Universidad Tecnológica de Santiago

UTESA

Facultad de Arquitectura e Ingeniería



Presentado A:

Iván Mendoza

Asignatura:

INF-910-001

Programación de Videojuegos

Presentado por:

Yoel de Jesús Núñez Mejía

2-17-0171

24 de abril de 2022

Santiago de los caballeros,

República Dominicana

**INTRODUCCIÓN**

En la actualizad existen muchos juegos de diferentes categorías, por la cuales en base a su contenido y la historia que presentan la misma cambia adentrándose en otras clasificaciones mas que en otras. Estos juegos se hacen con la finalidad de entretener a las personas, pero no sim mencionar que es un ambiente de negocios muy lucrativo que compite con la industria del cine.

En este documento conoceremos el desarrollo de creación e implementación de un juego llamado JumpBall, en el cual se desarrolla en un ambiente tipo plataforma en la que el jugador tiene que superar obstáculos y alcanzar el mayor score en los diferentes niveles que tendrá que superar.

En el siguiente documento veremos toda la documentación relacionada a la planificación y desarrollo de esta idea hasta su ejecución.

**CAPÍTULO I: VIDEOJUEGO Y HERRAMIENTAS DE DESARROLLO**

* 1. **Descripción**  
       
     JumpBall consiste en atravesar una serie de obstáculos haciendo rotar una columna por la cual intentas deslizar una pelota sin que tarde atrapado en una de las zonas rojas de las plataformas que se van generando a lo largo del juego. A medidas que aumentas de nivel más difícil es atravesar cada plataforma.

* 1. **Motivación**

Fui motivado por los anuncios que veía de juegos similares a este, lo cual me llevo hacer mi propio juego implementando reglas a modo personal a fin de poder perfeccionarlos y considerar otro modo de juego quizás más difícil y novedoso.

* + 1. **Originalidad de la idea**

He moldeado un juego similar que vi en un anuncio de Instagram a mi gusto implementando nuevas reglas de juego y por ende agregando más dificultad al mismo. Esto con el fin de poner la práctica lo aprendido en Unity.

* + 1. **Estado del Arte**

Actualmente el mundo esta diseñado al igual como el nivel base para desarrollar todos los demás, así como marcadores de escore, puntaje actual, etc.

* 1. **Objetivo general**

El objetivo de este juego es establecer el score más alto por el jugador, el mismo tratara de superar el score antes logrado.

* 1. **Objetivos específicos**

-Evitar chocar las zonas rojas de las plataformas.

-Alcanzar la mayor puntuación estando en modo ultra velocidad, la cual se cumple después de pasar 3 plataformas normal en modo normal y que puedes resurtir la plataforma siguiente al estar en este modo en dado caso de que choques con una plataforma.  
-Superar todos los niveles.

* 1. **Escenario**

EL escenario del juego esta completado en una gran columna por la cual se ira bajando atravesando los obstáculos hasta completa cada nivel que se presente.

* 1. **Contenidos**  
       
     El contenido de este juego no es tan aplico ya que la escena se modifica sobre si misma cambiando aleatoriamente los obstáculos y colores para crear una visión óptica en el jugador y hacerlo pensar que esta en el mismo nivel sin que se de cuenta que la dificultad es mayor.

* 1. **Metodología**

Planificación: Se expondrán todas las ideas generales del juego, así como las reglas y efectos especiales que tendrá el mismo. En pocas palabras se definirán todos los aspectos del juego.

* 1. **Arquitectura de la aplicación**

El juego cerra desarrollado en C#.

* 1. **Herramientas de desarrollo**

-Unity  
-Visual Studio Code  
-Visual Studio

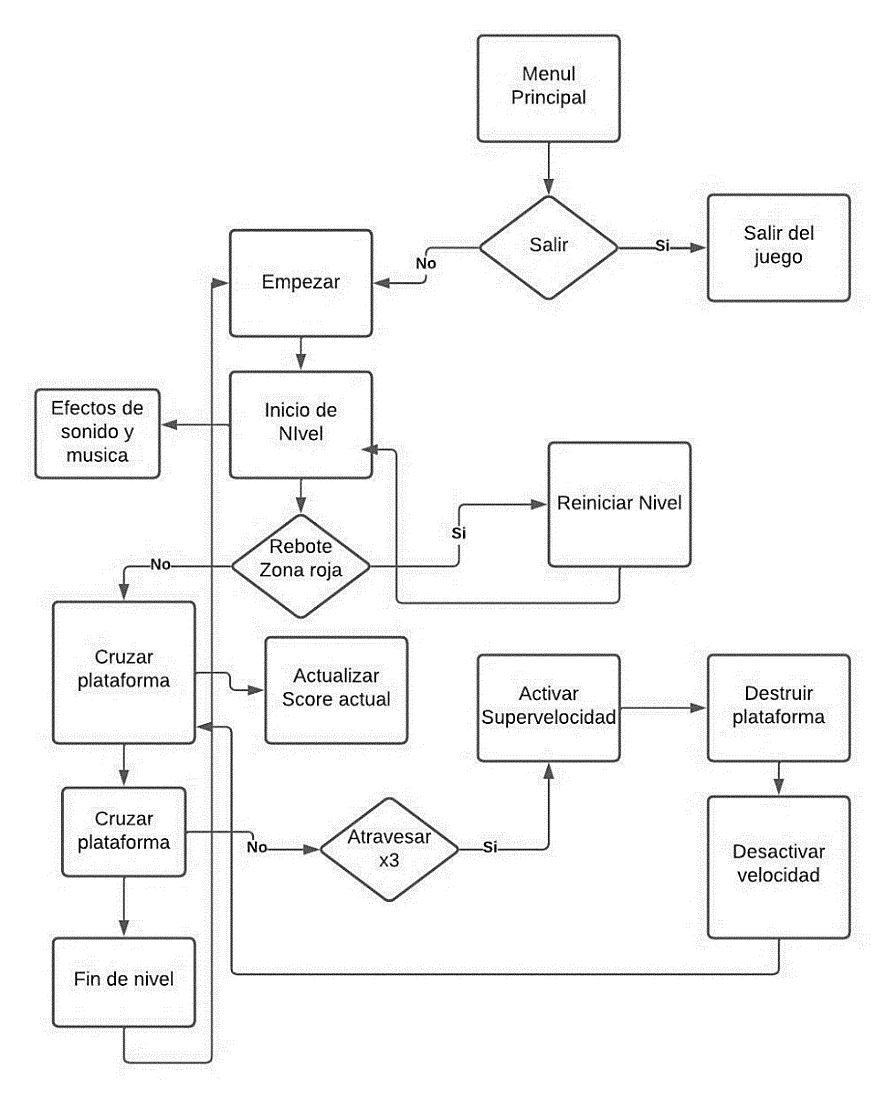
**CAPÍTULO II: DISEÑO E IMPLEMENTACIÓN**

2.1 Planificación (Diagrama de Gantt)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. Actividad | Inicio | Final | Responsable | Estado |
| VIDEOJUEGO Y HERRAMIENTAS DE DESARROLLO | 27 de marzo | 3 de Abril | Yoel | Terminada. |
| Diseño e implementación y elaboración del demo. | 27 de Marzo | 3 de Abril | Yoel | Terminada |
| Continuación de implementación y mejora al demo, agregando efectos de sonidos y más niveles al juego. | 3 de abril | 10 de Abril | Yoel | Terminado |
| Pruebas y verificación de errores. | 3 de abril | 10 de Abril | Yoel | Terminado |
| Optimización de Código y rendimiento del juego. | 3 de abril | 10 de abril | Yoel | Terminado |
| Corregir la tonalidad de las paletas de colores en los niveles | 3 de abril | 10 de abril | Yoel | Terminado |
| Desarrollo Capitulo 3 y 4 | 10 de abril | 17 de abril | Yoel | Terminado |
| Prueba de errores | 24 de abril | 24 de abril |  |  |
| Presentación y Publicación del juego. | 24 de abril | 24 de abril | Yoel | Terminado |
| Campaña de publicidad | 24 de abril | 24 de abril |  | Terminado |

**2.2 Diagramas y Casos de Uso**

**Diagrama de Flujo**



**Casos de Uso**

-Jugador  
Ir al menú principal del juego

Desactivar sonido

Desactivar publicidad

Salir del Juego

-Sistema

Cargar nivel

Reiniciar nivel

Actualizar score actual

Actualizar best Score

**2.3 Plataforma**

Plataforma: PC, Android.

**2.4 Género**

Género: Reacción y velocidad

**2.5 Clasificación**

Clasificación: E

**2.6 Tipo de Animación**

3D  
  
**2.7 Equipo de Trabajo**

Ingenieros de audio: Yoel Nuñez

Diseñadores: Yoel Nuñez

Ilustradores: Yoel Nuñez

Programadores: Yoel Nuñez, lenguaje C#

Animadores: Yoel Nuñez

**2.8 Historia**

En una torre muy alta se encuentra atrapada una bola mágica que debe llegar lo más pronto posible al nivel más bajo porque se encarga de suministrar la energía de los cuidándonos. Tendrás que guiarla por el camino correcto para poder superar todos los obstáculos que tiene que enfrentar.

**2.9 Guion**

La bola será guida por el jugador quien tendrá que dar vueltas a la torre para inducir la bolla por el camino correcto, con efectos de sonidos y música tendrás que tener habilidades de reacción muy rápidos para superar cada obstáculo.

**2.10 Storyboard**

El único escenario se encuentra conformado por figurar geométricas que conforman el mundo las cuales van cambiando de color y contorno.

Me base en figurar geométricas para el desarrollo del juego como son cilindro, triangulo y esfera. Para realizar las plataformas hice un triángulo y fue rotando las posiciones a través del cilindro hasta lograr rodearlo completamente. Y con la ayuda de la programación pude crear los huecos aleatoriamente, así como colorear las zonas rojas que provocan el reinicio del nivel.

La Bola consiste en una esfera que gana velocidad con ciertas restricciones. El mundo y los elementos cambian de color conforme al nivel en el que te encuentras.

Sonidos

Música de fondo, efectos de sonido Colisiones, paso de plataforma.

**2.11 Personajes**

El único personaje existente es una pelota que rebota en las plataformas y la que utilizamos para superar obstáculos.

**2.12 Niveles**

Actualmente hay 100 niveles en los que se dificulta cada vez más la superación de obstáculos y hay más probabilidades de que la posición de la bola se reinicie al colisionar con una zona roja.

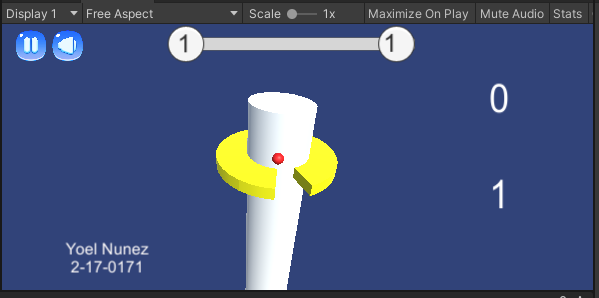
**2.13 Mecánica del Juego**

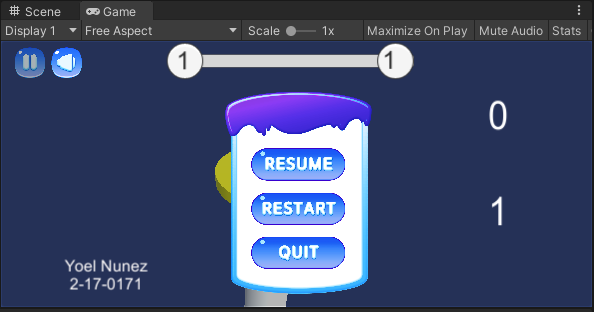
El objetivo de este juego es establecer el score más alto por el jugador, el mismo tratara de superar el score antes logrado.

**CAPÍTULO III: DESARROLLO**

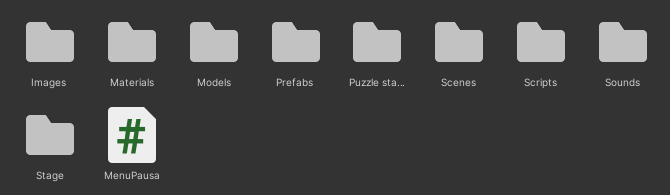
**3.1 Capturas de la Aplicación**

**(Documentación completa del desarrollo, Scripts, Sprites, Prefabs e imágenes)**









**3.1.2 Elementos del Hierarchy**

**Main Camara y Direccional Light**

Se encarga de seguir el elemento Ball (Jugador) a través del desarrollo del juego, así como también de dar iluminación sobras y cambios lumínicos a través de la carga de los elementos del entorno del juego. La luminosidad y sobras con compatibles pensando en los pixeles de celulares y pantallas de pc de gama alta y baja,

Es dirigida por el script CamComtroller y asistetida por el script BallController del elemento Ball.

**Ball**Es el elemento dinamico de nuestro juego en vista de que es la que tiene la mayoría de configuraciones, ya que puede hacer colisiones y a travesar las plataformas, así como también destruirles si alcanza su máxima velocidad en el juego, la cual se obtiene al cruzar 3 plataformas de forma no interrumpida.

Contiene un subelmento llamado Treail que es el rastro de movimiento que va creando la pelota al rebotar o en su movimiento de caída libre.

A través de su Script Ball Controller se pueden controlar las colisiones, posición y su estado, así como también editar su velocidad.

**Helix**  
  
Es el elemento principal de nuestro ya que es el entorno en el que se desenvuelve el personaje con el cual se hacen las mayorías de validaciones y configuraciones entorno a la jugabilidad y experiencia de usuario.

Contiene las plataformas que se requiere que atraviese el jugador para ganar puntos y al llegar al final pasar cada uno de los niveles disponibles en la versión actual del juego.

A través de su Script HelixController se pueden modificar las plataformas y cargar los stage desde el prefab.

**GameManager**  
  
Se encarga de manejar las escenas disponibles del juego y hacer los cambios de lugar.

**Canvas 1**

Dentro de estos elementos se encuentran todos los textos dinámicos que maneja el juego y s encarga de ponerlos activos y de modificarlos cuando se disparen las condiciones contempladas en el. Score, mejor score y el nivel en el que nos encontramos son manejados por este elemento.

Es manejado a través del Script UiManager modificando las propiedades de los textos y en su parte también del hélix.

**Audio Manager**

Se contiene los efectos del sonido del juego que con la ayuda del Script AudioManagerController son llamados a reproducirse en los distintos momentos del juego.

**Canvas 2**

Contiene los elementos del adicionales del juego, como el botón de pausa y sonido. Con la ayuda del Script MenuPausa con llamados cada uno de los métodos del menú.

**3.1.3 Scritps**

**AudioManager🡪 Controlar el efecto de sonido del juego**

public class AudioManager : MonoBehaviour {

public static AudioManager instance;

public Sound[] sounds;

void Start ()

{

if (instance != null)

{

Destroy(gameObject);

} else

{

instance = this;

DontDestroyOnLoad(gameObject);

}

foreach (Sound s in sounds)

{

s.source = gameObject.AddComponent<AudioSource>();

s.source.clip = s.clip;

s.source.volume = s.volume;

s.source.pitch = s.pitch;

s.source.loop = s.loop;

}

}

public void Play (string sound)

{

Sound s = Array.Find(sounds, item => item.name == sound);

s.source.Play();

}

}

**BallController🡪 Configurar velocidad, colisión y movimiento del jugador.**

public class BallController : MonoBehaviour

{

public Rigidbody rb;

public float impulseForce = 3;

private bool ingnoreNextCollision;

private Vector3 starPosition;

public int perfectPass;

public float superSpeed = 8;

private bool IsSuperSpeedActive;

private int perfectPassCount = 3;

public GameObject splash;

//public AudioSource BounceAudio;

private void Start()

{

starPosition = transform.position;

}

private void OnCollisionEnter(Collision collision)

{

//BounceAudio.Play();

FindObjectOfType<AudioManager>().Play("bounce");

AddSplash(collision);

if (ingnoreNextCollision)

{

return;

}

//Destruir plataforma con supervelocidad

if (IsSuperSpeedActive && !collision.transform.GetComponent<GoalController>())

{

Destroy(collision.transform.parent.gameObject, 0.2f);

}

else

{

DeathPart deathPart = collision.transform.GetComponent<DeathPart>();

if (deathPart)

{

GameManager.singleton.RestartLevel();

FindObjectOfType<AudioManager>().Play("game over");

}

}

//GameManager.singleton.AddScore(1);

rb.velocity = Vector3.zero;

rb.AddForce(Vector3.up \* impulseForce, ForceMode.Impulse);

ingnoreNextCollision = true;

Invoke("AllownextCollision", 0.2f);

perfectPass = 0;

IsSuperSpeedActive = false;

}

private void AllownextCollision()

{

ingnoreNextCollision = false;

}

public void ResetBall()

{

transform.position = starPosition;

}

private void Update()

{

if (perfectPass >= perfectPassCount && !IsSuperSpeedActive)

{

IsSuperSpeedActive = true;

rb.AddForce(Vector3.down \* superSpeed, ForceMode.Impulse);

}

// FindObjectOfType<AudioManager>().Play("whoosh");

}

public void AddSplash( Collision collision)

{

GameObject newSplash;

newSplash = Instantiate(splash);

newSplash.transform.SetParent(collision.transform);

newSplash.transform.position = new Vector3(this.transform.position.x, this.transform.position.y - 0.11f, this.transform.position.z);

Destroy(newSplash, 3);

}

}

**CamController🡪 Seguimiento e iluminación del jugador.**

public class CamController : MonoBehaviour

{

public BallController ball;

private float offset;

// Start is called before the first frame update

void Start()

{

offset = transform.position.y - ball.transform.position.y;

}

// Update is called once per frame

void Update()

{

Vector3 actualPos = transform.position;

actualPos.y = ball.transform.position.y+ offset;

transform.position = actualPos;

}

}

DeathPart🡪 Propiedades de las zonas de muerte del jugador y generar las mismas aleatorias.

public class DeathPart : MonoBehaviour

{

private void OnEnable()

{

GetComponent<Renderer>().material.color = Color.red;

}

}

GoalController🡪 Indica el final y comienzo de un nivel

public class GoalController : MonoBehaviour

{

private void OnCollisionEnter(Collision collision)

{

GameManager.singleton.NextLevel();

}

}

HelixController🡪 Configuración general de las plataformas y carga de niveles

public class HellixController : MonoBehaviour

{

private Vector2 lastTapPosition;

private Vector3 starPosition;

public Transform topTransform;

public Transform goalTransform;

public GameObject helixLevelPrefab;

public List<Stage> allStages = new List<Stage>();

public float helixDistance;

private List<GameObject> spawnedLevels = new List<GameObject>();

private void Awake()

{ //Saber posicion inicial del Helix

starPosition = transform.localEulerAngles;

helixDistance = topTransform.localPosition.y - (goalTransform.localPosition.y + .1f);

LoadStage(0);

}

// Update is called once per frame

void Update()

{

//Si hacemos click

if (Input.GetMouseButton(0))

{

Vector2 currentTapPosition = Input.mousePosition;

//si no hemos tocado la pantalla

if (lastTapPosition == Vector2.zero)

{

lastTapPosition = currentTapPosition;

}

//saber la distancia que se ha movido

float distance = lastTapPosition.x - currentTapPosition.x;

lastTapPosition = currentTapPosition;

//rotar el helix

transform.Rotate(Vector3.up \* distance);

//saber si dejamos de pulsar la pantalla

if (Input.GetMouseButtonUp(0))

{

lastTapPosition = Vector2.zero;

}

}

}

//Cargar el nivel

public void LoadStage(int stageNumber)

{

Stage stage = allStages[Mathf.Clamp(stageNumber, 0, allStages.Count - 1)];

if (stage == null)

{

Debug.Log("No hay mundo creado");

return;

}

Camera.main.backgroundColor = allStages[stageNumber].stageBackground;

FindObjectOfType<BallController>().GetComponent<Renderer>().material.color = allStages[stageNumber].stageBallColor;

transform.localEulerAngles = starPosition;

foreach (GameObject go in spawnedLevels)

{

Destroy(go);

}

float levelDistance = helixDistance / stage.levels.Count;

float spawnPosy = topTransform.localPosition.y;

for (int i = 0; i < stage.levels.Count; i++)

{

spawnPosy -= levelDistance;

GameObject level = Instantiate(helixLevelPrefab, transform);

level.transform.localPosition = new Vector3(0, spawnPosy, 0);

spawnedLevels.Add(level);

int partToDisable = 12 - stage.levels[i].partCount;

List<GameObject> disabledParts = new List<GameObject>();

while (disabledParts.Count < partToDisable)

{

GameObject randomPart = level.transform.GetChild(Random.Range(0, level.transform.childCount)).gameObject;

if (!disabledParts.Contains(randomPart))

{

randomPart.SetActive(false);

disabledParts.Add(randomPart);

}

}

List<GameObject> leftParts = new List<GameObject>();

foreach(Transform t in level.transform)

{

t.GetComponent<Renderer>().material.color = allStages[stageNumber].StageLevelPartColor;

if (t.gameObject.activeInHierarchy)

{

leftParts.Add(t.gameObject);

}

}

List<GameObject> deathParts = new List<GameObject>();

while (deathParts.Count < stage.levels[i].deathPartCount)

{

GameObject randomPart = leftParts[(Random.Range(0, leftParts.Count))];

if (!deathParts.Contains(randomPart))

{

randomPart.gameObject.AddComponent<DeathPart>();

deathParts.Add(randomPart);

}

}

}

}

}

PassScore🡪 Define los puntos del jugador durante el transcurso del juego

public class PassScorePoint : MonoBehaviour

{

private void OnTriggerEnter(Collider other)

{

GameManager.singleton.AddScore(1);

FindObjectOfType<BallController>().perfectPass++;

}

}

**Sound🡪 Edita las propiedades de los audios**

public class Sound {

public string name;

public AudioClip clip;

[Range(0f, 1f)]

public float volume;

[Range(-3f, 3f)]

public float pitch;

public bool loop = false;

[HideInInspector]

public AudioSource source;

}

**Stage🡪 crea, edita y carga los niveles del juego,**

[Serializable]

public class Level

{

//Crear objeto para los niveles

[Range(1, 11)]

public int partCount = 11;

[Range(0, 11)]

public int deathPartCount = 1;

}

[CreateAssetMenu(fileName="Nivel")]

public class Stage : ScriptableObject

{

public Color stageBackground = Color.white;

public Color StageLevelPartColor = Color.white;

public Color stageBallColor = Color.white;

public List<Level> levels =new List<Level>();

}

**UiManager🡪 Controla los textos**

public class UiManager : MonoBehaviour

{

public Text currentScoreText;

public Text bestScoreText;

public Slider slider;

public Text actualLevel;

public Text nextLevel;

public Transform topTransform;

public Transform goalTransform;

public Transform Ball;

// Update is called once per frame

void Update()

{

currentScoreText.text ="Score: "+GameManager.singleton.currentScore;

bestScoreText.text = "Best: " + GameManager.singleton.bestScore;

ChangeSliderLevelProgress();

}

public void ChangeSliderLevelProgress()

{

Debug.Log("Entro al metodo de cargar el Slider");

actualLevel.text =""+(GameManager.singleton.currentLevel+1);

nextLevel.text =""+(GameManager.singleton.currentLevel+2);

float totalDistance = (topTransform.position.y - goalTransform.position.y);

float distanceleft = totalDistance - (Ball.position.y - goalTransform.position.y);

float value = (distanceleft / totalDistance);

slider.value = Mathf.Lerp(slider.value,value,5);

}

}

**3.1.4 Sprites**

**HelixGoal**  
  
Consiste en una circunferencia



**Línea de meta y el inicio del nivel actual.**



**HelixPart y HelixDeath**

Con las partes en las que el jugador se posicione en zona segura o muerta las cuales consisten en una proporción de la circunferencia.

**Splash**



Es el efecto que se crea al momento de que la pelota colisiona con una plataforma.

**3.1.5 Imágenes**

Contienen los elementos del menú de pausa y sus botones.











**3.2 Prototipos**

Fase Inicial y Desarrollo

Primera parte:

-El juego tenia solo los componentes agregados sin ningún efecto de audio sonido, tampoco efectos visuales ni menú, solo había 1 nivel disponible.

Segunda parte

- Se configuraron las reglas del juego, así como el primer grado de dificultad y apariencia del entorno del juego, creación de Stage automáticos.

Fase Final

Adicionar efectos audios, apariencia visual, corrección de luces y efectos. Modificar la complejidad del juego aumentando zonas de muerte y velocidad de la bola.

**3.3 Perfiles de Usuarios**

Se pretende alcanzar un gran numero de usuarios con este juego. Debido a la clasificación del juego esta hecho para un publico mayor a 4 años.

3.4 **Usabilidad**

La mecánica del juego es sencilla ya sea que el usuario se encuentre en su celular o en su pc. AL estar en un computador tendrá la facilidad de hacer girar la torre con el mouse el cual debe arrastras de derecha a izquierda sin soltar el botón izquierdo del mouse.

En el caso del smartphone solo tendrá que tocar la pantalla y mover sus dedos de derecho a izquierda para hacer girar la torre hacia la dirección que el convenga según los obstáculos presentados durante el juego.

**3.5 Test**

Para las pruebas se eligieron 2 usuarios de diferentes edades y los invitamos a jugar nuestro juego.

Persona 1

|  |  |
| --- | --- |
| **Aspectos** | **Datos** |
| Sexo | M |
| Edad | 10 |
| Nivel de Estudio | Primarios |
| Aficiones | Estudiante |
| **Aspectos de Juego** | **Puntuación 1-10** |
| Jugabilidad | 10 |
| Dificultad | 6 |
| Guía de usuario | 9 |
| Apariencia Visual | 8 |

Persona 2

|  |  |
| --- | --- |
| **Aspectos** | **Datos** |
| Sexo | F |
| Edad | 19 |
| Nivel de Estudio | Secundarios |
| Aficiones | Contabilidad |
| **Aspectos de Juego** | **Puntuación 1-10** |
| Jugabilidad | 10 |
| Dificultad | 3 |
| Guía de usuario | 8 |
| Apariencia Visual | 7 |

**3.6 Versiones de la Aplicación**

Versión 0

Incluía valores por defecto del entorno visual y jugabilidad solo tenia un nivel y era la versión utilizada para probar el desarrollo del código.

Versión 0.5 Beta

Se mejoro la experiencia de juego y dificultad, así como efectos de audio y 7 niveles mas para probar la jugabilidad.

Versión de lanzamiento 1.0

Mejoras al sistema de audio, código y efectos visuales con errores corregidos.

**CAPÍTULO IV: PUBLICACIÓN**

**4.1 Requisitos de la instalación**

**Requisitos PC**

Requisitos Mínimos

SO: Microsoft Windows 2000 / XP

Procesador: Procesador Intel Pentium III a 800 MHz / AMD Athlon a 800 MHz / Intel Celeron a 1.2 GHz / AMD Duron a 1.2 GHz

Memoria: 128 MB de RAM

Gráficos: Dispositivo de 32 MB con controladores compatibles con DirectX 9.0 ("GeForce" o superior)

DirectX: Microsoft DirectX 9.0

Disco Duro: 100 MB

Tarjeta de Sonido: Tarjeta de sonido con DirectX 9.0

-Descargar Jumball.exe

**Requisitos mínimos en Android**

Android 4.4 o versiones más avanzadas.

Espacio disponible de 57MB.

Memoria RAM de 1 GB.

-Descargar APK

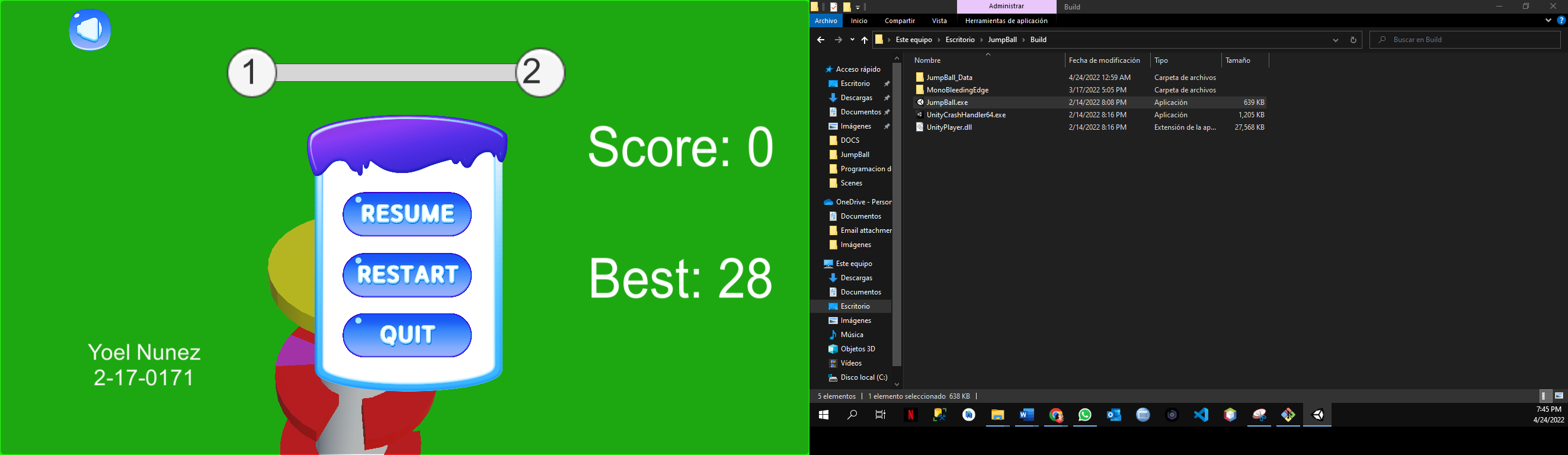
**4.2 Instrucciones de Uso**

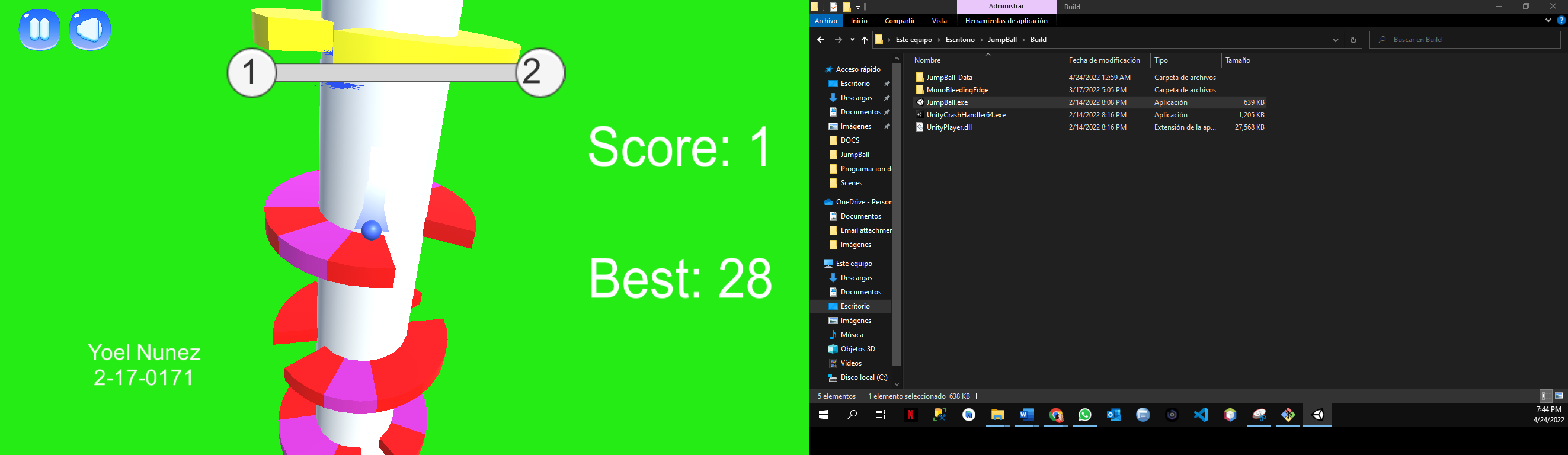
El juego tendrás que hacer girar la torre tocando la pantalla y deslizando de derecha a izquierda o viceversa según sea conveniente para atravesar la plataforma, teniendo pendiente que puedes caer en una zona de muerte al cruzar por eso deberás observar bien donde caer.

El cruzar tres plataformas continuas sin colisionar puedes activar la super velocidad, con la que puedes romper la siguiente plataforma con la que colisiones.

**4.3 Bugs**

-Cuando el juego está en pausa puedes hacer girar la torre.





La zona de colisión de la pelota con la plataforma está un poco alejada de la plataforma por lo que si no giras rápido la plataforma al girar y apenas roza el nivel se puede reiniciar.

**4.4 Proyección a Futuro**

Para hacer el juego conocido se hará una campaña de promoción en las redes sociales, creando una campana que logre llegar a la mayor cantidad de usuarios posibles que visualicen otros juegos busquen juegos parecidos a este.

Se elaborará una campana también en YouTube con a través de Google ADS para que este juego sea visto por aun más personas mientras buscan gameplays de juegos de la misma clasificación y categoría.

Se hará dará una especie de recompensa hacienda un torneo donde lo usuarios compartan en su historia mencionando la cuenta principal del juego mostrando el score alcanzado y el más alto recibirá la recompensa.

**4.5 Presupuesto**

|  |  |  |
| --- | --- | --- |
| **ROL O INVERSION** | **TIEMPO** | **COSTO** |
| Diseñador Grafico | Tiempo Completo | 1,250. US Mensual |
| Programador | Tiempo Completo | 2,500 US Mensual |
| Testers | Mensual | 200 US Por Pruebas |
| Servicios Web en la Nube | Tiempo Completo | 3,500 US |
| Redes Sociales Marketing | Tiempo Completo | 1,000 US Mensuales |
| Total |  | 8,450 US Mensuales |

**4.6 Análisis de Mercado**

Entre el 2020 y el 2021 el sector gaming se volvió en uno de los más relevantes y con menores afectaciones a causa de COVID-19, pues al estar los usuarios en casa se incrementó el número de gamers en el mundo, además del gasto que se hizo dentro de aplicaciones o títulos. De acuerdo con un reporte de la firma de análisis NewZoo, en el 2021 los ingresos que tuvo este sector fueron de 180,300 millones de dólares, un 1.4% más de lo que tuvo en 2020.

Dentro de los mayores ingresos, Wijman destaca el mercado de mobile gaming es el que mayores ingresos tiene dentro de sus apps de gaming, pues el 52% de los ingresos que hubo en 2021, fueron directamente en videojuegos móviles.

Del mismo modo, las audiencias de juegos y contenido de juegos han seguido creciendo, incluso después de la incorporación sin precedentes de nuevos jugadores en 2020, pues durante este año hubo más de 3,000 millones de nuevos gamers y de estos el 55% provino de Asia Pacífico.

Fuente: https://expansion.mx/tecnologia/2021/12/30/el-mercado-de-videojuegos-en-2021.

**4.7 Viabilidad**

-Las campañas de publicidad están en su mayor auge por lo que el juego llegara a muchas personas.

-Se puede incluir anuncios en el juego y de esta forma generar ingresos

-Gestionar el copyright

-La mayoría de personas dispone de un computador portátil y un smartphone por lo que la viabilidad de este proyecto se ve muy bien si todo de implemente correctamente.

**Conclusión**

El juego presentado en este documento centra su tendencia en ser apto para todo publico y trata de ser adictivo para las personas, pues al final tendrás que optar por no rendir tras escuchar el sonido del juego una y otra vez mientras intenta pasar los diferentes niveles existentes en él. Estamos seguros de que este juego llegara a muchas personas debido a su contenido fácil jugabilidad.

Hemos demostrado que este proyecto en viable ya que muchas personas juegan esta clase de juegos sencillos en su teléfono celular o pc mientras no tienen conexión o simplemente buscan una forma de recrearse.

Enlaces

GitHub  
  
<https://github.com/area002/JumpBall>

Enlace del Juego Publicado

<https://area0022.itch.io/jumpball>

Video en YouTube