Departamento de
Tecnología Digital

Tecsup TECNOLOGÍA CON SENTIDO

LABORATORIO N°

04

Algoritmos y Estructuras de Datos

DOCENTE:

CURSO:
Juan Neyra Alonzo

Algoritmos y estructuras de datos
3 - C26 - Sección A

Fundamentos de Programación para Videojuegos:

I. Capacidades

- Conocer sobre Listas enlazadas.
- Saber su importancia y los tipos de listas enlazadas.
- Saber implementar listas enlazadas.

II. Seguridad

- En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red. Así como la ingesta de alimentos y bebidas.
- Ubicar maletines y/o mochilas en lugar destinado para tal fin.
- Dejar la mesa de trabajo y la silla utilizada limpias y ordenadas.

III. Fundamento teórico

- Revise el material de la semana correspondiente antes del desarrollo del laboratorio.

IV. Normas empleadas

- *No aplica*

V. Recursos

- En este laboratorio, cada estudiante trabajará con una computadora con Windows 11.
- Los softwares requeridos ya se encuentran instalados en los laboratorios.

VI. Metodología para el desarrollo de la tarea

- El desarrollo del laboratorio es individual

VII. Procedimiento:

Sistema de Turnos con Historial

Objetivo

Implementar un sistema que registre las acciones realizadas en cada turno, permitiendo:

- Moverse hacia turnos anteriores.
- Añadir nuevos turnos.
- Observar cómo cambian la **posición** y **estadísticas** del jugador a lo largo del tiempo.

Requisitos

Debe registrarse en cada turno:

- La **posición** del jugador.
- Sus **estadísticas**.(opcional)
- El estado general de las **entidades** del juego.

Se debe utilizar una **lista doblemente enlazada personalizada** para almacenar la información correspondiente a cada turno.

Estructura sugerida de clases

GameUtils
Clase que contiene los **métodos genéricos** trabajados en clase y su implementación dentro de `GameManager`. El namespace debe ser su nombre

GameManager
Clase principal del juego. Debe incorporar, mediante **Odin Inspector** o mediante **botones en el Canvas**, toda la lógica necesaria para:

- Controlar las acciones del juego.
- Referenciar a las **entidades** actuales.
  Implementar la **lista personalizada**

Entity
Clase simple que debe contener un **nombre** y una **posición**. Se recomienda implementar una **interfaz** para estandarizar su comportamiento.

CustomDoubleLinkedList
Lista doblemente enlazada personalizada. Debe incluir un nuevo puntero llamado **`Peak`**, que permita moverse libremente entre los turnos registrados y acceder a turnos anteriores.
Además, se debe **sobrescribir el método `Add()`** con la siguiente lógica:

- Si se añade un nuevo turno mientras `Peak` apunta a un nodo intermedio (es decir, no al último), todos los turnos posteriores deben eliminarse.
- El nuevo nodo se inserta en la posición actual de `Peak`.
- Este nuevo nodo se convierte en el nuevo **último nodo** (`Last`).
- `Peak` se actualiza para apuntar al nodo recién añadido.

CustomNode
Clase que representa un turno. Debe contener:

- El **número del turno**.
- La **lista de entidades** correspondiente a ese turno.

Node y **DoubleLinkedList**
Utilizar como base lo **implementado y trabajado en clase**.

Entregables

- Repositorio en **GitHub** con todos los scripts.
- **Unity Package** exportado con la escena de prueba.

public class Node<T>

```
{
    #region Settings
    private T value;
    private Node<T> next;
    private Node<T> prev;
    #endregion
    #region Setters
    public Node(T _value)
    {
        value = _value;
        next = null;
        prev = null;
    }
    public void SetNext(Node<T> _next)
    {
        next = _next;
    }
    public void SetPrev(Node<T> _prev)
    {
        prev = _prev;
    }
    #endregion
    #region Getters
    public T Value => value;
    public Node<T> Next => next;
    public Node<T> Prev => prev;
    #endregion
}
public enum Emotion
{
    None,
    Happy,
    Angry,
```

```
    Scared,

}
public class Dialogo
{
    public string Name;
    public string Texto;
    public float Velocity;
    public Emotion emotion;
    public Dialogo(string Name, string Text, Emotion emotion, float velocity)
    {
        this.Name = Name;
        Texto = Text;
        this.emotion = emotion;
        Velocity = velocity;
    }

    public string GetDialog( out Emotion emotion)
    {
        emotion = this.emotion;
        return Texto;
    }

}

public class DoubleLinkedList<T>
{
    public Node<T> Head = null;
    public Node<T> Tail = null;
    public int Count;

    public virtual void AddNode(T dato)
    {
```

```
    Node<T> newNode = new Node<T>(dato);
    #region Caso de primer elemento
    if (Head == null && Tail == null)
    {
        Head = newNode;
        Tail = newNode;
        Count++;
        return;
    }
    #endregion
    #region Case de dos a mas elementos
    if(Head != null)
    {
        newNode.SetNext(Head);
        Head.SetPrev(newNode);

        Head = newNode;
        Count++;
    }
    #endregion
}
public Node<T> Find(T target , Node<T> start, int depth = 1000)
{
    if (start == null || depth <= 0) return null;//-> target no exista en la lsita doblemente enlazada

    if(start.Value.Equals(target))
    {
        return start;//-existe se encuentra y los retrno
    }


    return Find(target, start.Next, depth - 1);//->Analize el siguiente y reduzca el depth en 1
}
```

```
public virtual void ReadForward(Node<T> value,int depth =1000)
{
    if (value == null || depth <=0) return;

    Debug.Log(value.Value.ToString());

    if(value.Next != null)
        ReadForward(value.Next, depth - 1);
}
public virtual void ReadBackward(Node<T> value, int depth = 1000)
{
    if (value == null || depth <= 0) return;

    Debug.Log(value.Value.ToString());

    if (value.Prev != null)
        ReadBackward(value.Prev, depth - 1);
}

public void InsertAfter(T target, Node<T> value)
{
    /*if (target == Tail)
    {
        value.SetPrev(Tail);
        value.SetNext(null);

        Tail.SetNext(value);

        Tail = value;
    }*/
    Node<T> temp = Find(target ,Head);
    if (temp == null)
    {
```

```
            Debug.LogError("NULLO");
            return;
        }
        if(temp.Next != null)
        {
            value.SetNext(temp.Next);
            value.SetPrev(temp);

            temp.Next.SetPrev(value);
            temp.SetNext(value);

        }
        else
        {
            value.SetPrev(temp);
            value.SetNext(null);

            temp.SetNext(value);

            Tail = value;
        }

    }
    public void Delete(T target)
    {
        if(Head.Value.Equals(target)) //-> que se busca eliminar la cabeza
        {
            Head = Head.Next;
            Head.Prev.SetNext(null);
            Head.SetPrev(null);

            Count--;
            return;
```

```
        }
        if (Tail.Value.Equals(target))//->Eliminacion al final
        {
            Tail = Tail.Prev;
            Tail.Next.SetPrev(null);
            Tail.SetNext(null);

            Count--;
            return;

        }
        //-Eliminacion en el medio
        Node<T> temp = Find(target, Head);
        if (temp == null) return;

        temp.Prev.SetNext(temp.Next);
        temp.Next.SetPrev(temp.Prev);

        temp.SetNext(null);
        temp.SetPrev(null);

        Count--;

    }

}

public class CustomDoubleLinkedList : DoubleLinkedList<Dialogo>
{

    public IEnumerator ReadText(Node<Dialogo> nodeRead, TextMeshProUGUI TextHudName,
TextMeshProUGUI TextHUDDialogue)
    {
```

```
        GameManager.Instance.Changebool(true);
        string TextToRead = nodeRead.Value.Texto;
        TextHudName.text = nodeRead.Value.Name;
        TextHUDDialogue.text = "";
        for (int i = 0; i < TextToRead.Length; i++)
        {
            TextHUDDialogue.text += TextToRead[i];
            yield return new WaitForSeconds(0.05f);
        }
       GameManager.Instance. Changebool(false);
    }


}public class GameManager : MonoBehaviour
{
    public static GameManager Instance;
    public CustomDoubleLinkedList list = new CustomDoubleLinkedList();
    public TextMeshProUGUI TextHudName;
    public TextMeshProUGUI Textdialo;
    public Node<Dialogo> current;
    public bool OnWrite = false;
    public GameObject Hud;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(this);
        }
        else
        {
            Destroy(this);
        }
```

```
    }
    void Start()
    {
        list.AddNode(new Dialogo("chara", ".....", Emotion.Happy,0.1f));
        list.AddNode(new Dialogo("Sans", "Es un dia hermoso alla afuera", Emotion.Happy,0.3f));
        list.AddNode(new Dialogo("Sans", "Las aves cantan, las flores florecen....",
Emotion.Happy,0.1f));
        list.AddNode(new Dialogo("Sans", "Asi niños como tu .....", Emotion.Angry,0.2f));
        list.AddNode(new Dialogo("Sans", "deberian arder en el infierno", Emotion.Angry, 0.4f));
        current = list.Tail;
        Read();
    }
    public void Read()
    {
        if (current == null) return;
        StartCoroutine(list.ReadText(current,TextHudName,Textdialo));
    }
   public void Changebool(bool change)
    {
        OnWrite = change;
    }
    public void Next()
    {
        if (current != null && current.Next != null && OnWrite == false)
        {
            OnWrite = true;
            current = current.Next;
            Read();
        }
    }
    public void Prev()
    {
        if (current != null && current.Prev != null && OnWrite == false)
```

```
        {
            OnWrite = true;
            current = current.Prev;
            Read();

        }

    }
}
```

Conclusiones: