

**Universitat de Barcelona**

**Escola de Noves Tecnologies Interactives**

*Grau en Continguts Digitals Interactius*

*Mecánica de Ritmo*

*Marc Ramis Caldés*

**Projecte Aplicat de Videojoc**

**Curs 2022-2023**

*Marc Ramis Caldés*

*Álex Pérez Ruiz*

## Abstract

The main goal of this project is to develop a robust rhythm mechanic that enhances the final degree project game. Throughout this document, readers will gain knowledge on working with song rhythms and explore the intricacies of the implemented rhythm mechanics in the game.

The primary focus is on creating an immersive and captivating gameplay experience by incorporating rhythmic elements. Various techniques and strategies will be explored to ensure that music and rhythm are integral to the game's gameplay. Readers will have the opportunity to dive into the design process, starting from selecting and adapting suitable songs to implementing rhythm mechanics within the game engine.

To summarize, this project aims to provide readers with a comprehensive understanding of creating and implementing a rhythm mechanic in a game. By combining theoretical insights, practical examples, and actionable advice, it offers a complete guide to achieve a musically enriched and highly satisfying gameplay experience.

## Keywords

Rhythm, Combo, Symon Says, Soundtrack Manager, Buttons Sequence, Spectrum, Beat.

## Continguts

Abstract	2
Keywords	2
1. Introducción y Estado del arte	5
1.1. Contexto	5
1.2. Área de conocimiento	5
1.2.1. Tipo de ritmos	6
1.2.2. Calcular el ritmo de la canción	7
2. Objetivos	8
3. Diseño de la solución	9
3.1. Ritmo pautado	9
3.1.1. Diferenciar instrumentos musicales	9
3.1.2. Combos	10
3.2. Ritmo libre	10
3.2.1. Simón dice	10
3.2.1.1. Primera propuesta	10
3.2.1.2. Segunda propuesta	10
3.2.2. Varios instrumentos musicales	11
4. Desarrollo	12
4.1. Ritmo pautado	12
4.1.1. Calcular el ritmo	12
4.1.1.1. Volumen	13
4.1.1.2. Sincronización ≠ Reacción	13
4.1.2. Configuración de la canción	14
4.1.3. Controlador del ritmo	15
4.1.4. Beat	16
4.1.5. Estructura de código	19
4.1.6. Combo	20
4.1.7. Mecánica	21
4.1.7.1. Prototipado	21
4.1.7.2. Resultado final	23
4.2. Ritmo libre	25
4.2.1. Inputs	25
4.2.2. Secuencia de botones	26
4.2.3. Lógica	26
4.2.3.1 Primera propuesta	27
4.2.3.2 Segunda propuesta	29
4.2.3.3. Estructura de código	30

5. Conclusiones y líneas de futuro	31
5.1. Conclusión general	31
5.2. Conclusión de los objetivos	31
5.3. Líneas de futuro	32
6. Bibliografia	34

## 1. Introducción y Estado del arte

### 1.1. Contexto

Se plantea desarrollar una mecánica de ritmo que implica sincronizarse con el ritmo de la música para generar dinámicas de juego que complementen como mecánica a un juego de Trabajo de Final de Grado para Unity y con C#.

### 1.2. Área de conocimiento

El área de conocimiento en el que se está trabajando se centra en la mecánica de pulsar secuencias de botones en videojuegos, aprovechando el ritmo de la música como elemento fundamental. Esta mecánica ha sido ejemplificada en juegos como Guitar Hero y Hi-Fi Rush.

Guitar Hero ([Guitar Hero Live Controller Issues, n.d.](#)), aunque considerado un referente antiguo, sigue siendo un claro ejemplo de cómo se basa en la pulsación precisa de secuencias de botones para su jugabilidad. El juego utiliza esta mecánica como núcleo, permitiendo a los jugadores simular tocar la guitarra al seguir los patrones de notas en pantalla, sincronizados con la música.



Figura 1. Mecánica de guitar hero con su controlador

Un ejemplo más reciente es Hi-Fi Rush ([Hi-Fi RUSH, n.d.](#)), un videojuego que también se apoya en el ritmo de la canción para mejorar las dinámicas de juego y brindar una experiencia de usuario altamente satisfactoria. En este juego, el sistema de ritmo se utiliza para llevar a cabo secuencias de botones en determinadas situaciones, lo que proporciona un desafío adicional y permite a los jugadores sentir que están creando el ritmo musical a medida que interactúan con el juego.

Estos ejemplos demuestran cómo la pulsación de secuencias de botones en sincronía con el ritmo musical se ha convertido en un elemento distintivo en el diseño de juegos, ofreciendo una experiencia envolvente y entretenida para los jugadores. El uso de esta mecánica no solo agrega

diversión y desafío, sino que también permite una mayor interacción con la música y el juego en sí, creando una experiencia de usuario más inmersiva y gratificante.

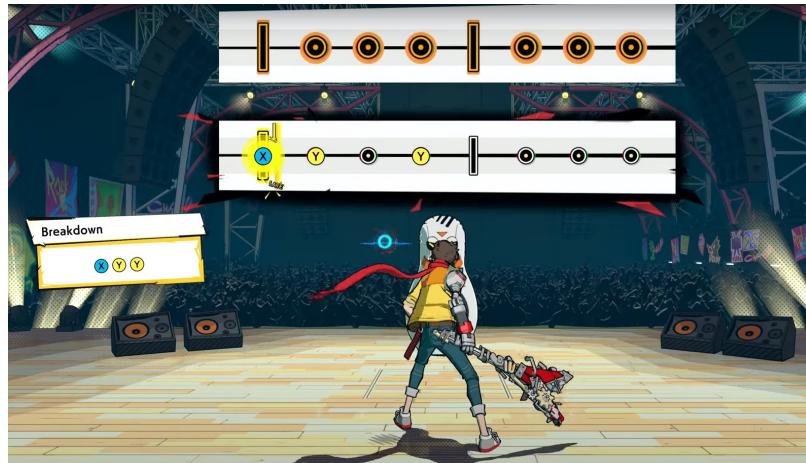


Figura 2. Hi-Fi Rush secuencia de botones

### 1.2.1. Tipo de ritmos

En Hi-Fi Rush se pueden distinguir dos tipos de mecánicas de ritmo. En primer lugar, el juego utiliza una secuencia de ritmo que coincide con el feedback que recibe el jugador en el momento en que debe realizar una acción. Por ejemplo, el jugador puede recibir una indicación visual o auditiva de que debe pulsar un botón o realizar una acción en el momento preciso en que la canción alcanza un cierto punto de ritmo o de tempo.

En segundo lugar, Hi-Fi Rush también emplea un tipo de mecánica de ritmo que se genera por el jugador cuando este coincide acciones con el ritmo de la canción. En este caso, el juego premia al jugador por realizar acciones en sincronía con la música, lo que puede mejorar la experiencia del usuario y aumentar su puntuación en el juego.

Estos dos tipos de mecánicas de ritmo se combinan en Hi-Fi Rush para crear una experiencia de juego emocionante y desafiante para el jugador. La combinación de ritmos que responden al feedback del jugador y la capacidad del jugador para generar ritmos propios basados en su propio sentido del ritmo, hacen que Hi-Fi Rush sea un juego de ritmo único y atractivo para una amplia audiencia. Además, el uso de estas mecánicas de ritmo también puede mejorar la capacidad del jugador para detectar patrones de ritmo y mejorar su habilidad en la sincronización con la música, lo que puede tener beneficios más allá del mundo del juego.

### 1.2.2. Calcular el ritmo de la canción

Para calcular el ritmo de una canción en Unity, hay dos enfoques principales que se pueden utilizar. El primero es calcular el beat<sup>1</sup> del audio a partir del beat per minute (BPM) de la canción, mientras que el segundo es calcular los datos del espectro del audio de la canción en Unity.

El cálculo del beat del audio se realiza a través de la detección de los golpes o pulsos de la canción y la medición de su frecuencia en BPM. Para ello, se puede utilizar la técnica de detección de picos en el espectro de audio que se mencionó anteriormente. Al detectar los picos de energía en el espectro de audio, se pueden identificar los golpes de la música y calcular su frecuencia en BPM. Una vez que se conoce la frecuencia en BPM de la canción, se puede utilizar para sincronizar los elementos del juego con la música. ([Tattersall, n.d.](#))

Por otro lado, el cálculo de los datos del espectro<sup>2</sup> del audio se realiza mediante el análisis del espectro de frecuencia de la canción en tiempo real. Para ello, se pueden utilizar las herramientas de análisis de audio que se encuentran en la API de Unity. Una vez que se han calculado los valores de energía para las diferentes bandas de frecuencia del espectro de audio de la canción, se pueden utilizar para crear efectos visuales o elementos de juego que se sincronicen con la música. ([Algorithmic Beat Mapping in Unity: Real-Time Audio Analysis Using the Unity API, 2018](#))

Ambos enfoques tienen sus ventajas y desventajas. El cálculo del beat del audio es más rápido y fácil de implementar, pero puede no ser tan preciso en algunas situaciones, especialmente si la canción tiene un ritmo complejo. Por otro lado, el cálculo del espectro de audio es más preciso pero también es más exigente en cuanto a procesamiento y puede ser más difícil de implementar.

---

<sup>1</sup> El "beat" es el pulso o ritmo básico de una canción, es decir, la unidad de tiempo que se repite a lo largo de la música y que permite que los oyentes se muevan al compás de la misma.

<sup>2</sup> El espectro de un sonido se define como la representación de la distribución de energía sonora de dicho sonido en función de la frecuencia. El espectro es importante porque la percepción auditiva del sonido es de naturaleza predominantemente espectral.

## 2. Objetivos

Los objetivos del proyecto son:

1. Investigar sobre mecánicas de ritmo en Unity:
  - a. Investigar mecánicas que se sincronizan con el ritmo de la música.
  - b. Investigar mecánicas que sincronizan y crean el ritmo de la música.
2. Implementar la mecánica de ritmo que se sincroniza con el ritmo de la música:
  - a. Implementar el ritmo en mitad de un sistema de pelea.
  - b. Implementar un sistema de combos que funciona con el ritmo de este sistema de pelea.
3. Implementar la mecánica de ritmo que crea el ritmo de la música:
  - a. Implementar la mecánica en mitad de un sistema de lanzamiento.
4. Implementar un editor de la mecánica de ritmo:
  - a. Implementar que se puedan crear en el editor secuencias de botones.
  - b. Hacer el package del editor para poder importarlo a otros proyectos.
5. Implementar telemetrías

### **3. Diseño de la solución**

Si recordamos la sección de tipos de ritmo en el estado del arte sobre la investigación llevada de Hi-Fi Rush se pueden distinguir dos tipos de ritmo sobre los que se van a diseñar las mecánicas de juego. Una primera, a la que se puede denominar cómo ritmo pautado, que obliga al jugador a realizar acciones durante el ritmo de la canción y por otro lado, el ritmo libre, que genera música que va al compás de la canción para premiar al jugador cuando realiza acciones al ritmo de esta.

#### **3.1. Ritmo pautado**

Teniendo en cuenta que se diseña para luego poder exportarlo a otros proyectos, más en concreto que sea una feature que pueda complementar el trabajo de Trabajo de Final de Grado, se desea que este ritmo sea escalable y optimizado según la medida de lo posible.

Lo primero que se piensa para desarrollar un juego de ritmo es coincidir actuaciones del jugador y sincronizarlas al ritmo de la canción.

Por tanto, se piensa cómo calcular el ritmo de la canción, que haciendo caso al estado del arte, a las maneras de calcular el ritmo de una se obtiene la del cálculo del beat y el cálculo del espectro. En este caso se escoge la segunda opción porque en este proyecto sólo es necesario saber cuándo ocurre el ritmo de la canción y no existe una complejidad mayor que observar el espectro y determinar la intensidad con una constante.

Además, la primera opción trabaja sobre una sola canción y con un beat per minute (bpm) determinado por la misma canción. Si se desea trabajar con canciones externas es más complicado determinar cuál ha sido el bpm usado.

##### **3.1.1. Diferenciar instrumentos musicales**

A pesar de que ya parece que se ha creado la mecánica de ritmo, existe aún un problema con esta manera de implementarlo. Esta consiste en que recoger la intensidad a la que suena un espectro repercute sobre todos los instrumentos de la canción, por tanto no se puede determinar con qué instrumento tiene que coincidir el jugador, lo que podría causar confusión y frustración.

Para solucionar este problema hay varias maneras:

- La primera es la más sencilla y la más barata, pero también la más exhaustiva, que consiste en remarcar la constante en la que debe encontrarse la intensidad del espectro para encajar el ritmo.
- La segunda, un poco más cara y complicada porque consiste en calcular la intensidad de cada uno de los instrumentos que se usan en una canción y obtener sin ruido externo la intensidad de manera más sencilla.

En el caso de este proyecto se ha usado la segunda porque a nivel de proyecto ofrecía más opciones que se comentarán más adelante.

### **3.1.2. Combos**

Además, otra de las mecánicas que complementa el ritmo es presionar correctamente secuencias de ritmo. Es decir, la definición de combo: varias cosas que vienen juntas.

En este caso, el objetivo de tener combos es generar feedback después de generar varios hasta llegar a un clímax final con el final combo. Este último combo representaría una situación satisfactoria para los jugadores.

## **3.2. Ritmo libre**

Por otra parte, también se desea desarrollar un sistema de música que permite recibir feedback tras una actuación efectiva con la coincidencia del ritmo pautado, es decir, a partir de hacer coincidir el beat en el que suena el ritmo crear la composición musical.

### **3.2.1. Simón dice**

#### **3.2.1.1. Primera propuesta**

Esto se puede conseguir de una manera sencilla, y es poniendo primero como ejemplo a un agente exterior que ya conoce el patrón y enseña al jugador cómo hacerlo, dónde ya se genera una dinámica de juego.

Un minijuego como Simón dice puede ser una solución efectiva porque primero existe un agente que enseña la secuencia que debe realizar y con qué ritmo que el jugador debe recordar y en la siguiente realizar el mismo tempo<sup>3</sup>.

#### **3.2.1.2. Segunda propuesta**

Sin embargo, con el objetivo de lograr un diseño más cuidado y ofrecer una experiencia de juego más satisfactoria con mayores posibilidades de éxito, se recomienda utilizar esta secuencia específica de botones. De esta manera, se logra una combinación musical concreta y se le da un propósito a la secuencia de botones.

Por otro lado, si el jugador presiona cualquier botón sin seguir una secuencia específica, se generará una composición musical aleatoria. Esto implicará la incorporación gradual de cada instrumento a medida que el jugador sigue el ritmo de los instrumentos. Esta dinámica hará que el jugador crea que está creando el ritmo de la música.

---

<sup>3</sup> El término tempo hace referencia a la velocidad con la que debe ejecutarse una pieza musical.

### **3.2.2. Varios instrumentos musicales**

Pero esta mecánica realmente ya está creada durante la primera parte en la mecánica de ritmo pautada. Lo interesante en esta parte es que se pueden recoger los diferentes instrumentos de una canción y ponerlas en silencio durante la parte en la que el jugador no está siguiendo correctamente el tempo y cuando sí lo sigue subir el volumen.

De esta manera, cómo los beats de cada instrumento realmente se están reproduciendo pero con el volumen bajado, parece que el jugador está creando música al ritmo.

## 4. Desarrollo

El apartado de desarrollo se divide en los mismos apartados que la parte centrada en el diseño de la solución. Aquí se concretan los pasos que se han llevado a cabo para desarrollar las mecánicas y cómo se ha compaginado con el proyecto de TFG para poder implementarlo y llevar a cabo tanto la parte visual como sonora del feedback.

Se ha de destacar que el proyecto se ha producido con Unity y que la mecánica de ritmo primero ha pasado por una etapa de prototipado en un proyecto aparte al del TFG. Dentro del proyecto de TFG, se ha creado una escena llamada RhythmLevel desde la que se han llevado las implementaciones finales que tienen que ver más con el aspecto visual de cómo ha quedado finalmente la mecánica.

### 4.1. Ritmo pautado

#### 4.1.1. Calcular el ritmo

Cómo ya se ha explicado en el apartado de diseño de la solución, lo primero que se quiere encontrar en una mecánica de ritmo es el momento del ritmo de una canción en concreto. Para hacerlo, se ha creado una clase llamada Instrument, que es quien se encarga de calcular el ritmo de su mismo instrumento. Se define cómo un instrumento porque, cómo se explicaba en el apartado anterior, tener los instrumentos de una canción separados en lugar de la canción entera es útil más adelante para poder desarrollar la mecánica de ritmo libre.

En la figura 3 se calcula la intensidad del ritmo de una fuente de audio dada, utilizando la transformada de Fourier rápida (FFT).

```
I reference
public static float CalculateSpectrumIntensity(Instrument instrument)
{
    // Obtener los datos de audio del audio source
    instrument.instrumentRef.GetSpectrumData(instrument.audioSamples, 0, FFTWindow.BlackmanHarris);

    // Sumar los valores absolutos de las muestras de audio para obtener una medida de la intensidad del ritmo
    float sum = 0f;
    for (int i = 0; i < instrument.audioSamples.Length; i++)
    {
        sum += Mathf.Abs(instrument.audioSamples[i]);
    }

    // Obtener medida de intensidad promedio del ritmo
    instrument.intensity = sum / instrument.audioSamples.Length;

    // Se multiplica por un factor porque los valores pueden ser muy pequeños y la constante con la
    // que poder comprender cuando es el momento de mayor intensidad podría ser difícil de encontrar
    instrument.intensity *= instrument.multiplierNeeded;

    return instrument.intensity;
}
```

Figura 3. Cálculo del ritmo por el espectro

La línea de código “instrument.instrumentRef.GetSpectrumData(instrument.audioSamples, 0, FFTWindow.BlackmanHarris)” obtiene los datos de audio del objeto “instrumentRef”, que es un AudioSource de Unity, y los almacena en un arreglo llamado “audioSamples”. Se utiliza la función “GetSpectrumData” para aplicar una ventana de Blackman-Harris<sup>4</sup> en la transformada de Fourier, lo que mejora la resolución en frecuencia y reduce el efecto de las fugas espectrales.

A continuación, se suman los valores absolutos de las muestras de audio en el arreglo “audioSamples” en la variable “sum”. Luego, se divide la suma por la longitud del arreglo para obtener una medida de la intensidad promedio del ritmo. Finalmente, se multiplica el resultado por “multiplierNeeded”, qué es un factor de escala que puede ser ajustado según se deseé.

Una vez se ha realizado el cálculo, se ha de especificar cuál es el rango de intensidad a superar con un threshold<sup>5</sup> para devolver el momento del ritmo en cuestión. (Figura 4)

```
public bool IsIntensityGreater() { return intensity > threshold; }
```

Figura 4. Obtener la intensidad más alta

#### 4.1.1.1. Volumen

Existe un problema con este método si se modifica el volumen de la canción y es que si el volumen se aumenta o disminuye, el valor de intensidad varía y por tanto el threshold que se propone sería inconsistente. Para arreglar este problema se propone multiplicar ambas variables por el volumen de la canción, de esta manera tanto la intensidad como el threshold propuesto sería siempre relativo en función del volumen de la música. (Figura 5)

```
public bool IsIntensityGreater() { return (intensity * instrumentRef.volume) > (threshold * instrumentRef.volume); }
```

Figura 5. Obtener la intensidad más alta relativo al volumen

Otro problema que puede surgir por variar el volumen, es si este puede ser modificado por el jugador y llega a 0, la música dejaría de escucharse y por tanto la intensidad del espectro sería nula, en este caso debería haber un límite mínimo como por ejemplo 0.001 que fuera imposible de escuchar por el oído humano pero si para la máquina.

#### 4.1.1.2. Sincronización ≠ Reacción

Un problema que se presenta durante la integración del ritmo es la falta de sincronización por parte del jugador, ya que reacciona al ritmo en lugar de mantenerse en sincronía con él. Esto ocurre debido a que el período de tiempo en el que se abre la oportunidad de respuesta comienza cuando se alcanza la intensidad más alta.

<sup>4</sup> Las ventanas de Blackman-Harris son una familia de funciones que se utilizan para reducir este efecto de fuga espectral. Estas ventanas son suaves y se desvanecen gradualmente hacia cero en los extremos, lo que reduce el efecto de las discontinuidades en los bordes de la ventana.

<sup>5</sup> Un threshold es valor límite o punto de corte que se utiliza para tomar decisiones o realizar filtrado en un proceso de análisis de datos.

Para abordar esta cuestión, se ha propuesto una solución consistente en crear duplicados de los instrumentos que se reproducirán. Los instrumentos originales se establecen en un volumen de 0.001 para que no sean audibles para el oído humano, mientras que los duplicados son los que el jugador escucha. Sin embargo, en realidad, estos duplicados se sincronizan con los originales, que comienzan la canción ligeramente antes para lograr el efecto sincronizado deseado.

#### 4.1.2. Configuración de la canción

Una vez se ha obtenido el cálculo del ritmo, ahora se requiere configurar la canción, para poder hacerlo se ha creado una clase llamada SoundtrackManager que contiene las referencias de los instrumentos y también del instrumento que es el que proporciona el momento del ritmo.

Esta clase está pensada de tal manera que pueda ser configurable desde el Inspector de Unity. Cómo se puede observar en la Figura 6, los valores de threshold y multiplierNeeded son los mismos porque cómo se explicaba en el apartado de diseño de la solución, al dividir las canciones por instrumentos, encontrar la constante que deba superar la intensidad más alta no es tan preciso y por tanto los valores muy altos siempre serán los mismos.

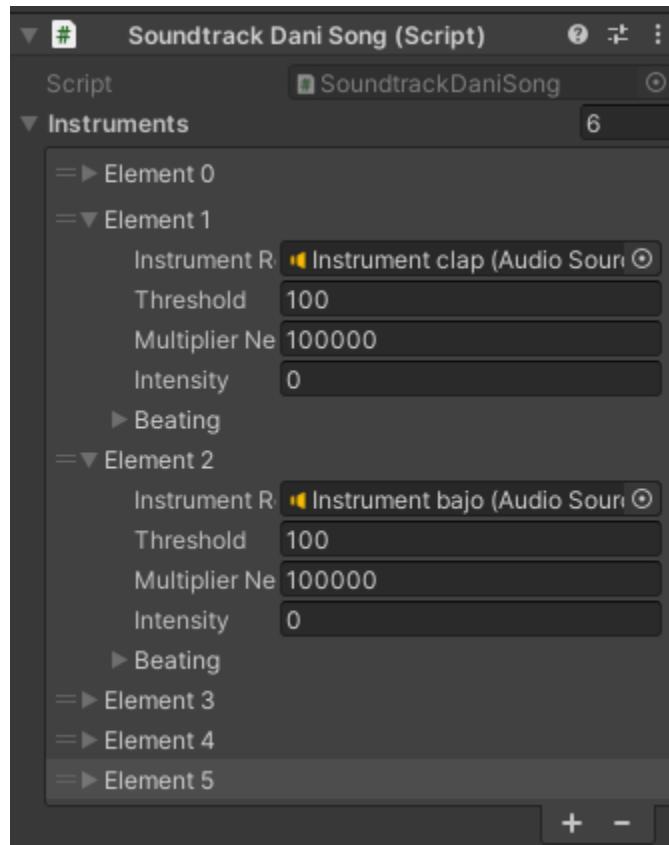


Figura 6. Configuración de la canción

#### 4.1.3. Controlador del ritmo

Por otra parte, también interesa saber cuál es la canción que ha de sonar y a la que los jugadores deben reaccionar, ya que si estuvieran sonando todas las canciones al mismo tiempo provocaría mucha discordancia al jugador y además, se estarían realizando cálculos extra de canciones a las que no se reaccionan.

Por ello se implementa la clase RhythmController, una clase Singleton<sup>6</sup> que contiene el SoundtrackManager de que la se va a calcular el ritmo. En las figuras 7 y 8 se puede observar cómo se hace uso de lo que necesita para que la canción calcule el ritmo.

```
④ Unity Message | 0 references
private void Start()
{
    soundtrackManager.InitializeSequence();

}

④ Unity Message | 0 references
private void Update()
{
    soundtrackManager.UpdateSoundtracks();
```

Figura 7. Llamada al Init y el Update de la canción en el RhythmController

```
INHERITED
public virtual void UpdateSoundtracks()
{
    CheckIfMusicFinalized();

    foreach (Instrument i in instruments)
    {
        i.UpdateSpectrumIntensity();
    }
}
```

Figura 8. Update de la canción

En la función UpdateSoundtracks existe una función que comprueba si la canción ha finalizado y la vuelve a ejecutar para que suene en loop (Figura 9), ya que al dividir la canción por instrumentos, si la secuencia no se exporta con la misma duración es posible que en la siguiente iteración para que suene la canción los instrumentos no estén sincronizados.

---

<sup>6</sup> Una clase Singleton es un patrón de diseño de software que se utiliza para asegurar que una clase tenga solo una instancia en toda la aplicación y que esta instancia sea accesible desde cualquier parte del programa.

```

private void CheckIfMusicFinalized()
{
    foreach (Instrument i in instruments)
    {
        if (i.instrumentRef != null)
        {
            if (i.instrumentRef.isPlaying)
            {
                return;
            }
            else
            {
                ReloadSong();
            }
        }
    }
}

```

*Figura 9. Comprobación para saber si todos los instrumentos han terminado*

#### 4.1.4. Beat

El Beat en este proyecto se ha definido como la intensidad más alta del espectro, el problema de esto, es que esta intensidad puede sonar durante varias milésimas de segundos y sólo interesa que lo que se considera un beat suene una vez cada vez que aparezca un nuevo Beat.

Para conseguir esto, se define en el código una clase Beat que maneja eventos de ritmo de una canción. La clase tiene un temporizador que se encarga de detectar los latidos de la canción y emitir un evento de ritmo correspondiente. El temporizador se inicializa con un tiempo de espera definido por el atributo “cd” y se actualiza en cada frame con el método Update. El método Update recibe un parámetro “isBeat” que indica si se ha detectado un latido en el momento actual, y si es así, se inicia el temporizador y se emite el evento de ritmo correspondiente. Además, el código define una delegate<sup>7</sup> “BeatEvent” y un evento “OnBeat” que se emite cada vez que se detecta un latido en la canción para que sea fácilmente accesible desde otros códigos. (Figura 10, 11 y 12)

---

<sup>7</sup> Un delegate es un tipo de dato que representa un puntero a una función y se utiliza para definir métodos que pueden ser pasados como argumentos a otros métodos o ser almacenados en una variable. Los delegates son útiles cuando se quiere encapsular una operación en un objeto que se puede pasar como parámetro o que se puede almacenar en una variable y llamar en cualquier momento. En C#, los delegates se definen utilizando la sintaxis `delegate`, y se pueden usar para crear eventos, que permiten que los objetos notifiquen a otros objetos cuando ocurre un evento.

```
// Constructor de la clase
2 references
public Beat()
{
    // Inicializa el temporizador con el valor de "cd"
    rythmTimer = new MTimer(cd);

    // Asigna un manejador de eventos "OnTimerEnd" al temporizador
    rythmTimer.OnTimerEnd += ResetRythm;
}
```

Figura 10. Constructor de la clase Beat

```
// Método utilizado para actualizar el estado del objeto "Beat" cada vez que se actualiza el marco del juego
1 reference
public void Update(Instrument[] instruments)
{
    foreach (Instrument i in instruments)
    {
        // Si se detecta un beat y no se ha detectado otro previamente
        if (i.isBeating && i.IsIntensityGreater() && !canRythm)
        {
            // Inicia el temporizador
            rythmTimer.StartTimer();

            // Invoca el evento "OnBeat"
            OnBeat?.Invoke();

            // Indica que no se puede detectar otro beat hasta que el temporizador alcance su límite de tiempo
            canRythm = true;
        }
    }

    // Actualiza el temporizador
    rythmTimer.Update(Time.deltaTime);
}
```

Figura 11. Update del Beat de todos los instrumentos en la clase Beat

```
// Método utilizado para restablecer la variable "canRythm" a false cuando el temporizador alcanza su límite de tiempo
1 reference
private void ResetRythm()
{
    canRythm = false;
}
```

Figura 12. Método que reinicia el momento del ritmo

En la Figura 13, dentro de la clase del Rhythm Controller en el Update se llaman a los Beats que se quieren acceder con tal de poder llamarlos desde otras implementaciones y así tener el momento del ritmo.

```
④ Unity Message | 0 references
private void Update()
{
    soundtrackManager.UpdateSoundtracks();

    beat.Update(soundtrackManager.GetBaseInstrument().IsIntensityGreater());
    beat2.Update(soundtrackManager.GetAllInstruments()[2].IsIntensityGreater());
}
```

Figura 13. Cálculo del latido de una canción

A partir de ahora, para poder usar el beat desde otras implementaciones. Es tan fácil cómo acceder a la clase Singleton, acceder a la delegate de la clase Beat con el que se quiere seguir el ritmo de la canción y aplicar operaciones. (Figura 14)

```
private void Awake()
{
    RythmController.instance.beat.OnBeat += Rythm;
}
```

Figura 14. Aplicar operaciones con el Beat

#### 4.1.5. Estructura de código

A continuación (Figura 15) se muestra cómo queda la estructura de código de esta secuencia tras haber completado la mecánica de ritmo pautada. Se puede observar que las clases RhythmController y SoundtrackManager heredan de MonoBehaviour, ya que para poder usarlas desde el inspector es necesario heredar de esta clase. Por otra parte, se pueden destacar dos clases más no mencionadas anteriormente, la primera es MTimer, que funciona como temporizador para resetear el cálculo del beat y la segunda es SpectrumOperations, una clase estática que sirve para separar lo que son las operaciones realizadas en el espectro de la clase Instrument.

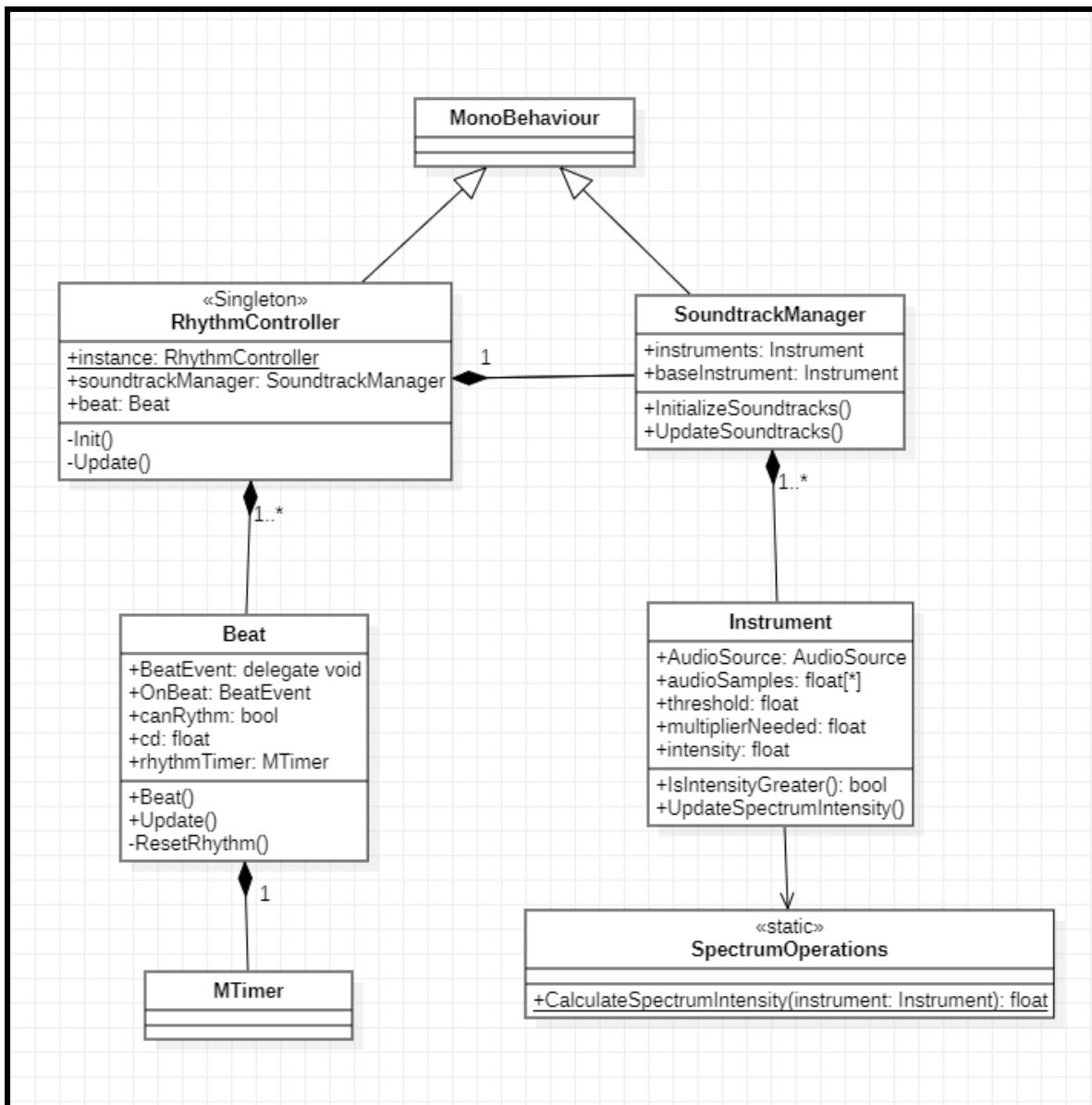


Figura 15. Diagrama de clases de la mecánica de ritmo pautada

#### 4.1.6. Combo

Otra clase que también es necesaria aunque no se relaciona directamente con el ritmo, es la clase de Combo, que se utiliza en el controlador de cada actuación realizada por el jugador. Se enfoca directamente en contener el contador de combos, el máximo de combos para llegar a un combo final y métodos útiles que se puedan utilizar sin tener que tenerlos implementados en el controlador que lo use.

```
// Esta clase Combo define el comportamiento del sistema de combos en un videojuego.
1reference
Epublic class Combo
{
    // Variables de la clase
    private int comboCounter = 0; // contador de combo actual
    private int maxCombo = 0; // contador máximo de combo posible

    // Método que devuelve el contador de combo actual
    0references
    public float GetComboCounter() { return comboCounter; }

    // Método que establece el contador máximo de combo posible
    1reference
    public void SetMaxCombo(int _maxCombo) { maxCombo = _maxCombo; }

    // Método que suma uno al contador de combo actual
    1reference
    public void SumCombo() { comboCounter++; }

    // Método que reinicia el contador de combo actual
    1reference
    public void ComboFailed() { comboCounter = 0; }

    // Método que comprueba si se ha alcanzado el contador máximo de combo posible
    // Si es así, reinicia el contador actual y devuelve true, sino devuelve false
    1reference
    public bool ComboAccomplished()
    {
        if (comboCounter >= maxCombo)
        {
            comboCounter = 0;
            return true;
        }

        return false;
    }
}
```

Figura 16. Clase Combo

#### 4.1.7. Mecánica

Para crear la mecánica primero ha sido necesario tener una canción que poder configurar. Cómo la idea de concebir o buscar una canción por diferentes instrumentos estaba siendo muy complicada. Se ha tenido que recurrir a componentes externos al proyecto para poder conseguir una. Estos componentes son dos personas que habían accedido a formar parte del proyecto.

Por lo que una vez se obtienen canciones divididas por instrumentos se implementan subclases para configurar cada canción (Figura 17).

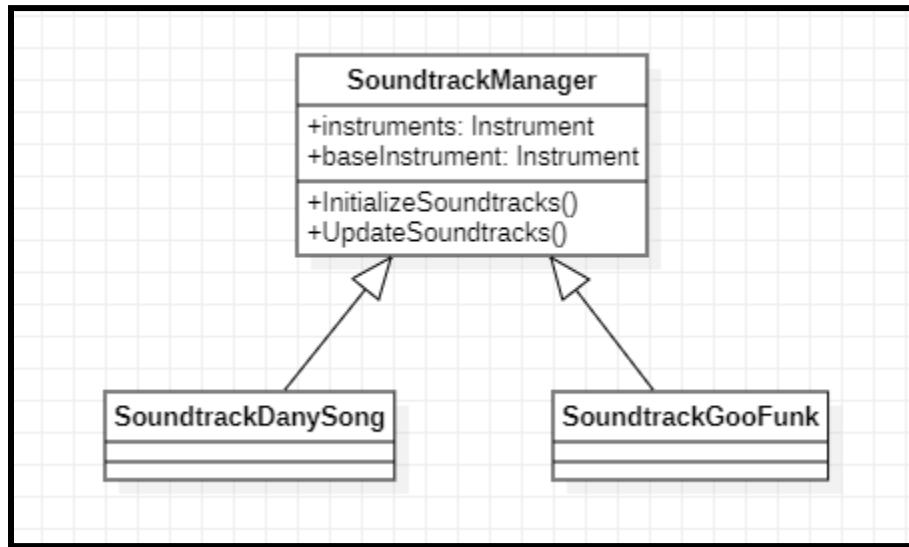


Figura 17. Subclases de las canciones propuestas

##### 4.1.7.1. Prototipado

Cómo se ha explicado al principio, durante la implementación de la mecánica se ha estado trabajando en una escena prototipo. Esta escena contiene elementos visuales muy simples pero que ayudan a entender mejor cómo funciona el beat.

En el momento que ocurre el ritmo de la canción, en las Figuras 18 y 19 se diferencian por el cambio de color de la imagen. En las Figuras 20 y 21 se observa que el jugador recibe feedback cuando este ha presionado un input que coincide con el ritmo de la canción, esto además suma un combo que al llegar a su máximo de combos acumulados hace un combo final (Figura 22), en este caso, una animación que desplaza el jugador hacia delante mientras gira.

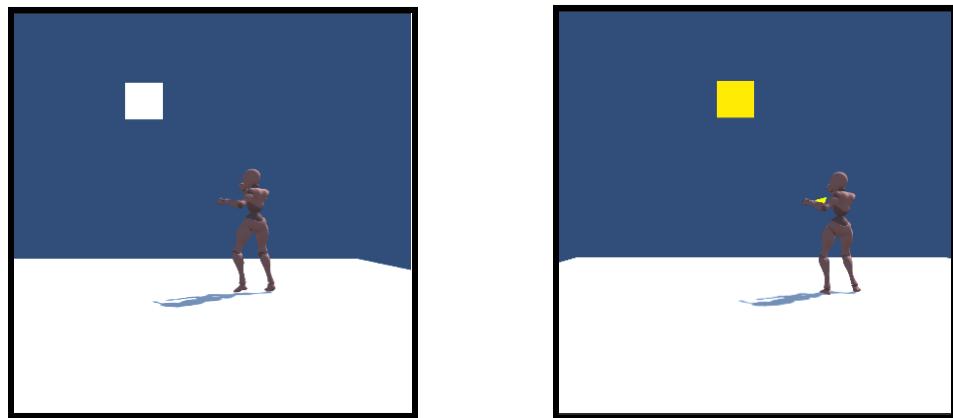


Figura 18 y 19. Momento del ritmo

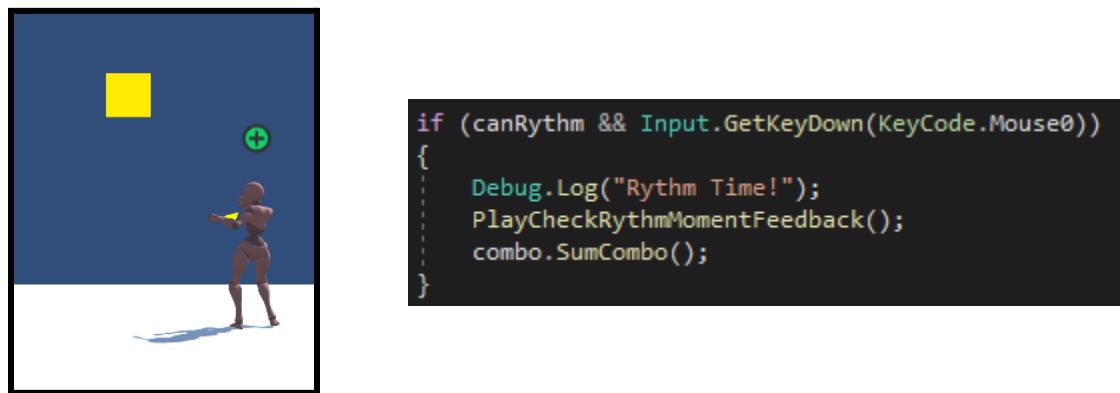


Figura 20 y 21. Momento de coincidencia del jugador con el ritmo

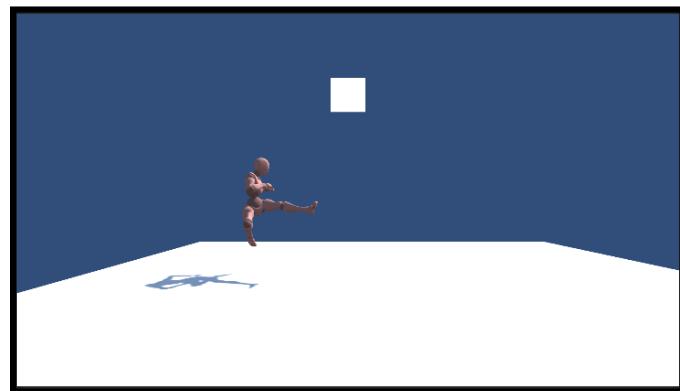


Figura 22. Final Combo

#### 4.1.7.2. Resultado final

El resultado final se compagina con el proyecto de TFG, esto significa que la mecánica de ritmo pautado se utiliza en conjunto con un sistema de combate realizado en el TFG y con un personaje modelado y animados para ese proyecto, en esta parte, lo que se ha hecho es vincular el sistema de combate cómo se explica en el apartado de Beat y se ha añadido feedback aplicando la misma secuencia que en el apartado de Prototipado.

Por tanto, en las Figuras 23 y 24, cuando el jugador puede sincronizarse con el ritmo, la sombra que tiene debajo se hace más grande. Esto se ha hecho para que además de feedback sonoro tenga también feedback visual. En la Figura 25, cuando el jugador realiza un ataque, aparece una nota musical y hay un efecto cromático que hace referencia al sonido y además se aplica un sonido de una tecla de piano que va subiendo el tono cuánto más combea hasta un máximo de 3. Por terminar, el combo final en la Figura 26, dónde se puede observar el cambio de color final que además cuadra con la estética del juego ya que el mundo cambia de color y además se aplica otra imagen de una explosión con un sonido de bombo para que el combo final sea tan satisfactorio cómo se quiere conseguir.



Figura 23 y 24. Momento del ritmo en el proyecto de TFG

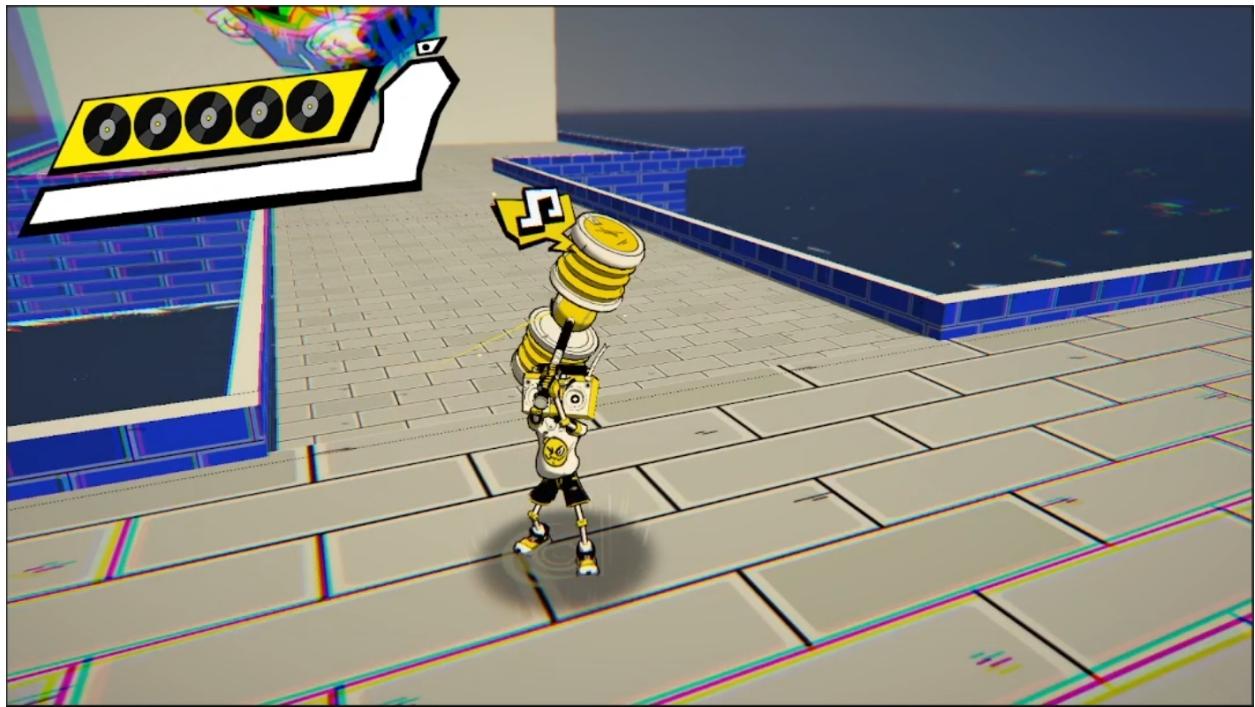


Figura 25. Momento de coincidencia del jugador con el ritmo en el proyecto de TFG



Figura 26.Final Combo en el proyecto de TFG

## 4.2. Ritmo libre

El ritmo libre parte de la programación implementada en la primera parte con el ritmo pautado. Teniendo en cuenta muchas de las cosas ya programadas, cómo la estructura del código ya permite separar los audios por instrumentos, es muy ágil poder reproducir una secuencia libre en la que el jugador sienta que está creando el la composición musical. Cabe destacar que siempre hay una música sonando con la que poder seguir el Beat porque si no existiera este ritmo el jugador podría presionar en cualquier momento.

En esta sección además se utiliza también la lógica de los combos para hacer reproducciones totales sobre la composición musical y se utiliza una secuencia de botones concreta que el jugador puede escoger usarla con la que se consigue sumar más combo.

### 4.2.1. Inputs

Definir una estructura de inputs que pudiera funcionar para otros proyectos ha sido una tarea complicada, ya que según el proyecto se pueden estar usando sistemas de inputs distintos. Por lo cuál se ha optado por utilizar el sistema de inputs por defecto que ofrece Unity y crear unos controles correspondientes a los botones de la derecha de un controlador de mando. (Figura 27)

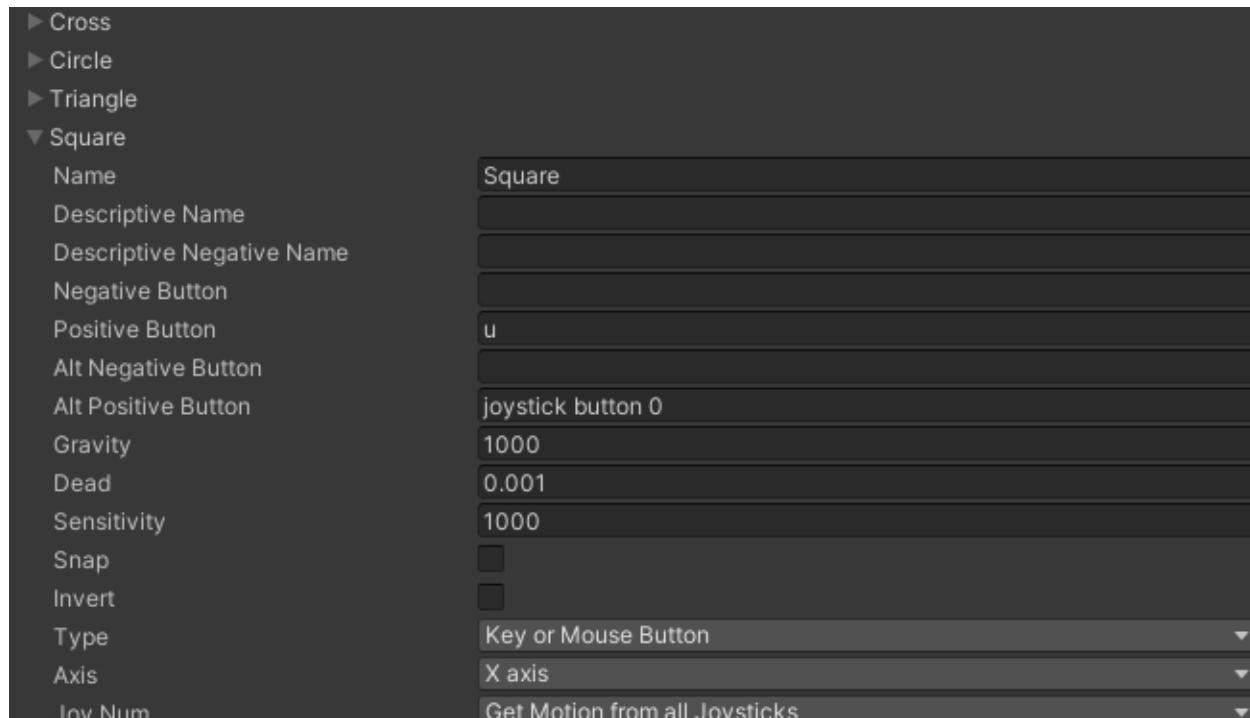


Figura 27. Definición de inputs en el Input Manager

#### 4.2.2. Secuencia de botones

Una vez estos controles están definidos, ahora es el momento de usarlos y comprobar si son correctos y el jugador está siguiendo una combinación concreta o simplemente está pulsando botones al azar. En este caso, el objetivo de tener una combinación concreta es que el jugador crea que puede crear el ritmo si sigue cierta composición musical.

Antes de continuar con la lógica de todo el juego del ritmo libre, hay que destacar cómo se ha diseñado esta secuencia de controles. Teniendo en cuenta que los inputs se están gestionando para un controlador de mando, se define el enumerador de la Figura 29 con la posibilidad de botones que podría seleccionar el jugador. En el caso del proyecto, sólo se han usado los botones de la parte derecha del mando porque los otros no eran necesarios.

```
public enum EControlType { NONE, SQUARE, CROSS, TRIANGLE, CIRCLE, UP, DOWN, RIGHT, LEFT }
```

Figura 28. Enum con todos los botones posibles a presionar.

Después, con una nueva clase llamada ButtonsSequence se definen en el inspector todas las secuencias de botones concretas como se puede observar en la Figura 29.

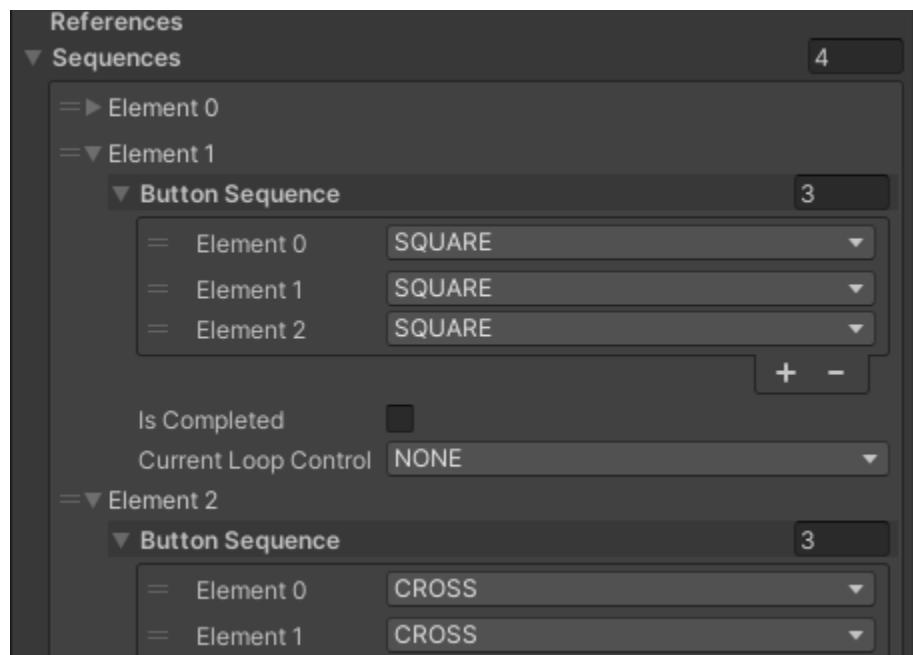


Figura 29. Ejemplo de secuencia de botones.

#### 4.2.3. Lógica

Aunque se ha podido trabajar de manera ágil con los instrumentos porque estaban separados en diferentes AudioSource, ha sido bastante complicado poder crear este juego con todas las variantes de opciones que podía haber.

Primero se crea una nueva clase llamada SimonController que gestiona el juego y otra clase llamada SequenceController que gestiona la secuencia de botones y se asocia con el RhythmController.

#### **4.2.3.1 Primera propuesta**

En una primera propuesta se realiza el simón dice comentado en la sección de diseño que obligaba al jugador a seguir la secuencia de control con mucho margen de error y poca satisfacción en la experiencia del usuario. Aunque cómo Onboarding podía ser beneficioso porque obligaba al jugador a aprenderse el tempo de la base inicial. Sin embargo para este proyecto se buscaba una sensación en la que el jugador estuviera creando la música y esta propuesta no cumplía con las expectativas.

Esta propuesta usaba mucho más la secuencia de controles que la segunda, ya que utilizaba dos estados para gestionar la secuencia de controles concreta que debía seguir el jugador. A continuación, en la Figura 30, se presenta un diagrama con la lógica que utiliza esta primera propuesta.

Se puede observar que cuando le toca al jugador se llevan a cabo muchas comprobaciones para poder gestionar correctamente todo el loop y conseguir un control absoluto para aplicar posteriormente feedback. En este caso, controlar que entre clicks el tiempo de respuesta no sea muy tarde, que el botón sea correcto y que coincida con el ritmo. Además de esto, se puede comprobar que cualquier error de todas estas condiciones llevan al fracaso del juego.

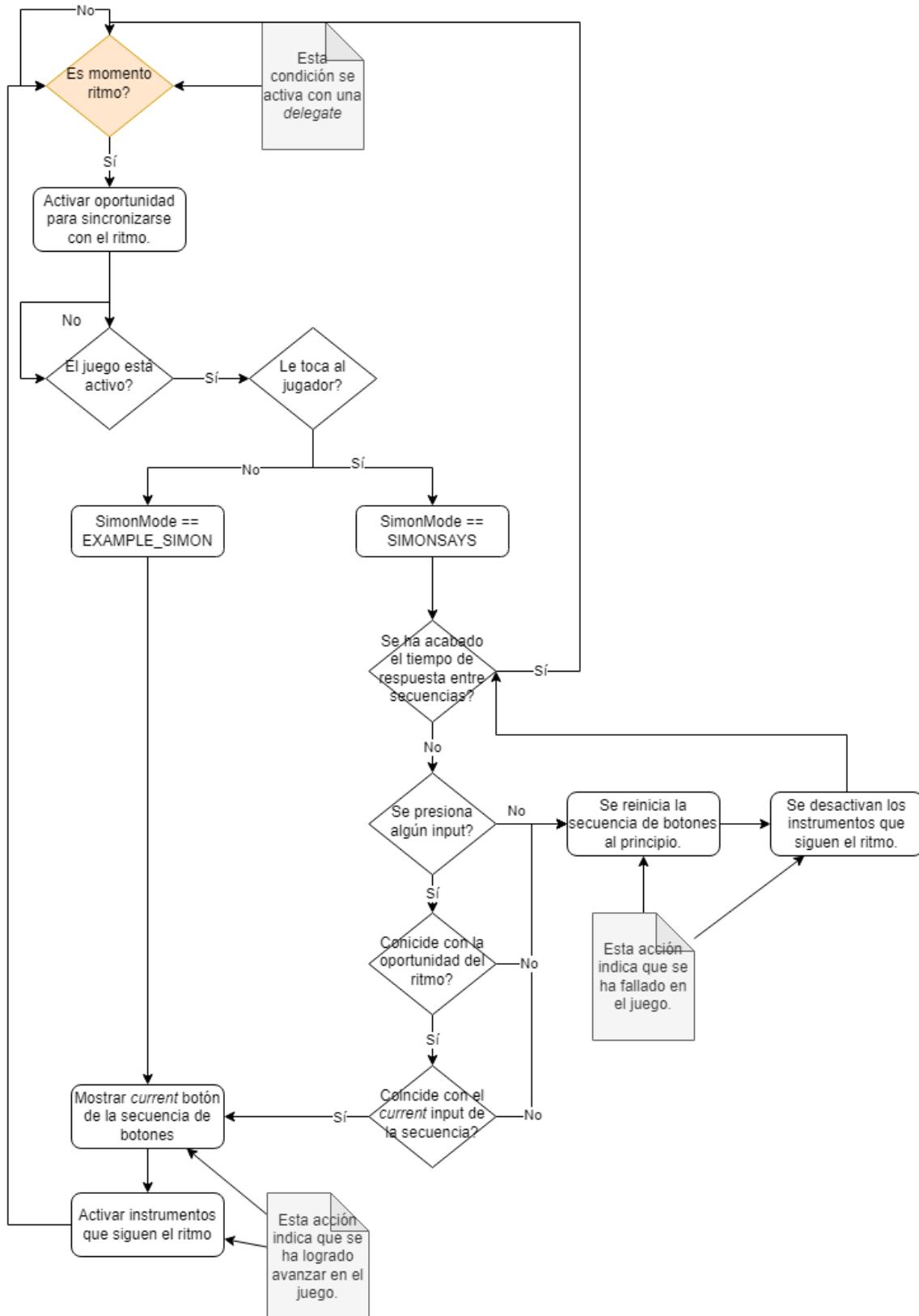


Figura 30. Primera propuesta de la lógica para el Ritmo libre.

#### 4.2.3.2 Segunda propuesta

La segunda propuesta parte de la primera, sin tener tanto en cuenta la vinculación del SequenceController con el SimonController ya que en este caso sólo se utiliza para gestionar el momento en el que el jugador está siguiendo una secuencia concreta. Esto se consigue a partir de observar si el primer control seleccionado corresponde con alguno creado, de esta manera siempre empiezas con los instrumentos de dicha secuencia si empiezas por ese control. A continuación, con los cambios de lógica, se presenta el diagrama actualizado en la Figura 31.

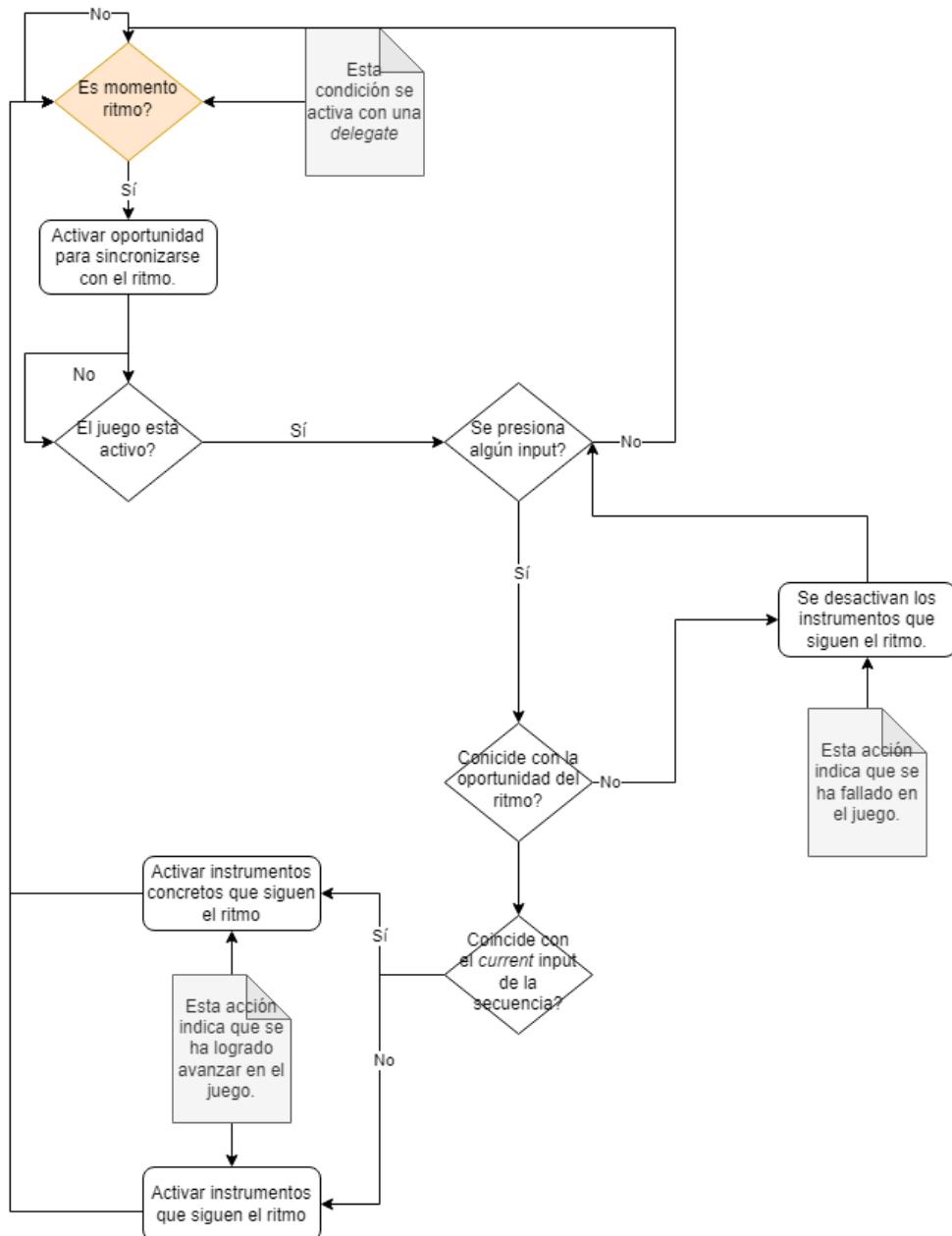


Figura 31. Segunda propuesta de la lógica para el Ritmo libre.

#### 4.2.3.3. Estructura de código

A continuación, se muestra la asociación de las clases según lo explicado con las del ritmo pautado, mostrando así la asociación entre las dos mecánicas. (Figura 32)

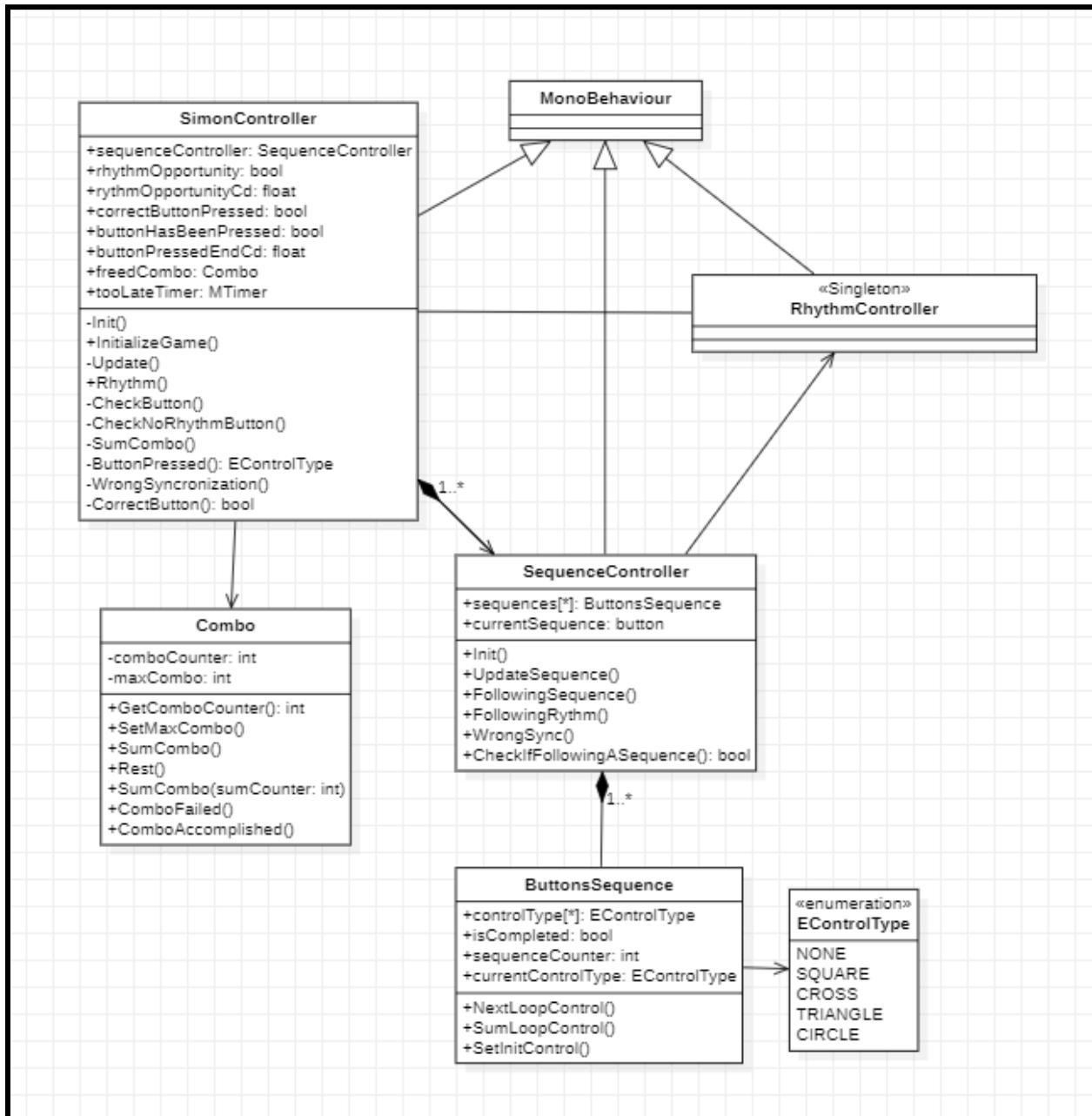


Figura 32. Diagrama de clases de la mecánica de ritmo libre.

## 5. Conclusiones y líneas de futuro

### 5.1. Conclusión general

Desarrollar una mecánica de ritmo ha sido un desafío considerable. Inspirado en la investigación pionera de juegos como Guitar Hero, con sus mecánicas de ritmo y secuencias de botones, al principio se esperaba que fuera un proceso rápido. Sin embargo, a medida que se exploraba el alcance y se consideraban los aspectos de diseño, surgieron complicaciones.

A medida que avanzaba el proyecto y se adquiría conocimiento, se hizo evidente las amplias posibilidades que se presentaban. Cada decisión de diseño y cada iteración proporcionaron una comprensión más profunda de cómo el ritmo puede transformar la experiencia de juego.

A lo largo de este proyecto, se aprendió a valorar la importancia de prestar atención meticulosa a los detalles y realizar iteraciones continuas. Aunque hubo momentos de frustración y complejidad, también se experimentó una gran satisfacción al ver cómo las ideas y conceptos se materializaban en una experiencia de juego musicalmente enriquecedora.

En conclusión, la creación de una mecánica de ritmo ha sido un proceso desafiante pero gratificante. Ha brindado una comprensión más profunda de cómo el ritmo puede transformar la interacción con un juego y ha abierto las puertas a un mundo de posibilidades creativas en el diseño de juegos musicales.

### 5.2. Conclusión de los objetivos

#### 1. Investigar sobre mecánicas de ritmo en Unity

Se han realizado investigaciones exhaustivas tanto en mecánicas de sincronización con el ritmo de la música como en la creación del propio ritmo. Estos aspectos se presentan y se explican detalladamente en cada sección del proyecto, demostrando así su cumplimiento. Gracias a la investigación se ha sido capaz llevar a cabo el desarrollo de la mecánica.

#### 2. Implementar la mecánica de ritmo que se sincroniza con el ritmo de la música

##### a. Implementar el ritmo en mitad de un sistema de pelea.

Este objetivo ha sido exitosamente alcanzado, ya que la mecánica de ritmo se integra de manera complementaria con el sistema de combate del juego de trabajo de final de grado. La sincronización del ritmo no solo agrega un elemento adicional de inmersión y diversión, sino que también tiene un impacto directo en el rendimiento y la estrategia del jugador durante las peleas. Al seguir el ritmo y realizar acciones en el momento preciso, los jugadores pueden desencadenar movimientos especiales, combos poderosos y obtener ventajas tácticas sobre sus oponentes.

**b. Implementar un sistema de combos que funciona con el ritmo de este sistema de pelea.**

Este objetivo ha sido exitosamente alcanzado mediante el uso de combos que generan una retroalimentación del ritmo con el cual el jugador se sincroniza. Además, se ha tenido en cuenta tanto el combo final como cada secuencia de ataques que se sincronizan correctamente con el ritmo.

**3. Implementar la mecánica de ritmo que crea el ritmo de la música**

Aunque si se ha creado la mecánica que crea el ritmo de la música y que está explicado en el apartado de ritmo libre, no se han podido lograr las subtareas correspondientes con el juego al que iba a estar asociado la mecánica. Esto ha ocurrido porque la complejidad del proyecto ha sido más alta de la esperada y porque el juego al que iba a estar vinculado no se ha podido desarrollar en el trabajo de final de grado. Sin embargo, aunque no se ha asociado al sistema de juego deseado, sí que lo está con el sistema de combate de la mecánica de ritmo pautada. Además, al usarlo con la mecánica de combate, se ha acortado un poco el scope del proyecto, lo que ha sido útil para poder pulir las mecánicas.

**4. Implementar un editor de la mecánica de ritmo**

- a. Implementar que se puedan crear en el editor secuencias de botones.**
- b. Hacer el package del editor para poder importarlo a otros proyectos.**

Este objetivo se ha logrado mediante la implementación del script SequenceController, que permite generar secuencias de botones desde el Inspector. Al considerar este objetivo desde el principio, no se tuvo plena conciencia de la amplia utilidad que este editor podría tener. Además, se logró el objetivo adicional de exportar e importar un paquete a otro proyecto, sin que esto genere problemas en el código del proyecto de trabajo de final de grado, lo que demuestra una desvinculación efectiva.

**5. Implementar telemetrías**

Este objetivo no se ha llevado a cabo por la falta de tiempo. Aunque sí que haría falta que se le dedicara tiempo para detectar qué canciones son más fáciles de seguir que otras.

### **5.3. Líneas de futuro**

Existen varias líneas de desarrollo futuro clave que merecen especial atención. En primer lugar, es fundamental seguir trabajando en la sincronización precisa del ritmo con el beat. Aunque la separación del espectro de audio en instrumentos y el cálculo del ritmo individual de cada uno resulta dinámico, este enfoque puede ser costoso por el exceso uso de AudioSource que están sonando al mismo tiempo. Es recomendable realizar una investigación más exhaustiva para explorar opciones adicionales y mejorar la eficiencia en el trabajo con el beat.

Otro aspecto interesante sería la ampliación del feedback en las mecánicas de ritmo. Tanto este feedback como la integración de la mecánica de ritmo con la mecánica de lanzamiento en el proyecto de final de grado añaden un elemento enriquecedor. Sería beneficioso explorar formas de ofrecer una retroalimentación más completa y satisfactoria para los jugadores, mejorando así su experiencia.

Además, se destaca la importancia de implementar telemetría. Dado que no todas las canciones o instrumentos pueden tener un beat adecuado para sincronizar acciones debido a la complejidad de la composición, la incorporación de telemetría sería esencial. Esto permitiría adaptar las mecánicas del juego de manera más inteligente, teniendo en cuenta las características específicas de cada canción y optimizando así la experiencia del jugador.

En conclusión, se recomienda continuar investigando y desarrollando mejores enfoques para trabajar con el beat, mejorar el feedback en las mecánicas de ritmo y realizar la implementación de telemetría. Estas acciones contribuirán a perfeccionar la sincronización, enriquecer la experiencia de juego y ofrecer una adaptación más precisa a la complejidad de las canciones.

## 6. Bibliografia

*Algorithmic Beat Mapping in Unity: Real-time Audio Analysis Using the Unity API.* (2018,

February 27). Medium. Retrieved May 5, 2023, from

<https://medium.com/giant-scam/algorithmic-beat-mapping-in-unity-real-time-audio-analysis-using-the-unity-api-6e9595823ce4>

Tattersall, G. (n.d.). " " - Wiktionary. Retrieved May 5, 2023, from

[https://www.gamedeveloper-com.translate.goog/audio/coding-to-the-beat---under-the-hood-of-a-rhythm-game-in-unity?\\_x\\_tr\\_sl=es&\\_x\\_tr\\_tl=en&\\_x\\_tr\\_hl=es&\\_x\\_tr\\_pto=wap&\\_x\\_tr\\_hist=true#close-modal](https://www.gamedeveloper-com.translate.goog/audio/coding-to-the-beat---under-the-hood-of-a-rhythm-game-in-unity?_x_tr_sl=es&_x_tr_tl=en&_x_tr_hl=es&_x_tr_pto=wap&_x_tr_hist=true#close-modal)

*Guitar Hero Live Controller Issues.* (n.d.). Activision Support. Retrieved May 5, 2023, from

<https://support.activision.com/guitar-hero-live>

*Hi-Fi RUSH.* (n.d.). Bethesda.net. Retrieved May 5, 2023, from

<https://bethesda.net/es-ES/game/hifirush>