

Juego de plataformas en 3D

Ander Orbegozo Igartua

May 22, 2016

Contents

1	Objetivos	3
1.1	Motivación personal	3
1.2	Objetivos específicos	3
2	Plan de trabajo	4
3	Funcionamiento del juego	4
4	Código	6
4.1	Estructura del código javascript	6
4.1.1	loader.js	6
4.1.2	avatar_controll.js	6
4.1.3	spawn.js	6
4.1.4	base_level.js	6
4.1.5	level1-4.js	7
4.1.6	mobile_platform	7
4.1.7	fall_platform	7
4.1.8	script	7
4.2	Diseño de niveles	7
4.2.1	Nivel 1	7
4.2.2	Nivel 2	8
4.2.3	Nivel 3	9
4.2.4	Nivel 4	10
4.3	Implementaciones destacadas	11
4.3.1	Detección de colisiones	11
4.3.2	Movimiento del personaje	11
4.3.3	Movimiento de la cámara	11
4.3.4	Plataformas móviles	11
4.3.5	Spawn	12
4.3.6	Gravedad y salto	12
4.3.7	Texturas	12
4.3.8	Movimiento de la lava	12
4.3.9	Musica y sonidos	12
4.3.10	Mensajes	13
4.3.11	Carga de los modelos	13
5	Recursos externos	13
5.1	Modelos	13
5.2	Texturas	13
5.2.1	BumpMap y NormalMap	13
5.3	Efectos de sonido y música	14
5.3.1	Música de fondo de los niveles	14
5.3.2	Efectos de música	14

1 Objetivos

El objetivo de este proyecto es realizar un juego de plataformas en 3D usando la librería [Three.js](#).

El código esta disponible en github:

<https://github.com/9and3r/Grafikoak-Proiekta>

Se puede descargar el proyecto comprimido desde el siguiente enlace:

<https://github.com/9and3r/Grafikoak-Proiekta/archive/master.zip>

1.1 Motivación personal

La idea del juego viene de un juego para la Playstation 1 el cual jugaba de pequeño. El juego es [Crash Bandicoot 3: Warped](#).

Este videojuego es un plataformas en 3D. A diferencia de otros plataformas la cámara no puede girar libremente ni nos permite girarla a los jugadores. Esto puede parecer una limitación de primeras, pero yo lo considero una buena decisión porque la cámara no molesta al realizar los saltos. En otros juegos donde la cámara gira según hacia donde nos movemos dificulta realizar los saltos de forma precisa mientras la cámara esta girando.

En el siguiente [video](#) se puede ver el funcionamiento de Crash Bandicoot 3.

1.2 Objetivos específicos

Se ha dividido el objetivo principal en los siguientes objetivos mas pequeños:

- Cargar modelos y texturas *
- Mover el personaje al pulsar teclas *
- Colisionar el personaje con paredes *
- Mover el personaje hacia abajo automáticamente (Gravedad) y pararlo al colisionar *
- Hacer el salto del personaje al pulsar la tecla *
- Creación y diseño de niveles *
- Crear plataformas que se mueven
- Música y efectos de sonido
- Enemigos que se mueven
- Matar los enemigos al pisarlos

Los objetivos marcados con * son aquellos que se consideran necesarios para realizar un juego funcional. Los otros objetivos se consideran secundarios.

2 Plan de trabajo

El plan de trabajo se ha dividido en semanas, aunque se estima que puede haber cambios significativos al tener poca experiencia en gráficos 3D y al usar una nueva herramienta.

- Semanas del 8 al 22 de febrero: Introducción a Three.js. Probar la herramienta sin realizar nada específico para este proyecto.
- Semana del 22 al 29 de febrero: Crear geometrías simples y colocarlas. Colocar la cámara en distintas posiciones. Cargar texturas y colocarlas a geometrías.
- Semana del 29 de febrero al 7 de marzo: Mover una geometría usando el teclado.
- Semana del 7 al 14 de marzo: Seguir la geometría con la cámara.
- Semanas del 14 al 28 de marzo: Mover una geometría usando el teclado pero teniendo en cuenta la dirección a la que esta mirando la cámara.
- Semanas del 28 de marzo al 11 de abril: Colisiones de los objetos.
- Semana del 11 al 18 de abril: Cargar modelos externos. Hacer que el personaje caiga si no colisiona y el salto del personaje.
- Semana del 18 al 25 de abril: Crear plataformas que se mueven.
- Semana del 25 de abril al 2 de mayo. Creación y diseño de niveles.
- Semana del 2 al 9 de mayo: Pulir detalles y añadir extras (Música, efectos de sonido...)
- Semana del 9 al 16 de mayo: Terminar y corregir los fallos y finalizar la documentación.

3 Funcionamiento del juego

Para empezar tenemos que abrir el fichero index.html con un navegador. Se ha comprobado que funciona correctamente en Chrome y en Firefox. Para que los recursos puedan cargarse es necesario comprobar que nuestro navegador [esta configurado correctamente](#).

El juego consta de 4 niveles que tenemos que superar. Cada nivel empieza en un spawn (Imagen 1). Nuestro objetivo es llegar al otro spawn (Imagen 1), generalmente situado al fondo del nivel, para superar y pasar al siguiente nivel (Para finalizar es necesario colocarse encima y centrado del cilindro del spawn). Para ello usaremos los siguientes controles:

- W: mover hacia delante

- A: mover hacia la izquierda
- D: mover hacia la derecha
- S: mover hacia atrás
- espacio: Saltar

Hay que tener en cuenta que los movimientos son relativos a la dirección de la cámara y no del personaje, por lo que la rotación del personaje no afecta en el control del movimiento.

En los niveles nos encontraremos diferentes obstáculos o problemas para llegar al final del nivel. Si por alguno de estos morimos, el juego empezara por el principio de ese nivel. Podemos morir por caer al vacío o a la lava y también podemos morir aplastados.

Para facilitar la prueba de los niveles tanto al programar y testar se puede pasar al siguiente nivel sin finalizarlo metiendo el código `nextLevel()` en la consola del navegador.

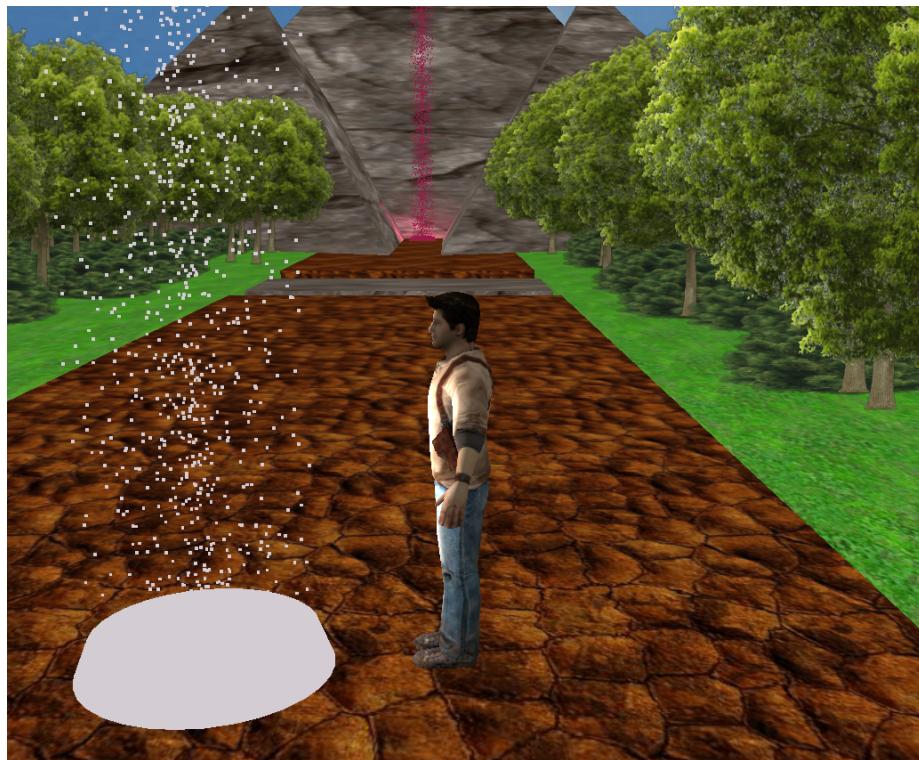


Figure 1: El spawn del comienzo y el spawn del final

4 Código

Hay tres carpetas y dos ficheros en el proyecto:

- Carpeta js: Incluye los ficheros javascript desarrollados por mi.
- Carpeta libs: Es el código descargado desde [el repositorio de Three.js](#).
- Carpeta recursos: Se encuentran las imágenes (texturas), sonidos y el modelo del personaje.
- Fichero style.css: Este fichero se encarga de dar estilo. Por ejemplo al mensaje del nivel o el mensaje del game over.
- Fichero index.html: Este fichero es el que debemos de abrir para comenzar el juego.

4.1 Estructura del código javascript

El código javascript se ha dividido en distintos módulos:

4.1.1 loader.js

Este fichero se encarga de cargar todas las texturas necesarias antes de empezar el juego. De este modo cuando comienza el juego todo está cargado y los objetos no van apareciendo según se van cargando sino que aparecen desde el primer momento. Además, también crea algunos materiales. Todos los recursos (texturas, modelo, materiales, sonidos) quedan guardados en listas y se han creado métodos para facilitar el uso de ellos desde otros scripts.

4.1.2 avatar_controll.js

Se encarga de comprobar si es posible mover el personaje, moverlo en caso de que sea posible y se haya pulsado las teclas de mover y también se encarga de gestionar el movimiento de la cámara para seguir al personaje. También permite cambiar el giro de la cámara.

4.1.3 spawn.js

Es el objeto que se usa al principio y final de cada nivel. Se trata de un cilindro, partículas que giran y una fuente de luz. También se encarga de llamar al método para pasar al siguiente nivel cuando el personaje se pone encima del cilindro.

4.1.4 base_level.js

Incluye BaseLevel. Esto es la representación abstracta de un nivel. Tiene como atributo la escena. Se encarga de llamar a los métodos para mover los objetos y el personaje. Comprueba si el personaje ha muerto en caso de caer y gestiona el comienzo y el final del nivel, mostrando o escondiendo los mensajes (Mensaje de muerte, indicador del nivel...).

4.1.5 level1-4.js

Cada uno de ellos representa un nivel. Estos son hijos de BaseLevel por lo que incluyen lo anteriormente comentando. Esto facilita enormemente la creación de nuevos niveles, ya que para ello lo único que hay que hacer es crear una clase que es hijo de BaseLevel y implementar el método onInit añadiendo los objetos a la escena.

En algunos niveles también se ha implementado el método onRender que se ejecuta en cada frame y que realiza la siguiente función:

- Nivel 2: Se mueve la luz a la misma posición que el jugador en el eje z.
- Nivel 3: Se mueve la textura de la lava. Se activa la puerta cuando el jugador se acerca. Se cambia la escala del jugador si se cae a la lava. Se cambia el angulo de la cámara según la posición del personaje.
- Nivel 4: Se cambia la escala del jugador si se cae a la lava. Se cambia el angulo de la cámara según la posición del personaje.

4.1.6 mobile_platform

Esta clase permite crear plataformas que se mueven. Estas tienen en cuenta si tienen que empujar el personaje o incluso aplastarlo en caso de no ser posible empujarlo.

4.1.7 fall_platform

Parecido al anterior. Permite crear plataformas que se mueven. En este caso se empieza a mover cuando el personaje los pisa y van bajando.

4.1.8 script

Este es el script principal. Es el encargado de llamar al loader, de crear los niveles y controlar en que nivel esta en cada momento. También es el encargado de crear la cámara.

4.2 Diseño de niveles

Para crear los niveles se han usado los materiales cargados por el loader. Para evitar que el jugador salga del camino se han usado algunas paredes transparentes. Como se ha comentado anteriormente todos los niveles incluyen dos spawns. Se han creado 4 niveles en total:

4.2.1 Nivel 1

Este nivel tiene una esfera gigante con la textura del cielo. El camino es recto con trozos de tierra y una plataforma que se mueve en vertical. A los lados se ha colocado la hierba y arbustos y arboles en posiciones aleatorias. Al final se han colocado tres pirámides usando la geometría cylinder de Three.js.

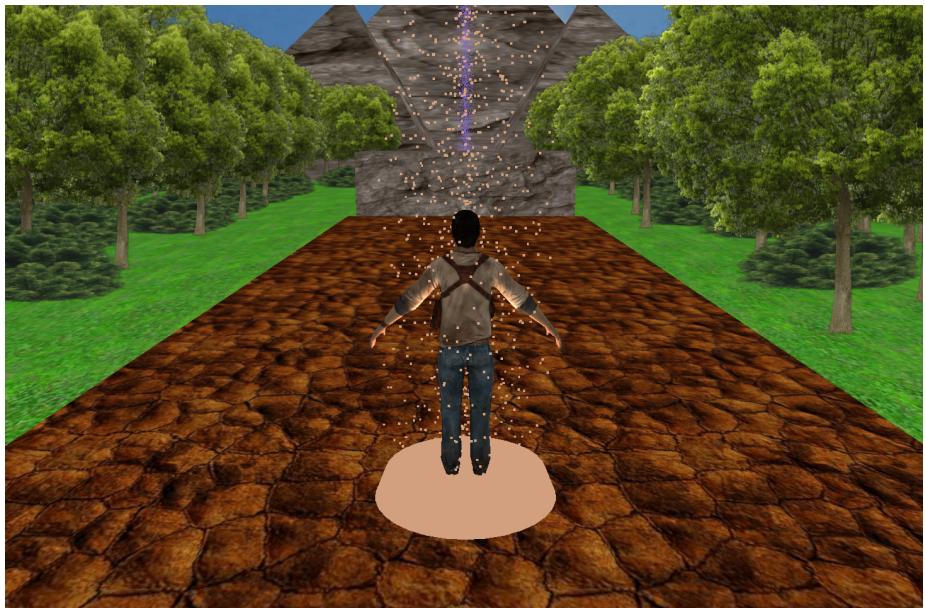


Figure 2: Nivel 1

4.2.2 Nivel 2

El nivel esta rodeado de paredes y hay una plataforma que se mueve en vertical que puede aplastar al personaje. También hay una plataforma que se mueve en horizontal para poder llegar al final. Este escenario es muy oscuro para añadir dificultad. Una luz seguirá al personaje en el eje z para ir iluminando la zona que este cerca.



Figure 3: Nivel 2

4.2.3 Nivel 3

Es el nivel mas complejo. Incluye una habitación con puertas que se activan cuando el personaje se acerca. Estas puertas pueden aplastar al personaje. Al poco de pasar la puerta la camara gira automaticamente. A los lados tenemos cascadas de lava y debajo también hay lava. Para llegar al final hay una plataforma con movimiento horizontal y otra con vertical.



Figure 4: Nivel 3

4.2.4 Nivel 4

El final se encuentra en un volcán situado en el fondo. Para llegar tenemos que superar unas plataformas que bajan según cuando el personaje los pisa sin tocar la lava. Cada plataforma es un poco más rápida que la anterior por lo que la dificultad aumenta según vamos avanzando.

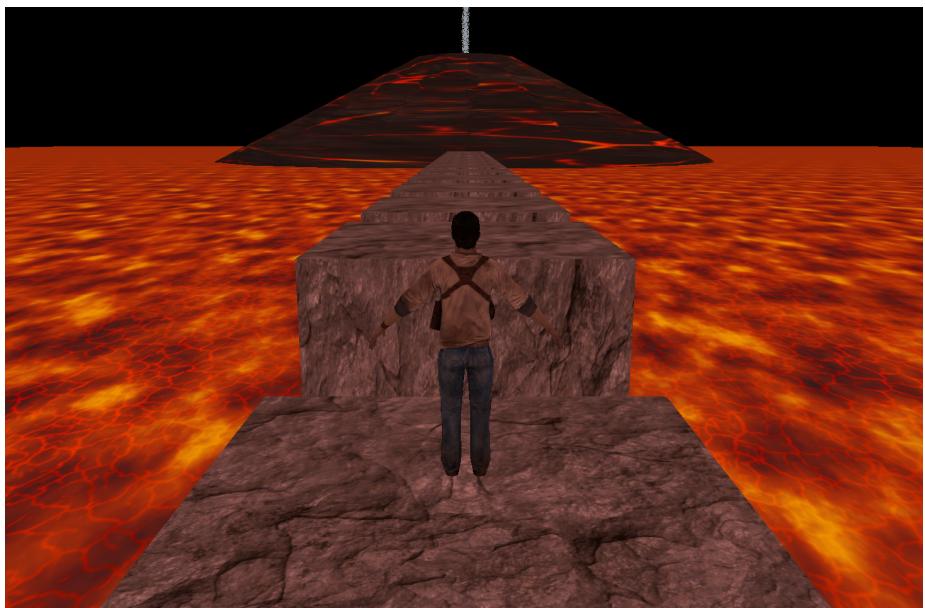


Figure 5: Nivel 4

4.3 Implementaciones destacadas

Algunos de las implementaciones tienen un mayor peso en el proyecto por lo que se explican en este documento a pesar de estar comentadas en el código:

4.3.1 Detección de colisiones

Para detectar las colisiones se han usado los [raycasts](http://webmaestro.fr/collisions-detection-three-js-raycasting/). Para hacerlo me he basado en la página <http://webmaestro.fr/collisions-detection-three-js-raycasting/>. El funcionamiento se basa en elegir algunos puntos alrededor del personaje y realizar un raycasting desde esas posiciones hacia la dirección que queremos mover con la distancia que vamos a mover el personaje. En el caso de que el raycasting nos devuelva algún objeto hay colisión.

4.3.2 Movimiento del personaje

Para mover el personaje con las teclas se detecta cuando una tecla se ha presionado y cuando ha dejado de presionarse. Mientras la tecla este presionada se mueve el personaje si no hay colisiones. Uno de los problemas que me he encontrado fue que quería mover el personaje según la dirección de la cámara. Para solucionarlo primero se rota el personaje poniendo con la misma rotación que la cámara. Después se realiza el movimiento. Finalmente se vuelve a restaurar la rotación.

4.3.3 Movimiento de la cámara

Para que la cámara siga al personaje lo coloco en la misma posición que el personaje y con la rotación Y deseada y luego lo muevo desde esa posición para alejarla un poco y se realiza un lookAt al personaje. He puesto dos variables que definen como de alejada está la cámara del personaje. Al cambiar estos valores (Ocurre al finalizar un nivel) la cámara realiza una animación cambiando poco a poco al valor que hemos asignado. Ocurre lo mismo con la rotación Y de la cámara (Se puede comprobar en el tercer nivel).

4.3.4 Plataformas móviles

Las plataformas móviles deben saber si colisionan con el personaje. Para ello he reutilizado la detección del personaje pero pasándole la dirección contraria a la que se mueve la plataforma. De esta forma se puede comprobar si el personaje colisiona con la plataforma y empujarlo. Si al empujar el personaje no es posible moverlo porque hay otro objeto el personaje se muere y se cambia la escala del personaje emulando que ha sido aplastado (Se puede comprobar en el segundo nivel siendo aplastado por la piedra que se mueve en vertical o en el nivel 3 aplastado por la puerta al cerrar).

Al principio se intento calcular la escala del personaje para que entrase entre los dos objetos, pero esto resultó ser muy complicado para programar y se ha optado por hacer una animación que va haciendo la escala más pequeña en cada frame hasta llegar a 0.1. Esto es mucho más fácil de implementar y da un resultado bastante bueno.

4.3.5 Spawn

El spaw esta formado por un luz, un cilindro y partículas. Las partículas van subiendo en cada frame hasta llegar a un punto donde vuelven al principio. El color va cambiando en cada frame.

4.3.6 Gravedad y salto

La gravedad se ha realizado utilizando una variable que indica la velocidad vertical. En cada frame se aumenta este valor. En caso de que estemos colisionando con el suelo se pone a 0. En caso contrario se mueve el personaje hacia abajo. Cuando se pulsa el botón de saltar (espacio) si estamos en el suelo se cambia la variable de velocidad vertical para que el personaje se mueva hacia arriba en los siguientes frames.

4.3.7 Texturas

Al realizar el skybox me he encontrado con un problema. El skybox esta hecho con una esfera gigante con una textura del cielo. El problema es que por defecto la textura se coloca en la parte externa de la geometría por lo que no se visualizaba nada. La solución es sencilla y consiste en indicarle en el material que queremos usar el lado opuesto usando la variable THREE.BackSide:

```
sky.material.side = THREE.BackSide;
```

Otro de los aspectos descubiertos en las texturas es la opción de repetir la textura. Esto permite repetir una textura en una geometría grande. Por ejemplo la textura de la hierba del primer nivel esta realizada de este modo, repitiendo la textura varias veces en una única gran geometría.

4.3.8 Movimiento de la lava

Para conseguir el efecto de la lava en movimiento se cambia en cada frame el [offset de la textura](#). De este modo se consigue el efecto deseado.

El problema es que la textura es compartida por varios materiales por lo que el [offset se cambiaba en todos](#) y no podía cambiarlo individualmente. Para conseguir que algunos fuesen mas rápido o en diferente dirección (por ejemplo en el nivel 4 hay dos lavas. Cada una se mueve en una dirección) se creo una copia de la textura con el método `.clone()`. Al principio esto no funcionaba y la textura clonada no se mostraba. Al final, el problema esta en que el método `clone` no pone `needsUpdate` a `true` por lo que hay que hacerlo manualmente (<http://stackoverflow.com/a/16707284>).

4.3.9 Musica y sonidos

La música y los efectos de sonido se han implementado utilizando los [métodos ofrecidos por HTML 5](#) sin hacer uso de la librería Three.js.

4.3.10 Mensajes

Para mostrar el mensaje del nivel, el mensaje al finalizar el juego o oscurecer la pantalla al morir se han utilizado div que están definidos en el html y se muestran o se esconden cuando es necesario.

4.3.11 Carga de los modelos

He tenido varios problemas al cargar los modelos. Algunos, no los cargaba, otros solo se cargaba la geometría pero no la textura... Al final he encontrado [un convertidor de modelos escrito en python](#) que permite convertir los modelos con formato objt/mtl a json. Usando esta herramienta para convertir los modelos y después cargándolos con JSONLoader no he encontrado ningún problema.

5 Recursos externos

5.1 Modelos

- [Modelo de personaje](#)

5.2 Texturas

- [Textura de roca](#)
- [Textura de tierra](#)
- [Textura de arbustos](#)
- [Textura de hierba](#)
- [Textura de cielo](#)
- [Textura de árbol](#)
- [Textura de lava y piedra con lava](#)

5.2.1 BumpMap y NormalMap

En el caso de la textura de la roca y la roca con lava se han generado las imágenes bumpMap y normalMap utilizando la herramienta [Crazybump](#). Esta herramienta genera automáticamente los mapas usando como entrada una imagen. Ademas permite ajustar valores y te da una previsualización de como será el resultado (6).

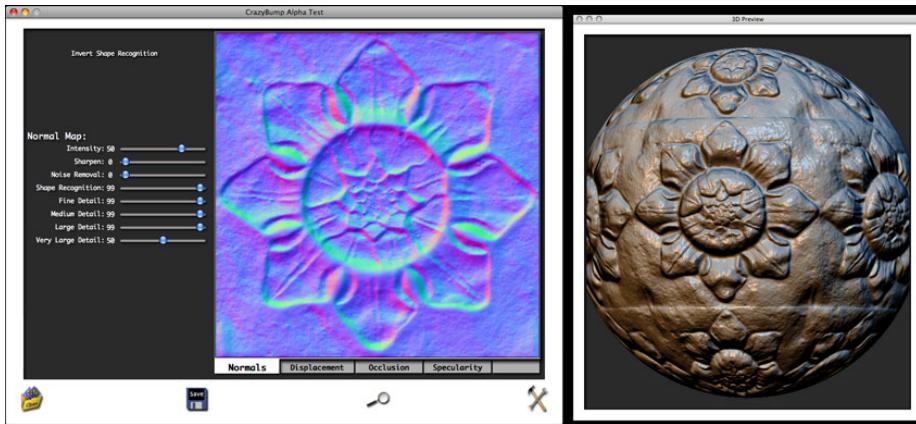


Figure 6: Crazybump

5.3 Efectos de sonido y música

Todos los efectos de sonido y la música utilizadas para este juego han sido descargadas desde [YouTube](#). Para poder descargar la música de los vídeos se ha utilizado la página <http://www.youtube-mp3.org/es>.

5.3.1 Música de fondo de los niveles

- Nivel 1: [Uncharted 3: Nate's Theme 3.0](#)
- Nivel 2: [Crash Bandicoot 3 - Tomb Time, Sphyxinator, Bug Lite Music](#)
- Nivel 3: [Uncharted 3 Soundtrack - Sink Or Swim](#)
- Nivel 4: [Uncharted 4 - Official Main Theme](#)

5.3.2 Efectos de música

- Sonido al entrar al nivel: [Crash Bandicoot Sounds - Warp](#)
- Sonido al finalizar nivel: [Crash Bandicoot Sounds - Crystal Collect](#)
- Sonido al morir: [Uncharted 3: Death Sound Effect](#)