UNIVERSIDAD TECNOLOGICA DE SANTIAGO (UTESA)

Sistema Corporativo
Facultad de Arquitectura e Ingeniería
Carrera de Ingeniería en Sistemas Computacionales

Programación de videojuegos
Entrega Final del Videojuego
Breakout

Presentado a:
Prof. Iván Mendoza

Presentado por:

Alberto de Jesús García Peña	2-17-1097
Jonathan Daniel Rodríguez Valenzuela	1-18-0622
Luis Arturo Rodríguez Rodríguez	1-18-0858

Santiago, Provincia Santiago de Los Caballeros,
República Dominicana.
Agosto, 2021.

Introducción

El siguiente proyecto es una recopilación tanto del esfuerzo como la nostalgia recogida por todos nuestros compañeros para la creación de un juego ya conocido y aunque no muy utilizado en estos días, pero que sigue vivo en nuestros corazones: Breakout. Este proyecto contiene nuestras motivaciones, un resumen de los datos del juego, además de muestras de los diferentes niveles, scripts y pantalla realizadas, además de un plan de negocios para la publicación del mismo e ideas para su mejora en años venideros.

ÍNDICE

CAPÍTULO I: VIDEOJUEGO Y HERRAMIENTAS DE DESARROLLO

1.1. DESCRIPCIÓN

Este juego es una versión moderna de Breakout, un juego de arcade lanzado por Atari en 1976. Tiene como objetivo romper bloques con el uso de una raqueta y una bola, además, no se debe dejar caer la bola porque es la forma en la que se puede perder el juego. En el desarrollo del juego, el jugador tiene la oportunidad de obtener power-ups que le dan ventajas como hacer más grande la raqueta o tener muchas bolas al mismo tiempo y desventajas como hacer la raqueta más corta. También cuenta con distintos niveles que se vuelven más complejos a medida que avanza el juego, poniendo a prueba las habilidades del jugador.

1.2. MOTIVACIÓN

Todos los integrantes del grupo hemos tenido alguna vez la oportunidad de jugar algún juego al estilo de breakout, ya sea desde uno de los teléfonos móviles, PC o incluso desde alguna consola portátil que llegamos a utilizar cuando éramos pequeños.

Mientras decidíamos cuál de los juegos tomar para el desarrollo del proyecto final estuvimos recordando cuando fue que jugamos un videojuego por primera vez, y la mayoría concidimos en que nuestro primer videojuego vino de la mano de un teléfono móvil. Snake, Tetris, Soccer e incluso Daimond Rush fueron de aquellos juegos que alegraron nuestra niñez, pero el que sin duda ha quedado en nuestra mente con más fuerza ha sido aquellos juegos al estilo breakout. Pasan los años y aun podemos tener la oportunidad de poder jugar este juego, por lo que podríamos decir que nuestra motivación aparte de que disfrutamos este juego es la nostalgia que nos trae poder crear un juego de este tipo.

Como inspiración tenemos al Alleyway que llego junto con nuestros Game Boys años atrás. Y las diferentes adaptaciones que podemos encontrar de este tipo de juegos en la Play Store o en Google Play, sin mencionar los que podemos descargar para la PC como BreakOut Invaders. Este juego nos proporciona por lo tanto una gran cantidad de información de apoyo y detalles que podemos permitirnos utilizar dentro del desarrollo de nuestro proyecto.

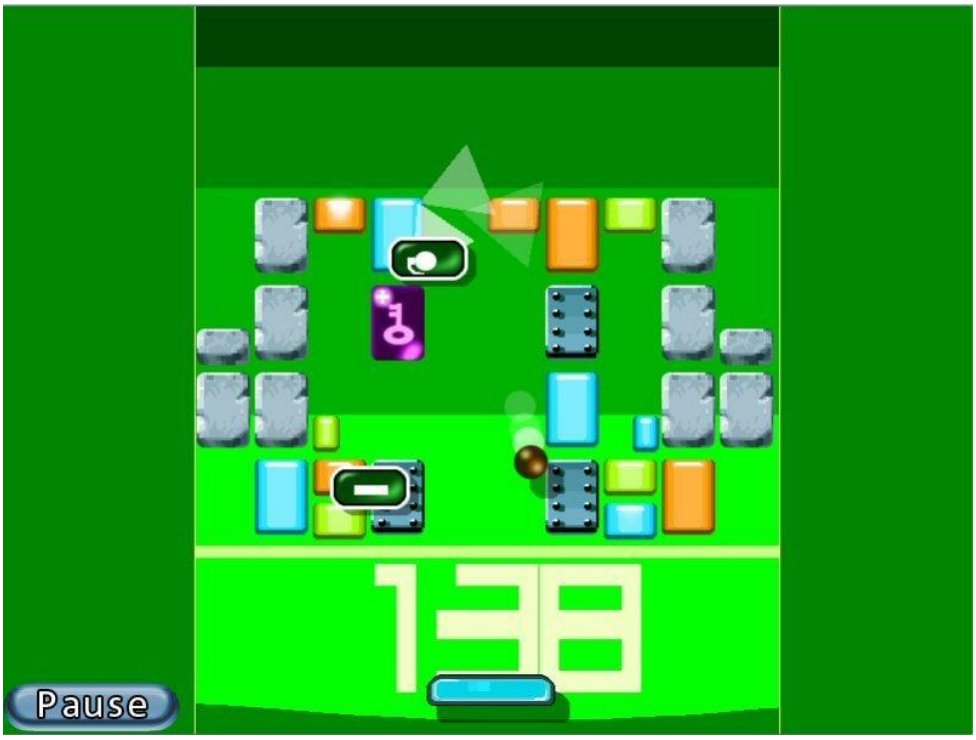
1.2.1. ORIGINALIDAD DE LA IDEA

En este apartado buscamos utilizar las bases del juego y darles un desarrollo más profundo en el apartado de la historia, ya que pocos de estos juegos poseen una historia más allá de avanzar en los niveles o alcanzar el mayor puntaje posible. Otro detalle a destacar es que se desea agregar una mayor dificultad con la implementación de diferentes enemigos según los niveles, lo que tendrá al jugador pendiente tanto de la seguridad de la raqueta como de la seguridad de la bola para que esta sigue en juego, esto irá en conjunto con los objetos típicos del juego, como objetos que mejoraran o empeoraran la dificultad para el usuario.

1.2.2. ESTADO DEL ARTE

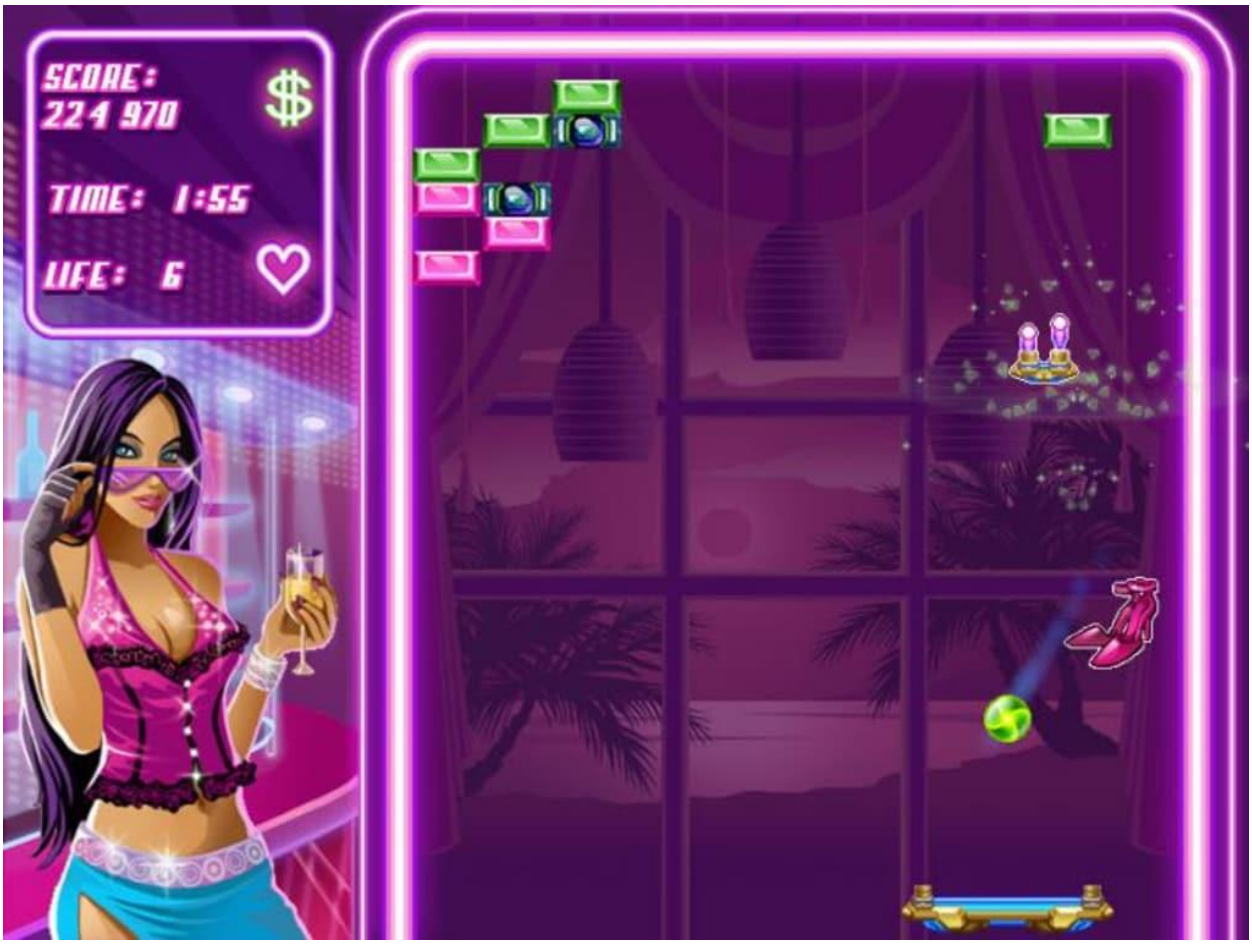
Los gráficos a utilizar dentro del juego son como la mayoría de sus antecesores, en 2D, esto para mantener el estilo artístico que ha destacado en este tipo de juegos durante años. Entre los principales ejemplos de juegos que utilizan este estilo artístico están:

- **Brick Breaker Revolution; desarrollador: Digital Chocolate.**



- **Break Breaker Star; desarrollador: Springcomes**

- **Block Breaker Deluxe; desarrollador: Gameloft**



1.3. OBJETIVO GENERAL

- Completar el nivel.
- Evitar ser destruido.

1.4. OBJETIVOS ESPECÍFICOS

- Destruir todos los bloques.
- Obtener las bonificaciones.
- Obtener la mayor puntuación del nivel.
- Destruir a los posibles enemigos.

1.5. ESCENARIO

El videojuego cuenta con un escenario único en donde se van a desarrollar todos los niveles. Este escenario tendrá una ambientación, que hará alusiones al espacio exterior, para tratar de ubicar al jugador entre la historia que se quiere perseguir con el juego. Este también tiene diferentes tipos de bloques, por lo que hay algunos más difíciles de romper que otros. La cantidad de bloques y la combinación de los mismos va a depender del nivel en el que se encuentre el jugador. Esto significa que, mientras más alto sea el nivel, la cantidad de bloques será mayor y la combinación de los bloques será complicada.

1.6. CONTENIDOS

El videojuego contiene:

- **Raqueta:** Vía por la que el jugador se puede mover.
- **Bloques:** Son los obstáculos que el jugador tiene que vencer para poder ganar el videojuego.
- **Bola(s):** Vía por la que el jugador puede romper los bloques para poder ganar el videojuego.
- **Bala(s):** Otra vía por la que el jugador puede romper los bloques del videojuego.
- **Power-ups o potenciadores:** Objetos que generan un efecto positivo o negativo en el juego al momento de ser recogidos por el jugador usando la raqueta.
- **Marcadores:** Presentan algunos datos del juego en curso. Estos datos son la puntuación, las vidas restantes y la cantidad de bloques presentes en el videojuego.
- **Enemigos:** Serán objetos, los cuales aparecerán cada cierto tiempo para disparar al jugador, con el objetivo de dificultarle el gameplay.

1.7. METODOLOGÍA

La metodología utilizada para el desarrollo del videojuego es la de SUM, ya que esta se ajustaba a las características de nuestro grupo de desarrollo y al tiempo en el que se desea realizar el proyecto. Las diferentes fases en las que se divide esta metodología de desarrollo

son las siguientes:

Roles: Lo primero que se realiza en esta metodología son la definición de los Roles de cada miembro del equipo, distribuyéndonoslas tareas de desarrollo, productor y probadores del juego, entre los miembros del equipo.

Ciclo de vida: Esta es la etapa en la que definimos toda la vida de desarrollo del videojuego, esta tiene diferentes fases en las cuales se va llevando a cabo el desarrollo del video juego, entre estas fases están:

Concepto: Definimos los aspectos del videojuego, con relación a los elementos que este contendrá, como por ejemplo las principales características, como debería ser el gameplay del juego, los niveles que tendría, como se desarrollaría el videojuego a lo largo del tiempo y también las herramientas que íbamos a utilizar, como lo fueron Unity 5 y VS Code.

Planificación: Aquí procedimos a definir el tiempo con el que contábamos y a realizar una división temporal entre las actividades que debíamos realizar, definiendo la importancia de cada una, para así elegir cuales tareas eran obligatorias de hacer y cuales más podía hacer el equipo en el tiempo que contábamos.

Elaboración: En esta etapa fue en la cual nos pusimos manos a la obra con el desarrollo de nuestro videojuego, con el objetivo de lograr una versión funcional del juego al finalizar esta etapa. Implementamos todas las ideas que fueron planeadas en etapas anteriores, respetando el espacio temporal de cada una y desarrollamos otras más de lo que se tenía pensado. Finalizando esta etapa también se realizó una evaluación en cuanto al estado en el que se encontraba nuestro videojuego.

Beta: Esta fase se realizó con el propósito de que pudiéramos experimentar la experiencia del juego y poder determinar los errores que no se pudieron detectar en las etapas anteriores, también se procedió a corregir dichos errores, al igual que ajustar la experiencia que se tenía al jugar.

Cierre: Esta fase se llevará a cabo como la entrega final del videojuego, se dará a cabo cuando los involucrados en el proyecto podamos llegar a un acuerdo entre todas las partes que consideren que el juego se encuentre en un buen estado para ser publicado y por supuesto antes del plazo temporal que se tiene.

Gestión de Riesgos: Esta fue una fase que se llevó a cabo junto con cada una de las demás fases, en especial en la de elaboración del videojuego, ya que esta contenía diferentes planes, para evitar que se complicara cualquiera de las diferentes fases, permitiéndonos llegar al final del videojuego en el tiempo establecido.

1.8. ARQUITECTURA DE LA APLICACIÓN

La arquitectura utilizada para desarrollar este videojuego, se necesitaba que permitiera la reutilización de los objetos y que al mismo tiempo nos permitiera poder desarrollar el juego en un principio para la plataforma de Windows y Web, pero que en un futuro nos permita poder optimizar el juego para poder ejecutarlos en dispositivos móviles. Por lo que decidimos utilizar una Arquitectura por Capas, en la cual tenemos la capa principal conteniendo los manejadores del juego, del audio, de los niveles y de carga del juego, y luego tenemos capas inferiores en las que se encuentran alojados todos los aspectos UI del juego.

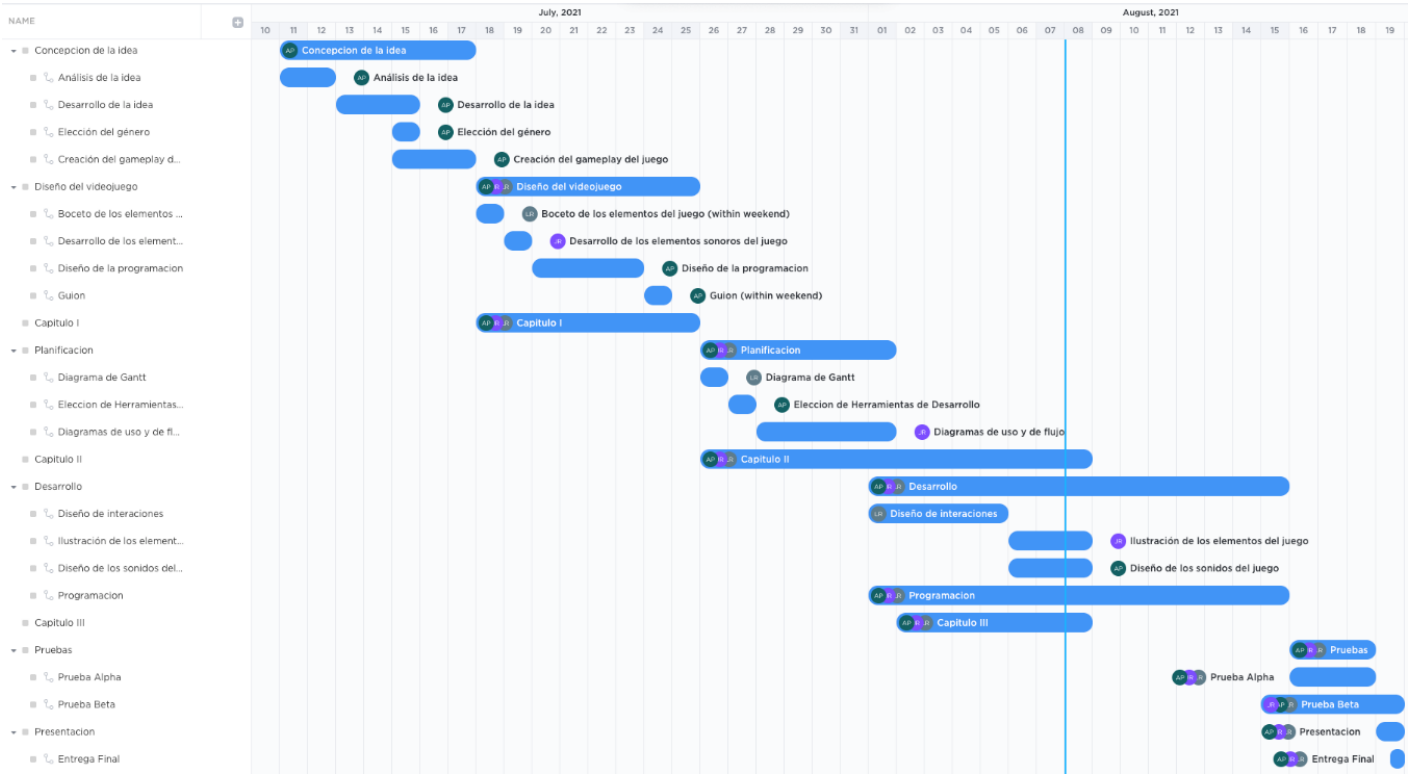
1.9. HERRAMIENTAS DE DESARROLLO

Las herramientas que utilizamos en el desarrollo de nuestro videojuego, están encabezadas por el motor de Unity 5, para desarrollar toda la parte con relación al juego. También utilizamos lo que viene siendo Visual Studio Code, como la herramienta de desarrollo de toda la parte del código necesario para el funcionamiento del videojuego.

CAPÍTULO II: DISEÑO E IMPLEMENTACIÓN

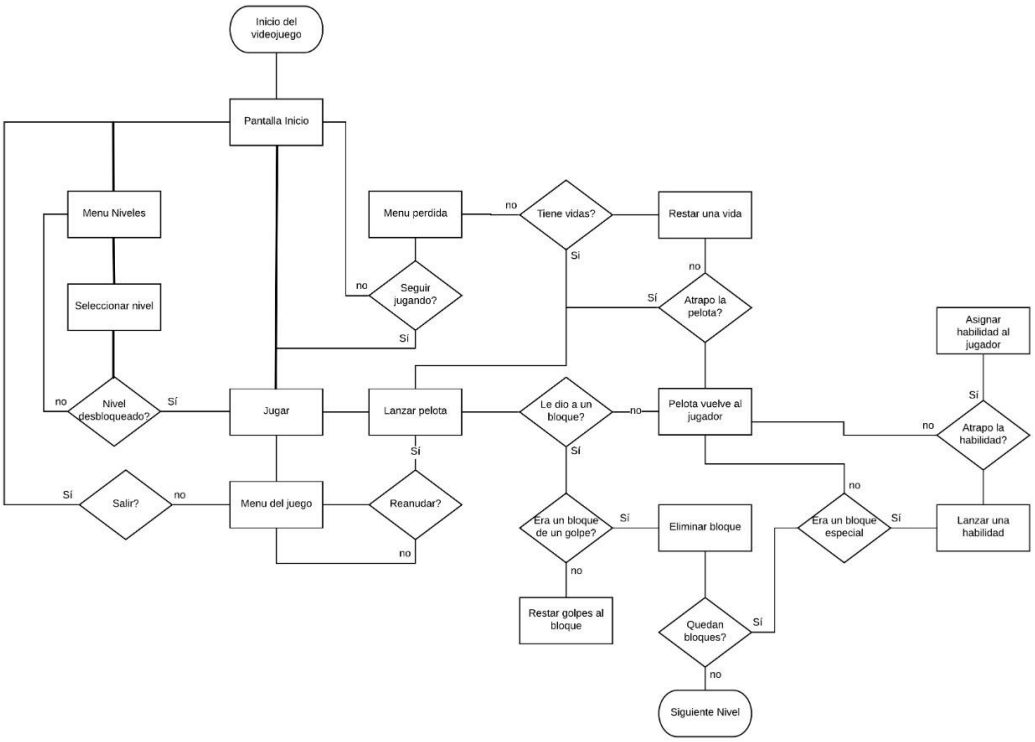
2.1. PLANIFICACION (DIAGRAMA DE GANTT)

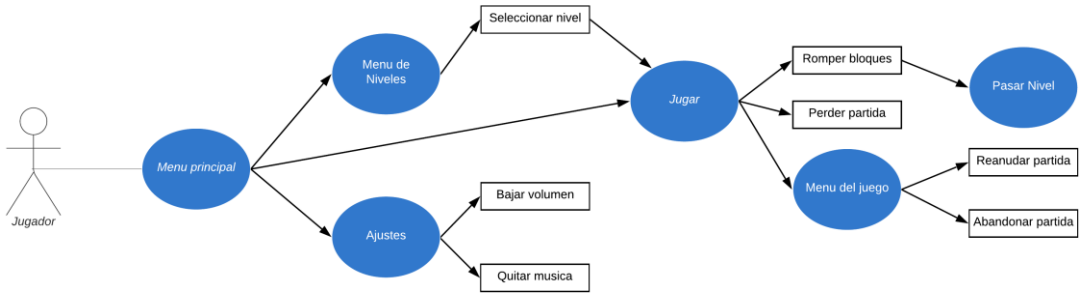
La foto también se encuentra anexada en la carpeta, para una mejor visibilidad.



2.2. DIAGRAMAS Y CASOS DE USO

Las fotos también se encuentran anexada en la carpeta, para una mejor visibilidad.





2.3. PLATAFORMA

Como será un juego web este podrá ser utilizado en varias plataformas, tanto en PC como en móvil.

2.4. GÉNERO

Por el tipo de juego que es este se clasificara dentro del género arcade.

2.5. CLASIFICACIÓN

Al ser un juego arcade, es un juego que no presentará un lenguaje no apropiado o algún tipo de violencia, por lo que este puede ser jugado por personas de todas las edades, lo que lo puede colocar en la clasificación para todas las edades o E.

2.6. TIPO DE ANIMACIÓN

La animación será de estilo 2D.

2.7. EQUIPO DE TRABAJO

Ingenieros de audio: Alberto García y Jonathan Rodríguez

Diseñadores: Luis Arturo Rodríguez y Jonathan Rodríguez

Ilustradores: Luis Arturo Rodríguez y Alberto García

Programadores: Alberto García, Jonathan Rodríguez y Luis Arturo Rodríguez

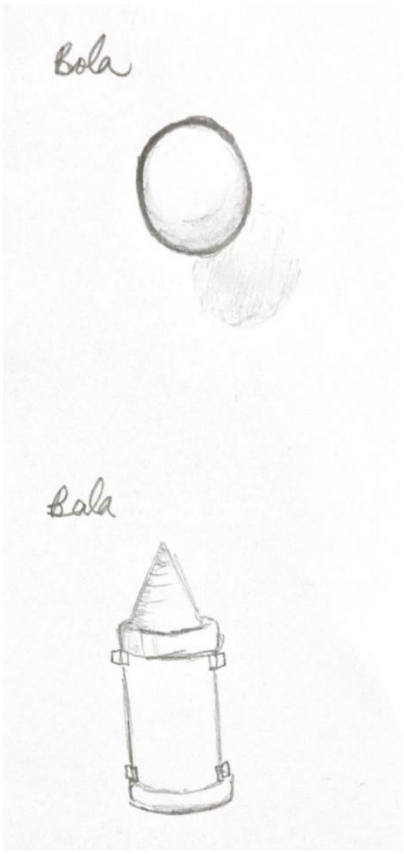
2.8. HISTORIA

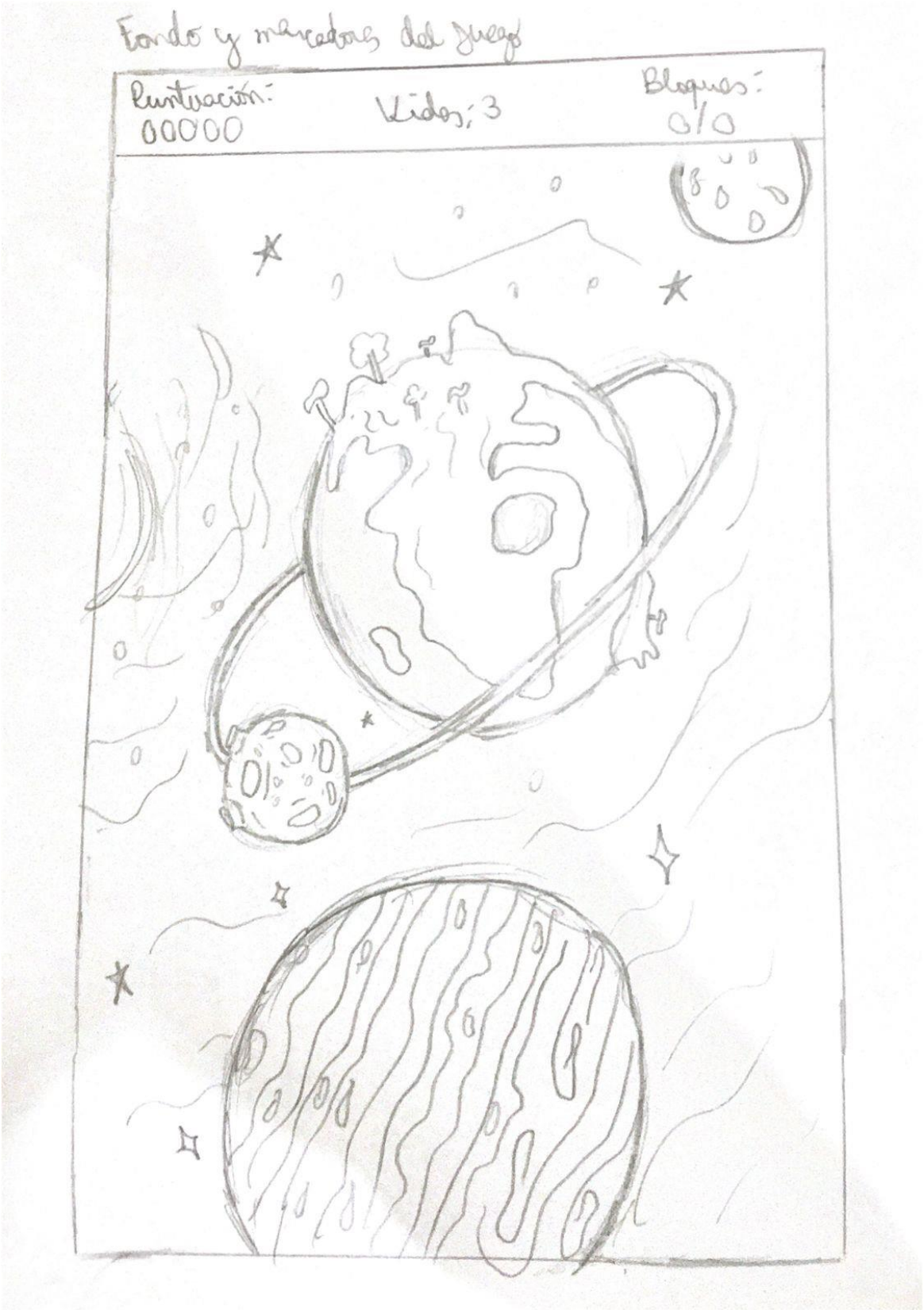
Este juego es una versión moderna de Breakout, un juego de arcade lanzado por Atari en 1976. Tiene como objetivo romper bloques con el uso de una raqueta y una bola. Para una mayor estética se le han agregado detalles que hacen alegoria al espacio exterior y se han agregado algunos enemigos para aumentar un poco la dificultad de este clásico.

2.9. GUION

Ya que el juego no cuenta con el uso de diálogos entre los personajes para el desarrollo del mismo este no necesitará la creación de un guion para su correcto funcionamiento.

2.10. STORYBOARD




Fondo y marcadores del juego
Puntuación:
00000
Vidas: 3
Bloques:
0/0

2.11. PERSONAJES

En este caso, el personaje sería la raqueta, ya que con esta es que el jugador puede interactuar y desempeñarse en el juego. La raqueta dispondrá de una pelota, la cual podrá lanzar y hacer rebotar a lo largo del juego.

Raqueta

2.12. NIVELES

Nuestros niveles están compuestos por diferentes tipos de bloques y colocados de diferentes formas. El color amarillo representa el cubo que más fácil se rompe, rompiéndose de un solo golpe, luego le sigue el naranja, teniendo que dar dos golpes para romperlos y por último tenemos los bloques de color rojo que necesitan de tres golpes para ser rotos, la dificultad de los niveles ira variando mediante la combinación de estos tres bloques.

2.13. MECÁNICA DE JUEGO

El jugador será el encargado de controlar una raqueta (nave espacial) para que a través de una bola (arma principal) que debe mantener en juego sin que llegue a caer al vacío pueda romper una cierta cantidad de bloques (planetas alienígenas), al ir destruyendo los bloques estos pueden dropear ciertos objetos que le pueden facilitar la tarea de romper los bloques. De vez en cuando podrán aparecer ciertas naves enemigas que intentaran distraer al jugador y hacer que este pierda, por lo que tendrá que encargarse de esquivar sus disparos y mantener la bola en juego. Las vidas que tendrá el jugador al iniciar una partida serán 3 y se verán afectadas cada vez que el jugador deje caer la bola al vacío o sea alcanzado por alguno de los disparos enemigos.

CAPÍTULO III: DESARROLLO

3.1. CAPTURAS DE LA APLICACIÓN

3.1.1. SCRIPTS

Ball.cs

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

public class Ball : MonoBehaviour
{
    public AudioClip paddle, rebound;

    private AudioSource AudioSource;

    private SpriteRenderer sr;

    public bool isLightningBall;

    public ParticleSystem lightningBallEffect;

    public float lightningBallDuration = 10;

    public static event Action<Ball> OnBallDeath;
    public static event Action<Ball> OnLightningBallEnable;
    public static event Action<Ball> OnLightningBallDisable;

    private void Awake()
    {
        this.sr = GetComponentInChildren<SpriteRenderer>();
        this.AudioSource = GetComponent<AudioSource>();
    }

    public void Die()
    {
        OnBallDeath?.Invoke(this);
        Destroy(gameObject, 1);
    }

    public void StartLightningBall()
    {
        if (!this.isLightningBall)
        {
            this.isLightningBall = true;
            this.sr.enabled = false;
            lightningBallEffect.gameObject.SetActive(true);
            StartCoroutine(StopLightningBallAfterTime(this.lightningBallDuration));

            OnLightningBallEnable?.Invoke(this);
        }
    }

    private IEnumerator StopLightningBallAfterTime(float seconds)
    {
        yield return new WaitForSeconds(seconds);

        StopLightningBall();
    }

    private void StopLightningBall()
    {
        if (this.isLightningBall)
        {
            this.isLightningBall = false;
            this.sr.enabled = true;
            lightningBallEffect.gameObject.SetActive(false);

            OnLightningBallDisable?.Invoke(this);
        }
    }

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Paddle")
            AudioSource.clip = paddle;
        else
            if(collision.gameObject.name == "LeftWall" ||
               collision.gameObject.name == "RightWall" ||
               collision.gameObject.name == "TopWall")
                AudioSource.clip = rebound;

        AudioSource.Play();
    }
}
```

BallsManager.cs

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

public class BallsManager : MonoBehaviour
{
    #region Singleton

    private static BallsManager _instance;

    public static BallsManager Instance => _instance;

    private void Awake()
    {
        if (_instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            _instance = this;
        }
    }

    #endregion

    [SerializeField]
    private Ball ballPrefab;

    private Ball initialBall;

    private Rigidbody2D initialBallRb;

    public float initialBallSpeed = 250;

    public List<Ball> Balls { get; set; }

    private void Start()
    {
        InitBall();
    }

    private void Update()
    {
        if (!GameManager.Instance.IsGameStarted)
        {
            // Align ball position to the paddle position
            Vector3 paddlePosition = Paddle.Instance.gameObject.transform.position;
            Vector3 ballPosition = new Vector3(paddlePosition.x, paddlePosition.y + .27f, 0);
            initialBall.transform.position = ballPosition;

            if (Input.GetMouseButtonDown(0))
            {
                initialBallRb.isKinematic = false;
                initialBallRb.AddForce(new Vector2(0, initialBallSpeed));
                GameManager.Instance.IsGameStarted = true;
            }
        }
    }

    public void SpawnBalls(Vector3 position, int count, bool isLightningBall)
    {
        for (int i = 0; i < count; i++)
        {
            Ball spawnedBall = Instantiate(ballPrefab, position, Quaternion.identity) as Ball;
            if (isLightningBall)
            {
                spawnedBall.StartLightningBall();
            }

            Rigidbody2D spawnedBallRb = spawnedBall.GetComponent<Rigidbody2D>();
            spawnedBallRb.isKinematic = false;
            spawnedBallRb.AddForce(new Vector2(0, initialBallSpeed));
            this.Balls.Add(spawnedBall);
        }
    }

    public void ResetBalls()
    {
        foreach (var ball in this.Balls.ToList())
        {
            Destroy(ball.gameObject);
        }

        InitBall();
    }

    private void InitBall()
    {
        Vector3 paddlePosition = Paddle.Instance.gameObject.transform.position;
        Vector3 startingPosition = new Vector3(paddlePosition.x, paddlePosition.y + .27f, 0);
        initialBall = Instantiate(ballPrefab, startingPosition, Quaternion.identity);
        initialBallRb = initialBall.GetComponent<Rigidbody2D>();

        this.Balls = new List<Ball>
        {
            initialBall
        };
    }
}
```

Brick.cs

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

public class Brick : MonoBehaviour
{
    public AudioClip hit, destroy, ExtendOrShrink, LightningBall, MultiBall, ShootingPaddle;

    private AudioSource AudioSource;

    private SpriteRenderer sr;
    private BoxCollider2D boxCollider;

    public int Hitpoints = 1;
    public ParticleSystem DestroyEffect;

    public static event Action<Brick> OnBrickDestruction;

    private void Awake()
    {
        this.sr = this.GetComponent<SpriteRenderer>();
        this.AudioSource = GetComponent<AudioSource>();
        this.boxCollider = this.GetComponent<BoxCollider2D>();
        Ball.OnLightningBallEnable += OnLightningBallEnable;
        Ball.OnLightningBallDisable += OnLightningBallDisable;
    }

    private void OnLightningBallDisable(Ball obj)
    {
        if (this != null)
        {
            this.boxCollider.isTrigger = false;
        }
    }

    private void OnLightningBallEnable(Ball obj)
    {
        if (this != null)
        {
            this.boxCollider.isTrigger = true;
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        bool instantKill = false;

        if (this.Hitpoints > 1)
        {
            AudioSource.clip = hit;
        }
        else
            if (this.Hitpoints == 1)
            {
                AudioSource.clip = destroy;
            }

        AudioSource.Play();

        if (collision.collider.tag == "Ball")
        {
            Ball ball = collision.gameObject.GetComponent<Ball>();
            instantKill = ball.isLightningBall;
        }

        if (collision.collider.tag == "Ball" || collision.collider.tag == "Projectile")
        {
            this.TakeDamage(instantKill);
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        bool instantKill = false;

        if (collision.tag == "Ball")
        {
            Ball ball = collision.gameObject.GetComponent<Ball>();
            instantKill = ball.isLightningBall;
        }

        if (collision.tag == "Ball" || collision.tag == "Projectile")
        {
            this.TakeDamage(instantKill);
        }
    }

    private void TakeDamage(bool instantKill)
    {
        this.Hitpoints--;

        if (this.Hitpoints <= 0 || instantKill)
        {
            BricksManager.Instance.RemainingBricks.Remove(this);
            OnBrickDestruction?.Invoke(this);
            OnBrickDestroy();
            SpawnDestroyEffect();
            Destroy(this.gameObject);
        }
        else
        {
            this.sr.sprite = BricksManager.Instance.Sprites[this.Hitpoints - 1];
        }
    }

    private void OnBrickDestroy()
    {
        float buffSpawnChance = UnityEngine.Random.Range(0, 100f);
        float debuffSpawnChance = UnityEngine.Random.Range(0, 100f);
        bool alreadySpawned = false;

        if (buffSpawnChance <= CollectablesManager.Instance.BuffChance)
        {
            alreadySpawned = true;
            Collectable newBuff = this.SpawnCollectable(true);
        }

        if (debuffSpawnChance <= CollectablesManager.Instance.DebuffChance && !alreadySpawned)
        {
            Collectable newDebuff = this.SpawnCollectable(false);
        }
    }

    private Collectable SpawnCollectable(bool isBuff)
    {
        List<Collectable> collection;

        if (isBuff)
        {
            collection = CollectablesManager.Instance.AvailableBuffs;
        }
        else
        {
            collection = CollectablesManager.Instance.AvailableDebuffs;
        }

        int buffIndex = UnityEngine.Random.Range(0, collection.Count);
        Collectable prefab = collection[buffIndex];
        Collectable newCollectable = Instantiate(prefab, this.transform.position, Quaternion.identity) as
        Collectable;

        return newCollectable;
    }

    private void SpawnDestroyEffect()
    {
        Vector3 brickPos = gameObject.transform.position;
        Vector3 spawnPosition = new Vector3(brickPos.x, brickPos.y, brickPos.z - 0.2f);
        GameObject effect = Instantiate(DestroyEffect.gameObject, spawnPosition, Quaternion.identity);

        MainModule mm = effect.GetComponent<ParticleSystem>().main;
        mm.startColor = this.sr.color;
        Destroy(effect, DestroyEffect.main.startLifetime.constant);
    }

    public void Init(Transform containerTransform, Sprite sprite, Color color, int hitpoints)
    {
        this.transform.SetParent(containerTransform);
        this.sr.sprite = sprite;
        this.sr.color = color;
        this.Hitpoints = hitpoints;
    }

    private void OnDisable()
    {
        Ball.OnLightningBallEnable -= OnLightningBallEnable;
        Ball.OnLightningBallDisable -= OnLightningBallDisable;
    }
}
```

BricksManager.cs

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

public class BricksManager : MonoBehaviour
{
    #region Singleton

    private static BricksManager _instance;

    public static BricksManager Instance => _instance;

    public static event Action OnLevelLoaded;

    private void Awake()
    {
        if (_instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            _instance = this;
        }
    }

    #endregion

    private int maxRows = 17;
    private int maxCols = 13;
    private GameObject bricksContainer;
    private float initialBrickSpawnPositionX = -1.95f;
    private float initialBrickSpawnPositionY = 3.325f;
    private float shiftAmount = 0.365f;

    public Brick brickPrefab;

    public Sprite[] Sprites;

    public Sprite[] BGSprites;

    public Color[] BrickColors, BrickColors2, BrickColors3;

    public List<Brick> RemainingBricks { get; set; }

    public List<int[,]> LevelsData { get; set; }

    public int InitialBricksCount { get; set; }

    public int CurrentLevel;

    private SpriteRenderer BGSpriteRenderer;

    private void Start()
    {
        this.bricksContainer = new GameObject("BricksContainer");
        BGSpriteRenderer = GameObject.Find("Background").GetComponent<SpriteRenderer>();

        if(this.CurrentLevel < 5)
            BGSpriteRenderer.sprite = BGSprites[0];
        else
            if(this.CurrentLevel < 10)
                BGSpriteRenderer.sprite = BGSprites[1];
            else
                BGSpriteRenderer.sprite = BGSprites[2];

        this.LevelsData = this.LoadLevelsData();
        this.GenerateBricks();
    }

    public void LoadNextLevel()
    {
        this.CurrentLevel++;

        if (this.CurrentLevel >= this.LevelsData.Count)
        {
            GameManager.Instance.ShowVictoryScreen();
        }
        else
        {
            if(this.CurrentLevel < 5)
                BGSpriteRenderer.sprite = BGSprites[0];
            else
                if(this.CurrentLevel < 10)
                    BGSpriteRenderer.sprite = BGSprites[1];
                else
                    BGSpriteRenderer.sprite = BGSprites[2];

            this.LoadLevel(this.CurrentLevel);
        }
    }

    public void LoadLevel(int level)
    {
        this.CurrentLevel = level;
        this.ClearRemainingBricks();
        this.GenerateBricks();
    }

    private void ClearRemainingBricks()
    {
        foreach (Brick brick in this.RemainingBricks.ToList())
        {
            Destroy(brick.gameObject);
        }
    }

    private void GenerateBricks()
    {
        this.RemainingBricks = new List<Brick>();
        int[,] currentLevelData = this.LevelsData[this.CurrentLevel];
        float currentSpawnX = initialBrickSpawnPositionX;
        float currentSpawnY = initialBrickSpawnPositionY;
        float zShift = 0;

        for (int row = 0; row < this.maxRows; row++)
        {
            for (int col = 0; col < this.maxCols; col++)
            {
                int brickType = currentLevelData[row, col];

                if (brickType > 0)
                {
                    Brick newBrick = Instantiate(brickPrefab, new Vector3(currentSpawnX, currentSpawnY, 0.0f -
                                                 zShift), Quaternion.identity) as Brick;

                    if(this.CurrentLevel < 5)
                        newBrick.Init(bricksContainer.transform, this.Sprites[brickType - 1],
                                      this.BrickColors[brickType], brickType);
                    else
                        if(this.CurrentLevel < 10)
                            newBrick.Init(bricksContainer.transform, this.Sprites[brickType - 1],
                                          this.BrickColors2[brickType], brickType);
                        else
                            newBrick.Init(bricksContainer.transform, this.Sprites[brickType - 1],
                                          this.BrickColors3[brickType], brickType);

                    this.RemainingBricks.Add(newBrick);
                    zShift += 0.00017f;
                }

                currentSpawnX += shiftAmount;
                if (col + 1 == this.maxCols)
                {
                    currentSpawnX = initialBrickSpawnPositionX;
                }
            }

            currentSpawnY -= shiftAmount;
        }

        this.InitialBricksCount = this.RemainingBricks.Count;
        OnLevelLoaded?.Invoke();
    }

    private List<int[,]> LoadLevelsData()
    {
        TextAsset text = Resources.Load("levels") as TextAsset;

        string[] rows = text.text.Split(new string[] { "|" }, StringSplitOptions.RemoveEmptyEntries);

        List<int[,]> levelsData = new List<int[,]>();
        int[,] currentLevel = new int[maxRows, maxCols];
        int currentRow = 0;

        for (int row = 0; row < rows.Length; row++)
        {
            string line = rows[row];

            if (line.IndexOf("--") == -1)
            {
                string[] bricks = line.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
                for (int col = 0; col < bricks.Length; col++)
                {
                    currentLevel[currentRow, col] = int.Parse(bricks[col]);
                }

                currentRow++;
            }
            else
            {
                // end of current level
                // add the matrix to the list and continue the loop
                currentRow = 0;
                levelsData.Add(currentLevel);
                currentLevel = new int[maxRows, maxCols];
            }
        }

        return levelsData;
    }
}
```

CamManager.cs

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

public class CamManager : MonoBehaviour
{
    void Start()
    {
        Screen.SetResolution(540, 700, false);
    }
}
```

DeathWall.cs

```
using UnityEngine;

public class DeathWall : MonoBehaviour
{
    private AudioSource AudioSource;
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Ball")
        {
            this.AudioSource = GetComponent<AudioSource>();
            AudioSource.Play();
            Ball ball = collision.GetComponent<Ball>();
            BallsManager.Instance.Balls.Remove(ball);
            ball.Die();
        }
    }
}
```

GameManager.cs

```
using System;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    #region Singleton

    private static GameManager _instance;

    public static GameManager Instance => _instance;

    private void Awake()
    {
        if (_instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            _instance = this;
        }
    }

    #endregion

    public AudioClip Intro, Final, Button, Victory;
    private AudioSource AudioSource;
    public GameObject gameOverScreen;
    public GameObject victoryScreen;
    public GameObject menuScreen;
    public int AvailibleLives = 3;
    public int Lives { get; set; }
    public bool IsGameStarted { get; set; }
    public static event Action<int> OnLiveLost;

    private void Start()
    {
        this.Lives = this.AvailibleLives;
        Ball.OnBallDeath += OnBallDeath;
        Brick.OnBrickDestruction += OnBrickDestruction;
        AudioSource = GetComponent<AudioSource>();
        AudioSource.loop = false;
        AudioSource.clip = Button;
        AudioSource.Play();
        gameOverScreen.SetActive(false);
        victoryScreen.SetActive(false);
        menuScreen.SetActive(true);
        AudioSource.loop = true;
        AudioSource.clip = Intro;
        AudioSource.Play();
    }

    private void OnBrickDestruction(Brick obj)
    {
        if (BricksManager.Instance.RemainingBricks.Count <= 0)
        {
            BallsManager.Instance.ResetBalls();
            GameManager.Instance.IsGameStarted = false;
            BricksManager.Instance.LoadNextLevel();
        }
    }

    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void Play()
    {
        AudioSource.loop = false;
        AudioSource.clip = Button;
        AudioSource.Play();
        gameOverScreen.SetActive(false);
        victoryScreen.SetActive(false);
        menuScreen.SetActive(false);
    }

    private void OnBallDeath(Ball obj)
    {
        if (BallsManager.Instance.Balls.Count <= 0)
        {
            this.Lives--;

            if (this.Lives < 1)
            {
                menuScreen.SetActive(false);
                victoryScreen.SetActive(false);
                gameOverScreen.SetActive(true);
                AudioSource.loop = false;
                AudioSource.clip = Final;
                AudioSource.Play();
            }
            else
            {
                OnLiveLost?.Invoke(this.Lives);
                BallsManager.Instance.ResetBalls();
                IsGameStarted = false;
                BricksManager.Instance.LoadLevel(BricksManager.Instance.CurrentLevel);
            }
        }
    }

    internal void ShowVictoryScreen()
    {
        gameOverScreen.SetActive(false);
        menuScreen.SetActive(false);
        victoryScreen.SetActive(true);
        AudioSource.loop = false;
        AudioSource.clip = Victory;
        AudioSource.Play();
    }

    private void OnDisable()
    {
        Ball.OnBallDeath -= OnBallDeath;
    }
}
```

Paddle.cs

UIManager.cs

```
using System;
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    public Text Target;
    public Text ScoreText;
    public Text LivesText;
    public int Score { get; set; }

    private void Awake()
    {
        Brick.OnBrickDestruction += OnBrickDestruction;
        BricksManager.OnLevelLoaded += OnLevelLoaded;
        GameManager.OnLiveLost += OnLiveLost;
    }

    private void Start()
    {
        OnLiveLost(GameManager.Instance.AvailibleLives);
    }

    private void OnLiveLost(int remainingLives)
    {
        LivesText.text = $"VIDAS: {remainingLives}";
    }

    private void OnLevelLoaded()
    {
        UpdateRemainingBricksText();
        UpdateScoreText(0);
    }

    private void UpdateScoreText(int increment)
    {
        this.Score += increment;
        string scoreString = this.Score.ToString().PadLeft(5, '0');
        ScoreText.text = $"PUNTUACIÓN:{Environment.NewLine}{scoreString}";
    }

    private void OnBrickDestruction(Brick obj)
    {
        UpdateRemainingBricksText();
        UpdateScoreText(10);
    }

    private void UpdateRemainingBricksText()
    {
        Target.text = $"BLOQUES:{Environment.NewLine}{BricksManager.Instance.RemainingBricks.Count} /
                      {BricksManager.Instance.InitialBricksCount}";
    }

    private void OnDisable()
    {
        Brick.OnBrickDestruction -= OnBrickDestruction;
        BricksManager.OnLevelLoaded -= OnLevelLoaded;
        GameManager.OnLiveLost -= OnLiveLost;
    }
}
```

Projectile.cs

```
using UnityEngine;

public class Projectile : MonoBehaviour
{
    private AudioSource AudioSource;

    private void Start() {
        this.AudioSource = GetComponent<AudioSource>();
        AudioSource.Play();
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```

Collectables/Collectable.cs

```
using UnityEngine;

public abstract class Collectable : MonoBehaviour
{
    private AudioSource AudioSource;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Paddle")
        {
            this.AudioSource = GetComponent<AudioSource>();
            AudioSource.Play();
            this.ApplyEffect();
        }

        if (collision.tag == "Paddle" || collision.tag == "DeathWall")
        {
            Destroy(this.gameObject);
        }
    }

    protected abstract void ApplyEffect();
}
```

Collectables/CollectablesManager.cs

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

public class CollectablesManager : MonoBehaviour
{
    #region Singleton
    private static CollectablesManager _instance;

    public static CollectablesManager Instance => _instance;

    private void Awake()
    {
        if (_instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            _instance = this;
        }
    }
    #endregion

    public List<Collectable> AvailableBuffs;
    public List<Collectable> AvailableDebuffs;

    [Range(0, 100)]
    public float BuffChance;

    [Range(0, 100)]
    public float DebuffChance;
}
```

Collectables/ExtendOrShrink.cs

```
public class ExtendOrShrink : Collectable
{
    public float NewWidth = 2.5f;

    protected override void ApplyEffect()
    {
        if (Paddle.Instance != null && !Paddle.Instance.PaddleIsTransforming)
        {
            Paddle.Instance.StartWidthAnimation(NewWidth);
        }
    }
}
```

Collectables/LightningBall.cs

```
public class LightningBall : Collectable
{
    protected override void ApplyEffect()
    {
        foreach (var ball in BallsManager.Instance.Balls)
        {
            ball.StartLightningBall();
        }
    }
}
```

Collectables/MultiBall.cs

```
using System.Linq;

public class MultiBall : Collectable
{
    protected override void ApplyEffect()
    {
        foreach (Ball ball in BallsManager.Instance.Balls.ToList())
        {
            BallsManager.Instance.SpawnBalls(ball.gameObject.transform.position, 2, ball.isLightningBall);
        }
    }
}
```

Collectables/ShootingPaddle.cs

```
public class ShootingPaddle : Collectable
{
    protected override void ApplyEffect()
    {
        Paddle.Instance.StartShooting();
    }
}
```

3.1.2. SPRITES

background 1.jpg

background 2.png

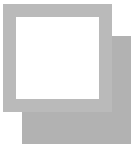
background 3.png

ball.png

paddle.png

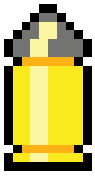
brick.png

brick2.png



brick3.png

bullet.png



expand-paddle-buff.png

lightningball-buff.png

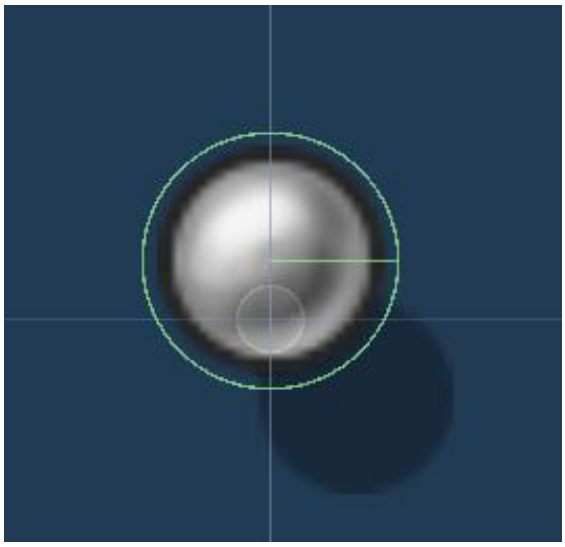
multiball-buff.png

shooting-paddle-buff.png

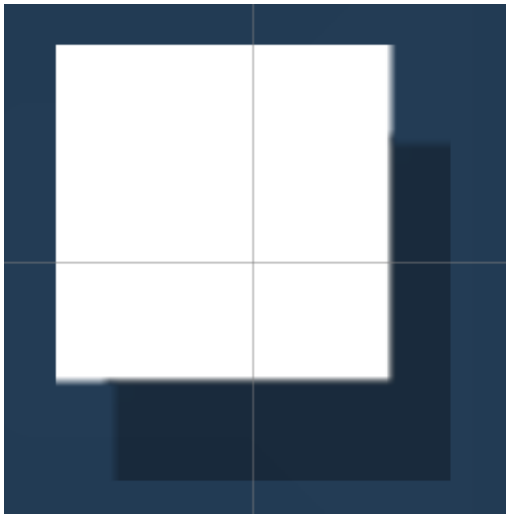
shrink-debuff.png

3.1.3. PREFABS

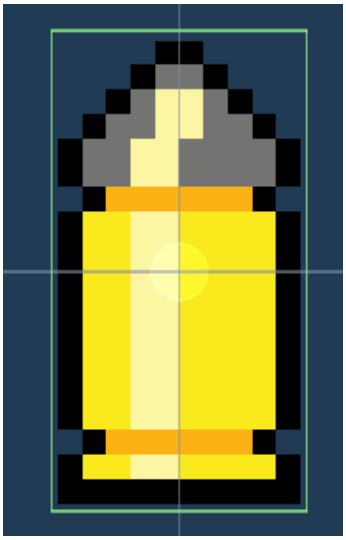
Ball.prefab



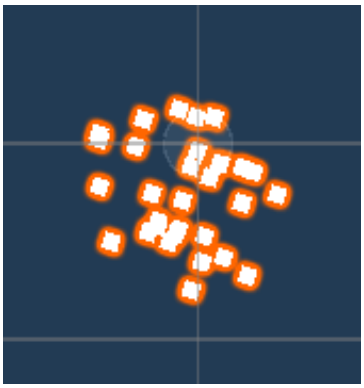
Brick.prefab



Bullet.prefab



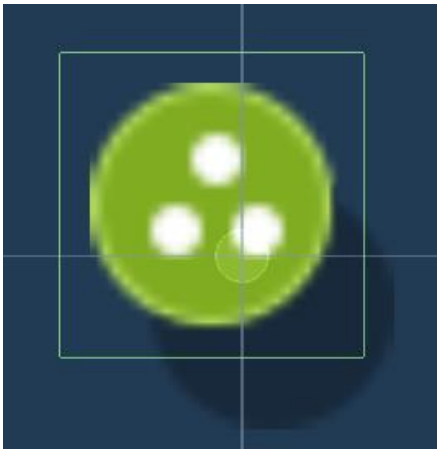
DeathEffect.prefab



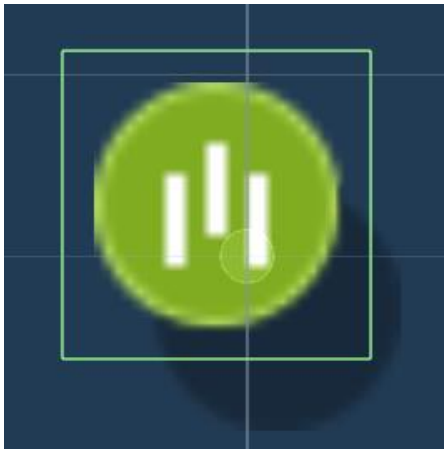
ExtendPaddle.prefab

LightningBall.prefab

MultiBall.prefab



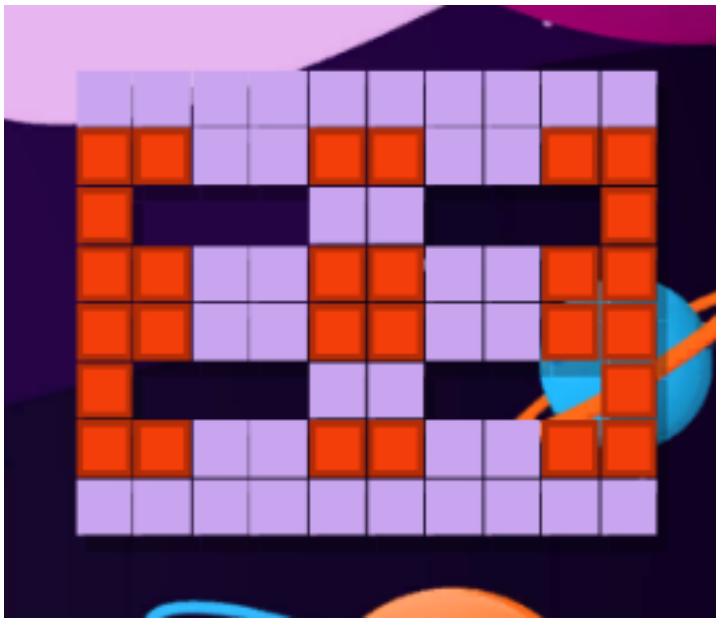
ShootingPaddle.prefab



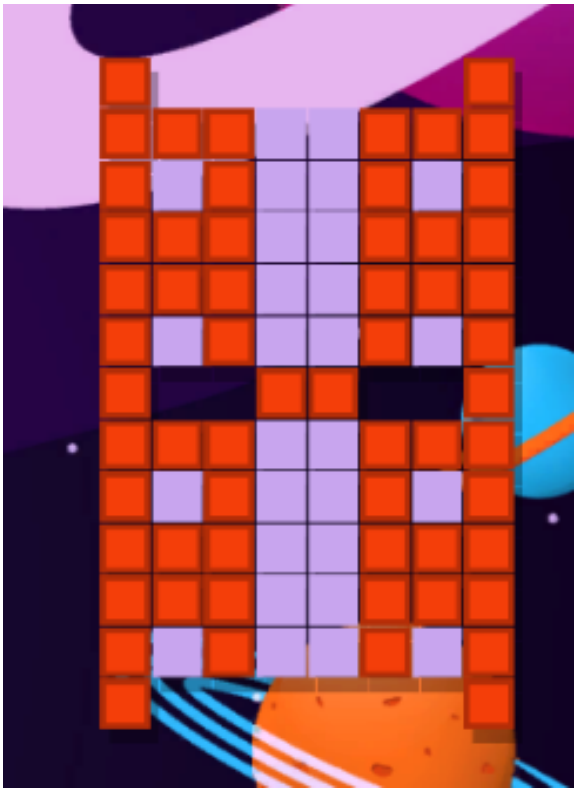
ShrinkPaddle.prefab

3.1.4. SONIDOS

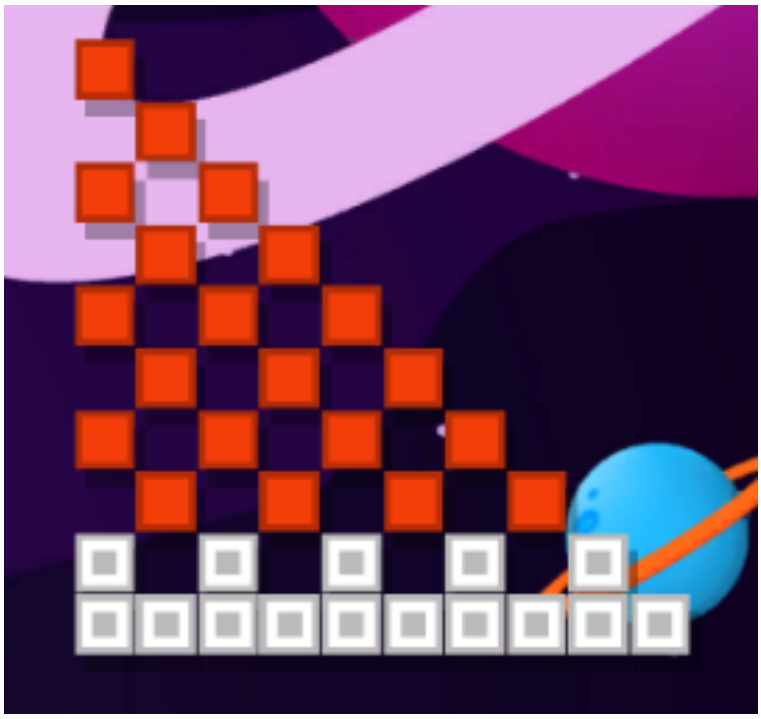
3.1.5. NIVELES



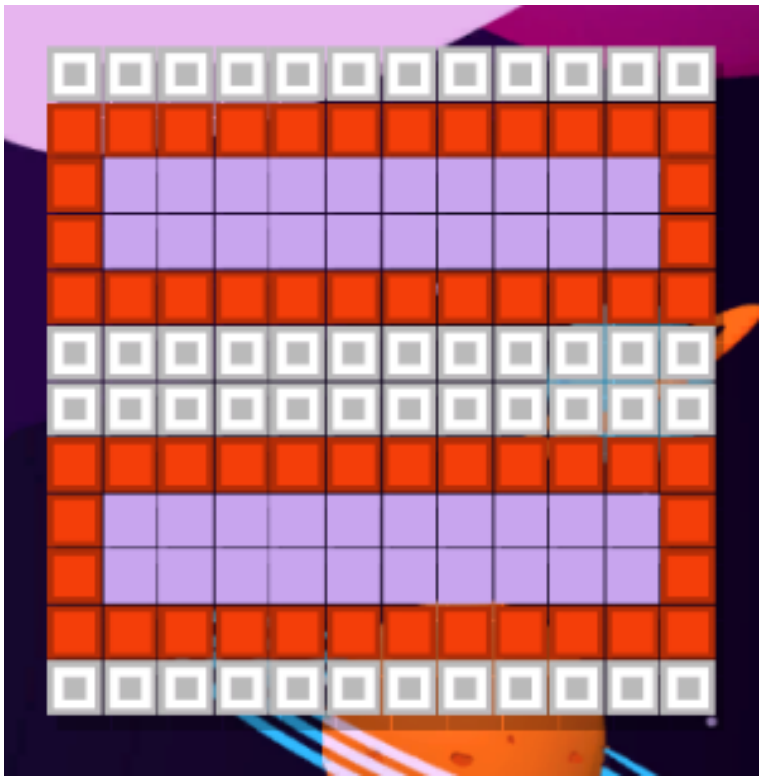
Nivel 1



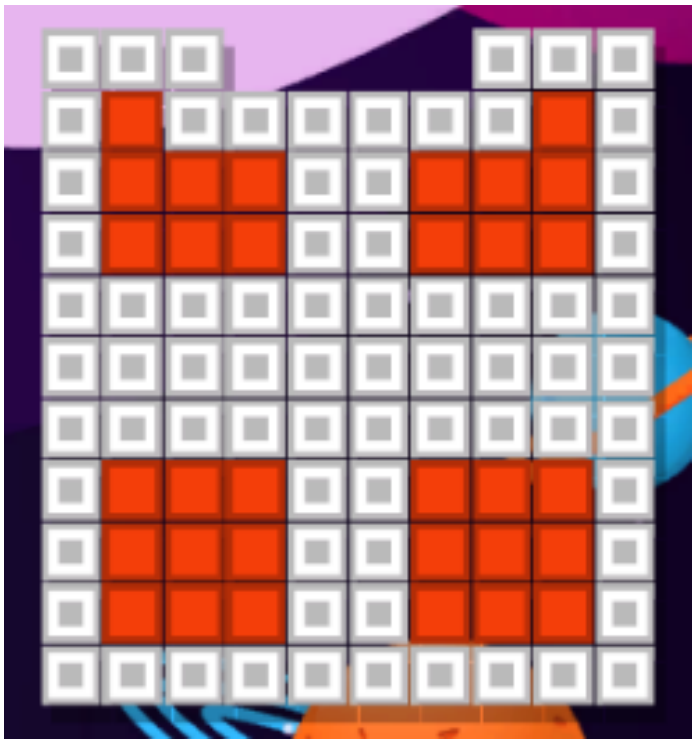
Nivel 2



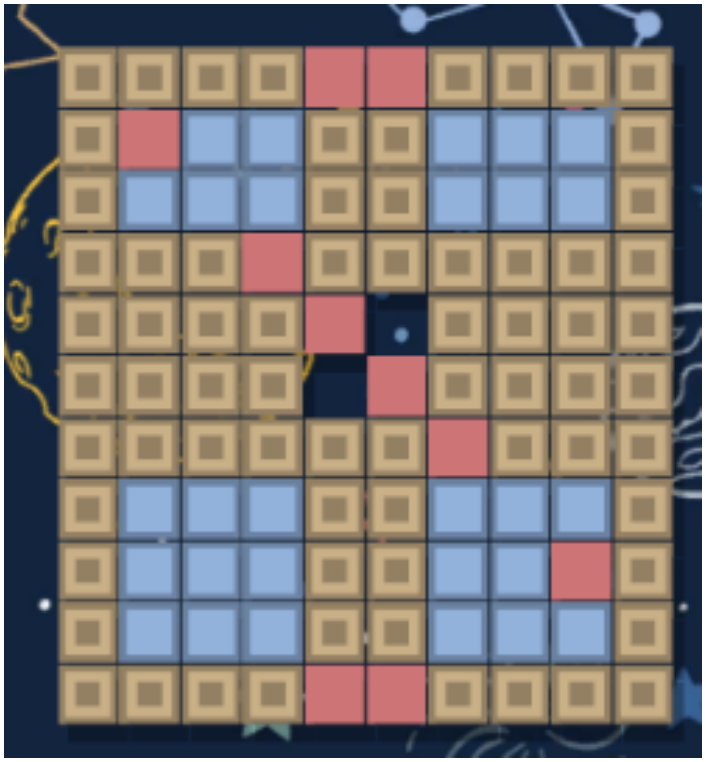
Nivel 3



Nivel 4

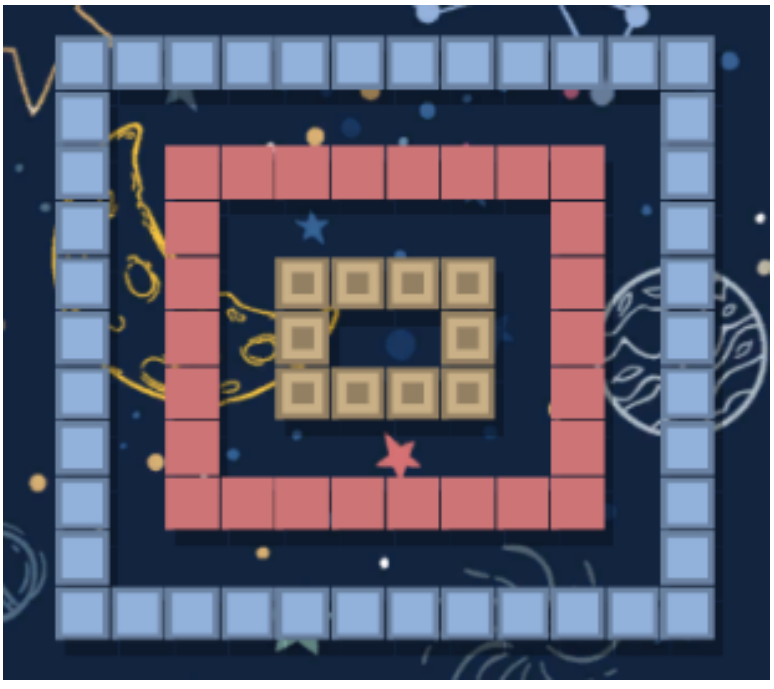


Nivel 5

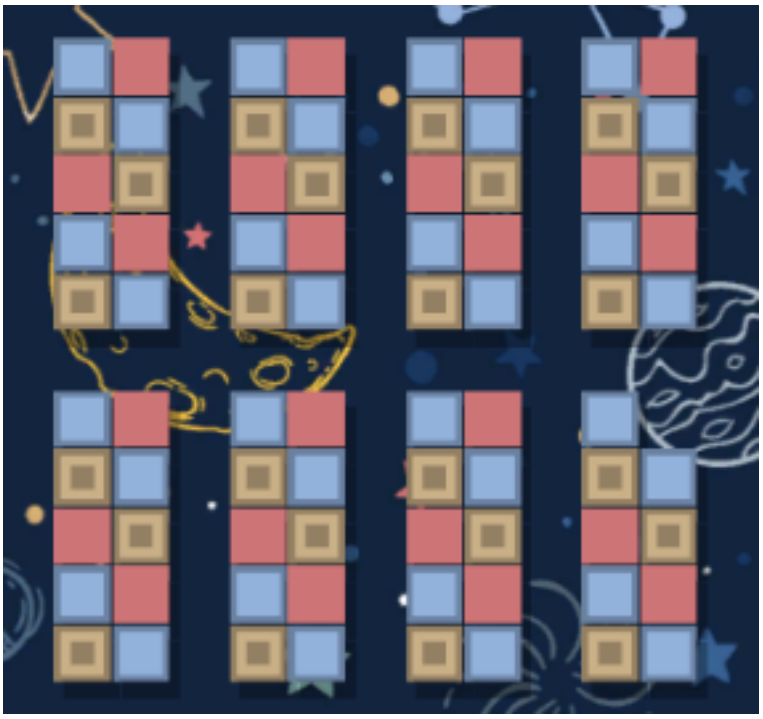


Nivel 6

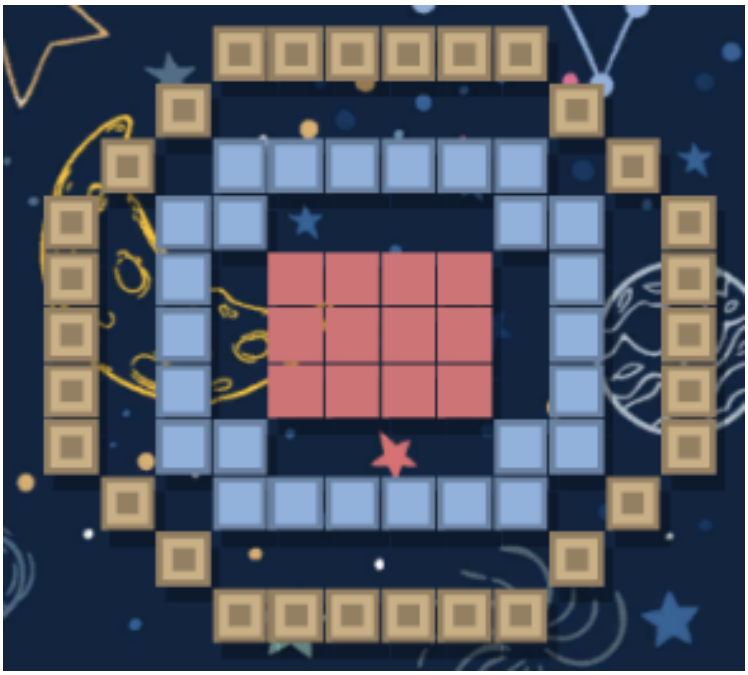
Nivel 7



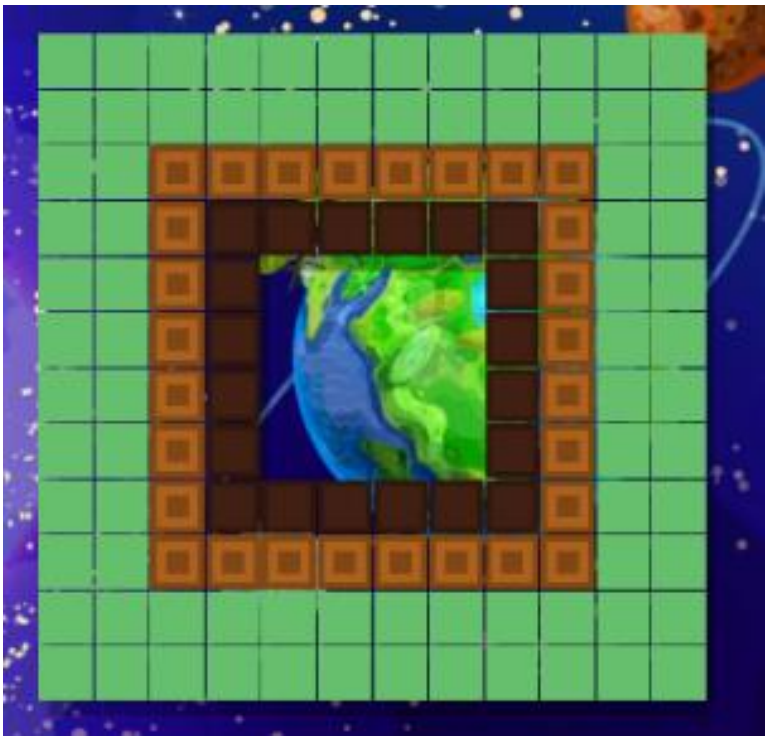
Nivel 8



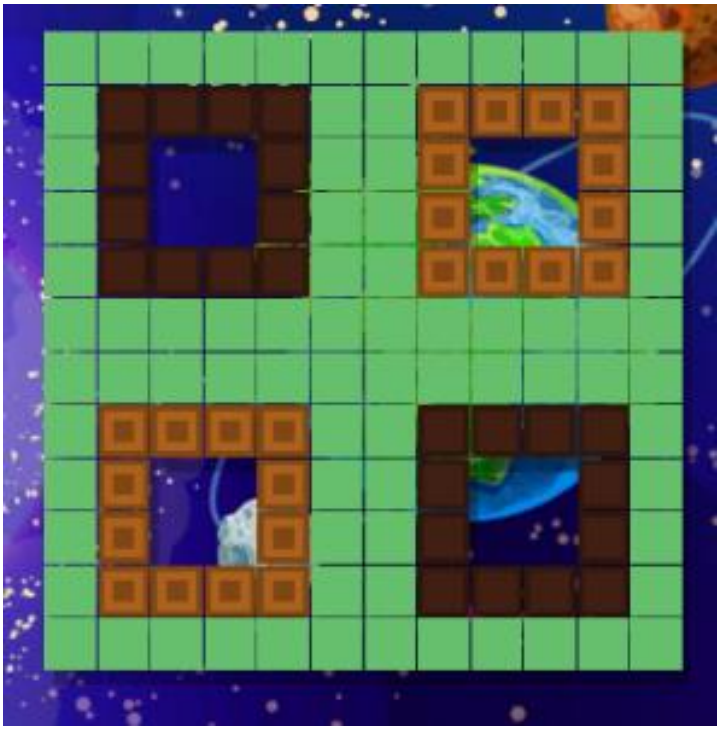
Nivel 9



Nivel 10



Nivel 11



Nivel 12

Nivel 13

Nivel 14

Nivel 15

3.2. PROTOTIPOS

Los primeros prototipos a realizar serán de tipo Lo-Fi (prototipos de baja fidelidad). Luego de ir avanzando con el diseño, comenzaremos a realizar los prototipos de tipo Hi-Fi (prototipos de alta fidelidad).

Lo-Fi

- Primer prototipo con los primeros niveles y que se pueda mover la raqueta.
- Segundo prototipo con la pelota moviéndose y rebotando en las paredes.
- Tercer prototipo con control del estado de los bloques.

Hi-Fi

- Primer prototipo con más niveles, un menú principal y control de power-ups.
- Segundo prototipo con marcadores, control de los efectos realizados por los power-

ups y control de los sonidos del juego.

3.3. PERFILES DE USUARIOS

Los públicos objetivos del videojuego son:

- Para todo público, excepto niños menores de 3 años.
- Personas aficionadas a los juegos de Arcade.
- Personas que tengan acceso a computadoras.

3.4. USABILIDAD

El juego posee una pantalla de inicio, la escena en donde se va a desarrollar el juego y una pantalla final en donde se dice si el jugador ganó o perdió.

La pantalla de inicio es sencilla y solo posee un botón para iniciar el juego, lo cual ayuda a que el jugador pueda comenzar sin ningún inconveniente.

En la escena del juego, el jugador tendrá que disponer de un mouse para poder jugar. Con este, podrá mover la raqueta y permitir que la bola se mantenga en juego. Para iniciar, solo tendrá que hacer click izquierdo en el mouse y provocará que la bola salga disparada. También cuenta con diferentes marcadores en donde puede ver cómo va el desarrollo del juego. Estos son: puntuación, vidas del jugador y los bloques restantes. Con esto se busca que el jugador se sienta más cómodo y que no esté muy confundido al momento de jugar.

Cuando finalice el juego, ya sea por perdida o por victoria, saldrá una pantalla presentando el resultado final del juego. También mostrará un botón con el cual puede volver a jugar el juego.

3.5. TEST

El desarrollo de este test es implementado de la siguiente manera: Las personas encargadas de probas nuestros juegos se harán pasar por consumidores finales y puntuarán los diferentes aspectos del juego, imaginando que ese es el producto final. Teniendo en cuenta que debe ser una crítica constructiva y con total sinceridad.

Cada usuario deberá de responder dar su opinión sobre puntos claves del juego que nos permitirá tener una mejor visión de aquello en lo que se debe de mejorar para incrementar la aceptación del juego por los usuarios finales, entre estos puntos están:

- La jugabilidad
- El control del personaje
- Guía del usuario
- La información proporcionada por el juego
- Diseño visual
- Mecánicas del juego

Para cada uno de estos puntos se deberá de dejar una puntuación y de ser posible algún comentario sobre el mismo, esto para mejorar la resolución de cualquier problema que presente el juego de manera más efectiva.

Primer Test

Individuo 1

Sexo	Masculino
Edad	45
Nivel de estudios	Ingeniero Industrial
Aficiones	Las matemáticas

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	4	“Muy nostálgico”
Control del personaje	2	“A veces la raqueta sale de la pantalla”
Guía del usuario	4	
Diseño visual	1	“Colores muy simples y pálidos”
Mecánica del juego	2	“No cuenta con muchos atractivos”

Individuo 2

Sexo	Masculino
Edad	20
Nivel de estudios	Estudiante Universitario
Aficiones	Los deportes

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	2	“Es muy simple”
Control del personaje	3	
Guía del usuario	5	“Es fácil de entender”
Diseño visual	2	“Me gustan sus colores”
Mecánica del juego	1	

Como resultado de las pruebas de nuestro primer prototipo obtuvimos un promedio de 3 en jugabilidad, 2.5 en control del personaje, 4.5, en la guía del usuario, 1.5 en el diseño visual del juego y 1.5 en la mecánica del juego. Nuestra prioridad será mejorar el diseño del juego y la mecánica del mismo para este nuevo prototipo, además de mejorar los errores mencionados en el control del personaje.

Segundo Test

Individuo 1

Sexo	Femenino
Edad	27
Nivel de estudios	Secretaria
Aficiones	La moda

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	4	“Entretenido”
Control del personaje	3	
Guía del usuario	3	
Diseño visual	2	“Algunos fondos podrían quedar mejor”
Mecánica del juego	3	

Individuo 2

Sexo	Masculino
Edad	12
Nivel de estudios	Estudiante de primaria
Aficiones	Los dulces

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	5	
Control del personaje	3	
Guía del usuario	4	
Diseño visual	3	
Mecánica del juego	4	“Los power-ups son divertidos”

Como resultado de las pruebas de nuestra versión 1.0 obtuvimos un promedio de 4.5 en jugabilidad, 3 en control del personaje, 3.5, en la guía del usuario, 2.5 en el diseño visual del juego y 3.5 en la mecánica del juego. Nuestra prioridad será mejorar aún el diseño del juego y poner en consideración la implementación de fondos para el juego, además de mejorar los errores mencionados en el control del personaje.

Tercer Test

Individuo 1

Sexo	Femenino
Edad	49
Nivel de estudios	Ama de casa
Aficiones	Las novelas

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	5	“Me recuerda a mi infancia”
Control del personaje	4	
Guía del usuario	5	
Diseño visual	5	“Muy lindos colores”
Mecánica del juego	4	

Individuo 2

Sexo	Masculino
Edad	18
Nivel de estudios	Estudiante de secundaria
Aficiones	Computación

Resultados

Tareas	Puntuación	Comentarios
Jugabilidad	5	“Entretenido”
Control del personaje	5	
Guía del usuario	4	
Diseño visual	5	“Buena ambientación”
Mecánica del juego	4	“Power-ups variados”

Como resultado de las pruebas de nuestra versión 2.0 obtuvimos un promedio de 5 en jugabilidad, 4.5 en control del personaje, 4.5, en la guía del usuario, 5 en el diseño visual del juego y 4 en la mecánica del juego. Nuestra prioridad será implementar mejoras en la jugabilidad y añadirles nuevas características para diferenciarnos de nuestros antecesores.

3.6. VERSIONES DE LA APLICACIÓN

Versión Alpha

Esta la primera versión funcional del videojuego. Solo contiene 2 niveles y el uso básico del mismo.

Versión Beta

Esta es una versión preliminar que aún es instable. Se añadió un nivel más y algunos power-ups.

Versión 1.0

Esta es la versión final del videojuego para su publicación. Se añadieron varios niveles, más power-ups y varios marcadores.

Versión 2.0

Esta versión contiene enemigos que se encargaran de dificultar más el juego.

Versión 3.0

Esta versión tiene mejoras en el control de los enemigos y niveles más complejos.

CONCLUSIONES

Pienso que este videojuego con algunos arreglos y más inversión, puede llegar a ser un producto atractivo para los consumidores de juegos de Arcade. También, con la tecnología de nuestro lado, se nos hará más fácil su distribución por el uso de las redes sociales.

En conclusión, con lo aprendido en esta materia, creo que es suficiente para poder comenzar a incursionar en el mundo de los videojuegos de forma profesional y hacer valer esos conocimientos.

Alberto de Jesús García Peña 2-17-1097

Realizar este tributo a un videojuego que formó parte de nuestros inicios en este maravilloso mundo, trae mucha nostalgia consigo, pero a pesar de esto el proceso fue muy entretenido y didáctico. Donde pudimos poner a prueba nuestros conocimientos sobre Unity, incluso de mi parte aprender a trabajar con la plataforma de GitHub. Sin duda alguna seguiré practicando el desarrollo de videojuegos y espero que siga siendo con mis compañeros

Luis Arturo Rodriguez 1-18-0858

Este juego fue unos de los jeugos de mi infancia por lo que me alegra haber podido conocer como era el funcionamiento interno de este increíble juego, además de poder agregarle aquellos detalles extras que pusieron ese toque final en nuestro juego, además puede conocer mejor como era el desarrollo de un juego a través del uso del GitHub para trabajar con varios compañeros. Estos nos facilito el desarrollo del mismo, incluso con esta pandemia.

Jonathan Daniel Rodriguez 1-18-0622

REFERENCIAS BIBLIOGRÁFICAS

Breakout (videojuego)

https://es.wikipedia.org/wiki/Breakout_(videojuego)

DX Ball Music

https://www.youtube.com/playlist?list=PLSYNim-HnIAa4V1evRIne08RE-wKeRGOn

Link de github

https://github.com/albrto-garcia/proyecto-final-videojuegos/

Link de itch.io

https://albrto-garcia.itch.io/breakout