1. Introducción
2. 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.

1. 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.

1. Objetivos.

* Mejorar la capacidad de auto aprendizaje.
* Aprender a realizar un proyecto de tamaño medio.
* Unificar diferentes tecnologías para realizar el proyecto.
* Realizar un videojuego desde cero.
* Superar los problemas que surgen a lo largo de la creación de un proyecto.

1. Diseño y especificación.
   1. Resumen.

Echoes es un juego de terror que utiliza una formula sencilla: eliminar la visión y agudizar el oído. 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.

* 1. 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.

* 1. Mecanicas del juego.

1. Mecanicas:

* Sonidos como mecánica.

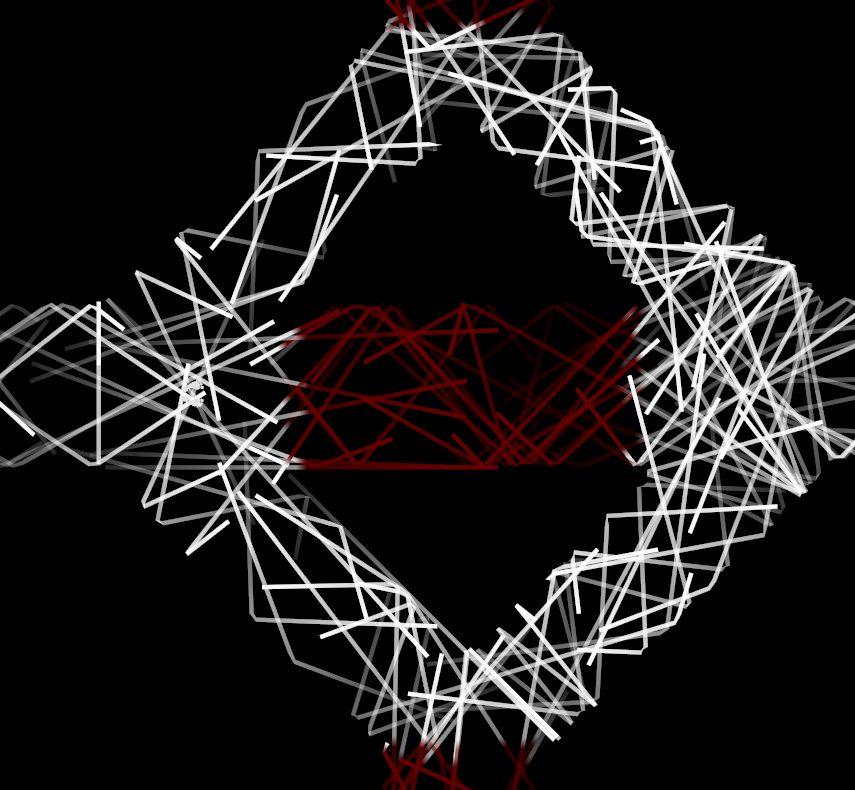
El sonido será la principal fuente de información de lo que sucede en el mundo que nos rodea, este será representado con líneas rectas cuya velocidad, intensidad y duración dependerán del la fuerza y volumen del sonido que la produce. Esto dará como resultado una pantalla cargada de información a través 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.

* Pinchos

Tendremos que tener cuidado de hacia dónde nos dirigimos, ya que la mazmorra estará llena de pinchos (los cuales pintarán de rojo las ondas) que tendremos que sortear.



* Enemigos.

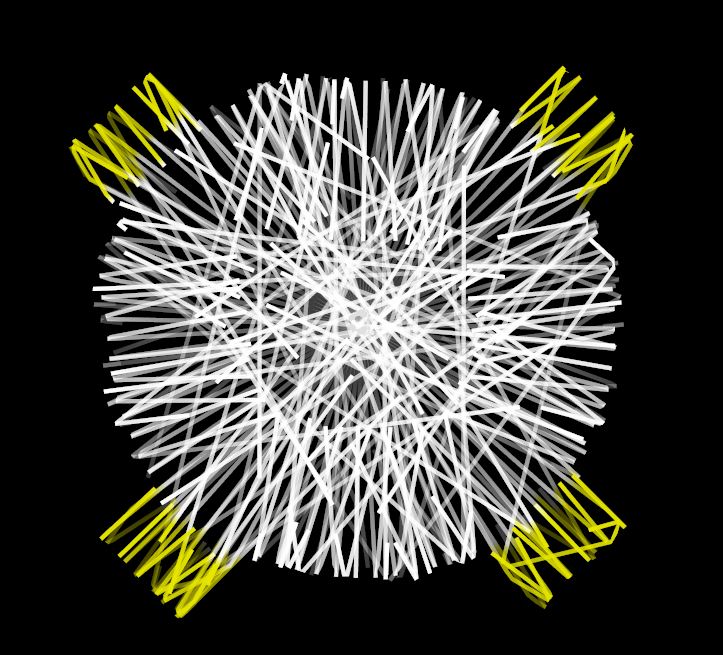
A lo largo y ancho de los mapas nos iremos encontrando con unos peculiares enemigos, de los cuales no conoceremos su aspecto puesto que no los vemos, pero si los escuchamos, y ellos a nosotros, ¡tened cuidado! Estos enemigos se desplazarán por el mapa solamente guiados por sus oídos, se dirigen siempre hacia el origen del sonido que escuchan.



* 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.

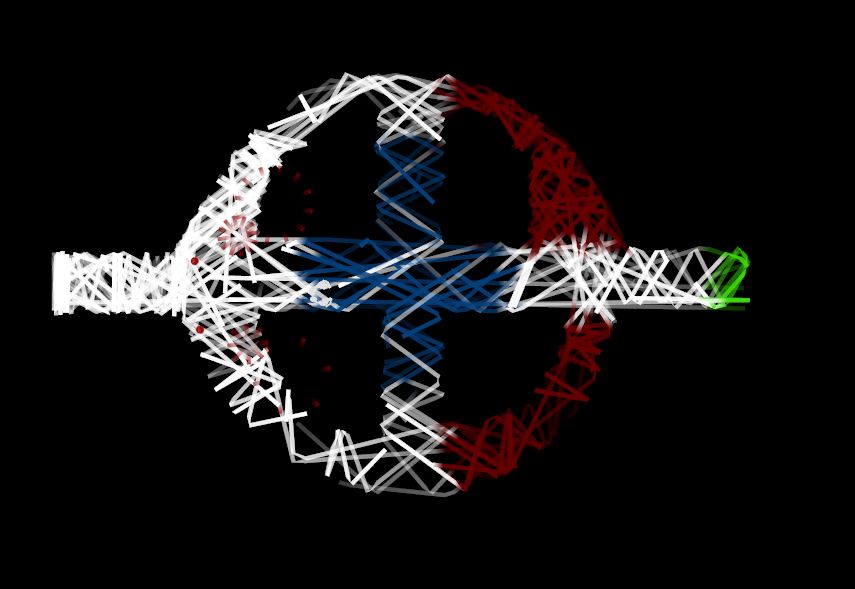
* Mecanismos y puertas.

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. 

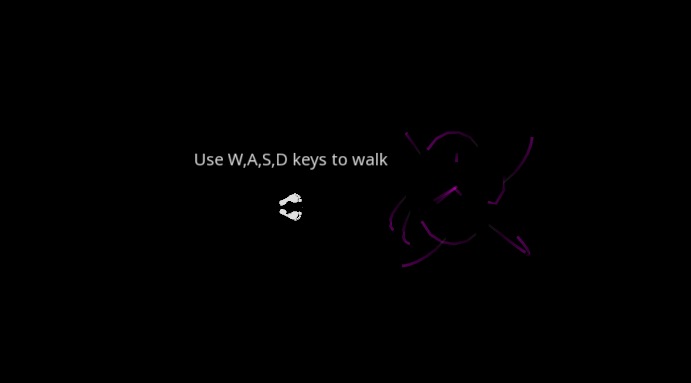
* Rocas.

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. 

* Agujero Negro (añadido especial)

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. 

* 1. Diseño del juego.

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.

* 1. Desarrollo del juego.

1. 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.

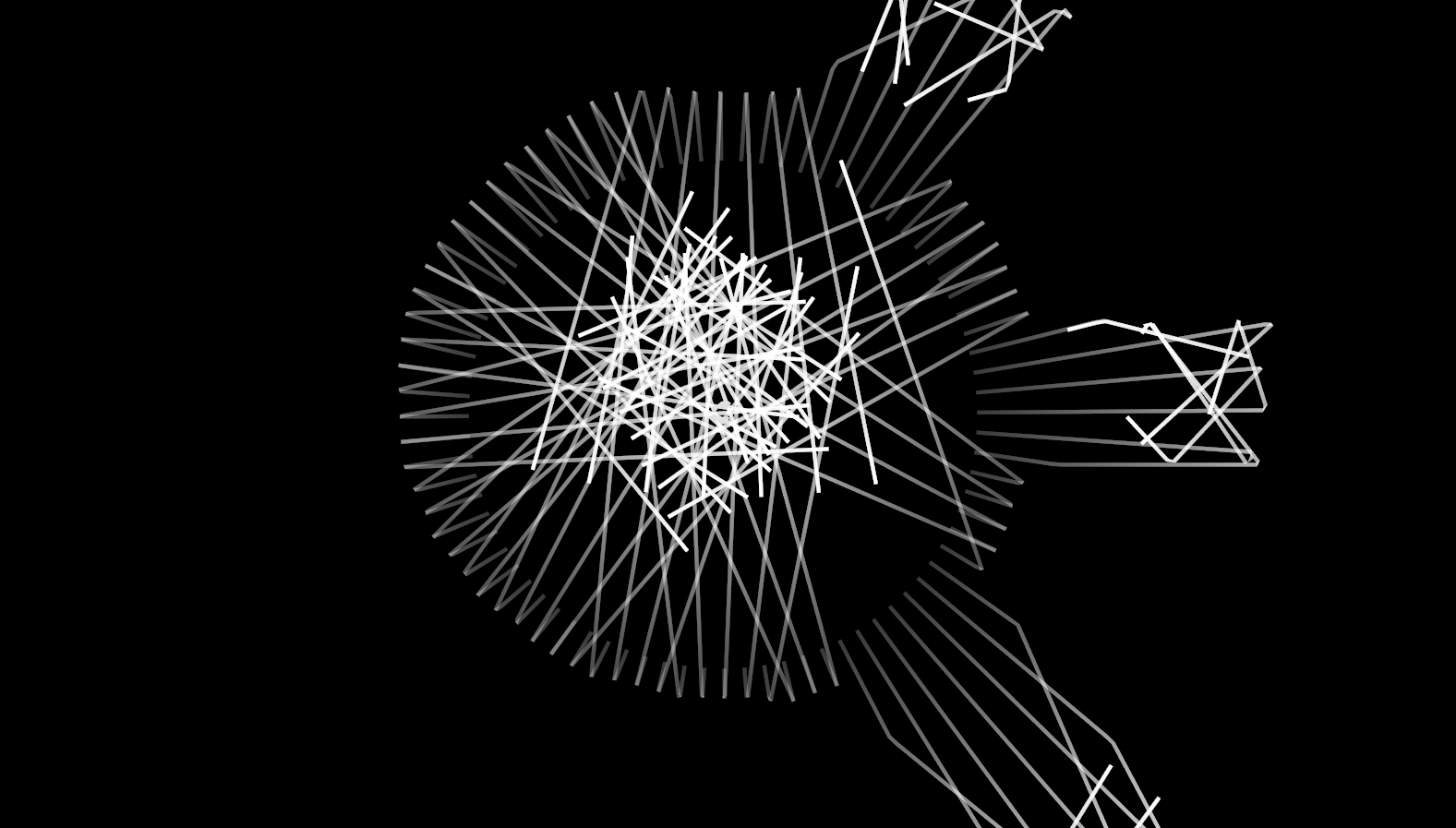
1. Físicos
   1. Personaje.

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.



* 1. Paredes.

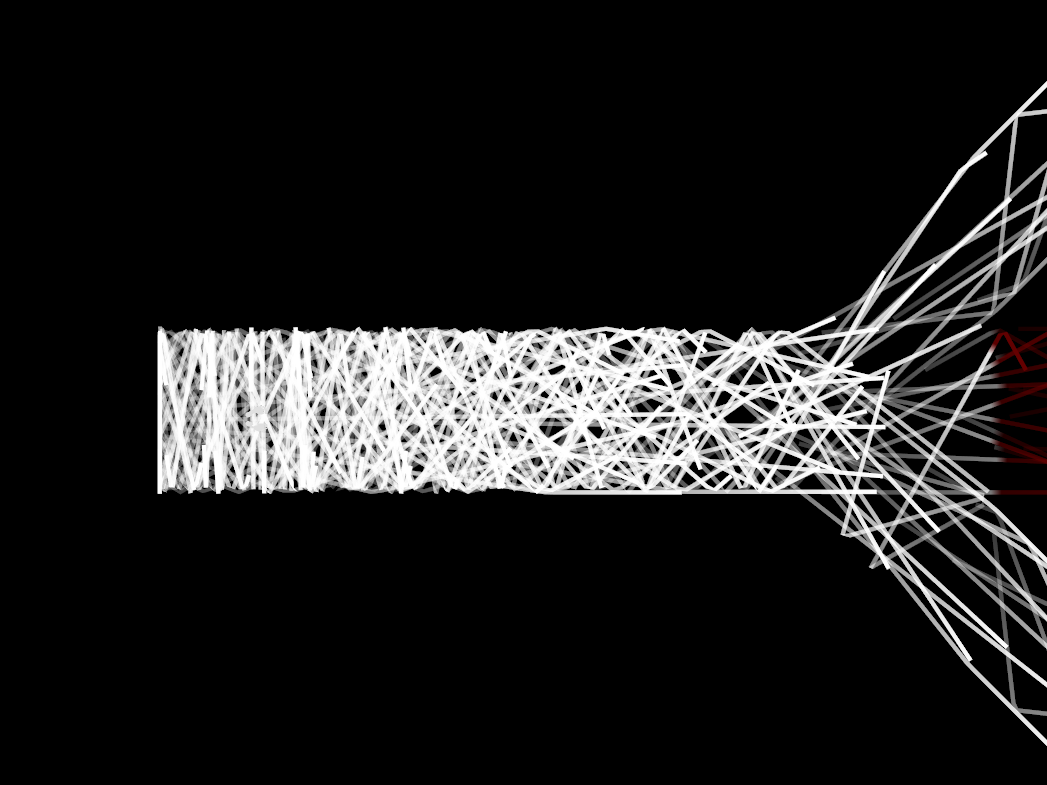
Objeto rectangular con colisión sencilla, no tiene más interacción que la de modificar la dirección del movimiento de los objetos que colisionan con ella. Esta estructura es la que define la forma del mapa, siendo un elemento muy importante.



* 1. 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.



* 1. Enemigos

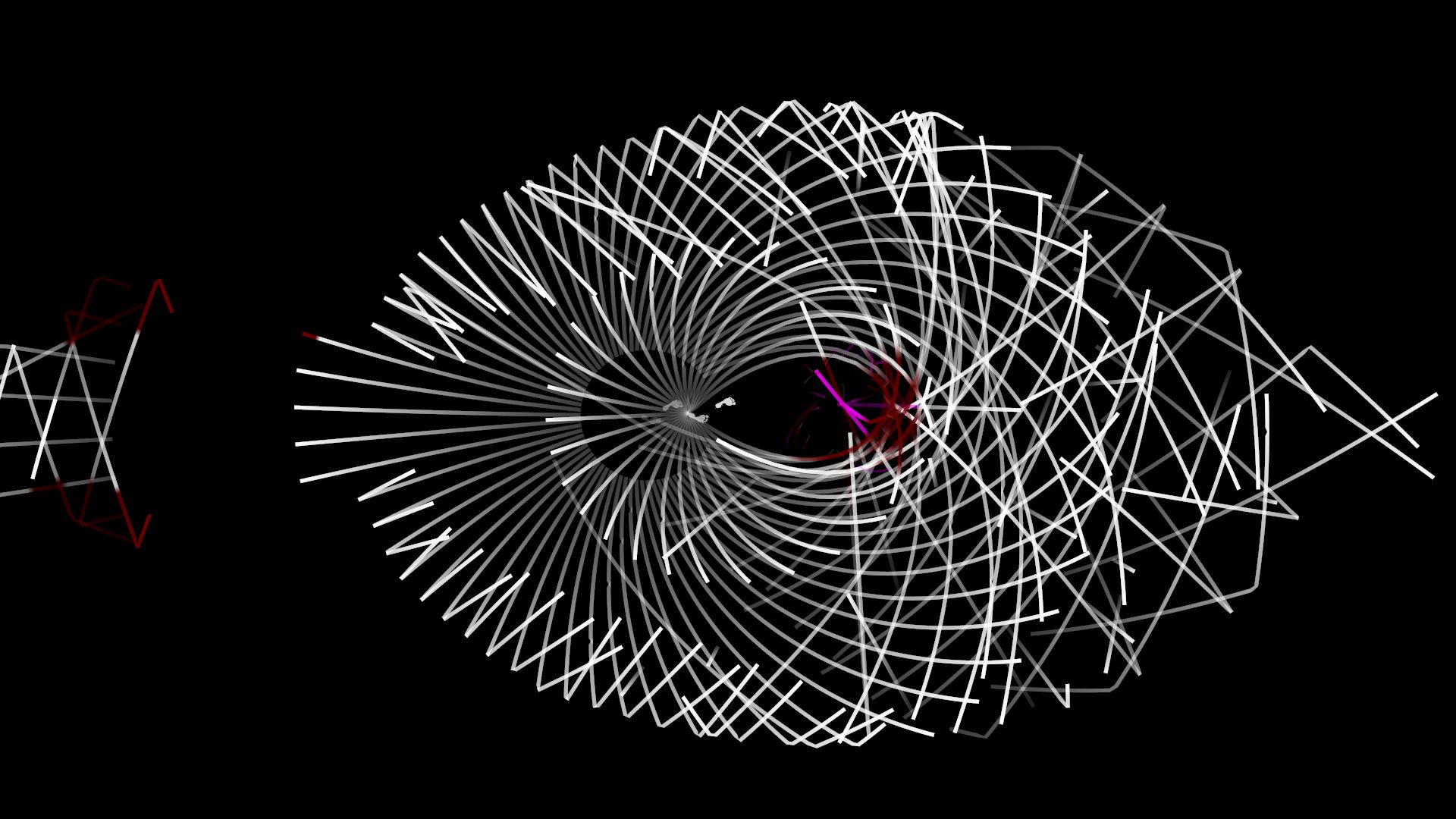
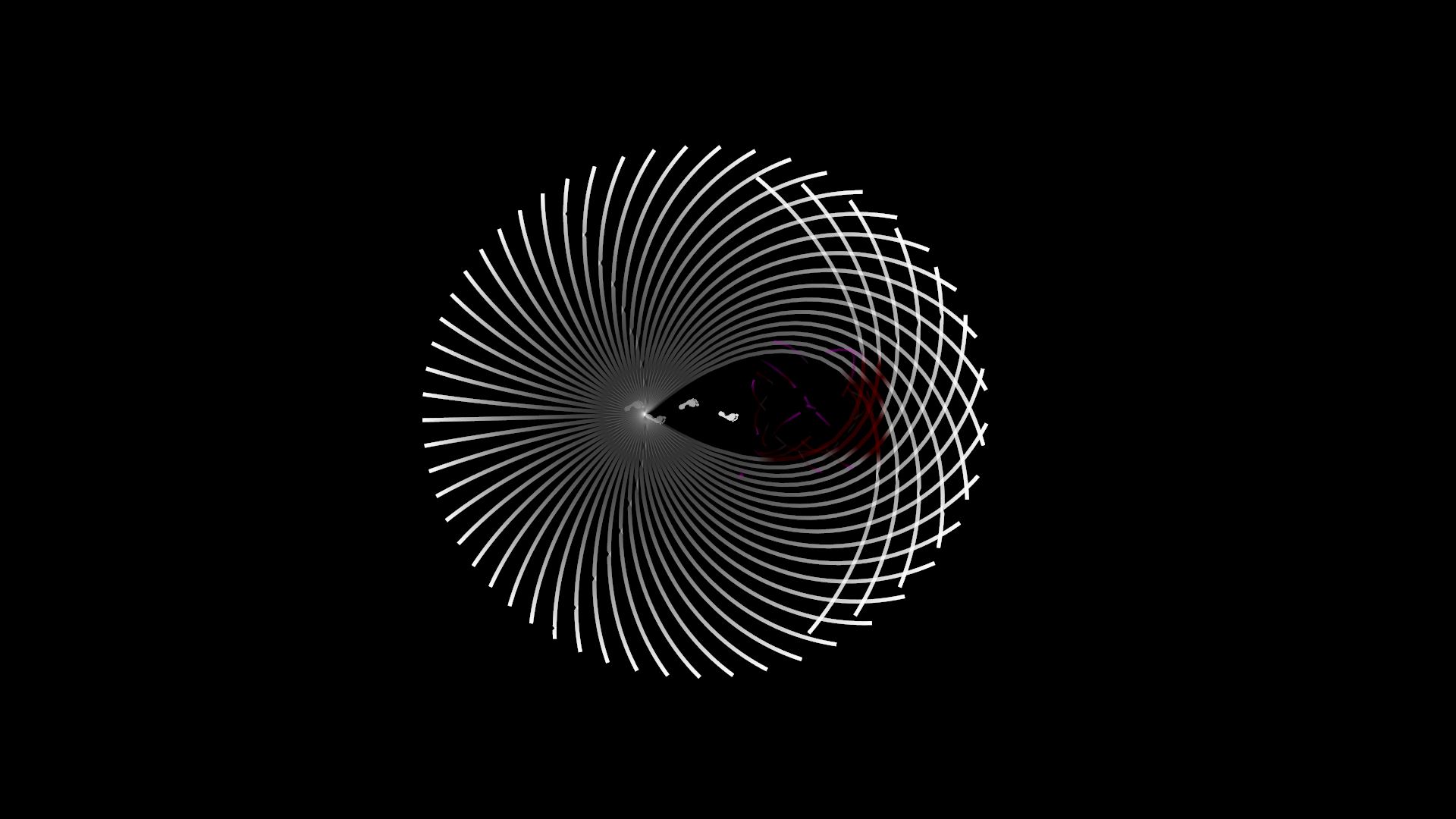
El enemigo es un objeto que se mantiene estático hasta que recibe la colisión de una onda sonora, a partir de ese momento comenzara a moverse en dirección al origen de la onda. En todo momento genera un sonido que visualmente se representa como pequeñas ondas saliendo de su centro.

* 1. Rocas

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.

* 1. Agujero Negro.

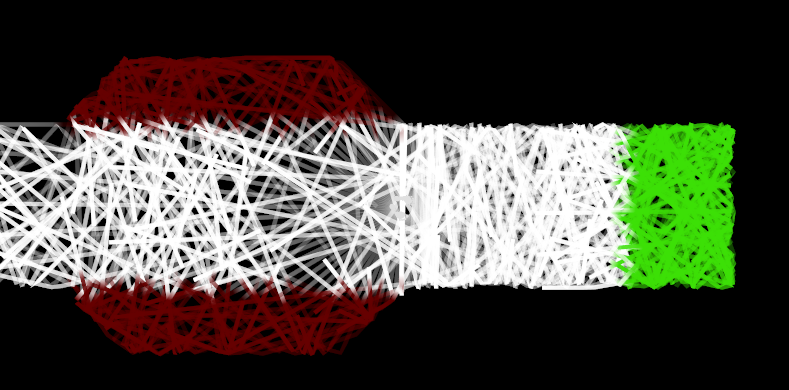
El agujero negro es un objeto especial que modifica físicamente a los demás, creando una atracción hacia el del personaje y las ondas sonoras, este generará una fuerza constante en todo momento desde el objeto hacia él, siendo mayor cuanto más cerca se encuentre de este.



1. Sensores
   1. Pinchos.

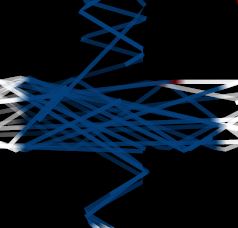
Los pinchos son un objeto rectangular cuya mascara de colisiones le permite generar un evento de entrada en colisión y fin de colisión con los objetos de tipo personaje y onda sonora, el evento de colisión con onda sonora permite que esta sea pintada durante esta colisión de color rojo, y la colisión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

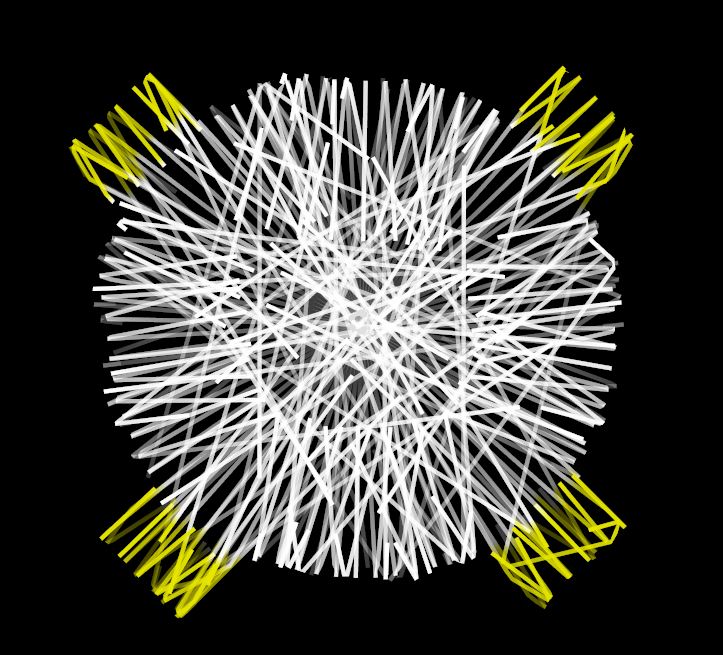


* 1. Agua.

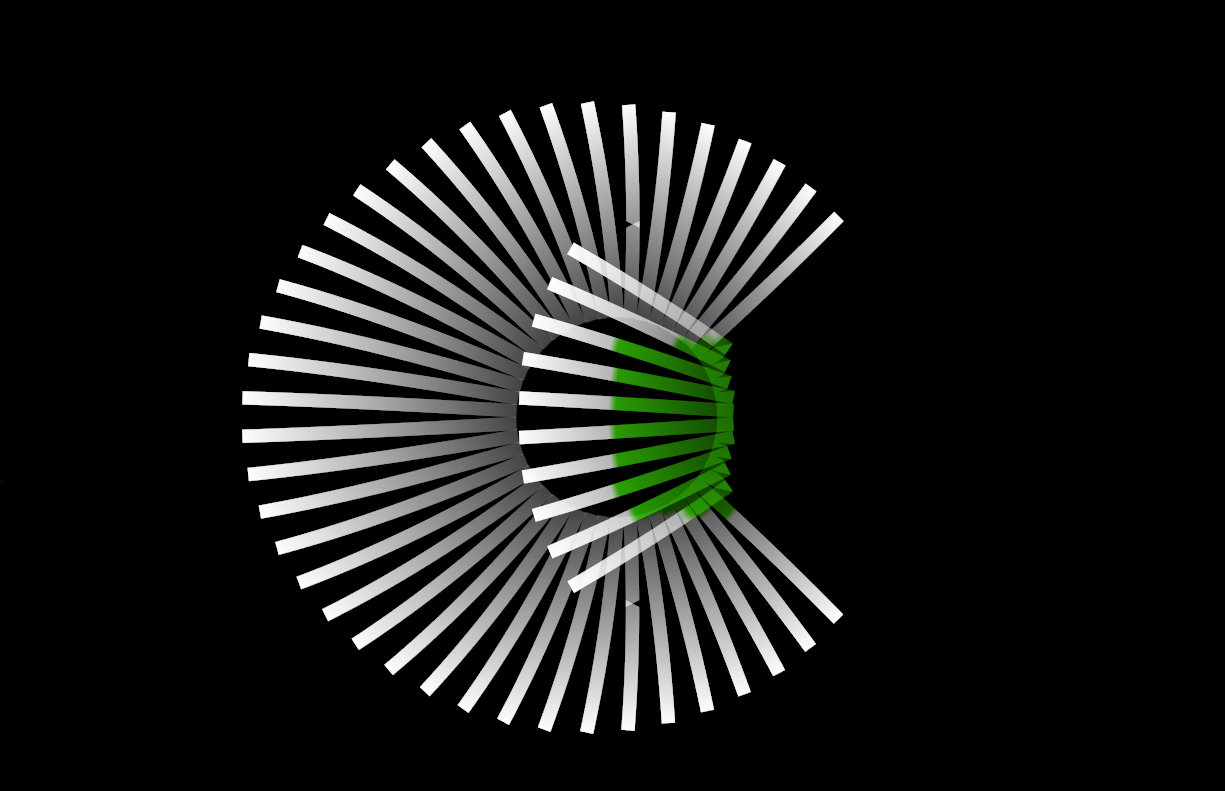
El agua es un objeto rectangular que sin tener propiedades físicas actúa como generador de eventos ante la colisión de esta con el personaje, haciéndolo más lento y ruidoso.



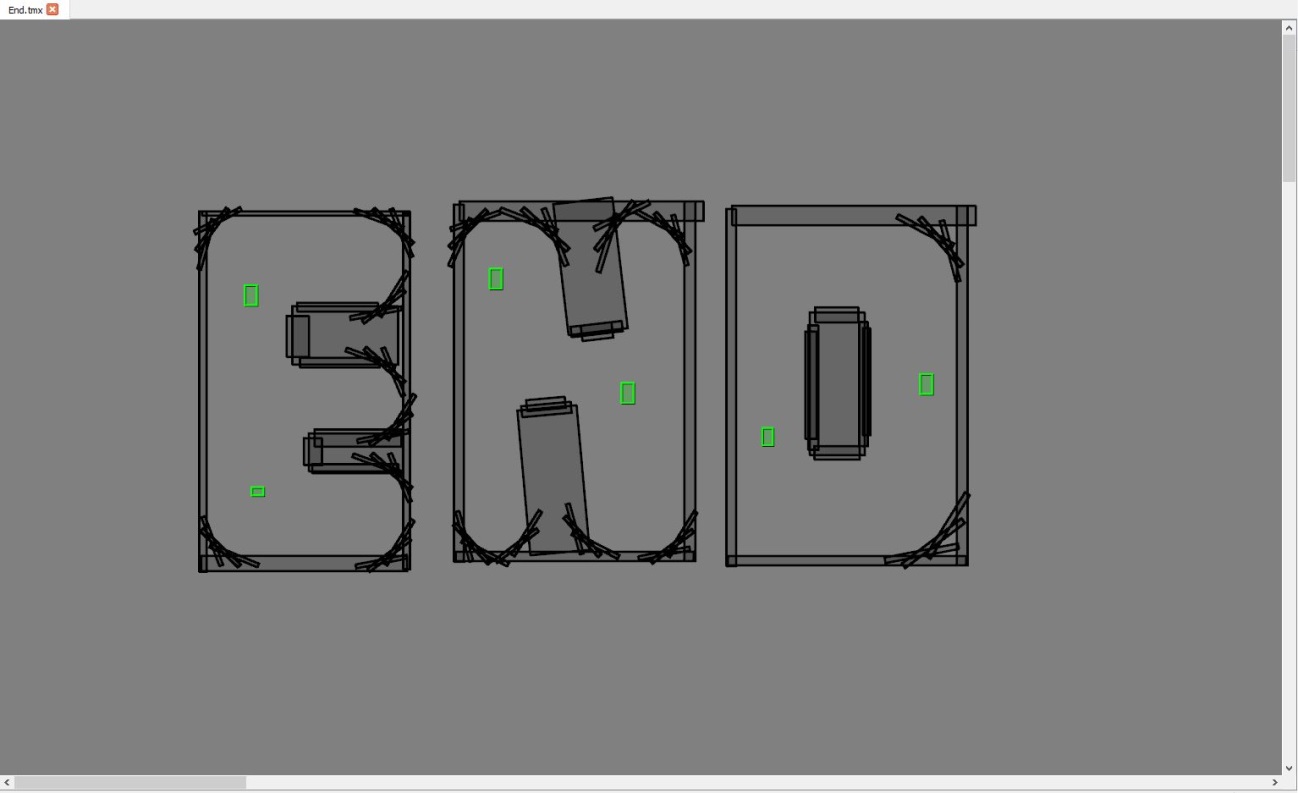
* 1. Mecanismo y puerta.

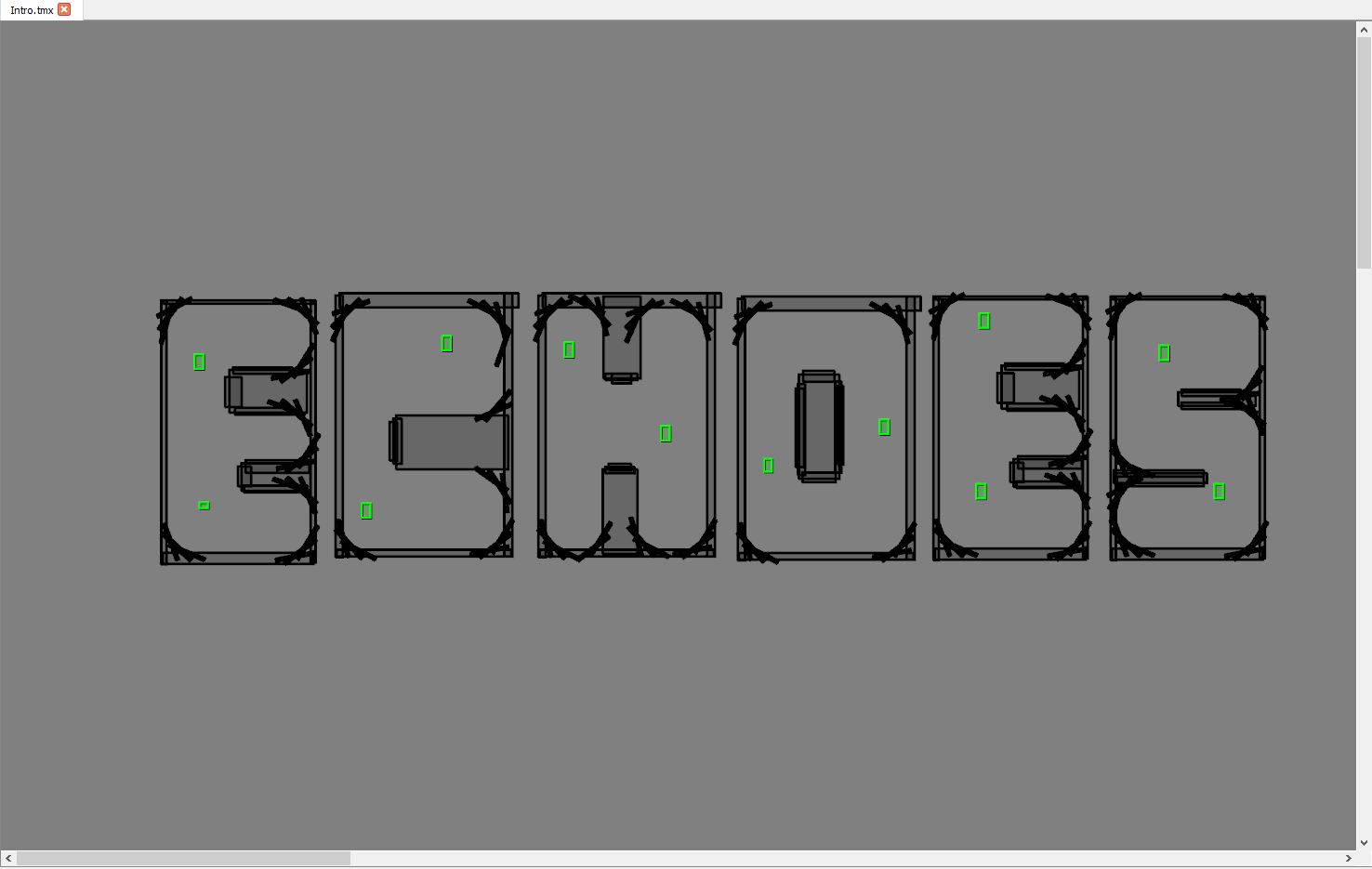
El mecanismo y puerta son 2 objetos que interactúan entre si, de tal manera que, si el personaje entra en contacto con el área definida por el mecanismo, se activa y abre la puerta asociada a él. 

* 1. Puerta Salida.

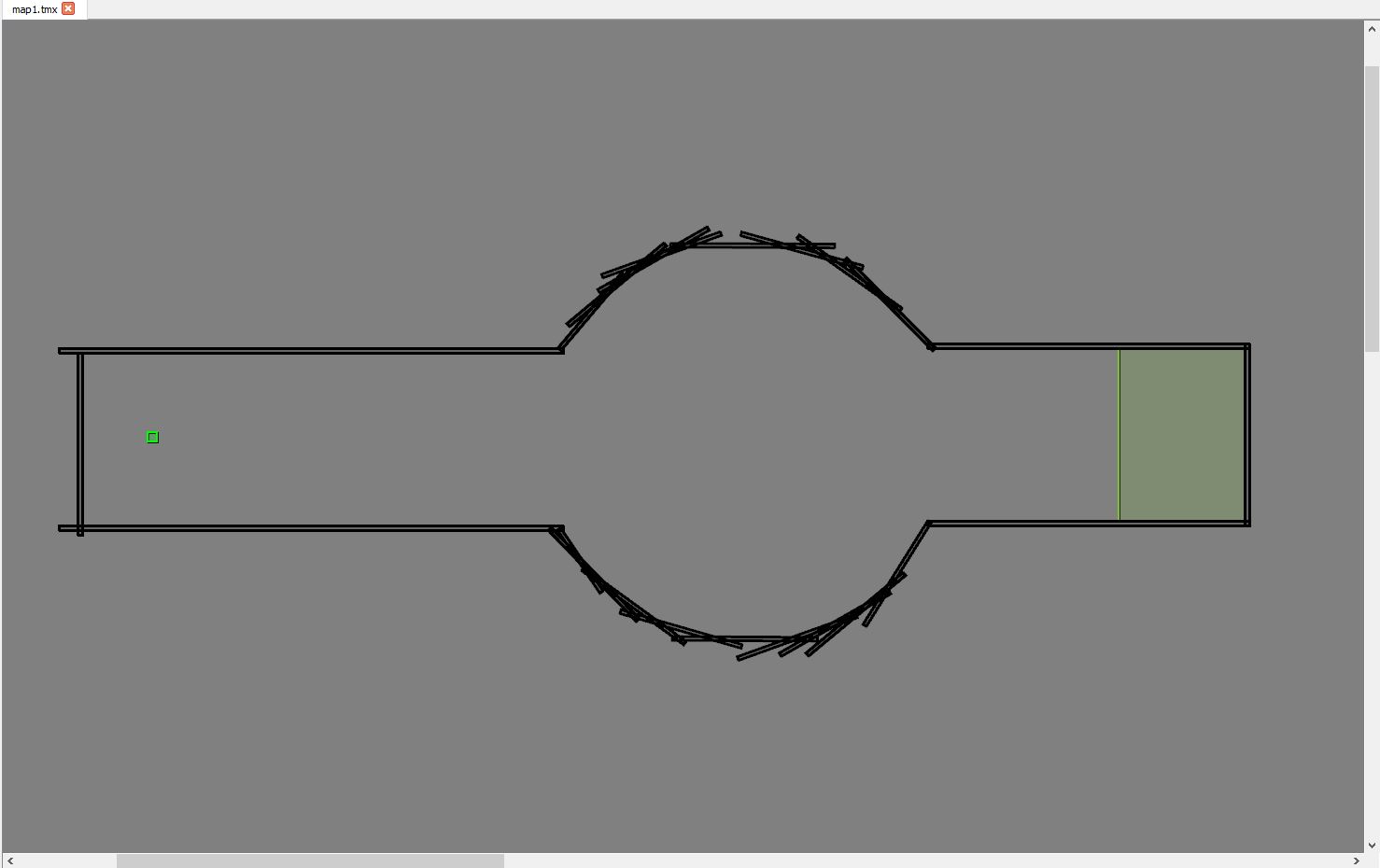
La puerta de salida es el sensor que representa el final del mapa, pinta de color verde a las ondas sonoras. 

1. Niveles:
   1. Intro y Fin

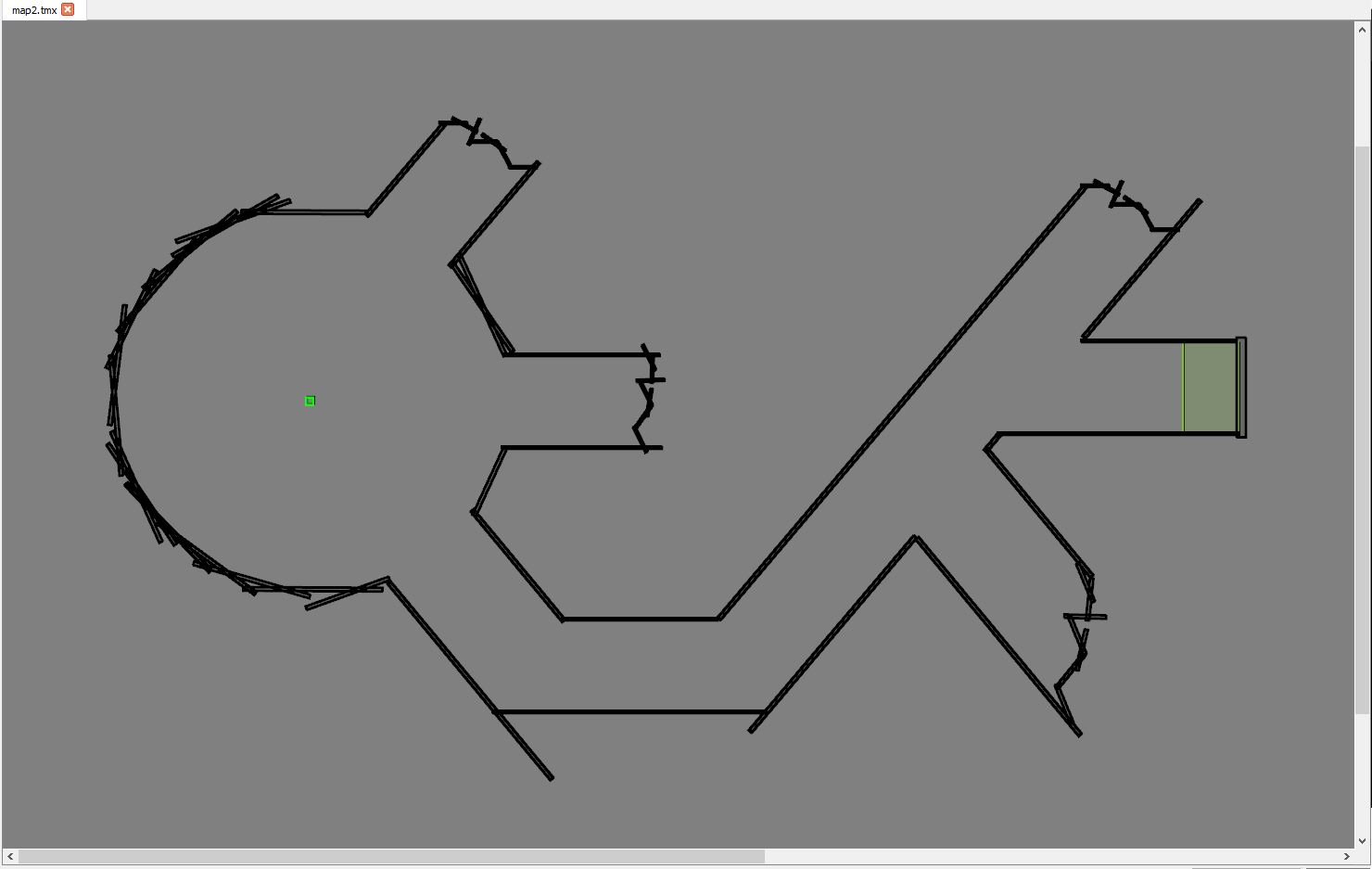




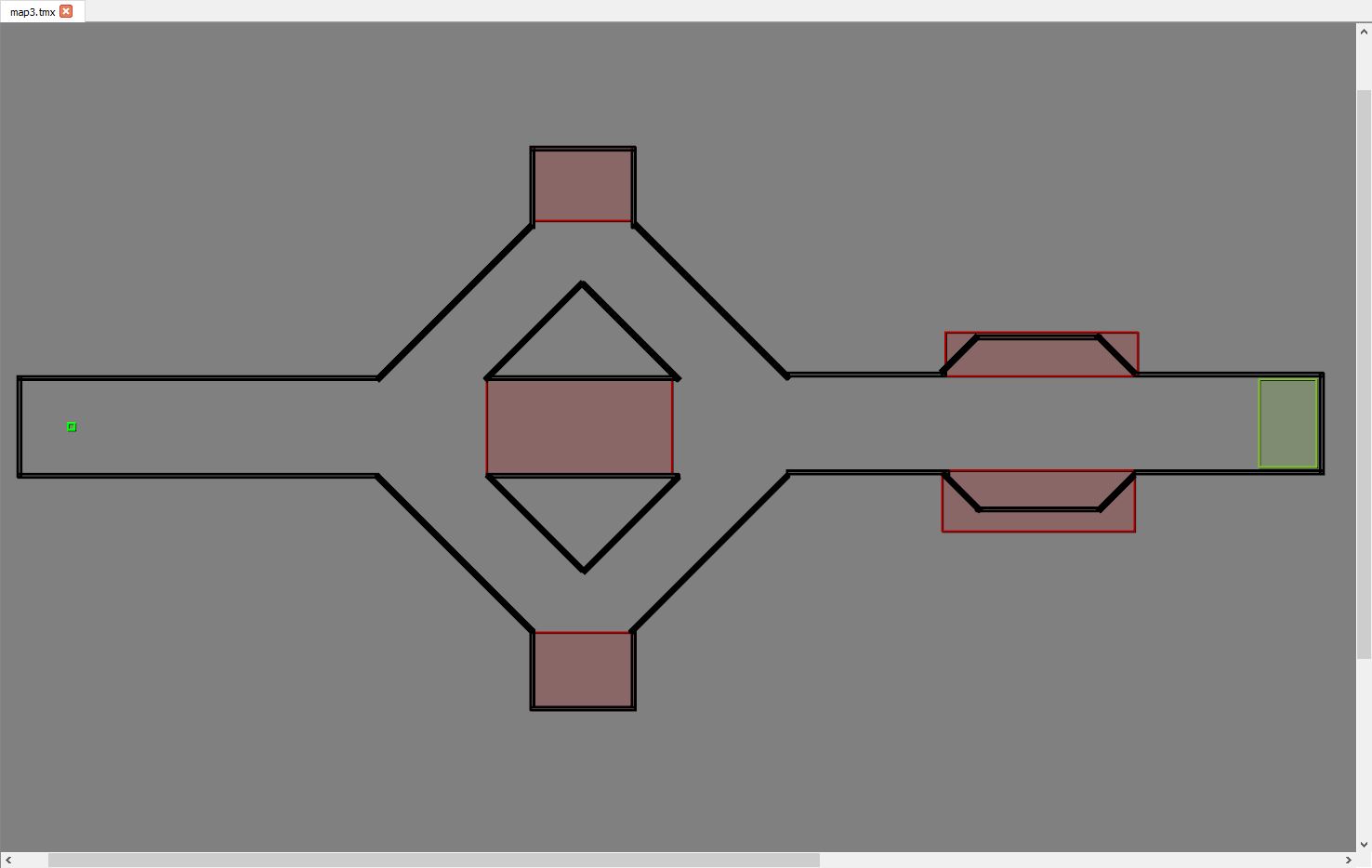
1. Blind



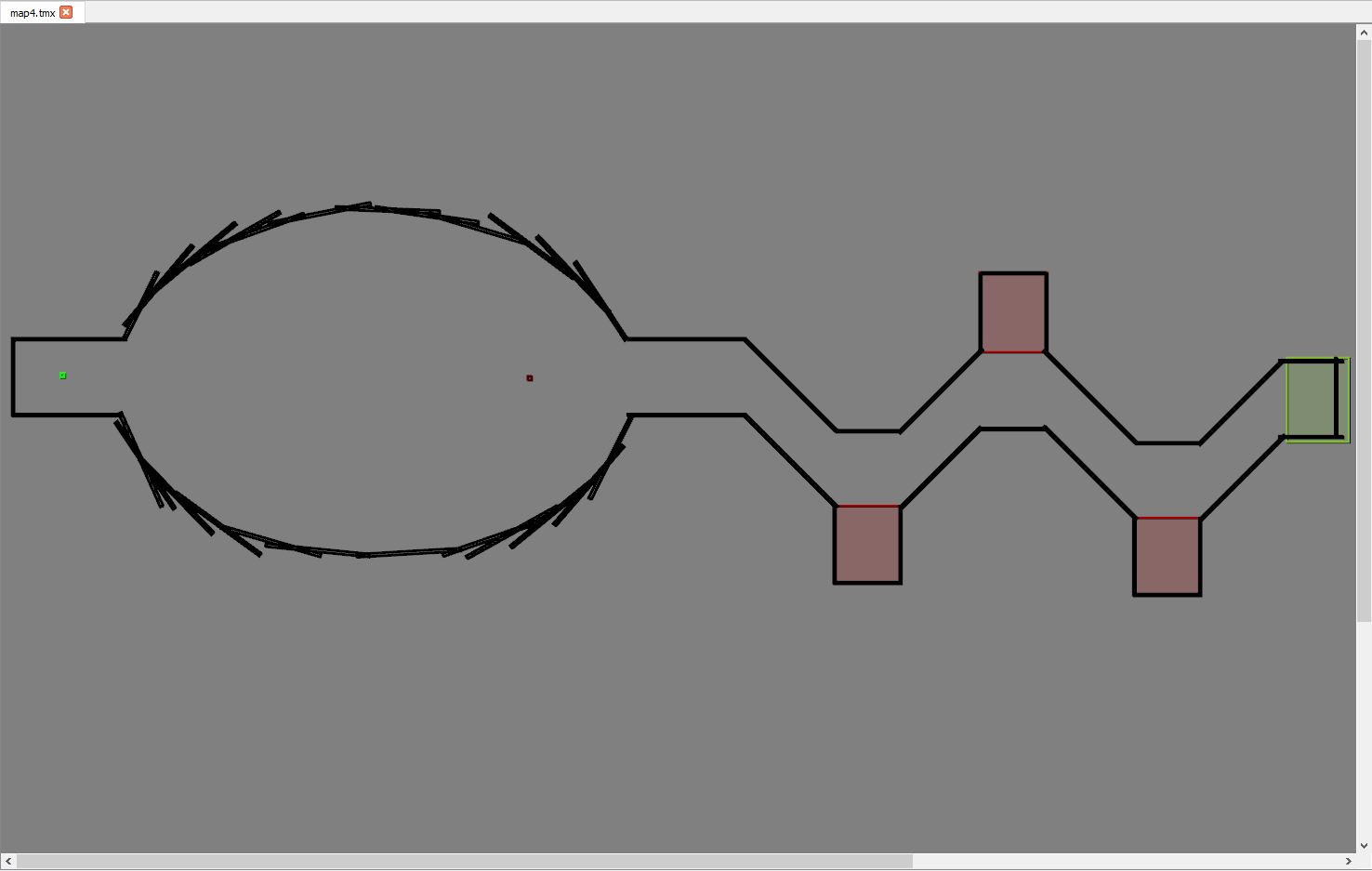
1. Scape



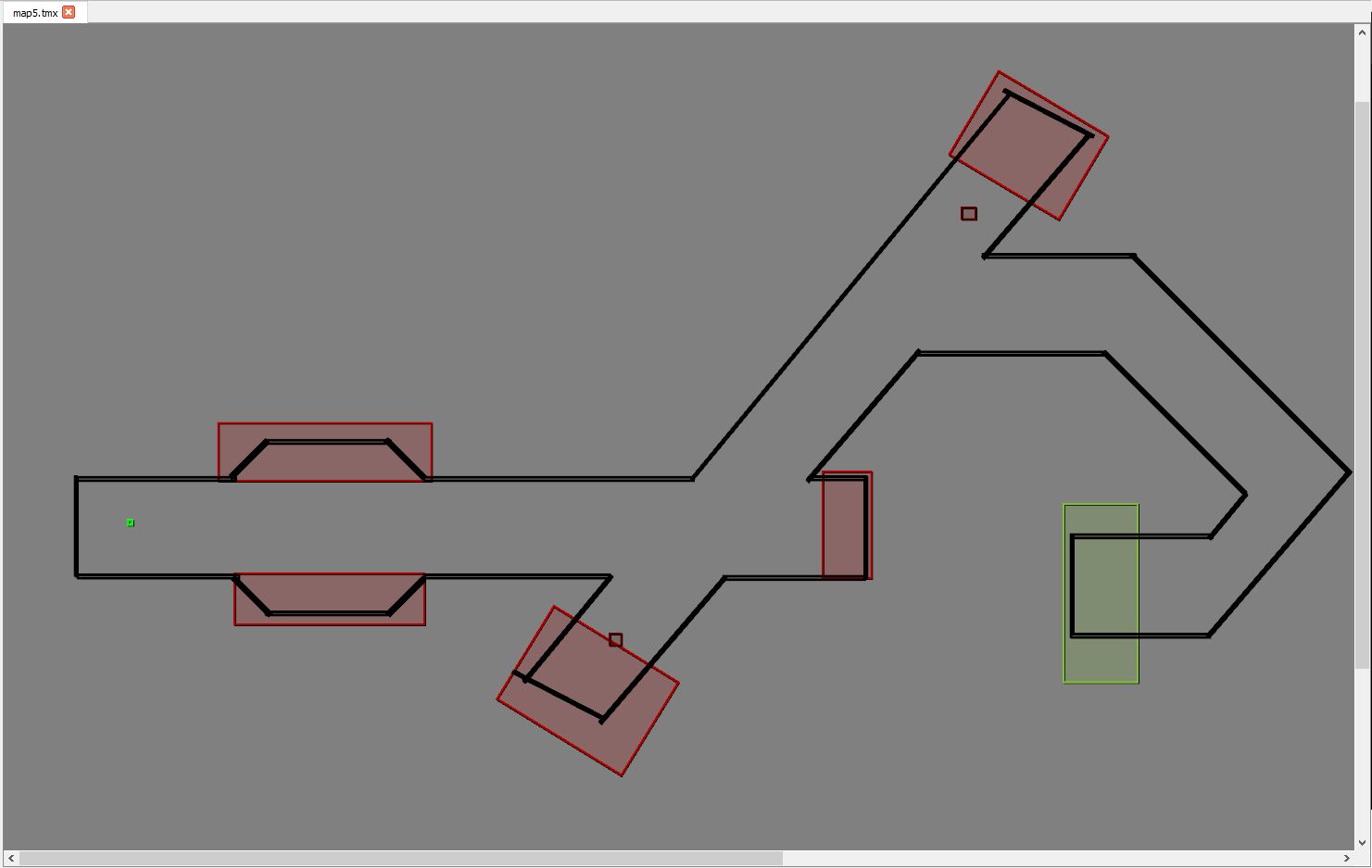
1. Death



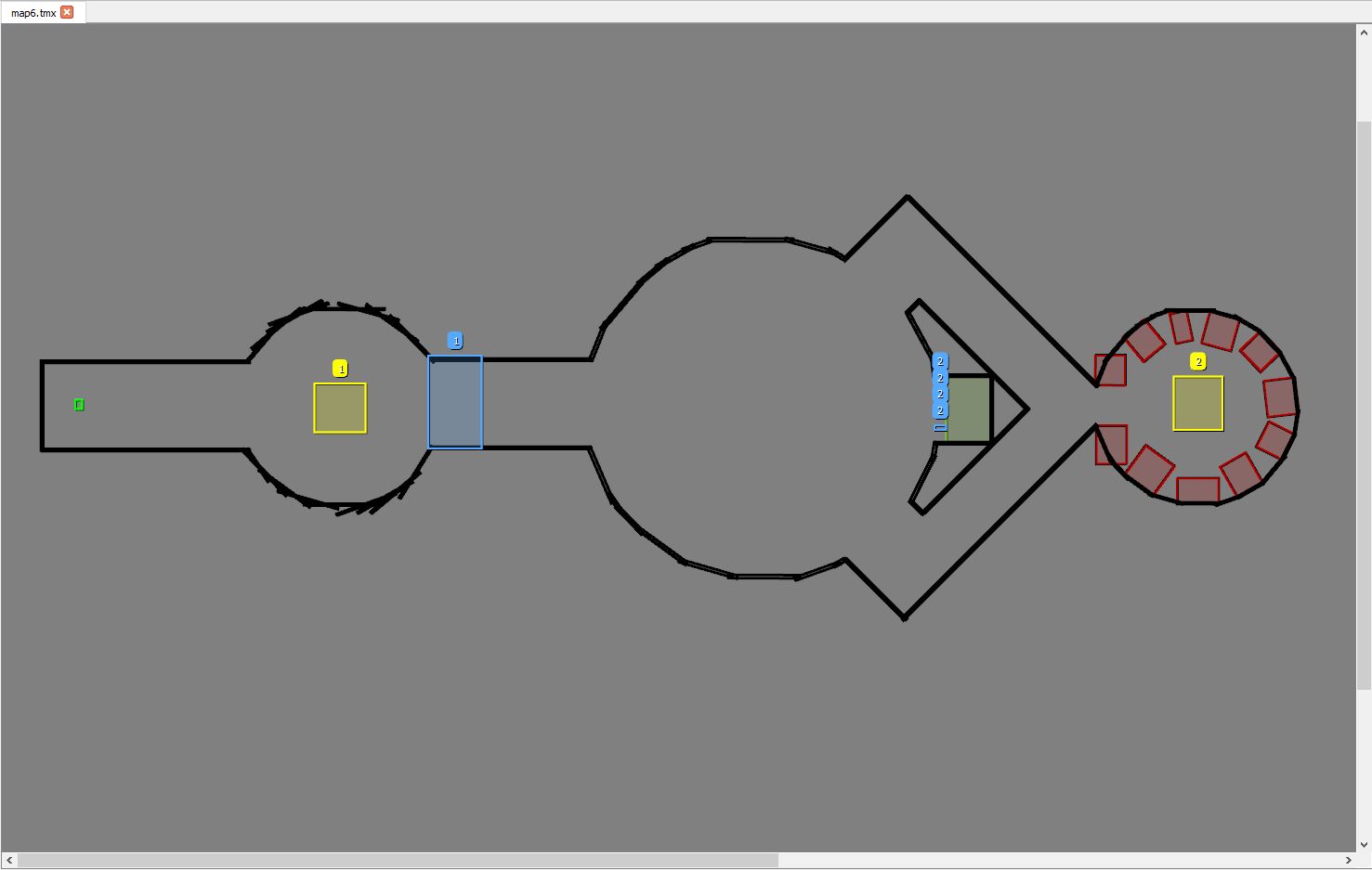
1. Fear



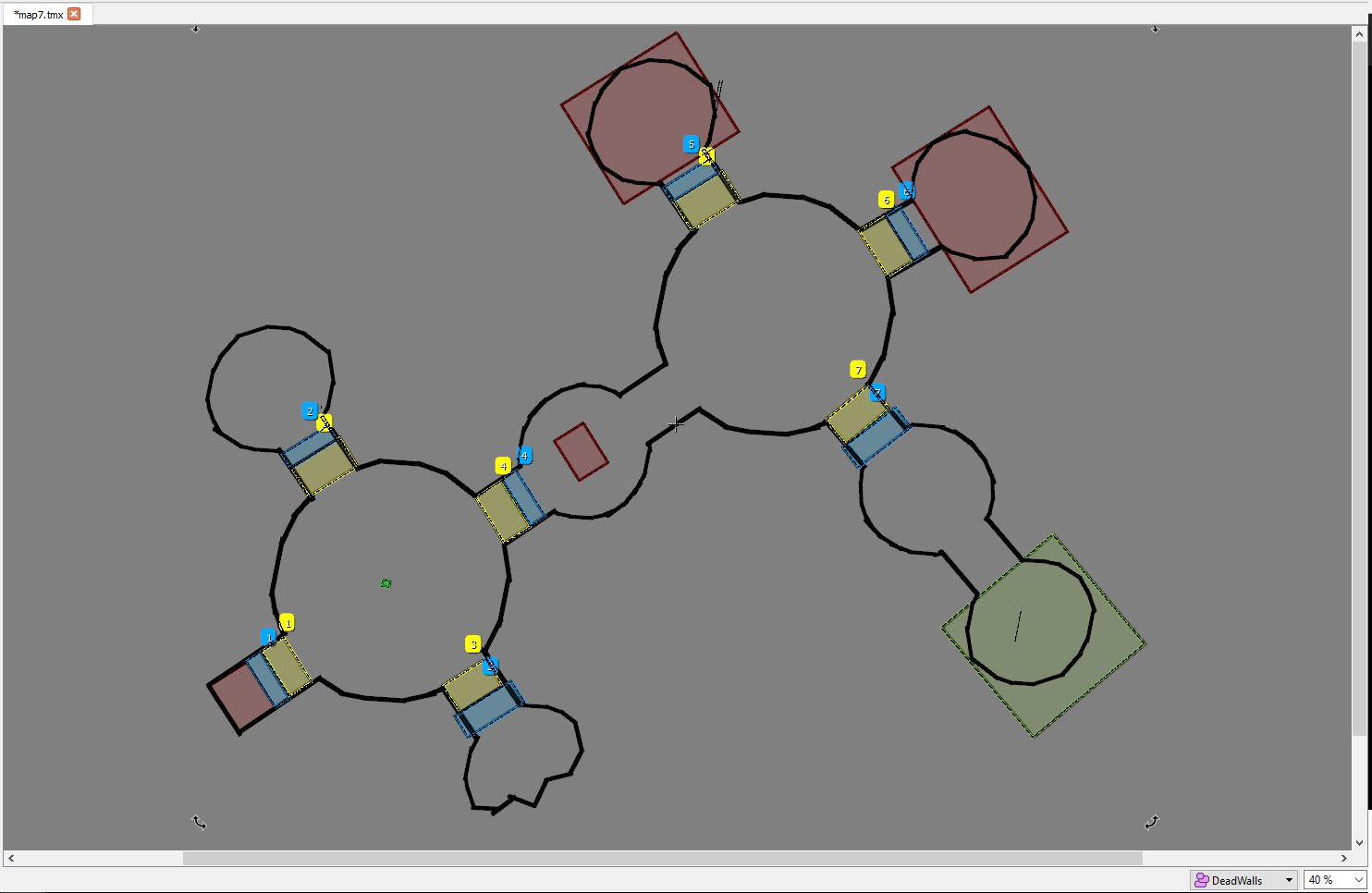
1. Hunt



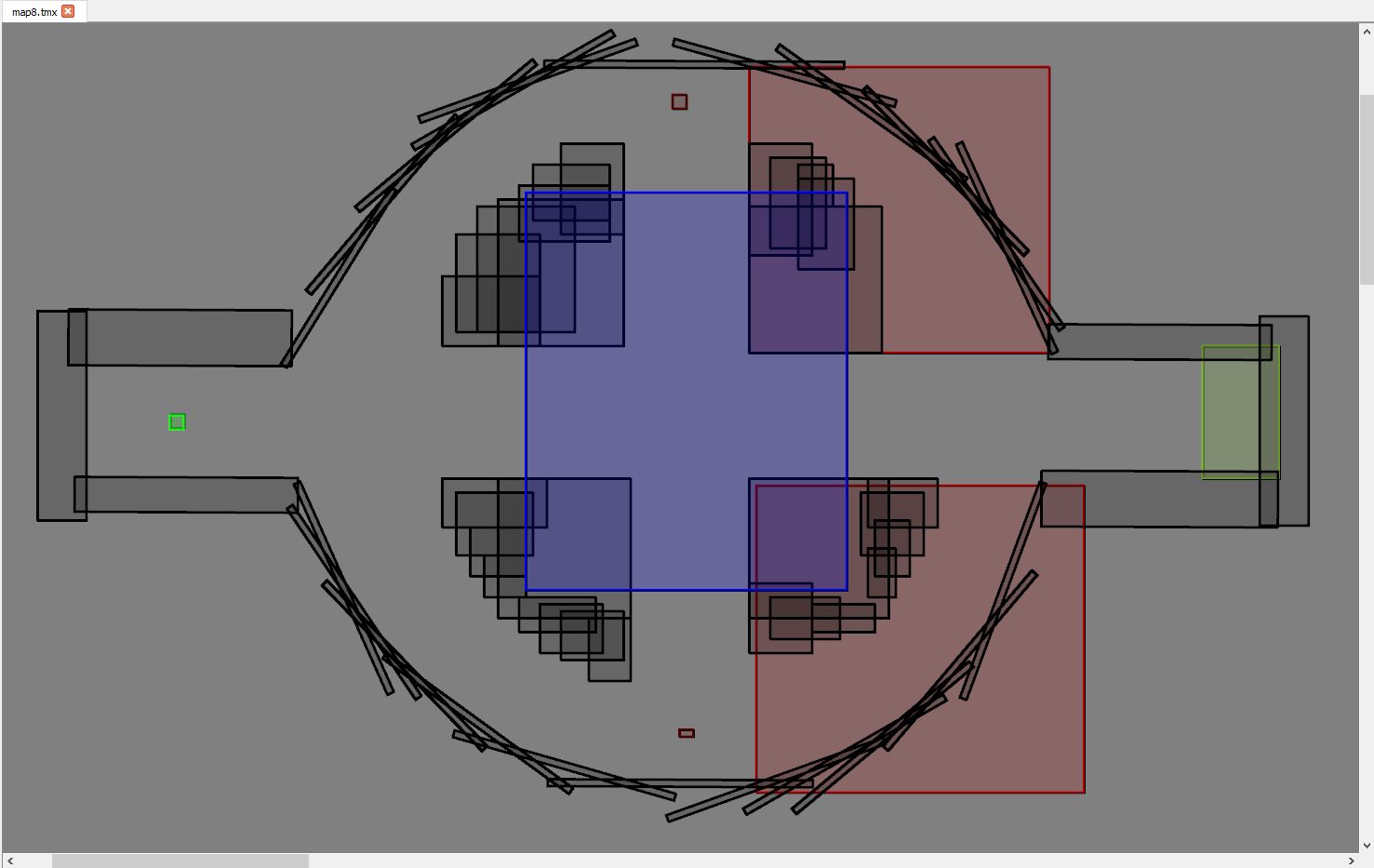
1. Calm



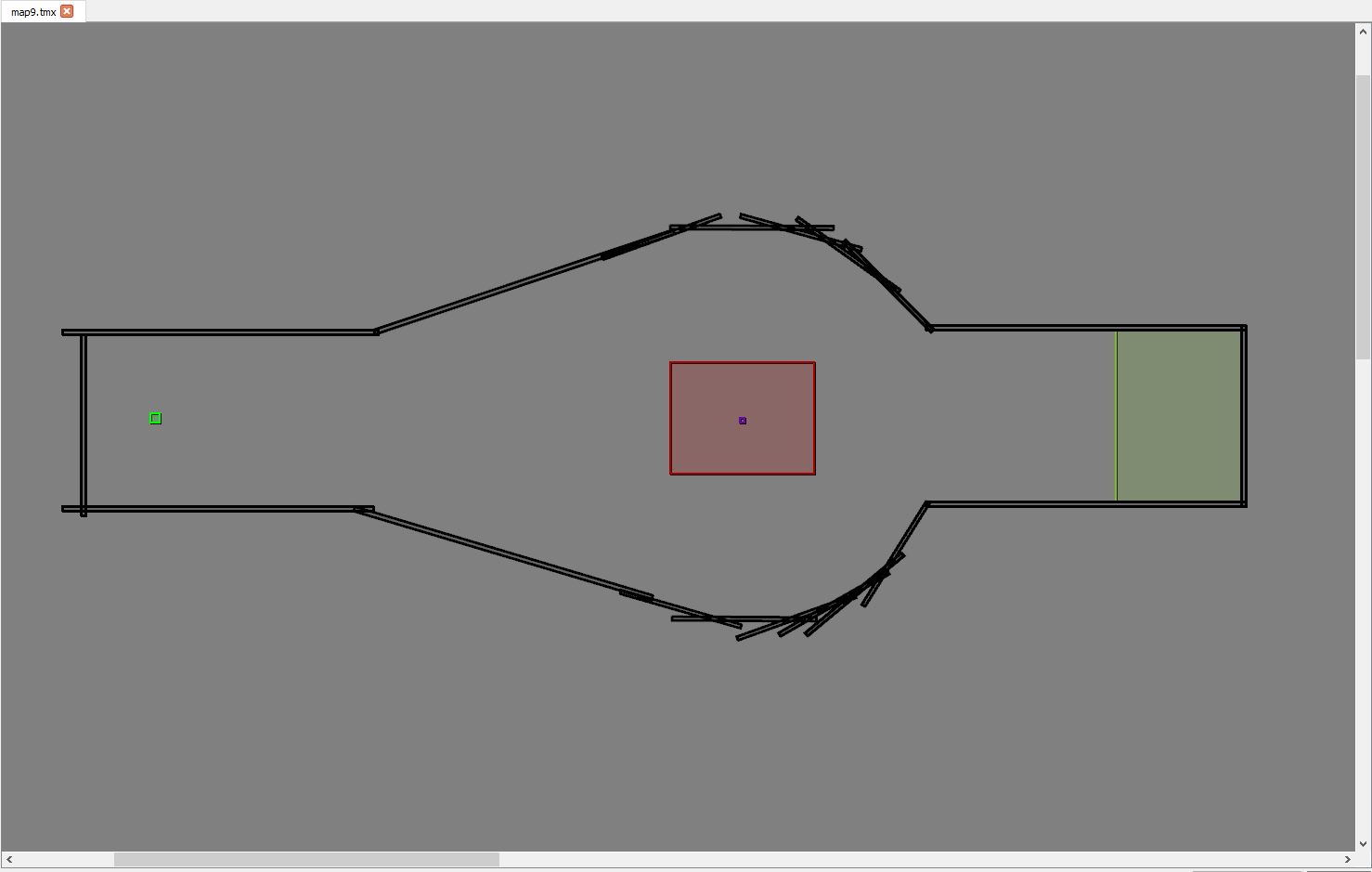
1. Strange



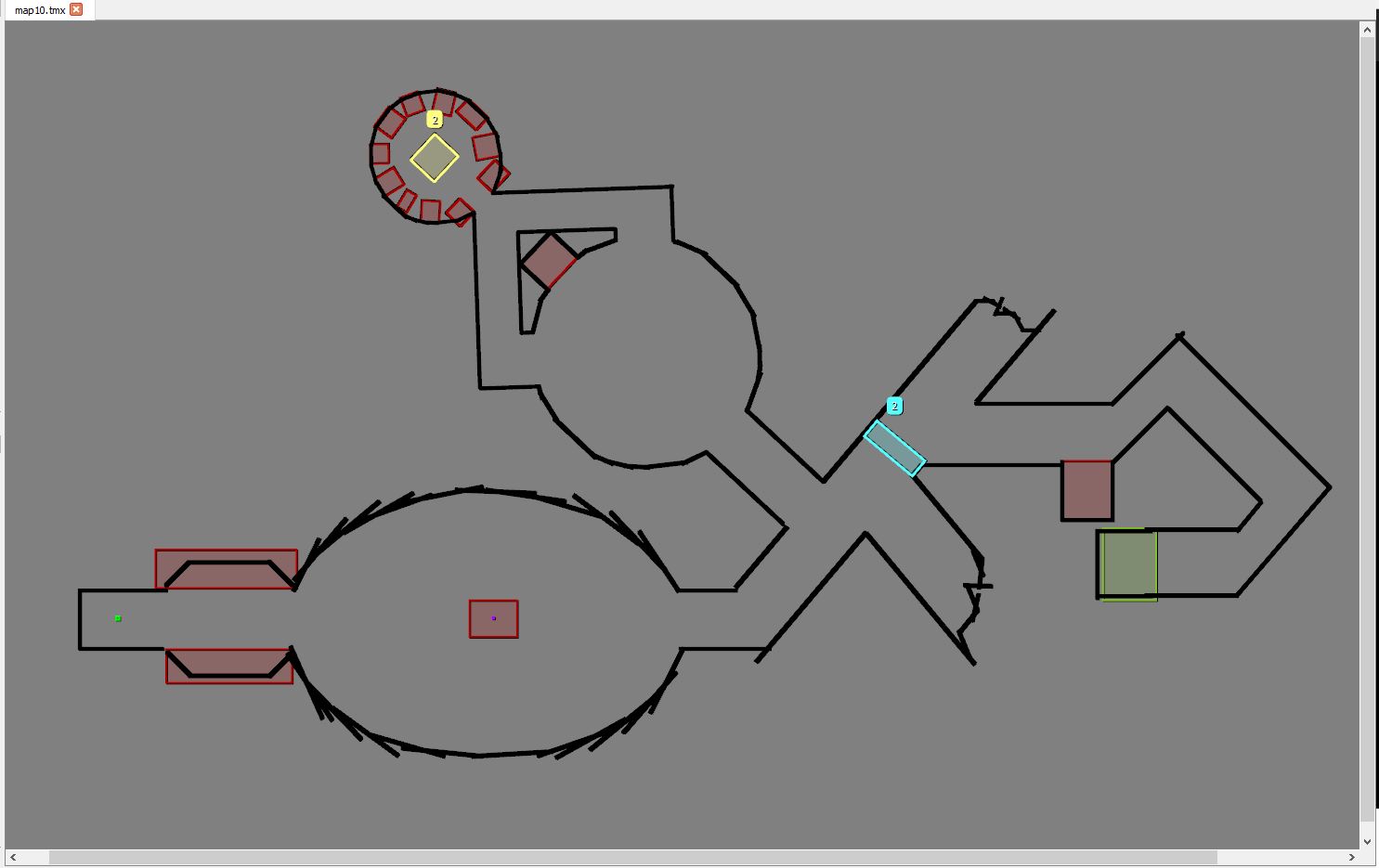
1. Give up



1. Atraction



1. End is nigh

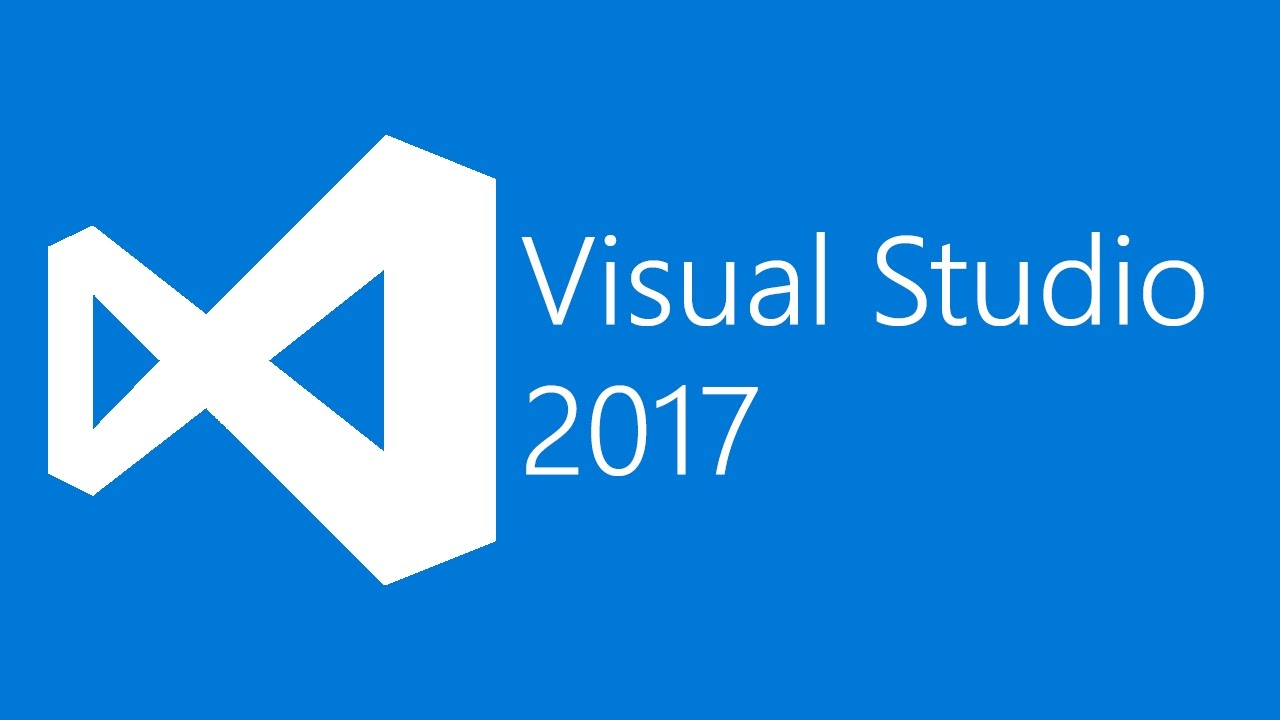


1. Sonido:

* Sonido 2D.
* Sonido Visual

1. Herramientas:

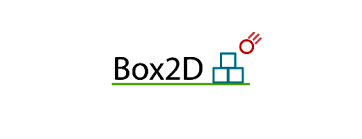
* Visual Studio.

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. 

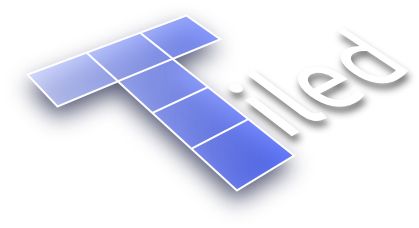
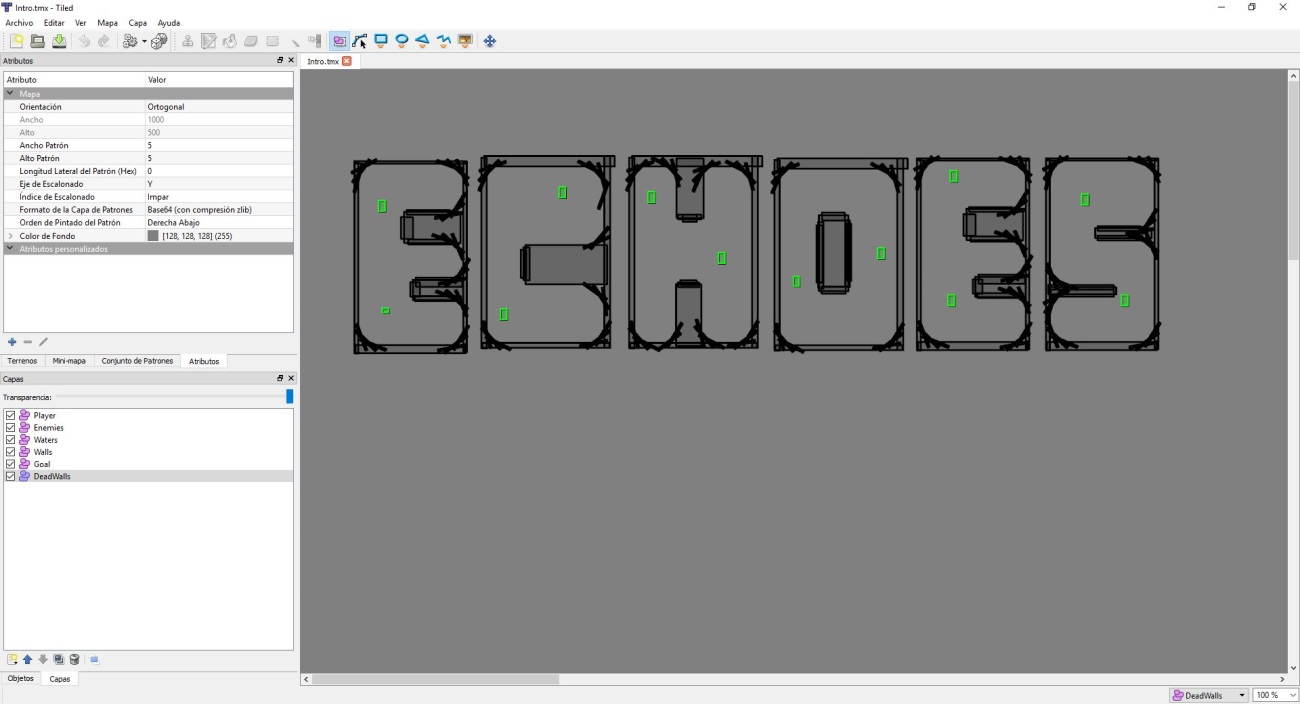
* SFML.

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. 

* Box2D.

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.

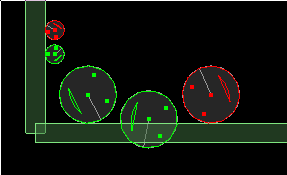
Con Audacity se han modificado y ajustado los audios capturados para el juego para adaptarlos a las necesidades que teníamos al realizar un juego de terror, como ecos o reverberaciones. 

* GitHub y SourceTree

Como repositorio y sistema de control de versiones se ha utilizado una cuenta en GitHub y el entorno grafico SourceTree de Atlassian. 

1. Físicas:

* 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. 

1. Motor Visual:

* 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.

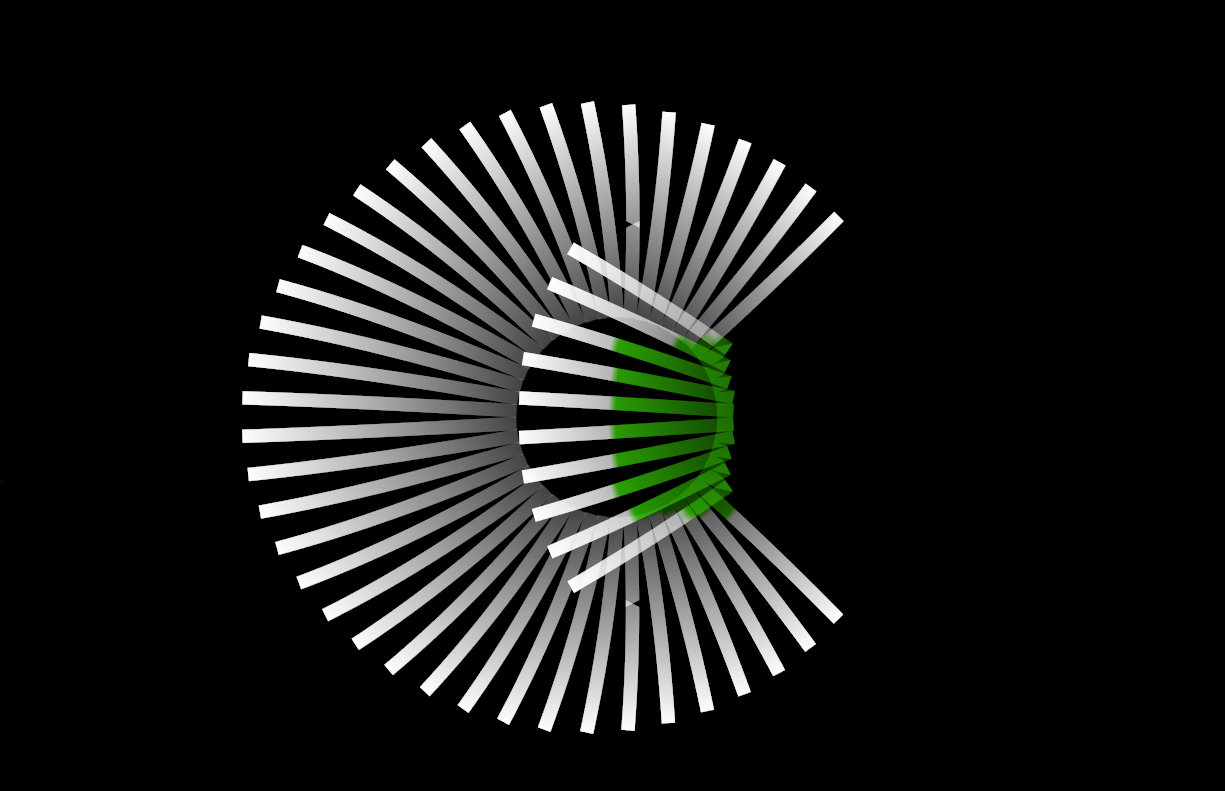
Esta solución a pesar de ser funcional, no daba el resultado final que se quería conseguir, además de ser muy costosa en tiempo de ejecución.

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.



1. Patrones de diseño:

* 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.

* Façade.

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.

1. Lenguaje:

* C++11