
**Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Querétaro**

# Multiagentes Gráficos

## TC2008B

## Actividad Integradora Gráficas

## NIVEL - FACIL

**Carlos Martínez Vazquez - A01711730**

# Introducción

Este proyecto implementa un sistema de bullet hell con tres patrones matemáticos distintos:

- - Patrón de Estrella (0-10 segundos)
- - Patrón de Ondas (10-20 segundos)
- - Patrón de Espirales (20-30 segundos)

Cada patrón utiliza funciones matemáticas específicas para crear formaciones visuales únicas

y desafiantes, implementadas en Unity 2D con C#.

# Explicación técnica de cada patrón

2.1 PATRÓN DE ESTRELLA (StarShoot.cs)

IMPLEMENTACIÓN:

- - Utiliza coordenadas polares para crear formaciones en estrella
- - Fórmula matemática: r = a + b*cos(k*θ) donde k = número de puntas
- - 6 balas por disparo con cooldown de 0.8 segundos

CÓDIGO CLAVE:

```csharp

float starRadius = 1f + 0.6f * Mathf.Cos(starPoints * angle);

Vector3 bulletDirection = new Vector3(

Mathf.Cos(angle) * starRadius,

Mathf.Sin(angle) * starRadius,

0f

).normalized;
```

```

CARACTERÍSTICAS TÉCNICAS:

- - starPoints = 5 (estrella de 5 puntas)
- - Rotación continua con rotationOffset += Time.deltaTime * 30f
- - Variación de velocidad basada en la posición en la estrella
- - Alternancia entre radio interior y exterior para efecto visual

2.2 PATRÓN DE ONDAS (LinearShoot.cs)

IMPLEMENTACIÓN:

- - Utiliza funciones seno para crear ondas matemáticas
- - Fórmula: y = A * sin(f * x + ϕ) donde A=amplitud, f=frecuencia, ϕ=fase
- - 3 balas por onda con cooldown de 1.0 segundos

CÓDIGO CLAVE:

```csharp

float sineInput = waveFrequency * Mathf.PI * t + wavePhase;

float y = waveAmplitude * Mathf.Sin(sineInput);

Vector3 waveDirection = new Vector3(

x * 0.3f,

- -1f + y * 0.2f,

0f

).normalized;

```

CARACTERÍSTICAS TÉCNICAS:

- - waveWidth = 6f (ancho de la onda)
- - waveFrequency = 2f (dos ondas completas)
- - waveAmplitude = 1f (altura de las curvas)
- - Animación con wavePhase += Time.deltaTime * 3f
```

2.3 PATRÓN DE ESPIRALES (RadialShoot.cs)

IMPLEMENTACIÓN:

- - Utiliza espirales logarítmicas y patrones circulares
- - Fórmula: r = a * e^(b*θ) para crecimiento exponencial
- - 6 balas por ráfaga + 4 balas interiores, cooldown de 0.8 segundos

CÓDIGO CLAVE:

```csharp

float radius = 0.5f + radiusGrowth * Mathf.Exp(spiralTightness * t * 3f);

Vector3 direction = new Vector3(

Mathf.Cos(angle) * radius,

Mathf.Sin(angle) * radius,

0f

).normalized;

```

CARACTERÍSTICAS TÉCNICAS:

- - spiralArms = 2 (dos brazos espirales)
- - spiralTightness = 0.5f (qué tan apretada es la espiral)
- - Rotación con spiralPhase += Time.deltaTime * 60f
- - Patrón circular interior para mayor densidad visual

# Justificación de diferencias entre patrones

3.1 DIFERENCIAS MATEMÁTICAS:

- - ESTRELLA: Usa coordenadas polares con coseno para crear puntas definidas
- - ONDAS: Usa función seno para movimiento fluido y ondulante
- - ESPIRALES: Usa función exponencial para crecimiento radial continuo

3.2 DIFERENCIAS VISUALES:

- - ESTRELLA: Patrones geométricos rígidos con puntas afiladas
- - ONDAS: Movimiento fluido y orgánico tipo sinusoidal
- - ESPIRALES: Crecimiento expansivo desde el centro hacia afuera
-

3.3 DIFERENCIAS EN GAMEPLAY:

- - ESTRELLA: Reto de esquivar patrones predecibles pero densos
- - ONDAS: Requiere movimiento lateral para esquivar ondas
- - ESPIRALES: Demanda movimiento circular y anticipación del crecimiento

3.4 DIFERENCIAS TÉCNICAS:

- - Velocidades diferentes: 0.8s, 1.0s, 0.8s respectivamente
- - Cantidad de balas: 6, 3, 10 balas por disparo
- - Algoritmos matemáticos completamente distintos
- - Patrones de movimiento del jefe únicos para cada fase

## Sistema de gestión

4.1 SISTEMA DE POOLING (BulletPool.cs):

- - Pool inicial de 200 balas, máximo 500
- - Reutilización de GameObjects para optimizar rendimiento
- - Evita Instantiate/Destroy constante que causa lag

4.2 CONTADOR EN TIEMPO REAL (BulletCounter.cs):

- - Muestra balas activas en pantalla instantáneamente
- - Colores de advertencia: Blanco (0-149), Amarillo (150-299), Rojo (300+)
- - Ayuda a monitorear rendimiento del juego

4.3 - Sistema de tiempo acelerado (3x velocidad)

- - Eventos OnSecondChanged para transiciones de fase
- - Ciclos de 30 segundos que se repiten infinitamente

4.4 CONTROLADOR PRINCIPAL (BossController.cs):

- - Coordina las tres fases con transiciones automáticas
- - Sistema de ciclos: 0-10s, 10-20s, 20-30s, repetir
- - Movimiento dinámico del jefe durante cada fase

Retos Enfrentados y soluciónes

## Retos Enfrentados y soluciones

5.1 RETO: Visibilidad de las balas

PROBLEMA: Las balas no aparecían en pantalla

SOLUCIÓN: Configurar Sorting Layer "Projectiles" y Order in Layer correcto

APRENDIZAJE: La importancia de las capas de renderizado en Unity 2D

5.2 RETO: Rendimiento con muchas balas

PROBLEMA: Lag severo con 200+ balas simultáneas

SOLUCIÓN:

- - Implementar object pooling eficiente
- - Reducir cantidad de balas por patrón
- - Optimizar cooldowns de disparo

APRENDIZAJE: El balance entre espectáculo visual y rendimiento

5.3 RETO: Sincronización de fases

PROBLEMA: Las fases no cambiaban correctamente

SOLUCIÓN: Sistema de eventos con TimeManager y ciclos modulares

APRENDIZAJE: La importancia de sistemas de eventos desacoplados

5.4 RETO: Matemáticas complejas en tiempo real

PROBLEMA: Cálculos trigonométricos causaban stuttering

SOLUCIÓN: Pre-calcular valores cuando sea posible y optimizar loops

APRENDIZAJE: Optimización de operaciones matemáticas en Update()

5.5 RETO: Debugging del contador de balas

PROBLEMA: El contador no reflejaba las balas reales

SOLUCIÓN: Implementar OnEnable/OnDisable correctamente sin double-counting

APRENDIZAJE: Manejo cuidadoso del ciclo de vida de GameObjects

## Aprendizajes

6.1 MATEMÁTICAS APLICADAS:

- - Implementación práctica de funciones trigonométricas
- - Conversión entre coordenadas polares y cartesianas
- - Uso de funciones exponenciales para crecimiento orgánico
- - Animación matemática con variables de tiempo

6.2 ARQUITECTURA DE CÓDIGO:

- - Separación de responsabilidades en scripts específicos
- - Uso de eventos para comunicación entre sistemas
- - Implementación de patterns como Object Pooling
- - Gestión eficiente de memoria y rendimiento

6.3 UNITY 2D ESPECIALIZADO:

- - Configuración correcta de Sorting Layers
- - Manejo de Collider2D para detección de límites
- - Optimización de SpriteRenderer para múltiples objetos
- - Sistema de coordenadas 2D y transformaciones

6.4 DISEÑO DE GAMEPLAY:

- - Balance entre dificultad y jugabilidad
- - Creación de patrones visualmente atractivos
- - Implementación de feedback visual en tiempo real
- - Diseño de ciclos de juego infinitos

## Conclusión

Este proyecto demuestra la implementación exitosa de patrones bullet hell matemáticamente diferenciados, cada uno con sus propias características técnicas y visuales. La combinación de matemáticas aplicadas, arquitectura de software sólida y optimización de rendimiento resulta en una experiencia de juego fluida y visualmente impactante.

Los principales logros incluyen:

- - Tres patrones matemáticos completamente distintos
- - Sistema de pooling eficiente para 500+ balas
- - Contador en tiempo real para monitoreo de rendimiento
- - Ciclos infinitos de gameplay
- - Arquitectura modular y extensible

## Recursos y Enlaces

VIDEO DEMOSTRATIVO:

**https://drive.google.com/drive/folders/1I2xgHw9lLInnotSP2I0nPM0EkhC-TZpM**

REPOSITORIO DE CÓDIGO:

https://github.com/CsVazquezz/BulletHellShooter_TC2008B