

# Bomberman 3D - Videojuego desarrollado en Unity

Presentación del Proyecto Final de Ciclo "Bomberman 3D", un videojuego desarrollado íntegramente en Unity por Miguel Ángel Cruz Cobo de Guzmán. Este proyecto, culminación de la formación en Desarrollo de Aplicaciones Multiplataforma (DAM) en el IES Las Fuentezuelas, representa más de 230 horas de dedicación y un esfuerzo significativo en programación y diseño. La documentación adjunta cuenta con 47 páginas, y el código incluye 13 scripts C# principales, demostrando un enfoque profesional y riguroso en cada etapa del desarrollo.



por Miguel Ángel Cruz Cobo de Guzmán





# Introducción y Objetivos: Bomberman 3D

- 1
- 2
- 3
- 4

## Objetivo Principal

Desarrollar un videojuego funcional que demuestre las competencias adquiridas en el ciclo DAM.

## Tecnologías Clave

- Unity 2017.1.0f3
- C#
- Visual Studio

## Alcance Real Conseguido

- Juego multijugador local (2 jugadores)
- 3 niveles únicos diseñados manualmente
- Sistema completo de menús y navegación
- Mecánicas clásicas de Bomberman implementadas

## Timeline de Desarrollo

2.5 meses de trabajo a tiempo parcial, tras las prácticas empresariales.

Este proyecto se centró en la creación de un videojuego funcional, no solo como un ejercicio académico, sino como una aplicación práctica de los conocimientos adquiridos. El objetivo principal era integrar Unity, C# y Visual Studio para construir una experiencia de juego sólida.

# Demostración Visual del Juego



## Menú Principal

Interfaz intuitiva con opciones claras: Jugar, Partida Rápida, Instrucciones y Salir.



## Selector de Niveles

Permite a los jugadores elegir entre los 3 mapas diseñados manualmente, cada uno con desafíos únicos.



## Gameplay Nivel 1

Muestra un layout simétrico básico, ideal para familiarizarse con las mecánicas de juego.



## Gameplay Nivel 2

Presenta corredores estratégicos que fomentan el pensamiento táctico durante la partida.

Todas las imágenes presentadas son capturas directas del proyecto real funcionando, destacando la interacción entre los jugadores (P1 naranja, P2 azul), el indicador de vidas y el acceso al menú de pausa.



# Arquitectura Técnica del Juego



## Sistema de Gestión de Estados

Controlado por **GlobalStateManager** (260 líneas), gestiona el flujo del juego.



## Sistema de Jugadores y Gameplay

**Player.cs** (392 líneas) y **Bomb.cs** (82 líneas), responsables de la lógica de juego.



## Sistema de Interfaz de Usuario

**LivesUIManager** (716 líneas) y scripts de menús, manejan la UI.



## Sistema de Utilidades

Scripts auxiliares y de optimización para un rendimiento eficiente.

La arquitectura del juego se basa en cuatro bloques principales, implementando patrones de diseño como Singleton, Observer y State Machine. Esto se traduce en 13 scripts principales, sumando más de 1.200 líneas de código documentado, asegurando modularidad y mantenibilidad.

# Desarrollo Técnico: Enfoque en Unity

## Assets de Unity Store

- Modelos 3D: personajes, bombas, bloques
- Efectos de partículas para explosiones

La utilización de assets preexistentes permitió centrar el esfuerzo en la programación y la lógica del juego, en lugar de en el arte 3D, optimizando el tiempo de desarrollo.

## Desarrollo 100% Propio

- Todo el código C#
- Diseño manual de niveles
- Sistema de menús completo
- Arquitectura del juego (estados, lógica, optimizaciones)

El enfoque profesional se basa en la programación experta y el diseño meticuloso de cada componente, utilizando recursos de calidad para garantizar un resultado robusto.

Este proyecto combina la eficiencia de Unity con un desarrollo de código C# totalmente personalizado.

# Funcionalidades Implementadas y Solución de Problemas

## Problema #1: Singleton Duplicado

**Síntoma:** Puntuaciones se reseteaban aleatoriamente. **Solución:** Validación robusta en **Awake()** y uso correcto de **DontDestroyOnLoad** para la persistencia de datos entre escenas.

## Problema #2: UI Incorrecta

**Síntoma:** Corazones de vidas aparecían en el menú principal.

**Solución:** Implementación de **HideGameUIOnMenu** con detección automática del estado del juego para ocultar elementos de la UI cuando no eran necesarios.

## Problema #3: Respawn Injusto

**Síntoma:** Jugadores reaparecían sobre bombas o explosiones.

**Solución:** Algoritmo de validación de posiciones seguras para asegurar un respawn justo y evitar muertes instantáneas al reaparecer.

La metodología de desarrollo incluyó el uso estratégico de **Debug.Log**, el control de versiones con Git, y un testing exhaustivo para identificar y resolver problemas, asegurando la estabilidad y jugabilidad del sistema.

# Código en Acción

## Singleton que Funciona

```
public class GlobalStateManager : MonoBehaviour {  
    public static GlobalStateManager Instance;  
    public int player1Wins = 0, player2Wins = 0;  
  
    void Awake() {  
        if (Instance == null) {  
            Instance = this;  
            DontDestroyOnLoad(gameObject);  
        } else if (Instance != this) {  
            Destroy(gameObject);  
            return;  
        }  
    }  
}
```

Este patrón Singleton asegura que solo existe una instancia del gestor de estado global, preservando los datos como las victorias de los jugadores entre escenas.

## Respawn Seguro

```
private bool IsPositionSafe(Vector3 position) {  
    Collider[] nearby = Physics.OverlapSphere(position, 1.5f);  
    return !nearby.Any(col => col.CompareTag("Bomb") ||  
        col.CompareTag("Explosion"));  
}
```

El algoritmo propio **IsPositionSafe** verifica que el punto de reaparición no contenga bombas ni explosiones cercanas, garantizando una experiencia de juego justa.

Estos son ejemplos del código real implementado, que demuestra la aplicación de buenas prácticas de programación y soluciones a problemas comunes en el desarrollo de videojuegos.

# Métricas de Calidad

60

## FPS Objetivo

Rendimiento sostenido con un mínimo de 30 FPS, buscando 60 FPS para una experiencia fluida.

<2

## Tiempo de Carga (s)

Transiciones rápidas entre escenas, manteniendo la inmersión del jugador.

380MB

## Memoria Promedio

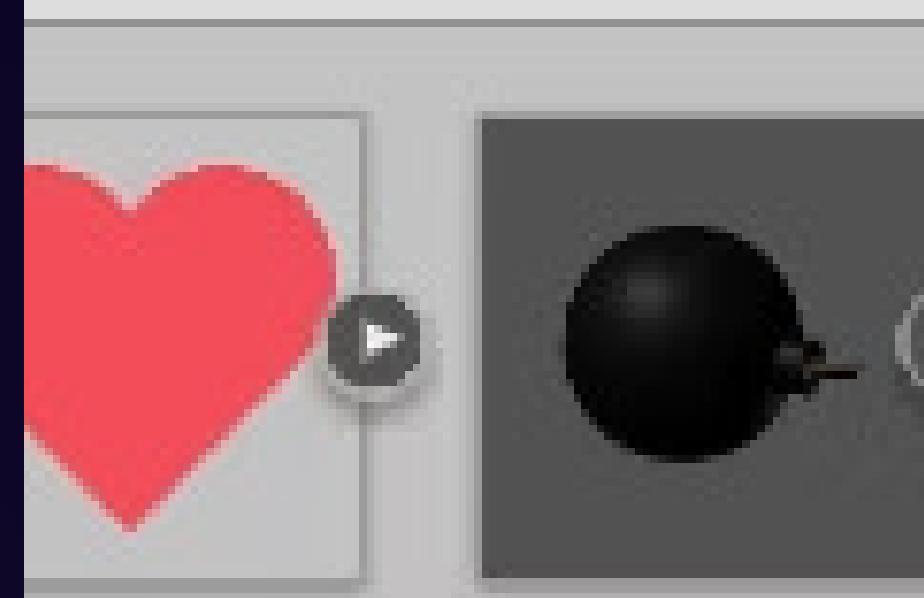
Uso optimizado de memoria, sin fugas detectadas durante las pruebas.

<50ms

## Input Lag

Respuesta consistente a los comandos del jugador, garantizando una jugabilidad precisa.

Se dedicaron más de 50 horas a pruebas manuales sistemáticas, resultando en cero crashes documentados en la versión final. Los 3 niveles fueron testeados con múltiples usuarios, abarcando incluso casos límite como la muerte simultánea. La calidad del código es alta, con más del 95% de los métodos documentados y la aplicación de patrones profesionales.



2723-r...

Bomb



# Futuras Ampliaciones

1

## Corto Plazo (1-3 meses)

- Power-ups (velocidad, rango de bomba, escudo)
- Sistema de configuración de controles
- Mejoras de accesibilidad visual

2

## Medio Plazo (3-6 meses)

- Inteligencia artificial para single-player
- Editor de niveles integrado
- Sistema de logros y estadísticas

3

## Largo Plazo (6+ meses)

- Multijugador online con Unity Netcode
- Port a móviles con controles táctiles
- Adaptación a Realidad Virtual

La arquitectura actual del proyecto está diseñada para permitir futuras expansiones de manera viable, asegurando la escalabilidad y adaptabilidad a nuevas funcionalidades y plataformas.



Game 1



Game 2



infoScenne



LevelSelec...



MainMenu



WinnerSce...

# Conclusiones y Valoración

## 1 Objetivos Cumplidos al 100%

Se ha entregado un videojuego completamente funcional con una arquitectura escalable y profesional, respaldado por 47 páginas de documentación técnica exhaustiva, demostrando las competencias DAM aplicadas en un contexto real.

## 2 Aprendizajes Clave

Este proyecto ha permitido una gestión completa desde cero, la resolución de problemas técnicos complejos, la aplicación de metodologías de desarrollo profesional y la comprensión de la importancia del testing y la documentación.

## 3 Preparación Profesional y Legado

Las competencias adquiridas en Unity y C# me sitúan en un nivel junior-intermedio, listo para afrontar desafíos en el sector. Además, este material es reutilizable para futuros estudiantes de DAM.

"De estudiante que aprende a programar a programador que puede resolver problemas complejos."