Universidad
Nacional Autónoma
de México

Sociedad de Desarrollo de Videojuegos

Proyecto del videojuego: Leyendas del metro

Equipo 12

Javier Enrique Díaz Rivera-Manager
Andrés Mauricio Lombana Bermudez- Artista
Martin Gabriel Ramirez Hilario-Coder

Descripción del juego:

El juego “Leyendas del Metro” es un Endless runner en 2D inesperado en las diferentes criaturas de las leyendas urbanas que habitan en el metro de la Ciudad de México.

Dónde el jugador tomará el rol del conductor del metro, dicho conductor debe de sobrevivir a las diferentes criaturas que saldrán de las profundidades del metro para atacar a nuestro jugador; estás criaturas son “La rata Gigante”, “El vampiro de Barranca” y “La niña sin cabeza”, y entre más tiempo sobreviva, mayor será su puntaje.

Una vez iniciado el programa se le mostrará al usuario una primera pantalla en dónde estarán 3 botones, el botón de “Iniciar”, “Opciones” y “Salir”.

Si se elige el botón de “Salir”, el programa se cerrará automáticamente.

Si se escoge el botón de Opciones, se abrirá un menú de opciones, en dónde podrá manipular el volumen del juego con una barra interactiva; igualmente se puede manipular el brillo con otra barra del mismo tipo, y el brillo que se desee tener, se guardará y así se mantendrá hasta que se vuelva a manipular.

Por último el menú de “Opciones” tiene 2 botones hasta abajo en dónde se podrá elegir si el juego se quiere en pantalla completa o no, y el último botón será para volver a la pantalla inicial.

Ahora, si se escoge el botón de “Iniciar”, cambiará la pantalla a la escena del juego principal en dónde empezará a ejecutar el juego.

El juego está ambientado dentro de los túneles del metro, en dónde la vista del jugador está fuera del convoy, por lo que podrá ver llegar los enemigos aleatorios que vayan apareciendo en su camino; en cuanto se inicie el juego estará ubicado en el carril central, una vez que vea al enemigo acercarse tendrá 3 opciones, quedarse en su mismo carril, moverse al carril izquierdo o al carril derecho con las flechas del teclado.

A cualquier dirección que se desee ir el convoy se moverá de carril, pero dichos movimientos están limitados a las direcciones ya mencionadas.

Mientras el juego está corriendo y el jugador no haya muerto, tendrá disponible un botón que dirá “Pausa” y un contador que irá aumentando siempre y cuando no choque con algún enemigo; si se presiona dicho botón se le abrirá un menú de opciones; este menú tendrá dentro otros tres botones a elegir, uno de “Reanudar”, “Reiniciar” y ”Menú principal”.

Si se decide presionar el botón de “Reanudar”, desaparecerá automáticamente el menú con dichos botones y se seguirá ejecutando el juego en el momento que el usuario haya de decido pausarlo, dejando disponible el botón de “Pausa” y seguirá aumentando su contador.

Ahora bien, si el usuario decide presionar el botón “Reiniciar” se borra el progreso que tenga el usuario en su contador y volverá al punto de inicio como si hubiera iniciado desde la pantalla inicial.

Por último si se presiona el otro botón de “ Menú Principal”, se hará un cambio de escena el cual llevará a la primera pantalla que se le mostró al jugador antes de iniciar el juego.

Por otro lado, si se elige la primera opción, el jugador morirá inmediatamente que su personaje choque con el enemigo; cuando esto suceda el juego se detendrá y aparecerá una pantalla de “Game Over”, en dónde tendrá dos botones como opciones, “Reiniciar” el juego o Volver al “Menú principal”.

Y como ya se menciono antes, cada botón realiza las mismas acciones que los botones con el mismo nombre, dejando al usuario escoger que botón oprimir.

Explicación de los scrips:

“AparicionEnemigo”:

Este script se llama “AparicionEnemigo” y su propósito es instanciar (crear) un objeto (por ejemplo, un enemigo) en la posición del objeto al que está adjunto el script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas importan las bibliotecas necesarias:
- System.Collections y System.Collections.Generic se utilizan para trabajar con colecciones como listas y arrays.
- UnityEngine es la biblioteca principal de Unity que proporciona acceso a todas las funciones básicas de Unity.

```
public class AparicionEnemigo : MonoBehaviour
{
    public GameObject objeto;
    // Start is called before the first frame update
    void Start()
    {
```

```
Instantiate(objeto,transform.position,Quaternion.identity)
;
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

1. Declaración de la Clase

```
    public class AparicionEnemigo : MonoBehaviour
```

La clase “AparicionEnemigo” hereda de “MonoBehaviour”, lo que significa que es un script de Unity que puede ser adjuntado a un GameObject.

2. Declaración de la Variable

```
    public GameObject objeto;
```

Esta línea declara una variable pública de tipo GameObject llamada objeto. Esta variable se puede asignar en el editor de Unity, permitiendo que cualquier

objeto de juego (por ejemplo, un enemigo) pueda ser instanciado.

3. Método Start

```
    void Start()
    {
       Instantiate(objeto, transform.position,
Quaternion.identity);
    }
```

El método “Start” se llama antes del primer frame update, es decir, justo al inicio de la ejecución del juego. Dentro de este método se utiliza la función “Instantiate” para crear una instancia del `objeto` en la posición del objeto que tiene este script (“transform.position”). “Quaternion.identity” se utiliza para mantener la rotación predeterminada del objeto.

“Destructor”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas importan los espacios de nombres necesarios para el script. “System.Collections” y “System.Collections.Generic” son bibliotecas estándar de C# para colecciones, y `UnityEngine` es la biblioteca

principal de Unity que contiene las clases y funciones necesarias para trabajar con el motor de juego.

```
public class Destructor : MonoBehaviour
{
    public float tiempo;
```

Define una clase pública llamada “Destructor” que hereda de “MonoBehaviour”. Dentro de esta clase, se declara una variable pública de tipo “float`” llamada “tiempo”. Esta variable se puede configurar desde el editor de Unity, lo que permite especificar el tiempo después del cual se destruirá el objeto al que está adjunto el script.

```
    // Start is called before the first frame update
    void Start()
    {
        Destroy(gameObject,tiempo);
    }
```

El método “Start” es un método especial en Unity que se llama una vez al inicio del ciclo de vida del script, justo antes de que empiece el primer frame de la actualización. Dentro de este método, se llama a la función “Destroy”, que es una función de Unity para destruir objetos de juego. Aquí, “Destroy(gameObject, tiempo)” indica que el objeto de juego (“gameObject”) al

que está adjunto este script será destruido después de un tiempo especificado por la variable “tiempo”.

“Generacion”

Este script en C# también está diseñado para ser utilizado dentro del motor de juego Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas importan los espacios de nombres necesarios para el script, similar al script anterior.

```
public class Generacion : MonoBehaviour
{
    public GameObject[] enemigos;
    public float tiempoGeneracion;
    public float decremento;
    public float limite;
    public float tiempoInicio;
    int numRandom;
```

Este bloque define una clase pública llamada “Generacion”, que también hereda de “MonoBehaviour”. Dentro de esta clase, se declaran varias variables públicas:

- “enemigos”: un arreglo de GameObjects que representan los enemigos que se generarán.

- “tiempoGeneracion”: un flotante que representa el tiempo entre generaciones de enemigos.
- “decremento”: un flotante que representa la cantidad en la que disminuirá el tiempo de generación después de cada generación.
- “limite”: un flotante que representa el tiempo mínimo entre generaciones.
- “tiempoInicio”: un flotante que representa el tiempo inicial entre generaciones.
- “numRandom”: un entero que almacenará un índice aleatorio para seleccionar un enemigo del arreglo “enemigos”.

```
    // Update is called once per frame
    void Update()
    {
        if(tiempoGeneracion<=0){

numRandom=Random.Range(0,enemigos.Length);

Instantiate(enemigos[numRandom],transform.position,Q
uaternion.identity);
            if(tiempoInicio>limite){
                tiempoInicio-=decremento;
            }

            tiempoGeneracion=tiempoInicio;
```

```
        }else {
            tiempoGeneracion-=Time.deltaTime;
        }
    }
}
```

El método “Update” es llamado una vez por frame y es donde ocurre la lógica principal del script:
- Comprueba si `”tiempoGeneracion” es menor o igual a cero. Si es así, se elige un índice aleatorio “numRandom” dentro del rango de índices de “enemigos”. Luego, se instancia un objeto de enemigo en la posición actual del objeto al que está adjunto este script (representado por “transform.position”) con una rotación predeterminada (“Quaternion.identity”). Después, comprueba si el “tiempoInicio” es mayor que el “limite”. Si lo es, disminuye el “tiempoInicio” por el valor de “decremento”. Finalmente, restablece el valor de “tiempoGeneracion” a “tiempoInicio”.
- Si “tiempoGeneracion” no es menor o igual a cero, se le resta al “tiempoGeneracion” el tiempo que ha pasado desde el último frame (“Time.deltaTime”), lo que hace que “tiempoGeneracion” disminuya gradualmente hasta llegar a cero, momento en el cual se genera un nuevo enemigo y se reinicia el ciclo.

“MovimientoEnemigo”:

Este script en C# también es para ser utilizado en Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas importan los espacios de nombres necesarios, al igual que en los scripts anteriores.

```
public class MovimientoEnemigo : MonoBehaviour
{
    public float tiempo;
    public float speed;
```

Aquí se define una clase pública llamada “MovimientoEnemigo” que hereda de “MonoBehaviour”. Dentro de esta clase, se declaran dos variables públicas:
- “tiempo”: un flotante que representa el tiempo después del cual el objeto de juego al que está adjunto este script será destruido.
- “speed”: un flotante que representa la velocidad a la que se moverá el objeto de juego.

```
    // Start is called before the first frame update
    void Start()
    {
        Destroy(gameObject, tiempo);
    }
```

El método “Start” es llamado antes del primer frame de actualización. En este caso, simplemente destruye el objeto al que está adjunto este script después de un tiempo especificado por la variable “tiempo”, utilizando la función “Destroy” de Unity.

```
    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector2.down * speed *
Time.deltaTime);
    }
}
```

El método “Update” es llamado una vez por frame y es donde ocurre la lógica principal del script. En este caso, mueve el objeto de juego hacia abajo (“Vector2.down”) a una velocidad determinada por la variable “speed”. Esto se hace utilizando la función “Translate” de Unity, que desplaza el objeto en el espacio. La velocidad se multiplica por “Time.deltaTime” para hacer el movimiento independiente de la velocidad de fotogramas y asegurar un movimiento suave y constante incluso en diferentes sistemas.

“PersonajeMovimiento”:

Este script controla el movimiento horizontal de un personaje en un juego de Unity.

```
using UnityEngine;
```

Esta línea importa el espacio de nombres necesario para acceder a las clases y funciones de Unity.

```
public class PersonajeMovimiento : MonoBehaviour
{
    private Rigidbody2D rb;
    public float velocidad;
    public float maxX;
    public float minX;
    public float XIncrement;
    [SerializeField] private GameObject menuGameOver;
```

Este bloque define una clase pública llamada “PersonajeMovimiento” que hereda de “MonoBehaviour”. Dentro de esta clase, se declaran las siguientes variables:
- “rb”: un campo privado que almacena una referencia al componente `Rigidbody2D` del objeto al que se adjunta el script.
- “velocidad”: un flotante que representa la velocidad de movimiento horizontal del personaje.
- “maxX” y “minX”: flotantes que definen los límites máximo y mínimo en el eje X para el movimiento del personaje.

- “XIncrement”: un flotante que define el incremento en el eje X.
- “menuGameOver”: un GameObject que representa el menú de juego sobre la pantalla de Game Over.

```
    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>(); // Obtiene el componente Rigidbody2D al inicio
    }
```

El método “Start” se llama antes del primer fotograma de la actualización. En este caso, obtiene una referencia al componente “Rigidbody2D” del objeto al que está adjunto el script y lo almacena en la variable “rb”.

```
    // Update is called once per frame
    void Update()
    {
        MoverEnX();
    }
```

El método “Update” se llama una vez por frame y llama al método “MoverEnX()”, que se encarga del movimiento horizontal del personaje.

```
    void OnCollisionEnter2D(Collision2D collision)
```

```
    {
        if (collision.gameObject.tag == "Enemigo")
        {
            Destroy(gameObject);
            Destroy(collision.gameObject);
            Time.timeScale = 0f; // Pausa el juego
            menuGameOver.SetActive(true);
        }
    }
```

Este método se llama cuando el objeto colisiona con otro objeto en 2D. Si la etiqueta del objeto con el que colisiona es "Enemigo", destruye tanto al personaje como al enemigo, pausa el juego y activa el menú de Game Over.

```
    void MoverEnX()
    {
        float inputMovimiento = Input.GetAxis("Horizontal");
// Obtiene el input horizontal

        // Aplica la velocidad en el eje X
        rb.velocity = new Vector2(inputMovimiento *
velocidad, rb.velocity.y);

        // Calcula la nueva posición en X
        float newXPosition = transform.position.x +
inputMovimiento * velocidad * Time.deltaTime;
```

```
        // Limita la posición en X dentro del rango
        newXPosition = Mathf.Clamp(newXPosition, minX,
maxX);

        // Aplica la nueva posición
        transform.position = new Vector2(newXPosition,
transform.position.y);
    }
```

Este método controla el movimiento horizontal del personaje. Primero, obtiene el valor del input horizontal con “Input.GetAxis("Horizontal")”. Luego, utiliza este valor para calcular la velocidad en el eje X del personaje y aplicarla al componente “Rigidbody2D” (“rb.velocity”). A continuación, calcula la nueva posición en el eje X y la limita dentro del rango especificado por “minX” y “maxX” utilizando “Mathf.Clamp”. Finalmente, aplica la nueva posición al objeto del personaje.

“CanvasStart”:

Este script está diseñado para manejar la lógica relacionada con la interfaz de usuario en la pantalla de inicio de un juego en Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
```

```
using UnityEngine.Audio;
using UnityEngine.UI;
```

Estas líneas importan los espacios de nombres necesarios para acceder a las clases y funciones de Unity, así como las bibliotecas de Unity relacionadas con el audio y la interfaz de usuario.

```
public class CanvasStart : MonoBehaviour
{
   [SerializeField] private GameObject menuOpciones;
   [SerializeField] private AudioMixer audioMixer;
   public Slider slider;
   public float sliderValor;
   public Image panelBrillo;
```

Aquí se define una clase pública llamada “CanvasStart” que hereda de “MonoBehaviour”. Dentro de esta clase, se declaran varias variables públicas y privadas:
- “menuOpciones”: un GameObject que representa el menú de opciones.
- “audioMixer”: un AudioMixer que controla el volumen de audio del juego.
- “slider”: un Slider que permite al jugador ajustar el brillo del juego.
- “sliderValor”: un flotante que almacena el valor actual del slider.

- “panelBrillo”: una imagen que representa el panel de brillo en la interfaz de usuario.

```
    public void Jugar(){
        PlayerPrefs.SetFloat("brillo", slider.value);
        SceneManager.LoadScene(1);
        slider.onValueChanged.AddListener(delegate {
CambiarBrillo(slider.value); });
    }
```

El método “Jugar()” se llama cuando el jugador presiona el botón de "Jugar". Guarda el valor actual del slider (que representa el brillo) utilizando PlayerPrefs, carga la escena principal del juego y añade un listener al evento “onValueChanged” del slider para llamar al método “CambiarBrillo()” cada vez que el valor del slider cambie.

```
    public void Salir(){
        Debug.Log("Se cierra el juego");
        Application.Quit();
    }
```

El método “Salir()” se llama cuando el jugador presiona el botón de "Salir". Simplemente cierra la aplicación.

```
    public void Opciones(){
        menuOpciones.SetActive(true);
    }
```

El método “Opciones()” se llama cuando el jugador presiona el botón de "Opciones". Activa el menú de opciones.

```
public void BotonRegresar(){
    menuOpciones.SetActive(false);
}
```

El método “BotonRegresar()” se llama cuando el jugador presiona el botón de "Regresar" en el menú de opciones. Desactiva el menú de opciones.

```
public void PantallaCompleta(bool pantallaCompleta){
    Screen.fullScreen=pantallaCompleta;
}
```

El método `PantallaCompleta()` se llama cuando el jugador cambia la opción de pantalla completa en el menú de opciones. Establece el modo de pantalla completa según el valor booleano pasado como parámetro.

```
public void CambiarVolumen(float volumen){
    audioMixer.SetFloat("Volumen",volumen);
}
```

El método “CambiarVolumen()” se llama cuando el jugador ajusta el control deslizante de volumen en el

menú de opciones. Cambia el volumen del audio del juego utilizando el AudioMixer.

```
    public void CambiarBrillo(float valor){
        sliderValor=valor;
        PlayerPrefs.SetFloat("brillo",sliderValor);
        panelBrillo.color=new
Color(panelBrillo.color.r,panelBrillo.color.g,panelBrillo.color.b,slider.value);
    }
```

El método “CambiarBrillo()” se llama cuando el jugador ajusta el control deslizante de brillo en el menú de opciones. Cambia el brillo del juego ajustando el color del panel de brillo.

```
    void Start(){
        slider.value=PlayerPrefs.GetFloat("brillo",0.5f);
        panelBrillo.color=new
Color(panelBrillo.color.r,panelBrillo.color.g,panelBrillo.color.b,slider.value);

        slider.onValueChanged.AddListener(delegate {
CambiarBrillo(slider.value); });
    }
```

El método “Start()” se llama antes del primer frame de la actualización. Establece el valor del slider de brillo al valor guardado en PlayerPrefs (o a un valor

predeterminado de 0.5f si no hay ningún valor guardado) y ajusta el color del panel de brillo en consecuencia. Además, añade un listener al evento “onValueChanged” del slider para llamar al método “CambiarBrillo()” cada vez que el valor del slider cambie.

“BrilloJuego”:

Este script, llamado “BrilloJuego”, se encarga de ajustar el brillo de un panel en el juego basado en el valor guardado en PlayerPrefs.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

Estas líneas importan los espacios de nombres necesarios para acceder a las clases y funciones de Unity, así como para trabajar con componentes de interfaz de usuario (UI).

```
public class BrilloJuego : MonoBehaviour
{
    public Image panelBrillo;
```

Aquí se define una clase pública llamada “BrilloJuego” que hereda de “MonoBehaviour”. Dentro de esta clase,

se declara una variable pública de tipo “Image” llamada “panelBrillo”, que representa el panel de brillo en la interfaz de usuario.

```
    // Start is called before the first frame update
    void Start()
    {
        // Recuperar el valor del brillo guardado y aplicarlo
        float sliderValor = PlayerPrefs.GetFloat("brillo",
0.5f);
        panelBrillo.color = new Color(panelBrillo.color.r,
panelBrillo.color.g, panelBrillo.color.b, sliderValor);
    }
```

El método “Start()” se llama antes del primer frame de la actualización. En este caso, recupera el valor del brillo guardado en PlayerPrefs utilizando la clave "brillo". Si no se encuentra ningún valor guardado, se utiliza un valor predeterminado de 0.5f. Luego, aplica este valor al componente “color” del panel de brillo, ajustando su transparencia para controlar el brillo.

MovimientoFondo:

Este script, llamado “MovimientoFondo”, controla el movimiento de un material en un objeto de fondo en Unity. Aquí tienes una explicación detallada del mismo:

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
```

Estas líneas importan los espacios de nombres necesarios para acceder a las clases y funciones de Unity.

```
public class MovimientoFondo : MonoBehaviour
{
    [Range(-2f,2f)]
    public float velocidad = 1f;
    private float offset;
    private Material mat;
```

Aquí se define una clase pública llamada “MovimientoFondo” que hereda de “MonoBehaviour”. Dentro de esta clase, se declaran las siguientes variables:
- “velocidad”: un flotante que representa la velocidad a la que se moverá el fondo. La anotación “[Range(-2f,2f)]” especifica que la velocidad debe estar en el rango de -2 a 2, lo que permite ajustarla desde el editor de Unity.
- “offset”: un flotante que representa el desplazamiento actual del fondo.
- “mat”: un objeto “Material” que representa el material del objeto al que está adjunto este script.

```
    // Start is called before the first frame update
    void Start()
    {
        mat = GetComponent<Renderer>().material;
    }
```

El método “Start()” se llama antes del primer fotograma de la actualización. En este caso, obtiene una referencia al componente “Material” del objeto al que está adjunto este script y lo almacena en la variable “mat”.

```
    // Update is called once per frame
    void Update()
    {
        offset += (Time.deltaTime * velocidad) / 10f;
        mat.SetTextureOffset("_MainTex", new Vector2(0,
offset));
    }
```

El método “Update()” se llama una vez por frame y es donde ocurre la lógica principal del script. En este caso, calcula el nuevo valor del `offset` del fondo multiplicando la velocidad por el tiempo transcurrido desde el último frame (“Time.deltaTime”). Dividir por 10f es una forma de ajustar la velocidad a un valor más apropiado para el movimiento del fondo. Luego, utiliza el método “SetTextureOffset()” del material para ajustar el desplazamiento del fondo en la textura principal

(`"_MainTex"`) en el eje Y según el nuevo valor del `offset`.

“Puntaje”:

-“public TMP_Text puntuTex;”: Esta línea declara una variable pública llamada "puntuTex" que es del tipo “TMP_Text”. Esto significa que puedes arrastrar un objeto de texto de TextMeshPro desde tu escena de Unity directamente a esta variable en el inspector. De esta manera, el script puede acceder y modificar el texto que se muestra en ese objeto.

-“public float contador;”: Aquí se declara una variable pública llamada "contador" que es de tipo “float”. Esta variable va a almacenar el puntaje del jugador. Al ser pública, esta variable puede ser vista y modificada desde el inspector de Unity si se desea.

-”void Update()”: Este método es parte del ciclo de vida de los objetos en Unity y se llama en cada frame del juego. Dentro de este método, se verifica si hay algún objeto en la escena con la etiqueta "Player". Si hay al menos uno, se procede a actualizar el puntaje.

-”contador += 1 * Time.deltaTime;”: Esta línea incrementa el contador de puntaje. “Time.deltaTime” representa la cantidad de tiempo que pasó desde el último frame. Multiplicar esto por 1 asegura que el

incremento sea proporcional al tiempo transcurrido, es decir, el puntaje aumentará de forma constante independientemente de la velocidad del juego.

-”puntuTex.text = ((int)contador).ToString();”: Aquí se actualiza el texto que se muestra en el objeto “TMP_Text”. Primero, el valor del contador se convierte a un entero mediante “(int)contador”, ya que probablemente querrás mostrar el puntaje como un número entero. Luego, se convierte ese entero a una cadena de texto mediante “ToString()”, y se asigna como el texto que se mostrará en el objeto “TMP_Text’.

“PausarJuego”:

Este script en C# está diseñado para gestionar la pausa y la resolución del juego en Unity.

-”using System.Collections;”: Esta línea importa el espacio de nombres System.Collections, que proporciona tipos y clases para trabajar con colecciones de datos en C#.

-”using System.Collections.Generic;”: Similar al anterior, esta línea importa el espacio de nombres System.Collections.Generic, que proporciona tipos y clases para trabajar con colecciones genéricas en C#.

-”using UnityEngine;”: Esta línea importa el espacio de nombres de Unity, que contiene las clases y funciones fundamentales para el desarrollo de juegos en Unity.

-”using UnityEngine.SceneManagement;”: Importa el espacio de nombres de SceneManager de Unity, que permite cargar, descargar y gestionar escenas en el juego.

-”public class PausarJuego: MonoBehaviour”: Define una clase pública llamada PausarJuego que hereda de MonoBehaviour, lo que significa que puede ser adjuntada como componente a un GameObject en Unity.

-”[SerializeField] private GameObject bottonPausa;”: Declara una variable privada serializable llamada "bottonPausa" que referencia a un GameObject. Este GameObject probablemente es un botón que activa la pausa en el juego.

-”[SerializeField] private GameObject menuPausa;”: Similar al anterior, esta línea declara una variable privada serializable llamada "menuPausa" que referencia a un GameObject. Este GameObject probablemente es un menú de pausa que se activa cuando se pausa el juego.

-”private bool juegoPausaTecla=false;”: Declara una variable privada de tipo booleano llamada "juegoPausaTecla" e inicializa su valor en falso. Esta variable se utiliza para controlar si el juego está pausado o no.

-”public void Update()”: Define un método público llamado "Update", que es llamado en cada frame del juego.

-”if(Input.GetKeyDown(KeyCode.Escape)){“: Verifica si se ha presionado la tecla de escape (ESC) en el teclado.

-”if(juegoPausaTecla){ Reanudar(); }else{ Pausa(); }”: Si el juego está pausado (la variable "juegoPausaTecla" es verdadera), llama al método "Reanudar()" para continuar el juego. De lo contrario, llama al método "Pausa()" para pausar el juego.

-”public void Pausa()”: Define un método público llamado "Pausa" que se encarga de pausar el juego.

-”juegoPausaTecla=true;”: Establece la variable "juegoPausaTecla" en verdadero, indicando que el juego está pausado.

-”Time.timeScale=0f;”: Establece el tiempo de escala del juego a cero, lo que detiene todos los movimientos y acciones en el juego.

-”bottonPausa.SetActive(false);”: Desactiva el GameObject "bottonPausa", probablemente un botón que activa la pausa.

-”menuPausa.SetActive(true);”: Activa el GameObject "menuPausa", que es el menú de pausa.

“public void Reanudar()”: Define un método público llamado "Reanudar" que se encarga de reanudar el juego.

-”juegoPausaTecla=false;”: Establece la variable "juegoPausaTecla" en falso, indicando que el juego ya no está pausado.

-”Time.timeScale=1f;”: Establece el tiempo de escala del juego a uno, lo que reanuda el juego a su velocidad normal.

-“bottonPausa.SetActive(true);”: Activa el GameObject "bottonPausa", que es el botón para activar la pausa.

-”menuPausa.SetActive(false);”: Desactiva el GameObject "menuPausa", que es el menú de pausa.

-”public void Reiniciar()”: Define un método público llamado "Reiniciar" que se encarga de reiniciar el juego.

-”Time.timeScale=1f;”: Establece el tiempo de escala del juego a uno, asegurando que el juego esté funcionando a su velocidad normal antes de reiniciar.

-”SceneManager.LoadScene(SceneManager.GetActiveScene().name);”: Carga la escena actual, reiniciando así el juego.

“bottonPausa.SetActive(true);”: Activa el GameObject "bottonPausa" después de reiniciar el juego.

-”menuPausa.SetActive(false);”: Desactiva el GameObject "menuPausa" después de reiniciar el juego.

-”public void MenuPrincipal()”: Define un método público llamado "MenuPrincipal" que se encarga de cargar la escena del menú principal.

-“SceneManager.LoadScene(0);”: Carga la escena con el índice 0, que probablemente sea la escena del menú principal.

Comentarios:

Desarrollar este juego fue una experiencia increíble que me permitió crecer como desarrollador de juegos y como persona. Me siento muy orgulloso del trabajo que he realizado y emocionado por las oportunidades futuras que me esperan en el mundo del desarrollo de juegos.