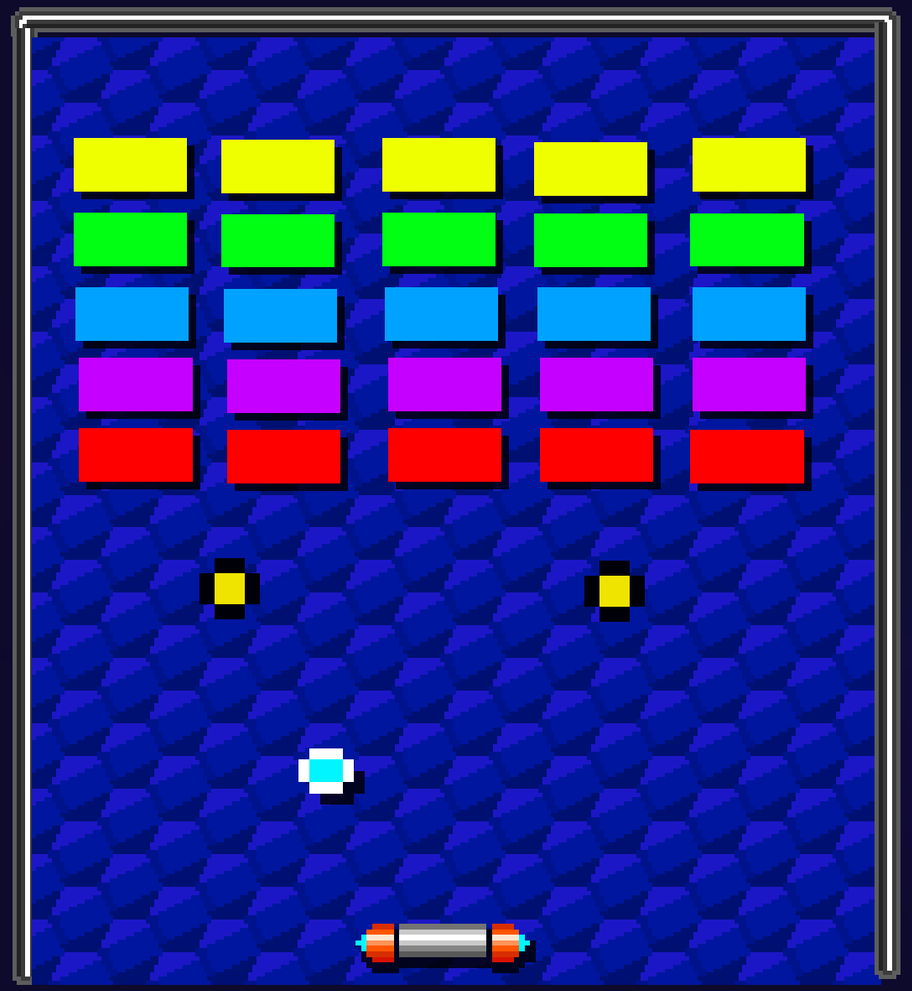
Documentación : Desarrollo del Juego Breakout

M. Rosario Palencia Vega
Centro Integrado de Formación Profesional Nº1 de Cuenca
C/Fuensanta, 3
16002 Cuenca (Cuenca)

Índice:



1. Informe Técnico:

Este documento detalla el desarrollo de una demo funcional del juego Breakout, implementado con el motor de Unity utilizando el lenguaje de programación C#. El objetivo principal del proyecto es crear un juego arcade clásico en el que el jugador controla una barra (paddle) para rebotar una pelota y destruir bloques, evitando que la pelota caiga fuera del campo de juego.

2. Decisiones de Diseño

Estilo Visual

El diseño visual del juego está inspirado en los juegos arcade de los años 80 y 90, para ello he diseñado y realizado con ayuda de Photoshop un fondo de un cielo estrellado en tonos azules, morados y rosados, con estrellas blancas. He escogido estos colores porque aportan al juego una sensación de exploración espacial, misterio, y relajación, ya que la intención es ayudar a que los jugadores se sientan más relajados mientras juegan y a la vez se sientan como si estuvieran jugando en el espacio exterior, puesto que el fondo representa una especie de galaxia lejana o un campo estelar, de alguna manera, es ayudar a el jugador a desconectar mientras hecha una partida a uno de los juegos mas clasicos de la historia.

Diseño fondo de pantalla de juego

2. Decisiones de Diseño

Elementos del Juego

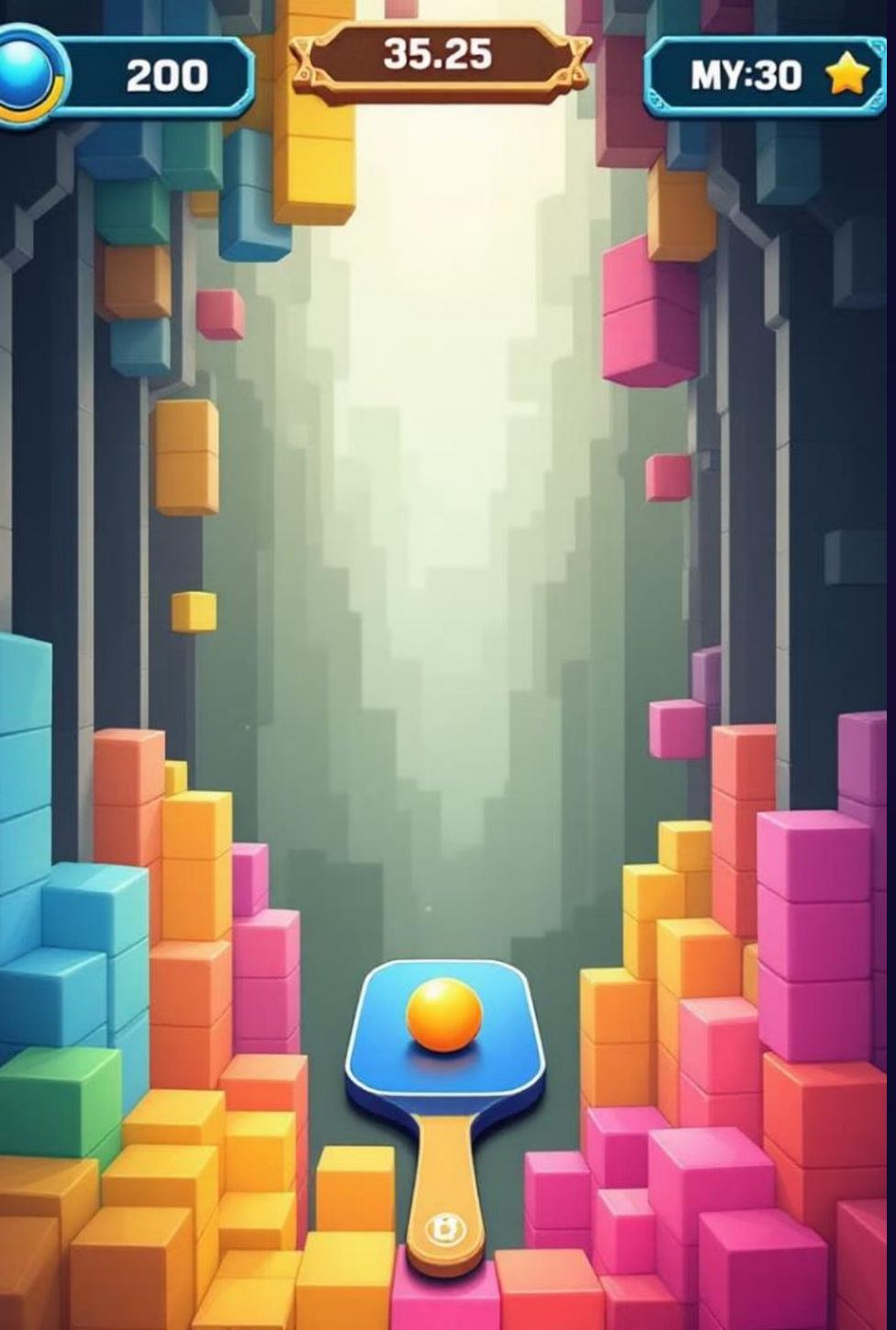
El paddle y los bloques mantienen un estilo retro, mientras que las pantallas de victoria y derrota presentan un bloque rojo para resaltar y romper la éstetica con el paddle en la mano y expresiones faciales (alegre o triste) según el resultado del juego, de forma que capta la atención sobre el fondo azul. Los mensajes de fin de juego están en inglés: "Try Again" (inténtalo de nuevo) o "Congratulations to the winner" (enhorabuena as ganado).
Por otro lado la pantalla de inicio mantiene un estilo clasico y retro, incorporando botones con la forma del clasico Brick o de Paddle.

Pantalla de inicio

Pantalla de victoria

Pantalla de derrota

Elementos zona de juego



3. Mecánicas del Juego

Control del Paddle

El jugador puede mover el paddle horizontalmente utilizando las teclas de dirección izquierda y derecha o las teclas a y d respectivamente. El movimiento está limitado a los bordes de la pantalla de juego.

Pelota

La pelota comienza sobre el paddle y se lanza con la tecla Espacio. Rebota contra el paddle, las paredes y los bloques, o bloques de recuperación de vida destruyéndolos al colisionar.

Bloques

Los bloques están dispuestos en una cuadrícula y se destruyen al ser golpeados, requieren varíos impactos para ser destruidos.

Interfaz de Usuario (UI)

Se incluye una interfaz que muestra la puntuación y las vidas restantes. También se implementan pantallas de menu principal, victoria y derrota.

4. Escenas del juego

MainMenu

Muestra la escena del menu principal.

Tiene 3 botones.

- Start game: inicia el juego
- Options: permite controlar el sonido y mutearlo (NO IMPLEMENTADO).
- Exit: permite salir del juego.

Level1Scene

Muestra la escena del juego.

En el podemos ejecutar el primer nivel del juego.

GameOverScene

Muestra la escena de perder.

WinnerScene

Muestra la escena de ganar,

5. Estructura del Código

Código de la pelota (Ball)

Lógica:

Movimiento inicial: La pelota empieza sobre el paddle y es lanzada con la tecla espacio. Esto se realiza usando la propiedad transform.parent para vincular la pelota al paddle inicialmente.

Rebote y física: Se utiliza el motor de físicas de Unity (Rigidbody2D y Collider2D) para gestionar los rebotes al colisionar con paddle, bloques o paredes. La dirección del rebote se calcula en función de la posición de impacto.

Velocidad: La velocidad de la pelota aumenta al destruir bloques, pero está limitada a un máximo (15).

Colisión con el paddle: El ángulo de rebote depende del punto de impacto relativo al paddle, mapeado en un rango de ángulos entre 45 y 135 grados.

Colisión con bloques: Al destruir un bloque, la pelota cambia su dirección en el eje Y. Si se destruyen varios bloques consecutivamente, el paddle se agranda como bonificación.

Explicación código:

Inicio (Start)

Configura la referencia al Rigidbody2D de la pelota para manejar la física.
Encuentra el objeto paddle (barra) usando el Tag "Paddle", necesario para posicionar la pelota al inicio y tras perder vidas.

Movimiento

La pelota está inicialmente vinculada al paddle como su hijo *(this.transform.parent = paddle.transform).LaunchBall*: Lanza la pelota en una dirección aleatoria cuando el jugador presiona la tecla Espacio. La velocidad inicial es constante y se calcula multiplicando un vector normalizado por la velocidad.

Colisiones con el paddle:

Calcula dónde golpea la pelota en el paddle y ajusta el ángulo de rebote.
Usa un rango de ángulos (45° a 135°) para determinar la dirección..
Con los bloques:

- ✓ Al impactar, destruye el bloque y aumenta la velocidad de la pelota (limitada a 15).
- ✓ Cuenta los bloques destruidos consecutivamente; si son más de 4, agranda el paddle.

- ✓ *OnCollisionEnter2D*: Detecta colisiones con la pala y los ladrillos. Si la pelota toca la pala, ajusta su dirección según el punto de contacto. Si toca un ladrillo, destruye el ladrillo y aumenta la velocidad de la pelota.
- ✓ *OnTriggerEnter2D:* Detecta colisiones con objetos como el prefab de recuperar vidas, y realiza la acción de recuperar una vida.

Colisiones con los bordes:

Rebota automáticamente gracias al motor de física de Unity.

- ✓ *CheckOutOfBounds*: Verifica si la pelota ha caído fuera de los límites de la pantalla y, si es así, pierde una vida y se restablece la posición de la pelota Si cae por debajo de un límite definido (lowerBoundY), se pierde una vida.

Métodos auxiliares

- ✓ ResetBallPosition: Coloca la pelota en la pala al inicio del juego o después de perder una vida..
- ✓ GrowPaddle: Agranda la pala si el jugador ha destruido 4 ladrillos consecutivos sin tocar la pala.

5. Estructura del Código

Código de los bloques (Brick)

Lógica:

Cada bloque puede requerir múltiples golpes (hitsToDestroy) para ser destruido.

Al ser destruido, el bloque:

- ✓ Incrementa el puntaje del jugador mediante GameManager.Instance.AddScore.
- ✓ Se elimina de la escena tras salir de los límites visibles.

Explicación código:

Variables:

- ✓ *hitsToDestroy*: Determina cuántos golpes necesita el bloque para ser destruido. En este caso, son 2 golpes.
- ✓ *scoreValue*: Es la cantidad de puntos que el jugador gana cuando destruye el bloque.
- ✓ *fallSpeed*: Controla la velocidad a la que el bloque caerá una vez que se destruye.
- ✓ *currentHits*: Contador de los golpes recibidos por el bloque, se incrementa cada vez que la pelota lo golpea.
- ✓ *isFalling*: indica si el bloque está cayendo después de ser destruido.

Método OnCollisionEnter2D:

Este método se llama cada vez que el bloque colisiona con otro objeto. Aquí se verifica si el objeto que colisiona es la pelota, detectada por su tag "Ball".

- ✓ Si el bloque recibe un golpe de la pelota, incrementa el contador de currentHits.
- ✓ Si el número de golpes alcanza o supera el valor de hitsToDestroy (2), se llama a StartFalling().

Método Update:

Si isFalling es true, significa que el bloque ha sido destruido y está cayendo. En cada actualización (frame), mueve el bloque hacia abajo usando la velocidad definida por fallSpeed.
Si el bloque ha caído fuera de la pantalla (cuando su posición Y es menor que el borde inferior de la cámara), se destruye.

Método StartFalling:

Marca el bloque como en estado de caída (isFalling = true).
Añade puntos al jugador usando el GameManager mediante el método AddScore(), con el valor de scoreValue de este bloque.
Finalmente, destruye el bloque. Aunque el bloque cae, la destrucción inmediata es necesaria para evitar que siga existiendo.

5. Estructura del Código

Código control del paddle (Paddle)

Lógica:

Controla el movimiento horizontal del paddle y restringe su posición dentro de los bordes de la pantalla.

El paddle se mueve horizontalmente según la entrada del usuario (Input.GetAxis ("Horizontal")) para moverse a la izquierda y derecha..

Está restringido a un rango de movimiento, calculado con Mathf.Clamp.

Explicación código:

Velocidad de la pala:

La variable *paddleSpeed* controla qué tan rápido se mueve la pala. Se declara como private pero se puede modificar desde el Inspector en Unity debido al atributo [SerializeField].

Método Update():

El método Update() es llamado cada frame y se encarga de mover la pala de acuerdo a la entrada del jugador.

Movimiento horizontal de la pala:

Input.GetAxis("Horizontal") obtiene la entrada horizontal del jugador. Este método es flexible, ya que funciona tanto con las teclas de flecha derecha e izquierda como con las teclas "A" y "D". Devuelve un valor entre -1 y 1, dependiendo de la dirección de la entrada (izquierda o derecha).

Se calcula un vector de movimiento en el eje X (new Vector3(horizontalInput, 0, 0)) y se multiplica por paddleSpeed y Time.deltaTime para hacer que el movimiento sea fluido y no dependa de la tasa de fotogramas.

Movimiento físico de la pala:

transform.Translate(movement) aplica el movimiento calculado a la posición de la pala.

Límites de la pantalla:

- ✓ *Mathf.Clamp(transform.position.x, -2.3f, 2.3f)* limita la posición X de la pala para que no se mueva fuera de los bordes de la pantalla.
  El valor -2.3f es el límite izquierdo de la pantalla, mientras que 2.3f es el límite derecho. Puedes ajustar estos valores dependiendo de las dimensiones de la pantalla y el tamaño de la pala.

Luego, se actualiza la posición de la pala utilizando *transform.position = new Vector3(xPos, transform.position.y, transform.position.z);* para mantener la pala dentro de los límites del campo de juego, mientras mantiene su posición en el eje Y y Z.

5. Estructura del Código

Gestión del juego (GameManager)

Lógica:

Maneja el puntaje, las vidas y el avance entre niveles.

Al perder vidas, reinicia la pelota sobre el paddle.

Si se destruyen todos los bloques, se carga el siguiente nivel o la pantalla de victoria.

Si se pierden todas la vidas, se carga la pantalla de Game Over.

Actualiza la interfaz gráfica (UI) para reflejar cambios en puntaje y vidas.

Explicación código:

Instancia Singleton:

InstanceSingleton es una variable estática que asegura que solo haya una instancia del GameManager en la escena. Esto es útil para que el GameManager sea accesible globalmente en cualquier parte del código.

Métodos de control de puntaje y vidas:

- ✓ *AddScore(int value)*: Este método suma los puntos a la puntuación y actualiza la UI.
- ✓ *LoseLife()*: Este método resta una vida al jugador, muestra la cantidad de vidas restantes y, si el jugador se queda sin vidas, muestra la pantalla de "Game Over".
- ✓ *AddLife()*: Incrementa el número de vidas del jugador, pero no permite que se excedan las 3 vidas.

Manejo de escenas:

- ✓ *GameOver()*: Este método carga la escena de "Game Over" cuando el jugador se queda sin vidas.
- ✓ *ReturnToMainMenu()*: Este método regresa al menú principal, destruyendo la instancia del GameManager para evitar problemas con múltiples instancias al volver a iniciar el juego.
- ✓ *NextLevel()*: Si el jugador destruye todos los ladrillos, se avanza al siguiente nivel cargando la siguiente escena en la secuencia. Si no hay más niveles, se muestra la pantalla de victoria.

Contador de ladrillos:

- ✓ *CountTotalBricks()*: Este método cuenta el número total de ladrillos en la escena para saber cuándo se han destruido todos y se pueda pasar al siguiente nivel.
- ✓ *BrickDestroyed()*: Este método se llama cada vez que un ladrillo es destruido, incrementando el puntaje y verificando si todos los ladrillos han sido destruidos.

Actualización de la UI:

- ✓ *UpdateUI()*: Actualiza la interfaz de usuario con la cantidad de vidas y el puntaje actual.

Reinicio del juego:

- ✓ *ResetGame()*: Reinicia el puntaje, las vidas y el contador de ladrillos destruidos, actualizando la UI.

5. Estructura del Código

Código Gestión de la UI (UIManager)

Lógica:

Muestra las vidas restantes con imágenes específicas.

Muestra el puntaje actualizado utilizando TextMeshProUGUI.

Cambia dinámicamente las imágenes de vidas y muestra una indicación de "sin vidas" si el jugador pierde todas.

Explicación código:

Vidas

- ✓ Muestra imágenes de vidas activas y desactiva las restantes a medida que se pierden vidas.
- ✓ Muestra una imagen especial cuando el jugador pierde todas las vidas.

Puntaje

- ✓ Actualiza el texto del puntaje en pantalla usando TextMeshProUGUI.

Explicación código:

Referencias a los elementos UI:

- ✓ *noLivesImage*: Es la imagen que se muestra cuando al jugador se le acaban todas las vidas. Se debe asignar en el Inspector.
- ✓ *livesImages:* Es un arreglo de imágenes que representan las vidas del jugador. Estas imágenes son asignadas en el Inspector (según cuántas vidas tenga el jugador).
- ✓ *scoreText*: Es un TextMeshProUGUI, que se usa para mostrar el puntaje del jugador en la pantalla.

Método UpdateLives(int lives):

Este método se encarga de actualizar las imágenes de vidas en la interfaz de usuario.

- ✓ *for (int i = 0; i < livesImages.Length; i++)*: Se recorre el arreglo de imágenes de vidas y se activa o desactiva cada imagen dependiendo de cuántas vidas tenga el jugador.
  Si el índice i es menor que el número de vidas (lives), entonces la imagen correspondiente se activa.
  Si el índice i es igual o mayor que el número de vidas, la imagen se desactiva.
- ✓ *if (noLivesImage != null)*: Si el jugador no tiene vidas (lives <= 0), la imagen que muestra la falta de vidas se activa.

Método UpdateScore(int score):

Este método se encarga de actualizar el texto que muestra el puntaje en la interfaz de usuario.
Si el objeto scoreText no es nulo, se convierte el puntaje a una cadena (score.ToString()) y se asigna al scoreText para actualizar la UI con el puntaje actual.

5. Estructura del Código

Código Menú principal y transición de escenas (MainMenuManager)

Lógica:

Implementa botones para iniciar el juego (playButton), salir (Exit) y abrir opciones (optionsButton).

Usa SceneManager para cargar las escenas correspondientes.

Botones

- ✓ Al presionar "*Jugar*", se carga la primera escena del juego (Level1Scene).
- ✓ Al presionar "*Opciones*", se muestra las opciones de sonido y muteado. (* No está implantado, solo pensado.)
- ✓ Al presionar "*Exit*", se sale del juego.

Explicación código:

Referencias a los botones:

En esta clase, se han creado cinco variables públicas, *Panel de obciones, botón jugar, botón opciones, botón salir y botón volver*, que se asignarán en el Inspector de Unity. Estas referencias apuntan a los botones y al panel de opciones de la interfaz de usuario del menú principal.

Método Start():

En el *método Start()*, asigna eventos a los botones utilizando onclick.Alistener(), cada botón ejecuta un método especifico al ser presionado; el botonjugar llama al método CargarJuego() para abrir la escena de juego, el botón opciones llama a AbrirPanelOpciones() para mostar el panel de opciones, el botón salir llama a SalirJuego() para cerrar la aplicación, y el botón volver llama a CerrarPanelOpciones() para volver al menú principal.

Métodos de manejo de botones:

- ✓ CargarJuego(): Utiliza sceneManager.LoadScene("Level1Scene") para cargar la escena principal del juego.
- ✓ AbrirPanelOpciones(): Activa el panel de opciones configurando SetActive(true) en el objeto panelOpciones.
- ✓ CerrarPanelOpciones(): Desactiva el panel de opciones configurando SetActive(false) en el objeto panelOpciones.
- ✓ SalirJuego(): Finaliza la ejecución de la aplicación con Application.Quit(). Tiene un Debug.Log("Juego cerrado") para mostrar un mensaje en la consola de Unity cuando se ejecuta en el editor.

5. Estructura del Código

Código (volverMenuPrincipal)

Lógica:

Gestiona el código para al pulsar space en las escenas

WinnerScene (escena de ganar) o GameOverScene

(escena de perder), volver al menu principal.

Permite al jugador regresar al menú principal desde las

escenas de victoria o derrota.

✓Al presionar Espacio, se carga la escena principal

(MainMenu).

Explicación código:

Variables serializadas:

GameOverScene y WinnerScene: Son las escenas de "Game Over" y "Victory", respectivamente. Se definen como string y se asignan en el Inspector de Unity. Estas escenas se usan para determinar si el jugador está en una de estas pantallas finales.

Método Update():

Este método se ejecuta en cada frame. Lo que hace es verificar si el jugador está en una de las escenas finales (ya sea "GameOver" o "Winner") y si ha presionado la tecla Espacio.

SceneManager.GetActiveScene().name == GameOverScene || SceneManager.GetActiveScene().name == WinnerScene: Comprueba si la escena actual es la de "Game Over" o "Winner".

Input.GetKeyDown(KeyCode.Space): Detecta si se ha presionado la tecla de espacio. Si se cumple la condición de estar en la pantalla de "Game Over" o "Winner" y se presiona espacio, se llama a la función ReturnToMainMenu().

Método ReturnToMainMenu():

Este método se encarga de cargar la escena principal (el menú principal) cuando el jugador presiona la tecla de espacio en la pantalla de "Game Over" o "Victory".

SceneManager.LoadScene("MainMenu"): Carga la escena del menú principal, cuyo nombre es "MainMenu".

5. Estructura del Código

Código Generación de la cuadrícula de bloques (BrickGridManager)

Lógica:

Gestiona la distribución de los Brick y otros elementos al ínicio del juego para no colocarlos manualmente.

*No está implantado, solo planteado.

Consiste en la generación dinámica de una cuadrícula de bloques ajustada al tamaño de juego.
Usa colores distintos para las filas, siguiendo un esquema predefinido.

Explicación código:

Variables:

- ✓ *brickPrefab*: Es el prefab del ladrillo que se va a instanciar para crear la cuadrícula.
- ✓ *rows*: El número de filas que tendrá la cuadrícula de ladrillos.
- ✓ *columns*: El número de columnas que tendrá la cuadrícula de ladrillos.
- ✓ *spacing*: El espaciado entre los ladrillos.
- ✓ *rowColors*: Un arreglo de colores que asigna un color diferente a cada fila de ladrillos (hasta 5 colores en este caso).

Método Start:

Este método se ejecuta al inicio y se encarga de inicializar el arreglo rowColors con los colores para las filas.
Después llama al método GenerateBrickGrid() para crear la cuadrícula de ladrillos.

Método GenerateBrickGrid:

Este método se encarga de generar la cuadrícula de ladrillos en la pantalla.
Primero busca el objeto "Background" en la escena, que sirve como referencia para calcular las dimensiones y posiciones de los ladrillos.
Si no encuentra el objeto "Background", muestra un error.
Luego, obtiene el SpriteRenderer del "Background" para obtener el tamaño del fondo y calcula las dimensiones de la cuadrícula de ladrillos.
La cuadrícula ocupará el 80% del ancho y el 50% de la altura del fondo.
Calcula el tamaño de cada ladrillo, luego genera las posiciones de los ladrillos y los instancia en la escena.
Cada ladrillo se coloca en la cuadrícula de acuerdo con su fila y columna, y se le asigna un color según la fila en la que está.

Posición de los ladrillos:

La posición de cada ladrillo se calcula para que se distribuyan uniformemente dentro de la cuadrícula, centrados dentro del área calculada en el fondo.
Los ladrillos se instancian en la escena y se posicionan en las coordenadas calculadas.

5. Estructura del Código

Código Menú de Pausa (MenuPausa)

Lógica:

Implementa un sistema de pausa en el juego que permite detener y reanudar el tiempo de juego, y también regresar al menú principal.

Pausa/Reanudar:
Con la tecla flecha arriba (UpArrow) o pulsando el botón pausa, el jugador puede alternar entre el estado de pausa y reanudación del juego.
Botones:
Los botones del menú de pausa permiten reanudar el juego o cerrarlo regresando al menú principal.

Explicación código:

Método Update():
- ✓Detecta si el jugador presiona la tecla flecha arriba (UpArrow).
- ✓Si el juego está pausado, llama a Reanudar().
- ✓Si el juego no está pausado, llama a Pausa().

Métodos principales:

1.Pausa():
- ✓ Detiene el tiempo del juego usando Time.timeScale = 0f.
- ✓ Oculta el botón de pausa (botonPausa.SetActive(false))
- ✓ Muestra el menú de pausa (menuPausaUI.SetActive(true)). Cambia la variable isPaused a true.

2.Reanudar():
- ✓ Restablece el tiempo del juego con Time.timeScale = 1f.
- ✓ Muestra el botón de pausa (botonPausa.SetActive(true)) y oculta el menú de pausa (menuPausaUI.SetActive(false)).
- ✓ Cambia la variable isPaused a false.

3.cerrar():
- ✓ Usa SceneManager.LoadScene("MainMenu") para cargar la escena del menú principal.

5. Estructura del Código

Interacciones entre scripts:

Ball y Paddle:

✓La pelota comienza como hija del paddle hasta que es lanzada.

Ball, Brick y GameManager:

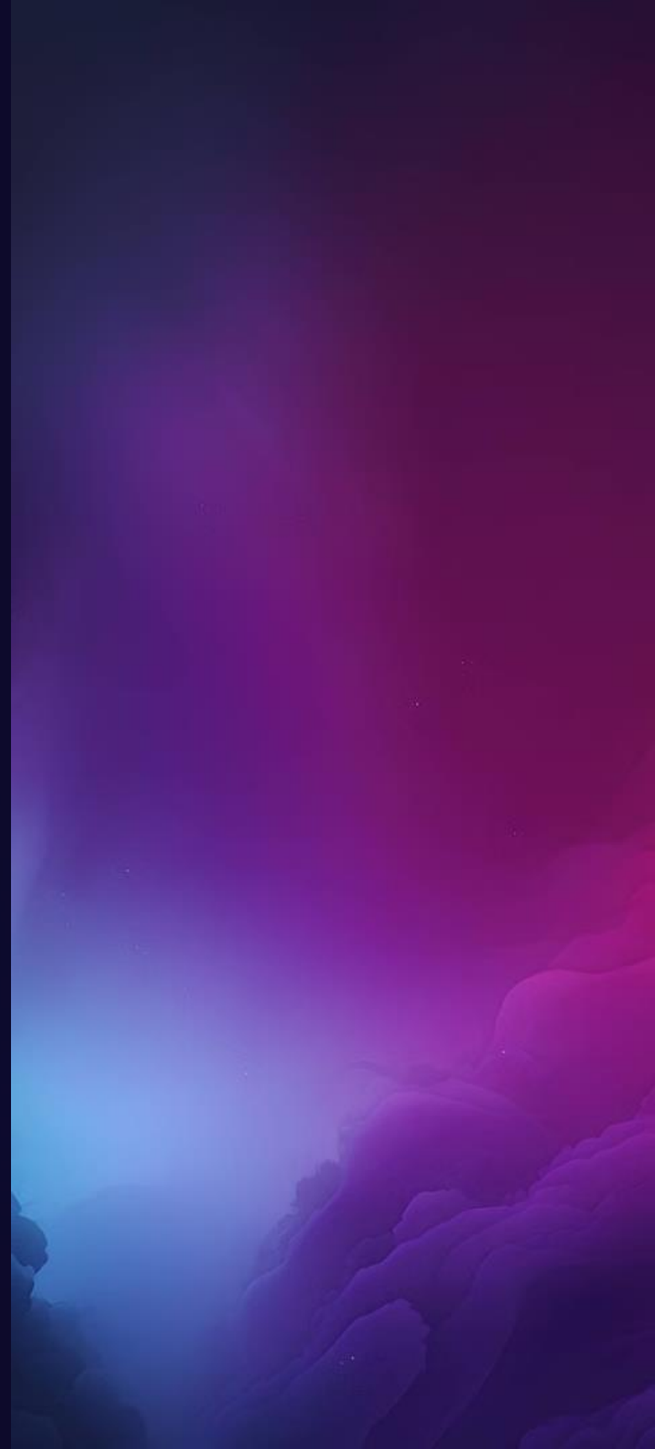
✓La pelota destruye bloques y notifica al GameManager para actualizar el puntaje.

GameManager y UIManager:

✓El GameManager informa cambios en puntaje y vidas al UIManager, que actualiza la UI.

MainMenuManager y SceneManager:

✓El menú principal controla la transición entre escenas.



6. Resolución y Optimización

El juego está optimizado para una resolución mínima de 1920x1080. Esta decisión de diseño asegura una experiencia visual consistente en la mayoría de los dispositivos modernos, manteniendo la calidad gráfica y la jugabilidad.

7. Problemas Encontrados y Soluciones

Gestión de Vidas

Problema: Al intentar gestionar las vidas desde el GameManager, las imágenes de la UI no se actualizaban correctamente.

Solución: Se delegó la actualización de las imágenes al UIManager, mientras que el GameManager mantiene la lógica de las vidas.

Posición Inicial de la Pelota

Problema: La pelota no seguía al paddle al inicio del juego, antes de ser lanzada.

Solución: Se estableció la pelota como hija del paddle y se añadió una línea de código para sincronizar su posición.

Potenciador de Pelota

Problema: Al implementar un potenciador que generaba tres pelotas simultáneamente, el jugador perdía todas sus vidas al caer las pelotas adicionales.

Solución: Se eliminó temporalmente esta funcionalidad, con la intención de implementarla correctamente en el futuro.

Scripts BrickGridManager

Problema: La cuadrícula de bloques no se ajustaba correctamente al área de juego.

Solución: Se pospuso la implementación para una futura iteración del proyecto.

8. Mejoras Futuras

1. **Implementar el BrickGridManager**

   Para generar cuadrículas de bloques dinámicas y ajustables.

2. **Añadir más tipos de bloques**

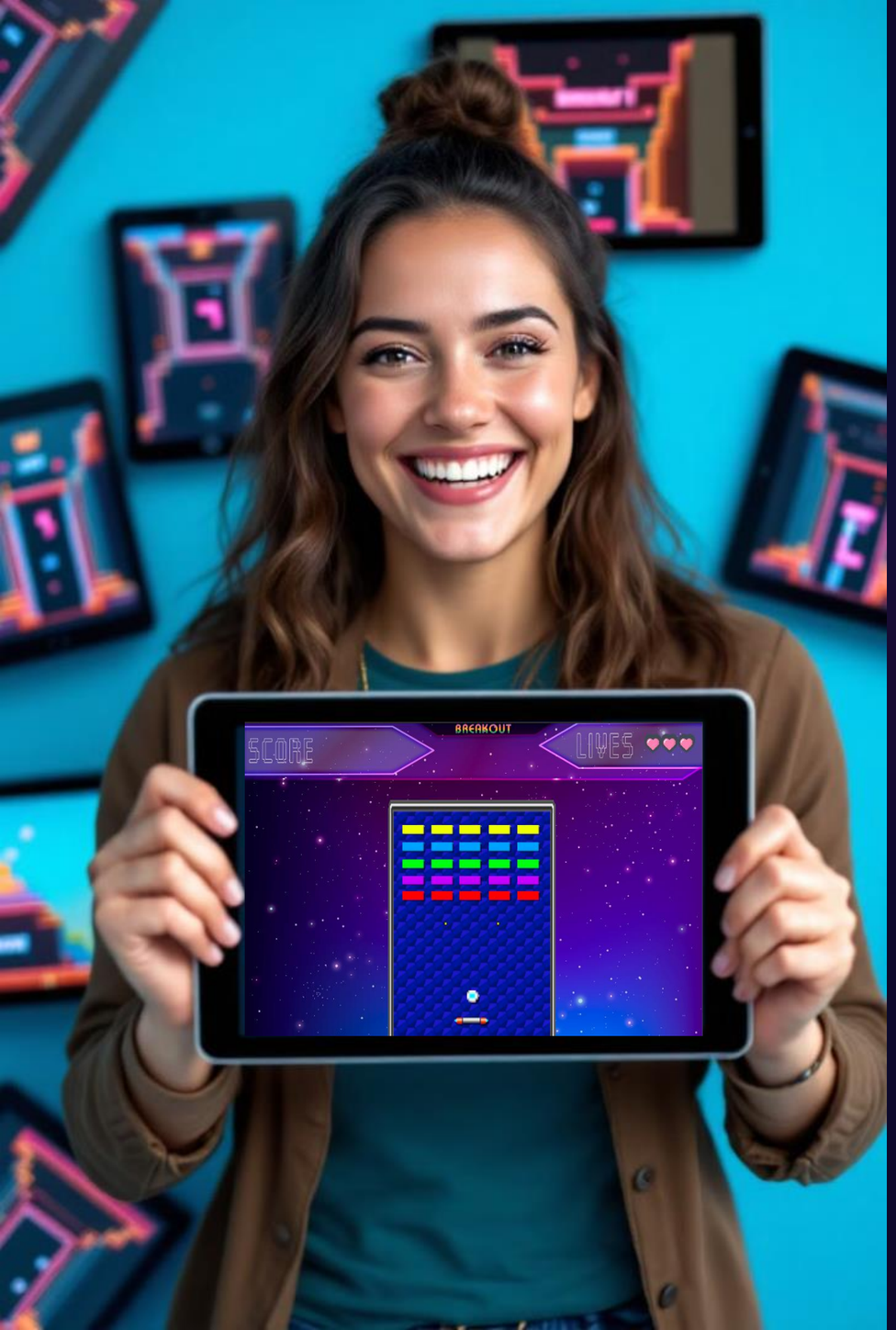
   Con diferentes comportamientos (por ejemplo, bloques indestructibles o con potenciadores de perder 1 vida etc.).

3. **Incorporar niveles adicionales**

   Con mayor dificultad.

4. **Implementar sonidos y melodías**

   Implentar sonidos de fondo en el juego y sonidos al romper Brick, ganar vida, agrandar o empequeñecer paddle.



9. Conclusión

El desarrollo del juego Breakout permite aplicar conceptos fundamentales de programación en Unity, como la gestión de físicas, colisiones y lógica de juego. A pesar de los desafíos encontrados, se está logrando crear una demo funcional que cumple con los requisitos establecidos. Este proyecto sirve como base para futuras mejoras y ampliaciones, con el objetivo de ofrecer una experiencia de juego más completa y pulida.

10. Anexos

Enlace a GitHub

Repositorio del proyecto disponible para revisión y colaboración.

Documentación Técnica: Desarrollo del Juego Breakout

Detalles adicionales sobre la implementación y estructura del código.

Demo del Juego

Versión jugable de la demo para pruebas y feedback dentro de Unity version 2022.3.16f1.