

PINGU

PROYECTOS II

Manuel Martínez Ramón II
INSO 2A

ÍNDICE

DESCRIPCIÓN DEL PROYECTO.....	3
FUNDAMENTACIÓN.....	3
METAS.....	5
¿DÓNDE SE VA A DESARROLLAR EL JUEGO?.....	5
¿CÓMO SE VA A DESARROLLAR EL JEUGO?.....	6
PROYECTO.....	7
Sprites.....	7
Físicas.....	8
Animaciones.....	11
Enemigos.....	11
Items.....	13
Final.....	14
Música/Audio.....	15
Interfaz de usuario/HUD.....	16
Cámara.....	17
Menú/Botones.....	19
CONCLUSIÓN.....	19
FOTOS DEL JUEGO.....	20
PROBLEMAS/UNIÓN.....	24

DESCRIPCIÓN DEL PROYECTO

En el marco de nuestro curso de Proyectos II: Tendencias de la Ingeniería del Software, se presenta la fascinante oportunidad de participar en un proyecto colaborativo que pondrá a prueba y consolidará los conocimientos adquiridos durante nuestras sesiones académicas. Este proyecto consiste en la creación de un juego básico en 2D, un desafío que nos permitirá aplicar de manera práctica las herramientas y técnicas que hemos explorado en clase. Para llevar a cabo este proyecto, se organizaron grupos de estudiantes que trabajarán de manera conjunta para diseñar, desarrollar y lanzar un juego único y atractivo.

El juego en cuestión deberá aprovechar al máximo las funcionalidades y capacidades de Unity, explorando aspectos esenciales del diseño de juegos en 2D. Desde la creación de personajes y escenarios hasta la implementación de mecánicas de juego envolventes, los participantes tendrán la oportunidad de demostrar su destreza técnica y creatividad. Este proyecto no solo representa una evaluación integral de los conocimientos adquiridos, sino también una oportunidad emocionante para construir habilidades prácticas y establecer las bases para futuros proyectos innovadores.

FUNDAMENTACIÓN

La elección de realizar un proyecto de desarrollo de juegos en grupo, específicamente utilizando Unity y herramientas previamente discutidas en el curso, se fundamenta en la idea de proporcionar a los estudiantes una experiencia de aprendizaje inmersiva y práctica. Los juegos ofrecen un terreno fértil para aplicar conceptos teóricos de programación, diseño y desarrollo de software de una manera altamente motivadora. Unity, como entorno de desarrollo de juegos líder en la industria, brinda una plataforma robusta y accesible para la materialización de ideas creativas en experiencias de juego tangibles.

La decisión de trabajar en grupos refleja la realidad de la colaboración en el desarrollo de juegos, donde la combinación de habilidades individuales contribuye al éxito del proyecto en su conjunto. Esta metodología fomenta la comunicación efectiva, la resolución conjunta de problemas y la asignación de roles, habilidades fundamentales en el ámbito laboral. Además, al participar en la creación de un juego 2D, tendremos la oportunidad de comprender y aplicar los principios esenciales del diseño visual, optimización de rendimiento y narrativa, enriqueciendo así su comprensión integral de la creación de juegos digitales.

En última instancia, este proyecto busca no solo consolidar los conocimientos teóricos adquiridos, sino también cultivar habilidades prácticas, la capacidad de trabajo en equipo y el pensamiento crítico, habilidades cruciales para el éxito en la industria del desarrollo de videojuegos y en el ámbito profesional en general.

OBJETIVOS

He tomado la iniciativa de formar un grupo de tres personas con el objetivo de crear un juego que refleje nuestra comprensión colectiva de las técnicas y herramientas aprendidas en clase. La decisión estratégica de trabajar con un número impar de integrantes busca no solo facilitar la colaboración, sino también brindar flexibilidad y resistencia frente a posibles desafíos individuales. Esta elección permite un enfoque más dinámico y adaptativo, asegurando que el grupo pueda superar obstáculos de manera eficiente.

La idea central de nuestro proyecto es la creación de un juego que conste de tres mini-juegos diferentes, una decisión motivada por la diversificación de la experiencia de juego y la mitigación de riesgos en el caso de contratiempos individuales. Esta estructura nos proporciona la oportunidad de especializarnos en aspectos particulares del desarrollo, maximizando así la eficacia de nuestro trabajo conjunto.

Cada mini-juego se convertirá en una pieza única y esencial de nuestro proyecto global, ofreciendo a cada miembro del equipo la oportunidad de contribuir significativamente a la creación conjunta. Mi elección personal recae en la creación de un juego de plataformas ambientado en el Polo Norte, con un adorable pingüino como protagonista. Este enfoque no solo aprovecha las técnicas fundamentales enseñadas en clase, sino que también permite la aplicación creativa de conceptos clave, como la física del juego, la detección de colisiones y el diseño de niveles. A través de esta elección, espero no solo demostrar mis habilidades técnicas, sino también perfeccionar mis habilidades de colaboración y gestión de proyectos, elementos cruciales en el competitivo mundo del desarrollo de videojuegos.

En resumen, nuestro proyecto no es simplemente un ejercicio técnico; es una oportunidad emocionante para fusionar conocimientos teóricos con la práctica real, desarrollar habilidades de trabajo en equipo y enfrentarnos a los desafíos del desarrollo de videojuegos de manera innovadora y efectiva.

METAS

En este proyecto, mi enfoque individual se centra en la creación de un juego de plataformas inspirado en el Ártico habitado por un entrañable pingüino. Mi meta principal es aplicar de manera creativa las técnicas y conceptos fundamentales aprendidos en clase, destacando especialmente en la implementación de la física del juego y la detección de colisiones. Al asumir la responsabilidad de este mini-juego, aspiro a perfeccionar mis habilidades de programación en Unity, garantizando una experiencia de juego fluida y envolvente para los jugadores.

Además, mi objetivo individual se amplía a la colaboración efectiva con mis compañeros en la integración de los tres mini-juegos. Busco no solo destacar en mi área de especialización, sino también asegurar una cohesión armoniosa entre los distintos componentes del proyecto. La comunicación abierta y la adaptabilidad serán clave para superar cualquier desafío y lograr un producto final que destaque tanto en términos técnicos como estéticos. A través de este proyecto, espero no solo consolidar mis habilidades técnicas, sino también fortalecer mi capacidad para trabajar de manera eficiente en un entorno de equipo colaborativo, sentando las bases para futuros éxitos en el ámbito del desarrollo de videojuegos.

¿DÓNDE SE VA A DESARROLLAR EL JUEGO?

Se llevará a cabo en el entorno de Unity, específicamente en la versión 2022.3.9f1, la más reciente hasta la fecha de inicio de nuestro proyecto, noviembre 2023. Este entorno de desarrollo nos ofrece la tecnología y las herramientas para la implementación de mecánicas de juego en 2D, lo cual es esencial para la realización de mi visión creativa.

La elección de Unity 2022.3.9f1 para el desarrollo de mi mini-juego se basa en la necesidad de aprovechar las últimas actualizaciones y características que la plataforma ofrece. Esta versión específica no solo proporciona estabilidad y rendimiento, sino que también garantiza que mi juego se beneficie de las mejoras continuas en la interfaz de usuario, las capacidades de diseño y las optimizaciones de rendimiento más recientes. Asimismo, facilitará la implementación de elementos clave como la física del juego, la detección de colisiones y el diseño meticuloso de niveles, elementos cruciales para el éxito de un juego de plataformas.

¿CÓMO SE VA A DESARROLLAR EL JUEGO?

El enfoque integral para el desarrollo de mi mini-juego se basará en la construcción progresiva a partir de conceptos fundamentales, haciendo uso de las técnicas esenciales que hemos explorado en clase. Partiendo desde cero, mi objetivo es establecer una sólida base conceptual que permita la expansión y mejora continua a lo largo del desarrollo del juego. Esta estrategia no solo busca garantizar la comprensión profunda de cada componente del juego, sino también facilitar la implementación de futuras funcionalidades.

El proceso de creación comenzará desde lo más básico, abordando la programación de personajes, la manipulación de objetos y la gestión de eventos. A medida que avanzamos, incorporaré progresivamente las técnicas enseñadas en clase, desde el diseño de niveles y la aplicación de física del juego hasta la optimización de rendimiento. La repetición constante de estos procesos será clave, permitiéndome ajustar y mejorar cada aspecto del juego a medida que adquiero un mayor dominio de las herramientas proporcionadas por Unity.

Además de las lecciones en clase, mi enfoque de aprendizaje será complementado mediante la consulta activa de diferentes foros especializados y video tutoriales. Explorar la comunidad online para buscar soluciones a desafíos específicos, aprender nuevas técnicas y comprender las mejores prácticas recomendadas por otros desarrolladores experimentados.

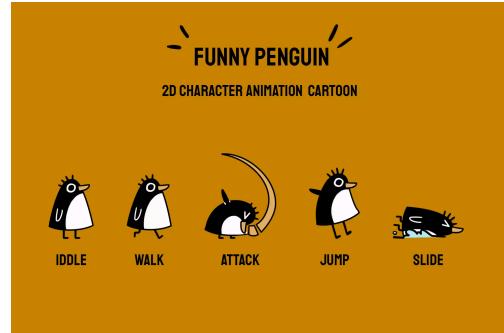
La modularidad y escalabilidad serán consideraciones centrales durante todo el proceso de desarrollo. Al estructurar el código y los recursos de manera eficiente desde el principio, garantizaré la capacidad de ampliar fácilmente el juego en el futuro. Esta mentalidad de diseño permitirá la incorporación de nuevas mecánicas, niveles y características sin comprometer la integridad del código existente, facilitando la evolución continua del proyecto.

En resumen, mi enfoque para el desarrollo del juego se basa en una construcción progresiva desde lo básico, aprovechando las técnicas aprendidas en clase, la exploración activa de recursos en línea y la utilización de la documentación oficial de Unity. Esta metodología no solo asegura la comprensión profunda de cada aspecto del juego, sino que también establece las bases para un crecimiento continuo y la expansión sencilla de las funcionalidades a medida que avanzamos en el emocionante viaje del desarrollo de videojuegos.

PROYECTO

Sprites

La selección cuidadosa de sprites en el proceso de desarrollo de un juego desempeña un papel crucial en la creación de una experiencia visualmente atractiva y coherente. En mi caso, he optado por elegir sprites de la página oficial de Unity Asset Store (<https://assetstore.unity.com/>), reconociendo la importancia de acceder a una fuente confiable y diversa que garantice la calidad y la legalidad de los recursos visuales.



AssetStore | 2D Character Sprite Animation - Penguin



AssetStore | Pixel Adventure 2

El hecho de que los sprites seleccionados sean gratuitos y de uso libre añade un componente valioso a mi proyecto, permitiéndome optimizar los costos y respetar las políticas de derechos de autor. La amplia variedad de sprites disponibles en la Asset Store ofrece una gama impresionante de opciones para satisfacer mis necesidades específicas, en este caso, la creación de un juego ambientado en el Ártico.

La temática del Ártico impone una estética única y, al seleccionar sprites que aunque no reflejen esta ambientación llaman la atención y estoy asegurando mas estímulos visuales que sumergirán al jugador de manera más profunda en la experiencia del juego. La Asset Store de Unity ofrece una categorización eficiente que facilita la búsqueda de sprites. Este enfoque no solo simplifica el proceso de selección, sino que también garantiza la cohesión estilística esencial para la inmersión del jugador.



AssetStore | Animated 2D Coins

Uno de los elementos más destacados de mi selección de sprites es el conjunto de animales descargados, que añadirá dinamismo y vitalidad al juego. Al dotar a estos sprites de capacidad de animación, proporcionaré a los personajes del juego un aspecto más orgánico y permitiré que el jugador interactúe de manera significativa con el entorno del Ártico. Este enfoque no solo añade un toque lúdico, sino que también enriquece la experiencia global del jugador al introducir elementos vivos y animados en el paisaje digital.



AssetStore | 2D Ice World

En resumen, la importancia de la selección de sprites radica en la creación de una experiencia visual coherente y cautivadora. La Asset Store de Unity, con su variedad de opciones y la garantía de calidad, se convierte en una elección estratégica para asegurar un desarrollo eficiente y estéticamente agradable. La temática del Ártico, respaldada por sprites específicamente seleccionados, promete ofrecer a los jugadores una experiencia única, respaldada por la diversidad de recursos visuales cuidadosamente elegidos y la posibilidad de interactuar con animales animados que enriquecerán la narrativa del juego.

Físicas

El script "Mover.cs" destinado a gestionar las físicas del jugador, en este caso, el pingüino, constituye un componente esencial y dinámico en el desarrollo del minijuego. Desde sus primeras versiones hasta su versión final, este script ha experimentado una evolución significativa, ajustándose y refinándose para lograr un comportamiento de movimiento que no solo sea realista, sino también cautivador para el jugador.

El script inicial abordó la recopilación de datos fundamentales del jugador, como la posición, la velocidad y la dirección. Estos datos proporcionaron la base para la implementación de un sistema de saltos acumulativos, permitiendo al pingüino realizar saltos más altos y potentes al mantener presionada la tecla de espacio. Este mecanismo no solo enriquece la jugabilidad al otorgar al jugador un mayor control sobre los saltos, sino que también agrega un elemento estratégico al juego, ya que el timing y la duración de la presión de la tecla afectan directamente la altura del salto.

Uno de los aspectos más destacados de las modificaciones realizadas a lo largo del desarrollo del script se centra en la adaptación progresiva del movimiento durante los cambios de dirección.

Implementé cambios en la aceleración y dirección del pingüino para que estos se produjeran de manera gradual, proporcionando una transición más suave y natural. Este ajuste no solo contribuye a la coherencia en el movimiento del pingüino, sino que también mejora la sensación de control para el jugador, evitando cambios bruscos que puedan resultar desorientadores.

La adaptabilidad del script a lo largo del desarrollo se ha beneficiado tanto de la experimentación práctica como de la consulta de recursos en línea y la revisión de la documentación oficial de Unity. Foros especializados, tutoriales y la comunidad de desarrollo han sido recursos valiosos para abordar desafíos específicos y explorar nuevas técnicas que enriquecieran la funcionalidad del script.

En definitiva, el script "Mover.cs" representa un elemento fundamental en el desarrollo del minijuego, evolucionando desde su concepción inicial para ofrecer un sistema de movimiento dinámico, preciso y atractivo para el jugador. La atención meticulosa a los detalles y la adaptabilidad a lo largo del

```
isSkidding = false;
currentVelocity = rb2D.velocity.x;
if(currentDirection > 0)
{
    if(currentVelocity < 0)
    {
        currentVelocity += (acceleration + friction) * Time.deltaTime;
        isSkidding = true;
    }
    else if(currentVelocity < maxVelocity)
    {
        currentVelocity += acceleration * Time.deltaTime;
        transform.localScale = new Vector2(1, 1);
    }
}
else if(currentDirection < 0)
{
    if(currentVelocity > 0)
    {
        currentVelocity -= (acceleration + friction) * Time.deltaTime;
        isSkidding = true;
    }
    else if(currentVelocity > -maxVelocity)
    {
        currentVelocity -= acceleration * Time.deltaTime;
        transform.localScale = new Vector2(-1, 1);
    }
}
```

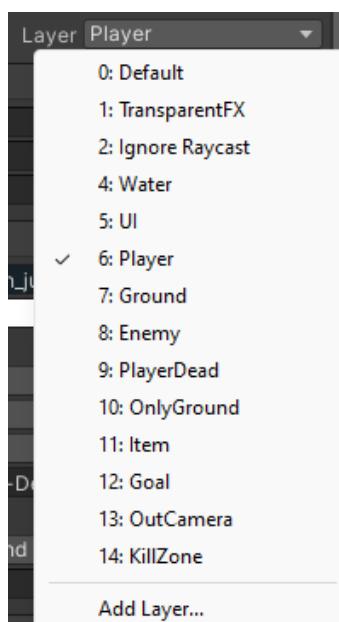
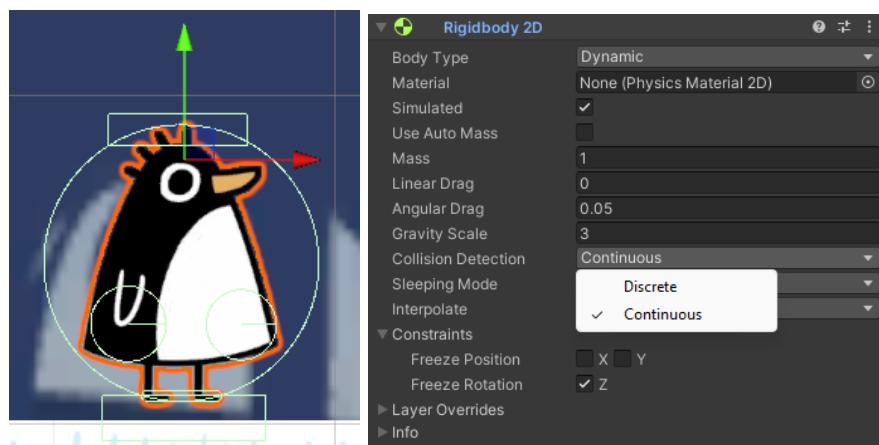
Mover.cs

proceso de desarrollo han sido clave para garantizar que este script cumpla con las expectativas de jugabilidad y realismo establecidas desde el principio. Este enfoque iterativo en la mejora continua es un testimonio del compromiso con la excelencia en el desarrollo del minijuego.

```
if(isJumping)
{
    //AudioManager.Instance.PlayJump();
    if(rb2D.velocity.y > 0f)
    {
        headBox.SetActive(true);
        if(Input.GetKeyDown(KeyCode.Space))
        {
            jumpTimer += Time.deltaTime;
        }
        if(Input.GetKeyUp(KeyCode.Space))
        {
            if(jumpTimer < maxJumpingTime)
            {
                rb2D.gravityScale = defaultGravity * 3f;
            }
        }
    }
    else
    {
        rb2D.gravityScale = defaultGravity;
        if(grounded)
        [
            isJumping = false;
            jumpTimer = 0;
            animaciones.Jumping(false);
        ]
    }
}
```

Mover.cs

El script "Colisiones.cs" se centra en gestionar colisiones durante el salto o caída, utilizando colisionadores específicos para prevenir detenciones innecesarias y evitar colisiones falsas al rozar objetos.



La integración de ambos scripts es esencial para asegurar un comportamiento coherente del personaje. Se han realizado pruebas exhaustivas y ajustes repitiendo estos procesos para garantizar un movimiento suave y preciso, así como para optimizar las detenciones de colisiones. En conjunto, estos scripts son vitales para proporcionar una experiencia de juego inmersiva y libre de problemas relacionados con el movimiento y las interacciones con el entorno.

Además, con las colisiones tuve bastantes problemas, ya que Unity por defecto no calcula las detenciones de manera continua para el ahorro de recursos, lo que hacía que el personaje atravesara el suelo en si la velocidad era muy alta.

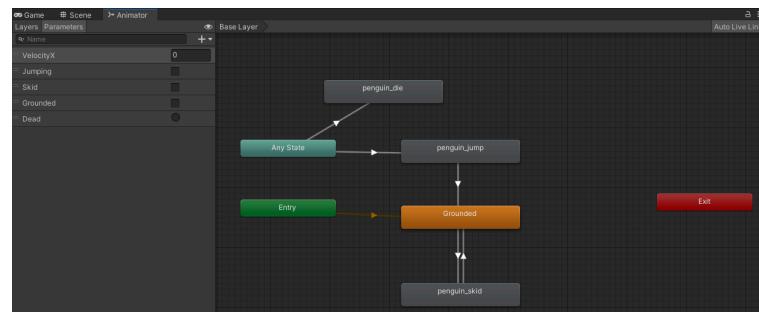
La solución fue poner que la detención del collider fue continua, esto aumentaba el uso de recursos del juego pero únicamente se lo tenía que implementar al jugador.

Animaciones

Opté por incorporar las animaciones preexistentes que venían incluidas con los sprites de los personajes que descargué. Estas animaciones proporcionan una base sólida y eficiente para el movimiento del personaje, permitiéndome centrarme en otros aspectos clave del desarrollo del juego. La elección de utilizar las animaciones predefinidas también agiliza el proceso de producción al proporcionar una solución lista para usar, evitando la necesidad de crear animaciones personalizadas desde cero.

Las animaciones descargadas no solo ofrecen un conjunto completo de movimientos, como caminar, correr y saltar, sino que también están optimizadas para trabajar armoniosamente con los sprites específicos de nuestros personajes.

Esta decisión no solo ahorra tiempo en el desarrollo, sino que también garantiza una coherencia visual entre las animaciones y los diseños de los personajes. Al incorporar estas animaciones preexistentes, hemos logrado una implementación eficiente y de alta calidad que contribuye positivamente a la experiencia global del juego.



Enemigos

En mi trabajo individual en el proyecto, me enfoqué en desarrollar varios scripts específicamente diseñados para el comportamiento de los enemigos. El primero de estos scripts es "AutoMovement", el cual desencadena un movimiento lateral del enemigo. Este script incorpora la lógica necesaria para gestionar las colisiones, cambiando la dirección del enemigo y ajustando el sprite en el eje X para mantener su orientación adecuada.

```
public void PauseMovement()
{
    if (!movementPaused)
    {
        currentDirection = rb2D.velocity.normalized;
        lastVelocity = rb2D.velocity;
        movementPaused = true;
        rb2D.velocity = new Vector2(0, 0);
    }
}
public void ContinueMovement()
{
    if (movementPaused)
    {
        speed = defaultSpeed * currentDirection.x;
        rb2D.velocity = new Vector2(speed, lastVelocity.y);
        movementPaused = false;
    }
}
public void ContinueMovement(Vector2 newVelocity)
{
    if (movementPaused)
    {
        rb2D.velocity = newVelocity;
        movementPaused = false;
    }
}
```

AutoMovement.cs

Cuando el enemigo colisiona con un objeto, el script garantiza una transición suave al invertir la dirección de movimiento y realizar las modificaciones necesarias en el sprite. Además, el script incluye una funcionalidad que permite al enemigo detenerse temporalmente, almacenando la dirección y velocidad actuales. Esta característica es crucial para preservar la coherencia en el movimiento del enemigo cuando se reanuda la acción, evitando cambios inesperados en la velocidad o dirección al reiniciar el movimiento automático.

Además del script "AutoMovement" para los enemigos, creé un script principal llamado "Enemy" que sirve como base para los scripts específicos de cada enemigo en el juego. Implementé tres enemigos distintos, cada uno con su propio comportamiento único.

El primero, "Chicken", representa una gallina. El script asociado controla el cambio de sprite cuando la gallina es golpeada, limita las colisiones al suelo y, después de un segundo, destruye el objeto para simular su derrota.

El segundo enemigo, "Snail", personifica un caracol. Este enemigo no puede ser derrotado, pero reacciona al salto del jugador cambiando su sprite. Además, dependiendo de la posición en el eje X en relación con el jugador, lanza su caparazón en esa dirección. Después de un tiempo, el caracol vuelve a su estado normal.

```
public override void Stomped(Transform player)
{
    if(!isHidden)
    {
        isHidden = true;
        animator.SetBool("Hidden", isHidden);
        autoMovement.PauseMovement();
    }
    else
    {
        if(Mathf.Abs(rb2D.velocity.x) > 0f)
        {
            autoMovement.PauseMovement();
        }
        else
        {
            if(player.position.x < transform.position.x)
            {
                autoMovement.speed = rollingSpeed;
            }
            else
            {
                autoMovement.speed = -rollingSpeed;
            }
            autoMovement.ContinueMovement(new Vector2(autoMovement.speed, 0f));
        }
    }

    gameObject.layer = LayerMask.NameToLayer("OnlyGround");
    Invoke("ResetLayer", 0.1f);
    stoppedTimer = 0;
}
```

Snail.cs

El tercer enemigo, gestionado por el script "ShowAndHide", representa un fantasma. Este script utiliza puntos visibles e invisibles, velocidad y tiempos de permanencia para controlar el movimiento del fantasma de manera precisa.

```
bool Locked()
{
    return Physics2D.OverlapBox(transform.position + Vector3.up, Vector2.one, 0);
}

private void OnDrawGizmos()
{
    Gizmos.DrawWireCube(transform.position + Vector3.up, Vector2.one);
}
```

ShowAndHide.cs

Además, implementé un colisionador mediante script que bloquea el movimiento del fantasma cuando detecta al personaje "Penguin" encima, agregando complejidad y desafío al encuentro con este enemigo.

En resumen, la creación de estos scripts específicos para cada enemigo contribuye a la diversidad y complejidad del juego, proporcionando a cada enemigo su propia identidad y desafíos únicos para el jugador.

Items

En el desarrollo de los elementos de recolección del juego, creé dos items base que servirían como prefabs modificables para futuros elementos. Estos elementos comparten un script principal llamado "Item," del cual se derivan scripts específicos para cada caso.

El primer item implementado fue la moneda, que presenta una animación y suma puntos al contador del jugador de manera incremental.

El segundo item, un bloque representando un animal volador, tiene dos funciones clave. Primero, proporciona monedas cada vez que se golpea desde abajo hasta alcanzar un límite establecido. Segundo, actúa como una plataforma sobre la cual el jugador puede caminar. Además, algunos bloques se rompen en diferentes trozos, en lugar de otorgar monedas, para añadir variedad al juego.

```
if(collision.CompareTag("HeadPenguin"))
{
    if(isBreakable)
    {
        Break();
    }
    else if(!isEmpty)
    {
        if(numCoins > 0)
        {
            if(!bouncing)
            {
                Instantiate(coinBlockPrefab, transform.position, Quaternion.identity);
                numCoins--;
                hud.AddCoins();
                //AudioManager.Instance.PlayCoin();
                Bounce();
                if(numCoins <= 0)
                {
                    isEmpty = true;
                    GetComponent<SpriteRenderer>().sprite = emptyBlock;
                }
            }
        }
    }
}
```

Radish.cs

```
IEnumerator Animation()
{
    while(true)
    {
        spriteRenderer.sprite = sprites[animationFrame];
        animationFrame++;
        if(animationFrame >= sprites.Length)
        {
            animationFrame = 0;
        }
        yield return new WaitForSeconds(frameTime);
    }
}
```

SpriteAnimation.cs

Para optimizar el rendimiento, desarrollé el script "SpritesAnimations," que utiliza arrays para seleccionar varias imágenes y cambiarlas en cada fotograma durante el upgrade. Esta solución redujo significativamente el consumo de recursos en comparación con las animaciones tradicionales, lo que me permitió reemplazar las

animaciones de los items y aplicar esta solución eficiente.

Específicamente para el animal volador, implementé el script "Radish," donde un valor público determina la cantidad de monedas que otorgará. Cuando se agota su suministro de monedas, cambia su sprite para indicar que ya no tiene más que ofrecer. Si se inicializa con cero monedas, en lugar de

cambiar el sprite, el animal explota, creando una animación de explosión mediante la invocación de sprites desde las esquinas del objeto con diferentes velocidades y sin colisiones para lograr un efecto visual impactante. En conjunto, estos elementos y scripts contribuyen a la dinámica y la riqueza visual del juego.

```
void Break()
{
    //AudioManager.Instance.PlayBreak();
    GameObject brickPiece;

    brickPiece = Instantiate(brickPiecePrefab, transform.position, Quaternion.Euler(new Vector3(0, 0, 0)));
    brickPiece.GetComponent<Rigidbody2D>().velocity = new Vector2(6f, 12f);

    brickPiece = Instantiate(brickPiecePrefab, transform.position, Quaternion.Euler(new Vector3(0, 0, 90)));
    brickPiece.GetComponent<Rigidbody2D>().velocity = new Vector2(-6f, 12f);

    brickPiece = Instantiate(brickPiecePrefab, transform.position, Quaternion.Euler(new Vector3(0, 0, -90)));
    brickPiece.GetComponent<Rigidbody2D>().velocity = new Vector2(6f, 8f);

    brickPiece = Instantiate(brickPiecePrefab, transform.position, Quaternion.Euler(new Vector3(0, 0, 180)));
    brickPiece.GetComponent<Rigidbody2D>().velocity = new Vector2(-6f, 8f);

    Destroy(gameObject);
}
```

Radish.cs

Final

Para la fase final del juego, incorporé un elemento estratégico utilizando el sprite de un árbol. En la parte superior del árbol, implementé un colisionador que, al entrar en contacto con el pingüino, activa automáticamente un movimiento descendente.

```
public class GoalPole : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Penguin penguin = collision.GetComponent<Penguin>();
        if(penguin != null)
        {
            penguin.Goal();
        }
    }
}
```

GoalPole.cs

Este movimiento es controlado por el script "GoalPole," el cual, al invocar la función "Goal" en el script principal del jugador "Penguin," desencadena la ejecución de un script llamado "Mover".

```
public void Goal()
{
    //AudioManager.Instance.PlayGoal();
    mover.DownFlagPole();
}
```

Penguin.cs

Este script tiene la capacidad de restringir todos los movimientos del jugador y aplicar una velocidad negativa en el eje Y, forzando al pingüino a descender hasta hacer contacto con el suelo.

En el suelo, coloqué otro colisionador, asociado a un script sencillo, que detecta la colisión con los pies del pingüino y activa una animación junto con el cambio de escena. Este diseño crea una

secuencia de eventos fluida y lógica, donde la interacción con el árbol conduce al jugador hacia el desenlace de la partida (iglú).

En resumen, la combinación de estos elementos y scripts finales proporciona un cierre coherente y satisfactorio para la experiencia de juego.

```
public void DownFlagPole()
{
    inputMoveEnabled = false;
    rb2D.velocity = new Vector2(0, -5f);
}
```

Mover.cs

Música/Audio

```
 AudioSource audioSource;
public static AudioManager Instance;

private void Awake()
{
    if(Instance != null)
    {
        Instance = this;
        audioSource = GetComponent<AudioSource>();
    }
}

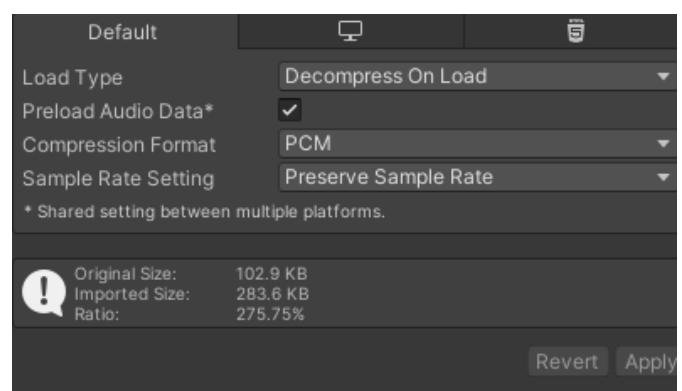
public void PlayJump()
{
    audioSource.PlayOneShot(clipJump);
}
```

Mover.cs

En el desarrollo del apartado de audio del juego, implementé un script dedicado llamado "AudioManager." Este script facilita la configuración de pistas de audio y su invocación desde cualquier otro script.

Sin embargo, enfrenté un problema persistente durante la ejecución: la reproducción del audio se detenía, impidiendo que el juego avanzara como se esperaba. A pesar de realizar diversas pruebas y explorar distintas soluciones, el problema persistió, resultando en una interrupción constante del audio durante la ejecución.

A pesar de este inconveniente, he dejado comentadas las líneas de código en los diferentes scripts que invocan los audios, en caso de que se resuelva el problema en futuras versiones del entorno de desarrollo o se encuentre una solución alternativa. Este enfoque permite mantener la funcionalidad del audio en suspenso, pero listo para su implementación cuando las condiciones lo permitan. Aunque el desafío persiste, la documentación detallada de las líneas de código y los intentos realizados proporcionan una base para abordar y resolver este problema en versiones futuras del juego o del entorno de desarrollo.



Interfaz de usuario/HUD

En la implementación de la interfaz de usuario (UI) y el HUD para el juego, diseñé un lienzo sencillo que exhibe información crucial, incluyendo el mundo actual, el contador de monedas, la puntuación y, en teoría, el tiempo restante. Además, para mejorar la estética del texto, integré un tipo de letra personalizado descargado como un archivo .ttf y luego importado a Unity para ser utilizado en los elementos de la interfaz.

```
public class LevelManager : MonoBehaviour
{
    public HUD hud;
    int coins;

    public static LevelManager Instance;
    private void Awake()
    {
        if(Instance != null)
        {
            Instance = this;
        }
    }
    // Start is called before the first frame update
    void Start()
    {
        coins = 0;
    }

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

    public void AddCoins()
    {
        coins++;
        hud.UpdateCoins(coins);
    }
}
```

LevelManager.cs

Aunque inicialmente se incluyó la función de puntuación por derrotar enemigos y la visualización del tiempo restante, se presentaron desafíos técnicos al compilar el proyecto con otros dos mini-juegos de tus compañeros. Esto llevó a la eliminación temporal de la función de puntuación y la incapacidad para mostrar adecuadamente el tiempo restante en pantalla.

A pesar de estos obstáculos, mantuve el código relacionado con la puntuación por derrotar enemigos y el tiempo restante en los scripts correspondientes, facilitando su reintegración futura. La inclusión de estas funciones proporciona una base sólida, y con el tiempo y recursos adecuados, es posible resolver los problemas técnicos y restaurar la funcionalidad completa del HUD.

```
// Update is called once per frame
void Update()
{
    if (countdownTime > 0)
    {
        countdownTime -= Time.deltaTime;
        //Debug.Log("Tiempo restante: " + Mathf.Floor(countdownTime));
    }
    //score.text = Score.Instance.puntos.ToString("D6");
}
public void AddCoins()
{
    coins++;
    numCoins.text = "x" + coins.ToString("D2");
}
public void UpdateCoins(int totalCoins)
{
    numCoins.text = "x" + totalCoins.ToString("D2");
}
```

HUD.cs

```
using TMPro;

public class HUD : MonoBehaviour
```

HUD.cs

Cámara

En la implementación de la cámara para el juego, opté por un enfoque que asegura una adaptabilidad efectiva a diferentes tamaños y proporciones de pantalla. Utilicé un cálculo basado en los píxeles que se muestran en la pantalla, permitiendo que la cámara se ajuste de manera óptima a las variaciones de tamaño sin perder información del juego. Además, introduce límites tanto en el lado izquierdo como en el derecho para garantizar que la cámara no revele más allá de estos puntos definidos.

La cámara se encuentra restringida a movimientos exclusivamente en el eje Y, y para evitar que el jugador retroceda indefinidamente, implementé una variable que establece el límite izquierdo. Este límite se actualiza dinámicamente, asegurando que, a medida que el jugador avanza, el último píxel visible en el lado izquierdo se convierte en el nuevo límite.

```
void Start()
{
    /*
    float camWidth = Camera.main.ScreenToWorldPoint(new Vector3(Screen.width, 0, 0)).x;
    Debug.Log("Ancho Camera: " + camWidth);
    float height = Camera.main.orthographicSize;
    Debug.Log("Camera.main.aspect: " + Camera.main.aspect);
    float width = height * Camera.main.aspect;
    Debug.Log("Otro Ancho: " + width);
    */
    camWidth = Camera.main.orthographicSize * Camera.main.aspect;
    minPosX = limitLeft.position.x + camWidth;
    //Debug.Log("Posicion min camara: " + minPosX);
    maxPosX = limitRight.position.x - camWidth;
    //Debug.Log("Posicion max camara: " + +minPosX);
    lastPos = minPosX;

    colLeft.position = new Vector2(transform.position.x - camWidth - 0.65f, colLeft.position.y);
    colRight.position = new Vector2(transform.position.x + camWidth + 0.65f, colRight.position.y);
}

// Update is called once per frame
void Update()
{
    float newX = target.position.x + followAhead;
    newX = Mathf.Clamp(newX, lastPos, maxPosX);
    transform.position = new Vector3(newX, transform.position.y, transform.position.z);
    lastPos = newX;
}
```

CameraFollow.cs

Como precaución adicional, incorporé colliders a cada lado de la cámara. Estos colliders, configurados para colisionar únicamente con el jugador, garantizan que, independientemente de las circunstancias, el jugador permanezca dentro de la vista de la cámara en el eje Y. Este diseño proporciona una experiencia de juego fluida y visualmente coherente, adaptándose eficazmente a diversos escenarios y resolviendo la problemática del retroceso indefinido.

Para abordar el escenario "fuera de cámara," implementé un enfoque simple pero efectivo. Coloqué un collider en la parte inferior del juego, con el layer “enemigo”. De esta manera, cuando el jugador se cae fuera de la vista de la cámara y entra en contacto con este collider, se activa un proceso de redirección que simula la muerte del jugador.

Este método garantiza que caer fuera de la cámara resulte en una consecuencia coherente y predecible, añadiendo un elemento de desafío al juego. Además, al utilizar un collider específico para objetos del layer de enemigos, se evita cualquier interferencia no deseada con otros elementos del juego, manteniendo una ejecución limpia y eficiente de la lógica del juego en situaciones fuera de la vista de la cámara.

Menú/Botones

La configuración de las diferentes pantallas en tu mini-juego, que incluye instrucciones y animaciones, representó una tarea más desafiante de lo inicialmente previsto. La atención dedicada a estos detalles evidencia tu compromiso con ofrecer una experiencia de juego completa y envolvente. Aunque llevó más tiempo del esperado, la inversión en la calidad de las pantallas seguramente contribuirá a una experiencia de usuario más gratificante.



PINGU | Escenas del juego

La inclusión de un botón en la escena del juego para trasladar a una pantalla configurada específicamente para el reinicio del nivel o la salida del juego es una adición útil. Este elemento proporciona a los jugadores una opción clara y accesible para gestionar su progreso en el juego, lo que contribuye a una experiencia más intuitiva y satisfactoria. En resumen, tu enfoque meticuloso en la presentación y la usabilidad demuestra un esfuerzo proactivo para garantizar una experiencia de juego completa y bien diseñada.

CONCLUSIÓN

Esta travesía de crear mi primer juego ha sido todo un viaje emocionante y gratificante en mi recorrido como estudiante de programación. El desafío de materializar un juego funcional en mi primer año, mientras aún estaba aprendiendo los conceptos básicos de la programación, fue un reto formidable. A lo largo de este proceso, experimenté momentos agridulces y enfrenté desafíos que, aunque al principio parecían abrumadores, se convirtieron en oportunidades de aprendizaje valiosas.

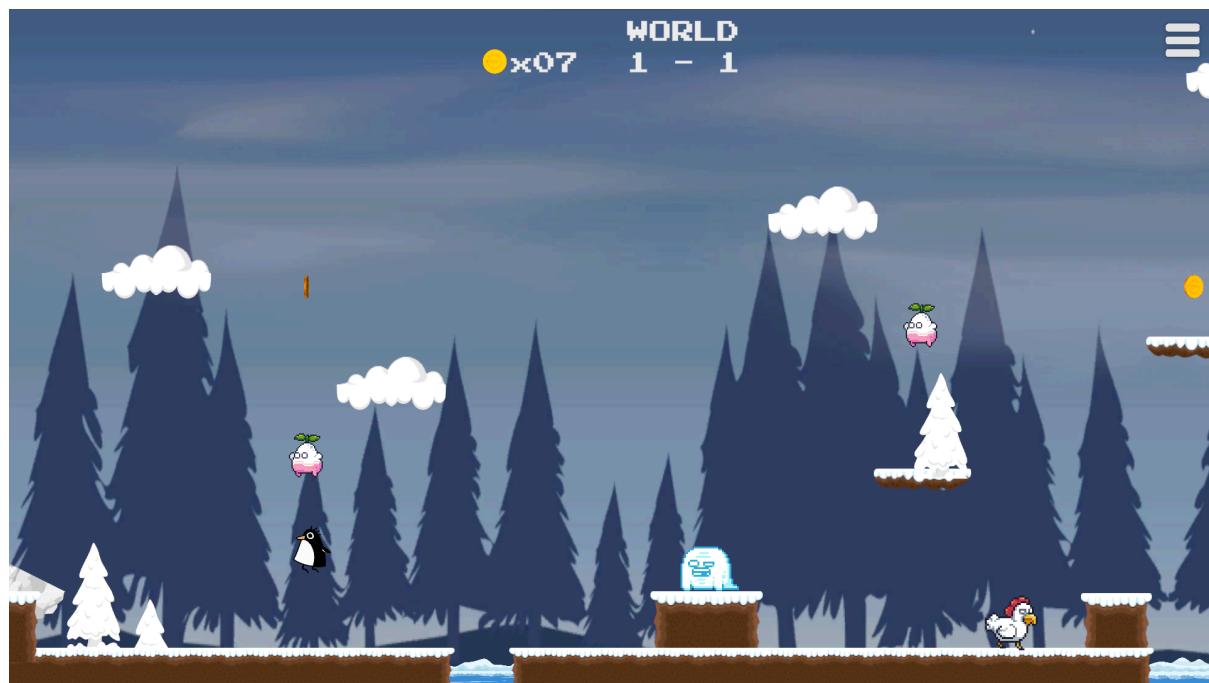
A medida que avanzaba en mis estudios de programación, aplicaba nuevos conocimientos al desarrollo del juego, ajustando y mejorando constantemente mi trabajo. En una ocasión, la pérdida de progreso debido a un pantallazo azul me mantuvo en vilo, pero perseverar en la finalización del juego después de este contratiempo fue una victoria personal.

Este proyecto me ha revelado la complejidad y el esfuerzo detrás de lo que inicialmente parecían tareas simples. Cada línea de código y cada solución encontrada, incluso las que llevaban días, contribuyeron a mi crecimiento como desarrollador. Valorar cada pequeño logro me ha proporcionado una nueva perspectiva sobre el mundo de la programación.

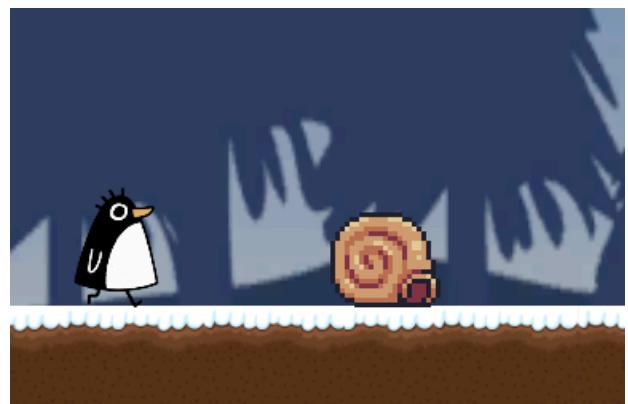
En última instancia, este desafío no sólo ha elevado mi nivel de programación, sino que ha encendido en mí una fascinación por el desarrollo de juegos. La experiencia ha sido gratificante y, aunque ha habido momentos difíciles, ha despertado una curiosidad y un entusiasmo que estoy ansioso por explorar aún más en mi viaje como programador de juegos.

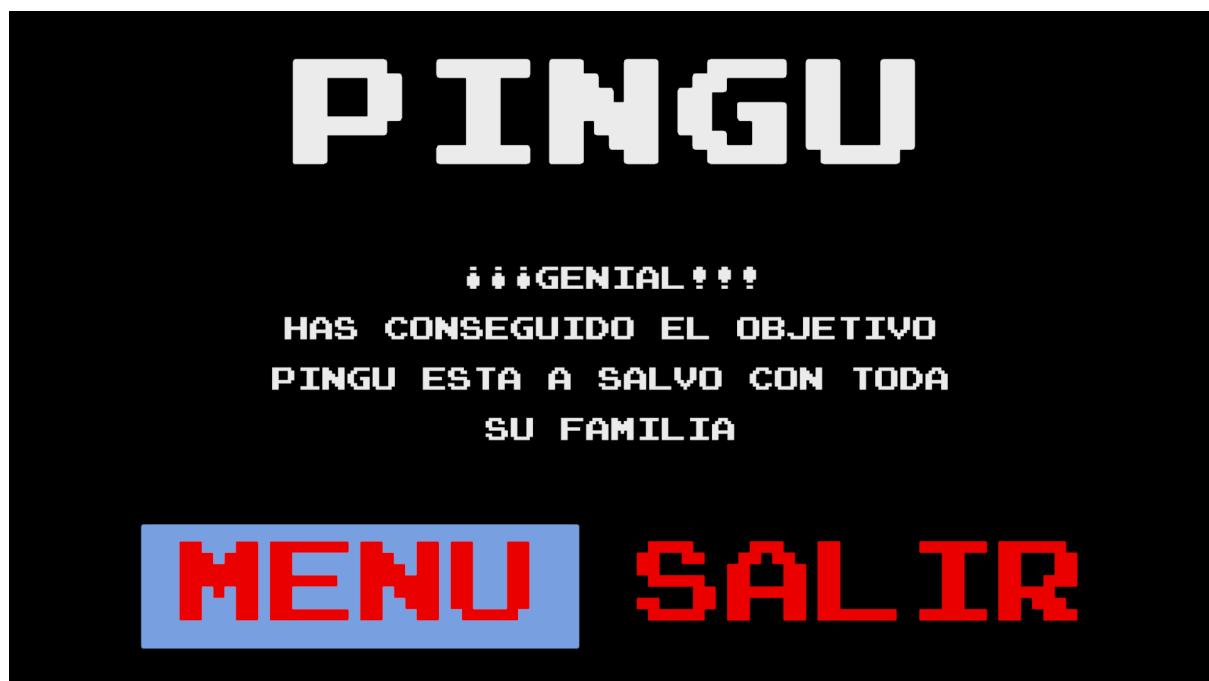
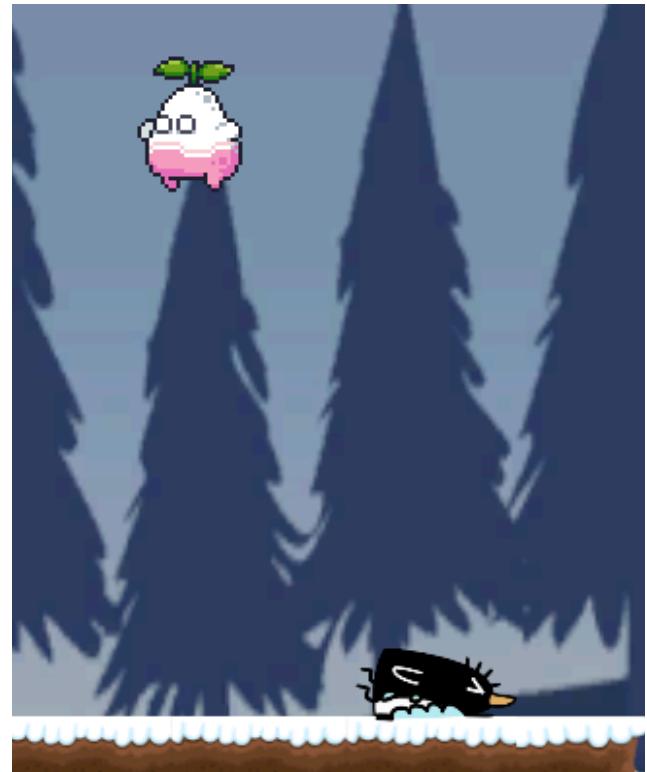
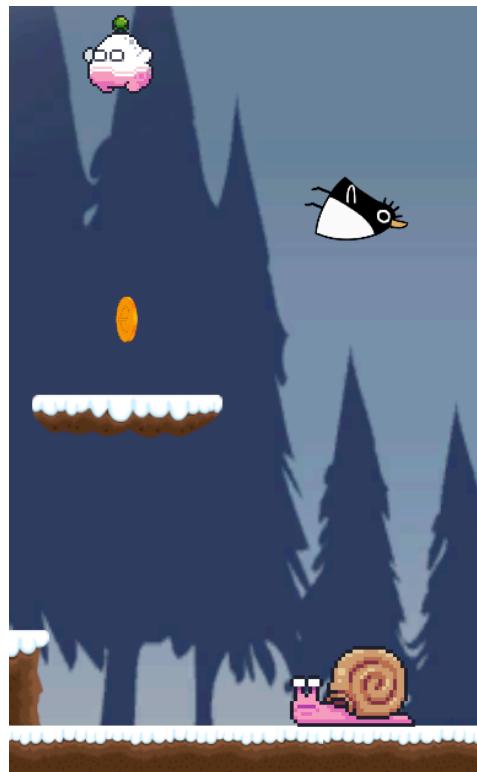
FOTOS DEL JUEGO





Radish Explotando





PROBLEMAS/UNIÓN

En un principio teníamos pensado unir los juegos en la última semana.

Las escenas al no compartir la misma escala empezó a darnos muchos errores, intentamos adaptar los juegos pero algunos dejaban de funcionar las físicas.

Intentamos cambiar las escenas, importar archivos, e incluso dividir los juegos dentro de un mismo proyecto en diferentes carpetas, pero los scripts que compartían librerías como la de TMPro daban errores ya que intentaba importarla tres veces.

Intentamos unificar los scripts que nos daban problemas para así adaptar los juegos a los nuevos scripts pero aún así, los canvas daban problemas por las capas de configuración.

Hemos decidido entregar, aún así, algunas versiones que funcionan a medias, muy a medias para que puedas ver los diferentes intentos que hemos realizado.

Aun así, pido encarecidamente que por favor, que se valore el trabajo de cada juego de manera individual y no las versiones con varios juegos unidos con los errores que estos presentan.

Por último, aclarar, que el juego Pingu, en el editor de Unity lanza una alerta de que no encuentra un Sprite pero es simplemente informativa y un falso positivo. Ya que el sprite, en ese caso, prefab, si lo tiene localizado. Dicha alerta no interfiere en absoluto en la jugabilidad.