Worm it!

Adrián Termenón Riera
Grado Multimedia
TFG Videojuegos

Consultores: Helio Tejedor Navarro, Jordi Duch Gavaldá

Profesor: ***Joan Arnedo Moreno***

junio de 2016

FICHA DEL TRABAJO FINAL

Título del Trabajo:	*Worm it!*
Nombre del autor:	*Adrián Termenón Riera*
Nombre de los consultores:	*Helio Tejedor Navarro, Jordi Duch Gavaldá*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	*01/06/16*
Titulación o programa:	*Grado Multimedia*
Area del Trabajo Final:	*Videojuegos*
Idioma del Trabajo:	*Español*
Palabras clave:	*Videojuego, Puzle, Android, PC*

Resumen del Trabajo (máximo 250 palabras: *Con la finalidad, contexto de la aplicación, metodología, resultados y conclusiones del Trabajo.*

Worm it! es una adaptación a las tres dimensiones de Worm World, un *freeware* desarrollado por Kevin Ng en el año 1993. Así pues, se trata de un juego tipo puzle en el que se requiere de cierto ingenio para ir superando los distintos niveles que lo componen. La aplicación va dirigida a personas de todas edades y funciona en dispositivos Android y ordenadores con sistema operativo Windows 7 o posterior.

El primer paso ha consistido en planificar el diseño del juego, lo que incluye aspectos como su jugabilidad, la interacción sistema-usuario o el diseño de los niveles. Una vez cerrada la planificación, la siguiente tarea ha sido el diseño de los elementos 3D y 2D, necesarios para generar los escenarios y la interfaz del juego. El proceso de desarrollo se ha llevado a cabo en Unity, un *framework* que permite crear videojuegos multiplataforma. Esto es posible gracias al motor gráfico que incorpora, el cual ofrece unas prestaciones de rendimiento excelentes y una gran versatilidad a la hora de cambiar de entorno.

Más allá del desarrollo, el Trabajo también ha incluido tareas de publicación y promoción, para lo que se ha realizado un pequeño vídeo promocional que puede verse desde la página de Google Play. Además también se ha registrado la aplicación en la plataforma Appszoom con la finalidad de ganar visibilidad.

Abstract (in English, 250 words or less):

This document contains the detailed process carried out during the Degree Final Job in his specialty of Videogames, which is essential in order to obtain the Multimedia Degree at *Universidat Oberta de Catalunya* (UOC).

The main goal of this work is to design and create a videogame: from the initial concept, going through later Unity development until the publication on Google Play platform.

The motivation for this game development comes from the fact that leisure and entertain play a big role in the intelectual development of an individual. This is the reason to choose a puzzle game, which can test the ability of a user to resolve problems in an attractive and amusing way.

Developing the aplication with Unity has given a bunch of benefits such as the ease of working with 3D models, textures, animations and sounds. Moreover, Unity allows to bring the result to many platforms like Android and PC, which have been chosen due to their simplicity to publish and distribute aplications.

The main challenge has been to achieve a good user experience using a touch interface, specially since the action takes place in a 3D environment.

Índice

1. Introducción

1.1 Contexto y justificación del Trabajo

El juego es un aspecto intrínseco del ser humano. De hecho las primeras referencias sobre actividades lúdicas se remontan al año 3000 a.C. en la cuna de la civilización. Así pues, no es de extrañar que su práctica se utilice, en muchas ocasiones, como método de aprendizaje.

En este contexto, adquieren especial importancia los puzles y juegos de ingenio los cuales proporcionan a quien los practica la oportunidad de desarrollar sus habilidades cognitivas.

En el caso de **Worm it!**, el usuario debe visualizar y analizar el problema antes de buscar una solución. Un esfuerzo que se ve disimulado por la estética inocente y despreocupada del producto. Y esto es, ante todo, lo que que se espera de un juego, que sea entretenido y que aporte valor.

1.2 Objetivos del Trabajo

- Prototipar y desarrollar un videojuego tipo puzle 3D mediante el uso del software Unity.

- Planificar y documentar todo el proceso de desarrollo.

- Alcanzar un nivel profesional en el uso de las herramientas necesarias para llevar a cabo el Trabajo desde los ámbitos del diseño, la programación y la gestión del proyecto.

- Poner en práctica diferentes aptitudes adquiridas durante el Grado, como arquitectura de la información, diseño gráfico, programación, integración digital de contenidos y en definitiva, todo lo que pueda ser de utilidad para la elaboración de un videojuego.

- Diseñar una jugabilidad sencilla y una curva de dificultad coherente que, entre ambas logren enganchar al usuario y lo lleven a resolver situaciones más complicadas.

- Conseguir que el *gameplay* sea atractivo y adictivo para personas de todas las edades.

- Crear versiones del juego para escritorio y *smartphone* y publicar esta última en Google Play.

- Realizar un vídeo que ilustre todo el proceso de desarrollo.

1.3 Enfoque y métodos seguidos

Mi idea inicial era hacer videojuego de tipo puzle basado en el clásico Lemmings, donde el jugador debe guiar a uno o más personajes hasta la meta, evitando ciertos obstáculos, para lo cual es indispensable el uso de diferentes ítems a su disposición.

No obstante, el guión de interacción de **Worm it!** es bastante diferente. Así pues, en Lemmings y otros juegos similares el usuario debe ir posicionando los ítems a medida que los personajes van avanzando por el escenario. En cambió en **Worm It!**, el usuario dispone de todo el tiempo que desee para estudiar cada problema y gestionar sus recursos con la finalidad de superar el nivel.

Tras llevar a cabo un estudio inicial de algunos videojuegos con mecánicas similares, decidí tomar como referencia y punto de partida uno de ellos. De esta manera, **Worm it!** es una adaptación a las tres dimensiones de un *freeware* llamado Worm World desarrollado por Kevin NG en el año 1993. El autor declara explícitamente su conformidad ante cualquier proyecto desarrollado a partir de la idea original.

La decisión de trabajar a partir de una idea prediseñada me ha permitido centrarme en aspectos más propios del desarrollo. Pero sobre todo, he tenido la oportunidad de enfrentarme cara a cara con una de las quimeras más ubicuas en el mundo de los videojuegos, como es la tarea de adaptar un juego 2D a 3D, manteniendo su esencia e incluso mejorando su jugabilidad.

1.4 Planificación del Trabajo

La primera tarea ha consistido en al redacción del GDD o documento de diseño del juego. Aquí han quedado definidos los objetivos del juego, los ítems y objetos que componen cada nivel, el personaje y la interfaz de usuario. En cambio, otros aspectos como la música y los efectos sonoros se han dejado para más adelante.

De esta manera, y siguiendo la planificación inicial, los siguientes pasos han consistido en el diseño y modelado de los elementos 2D y 3D, la implementación de dichos modelos en Unity, la escritura del código necesario para recrear la mecánica de juego, la inserción de música y sonidos y, por último, la depuración del código con el objetivo de eliminar posibles errores en el comportamiento del personaje o en la interacción del usuario.

El último paso del proceso de Trabajo ha consistido en la publicación y promoción de la aplicación. Para su publicación se ha elegido la plataforma Google Play y para su promoción se ha realizado un pequeño vídeo donde se pueden ver las principales características del producto

1.5 Breve sumario de productos obtenidos

- El producto en si, la aplicación, tanto para Android como PC
- Dos vídeos cortos explicando las principales características.
- Un vídeo largo donde se puede ver el proceso llevado a cabo.
- Un vídeo promocional del producto.
- Un documento con la planificación inicial o *GGD*.
- Dos documentos con los pasos seguidos durante el proceso.
- La memoria del Trabajo.

1.6 Breve descripción de otros capítulos de la memoria

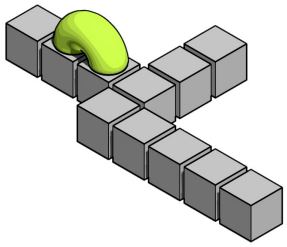
En los siguiente capítulos se explica con más detalle el proceso llevado a cabo para el desarrollo de la aplicación, empezando por el diseño y modelado de los elementos del juego y finalizando con las acciones necesarias para su publicación y promoción en Google Play.

2. Resto de capítulos

2.1 Planteamiento inicial y mecánica del juego.

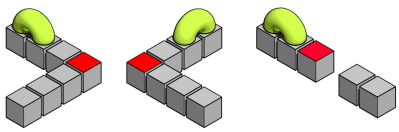
Basándose en la idea de Worm World, el personaje principal, es decir el gusano, se mueve y se manera idéntica a como lo haría el clásico muelle multicolor de juguete.

Esta característica fuerza al personaje a desplazarse distancias discretas, lo que justifica la interacción del mismo con un entorno cuadriculado por el que deberá avanzar hasta llegar a la meta. La disposición de dicho entorno respecto al jugador es una vista isométrica tal y como se puede ver en la siguiente imagen.



Por defecto el personaje siempre avanza en linea recta con lo que es necesario utilizar ciertos ítems con la finalidad de que no se salga del recorrido, lo que supondría la muerte del gusano. De está manera, el usuario tiene a su disposición tres ítems diferentes que podrá poner sobre cualquier casilla del recorrido y que se activarán cuando el gusano pase sobre dicha casilla. Cada uno de estos ítems permite al gusano realizar las siguiente acciones.

- Girar 90 grados hacia la derecha.
- Girar 90 grados hacia la izquierda.
- Crear un puente y salvar una caída.



Además, el gusano podrá interaccionar en cierta medida con el escenario. Es decir, si el gusano se topa con un bloque de frente se puede dar dos casos:

- Que el bloque sea fijo con lo que el gusano da media vuelta y continua su camino por donde había venido.

- Que el bloque sea móvil con lo que el gusano empuja dicho bloque hacía delante hasta que este no pueda moverse por estar obstaculizado su avance por otro bloque móvil o uno fijo.

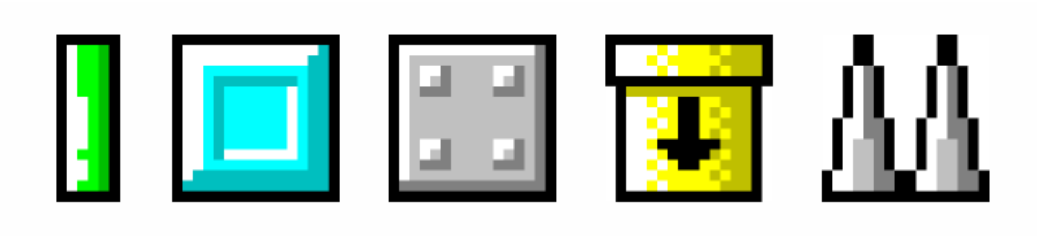
Por último, se puede dar la situación de que el gusano caiga por un hueco, lo que puede terminar de maneras:

- Que la altura entre el nivel donde está el gusano y en el que va a caer sea de un solo bloque. En este caso el gusano desciende sin problemas.

- Que la altura sea superior a un bloque, con lo que el gusano no puede descender con éxito y muere en la caída.

2.2 Diseño y modelado de los elementos.

En lo referente al diseño de los elementos he intentado mantener la esencia del juego original desarrollado por Kevin NG.

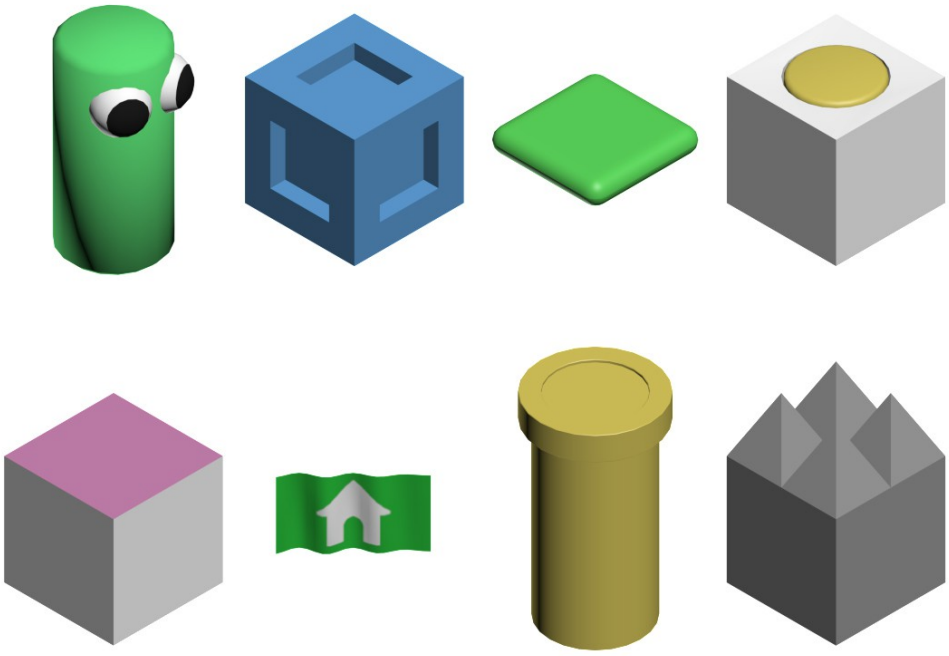


Para su creación he optado por usar el software 3Ds MAX debido a la experiencia que he adquirido en su manejo a lo largo de los últimos años. A la hora de exportar los modelos he optado por el formato *FBX*, mediante el cual ha sido posible importarlos en Unity.

En este punto, me he topado con un primer obstáculo que no tenía previsto. y es que, o bien el formato *FBX* no soporta ciertos modificadores de MAX *(*incluido *point cache*), o bien Unity no es capaz de procesarlos una vez importado el archivo. Así pues, para las animaciones del gusano y algunos elementos me he visto obligado a crear una malla diferente por *frame*. Otra posible solución hubiese sido genera un *morph* e irle añadiendo pose a pose la animación. El resultado no hubiese variado mucho en cuanto al rendimiento, ya que al final, el *morph*, guarda cada posición de la malla como un objeto diferente.

Estos son los elementos que he modelado en 3Ds MAX:

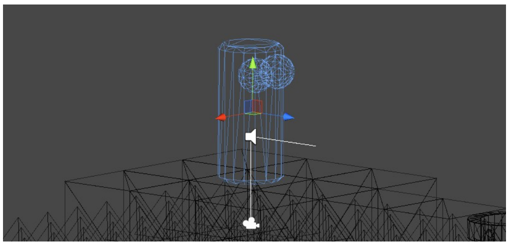
- Gusano: El protagonista del juego.
- Cajas: Representan los bloques móviles del escenario.
- Puentes: Mediante este ítem el gusano puede evitar caídas.
- Botón: En algunos niveles es necesario pulsarlo para que la madriguera se abra.
- Cubos: Son los ladrillos que conforman el escenario, los hay de dos tipos, editables y no editables.
- Bandera: Ayuda al usuario a visualizar mejor la madriguera.
- Madriguera: Llegar hasta ella es el objetivo principal.
- Pinchos: Componen el resto del escenario. Si el gusano cae sobre ellos su muerte esta asegurada.



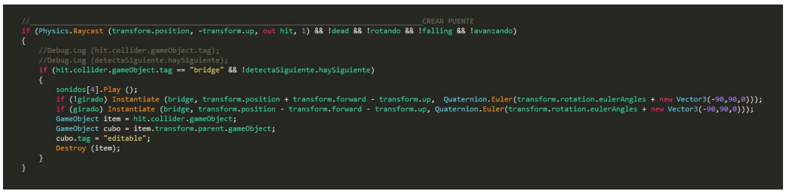
2.3 Programación de la mecánica del juego

Ya con los elementos importados en Unity, el primer paso ha sido establecer qué acciones puede realizar el gusano y escribir los métodos necesarios para su correcta ejecución. Estas acciones son Avanzar una casilla, girar 90º en sentido horario o antihorario, crear un puente en la siguiente casilla, empujar un bloque móvil, colisionar contra un bloque inmóvil, cambiar de dirección, caer, morir y entrar en la madriguera.

Todas estas acciones se han definido en la clase *worm,* la cual se ha aplicado al objeto que contiene el modelo del gusano. Además, para que estas acciones funcionen correctamente ha sido necesario establecer una lógica bastante compleja sobre qué cosas deben suceder primero y cuales después.



Por ejemplo, mediante el método *Raycast()* predefinido en el objeto *Phisics* de Unity, el *GameObject* que contiene al gusano es capaz de detectar si enfrente suyo hay una caja móvil. En caso afirmativo, la clase *worm* manda un mensaje a la clase *box* que, a su vez, chequea si la caja puede avanzar o, por el contrario, está frente a un obstáculo. Tras verificar esto, *box* devuelve un mensaje a *worm* para indicarle si puede o no puede avanzar y empujar la caja.



```
//______________________________________________________________________________CREAR PUENTE
if (Physics.Raycast (transform.position, -transform.up, out hit, 1) && !dead && !rotando && !falling && !avanzando)
{
    //Debug.Log (hit.collider.gameObject.tag);
    //Debug.Log (detectaSiguiente.haySiguiente);
    if (hit.collider.gameObject.tag == "bridge" && !detectaSiguiente.haySiguiente)
    {
        sonidos[4].Play ();
        if (!girado) Instantiate (bridge, transform.position + transform.forward - transform.up,  Quaternion.Euler(transform.rotation.eulerAngles + new Vector3(-90,90,0)));
        if (girado) Instantiate (bridge, transform.position - transform.forward - transform.up, Quaternion.Euler(transform.rotation.eulerAngles + new Vector3(-90,90,0)));
        GameObject item = hit.collider.gameObject;
        GameObject cubo = item.transform.parent.gameObject;
        cubo.tag = "editable";
        Destroy (item);
    }
}
```

Como se puede observar en el proceso que acabo de describir, es muy importante que todas las acciones y verificaciones sucedan en un orden concreto, ya que de lo contrario, podría darse que el gusano reciba la orden de avanzar antes de que la caja le informe de que no puede hacerlo, con lo que se produciría un *bug* en el que el gusano atravesaría la caja.

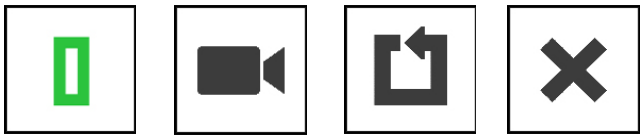
Para facilitar la tarea de diseñar dicha lógica, me he apoyado en el uso de numerosas variables de tipo booleano que indican las posibilidades que tiene cada objeto. Por ejemplo, en la clase *worm*, podemos encontrar variables como *isDead*, *isTurning*, *girado,* etc.

Por desgracia, esta manera de enfocar la programación de la me

Lo más sensato hubiese sido seguir un guión basado en *frames* o estados del objeto e incluir cada método en su respectivo lugar mediante sentencias *switch*. La razón de no haber procedido de tal manera es mi escasa experiencia en la programación y el hecho de que, a pesar de tener claras las funcionalidades que el personaje debía presentar, no llevé a cabo una buena esquematización inicial del código.

2.4 Interfaz de usuario

En este punto diseñé en *Adobe Illustrator* los iconos, tanto para los botones de navegación, como para los ítems que permiten al usuario controlar las acciones del gusano. Estos controles son los siguientes:



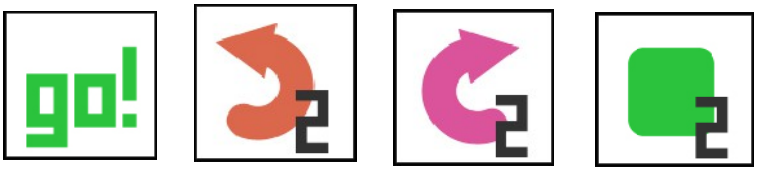
- Indicador de nivel: Este indicador no presenta ninguna interacción. Sencillamente informa al usuario de en que nivel se encuentra.

- Bloqueo de cámara: Permite al usuario elegir entre si la camára debe seguir al gusano o no hacerlo.

- Reiniciar nivel: Reinicia el nivel actual.

- Salir: Redirecciona al usuario a la pantalla inicial del juego.

- Música: Permite al usuario activar y desactivar la música.



- Iniciar movimiento: Este botón provoca que el gusano empiece a avanzar. A diferencia del resto de la interfaz no tiene una posición fija en la pantalla, sino que se sitúa sobre el gusano siempre que este permanezca inmóvil. El propio gusano es una extensión de este control, con lo que el usuario también puede iniciar el movimiento pulsando sobre él.

- Girar hacia la izquierda: Permite desplegar el ítem de giro hacia la izquierda en el recorrido. El número indica la cantidad de giros disponibles.

- Girar hacia la derecha: Exactamente igual que el giro hacia izquierda pero en sentido opuesto.

- Crear Puente: Permite al usuario desplegar el ítem de creación de puente. Debe colocarse en la casilla inmediatamente anterior al hueco que se desea cubrir. De nuevo, el número indica la cantidad de puentes disponibles.

2.5 Programación de la interactividad

Sin duda esta ha sido una de las tareas más complicadas de llevar cabo. Desde un principio tenía claro que la interactividad del usuario debía ser tanto táctil como a través del ratón y del teclado. De esta manera, he diseñado la clase *toucher*, la cual combina ambas opciones.

En Worm It! la interactividad no solamente permite pulsar los botones de la interfaz, sino también ofrece al usuario la posibilidad de mover el escenario o incluso de rotarlo.

Por un lado, Unity ya incorpora el objeto *Input*, dentro del cual se puede acceder al objeto *Touch*, el cual proporciona una gran variedad de métodos y propiedades para la gestión de los toques que el usuario realiza sobre la pantalla. Así pues mediante el método *Touches.count()* es posible conocer con cuantos dedos está tocando la pantalla el jugador. Mediante la propiedad *TouchesPhase,* sirve para saber el estado actual del toque. Es decir, si este acaba de suceder, si el dedo está quieto en la pantalla o si está siendo arrastrado. Y mediante *Touches[x].position*, es posible conocer la posición de un dedo en la pantalla.

Por otro lado, también dentro del objeto *Input*, podemos acceder a *Mouse, MouseDown, MouseUp, GetKey*, etc, De forma similar a *Touch*, estos métodos proporcionan toda la información necesaria para gestionar la interacción con el ratón y el teclado.

Como ya he mencionada, diseñar esta interactividad no ha sido tarea fácil. Finalmente he planteado una lógica un tanto enrevesada que impide a la interacción táctil interferir con el control clásico y viceversa.

```
public void Tocar ()
{
	if (Input.touchCount > 1)
		paraTorpes ();
	if (Input.touchCount == 0 && !worm.dead && !estadojuego.win)
		pcControles ();
	if (!worm.dead && !estadojuego.win) {
		moverEscenario ();
		girarEscenario ();
	}
	if (!Input.GetKey (KeyCode.RightControl) && !Input.GetKey (KeyCode.LeftControl))
		desactivarSuperNull ();

}
```

2.6 Música y efecto de sonido

El sonido y la música son elementos muy importantes en un videojuego. Mediante los efectos de sonido se pueden reforzar las acciones que realiza el personaje y la interacción del usuario. De esta manera me he propuesto encontrar los sonidos que mejor se adapten a la estética del juego.

Los principales efectos sonoros son los relacionados con el movimiento del gusano incluida su muerte o su llegada a la madriguera. También he incluido sonidos en la manipulación de los ítems y en la pantalla principal.

Por otro lado, también tenia claro que la música debía ser alegre y ligera. Con lo que me he propuesto componer un tema en esa línea. Para ello, he usado el *software Fruity Loops* que incorpora una extensa variedad de ritmos, sintetizadores, efectos, etc.

Uno de los problema con los que me he encontrado a la hora de implementar la música es que, por defecto, *Unity* resetea todos los objetos del juego al cambiar de escena. Esto provoca que cuando el usuario avanza de nivel la música se reinicie.

Para solucionar este problema, he tenido que explorar varias opciones. Entre ellas, utilizar las preferencias de usuario para almacenar el tiempo de reproducción de la canción. Pero de esta manera no conseguía corregir una pequeña pausa que se generaba en cada salto de escena.

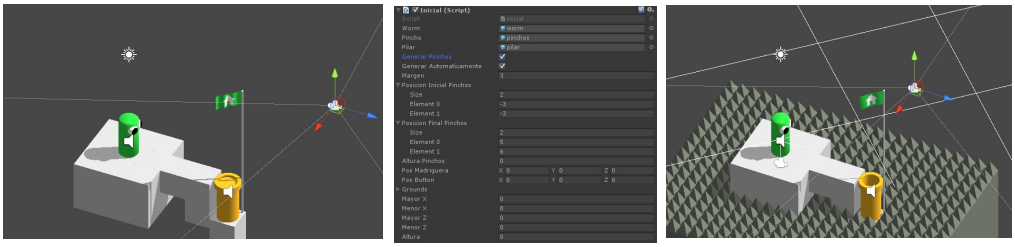
Finalmente opte por un remedio mucho más profesional. Por suerte, *Unity* permite declarar objetos estáticos que sobreviven a los cambios de escena. Así pues, incluí un objeto *AudioSource* en una clase estática, la cual instancie en el primer nivel.

```
public class musicPlayer : MonoBehaviour {
	public AudioSource songA;
	public AudioSource songB;
	public bool music;
	public int song;

	private static musicPlayer instance = null;
	public static musicPlayer Instance
	{
		get{
			if (instance == null) {
				instance = (musicPlayer)FindObjectOfType (typeof(musicPlayer));
			}
			return instance;
		}
	}
	void Awake()
	{
		if (Instance != this) {
			Destroy (gameObject);
		} else {
			DontDestroyOnLoad (gameObject);
		}
		iniciar ();
	}
```

2.7 Diseño de los niveles

Para esta tarea he diseñado un *script* que, una vez dispuesto el recorrido de cada nivel, genera de manera automática el resto de elementos necesarios para su correcto funcionamiento. Estos son por un lado, los pinchos que circundan el recorrido y sobre los cuales no puede caminar el gusano y, por otro lado los márgenes que delimitan el movimiento de la cámara al ser arrastrada por el usuario.



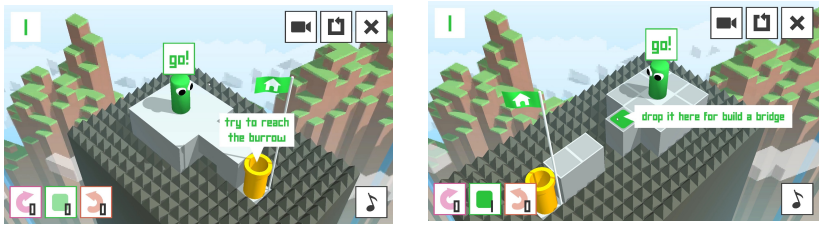
Por otro lado, he diseñado los 14 primero niveles del juego, siguiendo una curva de aprendizaje para los 6 primeros y una curva de dificultad para los restantes, tal y como expongo en el siguiente punto.

2.8 Curvas de aprendizaje y dificultad

Los 6 primeros niveles son muy básicos. En cada uno de ellos el usuario debe descubrir una por una las funcionalidades que le permitirán afrontar los niveles de mayor dificultad.

Por ejemplo, en el primer nivel no hay ningún obstáculo y lo único que el usuario debe hacer es pulsar sobre el botón *Go!* Para lograr que el gusano alcance la madriguera.

En el segundo y tercer nivel, el usuario debe comprender la mecánica necesaria para colocar los ítems sobre el recorrido.



Y en el cuarto, quinto y sexto, se enseña al usuario que el gusano puede morir si cae de cierta altura, que es posible empujar cajas móviles y que el escenario puede ser rotado con la finalidad de acceder a zonas ocultas del recorrido.

Respecto a la curva de dificultad, he intentado alternar niveles complicados con otros de más fácil resolución para dar la sensación de que tras un gran esfuerzo se obtiene una recompensa.

2.9 Primera versión para *smartphone*

Finalmente, he exportado mediante la opción Build Settings de Unity la primera versión jugable para teléfonos Android. Esta tarea requiere instalado la SDK de Android así como Java en el sistema.

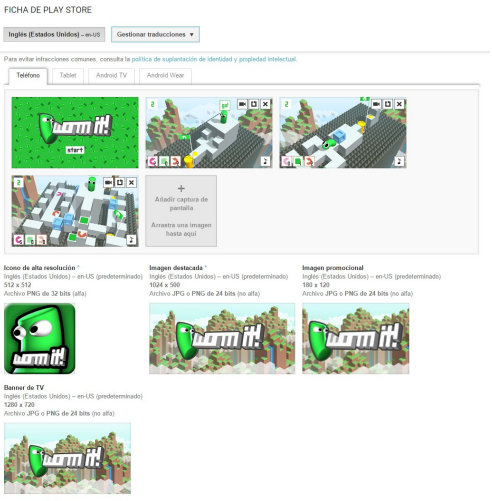
Además, hay que tener muy en cuenta ciertos aspectos relativos a su publicación, como pueden ser la compañía y el título de la aplicación y la versión de lanzamiento, ya que no es posible actualizar una aplicación en *Google Play* si la versión es idéntica o anterior.

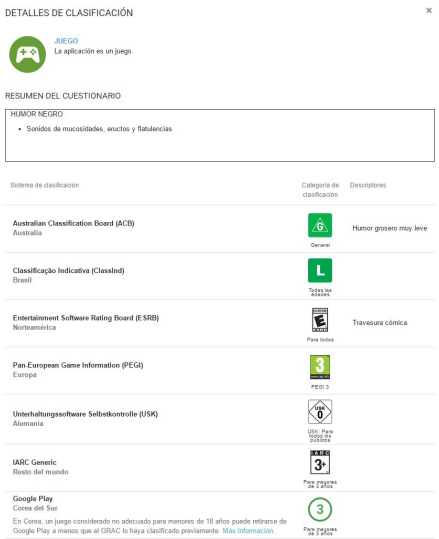
Por último, para que Unity exporte correctamente el archivo *APK* es imprescindible generar un *keystore* con los datos del desarrollador y una contraseña que será la firma del producto y, sin la cual, Google Play no permitirá realizar actualizaciones.

2.10 Publicación del producto en *Google Play*

Para poder publicar aplicaciones para Android en la tienda de Google, es necesario estar registrado como desarrollador. Para ello, basta con crear una cuenta de correo Gmail y pagar una tasa única de 12 euros. Comparado con Apple, los trámites y requisitos necesarios para desarrollar y publicar para Android son muy pocos.

Una vez se ha obtenido la cuenta de desarrollador el siguiente paso es subir la aplicación a la tienda junto a una descripción del producto y varias capturas del juego. Además hay que completar un pequeño formulario para la obtención del certificado IARC, según el cual se valora si el contenido de la aplicación es apto para todas las edades.

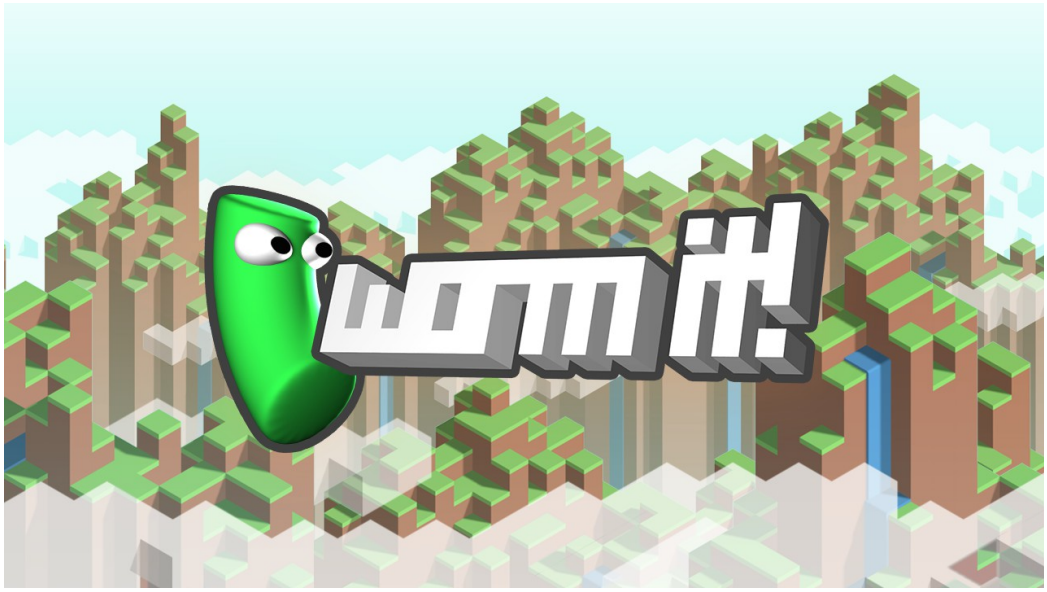




2.11 Promocionar el producto

Con la finalidad de dar a conocer el juego de una manera sencilla y directa he editado una pequeña pieza de vídeo de apenas 20 segundos de duración. Este spot muestra imágenes del juego combinadas con mensajes publicitarios.

El objetivo de este vídeo no es solo que los usuarios que lleguen a la página de la aplicación puedan ver rápidamente la dinámica del juego, sino también tener la posibilidad de mostrarlo y difundirlo en las redes sociales



2.12 Telemetría

Finalmente y siguiendo las recomendaciones de los consultores he incluido una sencilla funcionalidad que me permite conocer, cuanto tiempo tarda cada usuario en completar un nivel.

Para ello he empleado la clase *WWW* de Unity, mediante la cual envío una petición PHP que registra en una base de datos la información deseada.

No obstante, solo he implementado dicha funcionalidad en la aplicación para PC, ya que en Android, cualquier intento de acceder a Internet, provoca que el usuario tenga que aceptar unos permisos especiales antes de instalar la aplicación en su teléfono, cosa que he preferido evitar.

3. Conclusiones

En todos los trabajos surgen problemas y situaciones inesperadas. En ocasiones, estos conflictos son beneficiosos para el desarrollo del proyecto, ya que gracias a ellos es posible descubrir mecánicas de trabajo que de otro modo pasarían inadvertidas. Otras veces, el obstáculo es tan grande que es necesario claudicar para no exceder el tiempo y los recursos disponibles. En este sentido, **Worm it!** No ha sido una excepción.

Sin ir más lejos, el hecho de que tuviese problemas para reproducir la música durante el cambio de nivel, me ha ayudado a descubrir la utilidad de las clases estáticas. Por desgracia, también me he visto obligado a descartar la implementación de la *SDK* de Facebook por falta de tiempo y experiencia.

Por otro lado, en los comentarios de los usuarios que han probado el juego y también por parte consultores, he recibido una fuente de ideas enorme. Por este motivo, considero que, una buena práctica de desarrollo, es mostrar el trabajo realizado lo antes posible, tan pronto se disponga de algo jugable.

No es bueno pensar que el hecho de enseñar un Trabajo en proceso puede repercutir negativamente en su publicación final. De hecho, opino que en la mayoría de casos sucede todo lo contrarío. En **Worm it!** podría enumerar un sinfín de aspectos que nunca hubiese mejorado sin el feedback de las personas que han ido probando el juego a medida que lo desarrollaba.

Por poner un ejemplo, en un principio no tenía pensado que lo ítems colocados sobre el escenario pudiesen moverse o quitarse de lugar. Esto implicaba que si el usuario erraba la posición de algún ítem, debía reiniciar el nivel y empezar desde cero. Por otro lado, tampoco me había planteado implementar la opción de rotar el escenario, la cual me ha permitido crear diferentes tipos de niveles.

En definitiva estoy satisfecho con el trabajo realizado. Considero que el producto final puede proporcionar varias horas de entretenimiento a quien se anime a probarlo.

4. Glosario

APK (Android Application Package): Formato con el que se guardan los archivos de instalación para las aplicaciones de Android.

AudioSource: Clase de Unity que permite reproducir audio en las escenas.

Bug: error en la aplicación.

IARC (International Age Rating Coalition): Estandar de clasificación de contenido según país.

Input: Clase de Unity que permite conocer que entradas está manipulando el usuario.

FBX (FilmBox): Formato propiedad de la compañía Autodesk para el intercambio de contenidos digitales.

Frame: Estado de la aplicación.

Framework: Conjunto de estándares que forman una infraestructura para el desarrollo de software.

Freeware: Softawre que el autor libera para su uso libre.

Gameplay: Acción de jugar a un videojuego.

GDD (Game Document Design): Documento que recoge todos los aspectos relevantes sobre el desarrollo de un videojuego.

Morph: Modificador de 3Ds MAX que permite almacenar diferente posiciones para los vertices de un solo modelo.

PHP (Pre Hypertext Procesor): Lenguaje de programación de uso liberado.

Phisics: Clase de Unity que simula comportamientos físicos de los objetos virtuales.

Point cache: Modificador de 3Ds MAX, similar al Morph pero sin interpolación.

Raycast(): Clase de Unity que permite castear el espacio de la escena por medio de rayos.

Script: Código escrito para su interpretación.

SDK (Software Development Kit): Conjunto de repositorios que contienen todo lo necesario para el desarrollo de software en una determinada plataforma

Smartphone: Teléfono inteligente.

Software: Programas y aplicaciones destinadas a ser usadas en un ordenador.

Switch: Estructura que permite declarar diferentes sentencias en función de un único valor.

Touch: Clase de Unity que ofrece toda la información sobre la interacción del usuario con la pantalla táctil.

Vista isométrica: Proyección del espacio según la cual los tres ejes ortogonales forman angulos de 1**20º.**

WWW: Clase de Unity que permite emular la navegación en la red.

5. Bibliografía

http://www.lavasurfer.com/kevin-ng/games.html Febrero 2016

http://outletsbcn.com/catalogo-de-productos/juguetes-y-juegos/muelle-m%C3%A1gico-de-colores-arco-iris-rainbow-detail Febrero 2016

https://docs.unity3d.com/ScriptReference/ Febrero – Junio 2016

https://es.wikipedia.org/wiki/Unity_(software) Mayo 2016.

https://es.wikipedia.org/wiki/Proyecci%C3%B3n_isom%C3%A9trica Junio 2016

https://en.wikipedia.org/wiki/FBX Junio 2016

https://en.wikipedia.org/wiki/International_Age_Rating_Coalition Junio 2016