

Desarrollo de un videojuego de plataforma 2D en Unity “MathBoy”



FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA EN SISTEMAS
COMPUTACIONALES

Alumnos:

- Roger Liñan Burgos - N00100364
- Agurto Urcia, Andy - N00042198
- Mia Aguirre Edwar - N00296299
- Alvarado Varas Johan - N00233617
- Venturo Nuñez Manuel - N00283668

Docente: *Mitchell Paulo
Blancas Nuñez*
Curso: *Desarrollo de
Videojuegos*

ÍNDICE

INTRODUCCIÓN.....	2
Descripción del proyecto.....	2
Descripción Técnica.....	2
DISEÑO DEL JUEGO.....	3
Gameplay General y Mecánicas Principales.....	3
Objetivos.....	3
JUGADOR.....	4
INTERFAZ DE USUARIO.....	5
SISTEMAS.....	6
DIAGRAMA DE FLUJO.....	8
● Menú principal:.....	9
● Iniciar:.....	9
● Inicio de videojuego:.....	9
● Menú de pausa:.....	10
● Muerte o Victoria:.....	10
● Reglas:.....	10
● Opciones:.....	10
● Salir:.....	10
ENEMIGO.....	11
DISEÑO DE NIVELES.....	12
Descripción General de los Niveles.....	12
Mecánicas de juegos asociadas.....	13
Descripción General del Diseño Artístico.....	15
PERSONAJES.....	18
MathBoy.....	18
Animaciones.....	18
Cerebrito (NPC) :	21
Animaciones:.....	21
Código:.....	22
REFERENCIAS.....	27

INTRODUCCIÓN

Descripción del proyecto

MathBoy es un juego de plataformas donde el objetivo del jugador es encontrar, a través de la exploración del nivel, la alternativa correcta a la pregunta para poder completar el nivel y seguir avanzando, teniendo en cuenta el tiempo, obstáculos y el enemigo.

Nuestro protagonista llamado MathBoy, iniciará en un escenario donde tendrá que buscar la respuesta correcta a la operación matemática que se le ha pedido resolver, ya que se le mostrará varias opciones, de las cuales varias serán incorrectas. Cuando haya encontrado la alternativa correcta, pasará al siguiente nivel donde la dificultad de las preguntas aumentará.



Descripción Técnica

El juego se está desarrollando íntegramente en el motor Unity y la plataforma a la que va a ir diseñada es para computadoras con Windows 10.



DISEÑO DEL JUEGO

Gameplay General y Mecánicas Principales

EL protagonista avanza por el nivel evitando el ataque del enemigo y el nivel finaliza cuando se cumplen uno de los dos acontecimientos siguientes:

- Cuando llega a tocar la alternativa correcta.
- Cuando el personaje muere, ya sea cayendo al vacío, tocando la alternativa incorrecta, eliminado por el enemigo o se le termina el tiempo.

Se completa el nivel cuando el personaje llega a la alternativa correcta. Posteriormente se pasará al siguiente nivel donde aumentará la dificultad de las preguntas, al igual que la del enemigo hasta llegar al nivel 5.

El personaje puede perder por cuatro razones:

- Pierde cuando se queda sin corazones.
- Pierde cuando cae al vacío.
- Se termina el tiempo.
- Toca la alternativa incorrecta.

Objetivos

El objetivo del jugador para finalizar el nivel debe ser encontrar la alternativa correcta que estará esparcida por dicho nivel en el que se encuentre.

Si el objetivo del jugador es completar el juego, debe superar cada nivel mientras la dificultad va aumentando hasta completar el nivel 5.

JUGADOR

La vida del jugador cuenta con tres corazones, por lo tanto, dependiendo del ataque del enemigo, de seleccionar la alternativa incorrecta o caer por el precipicio, le quitará uno o la totalidad de los corazones.

Si el jugador muere, el nivel se reinicia y empieza desde el inicio del mapa.

El jugador puede ser atacado por el enemigo ya sea por los proyectiles o por entrar en contacto con él.

Si el jugador es alcanzado por un proyectil del enemigo, estará stuneado por unos segundos y posteriormente podrá volver a moverse.

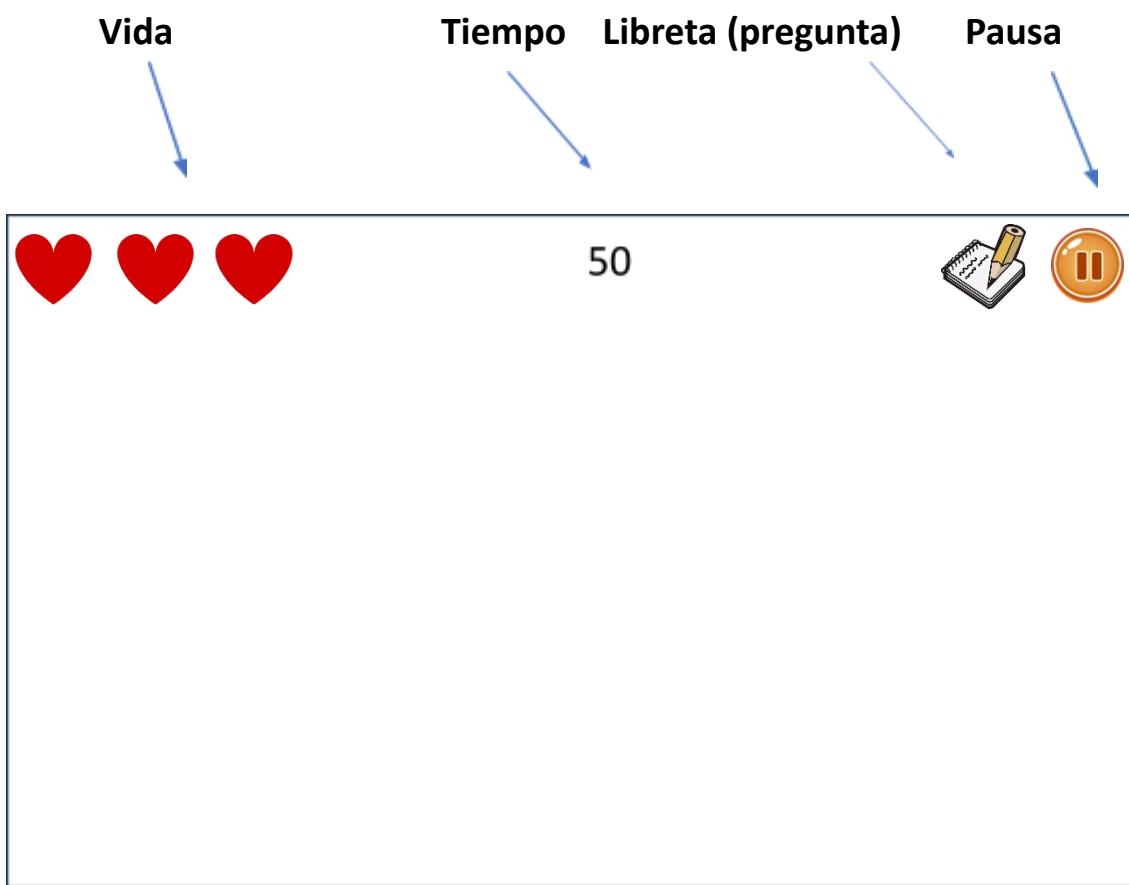
Para poder moverse por el escenario horizontalmente usará los botones A y D, y si pulsa el botón espacio podrá saltar y así podrá moverse verticalmente también.

Mientras nuestro jugador esté en el nivel puede poner pausa apretando el botón que aparecerá en la interfaz del nivel o presionando la tecla ESC. Al pulsarlo, le aparecerá un menú con la opción de reiniciar, continuar o volver al menú.



INTERFAZ DE USUARIO

A continuación, muestro la interfaz que tendrá el usuario siempre que esté en un nivel horizontal:



La jugabilidad será mediante teclas predeterminadas, se podrá mover por todo el escenario en busca de la alternativa correcta.

SISTEMAS

Sobre las vidas del personaje contará con 3 vidas cada que el enemigo le impacte reduce una

Estado normal:



Cuando le hacen daño:



Como hemos visto en la anterior página, el jugador tendrá la opción de volver a ver la pregunta que se mostró antes de iniciar el nivel justo en la parte superior derecha, de igual manera el botón de pausa, el cual hemos nombrado anteriormente que es para pausar el nivel en cualquier momento.

Además, como se mencionó anteriormente, el jugador debe usar las teclas A, S, D, W y saltar con la tecla ESPACIO, para poder moverse libremente por el nivel.



Así mismo, podrá consultar la pregunta desde el botón derecho superior en forma de lápiz.



Al momento de subir una escalera, este podrá saltar desde ella o caerse como se muestra en la imágenes:



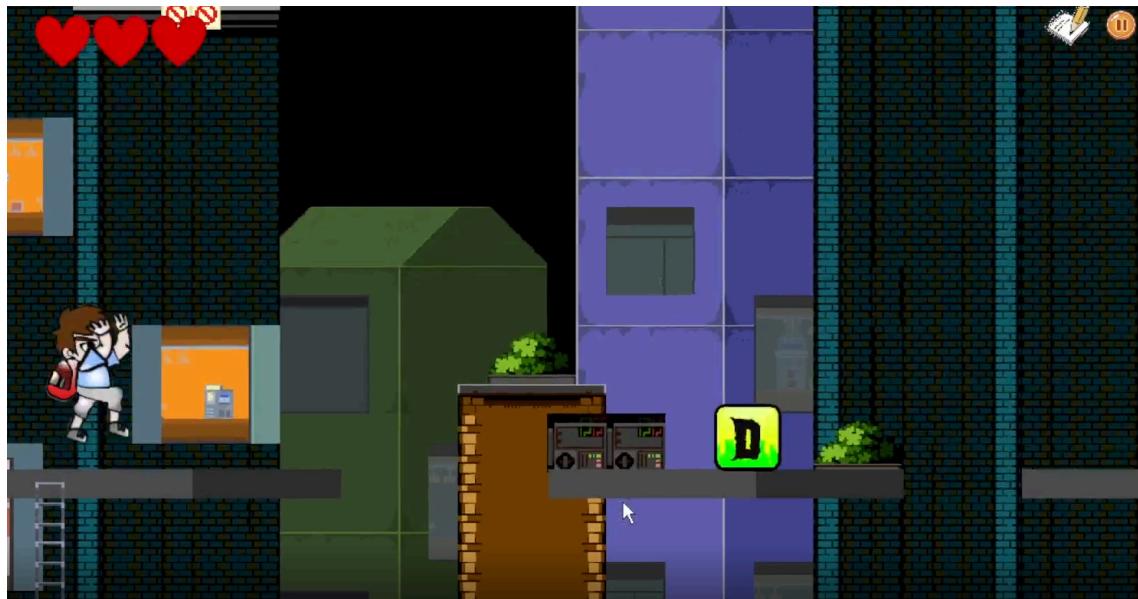
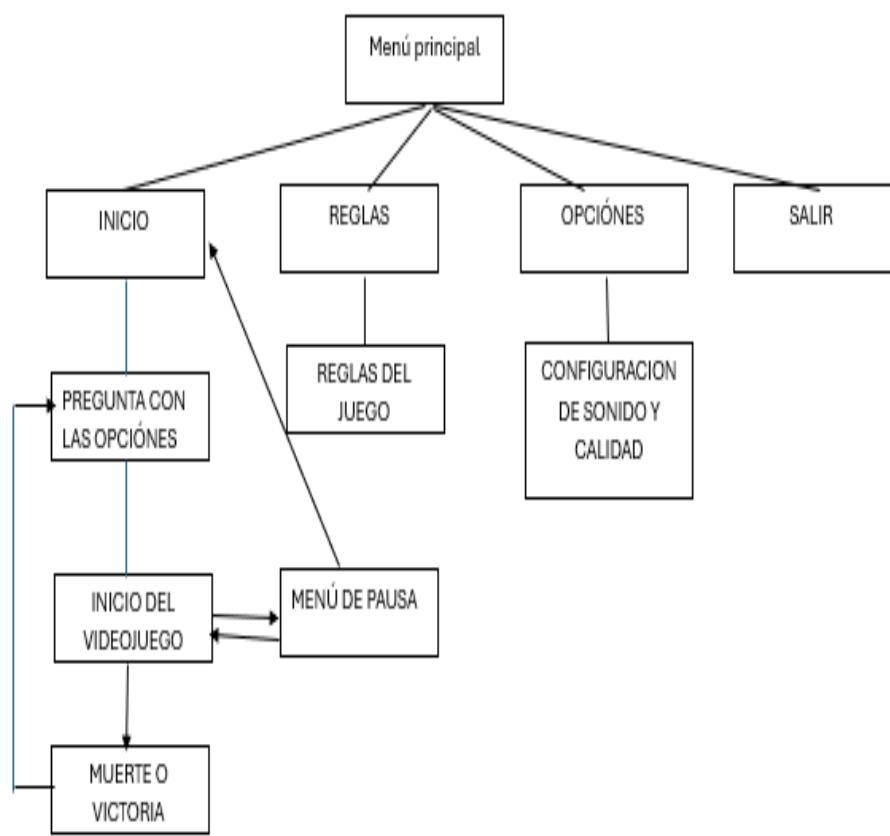
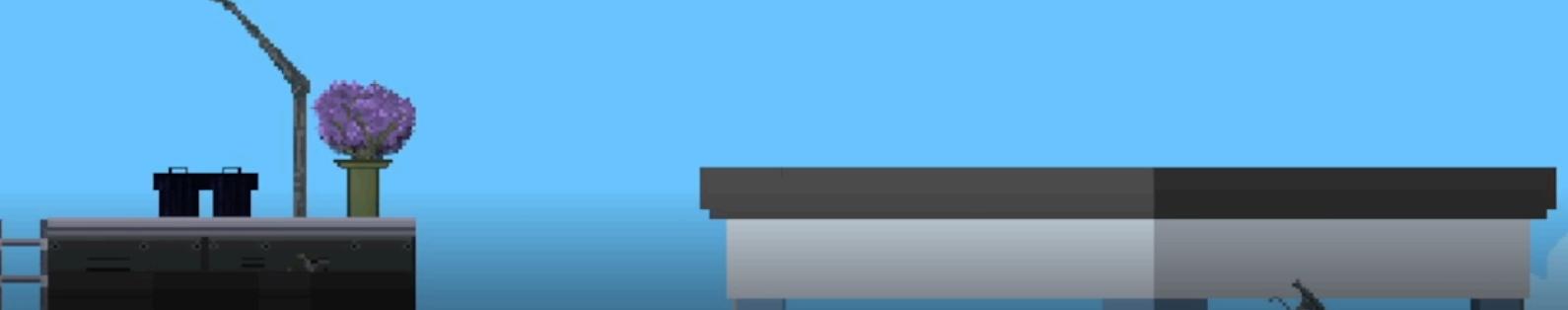


DIAGRAMA DE FLUJO



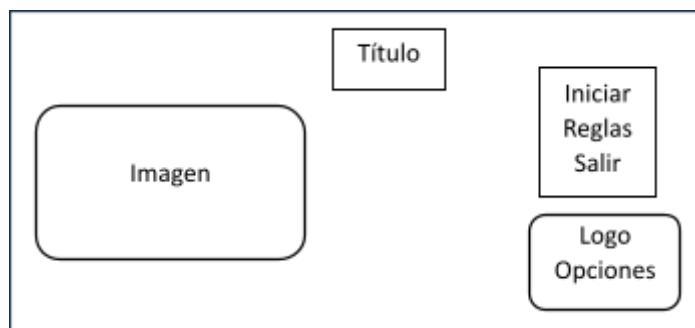


A continuación, explicamos el diagrama de flujo que hemos dado anteriormente.

- **Menú principal:**

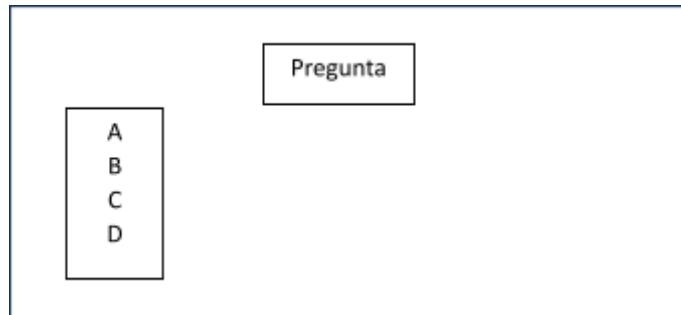
El menú principal consta de cuatro botones, los cuales al seleccionarlos nos llevará de un sitio a otro.

Los botones son: Iniciar, Reglas, Salir y el logo de opciones.



- **Iniciar:**

Al pulsar este botón nos dirigirá a la pregunta de lógica matemática con las 4 respuestas para elegir (A, B, C y D).



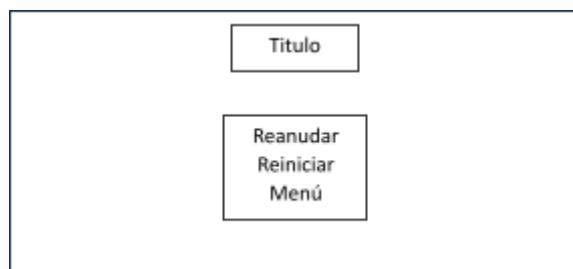
- **Inicio de videojuego:**

Se refiere al Gameplay del juego, es decir, estamos dentro de un nivel del juego, cuya interfaz está descrita anteriormente.



- Menú de pausa:

Durante el juego, podemos poner pausa cuando queramos y tenemos tres opciones, volver nivel por donde lo habíamos dejado, reiniciar el nivel o volver al menú de inicio.



- Muerte o Victoria:

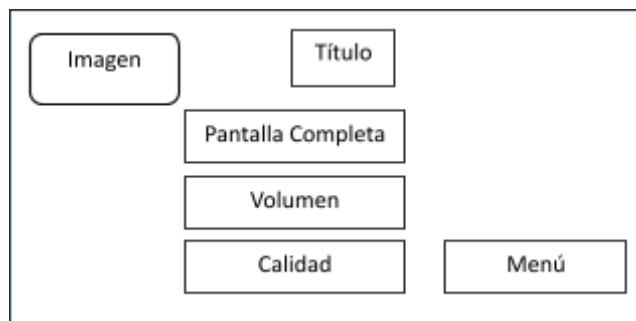
Aparecerá una pantalla donde las opciones serán: reiniciar el nivel si ha muerto o registrar su nombre en el ranking y volver al menú de niveles si ha ganado.

- Reglas:

Se mostrarán las reglas del juego.

- Opciones:

Al pulsar este botón nos dirigirá a una pantalla la cual muestra la opción de poner en modo pantalla completa así mismo una barra de volumen ajustable y calidad de imagen del juego, así como un botón para volver al menú principal.



- Salir:

Al pulsar este botón podremos salir del juego.

ENEMIGO

El Reprobado y sus bolas de papel

Habrá un enemigo de este tipo esparcidos por ciertos puntos del nivel.

El cual arroja bolas de papel con la intención de hacer que el personaje pierda y tenga que reiniciar el nivel.

Este se moverá por todo el mapa del nivel en el que esté en dirección de un punto A hacia un punto D.

No se podrá aniquilar al enemigo del nivel además que si enemigo golpea con sus bolas este le restará un corazón al jugador.



A medida que sube de nivel el enemigo lo seguirá al personaje siempre y cuando no lo mire.



DISEÑO DE NIVELES

Descripción General de los Niveles

Los niveles van a tener varias plataformas que se detallan más adelante, constará de múltiples escenarios en el cual estarán dispersadas las alternativas, es decir: (A, B, C, D). Los cuales también se describe a continuación.



Cada nivel presentará un problema al inicio de la partida, la cual será un problema de matemática, además que dará cuatro opciones de respuestas previamente mencionadas, dichas respuestas están dispersadas por todo el mapa y se deberá encontrar la correcta y evitar las incorrectas así mismo como al enemigo del nivel el cual será el reprobado.

Los niveles seguirán un patrón en el que el jugador podrá moverse libremente, es decir, que no se moverá simplemente hacia adelante y no podremos volver atrás.



Como se ha dicho anteriormente, para completar el nivel al 100% se necesitará escoger la alternativa correcta en los 5 niveles, los cuales irán aumentando su dificultad, tanto en la pregunta, como en la jugabilidad, ya que el enemigo cambiará su comportamiento mientras más cerca estés del último nivel.

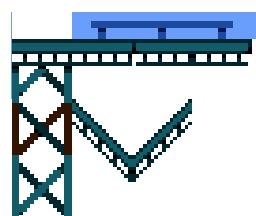
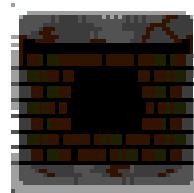
Además, cada nivel tendrá un contador regresivo de 60 segundos en la parte superior, que al llegar a 0, contará como una derrota directa y tendrá que reiniciar el nivel.

Mecánicas de juegos asociadas

Las plataformas por las que tendrá que saltar nuestro protagonista en los niveles del juego son las siguientes:

- **Suelo y Plataformas**

El suelo tiene un colisionador en ambos lados, por lo que, si el personaje intenta subir desde abajo, chocará contra él. A diferencia del suelo, las plataformas están diseñadas con un colisionador unidireccional en el eje Y, permitiendo que el personaje pueda atravesarlas desde abajo y aterrizar sobre ellas desde arriba.



- **Escaleras**

Son plataformas en las cuales el personaje se podrá trepar para poder llegar a otras zonas del mapa.





Durante el nivel el jugador podrá encontrarse con los siguientes objetos:

- **Respuestas:**

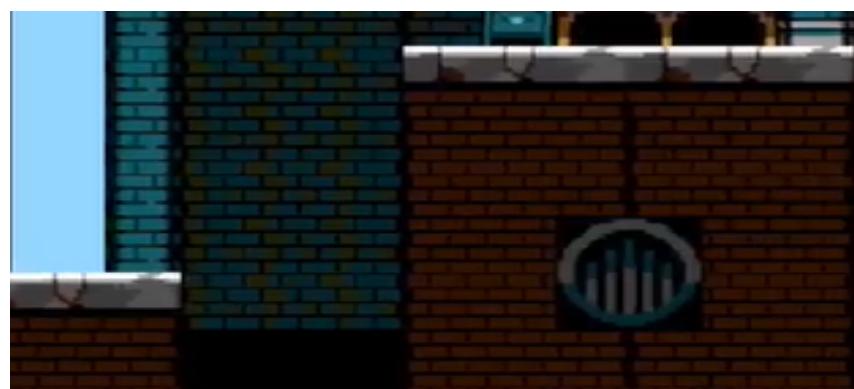
Al seleccionarlas, es decir, que el personaje se coloque en la posición de estos, perderá o ganará dependiendo si es o no la respuesta correcta.



A parte también podrá encontrarse con otros obstáculos:

- **Precipicio:**

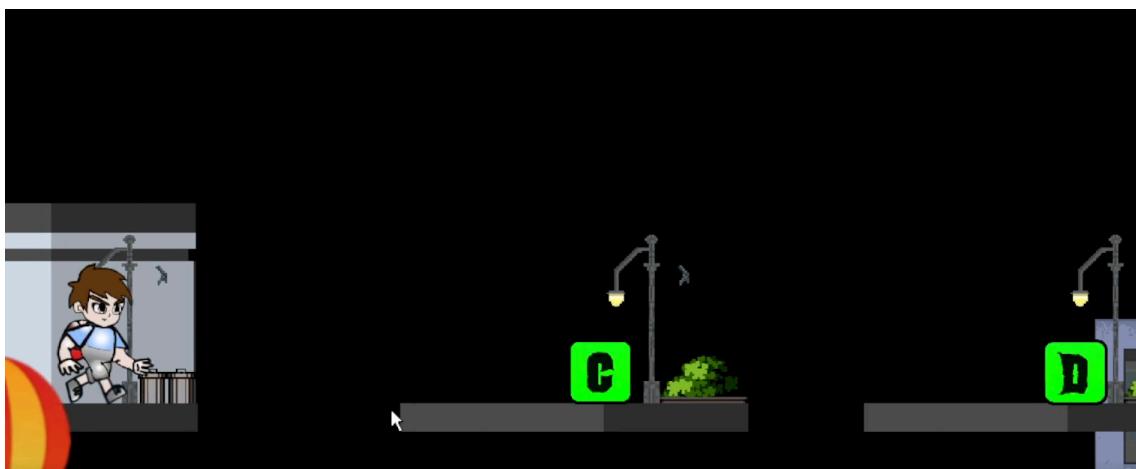
Serán un obstáculo más el cual al caer por estos, se dará por finalizado el juego, ya que se pierde la totalidad de los corazones.



ARTE

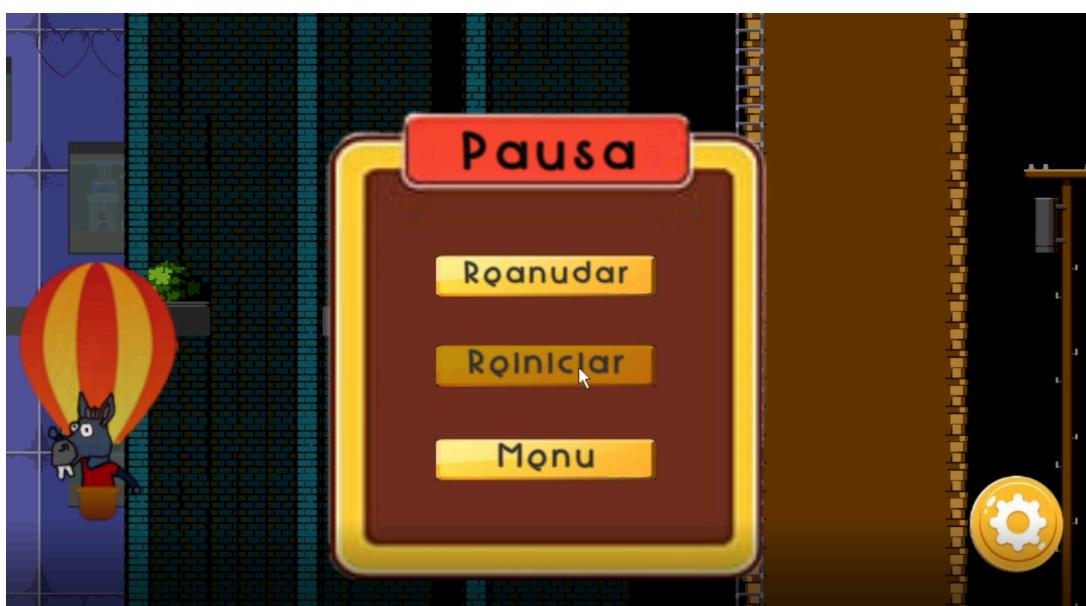
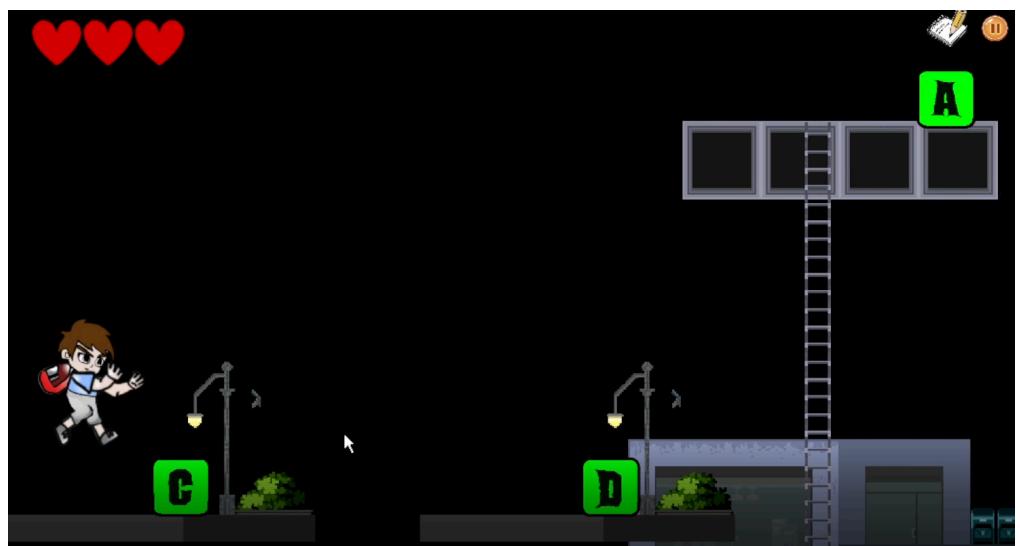
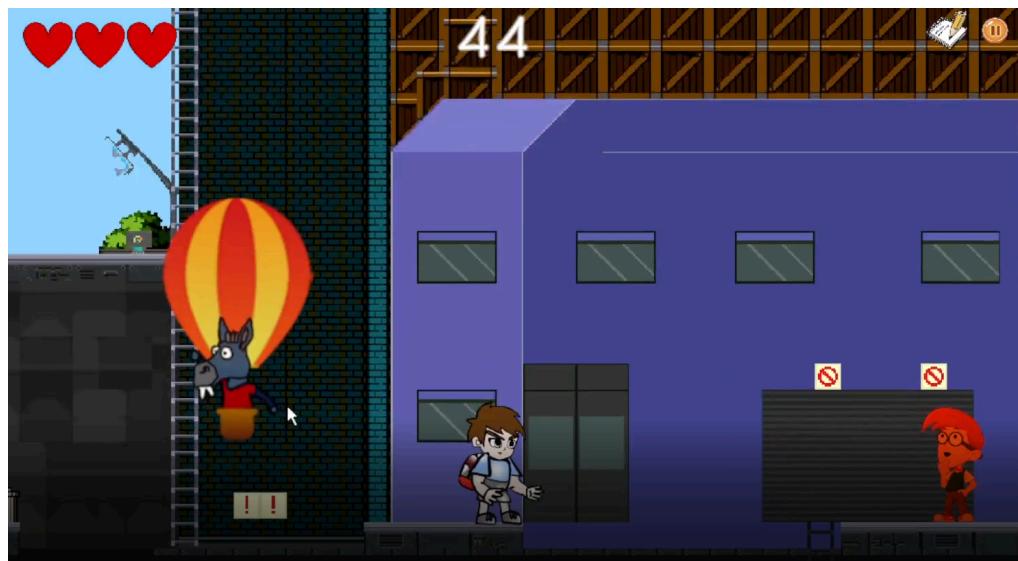
Descripción General del Diseño Artístico

El diseño artístico de este juego tiene un estilo cartoon tanto en el protagonista como en el entorno en el que se encuentra. Tiene colores llamativos para la persona que prueba el videojuego y facilita la visibilidad del personaje para poder asumir la respuesta correcta ya que los cuadros con las opciones resaltan en el entorno. Unos ejemplos del entorno serían:



A continuación, se muestran imágenes del juego dentro de un nivel:



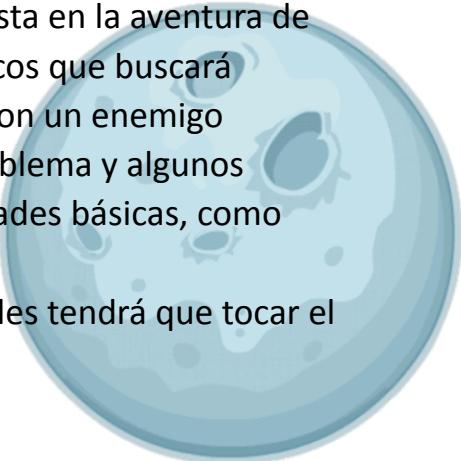


PERSONAJES

MathBoy

Nuestro protagonista llamada MathBoy, es un niño travieso pero que busca mejorar en sus matemáticas, por ello se enlista en la aventura de donde encontrara diferentes problemas matemáticos que buscará resolver, en esta aventura también se encontrará con un enemigo principal, el cual buscará evitar que resuelva el problema y algunos obstáculos que tendrá que superar con sus habilidades básicas, como saltar y escalar.

Existirán cajas con las respuestas correctas, las cuales tendrá que tocar el niño.



Animaciones

las animaciones que tendrá el personaje son:

- Saltar

Animación cuando el personaje quiere superar algún obstáculo



- Escalar:

Animación que ocurre cuando subes alguna escalera





- **Golpe:**

Animación cuando el persona entra en contacto con la bola de papel o alguna caída sobre púas



- **Parar/idle:**

Animación cuando el persona no hace ninguna acción





- Caminar:

Animación cuando el personaje se mueve por el mapa



- Morir:

Animación cuando el personaje muere



- Caída:

Animación cuando caemos desde una escalera o superficie



Cerebrito (NPC) :

Es un personaje masculino encargado de ayudar a nuestro personaje principal MathBoy con la intención de dar pistas u orientación para que logres resolver el problema matemático de tal forma que superes el juego. Este NPC te mostrará contenido de texto cuando nuestro personaje principal se acerque para interactuar.



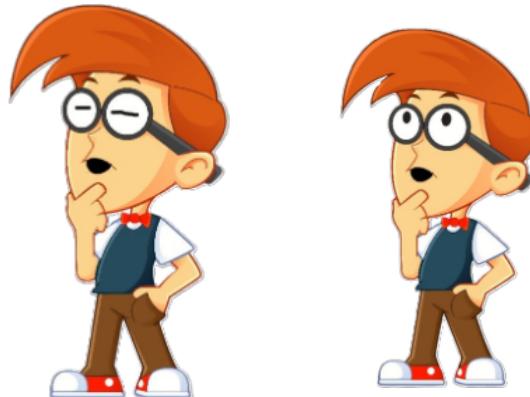
Animaciones:

Cuando este personaje entra en la escena del videojuego espera a detectar a MathBoy para mostrarle contenido en texto , puede mostrar animaciones de pensamiento y hablar para dar pistas a nuestro jugador principal

- Pensar:



- Hablar:





Código:

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

// Script de Unity (2 referencias de recurso) | 0 referencias
public class EnemigoPersecucion : MonoBehaviour
{
    [SerializeField] private float velocidadMovimiento;
    [SerializeField] private float distanciaDeteccion;
    private Transform jugador;
    private SpriteRenderer spriteRenderer;
    private bool jugadorEstaMirando;

    // Mensaje de Unity | 0 referencias
    private void Start()
    {
        GameObject jugadorObjeto = GameObject.FindGameObjectWithTag("Player");
        if (jugadorObjeto != null)
        {
            jugador = jugadorObjeto.transform;
        }

        spriteRenderer = GetComponent<SpriteRenderer>();
    }

    // Mensaje de Unity | 0 referencias
    private void Update()
    {
        if (jugador == null)
        {
            return;
        }

        Vector2 direccionHaciaJugador = jugador.position - transform.position;
        float distanciaHaciaJugador = direccionHaciaJugador.magnitude;

        jugadorEstaMirando = MirandoAlJugador(direccionHaciaJugador);

        if (!jugadorEstaMirando && distanciaHaciaJugador <= distanciaDeteccion)
        {
            transform.position = Vector2.MoveTowards(transform.position, jugador.position, velocidadMovimiento * Time.deltaTime);
            Girar();
        }
    }

    // Referencia
    private bool MirandoAlJugador(Vector2 direccionHaciaJugador)
    {
        Vector3 direccionJugador = jugador.localScale.x > 0 ? Vector2.right : Vector2.left;
        return Vector2.Dot(direccionHaciaJugador.normalized, direccionJugador) < 0;
    }

    // Referencia
    private void Girar()
    {
        if (transform.position.x < jugador.position.x)
        {
            spriteRenderer.flipX = true;
        }
        else
        {
            spriteRenderer.flipX = false;
        }
    }

    // Mensaje de Unity | 0 referencias
    private void OnCollisionEnter2D(Collision2D other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            other.gameObject.GetComponent<Interaccion>().TomarDaño(1, other.GetContact(0).normal);
        }
    }
}
```



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

// Script de Unity (5 referencias de recurso) | 11 referencias
public class interaccion : MonoBehaviour
{
    private int vida;
    public GameObject[] corazon;
    private JugadorController jugadorController;
    [SerializeField] private float tiempoPerdidaControl;
    private Animator animator;
    public event EventHandler Muertejugador;
    public event EventHandler NivelSuperado;
    private Rigidbody2D rb2D;
    public static bool muerteTiempo=false;
    public static bool RespuestaCorrecta=false;
    [SerializeField] private AudioClip sonidoDaño;
    [SerializeField] private AudioClip muerteSonido;
    [SerializeField] private AudioClip sonidoSuperado;

    // Mensaje de Unity | 0 referencias
    private void Start()
    {
        vida = corazon.Length;
        rb2D = GetComponent<Rigidbody2D>();
        jugadorController = GetComponent<JugadorController>();
        animator = GetComponent<Animator>();
    }

    // Mensaje de Unity | 0 referencias
    private void Update()
    {
        if (muerteTiempo)
        {
            ControladorSonido.Instance.EjecutarSonido(muerteSonido);
            muerteTiempo = false;
            rb2D.constraints = RigidbodyConstraints2D.FreezeAll;
            MuerteJugadorEvento();
        }
        if (RespuestaCorrecta)
        {
            ControladorSonido.Instance.EjecutarSonido(sonidoSuperado);
            RespuestaCorrecta = false;
            rb2D.constraints = RigidbodyConstraints2D.FreezeAll;
            NivelSuperadoEvento();
        }
    }
}
```

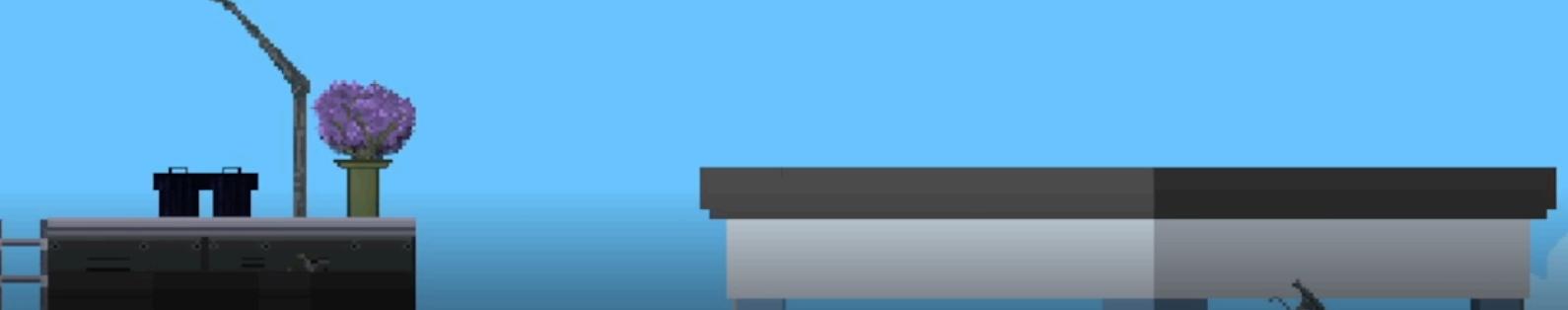
```
    Destroy(gameObject);
}

1 referencia
public void MuerteJugadorEvento()
{
    Muertejugador?.Invoke(this, EventArgs.Empty);
}

1 referencia
public void NivelSuperadoEvento()
{
    NivelSuperado?.Invoke(this, EventArgs.Empty);
}

2 referencias
private IEnumerator PerderControl()
{
    jugadorController.sePuedeMover = false;
    yield return new WaitForSeconds(tiempoPerdidaControl);
    jugadorController.sePuedeMover = true;
}

}
```



```
        }

    }

5 referencias
public void TomarDaño(int daño,Vector2 posicion)
{
    vida -= daño;

    if (daño == 3)
    {
        ControladorSonido.Instance.EjecutarSonido(muerteSonido);
        Destroy(corazon[0].gameObject);
        Destroy(corazon[1].gameObject);
        Destroy(corazon[2].gameObject);
        rb2D.constraints = RigidbodyConstraints2D.FreezeAll;
        animator.SetTrigger("Muerte");
        Physics2D.IgnoreLayerCollision(6, 10, true);
    }

    if (vida < 1)
    {
        ControladorSonido.Instance.EjecutarSonido(muerteSonido);
        Destroy(corazon[0].gameObject);
        rb2D.constraints = RigidbodyConstraints2D.FreezeAll;
        animator.SetTrigger("Muerte");
        Physics2D.IgnoreLayerCollision(6, 10, true);
    }
    else if (vida < 2)
    {
        ControladorSonido.Instance.EjecutarSonido(sonidoDaño);
        Destroy(corazon[1].gameObject);
        StartCoroutine(PerderControl());
        jugadorController.Rebote(posicion);

    }
    else if (vida < 3)
    {
        ControladorSonido.Instance.EjecutarSonido(sonidoDaño);
        Destroy(corazon[2].gameObject);
        StartCoroutine(PerderControl());
        jugadorController.Rebote(posicion);
    }
}
0 referencias
public void Destruir()

< using System.Collections;
< using System.Collections.Generic;
< using UnityEngine;
< using System;

@ Script de Unity (5 referencias de recurso) | 2 referencias
public class JugadorController : MonoBehaviour
{
    private Rigidbody2D rb2D;
    public bool sePuedeMover = true;
    public bool golpe=false;
    [SerializeField] private Vector2 velocidadRebote;

    private Vector2 input;

    [Header("Movimiento")]// se declara las variables a utilizar en movimiento horizontal
    private float MovH=0f;
    [SerializeField] private float VelocidadMov;//velocidad de recorrido
    [Range(0, 0.3f)][SerializeField] private float SuavizadoMov;//se le da un suavizado de movimiento en un rango de 0~0.3
    private Vector3 velocidad = Vector3.zero;// vector 3D de velocidad en 0
    public AudioSource sonidoMovimiento;
    private bool mirandoDerecha = true;

    [Header("Salto")]// se declara las variables a utilizar en el salto
    [SerializeField] private float fuerzaDeSalto;
    [SerializeField] private LayerMask queEsSuelo;// se declara un tipo etiqueta para detectar que es el suelo.
    [SerializeField] private Transform controladorSuelo;//se declara de tipo transform lo que permite usar sus propiedades como puntos de posicion.
    [SerializeField] private Vector3 dimensionesCaja;// se declara un objeto imaginario de la caja que se encuentra debajo del personaje.
    [SerializeField] private bool enSuelo;
    [SerializeField] private AudioClip saltoSonido;
    private bool salto = false;

    [Header("Salto Regulable")]
    [Range(0, 1)] [SerializeField] private float multiplicadorCancelarSalto;
    private float escalaGravedad;
    private bool botonSaltoArriba = true;

    [Header("Escalar")]
    [SerializeField] private float velocidadEscalar;
    private BoxCollider2D boxCollider2D;
    private float gravedadInicial;
    private bool escalando;
    public AudioSource escalarSonido;
```



```
[Header("Animacion")]
private Animator animator; // se declara las animaciones a utilizar
```

● Mensaje de Unity | 0 referencias

```
void Start()
```

```
{  
    rb2D = GetComponent<Rigidbody2D>();  
    animator = GetComponent<Animator>();  
    escalaGravedad = rb2D.gravityScale;  
    boxCollider2D = GetComponent<BoxCollider2D>();  
    gravedadInicial = rb2D.gravityScale;  
}
```

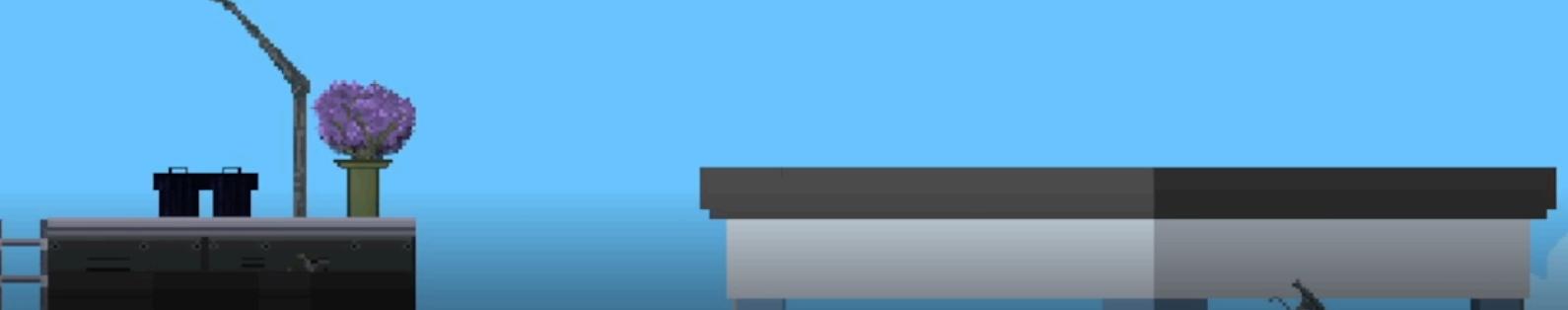
// Update is called once per frame

● Mensaje de Unity | 0 referencias

```
void Update()
```

```
{  
    input.x = Input.GetAxisRaw("Horizontal");  
    input.y = Input.GetAxisRaw("Vertical");  
  
    MovH = input.x * VelocidadMov;  
  
    animator.SetFloat("Horizontal", Mathf.Abs(MovH));  
    animator.SetFloat("VelocidadY", rb2D.velocity.y);  
  
    if (Input.GetButtonDown("Horizontal"))  
    {  
        sonidoMovimiento.Play();  
    }  
    if (Input.GetButtonUp("Horizontal"))  
    {  
        sonidoMovimiento.Pause();  
    }  
    if (Input.GetButtonDown("Vertical"))  
    {  
        sonidoMovimiento.Pause();  
        escalarSonido.Play();  
    }  
    if (Input.GetButtonUp("Vertical"))  
    {  
        escalarSonido.Pause();  
    }  
  
    if (rb2D.velocity.y < 0 && !enSuelo)
```

```
if (rb2D.velocity.y < 0 && !enSuelo)  
{  
    rb2D.gravityScale = escalaGravedad * multiplicadorGravedad;  
}  
else  
{  
    rb2D.gravityScale = escalaGravedad;  
}  
  
if (Mathf.Abs(rb2D.velocity.y) > Mathf.Epsilon) //cambia en valores pequeños que no es cero  
{  
    animator.SetFloat("VelocidadY", Mathf.Sign(rb2D.velocity.y));  
}  
else  
{  
    animator.SetFloat("VelocidadY", 0);  
}  
  
if (Input.GetButton("Salto"))  
{  
    sonidoMovimiento.Pause();  
    escalarSonido.Pause();  
    if (input.y >= 0)  
    {  
        salto = true;  
    }  
    else  
    {  
        descativarPlataformas();  
    }  
}  
if (Input.GetButtonUp("Salto"))  
{  
    BotonSaltoArriba();  
}  
enSuelo = Physics2D.OverlapBox(controladorSuelo.position, dimensionesCaja, 0f, queEsSuelo); //utiliza parametros de un vector2(en este caso  
//como posicion,el tamaño del vector2 como la dimensión,queEsSuelo,un angulo que seria en este caso 0  
//una etiqueta queEsSuelo que vendria ser la losa que el personaje esta encima  
animator.SetBool("EnSuelo", enSuelo);  
}
```



```
2 referencias
public void Rebote(Vector2 puntoGolpe)
{
    golpe = true;
    animator.SetBool("Golpe", golpe);
    rb2D.velocity = new Vector2(-velocidadRebote.x * puntoGolpe.x, velocidadRebote.y);
}

1 referencia
private void desactivarPlataformas()
{
    Collider2D[] objetos = Physics2D.OverlapBoxAll(controladorSuelo.position, dimensionesCaja, 0f, queEsSuelo); //devuelve un arreglo donde detiene
                                                                 //mismo componente a la etiqueta
    foreach (Collider2D item in objetos) //se recorre todos los objetos que toca
    {
        PlatformEffector2D platformEffector2D = item.GetComponent<PlatformEffector2D>(); //buscamos el componente dado en la plataforma
        if (platformEffector2D != null) //para saber si tiene este componente se pregunta si es diferente de nulo
        {
            Physics2D.IgnoreCollision(GetComponent<Collider2D>(), item.GetComponent<Collider2D>(), true); //se usa para ignorar colisiones, primero
                                                                 //el segundo del objeto que toca
                                                                 //true para desactivar la colisión de
        }
    }
}

private void OnDrawGizmosSelected() //funciones de visualización de la caja o gizmos
{
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireCube(controladorSuelo.position, dimensionesCaja);
}
```

He escogido fragmentos de código que pertenecen al comportamiento de un enemigo que persigue al jugador, ya que me ha parecido un desafío interesante. El enemigo solo se mueve hacia el jugador cuando está cerca y no lo está mirando, lo cual requiere coordinar varias variables como velocidad, distancia de detección, y orientación. Además, el script incluye la detección de colisiones para que el enemigo cause daño al jugador cuando interactúan, lo que añade complejidad al ajustar correctamente los movimientos, las colisiones y el sistema de daños.

También mostramos fragmentos de código, que pertenece al script de interacción del personaje, porque gestiona varios aspectos esenciales en el juego, desde el control de la vida hasta la activación de eventos como la muerte o superación de nivel. Esto hace que sea complejo de integrar y ajustar bien en el juego, ya que maneja efectos visuales, sonidos para distintas acciones, y colisiones que afectan el comportamiento del personaje, como perder el control temporalmente. Además, incluye la invocación de eventos que indican la muerte o el avance en el nivel.



REFERENCIAS

- Jašek, R. (2014). Multiplatform game development using the Unity engine (Master's thesis). Vysoké učení technické v Brně.
<http://www.nusl.cz/ntk/nusl-236075>
- Yang, Y. (2020). Wu: A cultural export game with dynamic difficulty (Master's thesis). Digital WPI. <https://digitalcommons.wpi.edu/etd-theses/1361>
- Mortazavi, R. S. S. (2015). Before eternity: An adventure game inspired by Sufi mysticism (Master's thesis). Digital WPI.
<https://digitalcommons.wpi.edu/etd-theses/832>
- Polancec, D., & Mekterovic, I. (2017). Developing MOBA games using the Unity game engine. In 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1-6). IEEE. <http://dx.doi.org/10.23919/mipro.2017.7973661>
- Nasution, S., Nasution, A. H., & Hakim, A. L. (2019). Tile-based game plugin for Unity engine. In The Second International Conference on Science, Engineering and Technology (pp. 55-63). SCITEPRESS - Science and Technology Publications.
<http://dx.doi.org/10.5220/0009103700550063>
- Jitendra, M., Srinivas, A. S., Surendra, T., Rao, R. V., & Chowdary, P. R. (2021). A study on game development using Unity engine. In ESSENCE OF MATHEMATICS IN ENGINEERING APPLICATIONS: EMEA-2020 (pp. 1-6). AIP Publishing.
<http://dx.doi.org/10.1063/5.0066303>