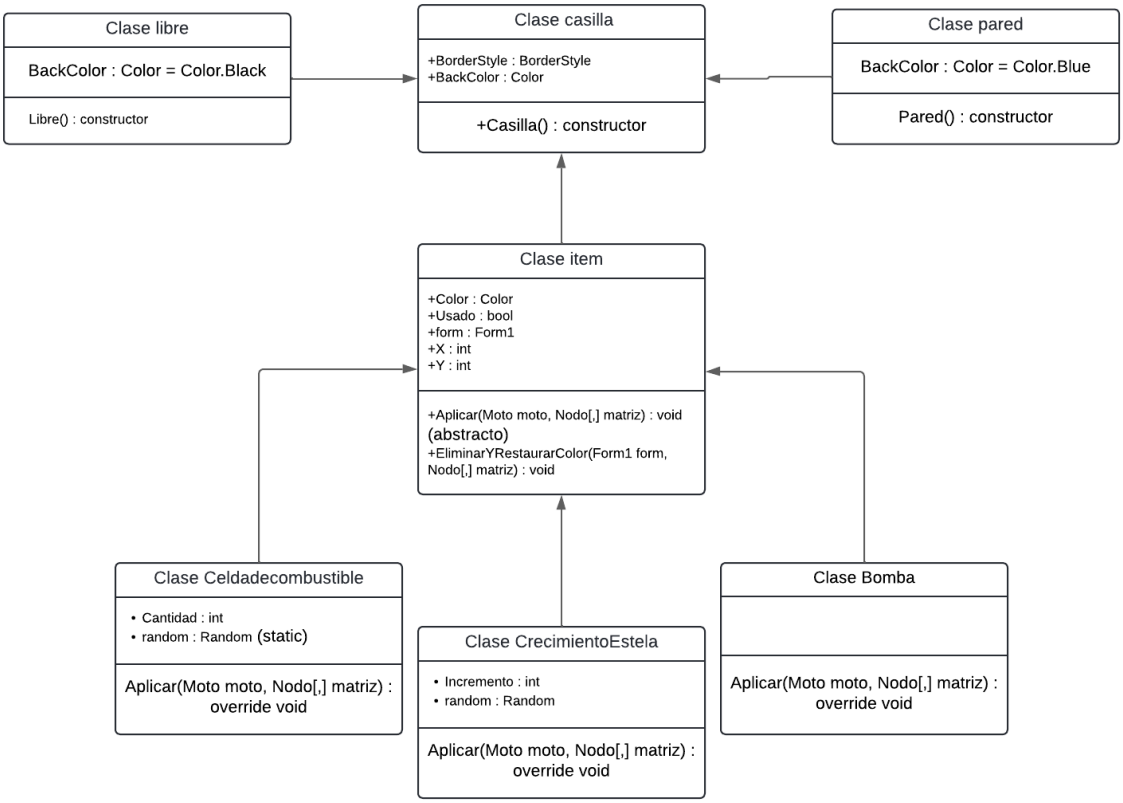
En la clase moto hay un método booleano que detecta las colisiones, ya sean con la estela propia, con el head de otra moto o con la estela enemiga, utiliza la variable nueva posición para saber si coincide con la posición del head de otra moto, la estela de otra moto o la propia estela.

Este método es verificado en la llamada del método mover, si es true y es una moto jugadora detiene el juego, si es enemigo simplemente elimina a este enemigo.

En esta parte encontré otro problema, lo que pasa es que cuando una moto enemiga “muere” deja una casilla pintada con el color de su estela, como si fuera su cadáver, la verdad no me moleste en corregirlo, me pareció divertido.

## Diagramas UML






## Repositorio de GitHub:

https://github.com/CLAVi7/TRON_like.git