

Agradecimientos

Agradecer a mi familia el apoyarme en todo momento durante la realización de este

Proyecto y durante los años que he estado cursando el Grado de Ingeniería Multimedia.

Agradecer también a mi tutor Fidel Aznar Gregori por ayudarme en todas las fases del proyecto y por estar siempre dispuesto a ayudarme cuando lo he necesitado, además de haber sido un gran profesor en su asignatura Fundamentos de los videojuegos.

Contenido

[1. Introducción 5](#_Toc492314859)

[1.1. Situación actual en la industria del videojuego 5](#_Toc492314860)

[2. Motivación 6](#_Toc492314861)

[3. Objetivos 7](#_Toc492314862)

[3.1. Mejorar la capacidad de auto aprendizaje. 7](#_Toc492314863)

[3.2. Aprender a realizar un proyecto de tamaño medio. 8](#_Toc492314864)

[3.3. Unificar diferentes tecnologías para realizar el proyecto. 8](#_Toc492314865)

[3.4. Realizar un videojuego desde cero. 8](#_Toc492314866)

[3.5. Superar los problemas que surgen a lo largo de la creación de un proyecto. 9](#_Toc492314867)

[4. Diseño y especificación 9](#_Toc492314868)

[4.1. Resumen 9](#_Toc492314869)

[4.2. Objetivo del juego 9](#_Toc492314870)

[4.3. Diseño 10](#_Toc492314871)

[5. Mecánicas 10](#_Toc492314872)

[5.1. Sonidos como mecánica 10](#_Toc492314873)

[5.2. Caminar 11](#_Toc492314874)

[5.3. Palmada 11](#_Toc492314875)

[5.4. Sigilo 11](#_Toc492314876)

[5.5. Mecanismos y puertas 11](#_Toc492314877)

[5.6. Rocas 12](#_Toc492314878)

[5.7. Agua 12](#_Toc492314879)

[5.8. Agujero Negro 13](#_Toc492314880)

[6. Objetos 13](#_Toc492314881)

[6.1. Físicos 13](#_Toc492314882)

[6.1.1. Personaje 13](#_Toc492314883)

[6.1.2. Paredes 14](#_Toc492314884)

[6.1.3. Onda sonora 15](#_Toc492314885)

[6.1.4. Enemigo 15](#_Toc492314886)

[6.1.5. Roca 16](#_Toc492314887)

[6.1.6. Agujero Negro 16](#_Toc492314888)

[6.2. Sensores 17](#_Toc492314889)

[6.2.1. Pinchos 17](#_Toc492314890)

[6.2.2. Agua 18](#_Toc492314891)

[6.2.3. Mecanismo y puerta 18](#_Toc492314892)

[6.2.4. Puerta Salida 19](#_Toc492314893)

[7. Niveles 19](#_Toc492314894)

[7.1. Intro y Fin 19](#_Toc492314895)

[7.2. Blind 21](#_Toc492314896)

[7.3. Scape 22](#_Toc492314897)

[7.4. Death 23](#_Toc492314898)

[7.5. Fear 23](#_Toc492314899)

[7.6. Hunt 24](#_Toc492314900)

[7.7. Calm 25](#_Toc492314901)

[7.8. Strange 26](#_Toc492314902)

[7.9. Give Up 26](#_Toc492314903)

[7.10. Atraction 27](#_Toc492314904)

[7.11. End Is Nigh 27](#_Toc492314905)

[8. Sonido 28](#_Toc492314906)

[8.1. Sonido 2D 28](#_Toc492314907)

[8.2. Sonido Visual 28](#_Toc492314908)

[9. Herramientas 28](#_Toc492314909)

[9.1. Visual Studio 28](#_Toc492314910)

[9.2. SFML 29](#_Toc492314911)

[9.3. Box2D 29](#_Toc492314912)

[9.4. Tiled 29](#_Toc492314913)

[9.5. TinyXML 30](#_Toc492314914)

[9.6. Gimp 30](#_Toc492314915)

[9.7. Audacity 31](#_Toc492314916)

[9.8. Github y SourceTree 31](#_Toc492314917)

[10. Físicas 32](#_Toc492314918)

[10.1. Mascaras de colisiones 32](#_Toc492314919)

[11. Motor Grafico 33](#_Toc492314920)

[11.1. VertexArray y TriangleStrips 33](#_Toc492314921)

[12. Patrones de diseño 34](#_Toc492314922)

[12.1. RAII 34](#_Toc492314923)

[12.2. Fachada 35](#_Toc492314924)

[12.3. Abstract Factory 35](#_Toc492314925)

[12.4. Singleton 36](#_Toc492314926)

[13. Lenguaje 36](#_Toc492314927)

[13.1. C++ 36](#_Toc492314928)

[13.2. C++ 14 37](#_Toc492314929)

# Introducción

## Situación actual en la industria del videojuego

Actualmente el videojuego ya no es considerado perjudicial, como se creía antes, sino que ha pasado a ser uno de los mayores entretenimientos a nivel mundial, cuya industria ha superado en ingresos incluso al cine. El mundo de los videojuegos ha ido entrando en la vida cotidiana poco a poco, y ha venido para quedarse.

En los últimos años la industria del videojuego ha cambiado de manera drástica, entrando en juego los desarrolladores independientes y la pequeña empresa, a base de desarrollos cortos, con mecánicas simples y apuestas arriesgadas a precios reducidos para poder competir con los juegos tripe A (juegos con grandes desarrollos ej: GTA, Assassins Creed) .

La puesta en escena de los desarrollos independientes, cuyos productos han revolucionado la industria, han forzado a las grandes empresas a competir de manera más agresiva y a ser cada día más competitivos y arriesgados en sus apuestas.

Dejando un panorama actual de un altísimo nivel artístico, la historia embebida en cada videojuego ha tomado el papel protagonista en muchos casos, donde los videojuegos más valorados se consideran obras de arte, donde los directores de cine se pelean por desarrollar guiones para videojuegos y donde los mejores actores y actrices obtienen papeles importantes en los videojuegos.

# Motivación

Ante un creciente y abarrotado mercado de videojuegos, cada día más grande y con más opciones (unas buenas y la mayoría de poca calidad) se crea la necesidad de diseñar juegos que sean distintivos, que tengan algo que los haga únicos y que les permita destacar sobre los demás y obtener visibilidad en esa marea de nuevos títulos, remasterizaciones y remakes.

Con esta premisa se buscaba una apuesta arriesgada, pero con posibilidades, y basándome en mis preferencias de títulos que generan tensión con algo de terror psicológico unido a puzles comencé a buscar juegos que cumplieran estos requisitos.

Después de una larga búsqueda encontré un juego que a pesar de una simpleza casi llevada al extremo y sin grandes pretensiones, con una mecánica simple pero muy original y a mi parecer acertada, este título usaba el sonido como medio de información de tal manera que sin generar sonido o ruido no obtienes información del entorno que te rodea, te aporta diferentes maneras de generar sonido a lo largo del juego, genera tensión mediante los cuidados apartados gráficos y sonoros y cada mapa te hace querer más y más.

El título en cuestión se llama “Dark echo”, está disponible en varias plataformas (ps4, Xbox, pc, Android y iOs) es un juego de terror y puzles en el que nos encontramos una serie de mapas en los cuales tendremos que ir avanzando generando ruido para poder visualizar el entorno y pasando una serie de pruebas y puzles hasta encontrar la salida.

Cuando encontré este título decidí enfocar este proyecto como un reto de programación en el cual intentar realizar un videojuego ya existente con mis propias herramientas y conocimientos obtenidos durante la carrera, y no tanto un reto de diseño de videojuegos, que sería la realización de un título original. Tome está decisión basándome en mi motivación de mejorar mi programación, aprender nuevos paradigmas y afrontar el reto de crear un juego desde cero, con todos los problemas que esto supone.

Mas adelante desglosaremos cada parte del videojuego y veremos las asociaciones de estos con cada una de las asignaturas impartidas durante el grado de Ing. Multimedia.

A pesar de la decisión de desarrollar un título ya existente si que quería darle un toque personal, un añadido original que aportara jugabilidad extra, más adelante veremos los detalles.

# Objetivos

## Mejorar la capacidad de auto aprendizaje.

Siempre ha sido muy importante para mí el ser capaz de buscar, filtrar y contrastar información sin la necesidad de una persona o personas que estén ahí para ayudarte, esta capacidad te ayuda a aprender sin necesidad de profesores y a mejorar de una manera más estimulante y enriquecedora, para esta tarea Ingeniería Multimedia ha sido un gran apoyo, pues es la filosofía de aprendizaje que más se utiliza durante el grado, los profesores te dan las pinceladas necesarias para que seas capaz de buscar información y saber utilizarla para cada uno de los proyectos y tareas que se proponen.

Estoy muy agradecido por haberme ayudado a mejorar en este sentido y por ello quiero seguir mejorándolo durante el proceso de este trabajo.

## Aprender a realizar un proyecto de tamaño medio.

A lo largo de todo el grado se han realizado una gran cantidad de proyectos de todos los tamaños, pequeños individuales, medios en grupo y el ABP que es un proyecto grande y en equipo.

La diferencia de todo esto con el TFG es que este es un proyecto de tamaño medio que se realiza de manera individual, por lo tanto, eres el responsable de cada uno de los apartados del proyecto, y esto no es sencillo.

En cuanto a este apartado me gustaría mejorar y aprender a llevar de manera efectiva y productiva el desarrollo del proyecto, y por supuesto acabarlo.

## Unificar diferentes tecnologías para realizar el proyecto.

Una tarea importante a la hora de realizar un videojuego, o cualquier aplicación que necesite o esté compuesta de varias partes claramente diferenciadas, es la de tomar la decisión de que partes vas a implementar desde cero y que partes vas a usar herramientas de terceros, como librerías o aplicaciones, todo esto lleva consigo un trabajo grande de cohesión entre las partes de las que consta el producto final, donde tienes que estudiar de que manera encajar las piezas para que sea usable, rápido y eficiente y a la vez comprensible para el programador, diseñador y desarrollador.

En este trabajo se ha optado por utilizar una serie de patrones de desarrollo muy conocidos para poder llevar a cabo esta tarea, como puede ser el patrón fachada, donde se desarrolla una capa de clases extra entre cada parte del videojuego, para evitar un acoplamiento entre ellas.

## Realizar un videojuego desde cero.

El desarrollo de un videojuego desde 0 no es una tarea sencilla, para esto hay que tener en cuenta muchas variables y organizar una serie de etapas en el desarrollo, empezando por el diseño del juego, durante el cual se organiza y se estructura el corazón del videojuego donde quedan establecidas las mecánicas básicas de este, su forma, su aspecto y su historia sin llegar a la programación.

La segunda etapa ya permite ponerse manos a la obra y trata de desarrollar una versión básica de la idea principal, sin detenerse en los detalles. Esta primera versión nos dará información de si estábamos en lo cierto con respecto a las mecánicas, si es un juego divertido y jugable.

Una vez hemos probado diferentes versiones de prueba y estamos conformes al funcionamiento básico es la hora de dar forma a los detalles en la última etapa de desarrollo.

Después de todo el desarrollo lo que queda es testear y corregir los posibles fallos de la aplicación.

## Superar los problemas que surgen a lo largo de la creación de un proyecto.

Cuando comienzas un proyecto de este tipo es todo felicidad y alegría, comienzas a programar, a ver como poco a poco el juego va cogiendo forma y color, pero durante el desarrollo se llegan a plantear una serie de complejos problemas que has de resolver.

Entre estos problemas están las mil decisiones de desarrollo que debes tomar, problemas de programación que cuestan días de encontrar, algoritmos a diseñar y un sinfín de dudas acerca del lenguaje y las librerías que estas usando.

# Diseño y especificación

## Resumen

Echoes es un juego de terror que utiliza una formula sencilla:. Consiste en recorrer una serie de mapas formados por salas totalmente a oscuras, para ello tendremos que guiarnos únicamente por el sonido de nuestros pasos, buscando la salida sin morir en el intento. Suena fácil, pero estaremos rodeados de peligros y cada mapa será más complicado que el anterior.

## Objetivo del juego

El objetivo principal del juego es recorrer una serie de mapas repletos de trampas y mecanismos ocultos en los que abra que resolver puzles para poder encontrar la salida.

Para poder movernos por el mapa tendremos que realizar ruido de diferentes maneras, mediante pasos, piedras o palmadas. Este ruido nos permitirá visualizar el entorno y saber si tenemos algún peligro cerca.

El juego está compuesto por 10 mapas dispuestos de manera creciente en dificultad, cada uno de ellos es un complejo laberinto de salas, cada mapa presenta una nueva mecánica, que sumada a las anteriores da opción a nuevos y más complejos puzles.

## Diseño

Solamente seremos capaces de situar visualmente nuestros pies, que serán unas huellas blancas en la pantalla totalmente negra, esto será así mientras no nos movamos, porque el sonido de nuestros pasos, o cualquier otro sonido, generará una serie de ondas visuales que nos permitirán vislumbrar la forma del mapa y los peligros que nos rodean, rebotando en las paredes y cambiando de color dependiendo de la zona por la que pasen.

Estas ondas sonoras serán más intensas y duraderas cuanto más fuerte sea el sonido producido, y a lo largo de los mapas se irán añadiendo mecánicas para generar más sonido, o menos, según nos convenga, lo que da una profundidad y posibilidad de creación de puzles infinita.

Además el proyecto está planteado de forma que podría considerarse una continuación de la formación académica, ya que muchas de las herramientas utilizadas no han sido utilizadas durante el Grado y requerirán un período de aprendizaje.



# Mecánicas

## Sonidos como mecánica

El sonido será la principal fuente de información de lo que sucede en el mundo que nos rodea, este será representado con líneas rectas cuya velocidad, intensidad y duración dependerán del la fuerza y volumen del sonido que la produce. Esto dará como resultado una pantalla cargada de información a través de estas ondas que rebotando contra los límites del mapa nos darán información de hacia donde tenemos que ir y que nos rodea, ya que cuando una de estas ondas pase sobre los diferentes objetos del juego (palancas, pinchos, etc) se colorearan durante el periodo en el que están situadas sobre el objeto de un color característico del objeto, por ejemplo, la salida de cada mapa se coloreara de verde.

## Caminar

El simple hecho de caminar producirá ruido a causa de nuestros pasos, estos generarán una buena cantidad de ondas sonoras que nos harán de guía en nuestro camino, podremos movernos en todo momento en 8 direcciones.

## Palmada

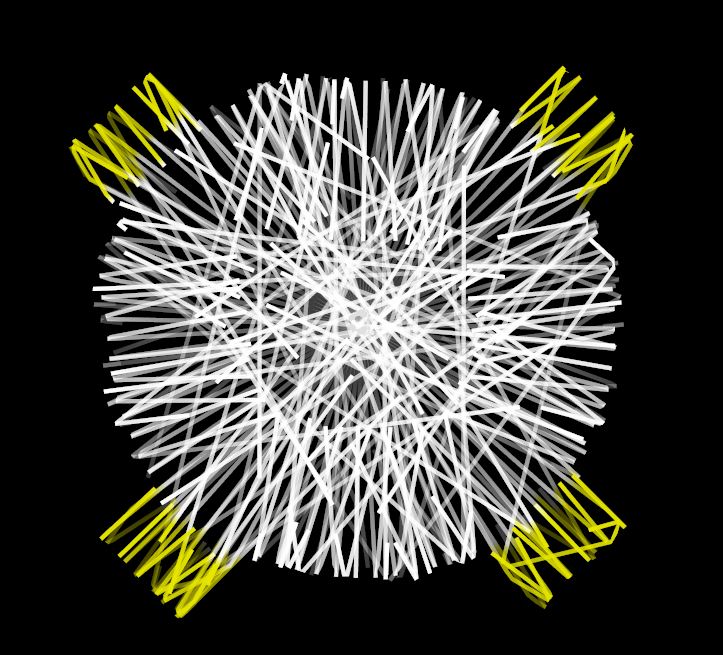
La palmada es el recurso más importante que tenemos en el juego, mediante la cual podemos hacer ruido a diferentes niveles de sonoridad mediante el mantenimiento de la tecla “Espacio”, cuanto más tiempo tardemos en soltar la tecla más sonido generaremos.

## Sigilo

A veces caminar es demasiado peligroso si nos acechan enemigos, para esto tenemos la capacidad de avanzar sigilosamente y sin hacer casi ruido, lo malo es eso mismo, que sin ruido no vemos hacia donde estamos yendo.

## Mecanismos y puertas

En nuestro camino nos encontraremos con una serie de mecanismos (pintan las ondas de amarillo) que al pisarlos nos darán acceso a nuevas zonas antes inaccesibles.



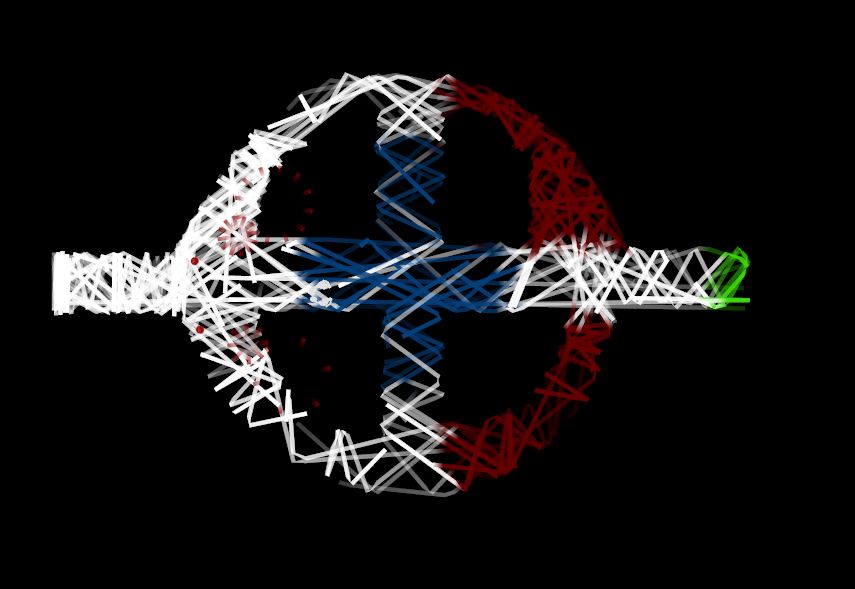
#### Sonido atravesando mecanismos que se pintan en amarillo.

## Rocas

Cuando una zona está vigilada por un terrible enemigo tendremos la posibilidad de lanzar una piedra para distraerle, puesto que siempre van a olfatear el lugar del cual les llega un ruido.

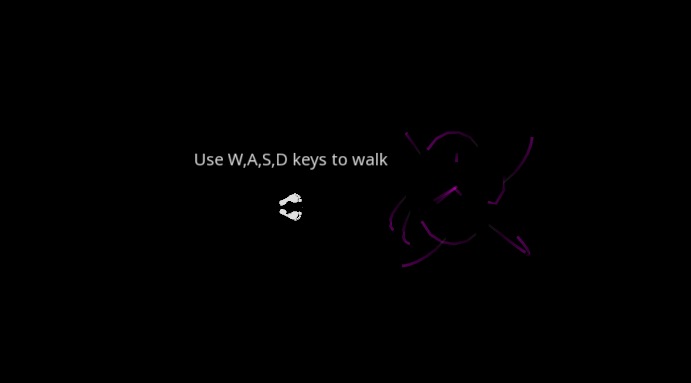
## Agua

El agua es un enemigo “silencioso”, no nos matará, pero nos hará avanzar más despacio y hacer más ruido, tened mucho cuidado con ella.



#### Diferentes zonas, entre ellas el agua en azul.

## Agujero Negro

Los agujeros negros son el único componente del juego que no se encuentra en el titulo original, este objeto está basado en uno de mis títulos favoritos “Binding of Isaac”, su funcionamiento es simple, atrae a todo lo que se mueve, las ondas del sonido se verán modificadas y atraídas hacia él, al igual que el personaje, esto es muy peligroso porque siempre los veremos acompañados de una buena dosis de pinchos a su alrededor.

#### Representación grafica ingame del agujero negro

# Objetos

Existen 2 tipos de objetos en el juego que atienden a las diferentes necesidades que han surgido en el desarrollo del videojuego, el primer tipo son los objetos que tienen componente física, es decir, se desplazan y colisionan con otros objetos físicos. El segundo tipo son los objetos sensores, estos se caracterizan por generar eventos cuando los objetos físicos entran en contacto con ellos.

## Físicos

### Personaje

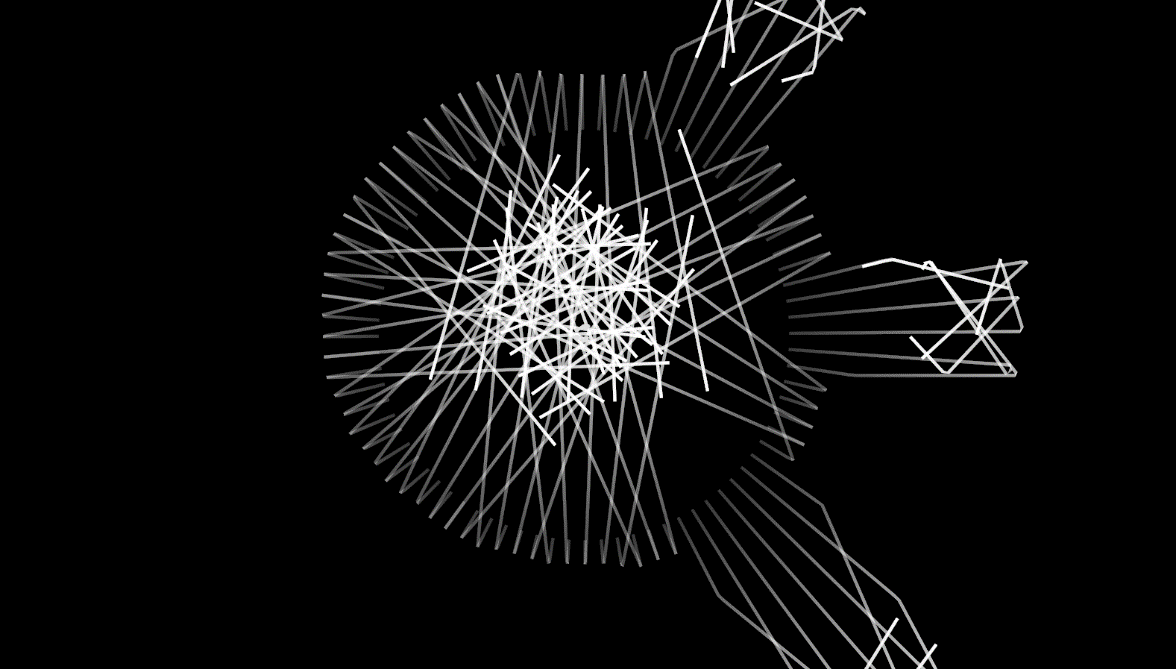
El personaje principal se desplaza por el mapa mediante eventos por teclado teniendo 8 direcciones posibles según la combinación de teclas pulsadas, este colisionará con el resto de objetos y generará ondas sonoras con su desplazamiento. La colisión de este con el resto de objetos del juego se traducirá en diferentes eventos según el tipo de objeto, estos eventos los veremos en la definición de cada uno de ellos.



#### Pies representando al personaje (ingame)

### Paredes

Objeto rectangular con colisión sencilla, no tiene más interacción que la de modificar la dirección del movimiento de los objetos que colisionan con ella. Esta estructura es la que define la forma del mapa, siendo un elemento muy importante.

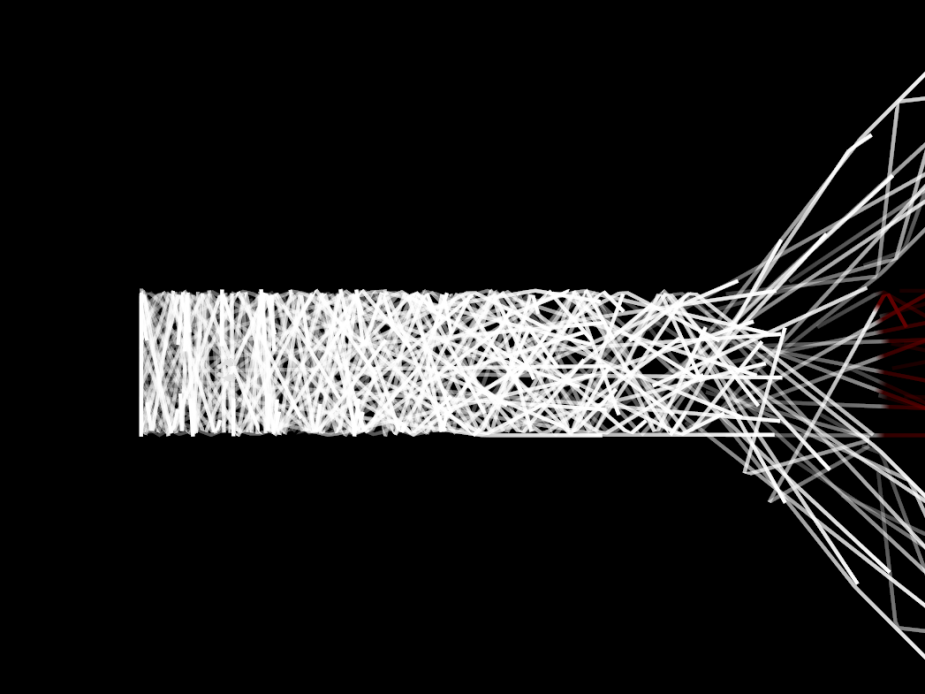


#### Una palmada del personaje que da forma al nivel a través de las ondas sonoras

### Onda sonora

La onda sonora es el elemento más importante del juego, sin ellas no habría jugabilidad, estas son un objeto físico sin rozamiento que se desplazan por el mapa colisionando con el resto de objetos y reaccionando a ellos de diferentes maneras, en su punto de partida son blancas, tienen una velocidad y dirección iniciales desde el origen del sonido y se desplazaran en línea recta (si no hay nada que modifique su dirección) hasta colisionar o hasta acabar su tiempo de vida, estas generan una estela a su paso que sigue su misma dirección e irá dibujando el mapa.

Cuando la onda choca con un objeto solido esta se desvía de su trayectoria, y cuando choca con un objeto no solido cambia de color dependiendo del tipo de objeto por el cual este pasando, ej: Azul del agua, rojo de los pinchos o verde de la puerta.



#### Imagen de múltiples ondas sonoras rebotando en el mapa

### Enemigo

El enemigo es un objeto que se mantiene estático hasta que recibe la colisión de una onda sonora, a partir de ese momento comenzara a moverse en dirección al origen de la onda. En todo momento genera un sonido que visualmente se representa como pequeñas ondas saliendo de su centro.



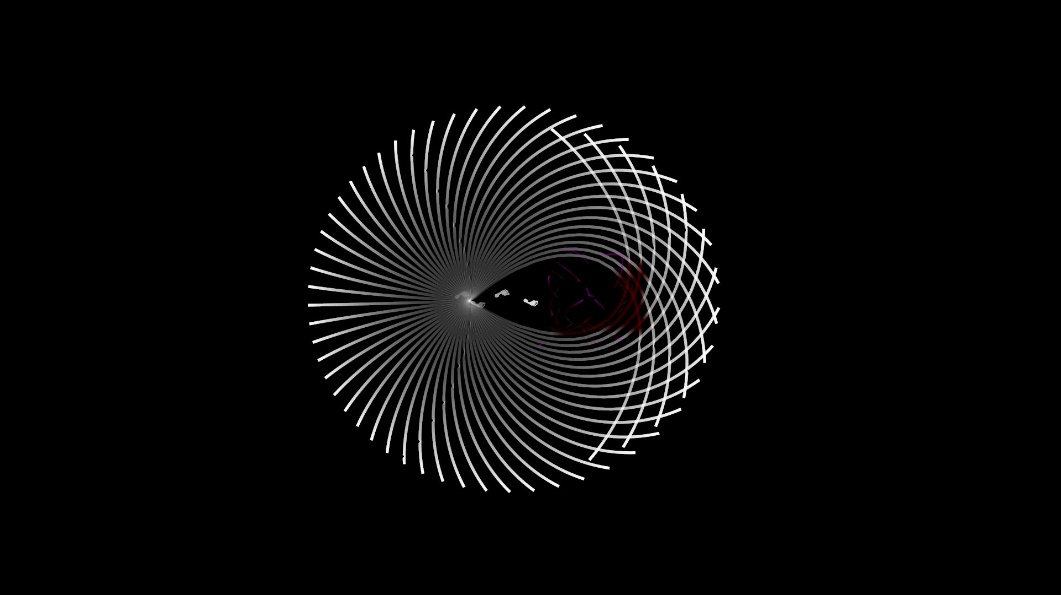
#### Imagen de un enemigo (ondas rojas) acercándose al personaje

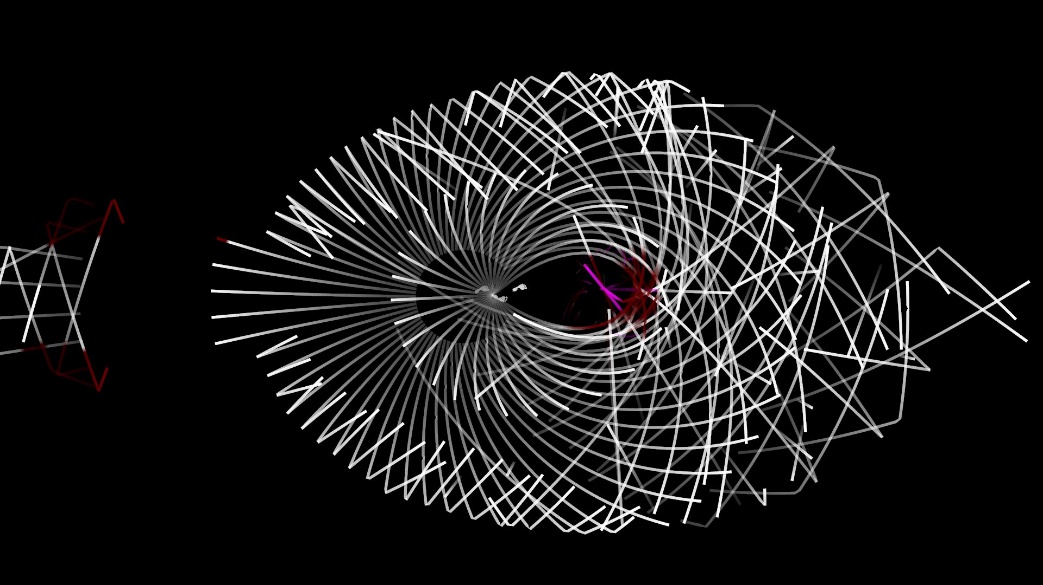
### Roca

La piedra es un recurso que tiene el personaje, pudiendo realizar la acción de lanzar una roca. Esta se dibujará como una línea recta que avanza por el mapa hasta encontrar una pared con la que colisionar. Una vez colisiona genera una serie de ondas sonoras en ese punto del mapa.

### Agujero Negro

El agujero negro es un objeto especial que modifica físicamente a los demás, creando una atracción hacia el del personaje y las ondas sonoras, este generará una fuerza constante en todo momento desde el objeto hacia él, siendo mayor cuanto más cerca se encuentre de este.





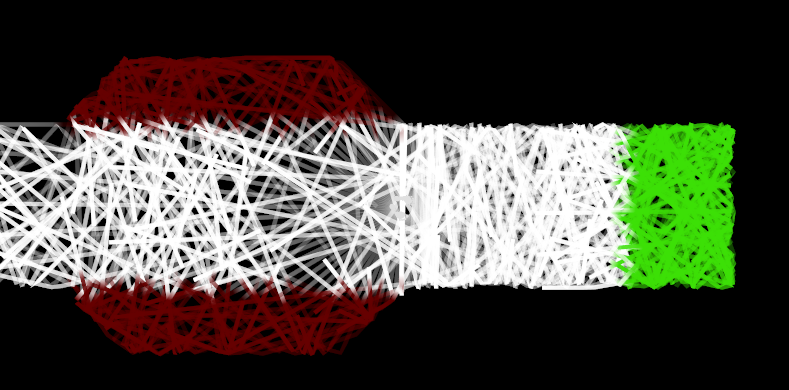
#### Varias imágenes de como el agujero negro atrae a las ondas sonoras.

## Sensores

### Pinchos

Los pinchos son un objeto rectangular cuya mascara de colisiones le permite generar un evento de entrada en colisión y fin de colisión con los objetos de tipo personaje y onda sonora, el evento de colisión con onda sonora permite que esta sea pintada durante esta colisión de color rojo, y la colisión con el personaje es mortal para este.

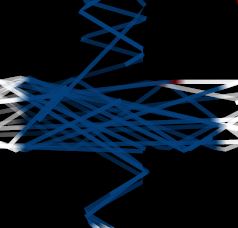
Estas colisiones no tienen un efecto físico real, el movimiento y dirección de los objetos que colisionan con ellos no se ven modificados.



#### Imagen del sonido atravesando pinchos (zonas rojas)

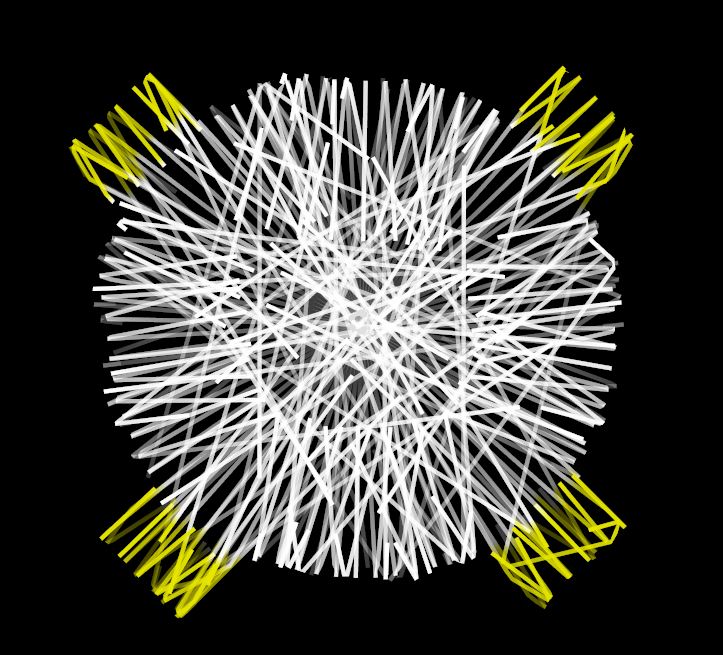
### Agua

El agua es un objeto rectangular que sin tener propiedades físicas actúa como generador de eventos ante la colisión de esta con el personaje, haciéndolo más lento y ruidoso.



#### Representación del agua

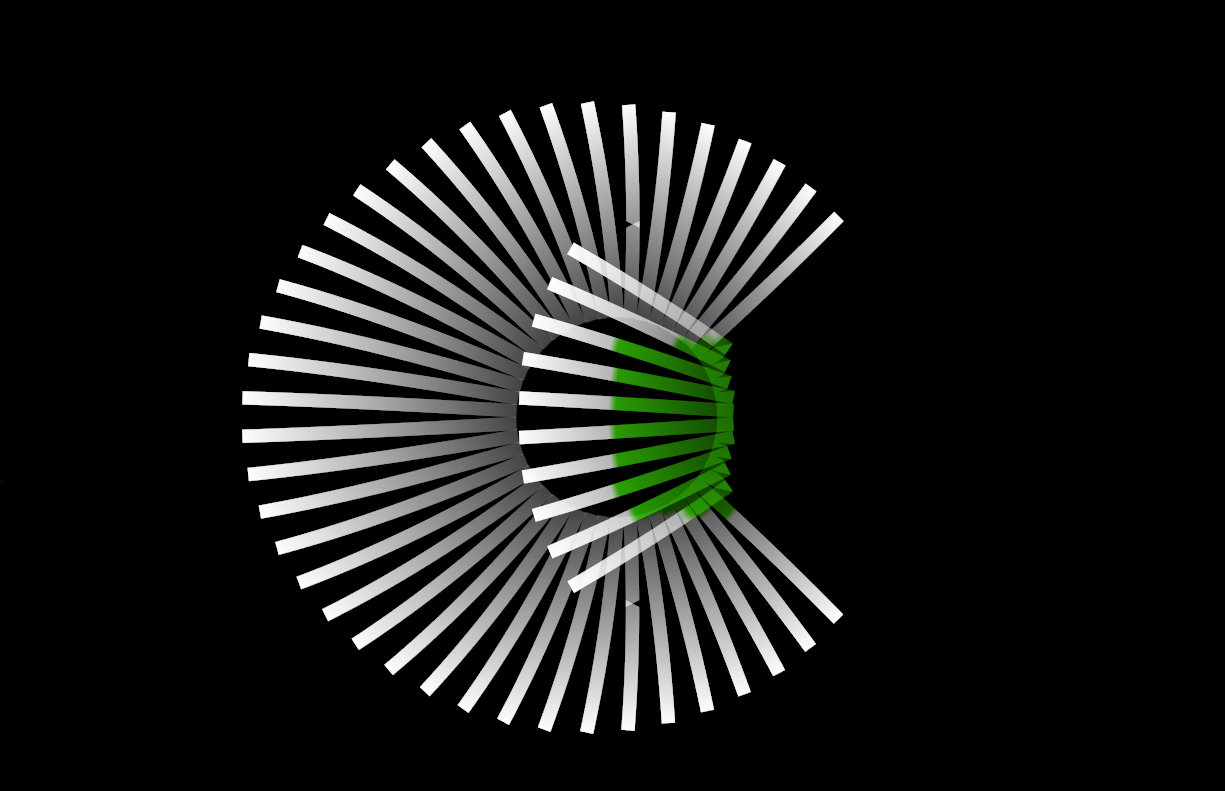
### Mecanismo y puerta

El mecanismo y puerta son 2 objetos que interactúan entre si, de tal manera que, si el personaje entra en contacto con el área definida por el mecanismo, se activa y abre la puerta asociada a él.

#### Representación de los mecanismos

### Puerta Salida

La puerta de salida es el sensor que representa el final del mapa, pinta de color verde a las ondas sonoras.

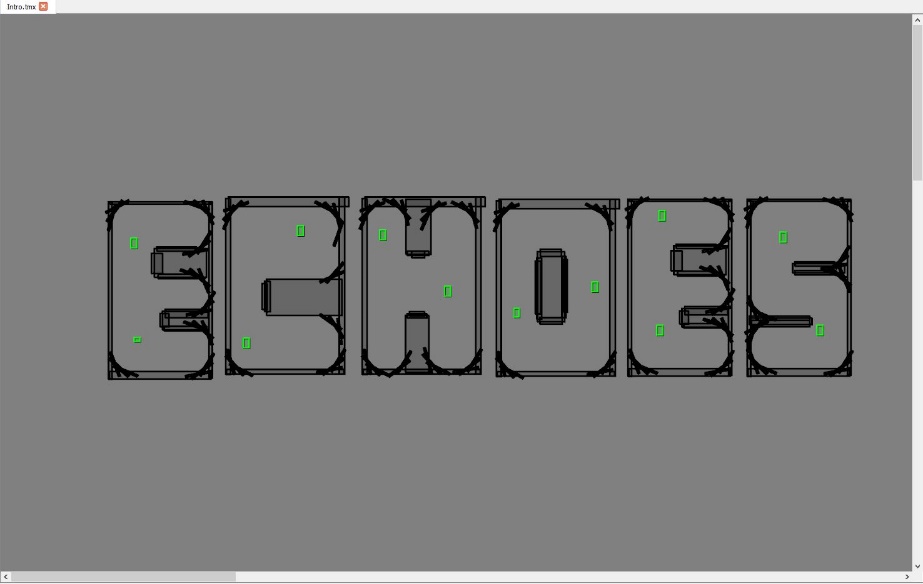
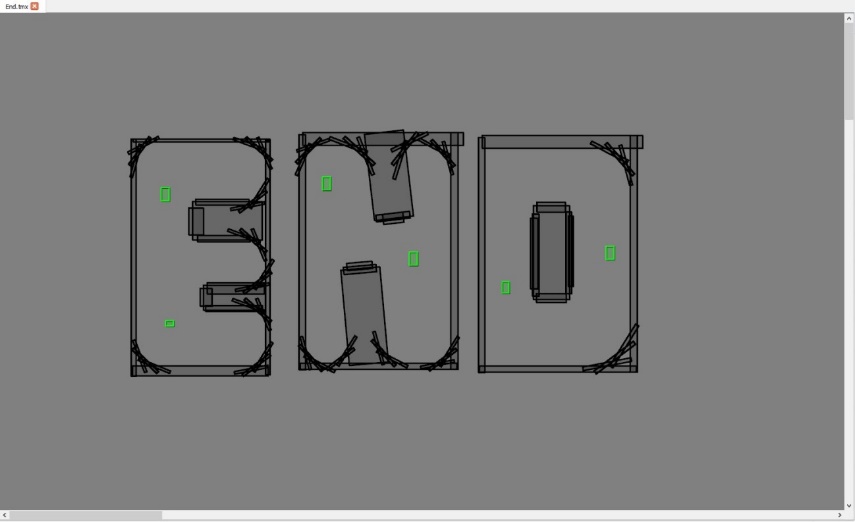


#### En color verde, sonido atravesando la salida del mapa

# Niveles

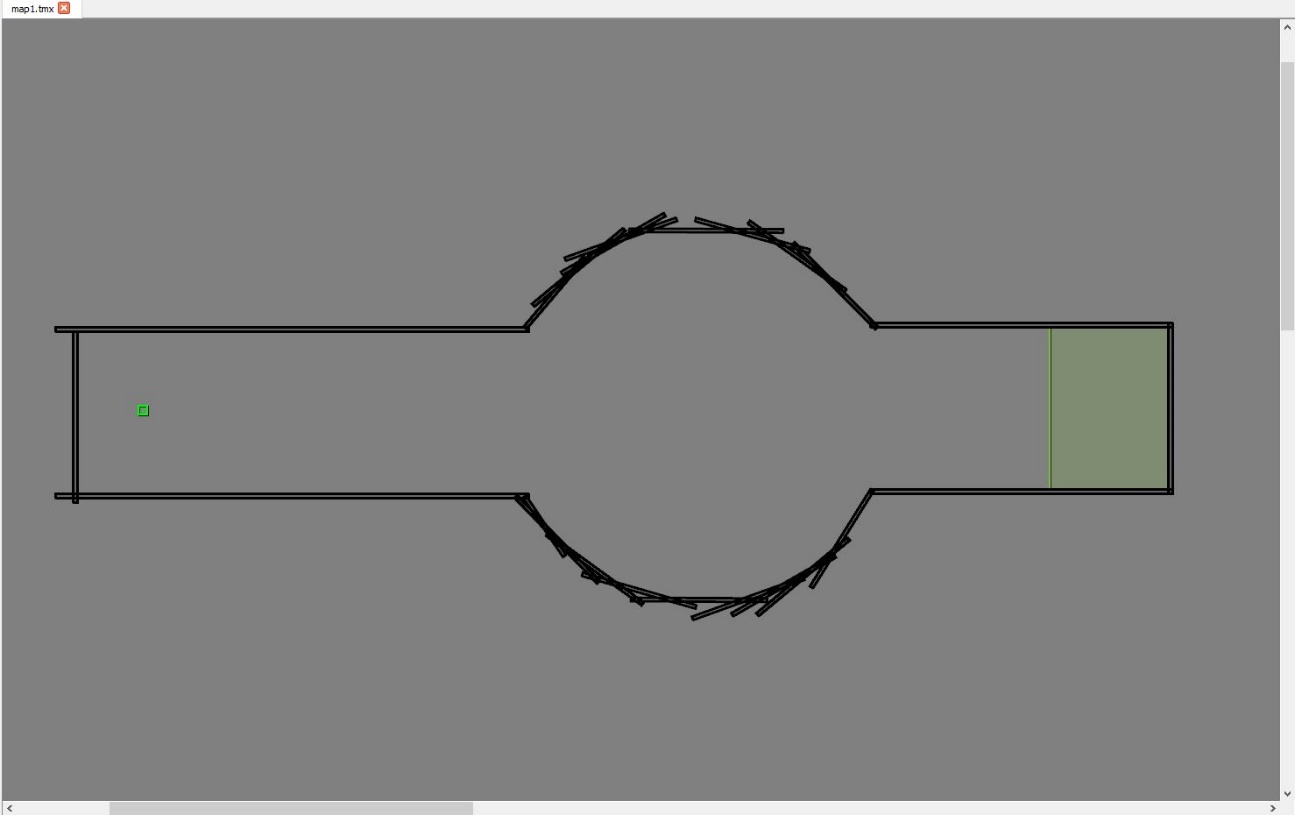
## C:\Users\jpuerto\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Intro2.pngIntro y Fin





#### Las 2 primeras son los niveles ingame de inicio y fin del juego, el resto son las maquetas realizadas con tiled

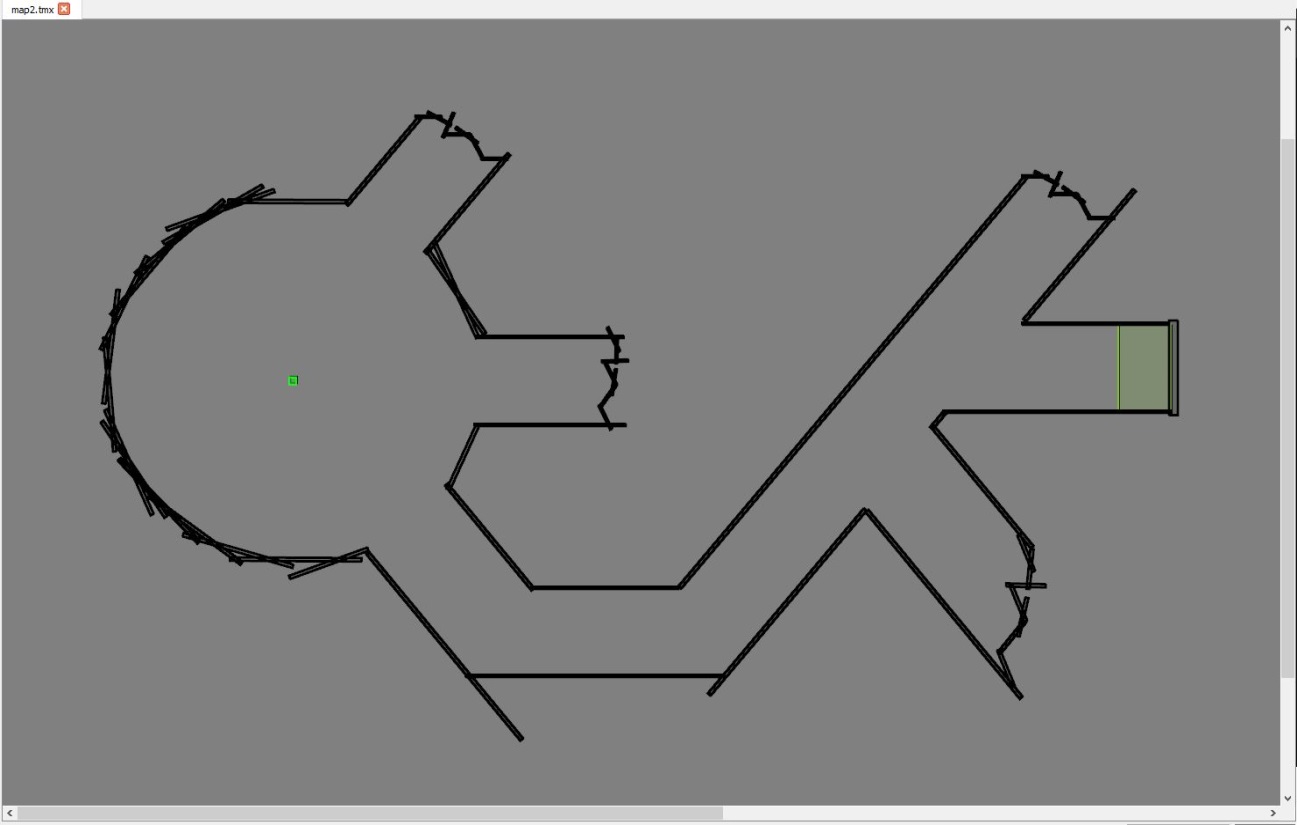
## Blind



#### Maqueta del mapa “blind”

En este primer nivel se trata de acostumbrarse al movimiento, en él se nos indicaran las teclas de movimiento y aprenderemos que sin sonido no vemos nada.

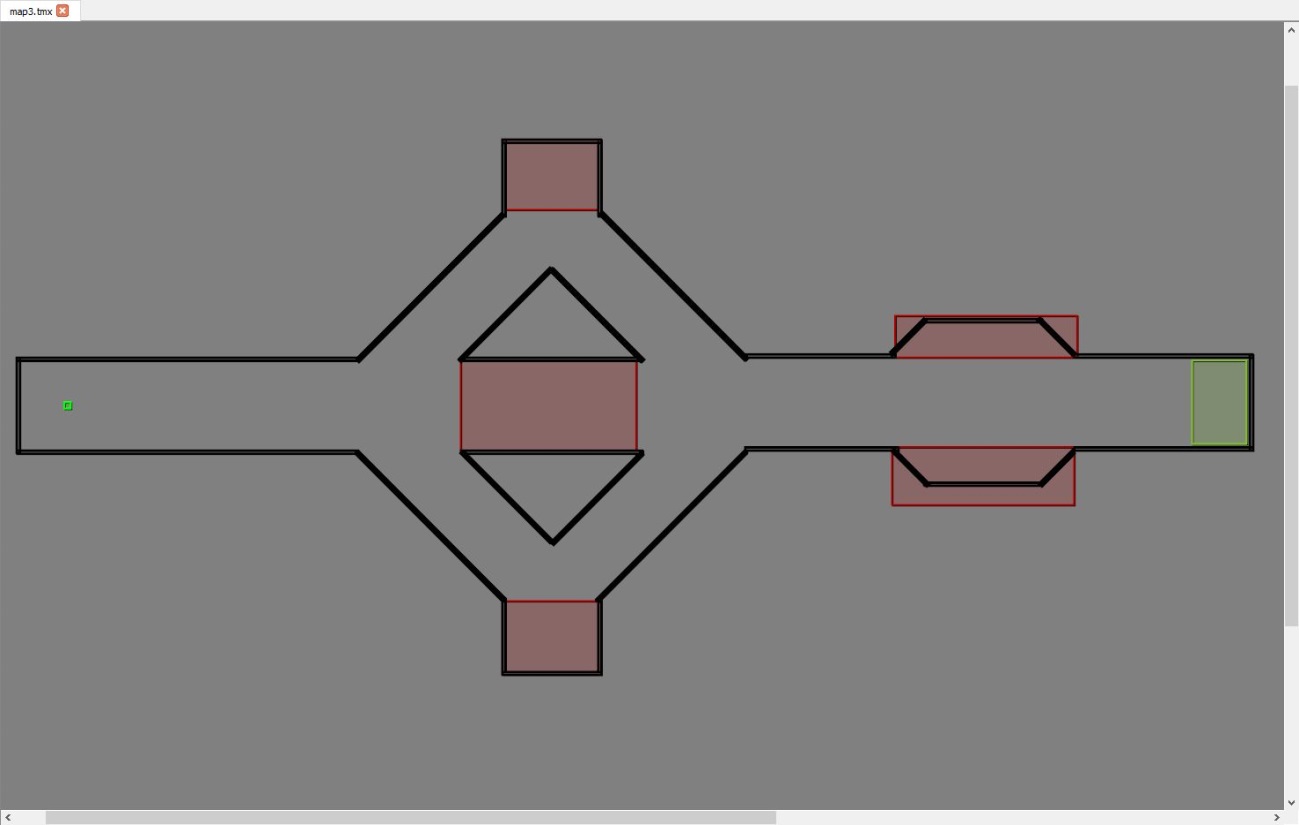
## Scape



#### Maqueta del mapa “scape”

Scape ya nos propone que demos palmadas para generar más sonido del que somos capaces de hacer solo con nuestros pasos, en el aprenderemos que manteniendo la barra espaciadora generaremos mucho más ruido. A su vez, este mapa ya es algo más complejo que el anterior, siendo necesario recorrer un pequeño laberinto para llegar a la salida.

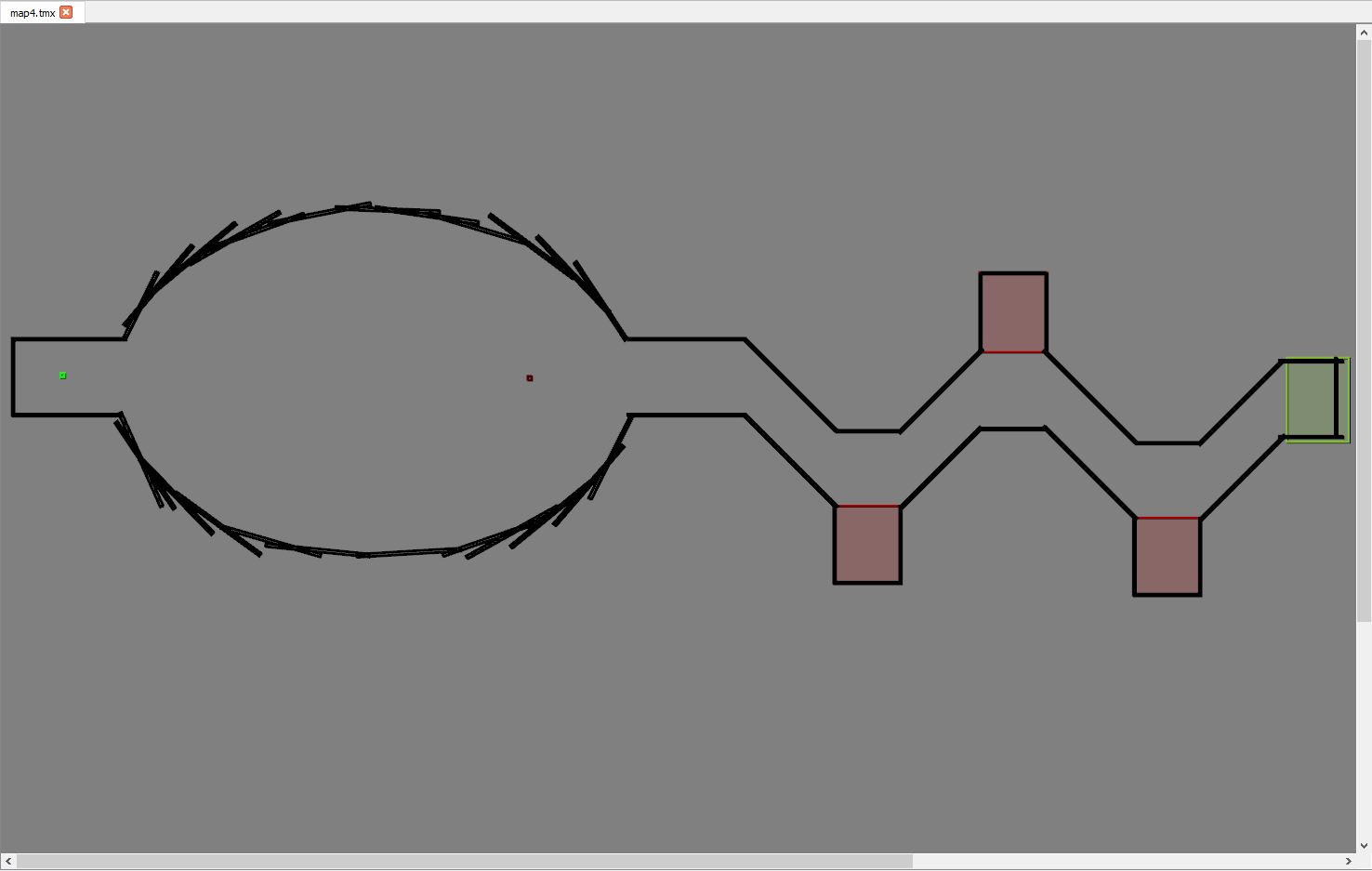
## Death



#### Maqueta del mapa “death”

En este mapa nos encontramos por primera vez un elemento peligroso, los pinchos, los cuales nos matarán si pisamos en ellos, el mapa está pensado para obligarte a dar un pequeño rodeo para no morir en los pinchos.

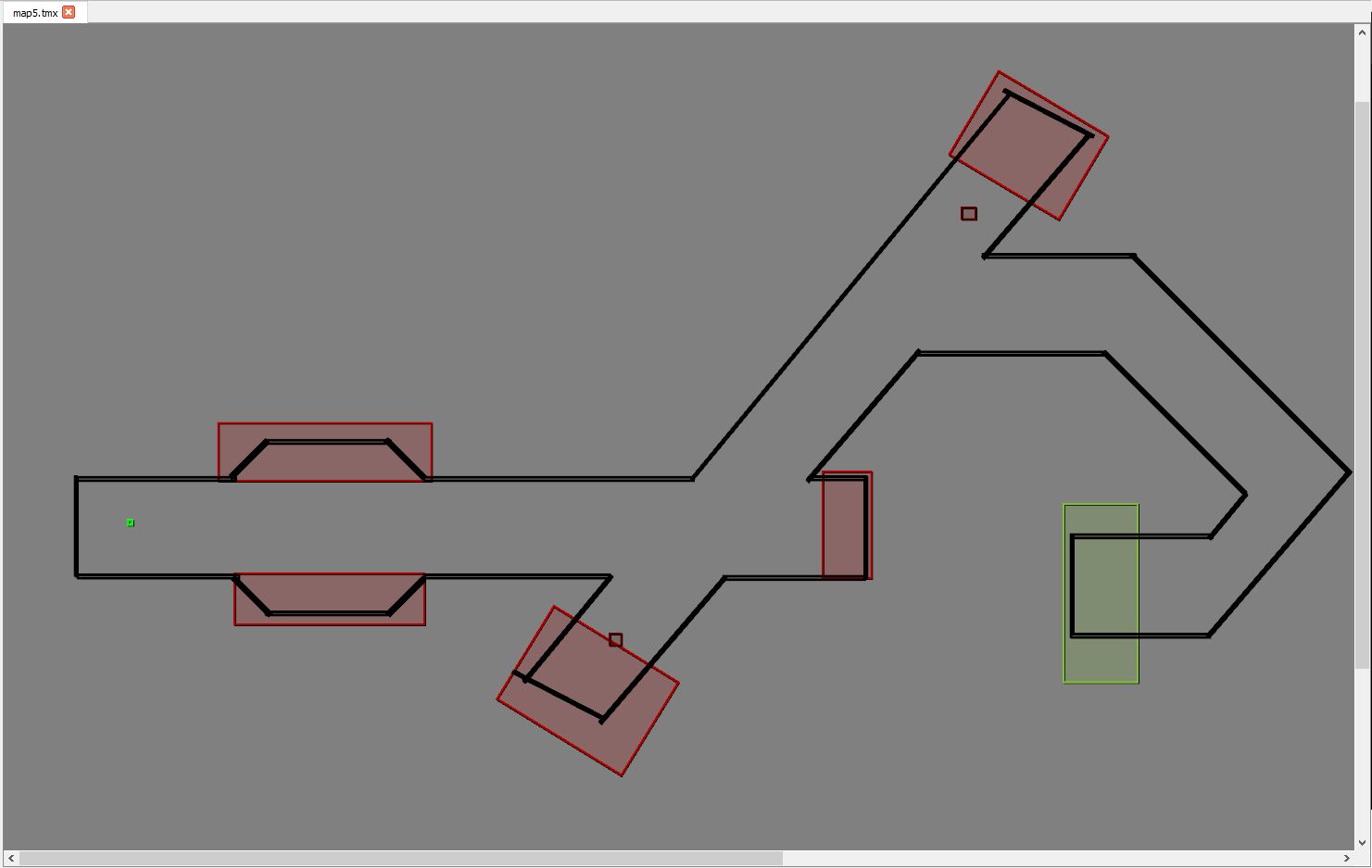
## Fear



#### Maqueta del mapa “fear”

Esta es la primera vez que vemos los peligrosos enemigos de Echoes, este mapa nos obligará a pensar una manera de escapar de ellos y sortear los obstáculos para llegar a la meta.

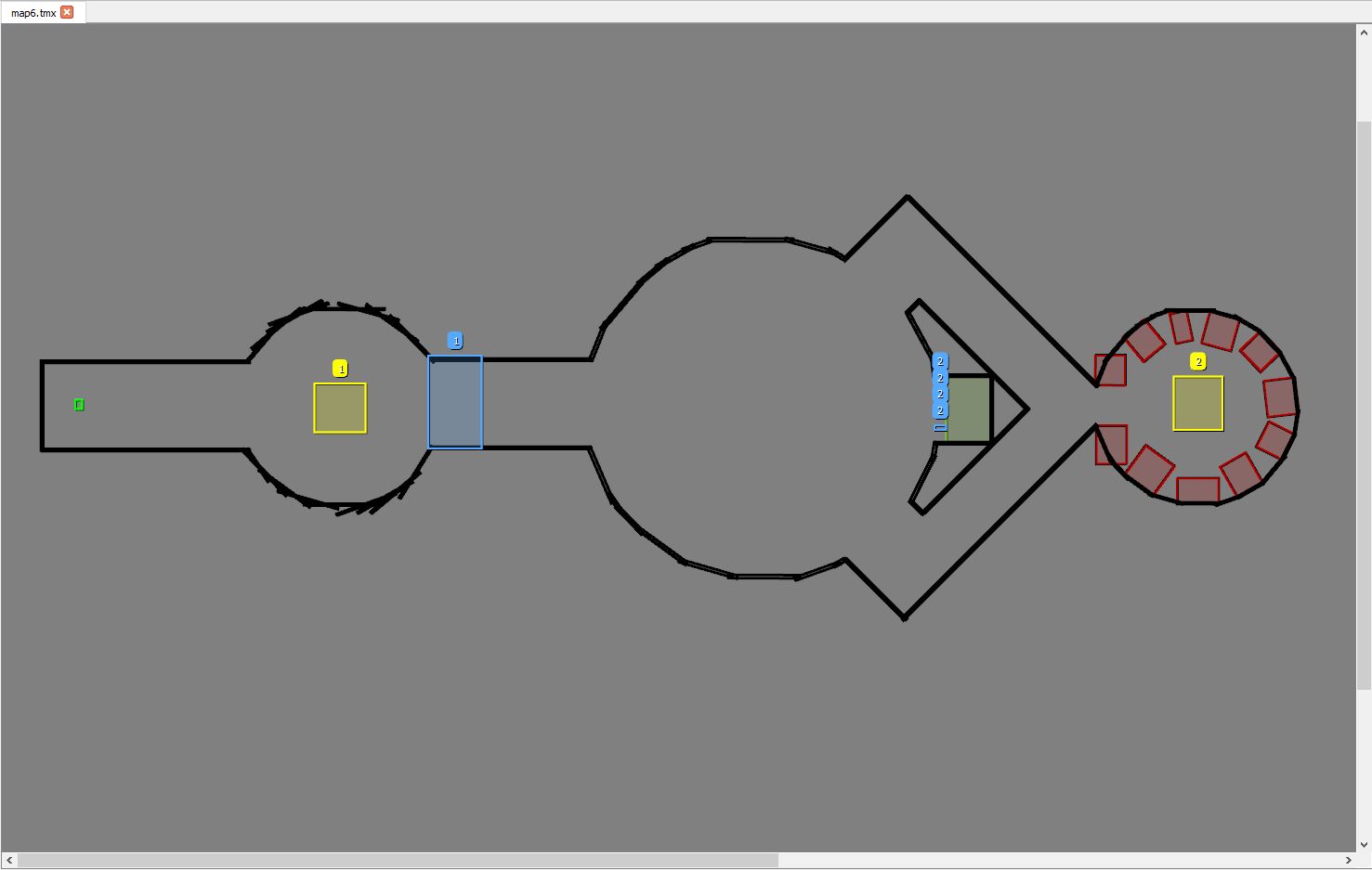
## Hunt



#### Maqueta del mapa “hunt”

Este mapa es una versión mas complicada del anterior, donde tendremos que correr delante del enemigo para salvar la vida.

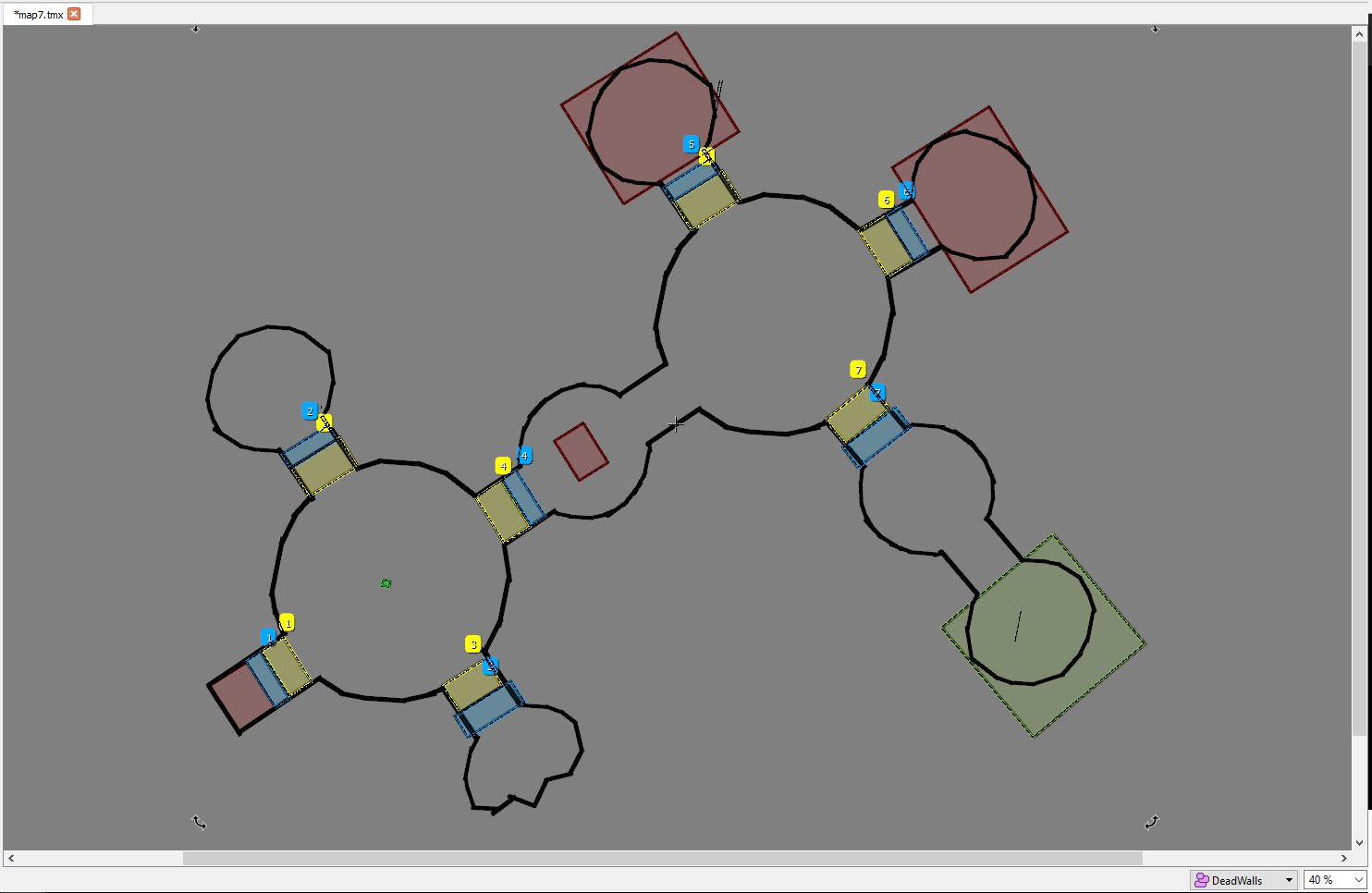
## Calm



#### Maqueta del mapa “cakm”

Nuevo nivel de dificultad, los mecanismos, en este nivel tendremos que aprender a usar las mecánicas de los mecanismos para poder superarlo.

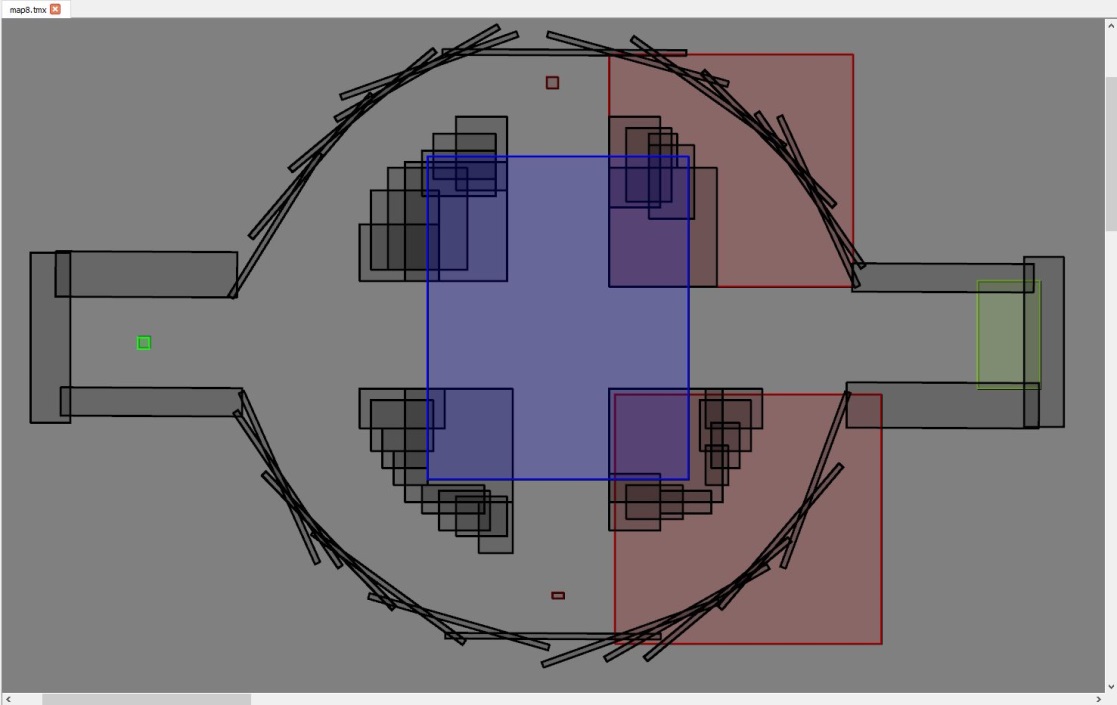
## Strange



Bienvenidos a una verdadera pesadilla de laberinto, Strange es el mapa donde no te quieres quedar encerrado, junta todo lo conocido anteriormente para intentar atraparte.

#### Maqueta del mapa “strange”

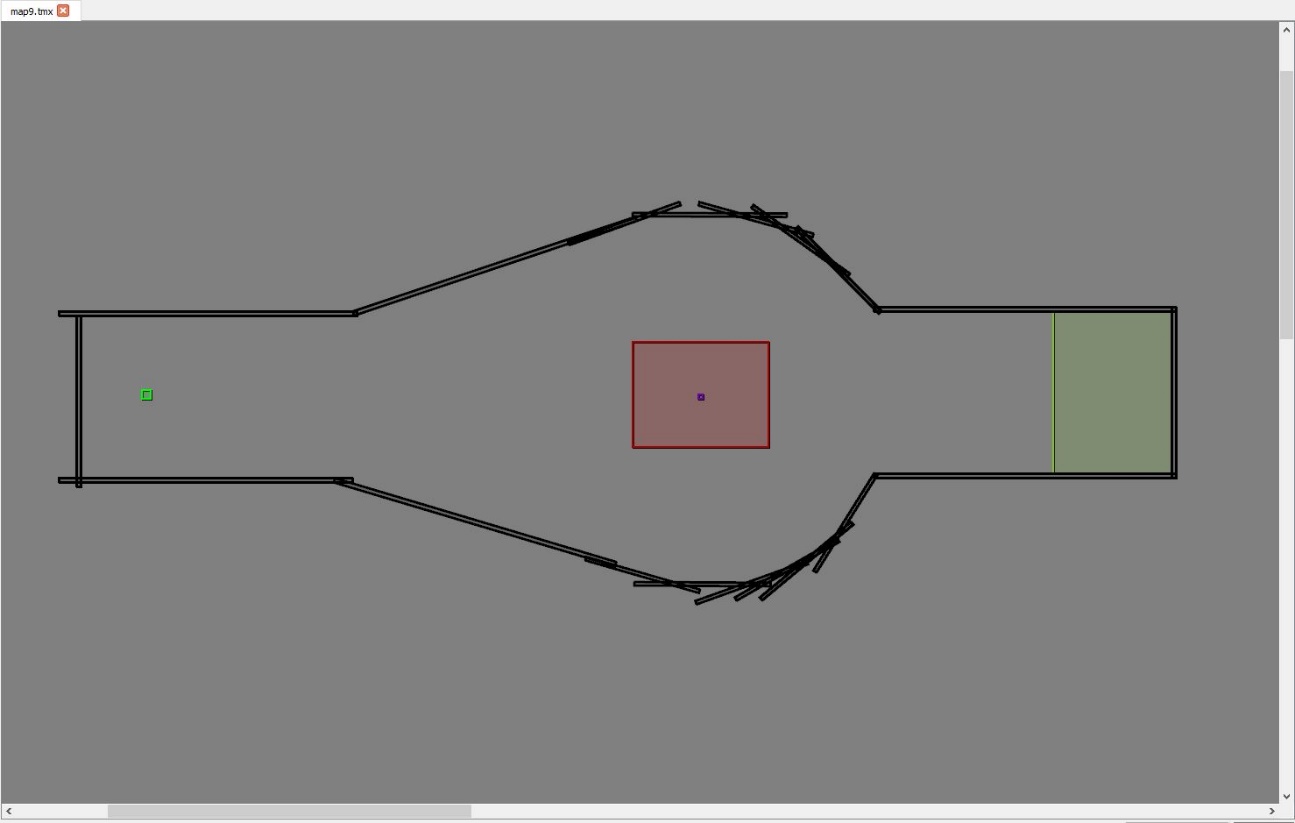
## Give Up



#### Maqueta del mapa “Give up”

En este mapa encontramos por primera vez el agua, donde aremos mas ruido.

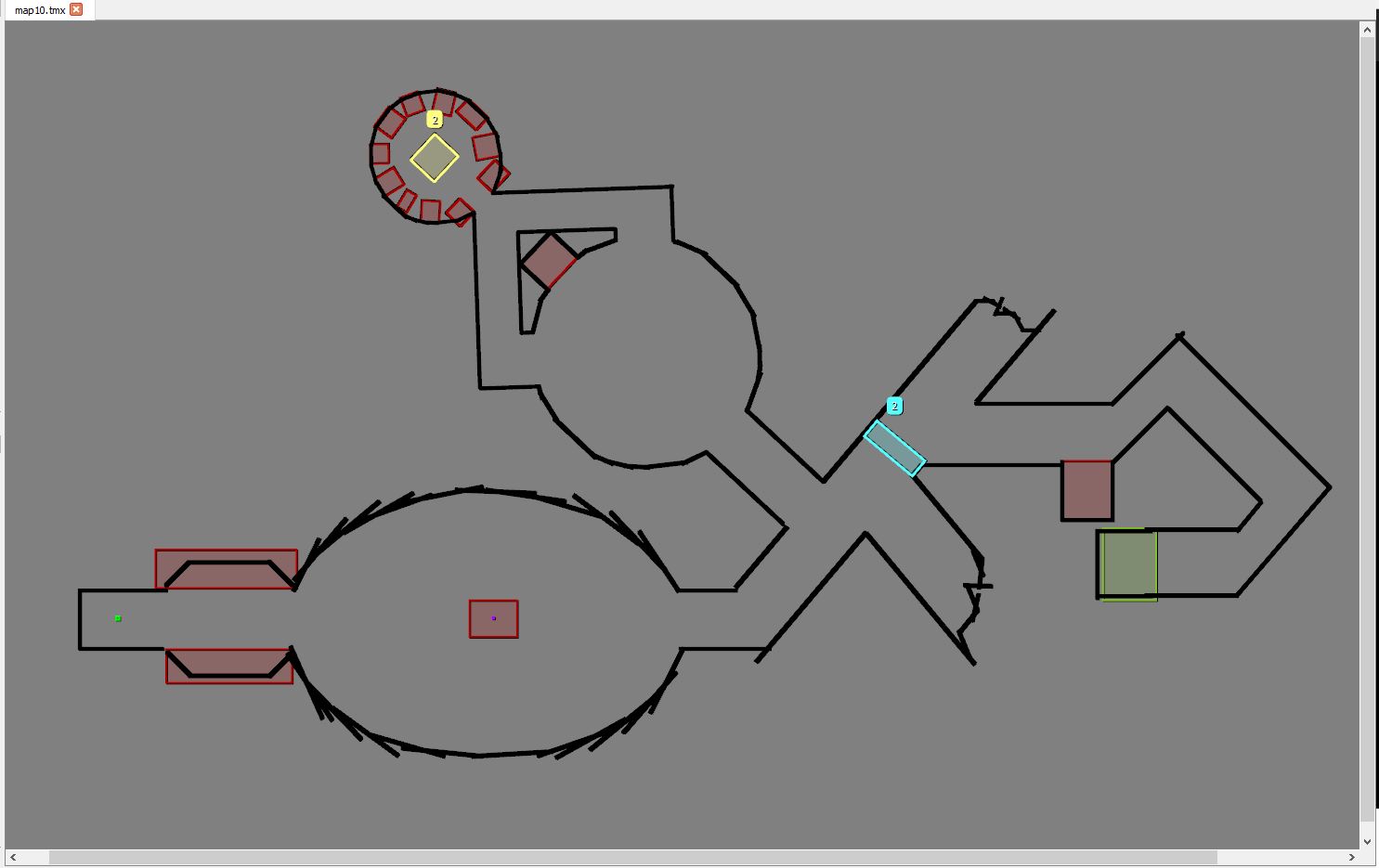
## Atraction



#### Maqueta del mapa “Atraction”

Nuestra primera vez con los agujeros negros, mucho cuidado con ellos.

## End Is Nigh



#### Maqueta del mapa “end is nigh”

Este es el último y más peligroso de todos los mapas de Echoes, mucho cuidado y buena suerte.

# Sonido

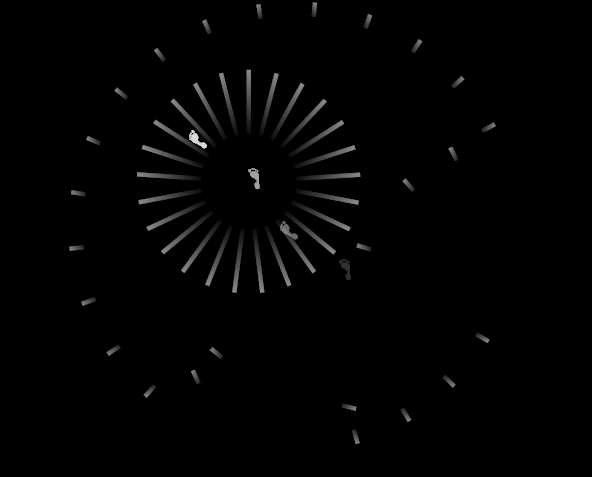
El sonido del juego es una parte fundamental de este, para la programación del sonido me he ayudado de las herramientas que SFML ofrece para el tratamiento de sonidos y música, y para la mejora de los sonidos he utilizado la herramienta Audacity, con la cual he podido modificar cada uno de los sonidos para darles un toque mas realista, quitar ruidos molestos y potenciar la inversión del videojuego.

## Sonido 2D

Usando la herramienta que nos ofrece SFML para ello, se ha utilizado la posición espacial del sonido en los objetos que se desplazan, como el personaje, los enemigos y los portales, de tal manera que podemos posicionar espacialmente los objetos con solo escucharlos, un detalle de mucha importancia para la inversión del jugador.

## Sonido Visual

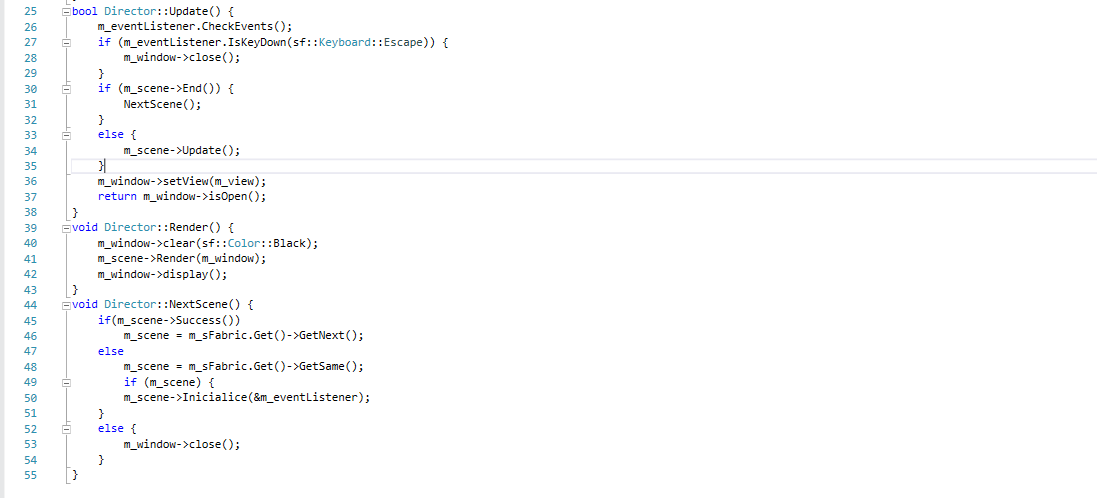
Como ya se ha comentado la parte más importante del sonido en este juego es la componente visual de este, ya que damos la apariencia de rayos a cada una de las ondas sonoras que existen en el juego, esto ha sido posible gracias a la potente herramienta grafica que es SFML y una serie de algoritmos creados por mí.



# Desarrollo

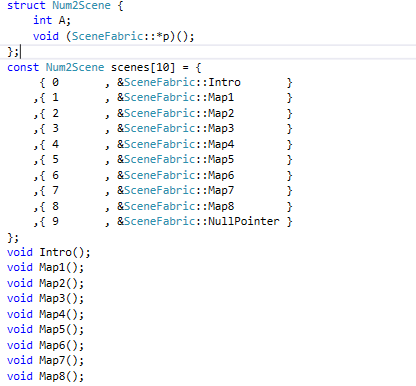
## Director

La figura del director es la clase que se ocupa de organizar las escenas durante el juego, es el que decide cuando dar paso a cada mapa.



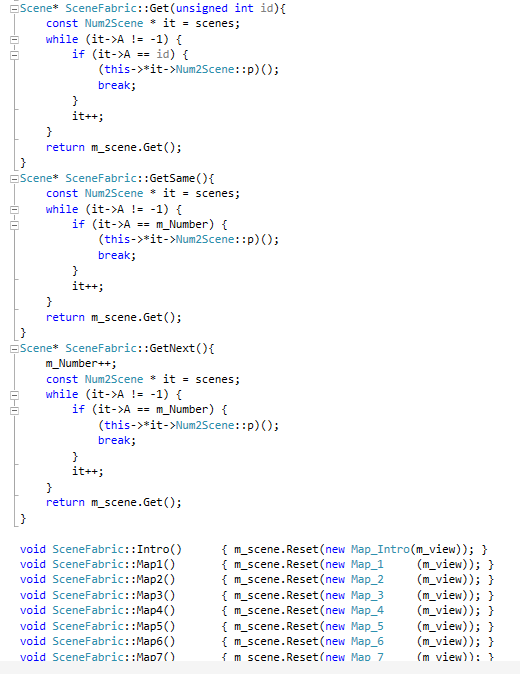
## Punteros a métodos

El uso de los punteros a métodos es una manera muy interesante de sustituir la encadenacion de varios condicionales if o su sustituto switch, de tal manera que tenemos diferentes métodos para cada situación y solo tenemos que llamar al método que toca en cada momento.



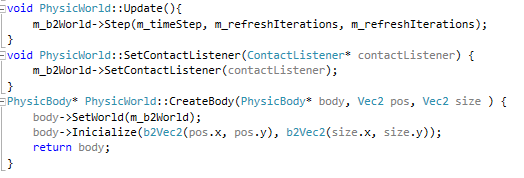
## Fabrica de mapas

Utilizando el patrón fabrica abstracta se ha realizado una clase encargada de generar objetos de un subtipo de la clase Mapa y enmascararlo en un mapa, usando los punteros a métodos que se ha comentado anteriormente.



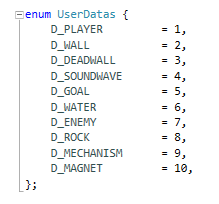
## Fachada del mundo físico Box2d

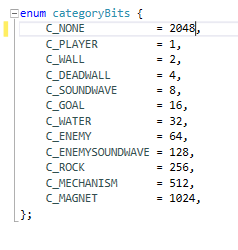
Aprovechando el patrón fachada se ha creado la clase PhysicWorld que enmascara cada acción que nos interesa del mundo box2d, esta nueva clase es la que se encarga de la comunicación con el.



## Mascara de colisiones

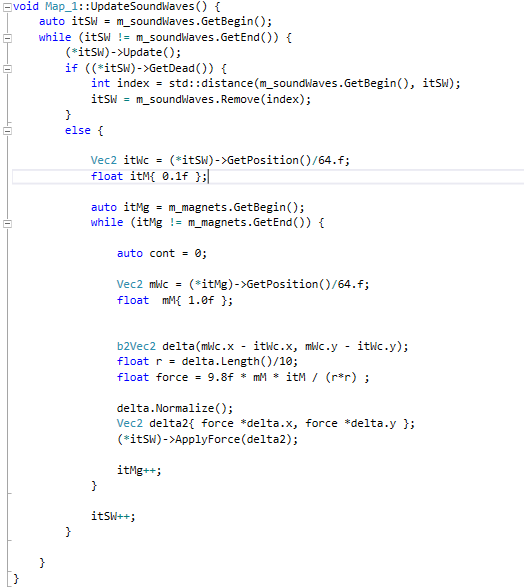
Para las mascaras de colisiones se ha dado un numero entero o un exponente de 2 a cada objeto para poder identificarlos.





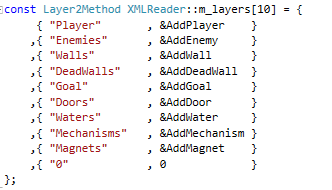
## Atracción magnética

Para la simulación de la atracción del agujero negro se calcula la fuerza provocada por cada agujero a cada onda sonora.



## Lectura de los ficheros XML

Para la lectura del fichero se ha creado una clase que usando la herramienta tinyXML traduce cada elemento del mapa en un objeto concreto del juego, para esto hace uso de los ya mencionados punteros a métodos de tal manera que busca el nombre de la etiqueta y comparando las cadenas de texto sabe qué tipo de objeto tiene que crear llamando al método que se encarga de ello.

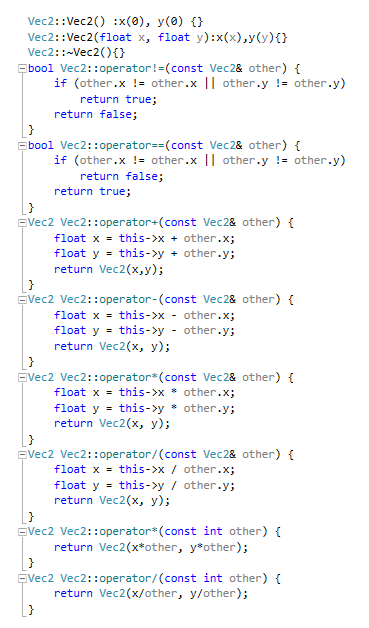




## Vector bidimensional

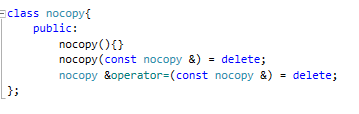
Para la unificación de las unidades usadas en cada uno de los componentes, como son SFML y Box2D, donde cada uno utiliza un tipo concreto de vectores, se ha creado una clase Vec2 que se usa desde el nivel fachada de cada uno de ellos.

Esta clase tiene redefinidos varios operadores para poder hacer algunas operaciones directamente con el operador que toca y no tener que operar cada una de sus componentes por separado.



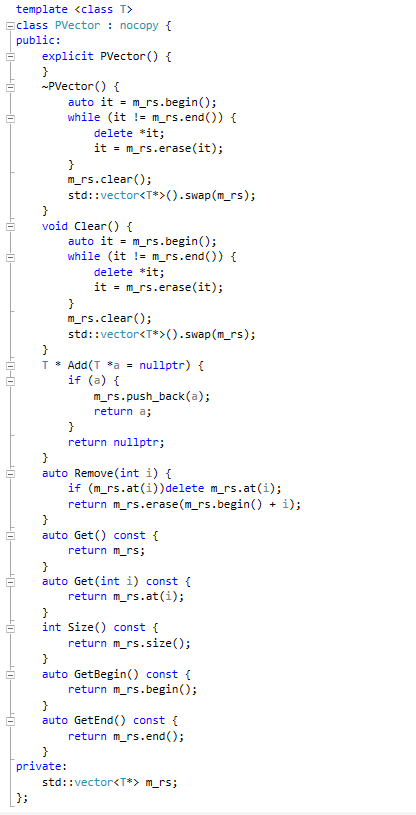
## Objetos sin constructor de copia ni operador asignación

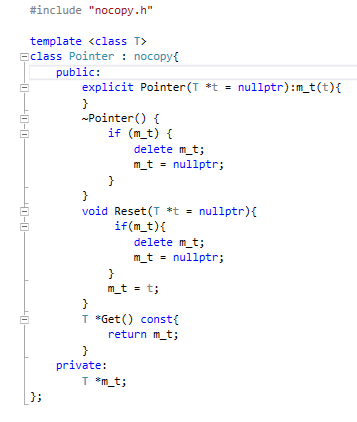
Se ha creado una clase muy sencilla y básica para que las clases que hereden de ella no tengan constructor copia ni operador asignación, así nos evitamos problemas de copia de recursos que no queremos.



## Clases puntero y vector

Para un control total sobre la destrucción de punteros se han creado unas clases especificas que se encargan de construir los objetos dinámicos, estas clases usan la herramienta template que ofrece c++ para ser capaces de crear cualquier tipo de objeto.

Estas clases se encargan de controlar y destruir el objeto dinámico que crea internamente en el momento adecuado

 .

## Clase GameObject

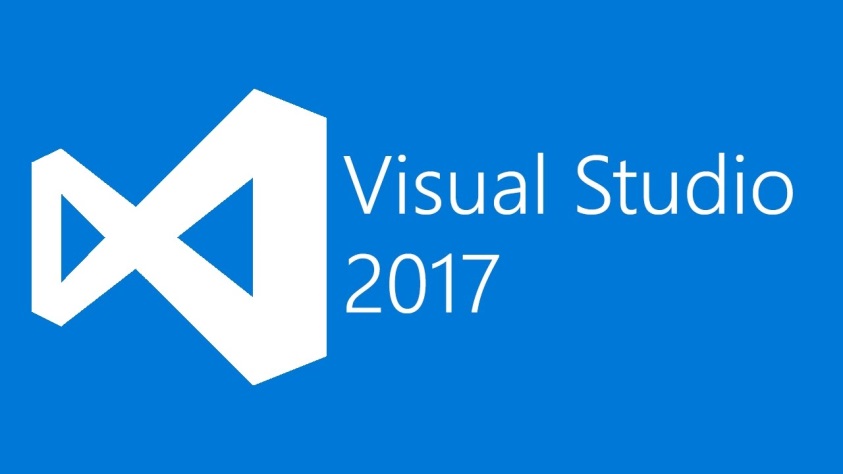
Para unificar en una sola clase los componentes físicos y gráficos de cada objeto se ha creado la clase GameObject, se encarga de actualizar la posición de los gráficos con la información aportada por el motor físico y de llamar a los métodos oportunos para obtener resultados físicos.



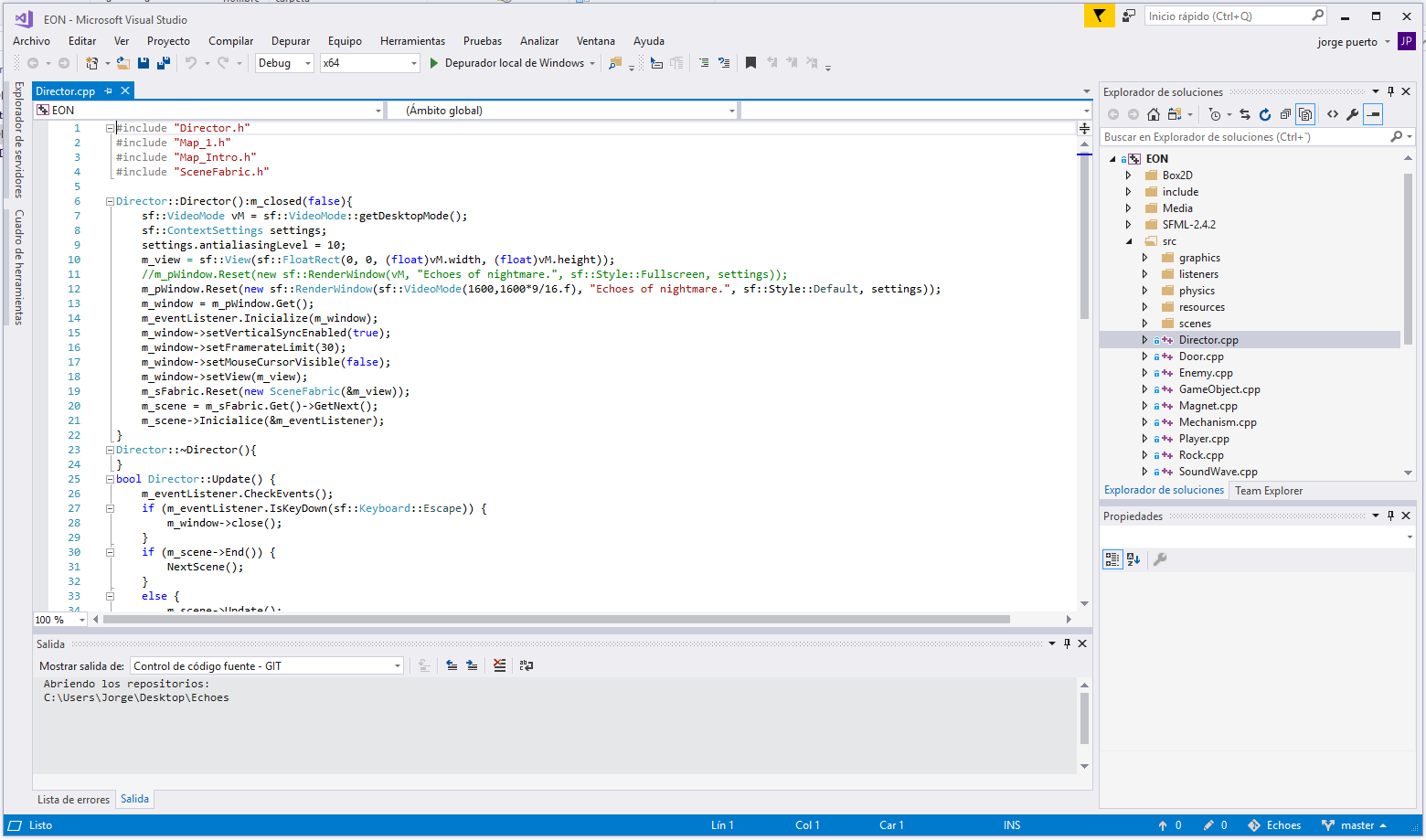
# Herramientas

## Visual Studio

Para la realización de este proyecto se ha optado por utilizar el entorno de desarrollo Visual Studio 2015 puesto que después de haber probado diferentes opciones a lo largo de la carrera, como son Codeblocks o eclipse, lo considero muy superior en calidad de las herramientas de depuración y ayuda a la hora de escribir código mas rápidamente.



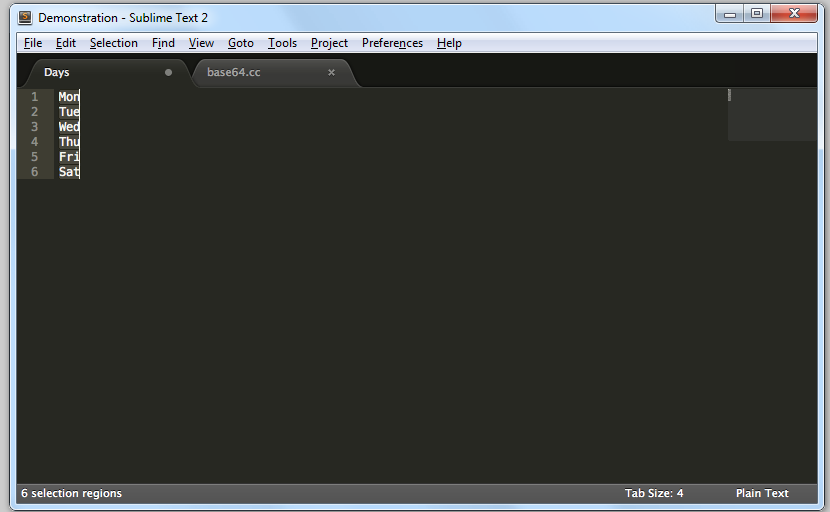
Visual studio es un compilador muy avanzado que tiene una serie de herramientas potentes de depuración.



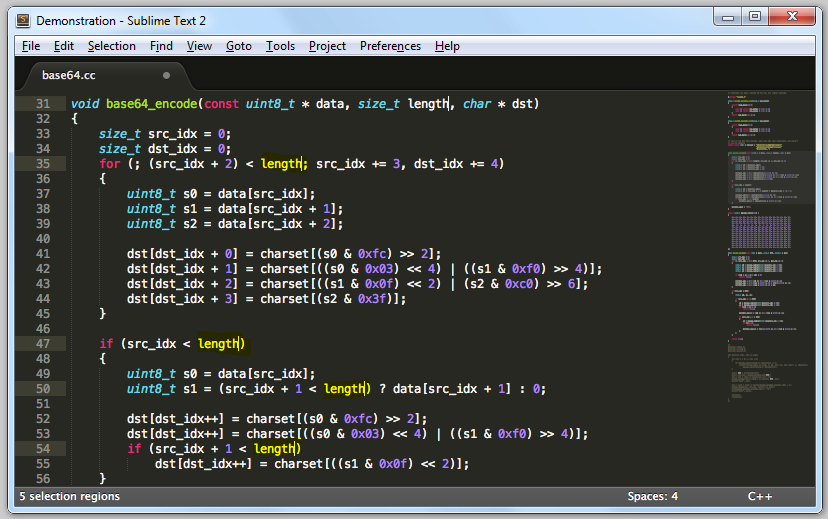
## SublimeText

SublimeText es un potente editor de texto que proporciona una serie de plugins y atajos que facilitan y mejoran mucho la experiencia del desarrollador, es mucho más potente en este sentido que visual estudio y junto a el hacen un gran trabajo a la hora de facilitar al programador ciertas tareas comunes de la programación. Por ejemplo:

### Edición múltiple de texto

Con la edición múltiple podemos editar a través de una serie de atajos de teclado varias líneas a la vez.

### Selección y renombre de variable

Con esta herramienta podemos seleccionar la misma variable en diferentes sitios y editarlos a la vez.

## SFML

Como motor gráfico se ha usado SFML por considerarlo una herramienta excelente que aporta tanto herramientas como tutoriales y ayudas para usarlas, es estupendo a la hora de utilizar gráficos en 2D y aporta librerías de audio, eventos y control de tiempo.

Ofrece una serie de herramientas que ayudan bastante a la hora de realizar videojuegos. Entre estas herramientas encontramos la posibilidad de dibujar por pantalla, herramientas para medir el tiempo, para crear rectángulos de colisión (bounding box), herramientas para el apartado de audio, vistas y cámaras, entre muchas otras. Sin embargo éstas herramientas son solo clases con ciertos métodos que permiten realizar una serie de funciones, de modo que es el propio desarrollador el que decide cómo y dónde utilizarlas.

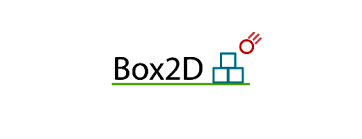
SFML proporciona una clase llamada Sprite que incluye aspectos básicos para tratar con ellos, como la posibilidad de rotarlos o escalarlos, poder establecer su centro u origen, asignarle una textura, así como asignarle una posición en pantalla, entre otros. Además, es uno de los elementos clave que la ventana de SFML puede dibujar por pantalla, facilitando mucho las cosas.

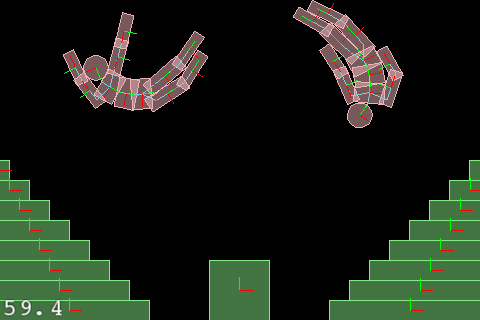


## Box2D

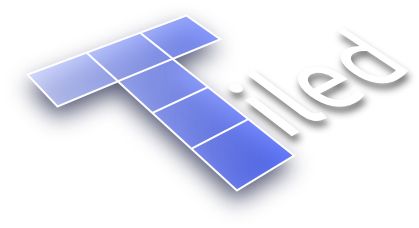
Box2D es una [biblioteca](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) [libre](https://es.wikipedia.org/wiki/Software_libre) que implementa un [motor](https://es.wikipedia.org/wiki/Motor_de_videojuego) físico en [dos dimensiones](https://es.wikipedia.org/wiki/Computaci%C3%B3n_gr%C3%A1fica_2D).

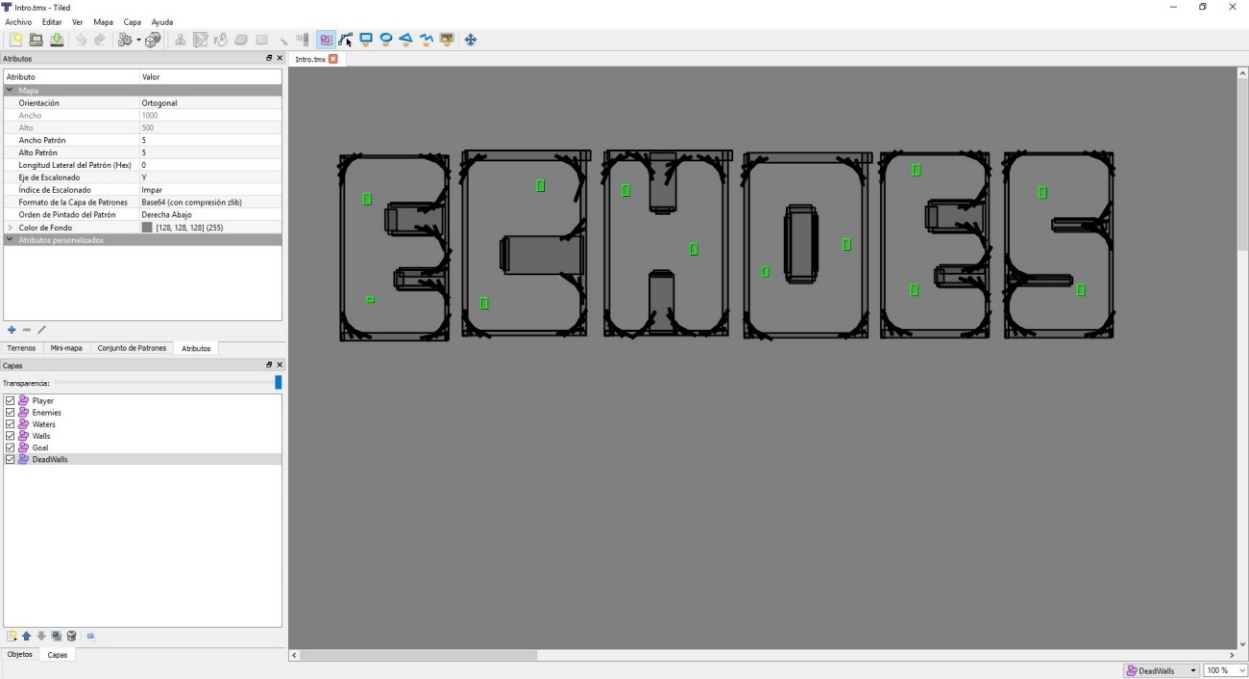
La utilización de una librería de físicas no es algo trivial, este tipo de herramientas requieren que todo el ciclo del juego se adapte a sus tiempos, a posicionar los objetos donde el motor le dice y a seguir la metodología que este sigue.

Puesto que es un juego que requería de una componente física muy potente, con manejo de mascaras de colisiones, sistema de partículas y resolución de colisiones complejas y rápidas se ha apostado por la librería de físicas Box2D, a la que se le ha dotado de una fachada con el patrón façade para poder sustituir la librería de manera rápida en caso de necesitarlo.



## Tiled

Para la realización de los diferentes mapas del juego se ha utilizado la herramienta tiled, de la cual se han utilizado únicamente las capas de objetos, esta herramienta te permite dibujar en una rejilla la forma del mapa deseado utilizando capas para distinguir entre los diferentes objetos y guarda la información en un fichero de texto XML, el cual leeremos de la manera que nos convenga desde la aplicación y podremos construir nuestros mapas.



## TinyXML

Como ya se ha comentado, al utilizar la herramienta Tiled para crear los mapas se requiere de una librería de lectura de ficheros XML, y para ello se ha utilizado tinyXML, que permite recorrer un fichero en este formato a través de las etiquetas de manera fácil e intuitiva.

## Gimp

Para la realización del sprite de los pies del personaje se ha utilizado la herramienta Gimp, que es un potente entorno de diseño gráfico gratuito.



## Audacity

Con Audacity se han modificado y ajustado los audios capturados para el juego para adaptarlos a las necesidades que teníamos al realizar un juego de terror, como ecos o reverberaciones.

## Github y SourceTree

El objetivo de usar un sistema de control de versiones es gestionar ágilmente proyectos. Parte de su principal propósito es que puedas volver a un estado anterior del proyecto o conocer, incluso, toda su evolución en el tiempo. Desde sus inicios hasta donde se encuentra actualizado. Puedes ver a los SCV como máquinas del tiempo, que permiten regresar a cualquier momento que quieras de tu proyecto.

El repositorio github es un entorno de consola, como repositorio y sistema de control de versiones se ha utilizado una cuenta en GitHub y el entorno grafico SourceTree de Atlassian.





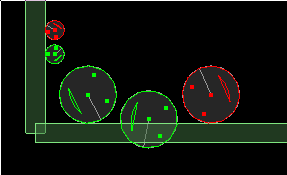
# Físicas

La física ha sido fundamental en el desarrollo de los videojuegos a través del tiempo. Los juegos emplean la física para alcanzar simulaciones realistas y crear efectos especiales espectaculares logrando entregar una sensación lo más creíble posible al espectador/jugador. De esto nace la necesidad de generar videojuegos capaces de simular la realidad, desde juegos de billar, simuladores de vuelo, entre varios; todos son parte de un proceso de modelamiento empleando los principios fundamentales de la dinámica.

Puesto que la programación de un motor de físicas es complejo si se requiere de una simulación realista y que este proyecto no

## Mascaras de colisiones

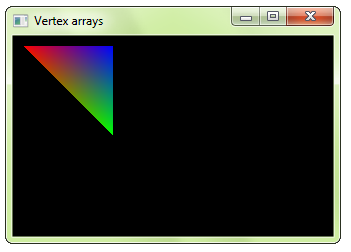
Dada la complejidad de las colisiones y la gran variedad de objetos con diferentes respuestas a las colisiones dependiendo que tipo de objeto choca con cada uno de ellos, se ha implementado un sistema de mascaras de colisiones para indicar a box2d que colisiones son sensores, cuales producen una colisión real y que resultado tiene cada una de las combinaciones.



# Motor Grafico

## VertexArray y TriangleStrips

Las ondas que simulan el sonido tenían que ser líneas continuas de un tamaño y grosor variable cuyo color fuera transparentándose a modo de estela y que en ciertas zonas del espacio cambiara de color, este tipo de objeto era nuevo para mí y no sabía cómo podía realizarlo utilizando las herramientas de SFML.



Para una primera aproximación de esta idea utilicé un vector variable de rectángulos, cada pocos milisegundos se generaba un rectángulo en la posición actual de la partícula situada en la cabeza del objeto, este vector se recorría en cada iteración de la aplicación y se otorgaba a cada rectángulo el color y transparencia que tocaba.

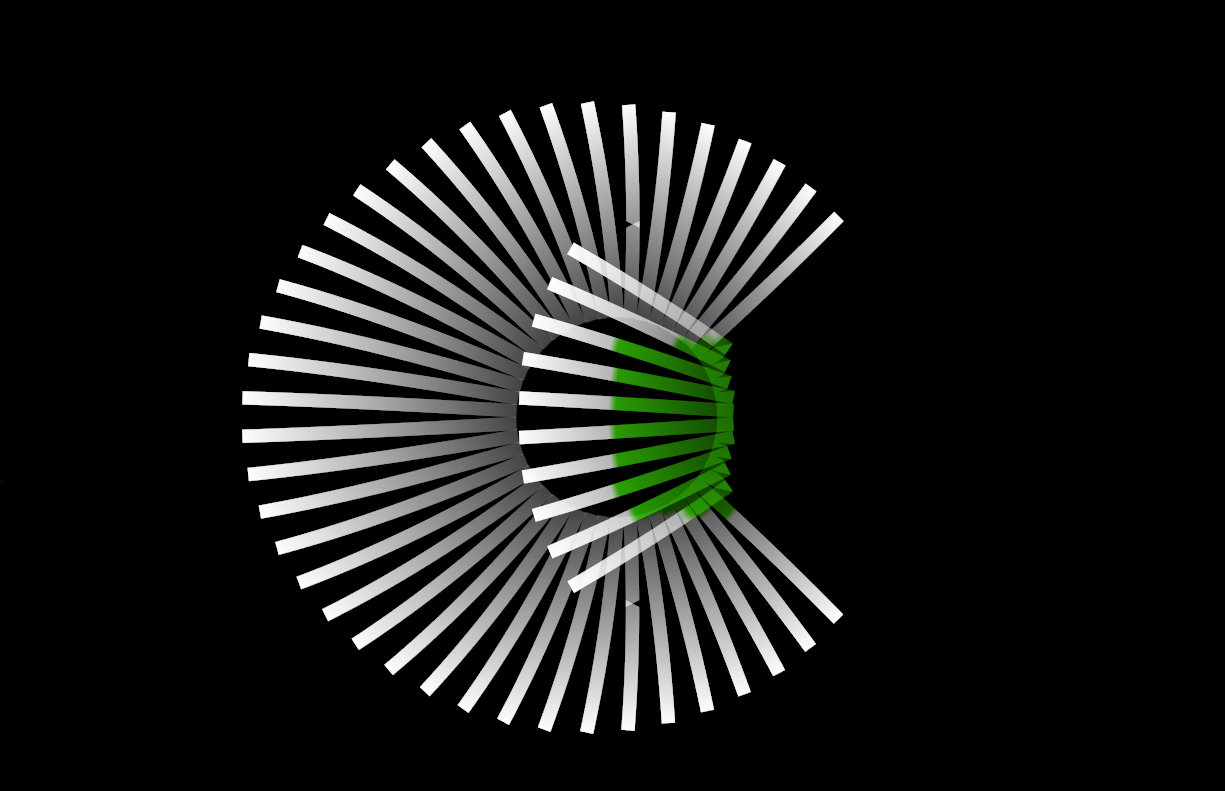
Esta solución a pesar de ser funcional no daba el resultado final que se quería conseguir, además de ser muy costosa en tiempo de ejecución.

Finalmente, y tras una exhaustiva búsqueda de una mejor implementación de esta idea encontré las herramientas VertexArray y TriangleStrips, que juntas me daban la solución perfecta a mi problema.

VertexArray permite almacenar la información de una serie de puntos geográficos, cada uno con su información de color, y triangleStrips permite formar una forma geométrica compleja a partir de un vertexArray, donde cada 3 puntos forma un triángulo cuyo color es un gradiente de los colores de los puntos que lo componen.

Con esta herramienta fui capaz de almacenar cada pocos milisegundos la posición de los vértices delanteros de la cabeza de la onda en un vertexarray, y cada iteración formar un TriangleStrip que generara la forma deseada con los colores deseados, dado que cada punto espacial guarda la información de su propio color.

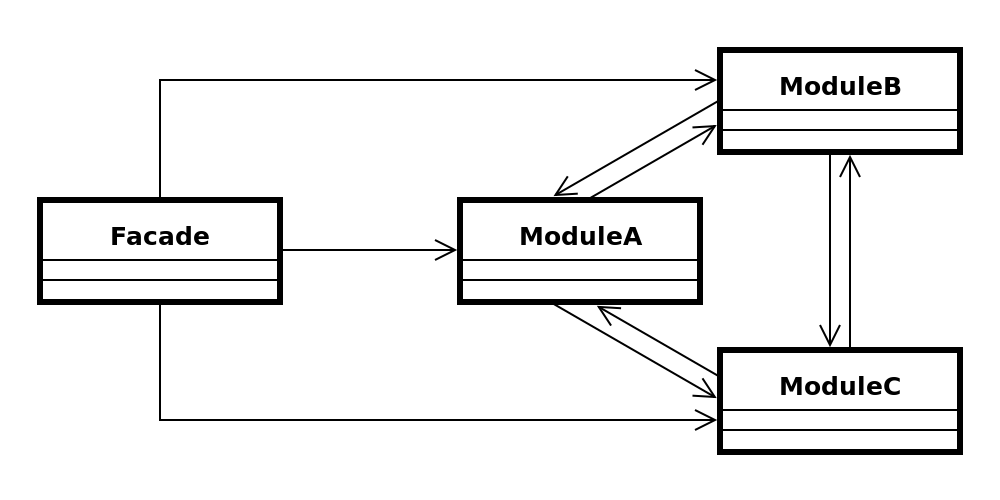
Solo faltaba saber cuántos vértices guardar en cada momento dependiendo del tamaño de estela deseado.



# Patrones de diseño

## Fachada

El patrón Façade se utiliza para desacoplar las diferentes funcionalidades, librerías o clases de una aplicación o para aportar interfaces simples a estructuras complejas, esto se realiza creando clases e interfaces intermedias que se comunican con la librería que se quiere usar, de esta manera aporta una interfaz de uso a esta parte de la aplicación.

En este proyecto se ha decidido implementarlo para minimizar las dependencias entre el código propio y los elementos proporcionados por SFML y Box2D, de modo que la utilización de elementos de estos sea mínima de forma directa utilizando primero por tanto aquel código que yo he creado. Algunos ejemplos de esto son las clases visuales VWall, VEnemy o VPlayer y las clases De físicas physicWorld y PhysicBody e entre otras. Con esto se ha conseguido que, llamando por ejemplo a la clase physicWorld, se pueda trabajar con sus métodos y variables (los cuales sí están relacionados con Box2D, haciendo de fachada) sin necesidad de llamar explícitamente a aquellos de Box2D. 

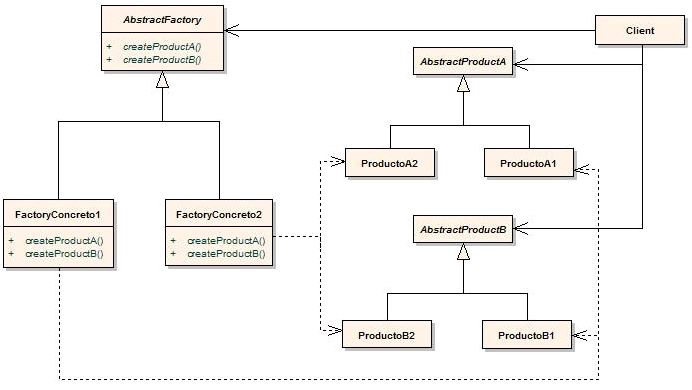
Como consecuencia de realizar clases, métodos y variables propias que sirvan de fachada a las que pudiese tener SFML o Box2D, se ha conseguido que el código sea mucho más intuitivo con respecto al desarrollador, ya que es uno mismo el que define esos elementos y por tanto pueda desarrollar con una mayor facilidad a la hora de recordar qué elementos contiene cada clase y para qué sirven.

## Abstract Factory

La factoría abstracta permite crear diferentes objetos de la misma familia encapsulados en un objeto superior en la jerarquía y ser utilizados todos de la misma manera, de esta manera fuera de la factoría no es necesario saber que tipo de objeto se creó realmente, simplemente se utilizan todos los objetos que nos devuelve de la misma manera.

En este proyecto se ha usado en varias ocasiones, en primer lugar se ha usado para generar todos los tipos de escenas posibles, menú y mapas, todos ellos bajo la clase Escena y manejados por la claseSceneFabric.

Otra implementación del patrón está en la parte física de los objetos del juego, que quedan todas ellas encapsuladas en el tipo physicBody. Por ejemplo: Para crear una onda de sonido el encargado de generarlo crea un PBSoundWave (cuerpo físico de la onda de sonido) y lo encapsula en un physicBody, que es el que recibe la onda de sonido y a partir de la cual obtiene su comportamiento físico.



## Singleton

Este patrón permite que un objeto sea creado una única vez en todo el transcurso de la ejecución, de modo que solo se creará cuando no exista ninguno, por lo que si se solicita su creación habiéndose creado ya anteriormente, en vez de crearse de nuevo se devolverá el que ya existe.

Esto es muy útil para controlar diversos elementos que pueden afectar de forma general a todo el código, como pueden ser relojes o gestores de cualquier tipo (de partículas, efectos, de recursos, etc.), los cuales solo es necesario crearlos una vez.

Esto ha sido muy útil en el proyecto actual ya que el hecho de que existan éstos elementos, los cuales pueden ser utilizados en diversos puntos del proyecto, ha permitido reducir bastante código y hacer más simple la elaboración de nuevos elementos tales como enemigos o ondas sonoras. Al tratarse de forma “global”, los gestores y toda aquella clase que tuviese éste patrón ha podido ser utilizada desde cualquier otra clase sin necesidad de crear una nueva instancia de ésta, simplemente se ha creado un puntero que llama al método Instanciar de la clase con Singleton, recibiendo su instancia ya creada.

## RAII

El patrón RAII hace el código seguro frente a excepciones, se utiliza para liberar recursos antes de permitir a las excepciones que se propaguen (para evitar fugas de memoria) escribiendo destructores apropiados nos ahorramos escribir código de “limpieza” duplicado y disperso por el código fuente entre bloques de manejo de excepciones que pueden ser ejecutados o no.

Este patrón se apoya en la característica de que posteriormente a una [excepción](https://es.wikipedia.org/wiki/Manejo_de_excepciones), el único [código fuente](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente) que con seguridad es ejecutado es el de los [destructores](https://es.wikipedia.org/wiki/Destructor_(inform%C3%A1tica)) de objetos que residen en la [pila](https://es.wikipedia.org/wiki/Pila_(inform%C3%A1tica)).

Además permite asegurarse que las clases de nuestra aplicación no crean constructores por defecto, constructores de copia ni operadores de asignación, esto se realiza creando una clase base que inhabilita estos constructores.

# Lenguaje

## C++

C++ es un lenguaje de programación orientado a objetos diseñado a mediados de los años 80 por Bjarne Stroustrup. Este lenguaje fue en su origen una extensión de C.

Hoy en día la mayor parte de los desarrollos en la industria de los videojuegos se realiza en C++, esto es porque es un lenguaje que te permite manipular de manera muy específica los componentes de la máquina que lo ejecuta, y permite optimizar de la mejor manera los programas realizados con este.

## C++ 14

Actualmente C++ se encuentra en su versión 14, la cual es la que se ha utilizado en el desarrollo de este proyecto, esta versión junto con la anterior (C++11) traen una serie de nuevas características muy interesantes.

# Bibliografía

## Effective C++: 55 Specific Ways to Improve Your Programs and Designs (Scott Meyers)

## Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14 (Scott Meyers)

## Dark Echoes

## Game Programming patterns (Rober Nysrom)