Universidad Zaragoza

Proyecto Fin de Carrera

Maño Invaders: Desarrollo de un videojuego completo usando Game Maker

Autor

Edgar Murillo Castillo

Director

Eduardo Mena Nieto

Ingeniería en Informática

Escuela de Ingeniería y Arquitectura

2017

Resumen

Actualmente los videojuegos han ido ganando terreno a otras áreas de ocio existentes en nuestro día a día. Los videojuegos han alcanzado cotas en las que no es extraño observar día tras día videojuegos que superan a grandes producciones cinematográficas, tanto en presupuesto como en recaudación. Gran parte de este éxito ha sido posible debido a la gran evolución que han sufrido los mismos a lo largo de los años pero sobre todo, gracias a la reciente accesibilidad a los mismos por parte de cualquier tipo de público mediante la irrupción de las llamadas *Stores* (repositorios *online* de software). Gracias a las *Stores*, los desarrolladores, que pueden ir desde un gran estudio de desarrollo hasta un único desarrollador libre, han obtenido un rápido y fácil acceso a una distribución directa a cliente final, pudiendo estos desarrollar un videojuego y que en cuestión de minutos este se encuentre accesible a millones de usuarios.

En este proyecto se va a implementar un videojuego de temática aragonesa inspirado en el cartel de la primera edición del evento RetroMañía, celebrada en Zaragoza en 2007. Dicho cartel estaba basado a su vez en el videojuego "Space Invaders" (Taito, 1978); videojuego que representa fielmente el enfoque clásico de los primeros *Shoot'Em Up* o más comúnmente conocidos como "mata-marcianos". El desarrollo del mismo será realizado mediante una plataforma de desarrollo, mundialmente conocida y usada por desarrolladores de videojuegos, denominada *GameMaker*.

El objetivo principal es implementar en su totalidad un videojuego, pasando por todas las fases de desarrollo que tiene un videojuego hoy en día, con el objetivo de que sirva tanto de estudio de todo el proceso, como de guía. Así pues, servirá de ejemplo a todos aquellos desarrolladores que quieran iniciarse tanto en la creación como adaptación de videojuegos antiguos y que dispongan o no, de nociones de programación, y cuyo objetivo sea hacer llegar su videojuego, en diversas plataformas de distribución y juego, a la mayor cantidad de usuarios posible.

Agradecimientos

A mi director, Eduardo Mena, por sus continuas aportaciones, ayudando siempre a avanzar y a mejorar, y en especial por todo lo logrado con RetroMañía 2016.
A las asociaciones "RetroAcción" y "ARPA" por toda la ayuda prestada durante la celebración de RetroMañía 2016, por dar la idea e instalar una recreativa con Maño Invaders, así como por la organización de un torneo en torno al juego como parte de sus actividades.
A los grupos "Los Berzas", "Los Gandules" y a la "Ronda de Boltaña" por permitir utilizar sus canciones en el videojuego.
A Mónica, por estar siempre animándome y ser siempre la primera en probar todas las versiones.
A las decenas de participantes que participaron en el torneo de Maño Invaders, tanto en la versión de PC, como Android y recreativa.
A todos los amigos que desinteresadamente fueron probando el videojuego en sus fases previas.
Al equipo de *GameMaker*, que gracias a sus versiones gratuitas ha sido posible llevar a cabo el proyecto.
Y por último, a toda mi familia por apoyarme en todo momento.

ÍNDICE DE CONTENIDO

Lista de Figuras

Capítulo 1

Introducción

Un videojuego es una aplicación interactiva orientada al entretenimiento que, a través de ciertos mandos o controles, permite simular experiencias en la pantalla de un televisor, una computadora u otro dispositivo electrónico.

Los videojuegos se diferencian de otras formas de entretenimiento, como por ejemplo, las películas, en que deben ser interactivos; en otras palabras, los usuarios deben involucrarse activamente con el contenido.

En la actualidad, un gran número de videojuegos son realizados utilizando plataformas o entornos de desarrollo. Estos entornos son aplicaciones informáticas pensadas para poder crear videojuegos, liberando a los desarrolladores de la ardua tarea de programarlos desde cero. Son por ello muy utilizados tanto para prototipado, por no expertos en programación, así como para implementar videojuegos tan complejos que sería imposible realizarlos desde cero en un tiempo razonable. Este tipo de software genera el código necesario para ejecutar un videojuego a partir de la definición de los elementos y la lógica del mismo a través de menús; aunque siempre se requiere de la programación de ciertos *scripts* para reflejar el comportamiento no trivial de algún objeto en el videojuego.

1.1. Motivación

Los entornos de desarrollo facilitan en gran medida la labor al desarrollador de videojuegos. Aun con todo, el proceso desde cero de crear un videojuego y conseguir que este llegue finalmente al público general, puede resultar tedioso si no se conocen bien todas las fases por las que todo videojuego debe pasar, así como las herramientas adecuadas a utilizar con objeto de implementarlo.

En Internet hoy en día es posible encontrar pequeños tutoriales que nos indican cómo realizar cierta tarea en algún entorno de desarrollo concreto, o cómo implementar juegos pequeños, pero la mayoría de ellos no nos detallan las fases por las que va

pasando hoy en día la realización de un videojuego: desde su concepción hasta su llegada al público final.

Teniendo en cuenta lo anterior y sumado a la celebración de la X edición del evento RetroMañía en 2016, se decide desarrollar un videojuego inspirado en el cartel de la primera edición de dicho evento (celebrado en Zaragoza en el año 1997, (ver figura 1.1) con objeto de que este a su vez tenga la suficiente envergadura como para pasar por todas las fases de desarrollo que un videojuego debería tener y que sirva de guía y ejemplo para futuros desarrolladores que busquen hacer llegar sus videojuegos al mayor número de usuarios y plataformas posibles.

Así pues, el videojuego a desarrollar toma como referencia el mencionado cartel de RetroMañía, que estaba basado a su vez en el videojuego "Space Invaders" (1978), pero dándole un aspecto visual (no desprovisto de humor) "muy aragonés", por lo que el nombre del videojuego queda definido como "Maño Invaders".

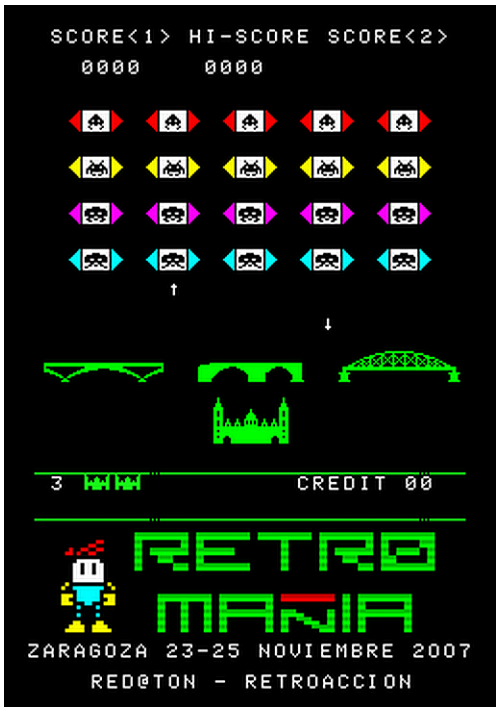

Figura 1.1: Cartel de la primera edición de RetroMañía (2007)

1.2. Objetivos

El objetivo es desarrollar el videojuego Maño Invaders, según estas fases:

– Diseñar los aspectos que caracterizarán al videojuego, desde los gráficos, efectos de sonido, música y menús, hasta el comportamiento de los distintos elementos del juego, diseño de niveles, curva de dificultad, y ajuste de su jugabilidad.

– Implementar el videojuego Maño Invaders utilizando el entorno *GameMaker*[1], programando los *scripts* necesarios para replicar comportamientos del juego no definibles a través de dicho entorno.
– Realizar distintas pruebas del videojuego realizado (por parte de distintas personas) y hacer las adaptaciones necesarias para su funcionamiento y despliegue tanto en equipos *Windows* como en dispositivos móviles *Android* (a partir de la misma definición en *GameMaker*).
– Crear un sistema de puntuaciones *online* y multiplataforma para que los distintos jugadores compitan entre sí por la puntuación global más alta.
– Hacer público el videojuego coincidiendo con la celebración del X aniversario de RetroMañía.

[1] http://www.yoyogames.com/gamemaker

Capítulo 2

Tecnología utilizada

Para la creación de cualquier videojuego deberemos utilizar tecnologías específicas según el cometido que queramos realizar. En el caso de Maño Invaders tenemos la tecnología destinada puramente a la programación del videojuego; la tecnología que estará relacionada con cómo persisten o se almacenan los datos del videojuego; la que se utilizará para toda la parte web del videojuego; así como la destinada al control de versiones del videojuego y de gestión de tareas.

2.1. Entornos de desarrollo de Videojuegos

Los entornos de desarrollo de videojuegos son los encargados de automatizar la gran mayoría de tareas de bajo nivel y comunes a muchos tipos de videojuego, aliviando al diseñador justo de aquellas tareas de programación más repetitivas y no propias de su videojuego. Dentro de los entornos de desarrollo, podemos tener algunos más indicados para juegos 3D y otros más propicios para el desarrollo de juegos 2D. Algunos de los entornos más usados en la actualidad son *GameMaker*, *Construct*[1], *Unity*[2], *Stencyl*[3], *GameSalad*[4]...

Toda la programación de Maño Invaders se realiza utilizando el entorno de desarrollo específico de videojuegos *GameMaker*, creado por el profesor Mark Overmars en el lenguaje de programación *Delphi*[5]. El programa es gratuito, aunque existe una versión comercial ampliada con características adicionales. Actualmente se encuentra en su versión "studio 2".

GameMaker se caracteriza por posibilitar dos vías para la creación de videojuegos, pudiéndose compenetrar una junto a la otra. Por un lado, una vía menos orientada a

[1]https://www.scirra.com/
[2]https://unity3d.com/es
[3]http://www.stencyl.com/
[4]http://gamesalad.com/
[5]https://www.embarcadero.com/products/delphi

programadores, que consiste en una forma de programación visual, en la que mediante el uso del ratón añadimos órdenes a los objetos y por otro lado, tenemos la vía de añadir código (introducido por nosotros) a los objetos, que podríamos considerar "la vía del programador". Esta última es sobre todo la vía más interesante ya que nos abre las puertas del lenguaje *GameMaker Language* (de ahora en adelante "GML") y todos los comandos del mismo, facilitando la tarea a aquellos que tengan nociones de programación.

2.2. Persistencia de Datos

Con objeto del almacenamiento de datos que pueda necesitar el videojuego se delega en *GameMaker* para gestionar mediante ficheros encriptados esta información. Por otro lado, toda aquella información susceptible de ser almacenada *online*, se almacena en una base de datos *MySQL*[6] alojada en un servidor externo.

2.3. Web

Para la gestión web, se utilizan las siguientes tecnologías:

- *JAVA*[7]: es el lenguaje de programación utilizado en el servidor.
- *JSF*[8] apoyado en *Primefaces*[9]: con objeto de hacer uso de las notaciones y componentes que nos ofrecen para implementar la vista del servidor.
- Una *API REST*[10] utilizada para la transferencia de datos entre el juego y el servidor.
- *Spring MVC*[11] utilizado como capa de interconexión.
- Una capa de *Hibernate*[12] para realizar las peticiones a base de datos.
- *Eclipse*[13]: es el entorno de desarrollo utilizado para programar.
- *Wildfly*[14]: es el servidor en el que corre la aplicación web.

[6]https://www.mysql.com/
[7]https://www.java.com/es/
[8]http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[9]http://www.primefaces.org/
[10]https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional
[11]https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[12]http://hibernate.org/
[13]https://eclipse.org/
[14]http://wildfly.org/

2.4. Control de versiones

Con objeto de ser capaz de volver a cualquier punto de la aplicación anterior y conocer en todo momento qué se ha ido haciendo se utiliza la herramienta *GIT*[15], que permite llevar un control de versiones, pudiendo tener distintas ramas para un proyecto, pudiéndolas mezclar a elección y trabajar en incidencias detectadas al mismo tiempo sin que una interfiera con la otra.

Como herramienta auxiliar de *GIT* se ha utilizado *SourceTree*[16], que nos ofrece una interfaz gráfica desde la que podemos observar los últimos cambios que se han subido a cada rama de nuestro proyecto, subir cambios, actualizar ramas, etc.

Por último, con objeto de facilitarnos la labor de visualizar todos los cambios que ha ido sufriendo un fichero concreto en *GIT*, se ha utilizado *TortoiseGIT*[17], el cual así mismo puede integrarse en el explorador de *Windows*.

2.5. Gestión de tareas

Con objeto de llevar un control de todas las tareas a realizar, así como de los errores detectados y que estos nunca caigan en el olvido, se utiliza la herramienta *Wrike*[18], que pese a no ser tan potente como herramientas conocidas como *JIRA*[19] o *Redmine*[20], nos permite:

- Crear tareas de manera simple.
- Añadir descripciones a las mismas.
- Categorizarlas.
- Establecerles prioridades.
- Programarlas estableciendo fechas de inicio y fin.
- Modificar los estados de las mismas: completada, pendiente, en pausa, anulada...
- Ser asignadas a recursos: en casos de equipos de más de una persona, se pueden distribuir.

[15] https://about.gitlab.com/
[16] https://www.sourcetreeapp.com/
[17] https://tortoisegit.org/
[18] https://www.wrike.com/
[19] https://es.atlassian.com/software/jira
[20] http://www.redmine.org/

Capítulo 3

Diseño de Maño Invaders

En esta fase de diseño se pensarán y plasmarán aquellos comportamientos del videojuego que serían comunes independientemente de las plataformas con las que lo implementemos o del aspecto gráfico del mismo.

En el caso de Maño Invaders aquellos puntos de diseño comunes se verán a continuación:

3.1. Pantallas del Juego

El juego estará formado por distintas pantallas (ver diagrama en figura 3.1) entre las cuales encontraremos:

- Menú principal: será la primera pantalla en aparecer y desde la misma accederemos al resto.
- Fases del juego: estarán formadas por cada una de las fases por las que iremos avanzando en el transcurso del juego.
- Pantalla de puntuaciones: pantalla que nos mostrará un histórico de las diez mejores puntuaciones obtenidas.
- Pantalla de instrucciones: es una pantalla que nos indicará los controles y la temática del juego.
- Pantalla de créditos: mostrará los créditos y agradecimientos del juego.

3.1.1. Menú principal

Será el primer menú al que llegaremos al iniciar el juego, y nos servirá de punto de partida para acceder al resto de menús. Nos mostrará una serie de botones para facilitar el acceso al resto de menús, así como el titulo del juego, y una imagen de fondo en movimiento. Los botones del menú cambiaran de imagen en función de si nos encontramos sobre ellos o no, o bien si los hemos seleccionado. También aparecerán

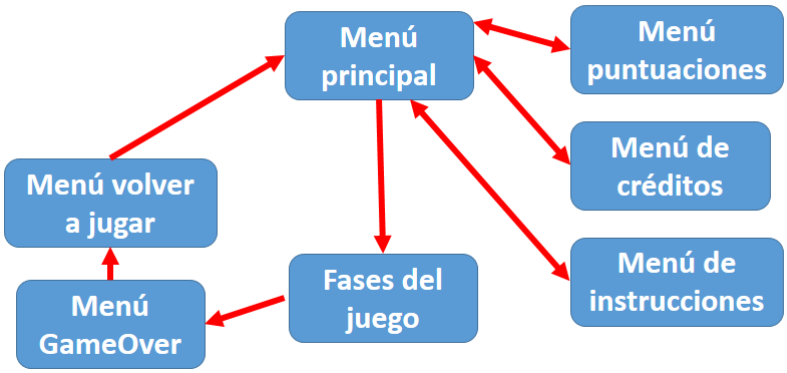

Figura 3.1: Diagrama de pantallas e interacción entre las mismas

aleatoriamente invasores que se irán moviendo por la pantalla. Por último, en su parte superior derecha aparecerá un botón representativo de un altavoz, que servirá para activar o desactivar el sonido en el juego.

3.1.2. Menús auxiliares

- Menú de puntuaciones: En este menú se mostrará el título del mismo en la parte superior, así como una tabla con las diez máximas puntuaciones locales conseguidas en el dispositivo. Estas se mostrarán ordenadas de mejor a peor, indicando la posición, el nombre de usuario y los puntos logrados. Por último, debajo de la tabla aparecerá un botón que nos permitirá volver al menú principal.
- Menú de créditos: En este menú aparecerá el título del mismo, así como una imagen central que irá cambiando para mostrar todos los diferentes créditos del juego. Las transiciones entre imágenes se llevarán a cabo produciendo un efecto de desvanecimiento de las mismas y posterior aparición progresiva de las nuevas. Por último, también contendrá un botón "volver" para regresar al menú principal.
- Menú de instrucciones: En este menú aparecerá el título del mismo, así como una imagen central que irá cambiando para mostrar las diferentes indicaciones del juego, explicándonos el objetivo del jugador y los controles. Las transiciones entre imágenes funcionarán del mismo modo que en los créditos. Así mismo, contendrá un botón "volver" para regresar al menú principal.

3.1.3. Fases de Maño Invaders

El juego se estructurará en 12 fases: nueve intermedias y tres de jefes finales. Siempre habrá tres fases intermedias que precederán a un jefe final. En caso de que el jugador supere las 12 fases, estas se repetirán incrementando su dificultad, hasta alcanzar un

máximo de la misma en la que cesará en su aumento ya que de ser esta incrementada, sería imposible superar la fase. La temática de las fases será la siguiente:

- Fases 1-3 Zaragoza: tendrán una temática basada en torno a Zaragoza, donde aparecerán puentes, monumentos y autobuses de la misma.
- Jefe final Zaragoza: el antiguo alcalde de la ciudad será el protagonista de la fase
- Fases 1-3 Expo: girarán en torno a la Exposición celebrada en Zaragoza en 2008 apareciendo puentes, pabellones y teleféricos de la misma.
- Jefe final Expo: el jefe representativo de la Expo será la que fuera su mascota, de nombre Fluvi.
- Fases 1-3 Aragón: fases de temática aragonesa, dónde aparecerán alimentos de la región, montañas y castillos.
- Jefe final Aragón: el jefe final de las fases de Aragón será "SuperMaño", un "héroe" de cómic que aparece de vez en cuando en la prensa de la comunidad.

Como detalle de la estructura y orden de las fases se ilustra la figura 3.2.

Figura 3.2: Estructura de fases

3.2. Nave del jugador

La nave del jugador deberá ser una representación de la basílica del Pilar con un aspecto similar al que se podía visualizar en el cartel de RetroMañía 2007. Dicha nave se moverá siempre horizontalmente y nunca en vertical. Como criterio adicional, se opta que como máximo la nave pueda exceder de los limites laterales de la pantalla hasta que solo sea visible la mitad de esta. La velocidad de la nave deberá ser definible y adecuarse correctamente a la jugabilidad, de tal modo que no se mueva ni demasiado rápido ni demasiado lento.

3.3. Invasores

Las naves invasoras tendrán como objetivo destruir con sus disparos la nave del jugador o bien, avanzar tanto, que invadan al mismo. Los invasores, estarán vinculados a las fases, de tal modo que tendremos:

- Fases intermedias de Zaragoza: invasores "tranvías".

– Fases intermedias de la Expo: invasores “adoquines de Zaragoza”
– Fases intermedias de Aragón: invasores “frutas de Aragón”

Los invasores estarán inspirados en los del cartel de RetroMañía, especialmente los adoquines, que serán los más similares a los que aparecían en dicho cartel. Todos ellos, sean de las fases que sean, heredarán los colores originales: rojo, amarillo, rosa y azul. Estos invasores así mismo, se utilizarán en los menús del juego a modo presencial, para darle un toque más personalizado al juego.

3.4. Movimiento de los invasores

Se establece que los invasores avanzarán siempre horizontalmente y en orden, comenzando primero la fila inferior, y continuando de una en una el resto de las filas. Así pues, primero se moverá una fila y se detendrá; avanzará la siguiente quedándose también en espera; y así las sucesivas (ver figura 3.3).

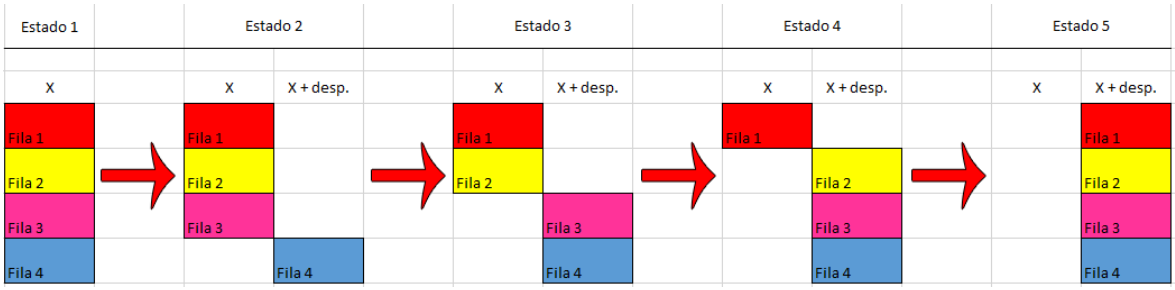

Figura 3.3: Ejemplo de desplazamiento horizontal de los invasores

Definiendo el tiempo de movimiento de cualquier fila de invasores como t_0. El tiempo t que tardará en volver a moverse será:

$$t = t_0 + t_0 + t_0 + t_0$$

Es decir, la suma de lo que tarda ella misma en moverse y lo que tardarán el resto de las otras tres filas.

El movimiento de los invasores se caracterizará por:

– Comenzar a una velocidad constante. Siendo esta velocidad regulada por el tiempo de movimiento acontecido entre una fila y la siguiente.
– Aumentar la velocidad inicial en cada fase que superemos hasta llegar a un máximo.
– Aumentar la velocidad cada vez que destruyamos a un enemigo.
– El desplazamiento en vertical ocurrirá siempre que cualquiera de las cuatro filas llegue a uno de los dos límites de la pantalla.

– El descenso en vertical, al contrario que el movimiento horizontal, será siempre simultaneo en todas las filas.
– Conforme avancemos de fase, los invasores comenzarán más cerca del jugador.
– En caso de que los invasores se encuentren en un punto muy cercano al jugador e intenten descender, estos no descenderán ya que el jugador se considerará invadido y perderá la partida.

3.5. Barreras defensivas

Las barreras defensivas de Maño Invaders podrán ser de dos tipos: horizontales y verticales. Independientemente del tipo, se comportarán de tal manera que al recibir un disparo en una parte concreta de la misma, tanto por parte del jugador como por parte de un invasor, perderá dicha parte dejando un hueco en la misma. Así pues, servirán de defensa para el jugador de cara a protegerse de disparos enemigos, y así mismo en algún momento al jugador le será posible abrirse hueco en las mismas.

Se define que las barreras en cada fase estarán ambientadas dependiendo de la temática de la misma. Por lo tanto, las barreras de cada fase serán las siguientes:

– Fase 1 Zaragoza: Puente de Santiago, Puente de Piedra y Puente de Hierro.
– Fase 2 Zaragoza: Puerta del Carmen, Palacio de la Aljafería y Murallas Romanas.
– Fase 3 Zaragoza: autobuses urbanos de Zaragoza.
– Fase 1 Expo: Pasarela Puente, Pabellón Puente y Puente del Tercer Milenio.
– Fase 2 Expo: Pabellón de Aragón, Torre del Agua y Pabellón de España.
– Fase 3 Expo: teleféricos Expo.
– Fase 1 Aragón: Jamón de Teruel, Vino de Cariñena y Trenza de Almudévar.
– Fase 2 Aragón: Castillo de Loarre, Castillo de Sádaba y Castillo de Alcañiz.
– Fase 3 Aragón: Sierra de Guara, Pico Aneto y El Moncayo.

3.6. Comportamiento de los disparos

Común a los disparos tanto de los invasores como del jugador será que estos se moverán siempre en vertical y nunca en horizontal, con una velocidad siempre continua.

– Disparos del jugador: en la parte del jugador se establece que para que el mismo pueda volver a disparar se tienen que cumplir dos condiciones: o bien el último disparo ha alcanzado un objetivo (enemigo o defensa) o bien ha cruzado la mitad de la pantalla. De este modo se evita que el jugador lance “ráfagas infinitas” o que tarde demasiado en poder volver a disparar.

– Disparos de los invasores: en la parte de los invasores, solo podrán disparar aquellos que no tengan a otro invasor de una fila inferior "delante". Es decir, poniendo un ejemplo, si el 5º invasor de la fila más cercana al jugador todavía no ha sido eliminado, el 5º invasor de cualquiera del resto de filas no podrá disparar; por el contrario, si este ha sido ya eliminado, el de la fila justo anterior podrá disparar. Se establece también que no todos los invasores tengan la misma probabilidad de disparo, si no que esta se base en unos "pesos" de tal modo que aquellos invasores que se sitúen horizontalmente más próximos al jugador tengan mayor probabilidad de poder disparar.

3.7. Potenciadores

Los potenciadores o también denominados *boosters*, son mejoras que podemos obtener en el transcurso de la partida y que nos facilitan avanzar en la misma, ayudándonos a superar las fases.

En Maño Invaders se establecen dos tipos de *boosters* que aparecerán siempre por un tiempo limitado. En las fases normales aparecerán en la parte superior de la pantalla y en las fases de jefes finales aparecerán entre el jugador y el jefe final, y por un tiempo más limitado que en las normales.

Los *boosters* estarán representados por dos banderas distintas:

– Bandera de Zaragoza: al ser alcanzada nos otorgará invencibilidad. De tal modo que cualquier disparo que se estrelle contra el jugador no provocará la perdida de ninguna vida ni la explosión de la nave.
– Bandera de Aragón: al ser alcanzada, nos otorgará el "multi-disparo". Cada vez que disparemos, dispararemos tres disparos simultáneos: un disparo que será el común y saldrá desde el centro de la nave, y dos que saldrán en cada una de las torres del Pilar, que actuarán como si de cañones se tratasen.

Los *boosters* tendrán un efecto limitado y no durarán siempre, por lo tanto con objeto de indicarnos que se encuentra en pantalla, que ha sido obtenido o bien que su efecto está a punto de desaparecer producirá distintos sonidos cuando se den dichas situaciones.

Cuando obtengamos un *booster*, el color de la nave del Pilar pasará a contener los colores de la bandera en cuestión, para que el jugador sea consciente de que ha obtenido el *booster*, volviendo la nave al color natural tras perder el efecto del mismo.

Por último, si se diera el caso de que en el transcurso del efecto de un *booster*, este no hubiera cesado todavía y obtuviéramos uno nuevo, este nuevo *booster* sustituiría

al anterior; y si obtuviéramos el mismo *booster* de nuevo, el contador de duración del efecto del mismo se reiniciaría.

3.8. Jefes finales

Se establece que cada 3 fases intermedias nos enfrentaremos a un jefe final que cerrará el ciclo/temática de dichas fases.

El movimiento general de los jefes será común y consistirá en lo siguiente:

- Se desplazarán siempre de lado a lado de la pantalla.
- Nunca descenderán verticalmente hacía el jugador, manteniéndose entre los dos cierta distancia; con lo cual, no hay posibilidad de colisión con el mismo.
- El desplazamiento horizontal será siempre en dirección hacía el jugador, de tal modo que parecerá que el jefe le este siguiendo por la pantalla.
- Al mismo tiempo que el jefe se mueve por la pantalla, este lanzará sus armas hacia el punto donde se encuentre el jugador (las armas se desplazarán tanto en horizontal como en vertical).

Todos los jefes finales partirán con una vida inicial, pero esta variará según el jefe final. Dicha vida, estará representada mediante un rectángulo que inicialmente aparecerá rellenado por dentro y conforme el jefe pierda vida, se irá vaciando hasta llegar a cero, momento en el cual el jefe final explotará y la fase será superada.

Otro aspecto común será que todos ellos estarán representados por ocho imágenes de los mismos en función del momento. Estos momentos serán los siguientes:

- Movimiento del jefe hacia la izquierda.
- Jefe alcanzado por disparo mientras se movía hacia la izquierda
- Movimiento del jefe hacia la derecha.
- Jefe alcanzado por disparo mientras se movía hacia la derecha.
- Jefe parado.
- Jefe alcanzado por disparo mientras estaba parado.
- Jefe lanzando ráfaga de disparos.
- Jefe alcanzado por disparo durante su ráfaga de disparos.

Por otro lado, cada uno de los jefes tendrá distintas ráfagas de ataque contra el jugador. Estas ráfagas interrumpirán su comportamiento de movimiento habitual, con el objetivo de atacar al jugador y que a este le sea más difícil protegerse.

Por último, se establece que cada uno de los jefes aporte una puntuación distinta al jugador cada vez que sea alcanzado por un disparo.

3.8.1. Jefe final fases Zaragoza: Alcalde

El primero de los jefes finales al que nos enfrentaremos será el "Alcalde". Su diseño estará inspirado en el "Mayor Quimby" (personaje y alcalde en la serie "Los Simpsons") del que heredará su cuerpo, pero se le añadirá la banda de alcalde de Zaragoza, y la cabeza del antiguo alcalde para darle un toque más cercano a la ciudad, no desprovisto de humor.

El comportamiento del "Alcalde" estará caracterizado por:

- Tornar su imagen a un tono rojizo y vibrar durante un breve periodo de tiempo tras ser alcanzado por un disparo del jugador.
- Lanzar dos tipos de armas para "impartir justicia": mazos y constituciones.

Por otro lado, como comentábamos antes, todos los jefes finales cada cierto tiempo tendrán al menos una ráfaga de disparos. En el caso del "Alcalde" consistirá en:

- Primero de todo, el "Alcalde" cesará su movimiento horizontal.
- Producirá una risa malévola.
- Comenzará a dar seis saltos verticales.
- Al comienzo de cada uno de los saltos verticales disparará cuatro disparos simultáneos, similares a los de los invasores, pero dirigiéndose cada uno de ellos hacia un ángulo distinto.
- El ángulo de los disparos se irá ampliando, para alcanzar con los disparos mayor área y obligar al jugador a moverse.
- Al terminar los seis saltos, volverá a su comportamiento habitual.

3.8.2. Jefe final fases Expo: Fluvi

"Fluvi" será el segundo de los jefes finales a los que nos enfrentaremos. Este heredará su diseño de la mascota oficial de la "Expo Zaragoza 2008", pero recibiendo ciertas modificaciones con el objetivo de parecer más peligroso.

Se diferenciará del "Alcalde" en:

- Sus imágenes representativas.
- Los sonidos que emitirá a lo largo de la fase.
- Dar pequeños saltos al moverse por la pantalla .
- Tener como armas: una gota de agua y un amigo de "Fluvi" (llamado "Po") perteneciente a unos cómics lanzados durante la "Expo Zaragoza 2008".
- Añadirá un paso previo a la ráfaga final del "Alcalde".

Su ráfaga final comenzará con un desplazamiento contrario al lugar en el que se encuentra el jugador, lanzando numerosas armas a su paso. Este desplazamiento será rápido y dando pequeños saltos a su vez. Tras llegar al límite de la pantalla, cesará su movimiento y comenzará con una ráfaga de disparo idéntica a la del "Alcalde", al mismo tiempo que ríe de forma malévola. Tras la ráfaga, retomará su comportamiento normal.

3.8.3. Jefe final fases Aragón: SuperMaño

"SuperMaño" será el último de los jefes finales y estará inspirado en el personaje de mismo nombre, que aparece con cierta frecuencia en cómics de la prensa aragonesa, partiendo sus diseños del mismo.

Los rasgos que lo caracterizarán y diferenciarán del resto serán:

- Sus imágenes.
- Sus armas: un garrote al más puro estilo "As de bastos" de la baraja española, y una piedra.
- Extender la ráfaga de movimiento lateral de "Fluvi" y modificar la de salto.

La ráfaga final del mismo comenzará al igual que la de Fluvi, moviéndose hacia el lado contrario de la pantalla, pero a diferencia de este, luego volverá también hacia el lado inicial, deteniéndose en el límite de la pantalla de ese lado. Tras lo anterior, conseguiremos que el jugador lo tenga más complicado para esquivar sus disparos. Al terminar dicha ráfaga se detendrá y comenzará con la habitual ráfaga de saltos, con la diferencia de que lo que lanzará en esta serán sus armas aleatoriamente, sumado a que cuando llegue al sexto salto, este no cesará y seguirá saltando cinco veces más, pero esta vez decrementando el ángulo en el que lanza sus armas hasta finalmente detenerse. Con estas dos modificaciones conseguiremos que sea el jefe final más complicado del juego.

3.9. Puntuaciones locales

El jugador irá acumulando puntuación fase tras fase hasta perder la partida y caer eliminado. En dicho momento podrá almacenar la puntuación establecida. Esta puntuación se almacenará en el dispositivo del jugador y si llegara a ser la más alta se mostraría en las sucesivas partidas en la parte superior de la pantalla a modo "puntuación a batir".

Dentro de las puntuaciones se establece que cada enemigo, según su tipo, otorgue una puntuación distinta.

3.10. Música del juego

El juego contendrá un *tracklist* (lista de canciones) determinado que incluirá siempre a artistas y grupos aragoneses, el cual, se irá reproduciendo durante los menús, así como durante las fases del juego.

En un inicio se plantean las siguientes canciones:

- Pantalla inicial: Jesús Gracia - "Virgen del Pilar que tienes la jota".
- Fases intermedias Zaragoza: Héroes del Silencio - "Entre dos tierras".
- Jefe final Zaragoza: La Ronda de Boltaña - "Avispas en el cierzo".
- Fases intermedias Expo: Oregon TV - "Soy de Zaragoza".
- Jefe final Expo: Amaral - "Llegará la tormenta".
- Fases intermedias Aragón: Los Berzas - "Yo amo el jamón".
- Jefe final Aragón: José Antonio Labordeta: "Canto a la Libertad". Canción introducida como homenaje al mismo.
- Pantalla de Fin de Partida: El Vicio del Duende - "No hay salida".

Con objeto de evitar problemas de copyright se ha contactado con los grupos que forman parte de la banda sonora solicitándoles permiso para utilizar sus canciones. Por ahora, en el momento de la realización de la memoria, se ha recibido respuesta afirmativa (permitiendo el uso de cualquiera de sus canciones) de los siguientes grupos (ver figura 3.4):

- Los Gandules.
- Los Berzas.
- La Ronda de Boltaña.

Por último, como hemos comentado en otras ocasiones, durante el juego se producirán distintos sonidos al disparar el jugador, moverse los invasores, aparición de potenciadores, etc.

Figura 3.4: Grupos colaboradores con MañoInvaders.

3.11. Menús de Pausa y de Fin de Partida

En cualquier momento del juego será posible pausar la partida, deteniendo todos los objetos en pantalla e introduciendo un menú que nos dará la opción de seguir jugando o salir. Si optamos por el botón Salir, nos llevará al menú de fin de partida; por el contrario, si optamos por continuar nos devolverá al juego.

En caso de que perdamos las tres vidas iniciales, que seamos invadidos o como hemos comentado antes, decidamos salir, el juego nos llevará al menú de fin de partida, en el cual si nuestra puntuación es mayor que cero, nos solicitará un nombre de usuario para almacenar la puntuación.

Por otro lado, se establece que no haya ninguna forma de conseguir vidas extras, puesto que se le quiere dar al juego ese aire *Survival* (juego de supervivencia) que también tenía "Space Invaders". Es decir, la idea es aguantar todo lo posible.

Capítulo 4

Implementación de Maño Invaders

En la fase de diseño quedó definido cómo iba a ser el juego y cómo debía comportarse el mismo. Es decir, quedaron definidos todos aquellos aspectos que podrían abstraerse de la forma de hacerlo. Por lo tanto, la fase de implementación es aquella en la que pasamos a abordar cómo hacer el juego propiamente, utilizando las herramientas de desarrollo pertinentes.

En el presente capítulo abordaremos todos los pasos para llegar a crear Maño Invaders, sin llegar a entrar en un alto nivel de profundidad. Dicho nivel de detalle se abordará ampliamente en el Anexo C. Así mismo, se ha omitido de este capítulo el proceso de aprendizaje en *GameMaker*, introduciendo una guía detallada de iniciación al mismo en el Anexo B.

A modo introductorio, *GameMaker* nos va a permitir:

- Trabajar con lo que él mismo denomina *room*, que no son más que pantallas entre las que el juego va iterando.
- Definir *sprites* sobre los que introduciremos imágenes.
- Crear objetos, a los que vincularemos los *sprites* y sobre los que añadiremos acciones y eventos. Estos objetos los añadiremos a las *rooms*.

4.1. Pantallas del juego

En Maño Invaders cada pantalla es implementada con una *room* distinta, salvo en el caso de las fases del juego, que todas las fases se implementan en la misma *room* con una serie de parámetros que nos permiten pintar una cosa u otra.

Como ayuda para crear el resto de objetos en las *room*, utilizaremos lo que denominaremos "objetos controladores de pantalla", que no son más que simples objetos de *GameMaker* a los que daremos una funcionalidad especial. La idea de estos objetos es añadirlos a la *room* para que comiencen desde un principio en ella y se

encarguen de inicializar y posicionar al resto de objetos. Se encargarán también de almacenar atributos que serán utilizados por el resto de objetos.

4.1.1. Menú principal

Con objeto de implementar el menú principal:

- Creamos una *room* específica sobre la que añadir objetos simples, que no llevan ninguna lógica, como pueden ser el fondo dinámico, el título del juego y el logotipo de "RetroAcción".
- Añadimos un objeto controlador para inicializar las variables globales, recuperar puntuaciones y configuraciones, sirviéndonos de "puente" con el resto de pantallas.
- Se crean unos objetos para implementar los botones. Estos a su vez tendrán un objeto padre, encargado de encapsular todo el comportamiento común a ellos. Añadimos los eventos específicos para poder interactuar con los botones mediante eventos de ratón.
- Se añade un controlador, que controle los eventos de teclado para movernos entre los botones utilizando las flechas direccionales.
- Por último, se añade un objeto para representar el botón de activación y desactivación del sonido en la parte superior derecha de la pantalla.

4.1.2. Menús auxiliares

Dentro de los menús auxiliares podremos distinguir los siguientes:

- Menú de puntuaciones:
  Al menú de puntuaciones accedemos mediante un evento del controlador del menú principal. Acto seguido, creamos una nueva *room* para el menú de puntuaciones sobre la que añadiremos el mismo objeto de fondo que en el menú principal. Se añade un nuevo objeto que representa al título, un objeto botón para volver atrás, con una lógica de transición de imagen similar a los definidos en el menú principal y, por último, un objeto controlador.
  El objeto controlador de esta fase, es el encargado de recuperar las diez máximas puntuaciones almacenadas en local y de pintarlas. Pintará pues, tres columnas, una para indicar la posición, otra para pintar el nombre de usuario y por último la puntuación.
- Menú de créditos:
  Este menú es idéntico al de puntuaciones excepto en el objeto controlador que se encarga de su lógica. Mientras que en el anterior se encargaba de pintar las

puntuaciones en este tiene asociado un *sprite* sobre el cual se encargará de producir efectos de *fade_in* y *fade_out* para realizar transiciones entre sus imágenes. De este modo se consigue que iteren los créditos.

– Menú de instrucciones:
  Este menú es prácticamente idéntico al anterior, pero cambiando el *sprite* asociado al objeto controlador. En este caso las imágenes serán las encargadas de darnos las instrucciones de cómo jugar y de cuál es el objetivo del juego.

4.1.3. Fases del Jugador

A la fase inicial en la que comenzará el juego se accede igual que a las anteriores: desde el controlador del menú principal.

Para definir la lógica de las fases nos servimos de:

– Definir un objeto que llevará asociado un fondo de estrellas estático sobre el que posicionamos el resto de objetos.
– Crear un objeto controlador que se encarga de pintar el resto de objetos.
– Crear una variable global que le indica al controlador qué objetos concretos debe pintar según la fase.
– Definir unos atributos en el controlador y que son susceptibles de ser utilizados más adelante por otros objetos (número restante de enemigos por fila, enemigos destruidos, si el jugador puede moverse, etc).
– Inicializar algunos objetos importantes, desde el controlador, entre los que figuran: la nave del jugador, vidas restantes, invasores o jefes finales en su defecto, etc (todos estos objetos se detallarán en próximos apartados).
– Inicializar desde el mismo dos nuevos controladores más específicos, con las funciones de: realizar las transiciones al comenzar y al finalizar una fase.

4.2. Nave del jugador

La elaboración del *sprite* de la nave del jugador se realiza mediante *Adobe Photoshop*[1] y a partir del cartel de RetroMañía 2007, obteniéndolo de manera sencilla (ver figura 4.1). Introducimos dicho *sprite* en *GameMaker* y lo asociamos a un objeto en el que insertamos toda la lógica de la nave del jugador.

Mediante un atributo del controlador establecemos que su movimiento responda a una velocidad máxima de desplazamiento para evitar que la nave se mueva demasiado rápida por la pantalla. Definimos los eventos encargados de tratar con los casos en los

[1] http://www.adobe.com/es/products/photoshop.html

que el usuario sea: invadido, destruido, se mueva hacia la izquierda, hacia la derecha y por último dispare.

Figura 4.1: Nave del Jugador: El Pilar

4.3. Invasores

Los *sprites* de los invasores originales se crean al igual que la fase del jugador a partir del cartel de RetroMañía , pero con la ayuda del *Photoshop* podemos conseguir darles el "efecto cachirulo" de los adoquines de Zaragoza intercalando el color de cada uno con cuadrados negros (ver figura 4.2).

Figura 4.2: Adoquines invasores

El diseño del resto de invasores se realiza utilizando nuevamente *Photoshop*: con imágenes de un tranvía real y frutas de Aragón, indexándolas posteriormente para reducir su paleta a cuatro colores y obtener un resultado pixelizado y que no desentona con los invasores ya creados (ver figuras 4.3 y 4.4).

Figura 4.3: Tranvías invasores

Figura 4.4: Frutas invasoras

Una vez implementadas todas las imágenes, el siguiente paso es introducirlas en *GameMaker*, introduciéndolas en un *sprite* por fila y asignando cada *sprite* a un objeto. Así pues, el objeto de la fila roja tiene las imágenes de todos los invasores rojos; esta

situación ocurre con el resto de invasores. La decisión de elegir la imagen con la que empiezan los invasores al comienzo de una fase se delega al controlador. Por último, todos estos objetos representantes de cada fila de invasores, tienen como padre al mismo objeto, que lleva la lógica genérica de todos ellos.

4.4. Movimiento de los invasores

En la fase de análisis y diseño establecimos como tenía que ser la sincronía de movimiento de todos los invasores. Con objeto de implementar dicha sincronía tomamos las siguientes decisiones:

- Crear un nuevo objeto encargado de controlar el movimiento.
- Introducir alarmas en el objeto controlador para ir moviendo las filas de una en una.
- Delegar en cada una de las alarmas creadas el movimiento de una fila concreta de invasores, siempre y cuando queden invasores en dicha fila.
- Comprobar en cada una de las alarmas si la fila a mover o alguna de las filas superiores se encuentran cerca de los límites de la pantalla, momento en el cual se procede a descender a todos los invasores.
- Comprobar si al descender los invasores, el jugador queda "invadido" (superamos la coordenada y definida), siendo enviado un evento encargado de notificar al objeto usuario de dicha invasión.
- Condicionar la velocidad de los enemigos al ser eliminados. Esta velocidad se disminuirá en el evento de colisión de los mismos llegando a un mínimo de dos *ticks*.
- Condicionar la velocidad inicial de movimiento a la fase en la que nos encontremos. Disminuyendo dicha velocidad conforme superamos fases hasta llegar a la velocidad mínima de dos *ticks*.
- Condicionar la posición de partida sobre la coordenada y a la fase (cada vez aparecerán más abajo).

4.5. Barreras defensivas

El diseño de las primeras barreras defensivas (ver figura 4.5) de *GameMaker* se obtiene del cartel de RetroMañía. Tomando como ejemplo estas tres barreras y siguiendo el mismo proceso utilizado en los invasores mediante *Photoshop*, se realizan el resto de las definidas en el apartado de diseño (ver figuras A.23 a A.31).

Las barreras están realizadas "a modo rejilla" (ver figura 4.6), es decir, una barrera de Maño Invaders se compone de una serie de objetos posicionados en pantalla de tal

Figura 4.5: Barreras fase 1: originales del cartel de Retro Mañía

modo que se encuentran tocándose unos con otros, dando la impresión de formar todos ellos una misma barrera. De este modo conseguimos que cada porción de barrera actúe de manera independiente y pueda ser eliminada individualmente. Por cada objeto de la rejilla, tenemos asociado su correspondiente trozo de imagen. La división es trivial, puesto que *Photoshop* nos permite partir imágenes fácilmente en los trozos que le indiquemos por ancho y por alto.

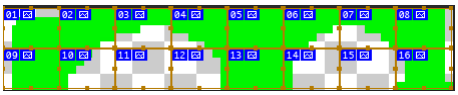

Figura 4.6: Ejemplo de barrera dividida a modo "rejilla"

Todos los objetos que representan un trozo de la barrera tienen como padre a un mismo objeto que la única lógica que recoge es que al recibir un disparo, de los invasores o del jugador, es destruido. Esto visualmente va produciendo el efecto de que la barrera está siendo destruida poco a poco.

En cada fase, el controlador de las fases se encarga de qué barreras defensivas se deben pintar y cómo debe hacerse.

4.6. Comportamiento de los disparos

Como vimos en el apartado de diseño, tenemos dos tipos de disparos: aquellos realizados por el usuario y los realizados por las naves invasoras. En ambos casos, realizamos los *sprites* de los disparos (uno para el usuario y cuatro distintos para los invasores) utilizando el editor de imágenes de *GameMaker* (ver imagen 4.7) y así mismo, encapsulamos la lógica en dos objetos distintos: uno que sirve para el disparo del usuario y otro que actúa como padre de los objetos representativos de cada uno de los *sprites* de disparo de los invasores.

4.6.1. Disparos del jugador

Para implementar toda la lógica que rodea a los disparos realizamos lo siguiente:

– En la creación de los mismos, se les define un movimiento en dirección vertical, en sentido negativo al eje y.

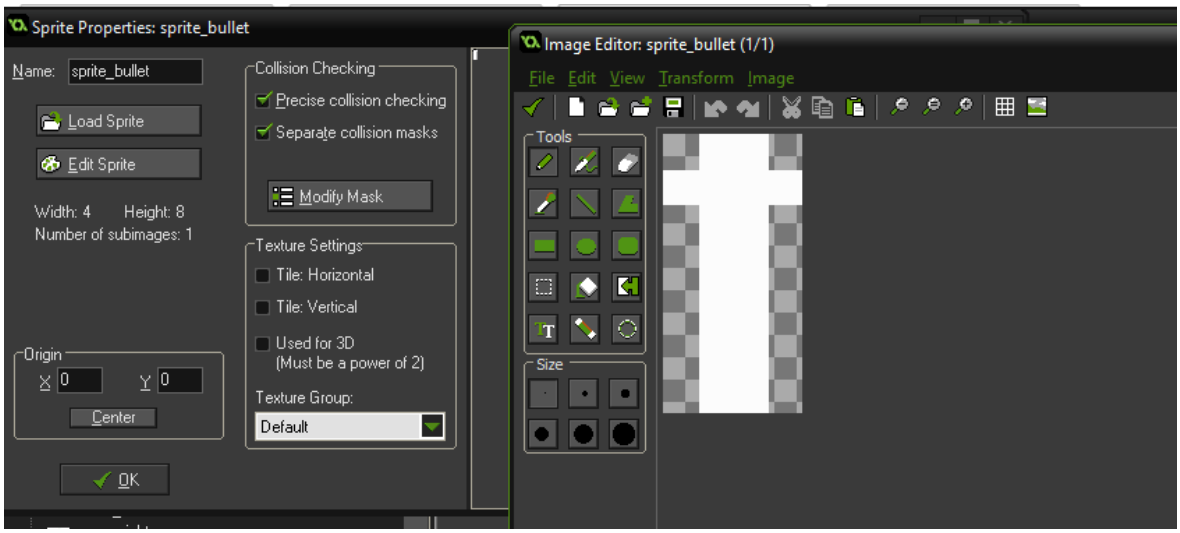

Figura 4.7: Ejemplo de creación de una imagen utilizando el editor de *GameMaker*

- Establecemos su velocidad en base a la variable global de velocidad de disparos del jugador.
- Introducimos un sonido en su evento de creación.
- Controlamos vía atributo global si se puede volver a disparar o no.
- Activamos el atributo global (vuelve a dejar disparar) en los casos que impliquen destrucción del disparo (salida de los límites de la pantalla o colisión) o bien este cruce la mitad de la pantalla.
- Comprobamos en su evento de colisión con enemigos si no queda ningún enemigo en pantalla, momento en el cual se invoca al controlador de fin de fase.
- Añadimos sonidos a las colisiones.
- Capturamos las interrupciones provocadas por la tecla "barra espaciadora" para intentar inicializar los disparos.

4.6.2. Disparos de los invasores

Para implementar los disparos de los invasores seguiremos el siguiente proceso:

- Lo primero de todo, crear los *sprites* de cada tipo de disparo junto con sus objetos asociados. A todos los objetos les añadimos un nuevo objeto padre del cual heredan su lógica.
- Definimos dentro del objeto padre, un evento de creación en el cual se establece que los disparos se mueven en el eje y, así como en sentido favorable al mismo.
- Definimos una velocidad de movimiento de los disparos constante e igual a la variable global representante de la misma.
- Añadimos los eventos de colisión con jugador, disparo del jugador y barreras.

Tras lo anterior, tenemos definido los objetos en sí, pero todavía faltará definir en qué casos los invasores crean objetos disparo, para ello:

- Establecemos una alarma que se ejecuta cada cierto tiempo en cada uno de los invasores.
- Cada una de estas alarmas, comprueba que los invasores pueden disparar (atributo específico activo) y que no exista ningún invasor en la misma posición en una fila inferior.
- En caso de cumplirse el punto anterior, comprobamos que todavía queden más invasores que los iniciales por fila. En caso afirmativo se comprueba la distancia horizontal con el usuario, si esta es menor de 50 el peso del invasor será de cuatro, si no, de uno.
- Mediante la función *random* de *GameMaker* obtenemos un número aleatorio entre cero y siete. Si dicho número es inferior al peso del invasor, este creará un objeto disparo, que partirá desde su misma posición y tendrá el mismo color.
- Desactivaremos la posibilidad de disparo en el momento que un invasor inicialice un disparo.
- Detectamos y establecemos mediante eventos que si cualquiera de los disparos de los invasores deja la pantalla o colisiona, vuelva a activar la posibilidad de disparo.

4.7. Potenciadores

Con objeto de implementar los potenciadores (ver figura 4.8):

- Creamos un *sprite*, un objeto padre de los potenciadores y un objeto para tratar cada uno de los mismos independientemente.
- Añadimos dos nuevas imágenes de la nave del jugador con los colores de las banderas de los potenciadores al *sprite* actual de la nave del jugador (ver figura B.1).
- La aparición o no de los potenciadores recae sobre el controlador de fase, que mediante alarmas los creará.
- En la creación de los potenciadores, programamos unas alarmas que controlan el tiempo máximo que van a estar en pantalla los mismos, tras el cual se destruirán.
- Definimos una distancia máxima aleatoria hasta la cual se moverán al aparecer en pantalla.
- Establecemos un evento de colisión del potenciador con el disparo del jugador, momento en el cual pasamos a activar el efecto del potenciador en el atributo correspondiente del controlador de fase.

– Al obtener un *booster*, cambiamos la imagen asociada a la nave del jugador, adecuándola al potenciador conseguido, produciendo un sonido y programando una alarma encargada de emitir un sonido en el futuro avisando al usuario de que el tiempo de uso está llegando a su fin.
– Se programará otra alarma de duración superior que precisamente ponga fin al potenciador, terminando el efecto del mismo y devolviendo a la nave del jugador a su imagen original.
– Modificamos los eventos externos para regular el comportamiento de "invencibilidad" y "multi-disparo".
– En el caso de tener activo el "multi-disparo", al disparar la nave, inicializamos dos nuevos objetos disparo del jugador, pero de color rojo, que comenzarán en la parte superior de las torres del Pilar.
– En el caso del *booster* de "invencibilidad" cualquier impacto que reciba el jugador, pasa a ser omitido.

Figura 4.8: Booster de multi-disparo e invencibilidad

Aprovechando que hemos implementado el *booster* de la invencibilidad modificamos el controlador encargado de crear la nave del jugador, ya sea al comienzo de la partida o tras haber sido destruida, para que durante un breve tiempo, la nave tenga una opacidad más baja y al mismo tiempo sea invencible. Para ello, simplemente modificamos el evento, cambiando la opacidad y activando el atributo encargado de mostrar si el efecto del *booster* está activo y por otro lado, activamos una alarma que desactive este efecto al poco tiempo. Con esto conseguimos que si el jugador pierde una vida, no pierda otra acto seguido.

4.8. Jefes finales

A todas las fases de los jefes finales (ver figura 4.9) accederemos vía el objeto controlador de fase, que al detectar que se trata de una fase de jefe final, no pintará invasores y barreras, si no que se encargará de instanciar a los objetos correspondientes a los jefes finales. Como decisión de implementación, se ha tomado que al tener todos los jefes un comportamiento bastante diferenciado se ha optado por que no partan de un mismo objeto en la jerarquía.

Figura 4.9: Los jefes finales de Maño Invaders.

4.8.1. Jefe final fases Zaragoza: Alcalde

Para crear toda la lógica del jefe final "Alcalde" seguimos el siguiente procedimiento:

- Primero de todo creamos las ocho imágenes (ver figura A.32) correspondientes al mismo, basándonos en los criterios establecidos en la fase de diseño y las introducimos en un *sprite* y objeto.
- Creamos un objeto que representa la vida restante del "Alcalde" en pantalla.
- El movimiento normal del jefe se define en el evento *Step* junto con la elección de la imagen correspondiente al jefe. Si no está saltando y no tiene muy cerca al jugador, calculamos la dirección de movimiento del jefe haciendo que se mueva en sentido horizontal siguiendo al jugador.
- Mediante el evento de colisión con disparos, restaremos la vida y comprobaremos si su vida ha llegado a cero para avanzar de fase y hacer explotar al "Alcalde".
- Creamos una alarma que provocará la vibración del "Alcalde" (desplazamiento rápido a lo largo del eje y).
- Definimos una alarma que servirá para dar comienzo con la ráfaga de saltos del alcalde haciendo uso de un contador de saltos que se decrementa con cada salto del "Alcalde".
- Mediante el uso de esta alarma definimos el ángulo de disparo de cada uno de los disparos del siguiente modo:

$$Disparoazul = 250 - 10 * (5 - saltosrestantes) \tag{4.1}$$

$$Disparorojo = 265 - 5 * (5 - saltosrestantes) \tag{4.2}$$

$$Disparorosa = 275 + 5 * (5 - saltosrestantes) \tag{4.3}$$

$$Disparoamarillo = 290 + 10 * (5 - saltosrestantes) \tag{4.4}$$

- Definimos una nueva alarma encargada de que el jefe lance armas durante su movimiento. Para ello primero de todo creamos los objetos de las armas y posteriormente, en la alarma, calculamos un número aleatorio para que dependiendo de si es par o impar cada vez instanciemos un arma distinta.
- En la inicialización de las armas las dirigimos siempre hacia el punto donde se encuentre el jugador en ese mismo instante.
- Para que el jefe no siempre lance las armas en el mismo intervalo de tiempo, usamos la siguiente formula:

$$Alarma = ticks_{min} + random(ticks_{max} - ticks_{min}) \tag{4.5}$$

  Siendo $ticks_{min}$ los ticks mínimos hasta los que se volverá a ejecutar (variable global definida al comienzo al comenzar las fases) y $ticks_{max}$ los máximos.
- Por último creamos una nueva alarma que al derrotar al jefe final nos llevará a la siguiente fase.

4.8.2. Jefe final fases Expo: Fluvi

El procedimiento inicial para "Fluvi" es idéntico al del "Alcalde", por lo tanto seguimos todos los pasos descritos en el "Alcalde" y modificamos las imágenes de los estados del mismo (ver figura A.33), de las armas, y añadimos los sonidos propios de "Fluvi".

El proceso concreto diferenciador de "Fluvi" será el que seguimos a continuación:

- Añadimos una ráfaga prevía a la de saltos.
- Definimos un atributo indicador de esta nueva ráfaga.
- Programamos una nueva alarma encargada de comenzar la ráfaga.
- En la nueva alarma se comprueba en qué posición se encuentra "Fluvi", dirigiéndolo al doble de velocidad hacia el lado contrario de la pantalla hasta llegar a los límites de la misma.
- Durante la ráfaga programamos una alarma encargada de lanzar armas a su paso, la frecuencia de esta alarma será del doble que la encargada de lanzar armas de normal.
- En el momento en el que detectemos que "Fluvi" llega a uno de los límites de la pantalla, daremos comienzo a su ráfaga de saltos.
- Conseguimos que "Fluvi" se mueva dando saltos, comprobando en todo momento (salvo en la ráfaga de saltos) que si "Fluvi" se encuentra en su posición inicial, automáticamente se le establezca una velocidad en contra del eje y, así como una gravedad a favor del mismo eje.
- Por último, modificamos su vida inicial, estableciéndola 20 puntos superior a la del "Alcalde".

4.8.3. Jefe final fases Aragón: SuperMaño

Partimos de un objeto idéntico en lógica a "Fluvi" pero con los *sprites* creados para el "SuperMaño" (ver figura A.34), así como dos nuevas armas, más vida inicial (130 unidades) y sonidos específicos.

En este caso no será necesario crear nuevas alarmas ya que heredará las ráfagas de "Fluvi", pero añadiéndoles nuevos comportamientos:

- Definimos un contador del número de rebotes de la ráfaga lateral. Cada vez que llegue a los límites de la pantalla se decrementará, de tal modo que comprobaremos si llega a cero, dando comienzo a la ráfaga de disparos.
- En caso de que el contador anterior no llegue a cero, comenzará de nuevo el movimiento lateral en sentido contrario.
- Definimos otro contador, que regulará las repeticiones de la ráfaga de saltos.
- Cada vez que completemos seis saltos, el proceso de amplitud de los ángulos de disparo cambiará, pasando a decrementarse en la siguiente iteración.
- En la ráfaga de saltos ahora se lanzarán armas aleatoriamente.

4.9. Música del juego

Para la introducción de la música (ver Sección B.2), primeramente obtenemos los archivos en formato *.mp3* de sus creadores. Una vez tenemos los ficheros, con cualquier programa de edición de sonido ajustamos su amplitud (volumen) al deseado.

Posteriormente, procedemos a añadir las canciones de las fases en el controlador de comienzo de fase, que es el encargado de que en cada fase suene una canción determinada u otra. El resto de canciones se añaden así mismo en sus controladores pertinentes. Añadiremos también, de la misma manera, el resto de sonidos a los objetos pertinentes. Ejemplos de sonidos son los producidos al disparar, al explotar un invasor, al explotar la nave del usuario, etc.

Al añadir las canciones y publicar el juego nos encontramos con el problema de que todas las canciones pensadas en un principio tienen copyright. Esto supuso un esfuerzo extra ya que fue necesario contactar con los grupos en cuestión. En el momento de la realización de la memoria se ha recibido confirmación por parte de tres grupos:

- Los Gandules
- Los Berzas
- La Ronda de Boltaña

Con objeto de poder liberar el juego y hacerlo llegar a mayor público, se toma la decisión de subir el juego utilizando solo canciones de los mismos, pasando a ser el nuevo *tracklist* (en el momento de realización de la presente memoria) el siguiente:

– Pantalla inicial: La Ronda de Boltaña - “La Tronada”.
– Fases intermedias Zaragoza: Los Gandules - “Malos tiempos para Sergei”.
– Jefe final Zaragoza: La Ronda de Boltaña - “Avispas en el cierzo”.
– Fases intermedias Expo: Los Gandules - “Carroña”.
– Jefe final Expo: Los Gandules - “Bayas-Bayas”.
– Fases intermedias Aragón: Los Berzas - “Yo amo el jamón”.
– Jefe final Aragón: José Antonio Labordeta: “Canto a la Libertad”. Canción introducida como homenaje al mismo.
– Pantalla de Fin de Partida: Los Gandules - “La ruindad del tabernero”.

4.10. Puntuaciones locales

Como hemos comentado en otras secciones, *GameMaker* nos facilita tanto un vector para almacenar las mejores puntuaciones, como una variable global donde almacenar la puntuación en curso. Al comienzo de las fases de juego, reiniciamos dicha puntuación a cero, para que comience a contar.

En el controlador de fase añadiremos un evento de dibujo que pinte la puntuación actual en la parte superior de la pantalla así como la máxima histórica, cada una bajo los literales correspondientes.

Cada vez que alcancemos un enemigo en pantalla aumentaremos la puntuación (ver Tabla 4.1), así pues será necesario añadir en los eventos de colisión con los disparos del jugador el incremento en la puntuación.

En el caso de los jefes finales, estos nos otorgarán puntos por cada disparo recibido y no a su destrucción.

Enemigo	Puntuación
Invasores Rojos	40
Invasores Amarillos	30
Invasores rosas	20
Invasores azules	10
Boosters	60
Alcalde	15 cada disparo
Fluvi	20 cada disparo
SuperMaño	25 cada disparo

Tabla 4.1: Tabla de puntuaciones.

4.11. Menús de Pausa y Fin de Partida

4.11.1. Menú de Pausa

Para tratar las acciones que ocurren al pulsar el menú de pausa procedemos del siguiente modo:

- Añadimos eventos de usuario a todos los objetos. Uno de ellos se encarga de almacenar el estado de los mismos, pausándolos y el otro de devolverles el estado almacenado, reanudándolos.
- En el controlador de fase recogemos el pulsado de la tecla *escape* que genera el menú de pausa vía un nuevo controlador.
- En el nuevo controlador, pausamos sonidos y enemigos e inicializamos los objetos pertenecientes al título y botones de "continuar", "salir" y desactivación de sonido.
- Añadimos la lógica pertinente a los botones: el botón "continuar" restablece el juego, mientras que el botón "salir" modifica el menú de pausa, preguntándonos si estamos seguros de que queremos salir del juego.
- En caso de confirmar la salida, volveríamos al menú principal. En caso de cancelar la salida, volveríamos al menú de pausa normal.

4.11.2. Menú de Fin de Partida

Al menú de "Fin de Partida" llegamos siempre que se mueva a la *room* de fin del juego. En el mismo definimos lo siguiente:

- En primer lugar, haciendo uso de un controlador, validamos que la puntuación sea mayor que cero. En caso negativo saltamos el menú y vamos al menú principal.
- En caso de que la puntuación sea mayor, mostramos un cuadro de diálogo preguntando por el nombre de usuario.
- Validamos la información introducida: si no es válida volvemos a preguntar y si está vacía tratamos como si pulsara cancelar, es decir, como si no deseara almacenar su puntuación. En este caso, informaremos con otro diálogo que su puntuación no se almacenará, preguntando si está seguro.
- En caso de que sí que introduzca correctamente el nombre de usuario, la puntuación será almacenada en memoria en el vector de puntuaciones y en el fichero codificado.
- Por último siempre volveremos al menú principal.

Capítulo 5

Evolución de Maño Invaders: Retro Mañía '16

En la realización de Maño Invaders surgió la posibilidad de presentar dicho juego durante el evento RetroMañía 2016, evento ideal puesto que representaba el décimo aniversario del evento en cuyo cartel está basado Maño Invaders, así como una gran oportunidad de dar a conocer el juego y realizar una evolución del mismo.

Previo a presentar Maño Invaders, se recibió *feedback* de algunos integrantes de la asociación "RetroAcción", encargada de la organización del evento RetroMañía. Mediante este *feedback* y solicitudes se definieron e implementaron una serie de requerimientos y mejoras que debía contener el juego de cara a su salida al público.

En el presente capítulo se definen todas estas mejoras, detallando los requerimientos y levemente su implementación (ver Capítulo C para mayor detalle). Se finaliza con unas conclusiones finales del evento RetroMañía y lo que este supuso para el proyecto.

5.1. Nueva pantalla de Configuración

Desde el menú principal, se añadirá un nuevo botón para poder acceder a un nuevo menú de configuración. En dicho menú de configuración en ambas plataformas daremos la posibilidad de introducir un filtro de *Scanlines* que nos simule visualmente estar jugando en una recreativa.
Para implementar lo definido anteriormente:

- Utilizamos un *shader* de *GameMaker*, que básicamente son filtros que se pueden aplicar sobre los juegos.
- Descargamos un *shader* libre para poder simular el efecto *Scanlines* en nuestro juego y lo adecuamos a nuestras necesidades.
- Creamos la pantalla de configuración del mismo modo que hicimos con el resto.
- Añadimos los objetos que representan los literales de las distintas opciones

de configuración, así como los dos botones pertenecientes a las opciones de configuración: botón “Sí” y botón “No”.

- Posicionamos los dos botones tocándose, y definimos el comportamiento de tal modo que cuando pulsemos uno, el otro se deseleccione y viceversa. De modo que actúen como interruptores.
- Siempre que realicemos un cambio en el botón seleccionado, actualizaremos el valor de la variable global y la escribiremos en el fichero de configuraciones.

5.2. Transiciones entre pantallas

Al comienzo de cada fase, debe definirse una secuencia de *fade in*, así como a su consecución, una de *fade out* que permita una transición entre fases más amigable.

Desde el punto de vista auditivo, el volumen de la canción de cada fase comenzará sin oírse e irá ganando volumen poco a poco hasta establecerse en el volumen normal; el caso inverso ocurrirá al terminar una fase, pasando del volumen normal al volumen cero. Con la imagen ocurrirá lo mismo: inicialmente el fondo tapará la pantalla de la fase y poco a poco este se irá difuminando apareciendo la fase junto con una cuenta atrás; por el contrario, al terminar la fase, comenzará a oscurecerse la pantalla hasta que esta se tape. En caso de que perdamos la partida, la transición a la pantalla de ”Fin de Partida” dejará de ser brusca, pasando a ser más liviana y será distinta según la plataforma.

Una vez definidos los requerimientos de cómo deben ser las transiciones, las implementamos del siguiente modo:

- El *fade_in* visual, lo logramos mediante una alarma del objeto controlador. Esta alarma se llama así misma modificando la opacidad de un fondo negro en cada momento.
- Con el sonido, logramos el *fade_in* estableciendo un volumen inicial, un volumen final y el tiempo para obtener la ganancia del mismo.
- El *fade_out* se implementa de manera idéntica, pero a la inversa en ambos casos.
- La transición a la pantalla de “Fin de Partida”, se realiza mediante una aparición, poco a poco, de un fondo negro sobre el que ubicamos unas letras en grande con el rótulo “GAME OVER”. Una vez estas letras sean totalmente opacas, introducimos un movimiento sobre las mismas, que acabe dejándolas en la parte superior de la pantalla, a modo título y pasando a la pantalla de “Fin de Partida”.
- Por último se añaden interrupciones de la tecla “espacio” para saltar las transiciones.

5.3. Mejora de la dificultad

En el caso de los jefes finales se les añadirá una dificultad creciente cada vez que nos enfrentemos a ellos, es decir, si conseguimos pasar las 12 fases y nos volvemos a enfrentar a ellos, estos tendrán más vida inicial y sus ráfagas serán cada vez más duraderas. Por otro lado, se define una nueva ráfaga final que ocurrirá al derrotar a cada uno de los jefes finales y que será propia de cada jefe final. Durante esta nueva ráfaga, el jefe final nos atacará durante una duración definida. Si conseguimos aguantar dicha ráfaga, el jefe final explotará, siendo derrotado.

5.3.1. Jefe final fases Zaragoza: Alcalde

Cada vez que superemos las 12 fases y nos volvamos a enfrentar al "Alcalde", este aumentará su ráfaga de saltos en una nueva iteración. De tal modo que en cada iteración irá abriendo o cerrando los ángulos de disparo. Por otro lado, se añadirá una ráfaga final cuando su vida llegue a cero antes de que este explote. Esta nueva ráfaga será una ráfaga de saltos, pero lanzando sus armas en lugar de disparos, al tiempo que suena una risa malévola y en la pantalla aparece en letras que van creciendo y disminuyendo de tamaño la palabra "Peligro". En caso de que el "Alcalde" nos quite una vida, este se "reirá".

Primero de todo, para implementar la ráfaga de saltos en función del número de fase en la que nos estemos enfrentando al "Alcalde" realizamos la siguiente cuenta:

$$NumIteraciones = floor(\frac{FaseActual}{12}) \tag{5.1}$$

Siendo *floor* un redondeo a la baja hacía el entero inferior.

Una vez tenemos el número total de iteraciones, implementamos las iteraciones de las ráfagas de salto del mismo modo que hicimos con "SuperMaño", es decir, con un contador que nos marque el número de iteraciones y que se decremente.

Por último, la ráfaga final, no será más que una ráfaga simple de salto, pero cambiando los objetos lanzados por armas. Al mismo tiempo que se produzca esta ráfaga introducimos una nueva alarma que vaya incrementando y decrementando el escalado de la imagen del alcalde, de tal modo que parezca que se hincha y deshincha hasta finalmente explotar. El escalado que establecemos como máximo es de 1.25 hacia arriba y 0.25 hacia abajo, con el objetivo de evitar que o bien no se vea casi o bien sobresalga de la pantalla.

5.3.2. Jefe final fases Expo: Fluvi

En el caso de "Fluvi", condicionaremos también la ráfaga de salto de tal modo que al igual que ocurría con el "Alcalde", pero intercalando una iteración en la que lance disparos y otra en la que lance armas.

Por último, también introduciremos una ráfaga final. Esta ráfaga consiste en que "Fluvi" se mueva de lado a lado de la pantalla con el doble de velocidad lanzando armas sin cesar a su paso al tiempo que va aumentando y disminuyendo de tamaño.

Esta última ráfaga la implementamos del mismo modo que hicimos la ráfaga de desplazamiento lateral para el "SuperMaño", en el cual se hacían dos pasadas: una hacia el lado contrario de donde se ubicaba el mismo y acto seguido un regreso hasta el otro lado. En este caso, lo que cambia es que no establecemos que sean concretamente un número de movimientos de lado a lado concreto; si no que se mueve siempre de lado a lado hasta que una alarma sea la que se encargue de cesar el movimiento y hacer explotar a "Fluvi". El crecimiento del tamaño de la imagen y decrecimiento lo conseguiremos del mismo modo que con la mejora del "Alcalde".

5.3.3. Jefe final fases Aragón: SuperMaño

Para la mejora de "SuperMaño", condicionaremos tanto su ráfaga de movimiento lateral, como su ráfaga de saltos, de tal modo que en cada uno de los casos, el número de iteraciones tanto en su ráfaga de movimiento como en su ráfaga de saltos será el siguiente:

$$NumIteraciones = 2 + floor(\frac{FaseActual}{12}) \tag{5.2}$$

Como ya teníamos implementado en el mismo el control de iteraciones, condicionar dichas iteraciones en el mismo es el caso más fácil de los tres jefes.

Por último, añadimos también una nueva ráfaga final previa a explotar, en la cual, al igual que en los jefes anteriores, aparecerá la palabra "peligro" al tiempo que "SuperMaño" salta, aumentando y disminuyendo de tamaño y cayendo armas desde el cielo. Estas armas están separadas entre ellas para que nunca colisionen entre sí. Caerán siempre verticalmente, pero no todas al mismo tiempo, así pues como *GameMaker* permite que definamos coordenadas y negativas, haciendo uso de números aleatorios establecemos un minimo y un máximo de dicha coordenada para la instanciación de dicha variable. Esto obligará al jugador a estar en continuo movimiento.

Para implementar dicha ráfaga final, creamos una nueva alarma encargada de realizar los saltos al tiempo que condicionamos el evento *Step* para que se encargue de conseguir que "SuperMaño" salte siempre que vuelva a su coordenada inicial y

que en ese instante "lluevan' armas desde arriba. Otra alarma se encargará de ir decrementando e incrementando el tamaño, como hemos hecho con los otros jefes finales y por último también una alarma que se encargue de que en el tiempo programdo, explote.

5.4. Juego Multiplataforma: PC, Android y Recreativa

Que el juego debía ser jugable tanto en PC, como en dispositivos *Android*, como en maquinas recreativas fue uno de los requerimientos principales, sobre todo para que el juego pudiera llegar al máximo número de personas y para conseguir darle un toque clásico al jugarlo en recreativa. Este requerimiento era base y llevaría consigo muchos otros derivados del mismo.

Lo primero de todo para comenzar con las implementaciones fue definir unas constantes "PC", "ANDROID" y "RECREATIVA" para desde el juego, poder evaluar en que plataforma nos encontramos y de este modo ir por un camino u otro.

Así mismo, *GameMaker* nos permite definir distintas "Configuraciones". Estas configuraciones, no son más que perfiles sobre los que definimos una serie de constantes. Esto será importante ya que para cada plataforma crearemos un perfil distinto.

5.4.1. Modificaciones especificas para *Android*

La primera de las modificaciones necesarias para *Android* era que todo el control del juego debía ser táctil.

La implementación del control en menús y fases del juego en *Android* se realizo del siguiente modo(ver figura A.10):

- La navegación de menú, de manera sencilla, ya que en *GameMaker* es equivalente el tocar la pantalla con el dedo a realizar un *click*. Mientras que las teclas "retroceso" del móvil y del teclado también lo son.
- El movimiento de la nave del jugador se implementa importando un controlador *joystick* (palanca de mando virtual) de *GameMaker* sobre el cual añadimos interrupciones de movimiento.
- En el lado opuesto al *joystick* se incluye un botón de disparo.
- Ambos dos controladores se incluyen con una opacidad tal que no impidan visualizar correctamente la pantalla de juego.

Con objeto de que tanto zurdos como diestros puedan jugar sin problema, se posibilitará que desde el menú de configuración puedan elegir una nueva opción que

contemplará configuración para zurdos o para diestros. La diferencia entre ambas será que tendrán intercambiados de lado los botones de disparo y movimiento.
El anterior requisito implicará las siguientes modificaciones en la lógica:

- Modificamos la pantalla de configuración añadiendo una nueva opción (ver figura A.7) para seleccionar configuración para zurdos o diestros del mismo modo que añadimos la configuración de *Scanlines*.
- Almacenamos la configuración en fichero local para que esta se recuerde siempre.
- En el controlador de fase introducimos la lógica que nos permita ubicar los botones en el lado que corresponda, en función de la configuración.

5.4.2. Modificaciones especificas para Recreativa

Hoy en día un gran número de maquinas recreativas llevan por debajo un sistema operativo *Windows*, que utilizan para cargar los juegos. Esto supone que sea necesario establecer un *mapeo* entre las teclas del teclado de *Windows* y los botones de una maquina recreativa. Por otro lado, en las recreativas, cuando nadie jugaba siempre quedaban en *attract mode*. El *attract mode* era un estado, en el que se iban alternando las puntuaciones, instrucciones, créditos, así como en algunos casos, una demo dónde se veía alguna pantalla del juego en acción. Se define que este *attract mode* se llevará a Maño Invaders.

Las decisiones de implementación tomadas para añadir soporte a maquina recreativa(ver figura 5.1) fueron las siguientes:

- Se mantienen las teclas de PC, acordando el mapeo por parte de la maquina recreativa y modificando únicamente que la misma tecla de disparo sirva para seleccionar opciones.
- Con objeto de añadir el *attract mode*, en el caso de maquina recreativa, llegamos a un nuevo menú principal a la espera de un interrupción de teclado para comenzar a jugar.
- Si en el nuevo menú principal, estamos sin tocar nada durante un tiempo, saltaremos a la pantalla de puntuaciones, comenzando un bucle que nos mueve mediante alarmas entre todos los menús salvo el de configuración.
- En el *attract mode* añadimos un controlador que comprueba si una canción ha finalizado, dando comienzo a la siguiente.
- Recogemos interrupciones de la tecla espacio para salir del *attract mode*.
- Si durante un tiempo tampoco introdujéramos nombre de usuario en la pantalla de “Fin de Partida”, también caeremos en el *attract mode*.

Figura 5.1: Maño Invaders en una máquina recreativa de RetroMañía 2016.

Por último la pantalla de puntuaciones deberá cambiar ya que no podemos utilizar cuadros de dialogo como en PC o Android. Por lo tanto, inspirándonos en cómo se hacía antiguamente, mostraremos un teclado virtual en pantalla, que nos proponga en primer lugar introducir el nombre del jugador y en segundo lugar el email. Conforme vayamos seleccionando caracteres, se irán mostrando en la parte superior del teclado, para que veamos que estamos escribiendo.

Para implementar dicha pantalla (ver figura 5.2), necesitaremos:

- Creamos las imágenes para representar el teclado virtual de mayúsculas y minúsculas, así como un recuadro a modo selector que nos permita movernos por el teclado. Asociamos estas imágenes a objetos.
- En dos matrices (una por teclado de mayúsculas y otra por teclado de minúsculas) mapeamos las teclas en función de filas y columnas.
- Capturamos las interrupciones de las flechas direccionales que nos sirven para desplazar el recuadro por el teclado, a la par que actualizamos las coordenadas de la matriz en la que nos encontramos.
- Añadimos dos botones que nos permitan confirmar el texto finalmente introducido o pasar.
- Recogemos la interrupción de la tecla “espacio” para seleccionar un carácter
- A la vez que vamos introduciendo caracteres, pintamos la palabra seleccionada en la parte superior del teclado.

Figura 5.2: Resultado final de la pantalla de Game Over para máquina recreativa.

5.5. Puntuaciones competitivas online

Otro de los requerimientos principales fue que el juego almacenara las puntuaciones *online*. De este modo, independientemente de la plataforma en la que fuera jugado el juego los jugadores podrían competir entre sí y realizar un torneo en el evento RetroMañía 2016: presencialmente jugando en la maquina recreativa, en sus PCs o bien en cualquier lugar utilizando sus móviles *Android*.

Con el objetivo de poder contactar con los usuarios cuando finalice el torneo, al finalizar una partida se les preguntará junto al nombre de usuario el *email*. En caso de que deseen publicar su puntuación *online*, introducirán su *email* y en ese momento su puntuación e información relevante de la partida que han jugado se enviará al servidor. Siempre que comencemos una partida, el juego recuperará las diez máximas puntuaciones *online*, mostrándolas en la pantalla de puntuaciones. Así mismo, la máxima puntuación *online* se mostrará en cada una de las fases en la parte superior de la pantalla, junto a la máxima puntuación local. Por último, cada vez que se descarguen automáticamente las puntuaciones *online*, se almacenarán en un fichero codificado en local, por si en un momento dado no hubiera conexión. En caso de que no hubiera

conexión al finalizar una partida, la puntuación se almacenaría en un fichero codificado reintentando su envío en un futuro.

5.5.1. Implementación de la Arquitectura específica *online*

Para realizar la implementación *online* de la arquitectura, lo primero de todo es diseñarla (ver figura 5.3) estableciendo todas sus interconexiones y pasando posteriormente a encontrar un proveedor que nos facilite un servidor donde alojar nuestra aplicación. En el caso de Maño Invaders, se ha optado por *Openshift*[1], que facilita unos *slots* (espacios donde instalar aplicaciones) en los que instalar fácilmente una base de datos *MySQL* y un servidor de aplicaciones *Wildfly*. Una vez tenemos acceso a la base de datos*MySQL* y al servidor de aplicaciones *Wildfly*, lo primero de todo es acceder a la misma y definir los esquemas y tablas que vamos a utilizar. En nuestro caso solo tenemos dos tablas: una que servirá para almacenar datos relevantes a las puntuaciones máximas de cada usuario y otra tabla que almacenará todas los datos referentes a las partidas jugadas por cada usuario.

Una vez creado lo anterior, pasaremos a definir nuestra aplicación que conectará con la Base de Datos. Para crear nuestro proyecto se ha utilizado el entorno de desarrollo *Eclipse*, creando un proyecto *Maven* que nos permite importar cualquier librería necesaria en tiempo de compilación, y nos abstrae de esa labor, simplemente definiendo las librerías deseadas. El proyecto tendrá una capa de *JSF* que nos permitirá desde la vista acceder a unos controladores, que pueden tener tres tipos importantes de *scope* (duración de los mismos): petición, vista y sesión. Sobre uno de nuestros controladores se incorpora una *API Rest*, sobre la cual lanzar peticiones *GET* y *POST* para recuperar y recibir puntuaciones externamente del servidor. Por otro lado, la conexión con la base de datos se realiza mediante *Hibernate*. El control de sesiones y la interconexión entre elementos se delega a una capa de *Spring*, que conecta controladores *JSF* - Clases *Hibernate* - Base de datos.

Por último para facilitar la labor de creación de una web en la cual cualquiera pudiera visualizar las puntuaciones *online*, así como ser enlazada desde la página de RetroMañía (pudiendo mostrar en esta última las puntuaciones en tiempo real); se añade una capa de *Primefaces*, que junto con *JSF* nos permite utilizar componentes avanzados.

Esta web servirá para mostrar dos tablas: por un lado, la de las máximas puntuaciones de cada usuario y por otro lado, todas las partidas jugadas; siendo la primera de ellas la que utilizará la web oficial de RetroMañía para mostrar las

[1]https://www.openshift.com/

puntuaciones en tiempo real.

Los métodos del *API Rest* a la cual podremos enganchar el Maño Invaders, serán: un método que actuará de *dummy* para poder comprobar que el servidor está ok, un método textitGET para poder recuperar las 10 máximas puntuaciones y un método *POST* para introducir una puntuación en base de datos.

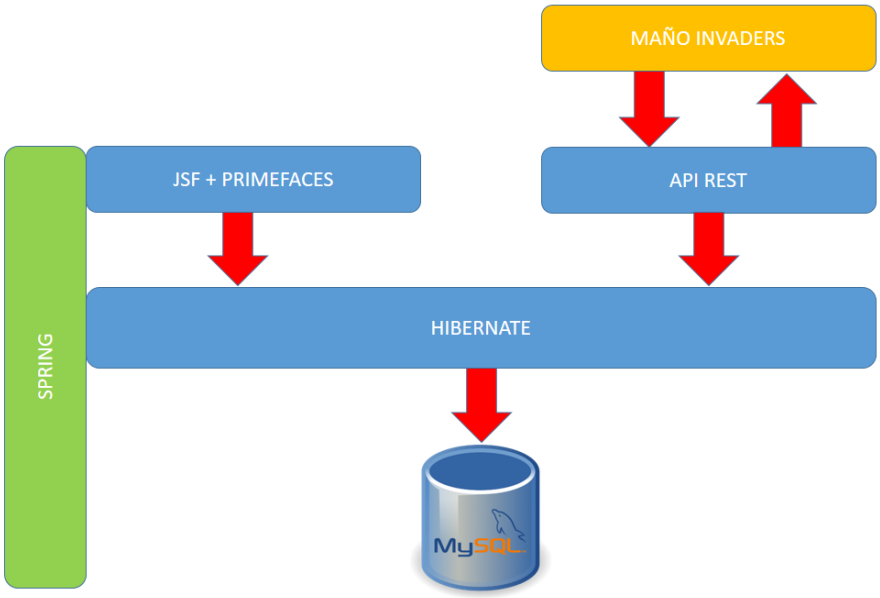

Figura 5.3: Arquitectura para la implementación de las puntuaciones online.

5.5.2. Implementación de la sincronización online desde Maño Invaders

Nada más comenzar el juego se añade un nuevo objeto que actuará de inicializador de toda la información relevante. Este objeto será el nuevo encargado de recuperar todos los parámetros de configuración, inicializar algunas variables globales necesarias y de realizar una solicitud *GET* al servidor para recibir en un objeto *JSON* con las puntuaciones actualizadas. Para poder tratar el *JSON* recibido, crearemos un nuevo objeto que encapsulará toda la lógica que trate con recepciones de información del servidor. Este objeto lo ubicaremos en todas las pantallas, puesto que la información podría llegar con retraso.

La información recibida, primeramente será decodificada y posteriormente procederemos a almacenarla en memoria, en una variable global, así como en un fichero donde almacenaremos los *tokens* (cadena de caracteres con contenido significativo) recuperados (los almacenaremos codificados). Este fichero servirá de *backup* (copia de seguridad) local, para poder mostrar algo si se cayera el servidor.

Tras finalizar una partida e introducir la información pertinente, codificaremos la puntuación a enviar y la almacenaremos en un fichero *.ini*, dónde a cada puntuación a enviar le asignaremos un identificador único que se utilizará como clave futura. Enviaremos mediante una petición *POST* dicho *token* al servidor. Al recibir la respuesta asíncrona, esta será capturada por el objeto encargado de capturar peticiones, se identificará la clave devuelta por el servidor y si coincide con alguna del fichero, el *token* se borrará de dicho fichero. En caso contrario, una vez haya pasado un tiempo considerable se re-intentará enviar dicha puntuación en determinadas ocasiones, como por ejemplo al comienzo del juego.

5.6. Seguridad de Maño Invaders

La seguridad en Maño Invaders se ha tomado muy enserio con el objetivo de evitar tramposos durante y después de RetroMañía 2016.

Así pues, todos los *token* que viajan hacia y desde el servidor van codificados cumpliendo con los pasos siguientes:

- Primero se realiza una codificación en *base 64*[2].
- Posteriormente se realiza un *cifrado XOR bit a bit*[3] con una palabra determinada elegida y consensuada entre el servidor y Maño Invaders.
- Por último, se realiza un nuevo cifrado en *base 64*.

Al introducir esta triple codificación se consigue que si un usuario tramposo llegara a identificar el *token* y viera fácilmente el cifrado base64, se encontraría ante algo que no tiene sentido (resultado del cifrado *XOR bit a bit*), y aun en el hipotético caso de saltar ese último cifrado, tendría por delante otro *base64*.

Durante Retro Mañía 2016, un usuario consiguió alcanzar la cifra de 54080 puntos, superando en ese momento al segundo clasificado en 40000 puntos. Esta cifra disparó todas las alarmas y sospechas, provocando que se endurecieran las medidas anti-tramposos.

En este punto, se decidió primero de todo contactar con el supuesto tramposo para averiguar si había conseguido saltarse la seguridad y en segundo lugar crear una nueva tabla de base de datos con el objetivo de recibir la siguiente información:

- ID de partida: identificador único de cada partida de un jugador.
- Puntuación inicial de partida.
- Vidas al inicio de la partida.

[2]https://www.base64decode.org/

[3]https://es.khanacademy.org/computing/computer-science/cryptography/ciphers/a/xor-bitwise-operation

– Tiempo al comienzo de la partida.

Para escribir en esta tabla creamos un método en el *API Rest* con el objetivo de recibir un *token* con dicha información. Por otro lado, enviaremos también el ID de partida al enviar la puntuación. Con esto, si una puntuación fuera sospechosa, miraríamos si tiene un ID de juego en la nueva tabla y si las vidas iniciales y la puntuación inicial son tres y cero.

Por último, como curiosidad el jugador que había levantado sospechas, se puso en contacto enviando un vídeo de juego en el que aparecía consiguiendo dicha puntuación. Lo cual, puede concluir en que la última medida de seguridad tomada fue preventiva ya que nadie logró saltarse la seguridad establecida.

5.7. Conclusiones de RetroMañía 2016

Las semanas previas al evento de RetroMañía fueron duras en cuanto a implementar en la medida de lo posible todo el *feedback* recibido, pero fueron las semanas de mayor avance del videojuego, dotándole al mismo de un aspecto más robusto y posibilitando que llegara al mayor número de personas.

Ejemplo de este alcance, fue que en la página de RetroMañía se alcanzó el número de 1126 lecturas[4].

El evento de RetroMañia 2016 supuso para Maño Invaders un escaparate donde darse a conocer y una repercusión inimaginable al principio, llegando incluso a aparecer en medios de comunicación como Aragón TV[5], en revistas especializadas de videojuegos como "HobbyConsolas"[6] y en el *podcast* del "I3A"[7] entre otros.

Gracias a la semana de RetroMañía el juego ha alcanzado una cota de mínimo (en el momento la elaboración de esta memoria) 539 (ver figura 5.4) partidas jugadas en total, 94 puntuaciones registradas *online* y 54 jugadores distintos compitiendo, siendo la plataforma más jugada el PC, seguido de la recreativa y por último *Android.*En los próximos días se espera revertir dicha estadística con la salida de Maño Invaders a la*Play Store* de *Google.*

[4]http://www.retroaccion.org/torneo-de-mano-invaders
[5]https://youtu.be/Zgjd1QsKHSM
[6]http://www.hobbyconsolas.com/noticias/retromania-2016-celebrara-7-11-noviembre-72694
[7]http://www.ivoox.com/retromania-audios-mp3_rf_13595927_1.html

(1 of 6) 1 2 3 4 5 6 10

Nick	Puntuación	Partidas	Fecha del high score	Fecha última partida	Plataforma high score	Fase alcanzada
Metr81	54080	10	2016-11-08 12:28:38.0	2016-11-09 12:23:12.0	PC	0
Bob	25365	7	2016-11-10 23:40:59.0	2016-11-10 23:40:59.0	PC	32
pipocesar	20610	13	2016-11-10 12:26:37.0	2016-11-10 12:26:37.0	PC	25
Edgar	16615	20	2016-11-10 23:18:13.0	2016-11-10 23:56:44.0	PC	21
FARYGABI	16105	1	2016-11-09 20:28:44.0	2016-11-09 20:28:44.0	RECREATIVA	20
ALIJARDE	14765	16	2016-11-07 12:50:22.0	2016-11-07 15:41:32.0	RECREATIVA	0
NashFire	13310	6	2016-11-06 01:16:33.0	2016-11-06 01:16:33.0	PC	0
eddiethewild	12805	10	2016-11-23 19:49:55.0	2016-11-23 19:49:55.0	PC	17
Garci	10645	9	2016-11-06 19:34:08.0	2016-11-10 00:01:47.0	PC	0
Moned	9165	31	2016-11-10 22:18:22.0	2016-11-14 22:13:44.0	PC	12

Figura 5.4: Top 10 de máximas puntuaciones de Maño Invaders

Capítulo 6

Conclusiones

6.1. Cronograma

Cuando comencé a desarrollar Maño Invaders a final de 2013 desconocía la tecnología *GameMaker* y por ello en repetidas ocasiones acabe aprendiendo mediante el ensayo-error. Ejemplo de ello fue el movimiento de los invasores, que tuve que rehacerlo hasta en tres ocasiones.

Al no saber exactamente muy bien como empezar en la plataforma, tampoco había organizado los objetos como debía, no había preparado la interacción entre los mismos, ni cómo iba a ser implementar las transiciones entre fases, no había pensado en objetos controladores, etc. Todo esto me hizo perder mucho tiempo ya que fueron necesarias varias refactorizaciones que mejoraran todo.

A mitad de desarrollo llegue a un punto en el que apareció una nueva versión de *GameMaker* más potente y que ofrecía nuevas mejoras, por lo tanto fue necesario actualizar todo el proyecto a la nueva versión, lo que supuso prácticamente rehacer de nuevo varias lógicas desde cero. Para la implementación del servidor también fue necesario el aprendizaje de ciertas tecnologías utilizadas, que llevó un tiempo nada desdeñable ya que era la primera vez que implementaba un servidor totalmente desde cero.

Tampoco desdeñable ha sido el tiempo requerido a aprender *Latex*, lenguaje utilizado con el objetivo de dotar la presente memoria de una mayor adecuación a un trabajo propio de investigación.

Por todo lo anterior, el tiempo del proyecto ha sido aproximadamente de 450 horas dedicadas (ver figura 6.1).

A pesar de todas las horas empleadas en su realización, la principal causa de que se haya alargado tanto tiempo se ha debido a que desde prácticamente el comienzo de la realización del proyecto solo he podido ir avanzando en mis ratos libres compaginándolo con el trabajo, cuestión que ha supuesto un esfuerzo extra, pero que también ha

aportado otro tipo de visión en su realización.

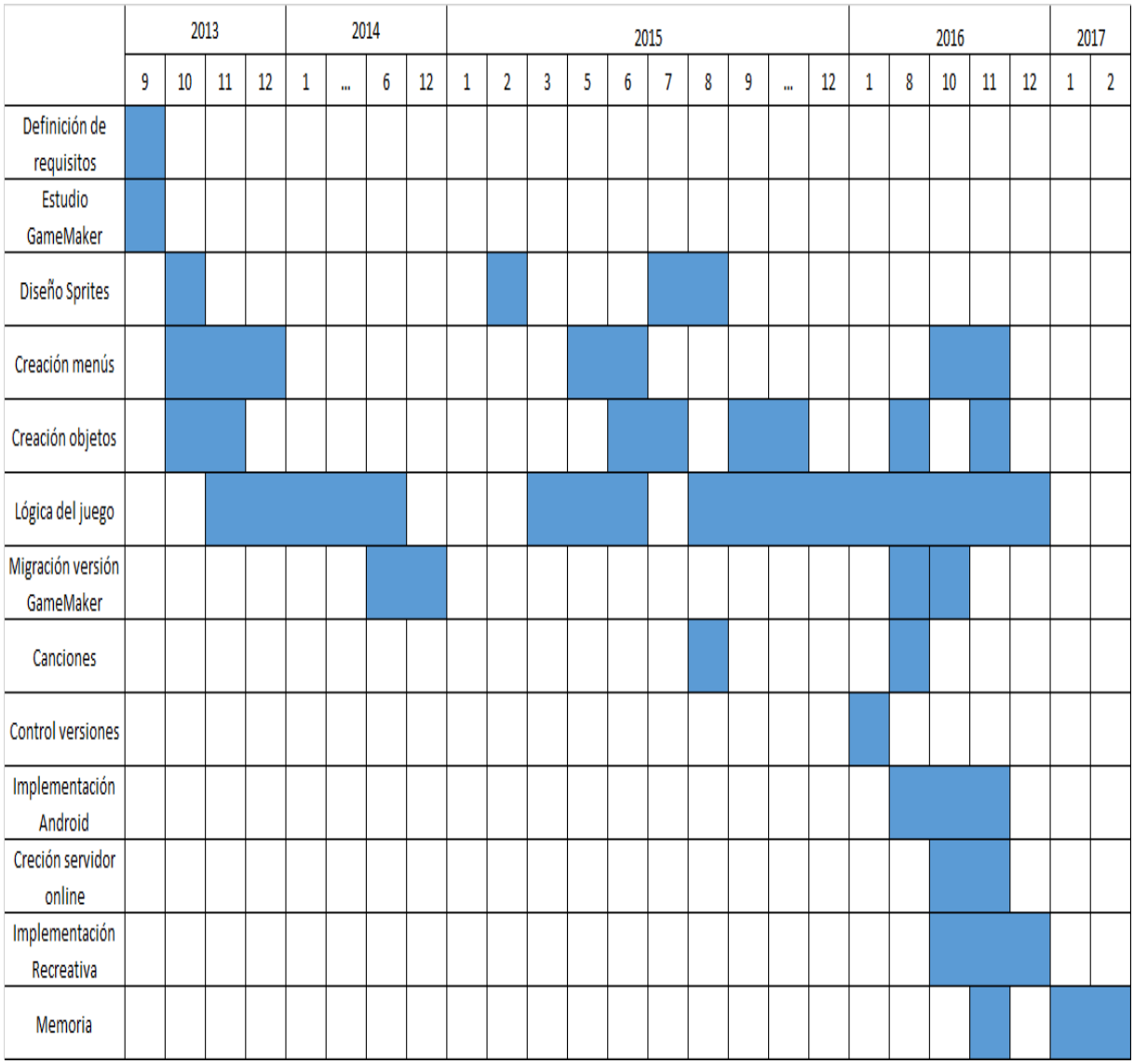

Figura 6.1: Constantes del juego

6.2. Posibles ampliaciones

– Juego multi-jugador:
  La primera de las posibles ampliaciones sería dar cabida a introducir partidas multi-jugador, que cabría implementarlo de dos modos: o bien compitiendo y jugando cada uno por separado o bien los dos en la misma pantalla, es decir, dos naves del Pilar compitiendo para ver quien consigue más puntos o bien colaborar entre ambos para superar el máximo número de fases posibles. Con este nuevo modo incluso se podría añadir una nueva tabla de histórico de puntuaciones, en partidas colaborativas.

– Mayor número de fases:
  Otra de las posibles ampliaciones, sería precisamente una ampliación de las fases del juego, añadiendo nueva temática aragonesa, ya sean monumentos nuevos que se descartaron en la fase de diseño como nuevos alimentos, nuevas ciudades como Huesca y Teruel, nuevos jefes finales, etc.
– Mayor dificultad de juego:
  También se podría aumentar un poco la dificultad añadiendo una nueva fila de enemigos una vez alcanzáramos cierta fase.
– Nuevos menús:
  Otra ampliación supondría la cabida a un nuevo menú donde un jugador pudiera buscar las diez mejores puntuaciones de un amigo.
– Nuevas plataformas de juego:
  Por último otra de las ampliaciones sería aprovechando el entorno de *GameMaker* y sus posibilidades, llevar el videojuego a nuevas *stores* como la de *Play Station*, *Iphone*, *Windows*...

6.3. Valoración personal

Al comienzo de la realización del proyecto mi principal objetivo personal era sumergirme en el mundo de los videojuegos, un sueño que desde pequeño siempre había tenido, y para ello estaba claro que la mejor manera de conseguirlo era realizando un videojuego desde cero.

Junto al objetivo de crear un juego, otro objetivo que tenía era conocer los distintos entornos de desarrollo que había para realizarlos, siendo esta otra de las ideas principales en el proyecto, estudiar y comparar los mismos. En un principio se pensó en crear Maño Invaders en distintos entornos con el objetivo de que el proyecto girara en torno a qué ventajas y desventajas ofrecen cada uno de ellos, realizando un estudio profundo de los mismos y sirviendo como base del estudio la realización de Maño Invaders. El problema surgió cuando Maño Invaders empezó a tener tanta complejidad y envergadura, que su realización desde cero, ya era un estudio por si mismo, de ahí se decidió finalmente que el proyecto girara únicamente a todo el proceso de realizar Maño Invaders.

Para hacer Maño Invaders desde cero ha sido necesario pasar primero por toda tormenta de ideas que puede tener uno en todo comienzo, discutir dichas ideas con mi director, concretar dichas ideas y plasmarlas en la fase de análisis y diseño. Posteriormente, intentar implementar dichas ideas y ver que quizá eso que se había pensado no era tan sencillo como se pensaba en un principio. Recibir *feedback* de

personas que van probando el trabajo realizado. Sentir la presión de cumplir con unos objetivos de fechas marcadas con RetroMañía. Todo esto, es lo que significa un proyecto de videojuegos en el mundo real.

Desde fuera, videojuegos con inspiración clásica como Maño Invaders, tal vez puedan parecer sencillos de realizar, pero no lo son, más aun si en su realización se añaden nuevos elementos como ha sido el caso en Maño Invaders. Pero sobre todo, el hecho más importante y nada fácil que todo juego debe perseguir es, que el juego realizado sea divertido. Es decir que un usuario juegue una partida y quiera volver a jugar, que se quede a las puertas de una fase y quiera ver qué hay después, que vea la puntuación de un amigo y se anime a superarle. Personalmente, he podido observar que lo anterior, se ha conseguido con Maño Invaders, viendo a amigos, familiares y gente en RetroMañía jugando a mi videojuego.

Otro de los objetivos conseguidos es el plasmar y servir de homenaje a la idea que en su día tuvieron los integrantes del grupo "RetroAcción" con el cartel realizado en su primera edición de RetroMañía.

Uno de los puntos que me gustaría destacar y que siempre recordaré es el contactar con grupos de música aragoneses, que desinteresadamente accedieron a colaborar con Maño Invaders, dotándolo de un repertorio musical completo y envidiable.

En el tintero ha quedado intentar llegar a conseguir aun mayor repercusión del videojuego, pero este tampoco era uno de los objetivos primordiales que se perseguía, si no aprender, y que el aprendizaje sirviera de guía futura. No obstante, con la salida a la *Play Store de Google* se esperá revertir esto.

Sobre el uso de *GameMaker*, puedo concluir que es una plataforma muy potente por todas las posibilidades que ofrece, quizá esta potencia implica que si se quiere explotar bien la herramienta, esta requiera de conocimientos de programación. Un usuario sin nociones de programación podría llegar a utilizar *GameMaker* y crear un videojuego, pero a nada que el videojuego creciera en complejidad se encontraría ante situaciones en las que la acción que desea para su juego no tiene un botón para arrastrarla al panel de acciones. O bien, le podría ocurrir que realizando condiciones complejas mediante bloques visuales, podría en el peor de los casos llevarle a perderse realizándolas, mientras que con un bloque de código lo habría resuelto fácilmente. Por otro lado, un aspecto muy positivo de *GameMaker* es la posibilidad de iniciarte en el mismo de manera gratuita, utilizando su versión específica para ello.

Por último, con la implementación de Maño Invaders, he podido comprobar que es posible iniciarse en el mundo de los videojuegos disponiendo de las herramientas adecuadas. En mi caso, desconocedor inicialmente de *GameMaker* y sin una guía clara, he pasado por muchos puntos al comienzo en los que implementaba ciertas lógicas

y comportamientos en *GameMaker* que luego no me servían para lo que finalmente pretendía hacer; en otras ocasiones el acercamiento realizado quizá no era el adecuado; cuestiones que hasta que no se sumerge uno en la implementación son imposibles de percibir, pero que tomando como base un trabajo detallado como el realizado en este proyecto se podrían evitar.

Capítulo 7

Bibliografía

[1] Sudheer Jonna Andy Bailey. *PrimeFaces Theme Development.* Packt Publishing Ltd, 2015.

[2] Benjamin G. Tyers. *Gamemaker: Studio Book - 100 Mini Games.* CreateSpace Independent Publishing Platform, 2016.

[3] Brandon Gardiner. *GameMaker Cookbook.* PACKT Publishing, 2015.

[4] Craig Walls. *Spring.* Anaya Multimedia/Manning, 2015.

[5] Eduardo Pérez Martínez. *Hibernate. Persistencia De Objetos En JEE.* Ra Ma, 2015.

[6] Francesco Marchioni. *WildFly 8 Administration Guide.* itBuzzPress.com, 2014.

[7] Jr. Jerry Lee Ford. *Getting Started with Game Maker.* Cengage Learning, 2010.

[8] Kent Ka Iok Tong. *Beginning JSF 2 APIs and JBoss Seam.* Apress, 2009.

[9] Santosh Kumar K. *Spring And Hibernate.* Tata McGraw-Hill Publishing, 2009.

Anexos

Anexos A

Imágenes del Juego

En el presente anexo ilustraremos imágenes capturadas durante el transcurso de partidas jugadas en las distintas plataformas de juego, con el objetivo de ser representativas de las mismas.

A.1. Menús y Fases

A.1.1. Menús

Cada una de las imágenes de esta sección representa un menú de juego distinto. Se han tenido en cuenta menús de todas las plataformas.

Figura A.1: Menú principal en PC y Android

Figura A.2: Menú principal en Recreativa

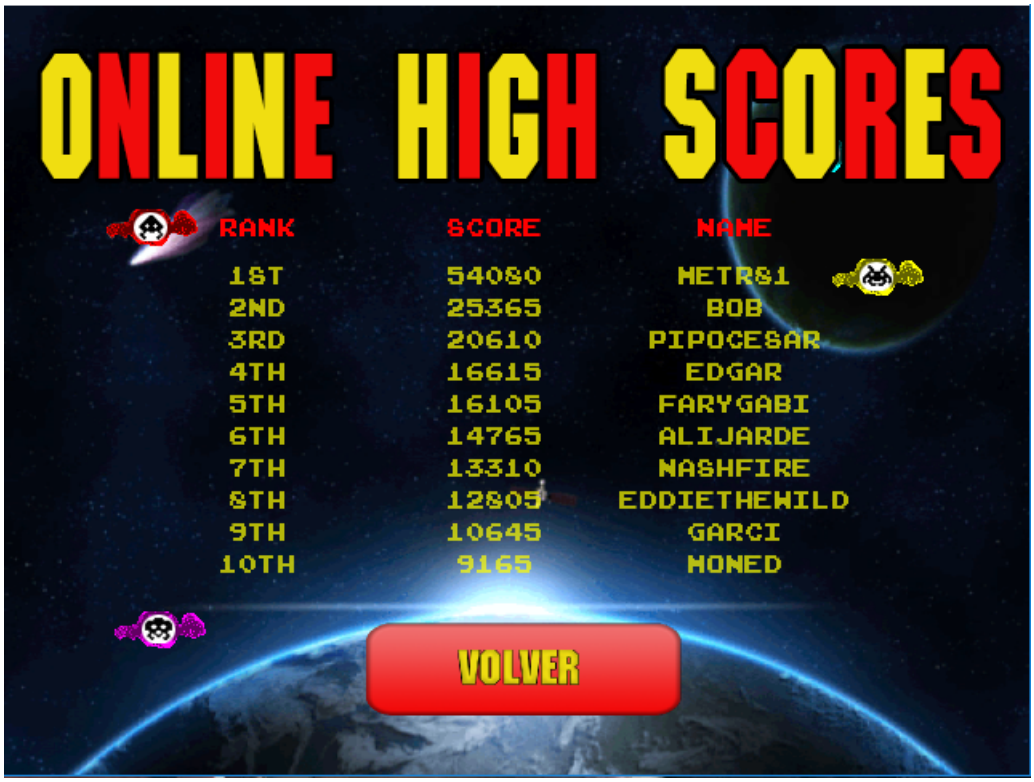

Figura A.3: Menú de puntuaciones.

Figura A.4: Menú de créditos.

Figura A.5: Menú de instrucciones.

Figura A.6: Menú de configuración (PC).

Figura A.7: Menú de configuración (Android).

Figura A.8: Fin del juego en recreativa.

A.1.2. Fases de Zaragoza

En la presente subsección se ilustrarán todas las fases pertenecientes a la temática de Zaragoza.

Figura A.9: Fase 1 Zaragoza: Puentes de la ciudad.

Figura A.10: Fase 1 Zaragoza en Android: Puentes de la ciudad.

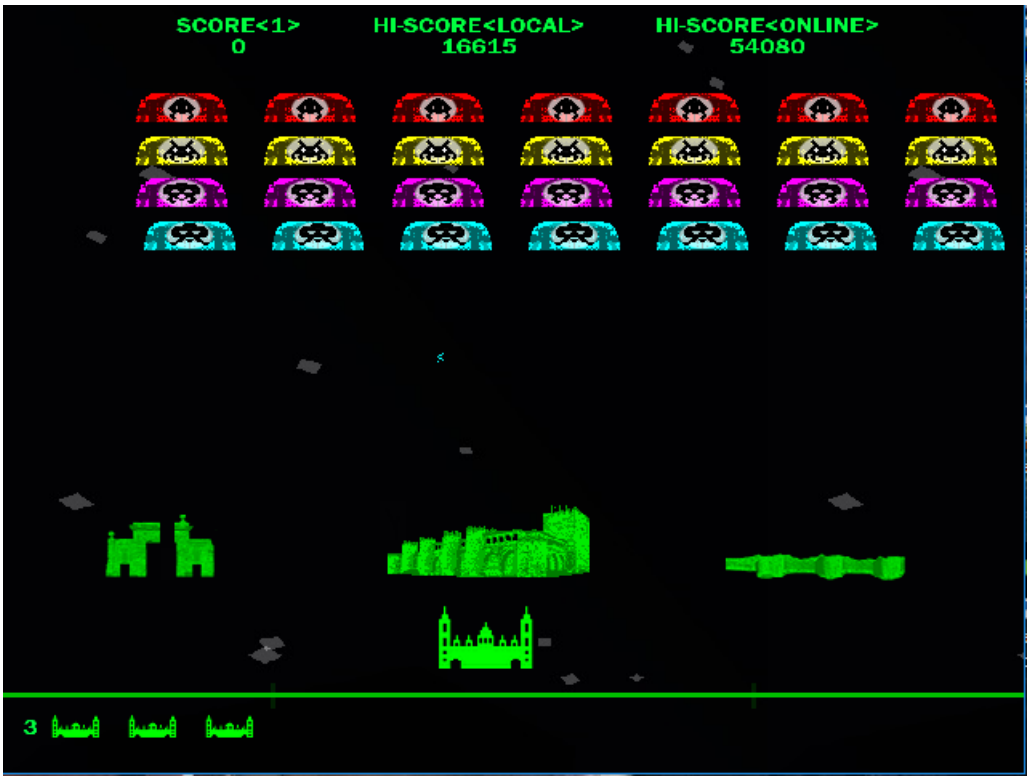

Figura A.11: Fase 2 Zaragoza: Monumentos de la ciudad.

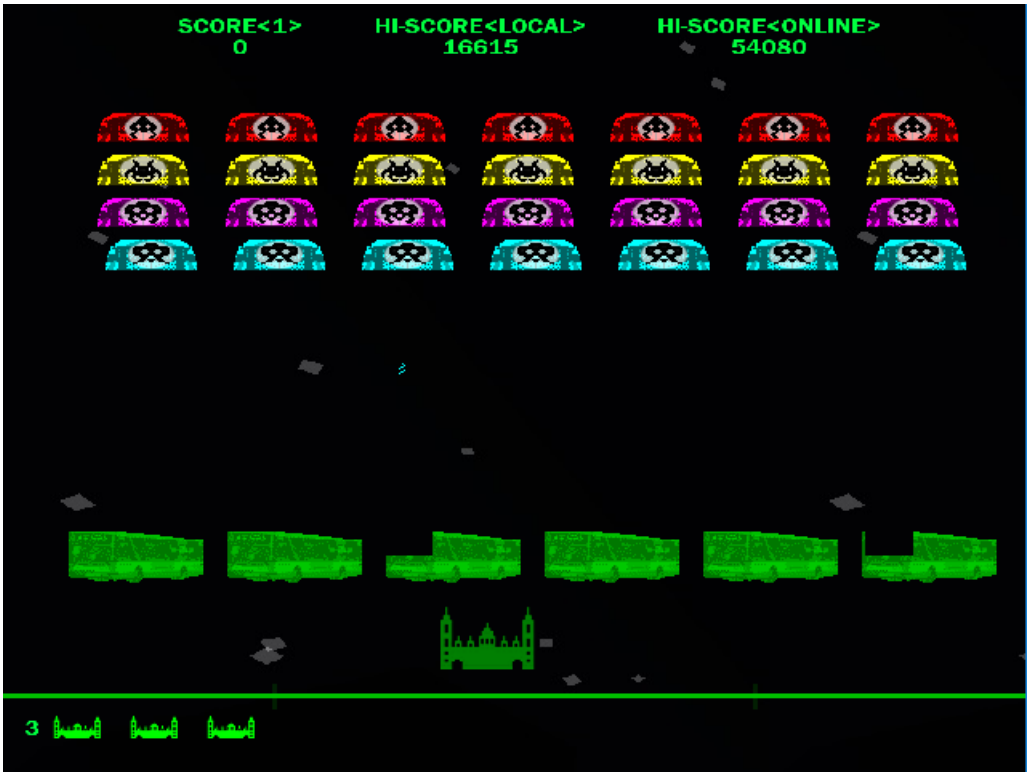

Figura A.12: Fase 3 Zaragoza: Autobuses urbanos.

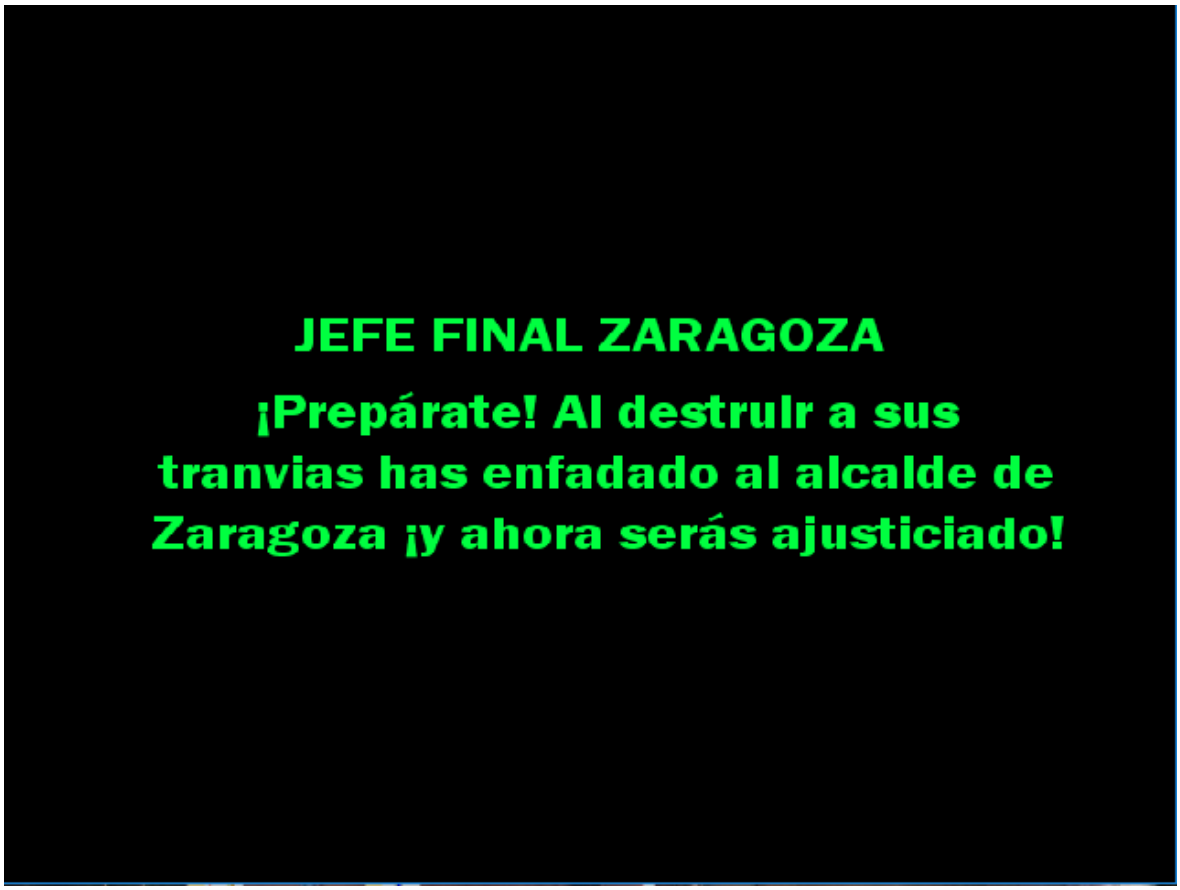

Figura A.13: Ejemplo de comienzo previo a una fase.

Figura A.14: Alcalde: Jefe final de Zaragoza.

A.1.3. Fases de la Expo

En la presente subsección se ilustrarán todas las fases pertenecientes a la temática de la Expo Zaragoza 2008.

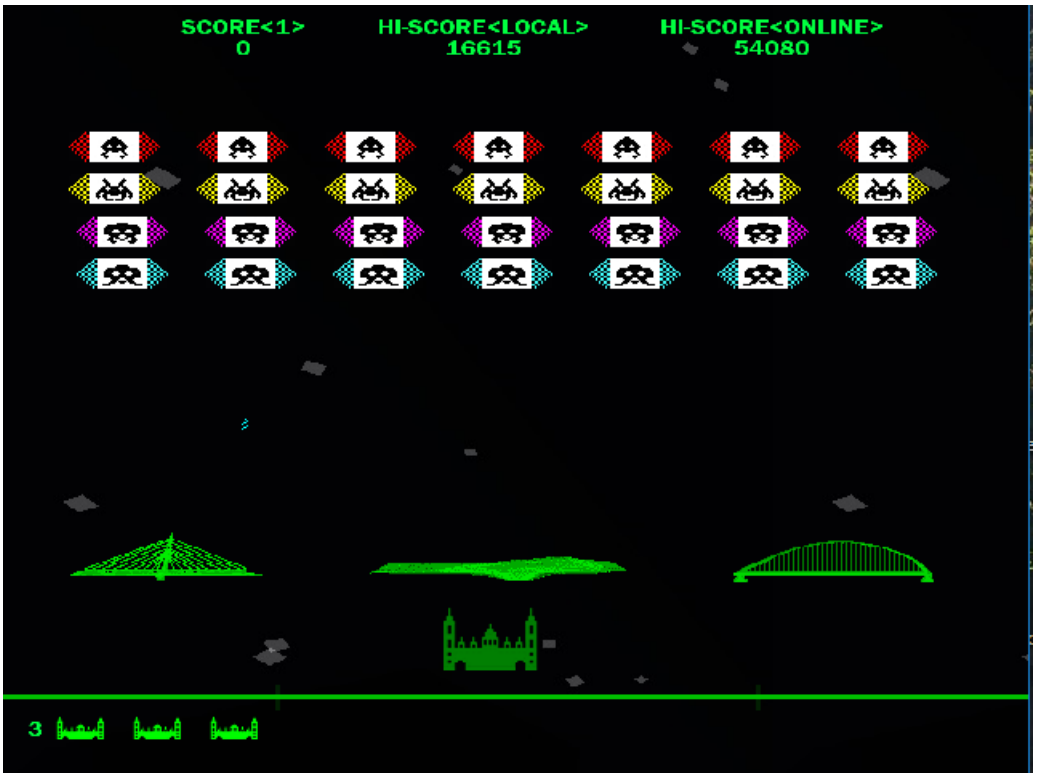

Figura A.15: Fase 1 Expo: puentes de la Expo.

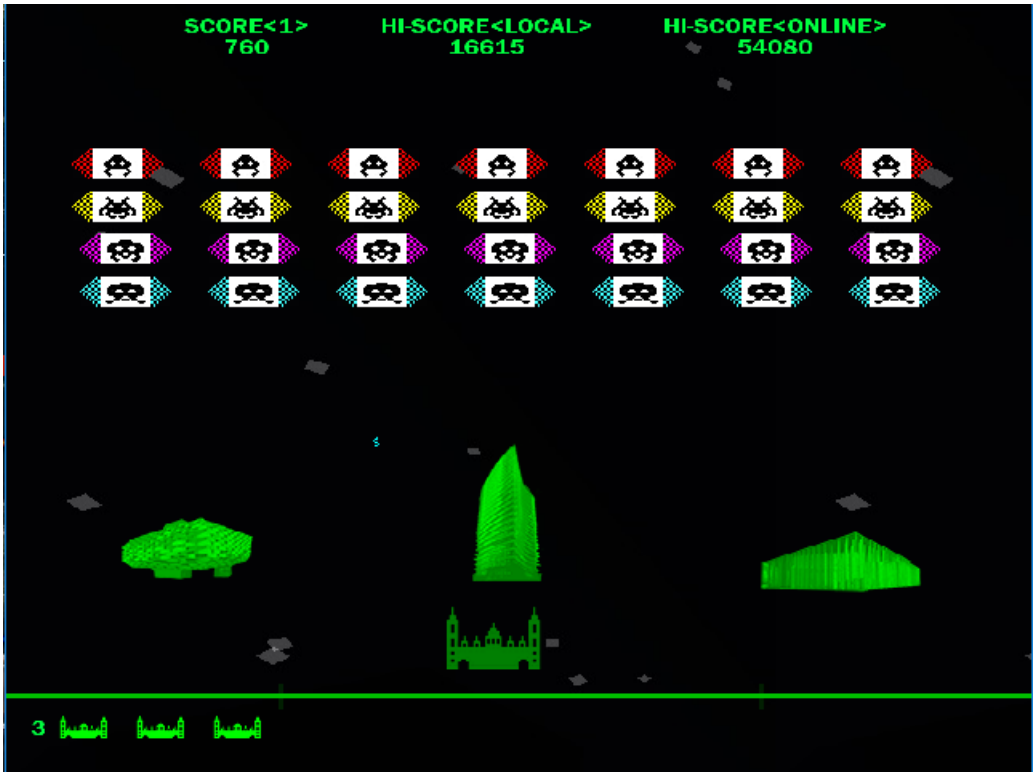

Figura A.16: Fase 2 Expo: edificios de la Expo.

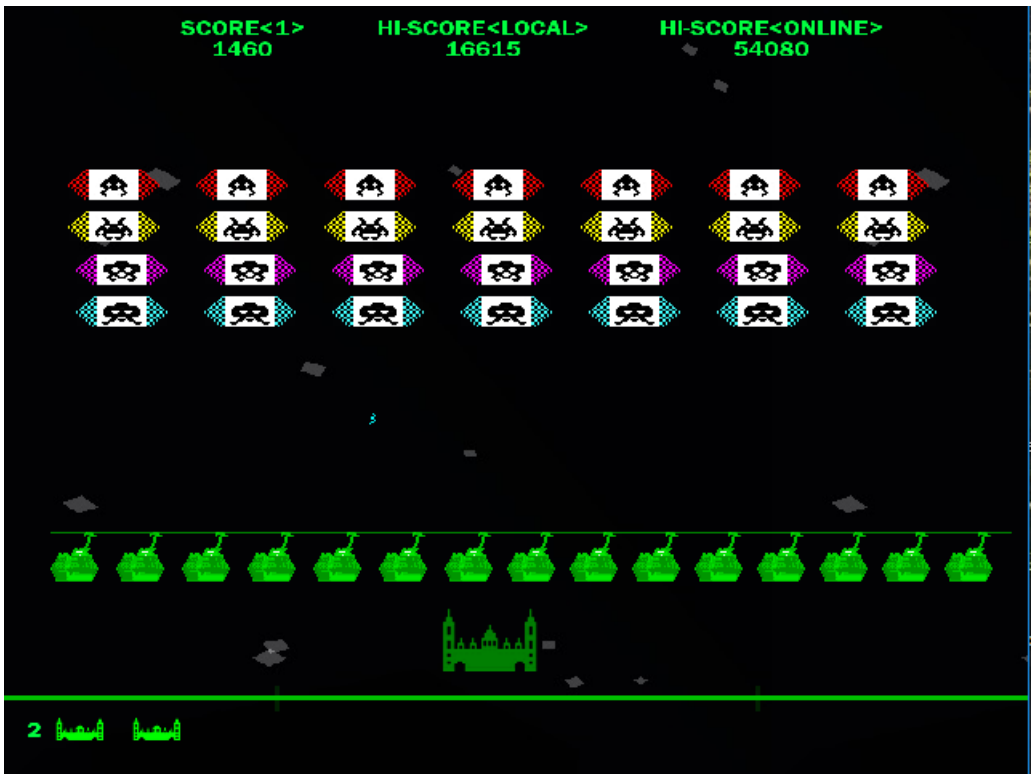

Figura A.17: Fase 3 Expo : teleféricos.

Figura A.18: Fluvi: Jefe final de las fases Expo.

A.1.4. Fases de Aragón

En la presente subsección se ilustrarán todas las fases pertenecientes a la temática de Aragón.

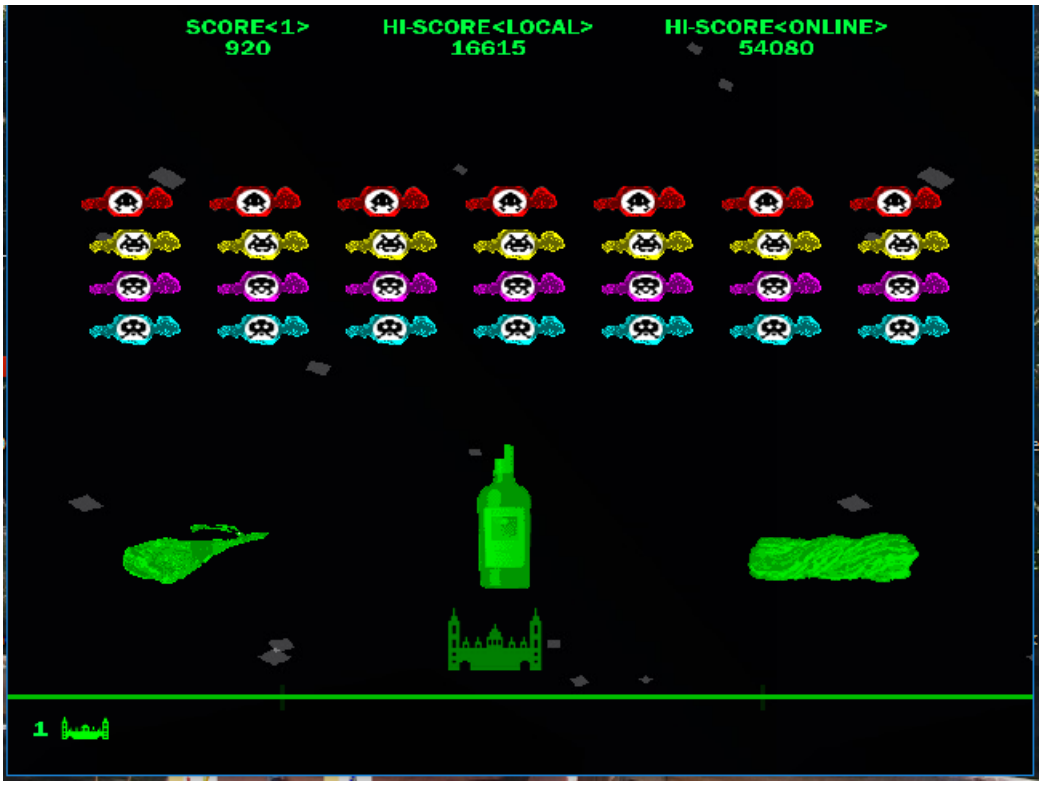

Figura A.19: Fase 1 Aragón: alimentos regionales.

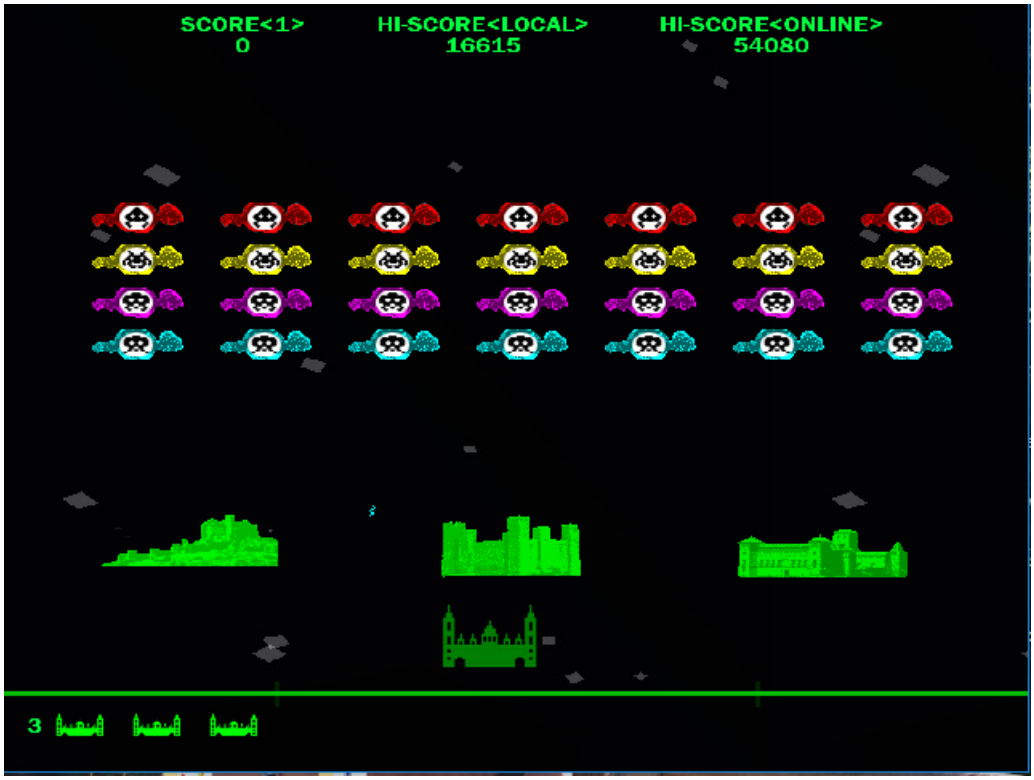

Figura A.20: Fase 2 Aragón: Castillos de la región.

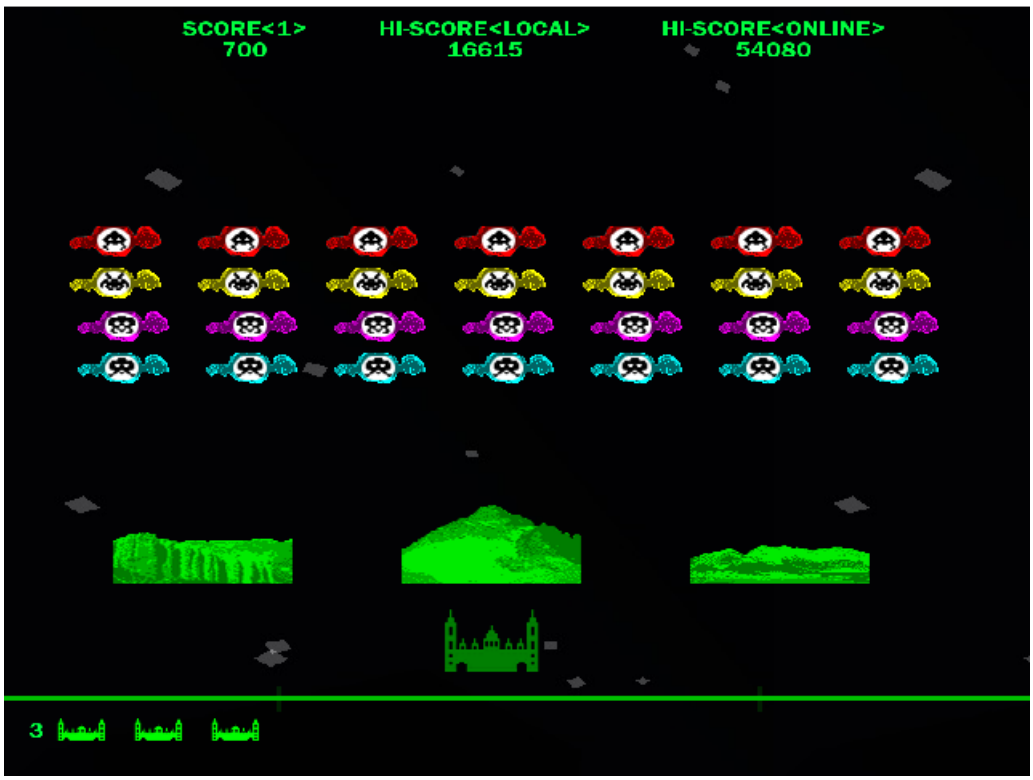

Figura A.21: Fase 3 Aragón: Montañas y picos regionales.

Figura A.22: SuperMaño: Jefe final de las fases de Aragón.

A.2. Barreras

A continuación se ilustrarán todas las barreras que hacen presencia en el videojuego y que servirán como defensas del jugador.

A.2.1. Barreas de las fases de Zaragoza

Figura A.23: Barreras fase 1 Zaragoza: originales del cartel de Retro Mañía. Puente de Santiago, Puente de Piedra y Puente de Hierro.

Figura A.24: Barreras fase 2 Zaragoza: Puerta del Carmen, Aljaferia y Murallas Romanas

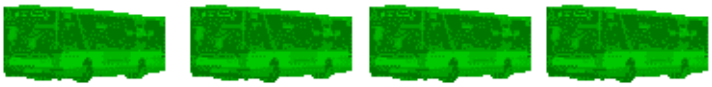

Figura A.25: Barreras fase 3 Zaragoza: Autobús urbano de Zaragoza

A.2.2. Barreas de las fases de la Expo.

Figura A.26: Barreras fase 1 Expo: Pasarela puente, Pabellón Puente y Puente del tercer milenio

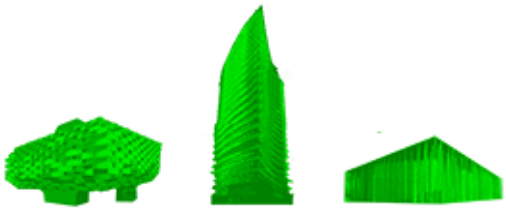

Figura A.27: Barreras fase 2 Expo: Pabellón de Aragón, Torre del agua, y Pabellón de España

Figura A.28: Barreras fase 3 Expo: Teleféricos de Expo Zaragoza

A.2.3. Barreas de las fases de Aragón

Figura A.29: Barreras fase 1 Aragón: Jamón de Teruel, Vino cariñena y Trenza de Almudevar

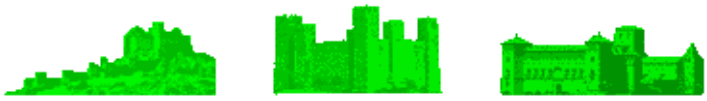

Figura A.30: Barreras fase 2 Aragón: Castillo de Loarre, Castillo de Sadaba y Castillo de Alcañiz

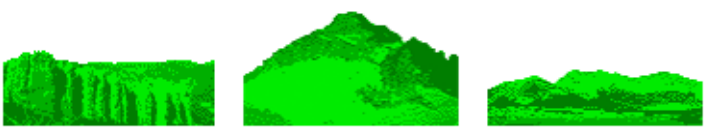

Figura A.31: Barreras fase 3 Aragón: Sierra de Gúdar, Aneto y El Moncayo

A.3. Jefes Finales

En la presente sección se ilustrarán todos los estados por los que podrá pasar un jefe final, siendo los estados rojizos, los producidos al recibir un disparo. De izquierda a derecha son: estado parado, moviéndose hacía la izquierda, moviéndose a la derecha y saltos de ráfaga.

Figura A.32: Posibles imágenes del alcalde y sus armas.

Figura A.33: Imágenes de Fluvi y sus armas.

Figura A.34: Imágenes de SuperMaño y sus armas.

Anexos B

Guía de uso de GameMaker

Este apartado tiene como finalidad introducir al usuario en el uso de *GameMaker*. Así pues, se explicarán en el mismo aquellos aspectos más importantes del entorno, junto a sus herramientas más útiles.

B.1. Gestión de *Sprites*

Un *sprite* en *GameMaker* no es más que una o varias imágenes contenidas en un mismo elemento. En la pantalla de edición de *sprites*, *GameMaker* nos permitirá definir el eje de coordenadas de dicha imagen/es (ver figura B.1), así como si va a trabajar con ella utilizando el *precise collision checking*, que no es más que un algoritmo que utiliza *GameMaker* para comprobar todos los pixeles de una imagen y ver si está ha sido colisionada o no. Destacar, que si añadimos más de una imagen a un *sprite*, este *sprite* al ser asociado a un objeto se comportaría como una imagen *GIF*[1], ya que iría iterando entre las distintas imágenes que lo componen (salvo que en el objeto expresáramos que no deseamos tal comportamiento).

[1] https://es.wikipedia.org/wiki/Graphics_Interchange_Format

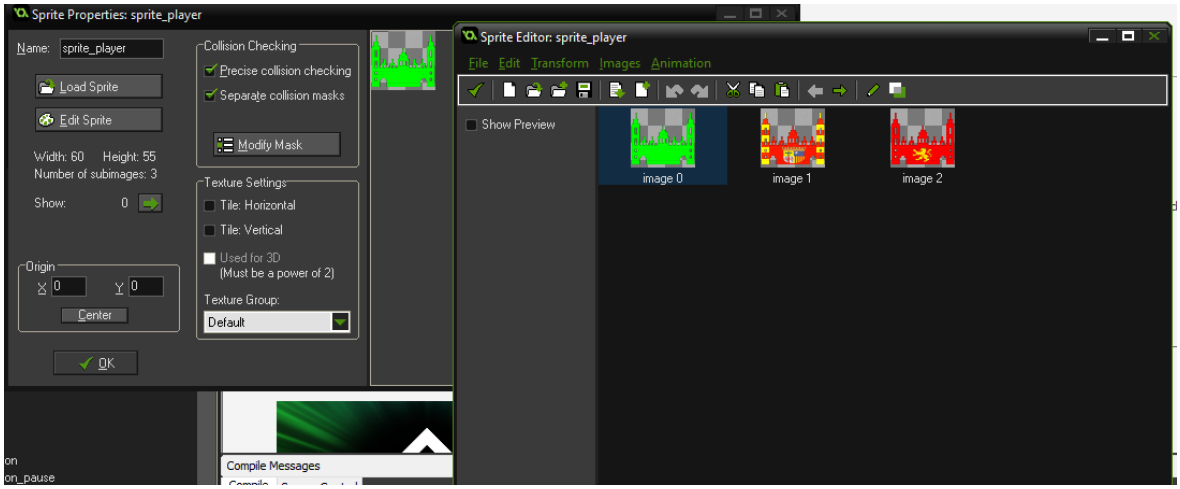

Figura B.1: Ejemplo de *sprite* en *GameMaker*

B.2. Gestión de sonidos

GameMaker nos permite añadir música y sonidos fácilmente. Simplemente deberemos crear un objeto sonido y añadirle un fichero en formato .mp3, .ogg o .wav. Una vez elegido el fichero, nos dará opciones para cambiar el *bitrate*[2] y el *Sample Rate*[3]. Tambien nos dará opción de elegir si deseamos escuchar el sonido en "mono" o "estéreo" y si el sónido es de 8 o 16 bits.

Por último nos dará tres opciones:

- Sin compresión y sin transmisión: en memoria y menos recursos de CPU.
- Comprimido y sin transmisión: en memoria y más recursos de CPU.
- Comprimido (no en carga) y sin transmisión: más memoria y poco de CPU.
- Comprimido y con transmisión: en disco, alto uso de CPU

Así pues, una vez tengamos añadida nuestra música, podremos ejecutarla mediante funciones de *GameMaker* o bien con una acción determinada.

[2]https://en.wikipedia.org/wiki/Bit_rate
[3]https://es.wikipedia.org/wiki/Frecuencia_de_muestreo

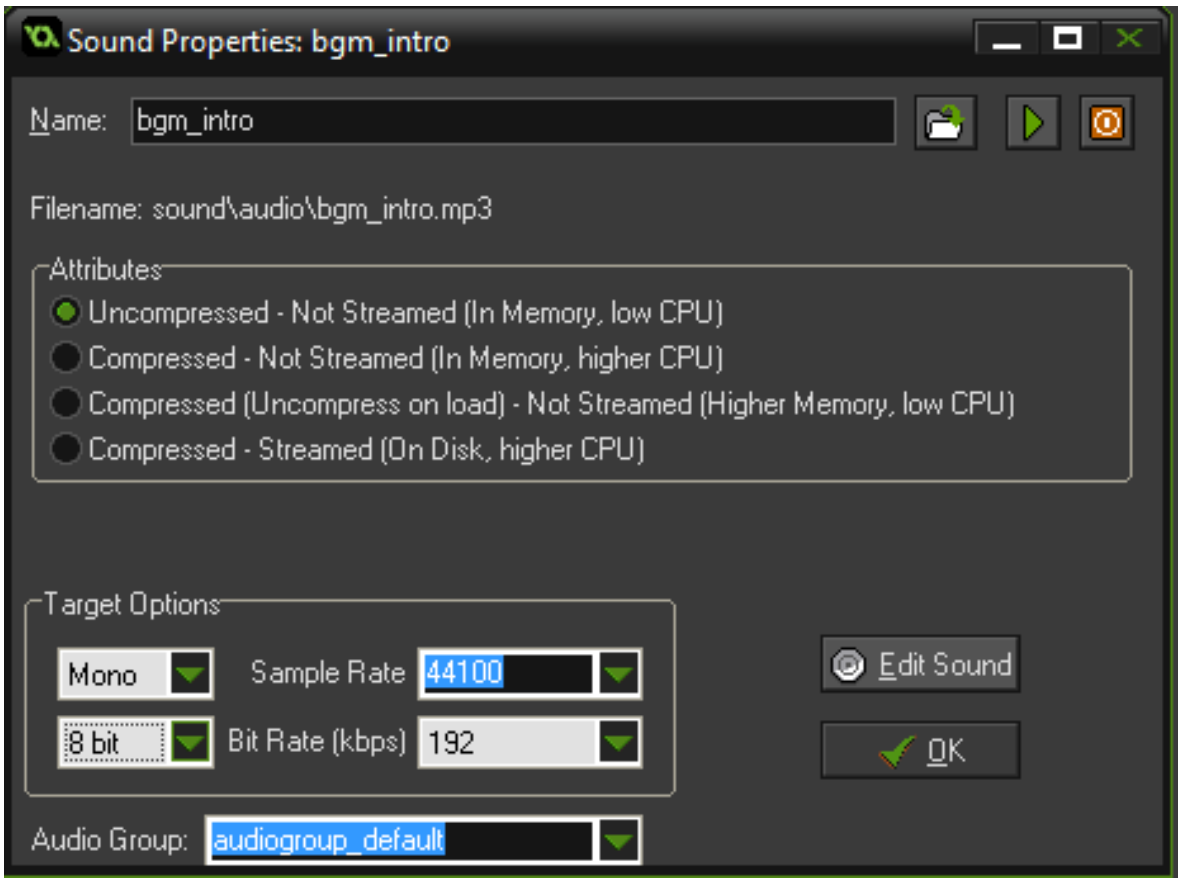

Figura B.2: Inserción de sonidos/músicas en *GameMaker*.

B.3. Gestión de objetos

Cualquier objeto de *GameMaker* nos permite a su vez definirle eventos (ver figura B.3) en los que ejecutará distintas acciones. Algunos eventos importantes son los siguientes:

- Evento *Create*: se ejecutará en el momento de la creación. Útil sobre todo para inicializar y definir atributos.
- Eventos *Alarm* : se ejecutarán previo haber sido programados. Actúan a modo alarma, de tal modo que primeramente será necesario programarlos para que ocurran en X *ticks* de reloj. Transcurridos esos *ticks* se ejecutarán
- Evento *Step*: este evento se ejecutará siempre en cada *tick* de reloj, por lo tanto no conviene abusar del mismo o introducir una lógica pesada en el mismo.
- Evento *Draw*: es otro evento que se ejecuta en cada *tick* de reloj. Es el encargado de dibujar el *sprite* del objeto o bien de pintar cualquier cosa.
- Evento *Collision*: son aquellos producidos por una colisión con otro objeto. *GameMaker* nos permite definir en los *sprites* la precisión con la que detectará colisiones. Al producirse una saltaría este evento, por ello siempre que añadamos un evento de este tipo será necesario indicar el objeto de colisión.

– Evento *Mouse*: nos permitirá elegir entre bastantes opciones de interrupción acerca del ratón, como por ejemplo, al producirse *click* con botón izquierdo, presionar el botón derecho, mover la rueda central, etc.

– Evento *Key*: este evento nos permitirá tratar con interrupciones provocadas por el teclado.

– Evento *Other*: encapsula al resto de eventos, como puedan ser colisión con los límites de la pantalla, eventos definidos por el usuario, comienzo del juego, fin de las vidas...

– Evento *Asynchronus*: es un evento que se producirá siempre que recibamos una respuesta asíncrona, como podría ser la respuesta ante una petición previa *HTTP*.

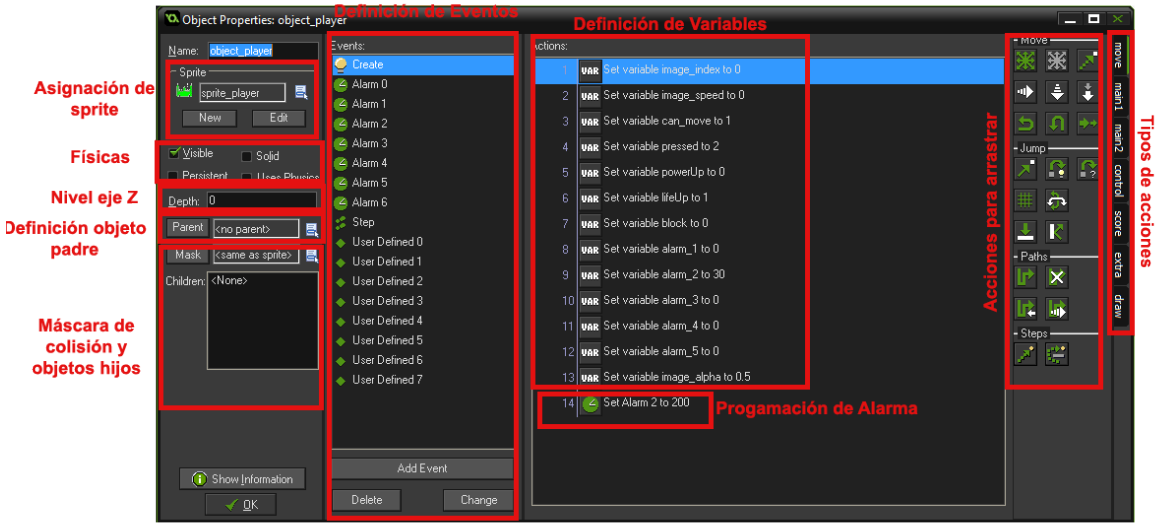

Figura B.3: Eventos posibles para un objeto

Dentro de cada evento será posible arrastrar acciones del panel de acciones de *GameMaker*, como pudieran ser, definir o modificar de valor una variable, desplazar al objeto, cambiar de imagen asociada, mover el juego a otra *room*, etc. La más importante de todas será la acción de *Execute a piece of code* (ejecutar un trozo de código), en la cual definiremos el código en GML como en cualquier otro lenguaje de programación.

A todo objeto a su vez, le podremos asignar un objeto padre, del que heredará la lógica. De este modo, podemos crear una jerarquía entre los objetos.

Por último, será posible definir un *sprite* asociado a un objeto, simplemente seleccionándolo en el apartado *sprite* en la pantalla de edición de un objeto.

B.4. Gestión de *rooms*

GameMaker nos permite trabajar con lo que él mismo denomina *room* (ver figura B.4), que no son más que pantallas entre las que el juego va iterando.

Figura B.4: Ejemplo de creación de una *room* en *GameMaker*

Dentro de una *room* tendremos la posibilidad definir tanto el *background*(fondo de la misma), *objects* (objetos que se inicializarán al comenzar la misma), su tamaño, anchura, y su velocidad, que se representará en *ticks* de reloj. Como ejemplo, si definimos un *speed* de 30, nuestro juego se ejecutará con 30 *ticks* por segundo. Esto es importante, ya que guarda relación directa con los *fps* (imágenes/frames por segundo).

Sobre una *room* podremos arrastrar y soltar objetos, que en caso de tener asociado un *sprite* se pintarán en la posición en la cual los soltemos y su inicialización tendrá lugar al comienzo de la misma. Por otro lado, podemos delegar la creación de objetos a otros, sin problema alguno, pero obligatoriamente si queremos añadir algo de lógica a nuestra *room* debe existir al menos un objeto.

Por linea general en cada *room* nos serviremos de objetos para definir cierta lógica o representar ciertos elementos del juego como pudieran ser las naves invasoras, nave del jugador, etc. Estos objetos podremos o bien inicializarlos en tiempo de comienzo de una *room*, simplemente arrastrándolos a la *room* en la pantalla de edición de la *room*, o bien inicializándolos desde otro objeto.

B.5. Gestión de fuentes

Con objeto de poder escribir un texto en *GameMaker* con un tipo de letra y estilo concreto será necesario que previamente creemos lo que *GameMaker* denomina como

Fonts (ver figura B.5). En estos *fonts* simplemente especificaremos, su nombre, su tipo de letra, la calidad del mismo, si es negrita o itálica y el tamaño de la letra.

Figura B.5: Ejemplo de creación de un *font* en *GameMaker*

B.6. Gestión de *scripts*

Los *Scripts* de *GameMaker* son porciones de código susceptibles de ser utilizados desde cualquier objeto de la aplicación. Vendrían a cubrir la funcionalidad de los métodos estáticos de otros lenguajes de programación.

B.7. Gestión de filtros

GameMaker ofrece la posibilidad de aplicar filtros en tiempo real sobre la pantalla (ver figura B.6). El objeto de estos filtros es modificar los pixeles de la pantalla con objeto de producir un efecto visual concreto. El principal problema del uso de *shaders* es que en algunos casos ha dado problemas en equipos con versiones de *DirectX*[4] inferiores a la versión 11 del mismo. Los filtros los introduciremos en lo que *GameMaker* denomina como *Shaders*. Estos *shaders*, tienen dos paneles: uno denominado *Vertex*, que define los parámetros que le pasaremos al *shader*, y otro denominado *Fragment*, que es donde introduciremos el código que ejecutará el *shader*.

[4]http://www.directx.com.es/

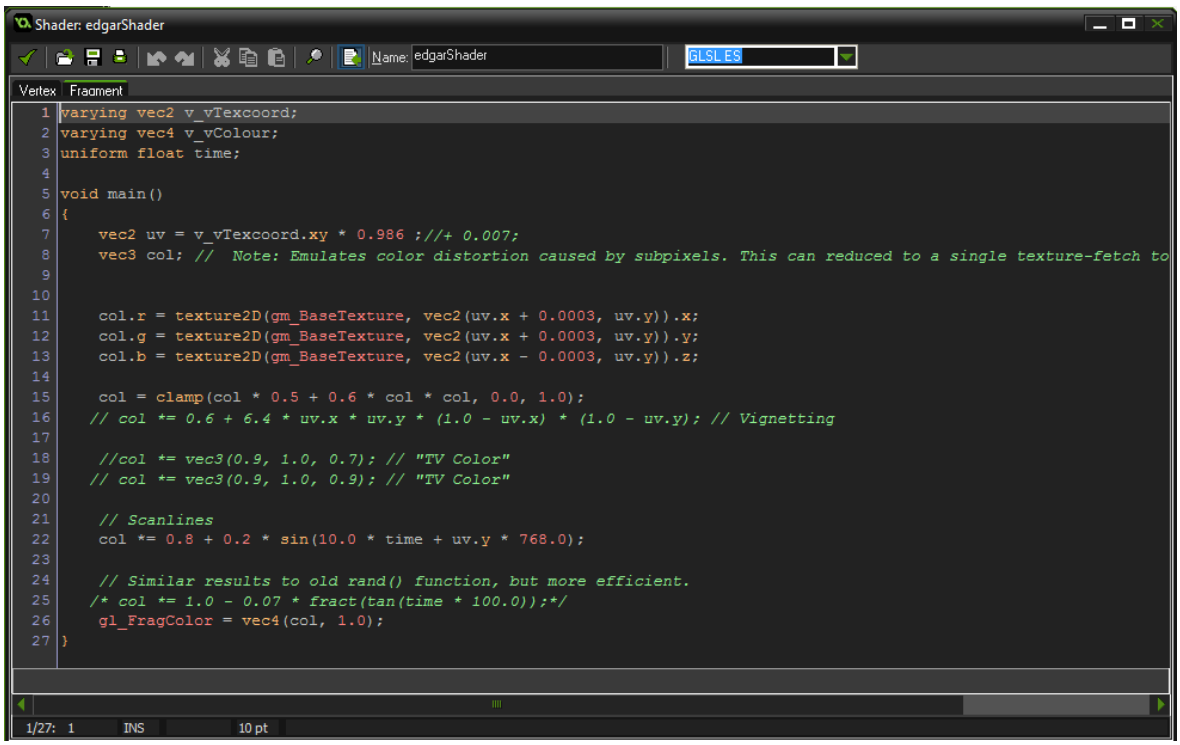

Figura B.6: Ejemplo de creación de un *shader* en *GameMaker*.

B.8. Configuraciones y Constantes

GameMaker nos permite guardar configuraciones (ver figura B.7). Estas configuraciones nos permiten tener distintas constantes definidas para cada una de ellas, así pues, previo a generar un ejecutable o bien simplemente compilar el juego, podemos seleccionar la configuración que más nos interese. Las configuraciones las podemos crear bajo el submenú “Macros”.

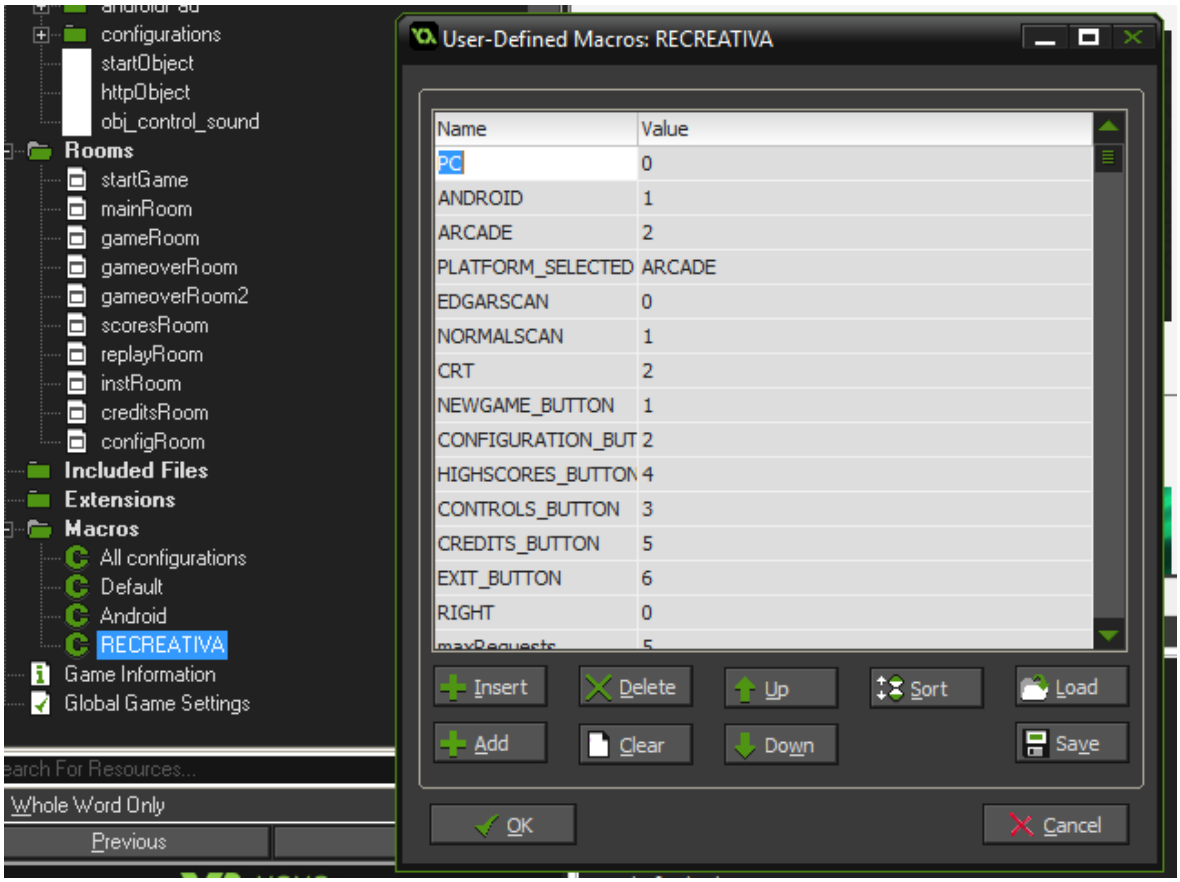

Figura B.7: Ejemplo de creación de constantes *GameMaker*

Anexos C

Implementación paso por paso de Maño Invaders

El objetivo de este apartado, es mostrar paso por paso como crear Maño Invaders, entrando en mayor profundidad que en el apartado de implementación y teniendo en cuenta las mejoras realizadas sobre el juego. No se abordará en el mismo el apartado de creación del servidor, puesto que ha quedado suficientemente cubierto en el apartado de implementación.

C.1. Pantallas del juego

El primer punto con el que comenzaremos es el diseño de las pantallas del juego. Para nosotros, cada pantalla será implementada con una *room* distinta, salvo en el caso de las fases del juego, en el que todas las fases se implementarán en la misma *room* con una serie de parámetros que nos permitan pintar una cosa u otra.

Para poder crear las pantalla pasaremos a definir lo que denominaremos "objetos controladores de pantalla". Estos objetos controladores no son más que simples objetos de *GameMaker* a los que les daremos una funcionalidad especial. La idea de estos objetos es que se inicialicen al comienzo de una *room*, para poder delegar en ellos la creación del resto de objetos que deberían aparecer en pantalla, así como posicionarlos desde los mismos. Estos objetos también contendrán atributos con una visibilidad que podríamos considerar de "vista", ya que durarán mientras dure el controlador, que en la mayoría de los casos será la duración de la *room*.

Lo primero de todo que crearemos al comienzo del juego serán cada una de las *room* que vamos a utilizar posteriormente así pues tendremos:

- *Room* inicial: Será imperceptible para el usuario y su objetivo será inicializar el juego definiendo y recuperando todo lo necesario para su funcionamiento. Será denominada "startRoom".

- *Room* principal: contendrá la primera pantalla visual a la que llegaremos al iniciar Maño Invaders. Será denominada "mainRoom".
- *Room* de puntuaciones: contendrá el histórico de diez máximas puntuaciones locales y online. Denominada "scoresRoom".
- *Room* de configuración: contendrá el menú de configuraciones. Denominada "configRoom".
- *Room* de instrucciones: contendrá el menú de instrucciones. Denominada "instRoom".
- *Room* de créditos: contendrá el menú de créditos. Denominada "creditsRoom".
- *Room* de juego: será la contenedora de todas las fases del juego. Se denominará "gameRoom".
- *Room* de gameOver: contendrá la pantalla de fin del juego. Denominada "gameoverRoom".
- *Room* de replay: contendrá la pantalla que nos mostrará las puntuaciones y nos dará la opción de ir al menú principal o volver a jugar.

C.1.1. *Room* inicial

Una vez tenemos creada nuestra "startRoom", definiremos un objeto "startObject" que será el encargado de recuperar toda la información relevante al comienzo del juego. Como este "startObject" incorporará una lógica pesada de definir mediante acciones, introduciremos una porción de código donde aglutinar todo, dejando simplemente fuera del código la acción que nos permita movernos a la primera *room* visible del juego: la "mainRoom".

En el "startObject" realizaremos lo siguiente:

- Declararemos absolutamente todas las variables globales que vamos a utilizar en el juego. De este modo las tendremos todas centralizadas en un mismo objeto.
- Haciendo uso de las funciones de *GameMaker* para trabajar con ficheros, recuperaremos de un fichero *.ini* todas aquellas puntuaciones codificadas, de las que no se haya recibido confirmación por parte del servidor en su almacenaje.
- Intentaremos enviar de nuevo las puntuaciones, para ello lanzaremos una petición *POST* por cada una de las puntuaciones.
- Recuperará todas las puntuaciones locales codificadas para mostrarlas en pantalla.
- Realizaremos una petición *GET* asíncrona al servidor para recuperar las puntuaciones del juego y almacenaremos el ID de petición en una variable global.
- En el caso de que nos encontremos ante una plataforma *Android*, recuperaremos el valor de configuración de juego (zurdo o diestro).

– En los casos PC y *Android* recuperaremos el valor asociado al filtro de *scanlines* (sí o no).
– En todos los casos recuperaremos el valor de si permitimos sonido o no.

La variable referente al sonido se recuperará de un fichero de parámetros de tipo *.ini* utilizando las funciones *ini_open()* e *ini_read_real()*. Por otro lado, las puntuaciones se leerán de un fichero normal para ello, primero de todo comprobaremos con la función *file_exists()* que dicho fichero exista para posteriormente abrirlo con la función *file_text_open_read()*. Lo recorreremos entero leyendo fila tras fila con la función *file_text_read_string()*. En dicho fichero primero el nombre del jugador y posteriormente en la fila inmediata su puntuación. Una vez recuperadas las puntuaciones, las volcaremos a un vector que viene predefinido en *GameMaker* con objeto precisamente de almacenar puntuaciones. La función que utilizaremos para escribir dicho vector es *highscore_add(nombre,puntuación)*

Añadiremos sobre todas las *rooms* un objeto que denominaremos "httpObject". Sobre este objeto únicamente añadiremos un evento de interrupción *HTTP* asíncrona. Dentro del evento, haciendo uso de la variable de *GameMaker* mapa *async_load* y del método *json_decode()* recuperaremos y decodificaremos el *JSON*[1] recibido y comprobaremos si el ID de petición recibido coincide con el enviado para la recuperación de puntuaciones. En caso de que el ID coincida procederemos a recuperar las puntuaciones, decodificando el *token* recibido del *JSON*: descifrado del *base64* utilizando el método *base64_decode()*, recorriendo la cadena resultante realizando un *XOR bit a bit* y por último una nueva invocación al método de descifrado *base64*. En caso de que el ID no coincida, asumiremos que se trata de una puntuación que ha sido procesada y de la cual estamos recibiendo respuesta. Para ello buscaremos en el fichero *.ini* de puntuaciones pendientes el identificador de puntuación. Si lo encontramos procederemos a eliminar dicha puntuación del fichero utilizando la función de *GameMaker ini_section_delete(ID)*.

C.1.2. Menú principal

Comenzaremos en nuestra "mainRoom", en la que pintaremos el menú principal y a la cual en primer momento habremos llegado desde la "startRoom". En primer lugar, crearemos un *sprite* sobre el que incorporaremos el que será nuestro fondo. Al ser un *sprite* dinámico, añadiremos varias imágenes con objeto de que iteren entre ellas dando sensación de movimiento. Posteriormente, Crearemos un objeto al que asociaremos dicho fondo; introduciéndole en su evento de creación una velocidad de

[1] http://www.w3schools.com/js/js_json_intro.asp

transición de imágenes de 0.25. Por último, posicionaremos el fondo sobre la *room* en la pantalla de edición de la misma.

Crearemos dos nuevos *sprites* y objetos, para realizar lo mismo pero con el logo principal del juego, así como con el logotipo de "RetroAcción".

El siguiente objeto que crearemos será el objeto controlador inicial, que se encargará de definir las variables globales que utilizará el juego y los desplazamientos a las siguientes pantallas. Los desplazamientos se realizarán en eventos alarma, para poder ser programadas externamente.

Los eventos del controlador serán los siguientes:

- Una alarma que introduzca valores a las variables globales que vaya a utilizar la fase inicial y que nos lleve a la room del juego.
- Alarma que nos lleva a la *room* de las puntuaciones
- Alarma que nos leva a la *room* de los créditos
- Alarma que nos lleva a la *room* de las instrucciones
- Alarma que nos lleva a la *room* de configuración.
- Alarma que simplemente tiene la acción de Salir del juego
- Evento *step* que comprobará interrupciones de teclado de varias teclas. Comprobará que se pulsen las teclas de teclado *Ctrl, O y P* mostrando un cuadro de dialogo con la función *get_string()* y evaluando si se ha introducido la clave correcta. Momento en el cual, utilizando de nuevo la anterior función solicitará el número de fase para comenzar a jugar. *Evento Draw, que comprobará si la plataforma es recreativa, en caso afirmativo escribirá por pantalla la frase "Pulse una tecla para comenzar."*
- Evento interrupción de teclado que si la plataforma es recreativa, invocará a la alarma de comienzo de fases.

Dentro de los objetos que se encargará de crear el controlador, en el caso de que la plataforma destino sea PC o *Android*, se encuentran aquellos que representarán a los botones que se mostraran en el menú, así como un controlador especifico, que se encargará de tratar el desplazamiento entre los botones mediante el teclado.

El primer paso para crear los botones será crear el objeto controlador que se encargará de todo lo que esté relacionado con ellos. Este controlador en su inicialización definirá que un atributo que nos servirá para identificar qué botón está seleccionado actualmente. Para poder definir qué número representa cada botón, iremos al apartado de *Macros* (o también conocido como constantes, ver figura C.1) de *GameMaker*, y vincularemos cada botón a un número. De este modo podremos comparar en un futuro el valor del atributo con las constantes en cuestión.

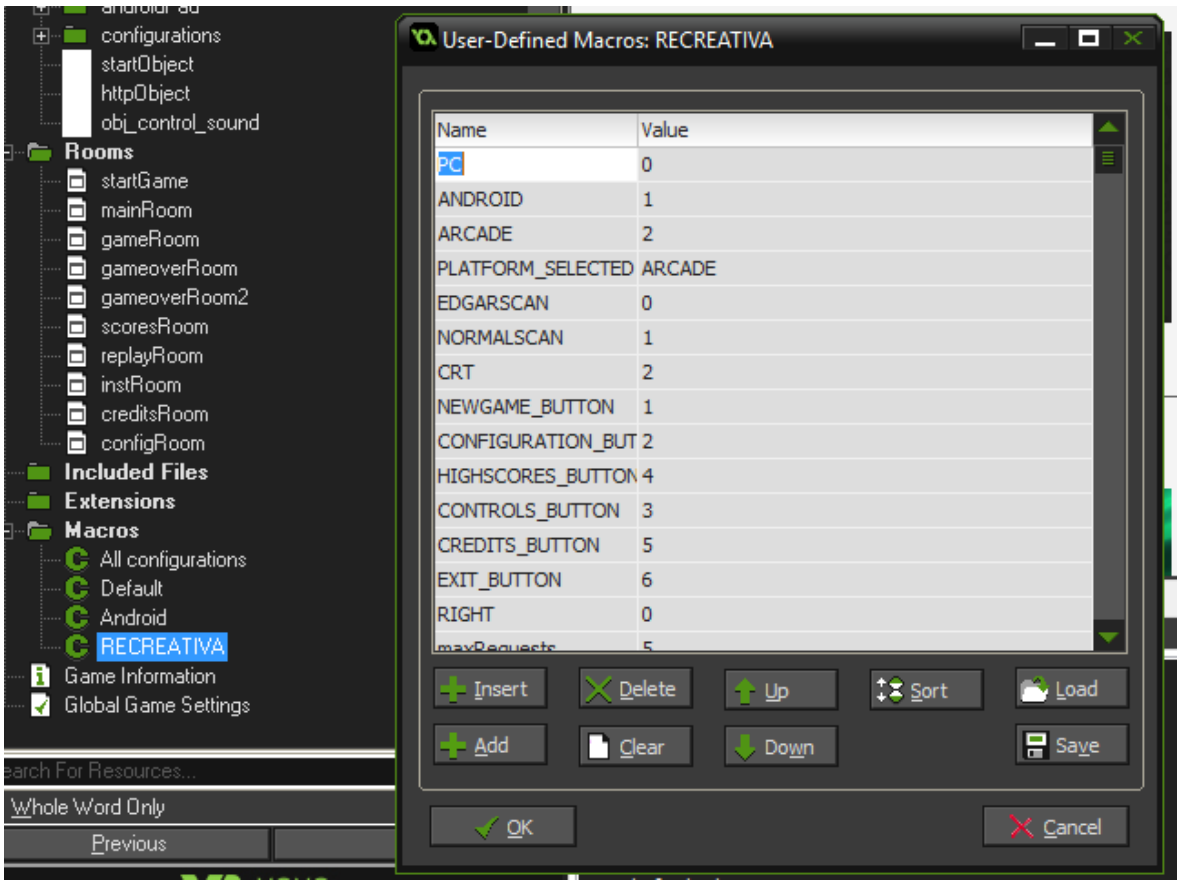

Figura C.1: Constantes del juego

Sobre el controlador de botones añadiremos un evento de interrupción de teclado al soltar la tecla *intro*. Dentro del evento para mayor comodidad, definiremos un bloque de código (ver figura C.2) en el cual compararemos el valor del atributo con todas las constantes definidas. Si coincide con alguna, programaremos la alarma del controlador inicial correspondiente, para cambiar de pantalla. También en este controlador añadiremos las interrupciones provocadas al presionar cada una de las flechas direccionales de teclado. En cada uno de estos cuatro eventos, realizaremos comprobaciones para que en función de que botón estuviera seleccionado y que flecha hayamos pulsado, el valor del atributo botón seleccionado cambie.

Una vez creado el controlador, faltará implementar los botones, para ello definiremos los *sprites* de cada uno de ellos. Dichos *sprites* contendrán tres imágenes para tratar los tres casos posibles: botón no seleccionado, botón seleccionado y botón presionado. Posteriormente asociaremos cada uno de estos *sprites* a un nuevo objeto.

Como todos los botones de esta pantalla tienen cierto comportamiento similar, crearemos un objeto padre, del cual heredarán todos la lógica, ampliándola en su caso. Este objeto padre, en su inicialización establecerá que la imagen inicial de todos los será siempre la del *sprite* de no seleccionado y que no iterará entre las otras dos

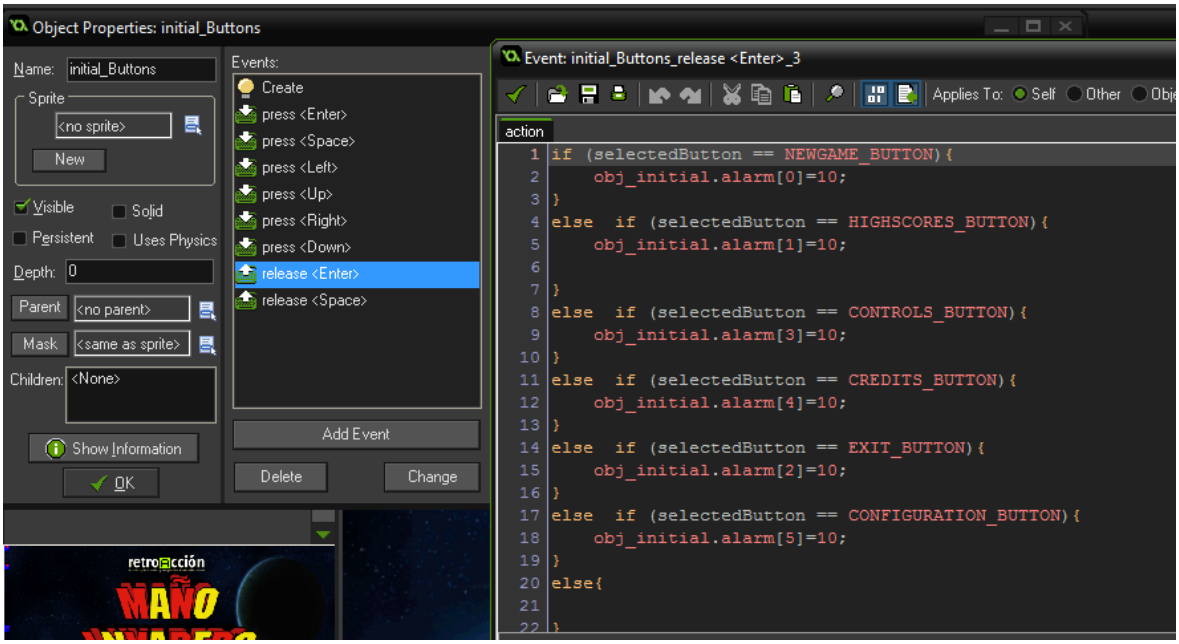

Figura C.2: Ejemplo de bloque de código

automáticamente. Definirá así mismo un evento que ocurrirá siempre que el ratón salga del objeto: *Mouse Leave*. Dentro de este evento, modificaremos el atributo botón seleccionado del controlador a 0 (ninguno seleccionado) y la imagen del objeto a la de por defecto. Crearemos otro evento, denominado *Left pressed* que indicará que el botón izquierdo del ratón esta presionado sobre el botón, cambiando en este evento el valor de la imagen seleccionada por la de botón presionado.

Tras definir el objeto padre de los botones, pasaremos a asociárselo al resto de ellos. Para ello haremos *click* en el apartado *parent*, seleccionando el objeto padre creado. Una vez establecida la relación solo nos quedará definir en cada botón dos eventos: *click* con el botón izquierdo del ratón y ratón sobre el objeto. En el primero de los dos simplemente programaremos la alarma correspondiente del controlador inicial para que realice el comportamiento deseado de dicho botón. En el segundo de ellos simplemente cambiaremos el índice de la imagen del objeto a la de botón seleccionado y cambiaremos el valor de la variable del controlador de botones que indica qué botón esta seleccionado.

El último objeto que faltará por crear del menú principal será el encargado de activar y desactivar el sonido. Crearemos su *sprite* con cuatro imágenes (activo, no activo, activo seleccionado y no activo seleccionado) y el objeto sobre el que asociaremos el *sprite*. Lo posicionaremos en pantalla mediante el controlador inicial en el cual habremos definido previamente una variable global que guardará el valor de si permitimos sonido o no. En el evento principal del botón del sonido, pintaremos su imagen por defecto en función del valor que tenga la variable global del sonido. Introduciremos tres eventos tratando:

– Interrupción de salida del ratón: se reiniciará la imagen a la de por defecto
– Presión de ratón: cambiaremos la imagen a la de seleccionado
– *Click* de ratón: cambiaremos el valor de la variable global de sonido en función de si permitimos sonido o no, así mismo, en función de dicha utilizaremos la función de *GameMaker audio_master_gain()* para activar o desactivar el sonido.Por último con objeto de que el valor se almacene para futuras partidas, mediante la función crearemos/abriremos (en función de si está creado o no) un fichero de propiedades, y haciendo uso de la función *ini_write_real()* escribiremos dicho valor en el fichero para leerlo en un futuro desde el controlador principal.

Ahora que hemos creado el primer parámetro que persistirá en futuras ocasiones, añadiremos una lectura del mismo desde el evento de creación del controlador principal para recuperar su valor siempre que empecemos el juego.

C.1.3. Menú de puntuaciones

Al menú de puntuaciones, como hemos visto antes, accederemos invocando a la alarma en cuestión que nos llevara al mismo mediante la función *room_go_to()*. La *room* encargada de dicho menú, será la denominada "scoresRoom" donde introduciremos todos los objetos que debieran intervenir en dicha pantalla.

Los objetos que aparecerán serán los siguientes:

– Objeto *background*: será el mismo que el utilizado en el menú principal, que nos mostrará el fondo.
– Título principal: al igual que en el menú principal será un objeto con un *sprite* asociado, que representará el título de la pantalla. El *sprite* tendrá dos imágenes asociadas (una con el título para puntuaciones locales y otra para online).
– Evento *Draw*: será el utilizado (ver figura C.3) para pintar propiamente por pantalla las puntuaciones.
– Botón volver: botón similar a los vistos en el menú principal. Tendrá el mismo número de *sprites* y los eventos serán iguales, pero en este caso trabajará sin controlador y el evento al pulsar la tecla *intro* recaerá sobre él mismo, el cual procederá a invocar a la función *room_go_to()* para volver al menú principal.
– Controlador de puntuaciones: será el encargado de recuperar las puntuaciones.

En nuestra pantalla de puntuaciones daremos cabida a puntuaciones locales y online. Así pues, en el evento de creación del controlador, definiremos un atributo que nos permitirá saber que tipo de puntuaciones estamos mostrando; añadiremos un atributo para almacenar la opacidad de las puntuaciones a pintar y programaremos

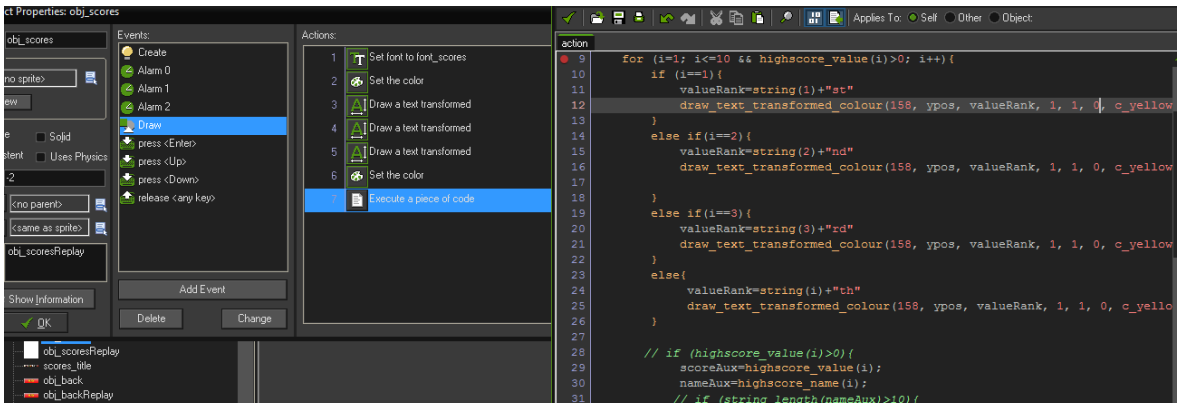

Figura C.3: Ejemplo de evento *Draw* para pintar las puntuaciones.

una nueva alarma. Esta nueva alarma, cuando se ejecute esta alarma invocará a una nueva encargada de comprobar si la opacidad es mayor que cero, en cuyo caso la decrementará en 0.02 unidades y volverá a invocarse. En caso, de que la opacidad llegue a cero. Si las puntuaciones que se estaban mostrando eran las locales, el atributo correspondiente cambiará a representar las puntuaciones globales y viceversa, cambiando así mismo, el título de la pantalla (se modificará el *image_index*). En ambos casos se invocará una nueva alarma. Esta nueva alarma, realizará lo contrario a la anterior; es decir, irá incrementando la opacidad hasta llegar a opacidad uno, momento en el cual programará la primera de todas las alarmas definidas para repetir indefinidamente el proceso.

Mientras modificamos el valor del atributo encargado de representar qué se está pintando; en el evento *Draw*, comprobaremos dicho valor y en función del mismo, si se tratan de puntuaciones locales se leerán las puntuaciones del vector de puntuaciones. Para acceder a las puntuaciones del mismo utilizaremos los métodos *highscore_value(i)* y *highscore_name(i)*. En caso de que estemos pintando puntuaciones online, las leeremos del vector introducido en memoria manualmente. Para finalizar el evento, definiremos también las posiciones donde pintaremos cada puntuación así como utilizaremos acciones para cambiar el color y tamaño del texto.

Por último, con objeto de que esta pantalla se utilice también en el *attract mode* de la maquina recreativa, condicionaremos el pintado del botón "Volver" a que la plataforma no sea recreativa y añadiremos una nueva variable contador. Esta variable la inicializaremos con valor cero y en el caso de la recreativa se incrementará en la segunda alarma creada cada vez que la opacidad llegue a cero. Tan pronto como alcance el valor tres, se saltará a la room de créditos. Añadiremos también interrupciones de cualquier tecla, que en el caso de recreativa se evaluaran llevándonos al menú principal.

C.1.4. Menú de créditos

Al igual que en los anteriores crearemos primero una *room* ("creditsRoom"), añadiremos el objeto fondo del menú principal, crearemos un título, un objeto controlador y el mismo objeto botón que en las puntuaciones.

Añadiremos todas las imágenes que mostraremos como créditos bajo el mismo *sprite* y posteriormente, por primera vez, añadiremos dicho *sprite* al objeto controlador. En el objeto controlador, en su inicialización comenzaremos estableciendo que las imágenes no cambien automáticamente y estableceremos una variable que nos indicará si estamos haciendo *fade_in* o *fade_out* de una imagen o no. Por último programaremos una alarma a los 70 *ticks* de reloj. Dicha alarma se encargará de realizar comenzar con el *fade_in* o *fade_out* de los créditos.

Así pues el evento alarma realizará lo siguiente:

- Comprobaremos si el atributo *image_alpha* (opacidad de la imagen) de *GameMaker* es mayor que cero y está realizando el *fade_out*, en cuyo caso, decrementaremos la opacidad un poco y re-programaremos la misma alarma.
- Comprobaremos si el *image_alpha* es inferior a uno y estamos realizando *fade_in*, en cuyo caso incrementaremos la opacidad un poco y re-programaremos la alarma.
- Para el resto de casos, si estábamos realizando el *fade_out* y hemos llegado a opacidad cero (nada visible), cambiaremos la imagen a la siguiente de la lista y re-programaremos la alarma. Por el contrario, si estábamos realizando el *fade_in* y hemos llegado a opacidad uno (totalmente visible), re-programaremos la alarma a 70 *ticks* de nuevo, para permitir leer tranquilamente el texto. En ambos dos casos se multiplicará por -1 el valor del atributo que nos indica si estamos haciendo *fade_in* o *fade_out* para pasar a indicar el caso contrario.

Con objeto de incluirse en el *attract mode* de la máquina recreativa, incluiremos también un contador que nos llevará a la *room* de instrucciones. Así mismo, se añadirán interrupciones para ir a la *room* inicial.

C.1.5. Menú de instrucciones

Este menú será prácticamente idéntico al anterior, pero cambiando el *sprite* asociado al objeto controlador. En este caso las imágenes serán aquellas que nos darán las instrucciones de cómo jugar y el objetivo del juego. La gran diferencia de este menú será, que al producirse el *fade-out* y cambiar de imagen tendremos que tener en cuenta el tipo de plataforma, ya que tendremos distintas imágenes cuando estemos mostrando los controles en PC o *Android*. Por último, se añadirá la misma lógica de

salto de *room* en el caso de que se trate de máquina recreativa. La *room* a la que saltaremos será a la de puntuaciones.

C.1.6. Menú de configuración

Este menú es distinto de los anteriores ya que en el no aparecerá ninguna imagen que vaya iterando. Solo será accesible vía menú principal y solo cuando se trate de PC o *Android*. El caso más complejo será *Android*, ya que necesitaremos mostrar dos opciones de configuración: si permitimos el filtro de *scanlines* y si la configuración es para zurdo o diestro.

Lo primero de todo será crear nuestros botones a modo interruptor. Para ello, partiendo de las distintas opciones de configuración que nos da cada opción: en el caso de *scanlines*, las palabras "Sí" o "No"; mientras que en el otro caso: "Izquierda" o "Derecha". Para cada opción crearemos dos imágenes, una que muestre el lado del interruptor como pulsado y la otra que lo muestre como no pulsado. La idea es que al juntar las imágenes de las dos opciones, como no podrán estar las dos activadas al mismo tiempo, dará impresión de interruptor. Dicho lo cual, creamos un objeto por cada opción y le asignamos los *sprites* creados. Una vez tenemos los objetos creados, creamos el objeto controlador, que se encargará de posicionar los objetos y tendrá en cuenta que si la plataforma es PC no mostrará el de la configuración para zurdos o diestros.

Dentro de cada uno de los objetos "parte del interruptor" añadiremos dos eventos, uno de creación y otro de interrupción de ratón.

- En el evento de creación evaluaremos siempre la variable global referente a la configuración y en cuyo caso, decidiremos cual de las dos imágenes/estado asignaremos al objeto.
- En el evento interrupción del ratón, evaluaremos el objeto sobre el que se ha hecho *click* con el ratón y en cuyo caso actualizaremos la variable global tanto en memoria como en fichero local, y actualizaremos las imágenes de los dos objetos "parte del interruptor".

C.2. Menú de Fin de Partida

La *room* que encapsulara la pantalla de *Game Over* será la "gameOverRoom" y a esta *room* accederemos siempre desde la *room* de las fases de juego. Crearemos un *sprite* para incluir la imagen del título del menú y un objeto controlador.

El menú de *Game Over* variará en función del tipo de plataforma en la que estemos jugando, así pues, por un lado para *Android* y PC será idéntico, pero no distinto para

recreativa. Con lo cual distinguiremos:

- PC y *Android*: Añadiremos un evento de creación al controlador y le añadiremos una porción de código condicionada a la plataforma PC y *Android*. En esta porción de código en primer lugar, mediante el método *get_string()* mostraremos un cuadro de dialogo preguntando en primer lugar por el *nick*. Evaluaremos que la respuesta no sea vacía y válida (que no contenga ningún carácter extraño). En caso de que todo este correcto almacenaremos la puntuación en el vector de puntuaciones y reescribiremos el fichero local. En caso de que el usuario presione cancelar, o bien el *nick* introducido este vacío mostraremos un nuevo dialogo haciendo uso del método *show_question()* en el que indicaremos que la puntuación no se almacenará, preguntándole si esta seguro. Caso afirmativo programaremos una alarma que se encargará de lanzar una petición *GET* para recuperar las últimas puntuaciones online y nos desplazaremos a la room de "volver a jugar". En caso negativo volveremos a preguntar por el *nick*. Una vez completada la parte del *nick*, si este se ha introducido, mediante un cuadro de dialogo pediremos el *email* y volveremos a realizar la misma validación, pero en este caso validaremos que contenga el carácter "@". Si todo ha ido correctamente, almacenaremos la puntuación en un fichero *.ini* con puntuaciones pendientes de recibir respuesta, en dicho fichero *.ini* estableceremos el número de intentos de envío a 1 (si una puntuación se intenta enviar más de N veces sin respuesta se borrará). Tras esto, realizaremos una petición *POST* para escribir dicha puntuación en el servidor indicando el tipo de plataforma. Por último, invocaremos la anterior alarma para recuperar las ultimas puntuaciones almacenadas en el servidor y moveremos a la *room* de "volver a jugar".
- En el caso de que la plataforma se trate de maquina recreativa, no podremos mostrar cuadros de dialogo; por lo tanto, crearemos un teclado virtual sobre el cual el usuario escribirá su *nick* y *email*. Una vez tengamos el *nick* y *email*, el procedimiento para enviar las puntuaciones será análogo, con la salvedad de que en lugar de movernos a la pantalla de "volver a jugar" nos moverá a la pantalla normal de puntuaciones, a modo *attract mode*.
  Para implementar el teclado crearemos un *sprite* con dos imágenes asociadas (teclado mayúsculas y teclado minúsculas), previamente creadas vía *Photoshop* y las asociaremos a un objeto "teclado". Los caracteres del teclado estarán todos a la misma distancia tanto vertical como horizontal. Por otro lado crearemos una imagen rectangular asociándola a un objeto "selector". La idea de este selector es que nos sirva para posicionarnos sobre el teclado, pudiendo moverse entre los caracteres del mismo.

Añadiremos dos objetos a modo botones de "Confirmar" y "Cancelar", para seleccionar cuando hayamos terminado de completar el *nick* o el *email*. En el evento inicial del controlador, añadiremos dos matrices, una por cada tipo de teclado virtual que encapsulará todas las teclas. Así pues capturaremos mediante evento de interrupción, el desplazamiento mediante las flechas del teclado. En función de la flecha pulsada, restaremos o sumaremos una unidad a columna o fila. Capturaremos el pulsado de la tecla barra espaciadora, en cuyo caso añadiríamos la tecla virtual pulsada a una variable donde almacenaremos el *nick*/*email* del usuario o si se tratará de la tecla *Shift*, cambiaríamos el tipo de teclado. Crearemos un evento *Draw* que se encargue de ir pintando en tiempo real la palabra que estamos formando en pantalla, justo encima del teclado. En caso de que en la fila anterior a los botones bajáramos, seleccionaríamos uno de los dos botones (dependiendo de si estamos más a la izquierda el de "Confirmar" y más a la derecha el de "Cancelar"), cambiando su *sprite* a seleccionado. Para la selección utilizaremos dos variables, una por cada botón, para comprobar si nos encontramos sobre cualquiera de los dos botones. De este modo si presionamos la tecla espacio cambiaremos el *sprite* de los mismos a "presionado". Capturaríamos también el soltado de dicha tecla. Si se suelta sobre teniendo seleccionado el botón "Confirmar", realizaremos las mismas evaluaciones que cuando lo introducíamos vía cuadro de dialogo y los mismos procedimientos. Si por un casual a la hora de introducir un *email* este no fuera válido (no tuviera ninguna "@") mostraríamos un mensaje a modo texto por pantalla, de que el *email* no es válido, instándole a pulsar en el botón continuar (no dejamos otra opción) que nos lleva de regreso a la edición del *email*. Si pulsamos en cualquiera de las introducciones el botón cancelar, o bien introducimos un *nick*/*email* vacio, mostraremos por pantalla un texto que informe de que no se almacenará y dando dos opciones, confirmar o cancelar. Si pulsamos confirmar(capturaremos la tecla espacio) llevaremos a la pantalla de puntuaciones donde comenzaremos el *attract mode*; en caso contrario volveremos a la edición. Como hemos comentado al principio si introducimos correctamente un *nick* o *email* se comportará igual que en PC o *Android*, pero con la salvedad de que al enviar las puntuaciones online, indicaríamos que se trata de la maquina recreativa.
Añadiremos una alarma para que en caso de que se produzca un periodo de inactividad sin que el usuario llegue a pulsar ninguna tecla, llevará al *attract mode* perdiendo la posibilidad de almacenar su puntuación

C.2.1. Pantalla de Volver a Jugar

Será una *room* idéntica a la de las puntuaciones en cuanto a funcionalidad se refiere, pero posibilitando un botón de "Nueva Partida" que invocará una alarma que inicializará las variables globales y nos llevará de nuevo a la *room* de juego. Esta nueva *room* se llamará "replayRoom".

C.2.2. Fases del Jugador

Como definimos anteriormente, a las fases del juego accederemos por vez primera desde el menú principal, al pulsar sobre el botón asociado a una nueva partida, que se encargará de programar una alarma. Dicha alarma, inicializará las variables globales iniciales del juego que siempre deberían comenzar siendo las mismas como:

- Las vidas: serán siempre tres.
- La velocidad de la nave del jugador.
- La velocidad inicial de los invasores.
- La altura inicial en la que comenzarán los invasores.
- La altura mínima en la que comenzarán los invasores.
- La velocidad máxima que alcanzarán los invasores.
- La velocidad de los disparos del jugador
- La velocidad de los disparos invasores.
- La puntuación inicial: será siempre cero.
- El estado de juego en pausa: inicialmente empezará no pausado.
- La velocidad mínima y máxima de los jefes finales.
- El número de fases totales antes de repetirse.
- La fase inicial en la que comenzará el juego: siempre será la fase uno de Zaragoza.

Una vez inicializadas las variables globales, la acción de mover de *room* nos llevará a la *room* del juego que denominaremos "gameRoom".

En esta nueva *room*, introduciremos un fondo estático de estrellas, así como una linea horizontal verde que sirva de separación entre el jugador y el apartado de las vidas restantes. Añadiremos un nuevo objeto que será el controlador de todas las fases del juego y que será el encargado de "pintar" todas las fases.

Así pues en la inicialización del controlador, inicializaremos todos los atributos que serán susceptibles de ser utilizados por el resto de los objetos intervinientes, así como los objetos participantes en sí.

Atributos que inicializaremos serán:

- Número total de enemigos que habrá en cada fila: constante

- Número de enemigos restantes en la fila roja.
- Número de enemigos restantes en la fila amarilla.
- Número de enemigos restantes en la fila rosa.
- Número de enemigos restantes en la fila azul.
- Número de enemigos restantes en total.
- Si la nave del jugador puede disparar.
- Si las naves invasoras pueden disparar.
- Cuatro vectores de *booleanos*, uno por cada fila, de dimensión igual al número de invasores por fila que nos indica si un invasor existe o no (si no ha sido destruido todavía tendrá un uno y en caso contrario un cero).
- Vector igual que el anterior, pero donde en lugar de almacenar *booleanos* guardamos la dirección de memoria de cada objeto invasor.

Entre los objetos que inicializaremos por ejemplo aparecen la nave del jugador, las vidas restantes, los invasores o los jefes finales en su defecto, etc. Todos estos participantes se detallarán en puntos siguientes.

Otro de los objetos controlador importantes crearemos en este momento son los de "comienzo de fase" y "fin de fase".

Comienzo de fase:

- Crearemos por cada objeto susceptible de intervenir en la lógica de las fases dos eventos manuales y un atributo por cada atributo que nos pueda interesar almacenar en un momento dado con objeto de ser recuperado más tarde (por ejemplo alarmas). En el primer evento manual haremos un *backup* de cada uno de estos atributos en los nuevos creados y pasaremos a definirlos con valor cero, es decir "pausará el juego". El segundo evento servirá para lo contrario, es decir, restablecer los valores anteriores.
- Invocaremos a los eventos de cada uno de los objetos para pausar el juego.
- Como imagen asociada tendrá un fondo negro
- Definirá un atributo que decidirá si tenemos que mostrar la cuenta atrás(1) o no(0).
- Un atributo cuenta atrás, que comenzará en 3
- En el evento *Draw*, en función del atributo correspondiente pintaremos o la cuenta atrás o el texto asociado a cada fase.
- Definiremos una alarma en la cual cambiaremos el valor del atributo que decide si estamos en cuenta atrás o no, para que pase a mostrarse la cuenta atrás.
- Definiremos una alarma que será invocada por primera vez desde la anterior, que comenzará a hacer un *fade_in* tanto auditivo como visual de la fase y

comprobará si la cuenta ha llegado a cero, en caso de que no, restará una unidad al contador y se re-programará en 60 *ticks*. En caso de que el contador haya llegado a cero, llamaremos al evento para eliminar la pausa producida en el juego, se destruirá el controlador y dará comienzo el juego.

El objeto controlador de fin de fase realizará lo implementaremos tal que:

- Creará un objeto fondo que tapará todo lo que haya debajo con opacidad cero.
- Pintará una frase a modo "Fase completada".
- Mostrará el texto de fase completada.
- Definirá una alarma que se encargará ir modificando la opacidad del fondo hasta llegar a opacidad uno así como el volumen de la canción que este sonando hasta llegar a volumen cero, momento en el que el fondo tapará toda la pantalla y reiniciará la *room*.

C.3. Nave del jugador

La elaboración del *sprite* de la nave del jugador, será sencillo puesto que mediante *Adobe Photoshop*[2] y a partir del cartel de RetroMañía 2007 podremos obtenerlo fácilmente. Introduciremos dicho *sprite* en *GameMaker* y lo asociaremos a un objeto en el que introduciremos toda la lógica de la nave del jugador.

En el evento de creación definiremos un atributo que nos indicará si se puede desplazar o no.

Se definirá a su vez una alarma para Volver a permitir moverse, es decir poner valor uno a la variable que regula el movimiento.

Añadiremos también los siguientes eventos:

- Usuario invadido: insertará el sonido de explosión de la nave, y establecerá la opacidad de la misma a cero para dar la impresión de haber desaparecido. Destruirá todo objeto representativo de vidas del jugador y el número de vidas pasará a ser 0.
- Usuario destruido: restaremos una vida al contador de vidas, pondremos opacidad cero, para que la nave deje de ser visible, insertaremos un sonido de explosión, destruiremos las instancias de los objetos vidas del jugador y volveremos a pintarlos en función del número que queden y programaremos una alarma.
- Alarma encargada de destruir al usuario y de invocar a la creación de un nuevo objeto nave del jugador.

[2]http://www.adobe.com/es/products/photoshop.html

– Desplazamiento de la nave hacia la izquierda: comprobará si el atributo que regula el movimiento está bloqueado y el juego no está en pausa. En tal caso desplazará el objeto 5 pixeles hacia la izquierda, cambiando de valor el atributo de movimiento para impedir que siga moviéndose y programando la alarma en los *ticks* marcados por la variable global que definía la velocidad del jugador. Dicha alarma en su ejecución volverá a permitir movimiento.
– Mismo evento que el anterior pero hacia el lado contrario
– Evento que en función de un atributo que regula el controlador, permitirá disparar al usuario o no.

Por último, destacar que los eventos de movimiento evitarán que el usuario se mueva por la pantalla demasiado rápido.

C.4. Invasores

Los *sprites* de los invasores originales se crearon al igual que la fase del jugador a partir del cartel de RetroMañía, pero con la ayuda del *Photoshop* se le consiguió dar el .efecto cachirulo"de los adoquines de Zaragoza intercalando el color de cada uno con cuadrados negros.

El diseño del resto de invasores se realizó utilizando nuevamente *Photoshop*, con imágenes de un tranvía real y frutas de Aragón e indexándolas posteriormente, reduciendo su paleta a cuatro colores para obtener un resultado pixelizado y que no desentonara con los invasores ya creados.

Una vez implementadas todas las imágenes el siguiente paso es introducirlos en *GameMaker*, introduciéndolos en un *sprite* por fila y asignando cada sprite a un objeto. Así pues el objeto de la fila roja, tendrá las imágenes de todos los invasores rojos, lo mismo ocurrirá con el resto. Se delegará al controlador de las fases el control de con qué imagen deben comenzar los invasores.

Todos estos objetos representantes de cada fila de invasores, tendrán como padre el mismo objeto, que llevará la lógica genérica de todos ellos.

C.5. Movimiento de los invasores

En la fase de análisis y diseño establecimos como tenía que ser la sincronía de movimiento de todos los invasores; para la implementación de la sincronía, vamos a utilizar un nuevo objeto, que hará las veces de controlador de movimiento.

Este objeto en su inicialización definirá los siguientes atributo: dirección inicial de los invasores (1 para mover a la derecha, -1 izquierda), la velocidad de movimiento que

se inicializará con el valor de la variable global(esta variable global se modificará al comienzo de cada fase dependiendo la misma), el número de pixeles que se desplazarán las filas en vertical en caso de tener que hacerlo, un atributo para regular si deben bajar las filas en vertical y por último programará una primera alarma que se ejecutará en el número de *ticks* definidos en el anterior atributo de velocidad de movimiento de los invasores .

Este controlador tendrá 4 alarmas distintas, cada una de ellas para regular el movimiento de cada fila. Siendo la encargada de mover las azules la que se programa en la inicialización del controlador.

El código dentro de estas alarmas es siempre el mismo:

- Si el juego no está en pausa, por cada columna de enemigos, haciendo uso de los vectores de existencia de enemigos y de posiciones en memoria de los enemigos, comprueba si alguno de los pertenecientes a filas que se muevan posteriormente está a menos de 30 pixeles de alcanzar uno de los dos límites laterales de la pantalla. En tal caso, activará el atributo de salto.
- Si el atributo salto está activo, comprueba si alguna de las filas, esta a menos de 32 pixeles de alcanzar el limite en el cual el jugador se consideraría invadido y se activaría el atributo del controlador "fin de juego".
- Si el atributo salto está activo, desplaza a todas las filas 20 pixeles hacia abajo.
- Si la variable fin del juego está activa en el controlador invoca al evento de invadido del objeto nave del jugador.

Tras ejecutar el código anterior, comprueba si el número de naves del color asociado a la alarma es superior a cero y en tal caso los desplaza en el sentido de la dirección del atributo movimiento, insertando a su vez el sonido asociado a dicha fila de invasores. Posteriormente comprueba si existen enemigos en la fila superior, programando su alarma, si no pasaría en la siguiente, si no, a la siguiente, y en último caso, volvería a programarse así misma. La programación será siempre en el número de *ticks* del atributo velocidad de movimiento de los invasores.

La velocidad de movimiento (*ticks* que acontecen entre alarma y alarma), vendrá definida como una variable global que se definirá al comienzo de la fase y que estará directamente relacionada con la fase. Inicialmente para la fase 1 se le otorgará el valor de doce, pero este valor inicial en cada fase será menor hasta llegar a un valor mínimo de dos. Por otro lado, durante las fases dicho valor también disminuirá conforme vayan desapareciendo enemigos, hasta llegar a un valor de velocidad dos. Por otro lado, conforme avancemos de fases, la altura a la que comiencen los invasores respecto de la nave del jugador disminuirá.

C.6. Barreras defensivas

El diseño de las primeras barreras defensivas de *GameMaker* se obtuvo del cartel de RetroMañía. Tomando como ejemplo estas tres barreras y siguiendo el mismo proceso utilizado en los invasores mediante *Photoshop*, se realizaron el resto de las definidas en el apartado de diseño.

Las barreras están realizadas .ª modo rejilla", es decir una barrera de Maño Invaders se compondría de una serie de objetos, posicionados en pantalla de tal modo que estuvieran tocando unos con otros, dando la impresión de formar todos ellos una misma barrera. De este modo conseguimos que cada porción de la barrera actué de manera independiente y pueda ser eliminada individualmente. Por cada objeto de la rejilla, tendremos asociado su correspondiente trozo de imagen. La división fue trivial puesto que *Photoshop* permite partir imágenes fácilmente.

Todos los objetos que representen un trozo de la barrera tendrán como padre a un mismo objeto que la única lógica que recogerá será que al recibir un disparo, de los invasores o del jugador, se destruirá. Esto visualmente produce un efecto de que la barrera ha sido destruida.

En cada fase, el controlador de las fases se encargará de qué barreras defensivas debe pintar y cómo debe hacerlo ello colocará cada objeto trozo de barrera donde correspondan.

C.7. Disparos

Como vimos en el apartado de diseño, tenemos dos tipos de disparos: aquellos realizados por el usuario y los realizados por las naves invasoras. En ambos casos, realizamos los *sprites* de los disparos (uno para el usuario y cuatro distintos para los invasores) utilizando el editor de imágenes de *GameMaker* y encapsularemos así mismo la lógica en dos objetos distintos, uno que servirá para el disparo del usuario y otro que hará de padre de los objetos representativos de cada uno de los *sprites* de disparo de los invasores.

C.7.1. Disparos del jugador

En su creación el objeto representante del disparo del jugador, comenzará a moverse en dirección vertical, en sentido negativo en el eje y. La velocidad del mismo la definiremos en base a la variable global de movimiento de los disparos del jugador. Introduciremos un sonido especifico para el disparo; por último definiremos un atributo que nos indicará si el disparo ha superado la mitad superior de la pantalla o no.

Como eventos definiremos los siguientes:

- Evento de colisión con invasor. Destruirá la bala y si el número total de enemigos es cero, incrementará el número de fase actual y programará una alarma en el controlador de fase encargada de terminar la fase
- Evento colisión con barrera, que destruirá el disparo
- Evento al salir de los límites de la pantalla que destruirá el objeto
- Un evento de *Step*, que compruebe la posición del disparo, comparándola con la de la mitad de la pantalla, en caso de superar la mitad, activará el atributo encargado de representar si ha superado la mitad de la pantalla y volverá a activar el atributo del controlador de fase que posibilita el disparo.
- Evento de destrucción del disparo: siempre que el disparo se destruya volverá a activar el atributo del controlador de fase que posibilita el disparo.

En el objeto padre de los invasores añadiremos un evento de colisión contra el disparo del jugador, que produzca una explosión, destruya al invasor e introduzca un sonido de explosión.

En el lado de las barreras, en su objeto padre añadiremos también un evento de colisión contra el disparo del jugador que se limite a destruir el objeto.

Por último en el evento controlador de fases, añadiremos un evento de captura del pulsado de la barra espaciadora. Dicho evento invocará a otro evento de usuario que definiremos en el objeto nave del jugador. Este evento de usuario comprobará en primer lugar si el juego no está pausado, posteriormente si el atributo que regula la posibilidad de disparo está activo y si también se cumple creara una instancia del objeto disparo, desde la cúpula del Pilar (nave del jugador), pasando a desactivar la posibilidad de disparar en el atributo correspondiente.

C.7.2. Disparos de los invasores

Para implementar los disparos de las naves invasoras, primero de todo crearemos los *sprites* de cada tipo de disparo junto con sus objetos asociados. A todos los objetos les añadiremos un nuevo objeto padre del cual heredarán la lógica.

Dentro de este objeto padre, definiremos en su evento de creación que deberá moverse en sentido vertical y a favor del eje y, estableciendo como velocidad de dicho movimiento la definida en la variable global “velocidad de disparos enemigos”.

Como eventos de colisión añadiremos:

- Colisión contra el jugador: creará una explosión en el lugar de impacto, destruirá al disparo y llamará al evento de usuario del objeto nave del jugador encargado de tratar con el recibimiento de un impacto.

– Colisión contra una objeto barrera: se destruirá el disparo.
– Colisión contra un disparo del jugador: se destruirán ambos disparos creando una explosión.

Para implementar la posibilidad de disparo de cada nave será necesario establecer una alarma que se ejecute cada cierto tiempo en cada invasor. Esta alarma en primera instancia mediante un contador comprobará que queden en total más invasores que los que forman una fila al inicio (en nuestro caso se establece como 7 este valor). En caso afirmativo, comprobará la distancia horizontal entre la nave invasora y la nave del jugador, si la distancia es menor que 50, establecerá un peso de 4. En cualquier otro caso asignará un peso de 1. Mediante la función *random()* de *GameMaker* obtendremos un numero aleatorio entre 0 y 7. Si el número obtenido es inferior al peso del invasor comprobaremos que no tenga ningún invasor delante gracias a los vectores de existencia de invasores, De este modo, bastará con comprobar las filas anteriores para saber si un invasor no tiene a nadie delante. Así pues, en caso de que un invasor no tenga a otro delante, y el número aleatorio obtenido sea inferior a su peso, creará un objeto disparo del tipo correspondiente a su color de invasor y desactivará el atributo del controlador de fase para que ningún otro invasor dispare en ese momento.

Por cada uno de los objetos disparos de fila, introduciremos un evento de tipo *Outside Room* que se ejecutará siempre que el disparo salga de la pantalla. En este evento incluiremos la acción de destruir el disparo.

Por último añadiremos un evento de destrucción, que se ejecutará siempre que el disparo sea destruido por cualquiera de las razones anteriores.En este evento volveremos a activar el atributo del controlador de fase para que los invasores puedan volver a disparar.

C.8. Potenciadores

Crearemos un *sprite* y un objeto para tratar cada uno de los *boosters*. Crearemos también dos nuevas imágenes de la nave del jugador con los colores de las banderas de los *boosters* y lo añadiremos al objeto nave del jugador, esto se utilizará para cambiar de imagen a la nave cuando obtenga un *boosters*.

Antes de comenzar con la lógica de los objetos de los *boosters*, como ahora el *sprite* de la nave del jugador tendrá tres imágenes, procedemos a definir en el evento de creación del objeto nave del jugador que la velocidad de transición es cero, y que la imagen inicial es la del estado normal. Aprovecharemos para añadir dos nuevos atributos al objeto nave del jugador, que controlarán si tiene activado cada uno de los *boosters* (en la práctica serán *booleanos*). Al mismo tiempo añadiremos en el controlador

de fase un atributo con dos valores (uno si hay un *booster* en pantalla, y cero en caso contrario) y dos alarmas. La primera de las alarmas introducirá un sonido que nos avisará de que el efecto de un *booster* se está terminando y la otra, será la encargada de devolver a la nave del jugador sus colores originales y desactivar los efectos.

Pasando ya a la implementación de los objetos *boosters*, primero de todo crearemos un objeto padre para establecer comportamientos comunes.

En el objeto padre definiremos lo siguiente:

- Alarma encargada de destruir el *booster*.
- En el momento de la creación añadiremos una acción de movimiento del *booster* en sentido a favor del eje X con una velocidad constante.
- En la creación programaremos la alarma de destrucción con un tiempo de 300 *ticks* si es una fase de jefe final o 900 si es fase normal.
- Reproduciremos un sonido mientras está en pantalla.
- Definiremos la coordenada X máxima hasta donde se moverá. Introduciremos un valor aleatorio entre 0 y 600.
- Añadiremos un evento *Step* que comprobará si hemos superado la coordenada máxima, momento en el cual detendrá el *booster* (velocidad cero).
- Añadiremos un evento de destrucción en el cual pararemos el sonido del *booster*. Y marcaremos que no hay ningún *booster* en pantalla en el atributo del controlador de fase (valor cero).

Dentro de los dos objetos *boosters* definiremos un evento de colisión con el disparo del jugador que contendrá en cada uno de los casos lo siguiente:

- Detención del sonido del *boosters*.
- Introduciremos un sonido de obtención del *booster*.
- Modificaremos la imagen de la nave del jugador, cambiándole el índice.
- Activaremos el atributo que corresponda según el *booster* para indicar que su efecto está activo.
- Programaremos las alarmas del objeto nave del jugador. Por un lado la que nos avisará de que el *booster* se está agotando, la programaremos en 400 *ticks*; mientras que la alarma que devolverá a la nave del jugador a su comportamiento original en 600 *ticks*.

Una vez, tengamos lo anterior modificaremos los eventos de colisión del jugador para condicionarlos a que el atributo del jugador "invencible" este activo o no. Por otro lado, en el objeto controlador de fase ampliaremos su evento interrupción del teclado de tecla barra espaciadora para permitir disparar tres disparos a la vez. Para ello, crearemos

un *sprite* disparo idéntico al existente pero de color rojo y lo asociaremos a un objeto nuevo, dicho objeto contendrá la misma lógica que el disparo normal. Así pues, solo quedará añadir la creación de dos instancias de este nuevo objeto que partan desde las coordenadas de los picos de las torres del pilar, para dar efecto "cañones" y añadir las colisiones entre el resto de objetos y el nuevo objeto disparo.

Aprovechando que hemos realizado el *booster* de la invencibilidad añadiremos en el evento de creación de la nave del jugador que inicialice la variable "invencibilidad" como activa, reduciendo así mismo la opacidad del jugador. Crearemos también una alarma la cual programaremos en dicho evento para que pasados unos pocos segundos desaparezcan los efectos anteriormente programados.

C.9. Jefes Finales

Todas las fases de los jefes finales al igual que en las fases normales accederemos vía el objeto controlador de fase, que al detectar que se trata de una fase de jefe final, no pintará invasores y barreras, y se encargará de instanciar a los objetos correspondientes a los jefes finales. Como decisión de implementación se ha tomado que al tener todos los jefes un comportamiento bastante diferenciado se ha optado por que no partan de un mismo objeto en la jerarquía.

C.9.1. Jefe final fases Zaragoza: Alcalde

El *sprite* del "Alcalde" se elaboró utilizando *Photoshop* y mezclando como se definió en la fase de diseño, imágenes del antiguo "Alcalde" de Zaragoza y del "Alcalde" de la serie animada "Los Simpsons", personalizándolo añadiéndole elementos diferenciadores. Así pues, una vez tenemos todas las imágenes del "Alcalde", las asociamos a un *booster* y lo vinculamos a un objeto "Alcalde" que crearemos. Dentro del evento de creación necesitaremos definir lo siguiente:

- Atributo que nos indique la cantidad de saltos que dará en su ráfaga de saltos y que trabajará a modo contador de saltos restantes.
- Atributo que nos indique si está saltando o no.
- Atributo que nos indique la vida máxima con la que ha comenzado al principio.
- Atributo que nos indique la vida actual.
- Atributo que nos indique la dirección del "Alcalde" (-1 izquierda, 1 derecha, 0 parado)
- Un atributo que nos indique si está enfadado tras recibir un disparo.
- Un atributo para almacenar la vida restante en porcentaje.
- Atributo para indicar que el jefe ha sido o no derrotado.

El siguiente punto será crear un objeto que nos servirá para mostrar la vida del jefe en pantalla. Para ello creamos un nuevo objeto con solo un evento *Draw* en el cual, pintaremos un rectángulo rojo no relleno, de 100 pixeles de anchura y 10 de altura. Sobre el mismo pintaremos otro rectángulo rojo, esta vez relleno, que nacerá desde el mismo punto que el anterior, pero su anchura será el porcentaje de vida restante (al ser un porcentaje sobre 100 y la anchura total 100 nos funciona perfecto).

Posteriormente definiremos una alarma que trabajará como nuestra alarma de ráfaga. Es decir cuando se active comenzará la ráfaga. En dicha alarma definiremos el siguiente comportamiento:

- Inicializaremos un contador de iteraciones de la ráfaga de saltos. Dependerá de la fase en la que nos encontremos. Básicamente realizará una iteración más por cada vez que hayamos superado todas las fases.
- Configuraremos la velocidad horizontal a cero.
- Marcaremos el atributo asociado a si está saltando con valor 1.
- Definiremos un movimiento con una velocidad constante de valor 5. En el sentido contrario del eje y, es decir, hacia arriba.
- Estableceremos una gravedad en dirección 270º, que producirá una deceleración de la velocidad de movimiento hasta el punto de que llegado un momento comenzará el objeto a invertir su dirección de movimiento a favor del eje y.
- Introduciremos un sonido de salto.
- Si acaba de comenzar la ráfaga (le quedan cinco saltos), introducirá un sonido a modo risa malévola.
- Creará cuatro instancias de objetos disparo de invasor y les establecerá a cada uno las siguientes formulas para calcular su dirección en grados:

$$Disparoazul = 250 - 10 * (5 - saltosrestantes)) \tag{C.1}$$

$$Disparorojo = 265 - 5 * (5 - saltosrestantes)) \tag{C.2}$$

$$Disparorosa = 275 + 5 * (5 - saltosrestantes)) \tag{C.3}$$

$$Disparoamarillo = 290 + 10 * (5 - saltosrestantes)) \tag{C.4}$$

- Definiremos un nuevo atributo que indique si esta abriendo o cerrando el angulo de disparo (inicialmente siempre se abrirá) y en cada iteración cambiará. Así pues si nos enfrentáramos al "Alcalde" por segunda vez, una vez llegará a cero el contador de saltos restantes, reiniciaríamos la ráfaga de saltos pero esta vez incrementando una unidad el contador. Hasta llegar al inicio, momento en el cual cesaría la ráfaga.

Una vez finalizada, programaremos la anterior alarma en el evento de inicialización a los 300 *ticks* de reloj.

La siguiente alarma que crearemos será la encargada de que el jefe final vaya lanzando "armas" a su paso, para ello, primero de todo crearemos los *sprites* de las armas y los asociaremos cada uno a un objeto. Crearemos un objeto padre de todas las armas cuyo comportamiento ante colisiones será idéntico que el objeto padre de los disparos de invasores. Este objeto se diferenciara del anterior, en que en su inicialización se dirigirá hacia la posición del jugador, es decir, no solo se moverá en el eje x si no también en el y. Por lo tanto haremos uso de la variable *Move towards point*. Una vez creadas las armas del "Alcalde", estableceremos en la alarma que calcule un número aleatorio entre cero y uno con la función *irandom()*. En función de lo obtenido instanciará un objeto u otro. Añadiremos así mismo un sonido para el lanzamiento del arma.Por último, programaremos dicha alarma, tanto en la misma alarma para que se vuelva a ejecutar como en el evento de creación del "Alcalde" con la siguiente formula:

$$Alarma = ticks_{min} + random(ticks_{max} - ticks_{min}) \tag{C.5}$$

Siendo $ticks_{min}$ los ticks mínimos hasta los que se volverá a ejecutar (variable global definida al comienzo al comenzar las fases) y $ticks_{max}$ los máximos.

Añadiremos una nueva alarma que se encargara de tratar con el cambio de fase una vez derrotemos al jefe final que se limitará a sumar uno al contador de fases jugadas, crear un objeto controlador fin de fase y destruir la instancia del objeto "Alcalde".

Añadiremos otra alarma para poderla llamar y que deje al jefe en su estado de no enfadado (valor cero en su atributo correspondiente).

Proseguiremos definiendo el movimiento concretamente del jefe. Para ello crearemos un evento de *Step*. En dicho evento comprobaremos que no este en pausa el juego y que el jefe no haya sido derrotado todavía. Si se cumplen las condiciones anteriores comenzaremos con la lógica que se tiene que realizar en cada *tick*:

- Comprobaremos si el "Alcalde" está enfadado, saltando y la dirección para decidir que imagen del mismo se debe pintar.
- Si el "Alcalde" no está saltando, primero de todo definiremos su dirección de movimiento, tal que así:
  - Distancia con el jugador más de 30 pixeles positivos: dirección izquierda (-1)
  - Distancia con el jugador más de 30 pixeles negativos: dirección derecha (1)
  - Si se encuentra a menos de 30 pixeles, se quedará parado. (0)

  Su velocidad será pues,

$$movimiento = velocidadbase * direccion \tag{C.6}$$

  Siendo la velocidad base de los jefes finales una variable global.

– Si está saltando y la coordenada y es mayor de 200, lo cual quiere decir que la gravedad ha hecho que bajará más de su posición inicial, volvemos a establecer la posición vertical del jefe final en 200 . Si el contador de salto es mayor que cero lo decrementamos, si por el contrario es igual a cero, programamos la alarma de disparos aleatorios y establecemos que no está saltando en su variable determinada.

Nos quedará definir qué ocurre cuando el "Alcalde" recibe un disparo (de cualquier tipo) del jugador.

Si su vida es mayor que cero:

– Decrementaremos su vida en cinco unidades.
– Introduciremos una explosión en el lugar de impacto.
– Estableceremos el atributo enfadado como activo (1).
– Destruiremos el objeto colisionado contra el "Alcalde".
– Si la vida tras el impacto es mayor que cero:
  - Aumentaremos 15 puntos la puntuación
  - Produciremos un sonido de impacto.
  - Programaremos una alarma que nos permitirá hacer que vibre el "Alcalde" tras recibir un disparo
– Si la vida tras el impacto es cero:
  - Ponemos a cero todas las alarmas del "Alcalde"
  - Establecemos el atributo "jefe derrotado" a valor uno.
  - Detenemos al "Alcalde" en cuanto a movimiento.
  - Programamos la alarma que cambiará de fase.

Por el contrario si la vida del "Alcalde" al recibir el disparo llega a cero, comenzará su ráfaga final. Definiremos lo siguiente:

– En el evento *Step*, detectaremos que su vida ha llegado a cero y obviaremos la lógica anterior, pasando a comprobar si su coordenada Y es la inicial, momento en el cual comenzará a "saltar" del mismo modo que en la ráfaga normal pero lanzando armas en lugar de disparo. Repetiremos la ráfaga de salto ampliando y reduciendo el angulo de disparo del mismo modo que lo hacíamos en el control de las iteraciones en la ráfaga normal.
– Programaremos una alarma que llegada su ejecución destruirá al "Alcalde" e inicializará al controlador de cambio de fase.
– Programaremos una nueva alarma que mediante el uso de un atributo, irá aumentando o disminuyendo de tamaño al "Alcalde". Esta alarma cada vez que finalice se invocará así misma, y cada vez que llegue a lo que considera tamaño máximo y mínimo cambiará el valor del atributo.

– Crearemos un objeto nuevo que pintará en pantalla la palabra “Peligro!”. Este nuevo objeto tendrá una alarma, que se irá invocando para hacer aparecer y desaparecer la palabra en pantalla.

Como última alarma del “Alcalde” nos quedará crear la alarma que provoque que el “Alcalde” vibre al recibir un disparo, para ello creamos un atributo que nos servirá a modo cuenta regresiva. Al recibir un disparo lo programaremos con valor cinco, así pues este será su valor inicial. Dentro de la alarma, primero de todo comprobaremos si la cuenta regresiva ha llegado a cero, en caso de que no lo haya hecho, en función de si el número de la cuenta regresiva es par o impar, sumaremos 5 a la coordenada -Y- actual del “Alcalde” o se lo restaremos.Restaremos en este caso también una unidad a la cuenta regresiva. En el caso de que la cuenta regresiva haya alcanzado el valor cero, definiremos al jefe como “no enfadado” (valor cero de su atributo correspondiente).

C.9.2. Jefe final fases Expo: Fluvi

El procedimiento para “Fluvi” será idéntico al del “Alcalde”, por lo tanto seguimos todos los pasos descritos en el “Alcalde” y modificamos los sonidos por aquellos relacionados con “Fluvi” e introducimos dos nuevas armas que sustituyan a las del “Alcalde”.

La primera diferencia radica en que debemos añadir una ráfaga de movimiento previa a la de disparo que ya existía. Para ello definimos un atributo que nos indicará si estamos en esta nueva ráfaga de movimiento. Crearemos una nueva alarma y en la inicialización de “Fluvi” la programaremos en lugar de la anterior de disparo, así como tras finalizar la ráfaga de disparo. Esta nueva alarma comprobará si la coordenada X de “Fluvi” es mayor de 320 (mitad de la pantalla) estableciendo un movimiento en horizontal con una velocidad el doble que la velocidad base y en dirección izquierda. Por el contrario, si la coordenada es menor a 320, el desplazamiento será hacia la derecha. Como a su paso deberá lanzar con más frecuencia armas, creamos una alarma igual que la del lanzamiento de armas normales, pero con al diferencia de que está alarma la programaremos cada 20 *ticks* solo. Por último en el evento *Step* detectaremos cuando “Fluvi” ha llegado a los limites laterales de la pantalla, momento en el cual lo detendremos y activaremos la alarma de la ráfaga de disparo.

La ráfaga de disparo de “Fluvi” cambiará respecto a la del “Alcalde” en futuras ocasiones que nos enfrentemos al mismo, en que intercalará siempre una ráfaga en la que dispara disparos (Siempre comenzará con disparos) y otra en la que lanzará armas.

La siguiente diferencia es que deberemos implementar que “Fluvi” se mueva dando saltos, tanto de normal como en la ráfaga de movimiento lateral. Para ello en el evento

Figura C.4: Posibles estados del “Alcalde” y sus armas utilizadas.

Step, si detectamos que “Fluvi” esta en la coordenada Y base o más abajo y no se encuentra en la ráfaga de saltos, definiremos su posición en la coordenada Y en su posición inicial y estableceremos una velocidad hacia arriba y una gravedad en 270º. De este modo, “Fluvi” saltará hacia arriba y caerá, dando la impresión de que va dando saltos al moverse.

Por último modificaremos tanto la vida base de “Fluvi” para que sea 20 unidades mayor que la del “Alcalde” y cambiaremos para que nos otorgue 20 puntos en cada disparo recibido.

Figura C.5: Estados de Fluvi y sus armas utilizadas.

Por último al igual que en el “Alcalde”, implementaremos una ráfaga final al llegar su vida a cero. Esta ráfaga final estará basada en la ráfaga de desplazamiento de “Fluvi”. Heredará de “Alcalde” la alarma que hace que aumente o disminuya de tamaño. En esta nueva ráfaga cuando “Fluvi” llegue al limite de la pantalla lo detectaremos y cambiaremos su dirección de movimiento, es decir no cesará, si no que cesará cuando la misma alarma que definimos para el “Alcalde” haga que explote.

C.9.3. Jefe final fases Aragón: SuperMaño

Partiremos con un objeto idéntico en lógica a "Fluvi" pero con los *sprites* creados para el "SuperMaño", así como dos nuevas armas, más vida inicial (130 unidades) y sonidos específicos.

En este caso no será necesario crear nuevas alarmas ya que heredará las ráfagas de "Fluvi", pero añadiéndoles nuevos comportamientos.

En primer lugar, definiremos un atributo "número de rebotes de la ráfaga lateral" con valor uno. Así pues, cuando estamos en la ráfaga de movimiento lateral y comprobamos en el evento Step si "SuperMaño" ha tocado el borde, en caso de que así sea, comprobaremos si el atributo rebotes es igual a cero, en caso de que si lo sea, daremos comienzo a la ráfaga de disparos, pero en caso contrario, haremos que "SuperMaño" comience una nueva ráfaga estableciendo una velocidad de movimiento constante al lado contrario. Cada vez que superemos las doce fases y nos volvamos a enfrentar a "SuperMaño", tendrá un rebote más.

Dentro de la lógica de la ráfaga de saltos, modificaremos el evento encargado de producir los disparos, cambiando los disparos por armas de "SuperMaño". Añadiremos un nuevo atributo que nos servirá a modo de contador de cuantas repeticiones de saltos realizaremos; lo definiremos con valor uno. Por otro lado definiremos un atributo que nos indicará si el contador de saltos normal se está decrementando o aumentando. Posteriormente en el evento *Step*, en la comprobación de si el contador de saltos ha llegado a cero, comprobaremos también si ha llegado a cinco. En caso de que la condición anterior se cumpla introduciremos otra comprobación que comprobará si el contador de repeticiones de saltos ha llegado a cero. En caso negativo estableceremos que el contador de saltos esta incrementando. Así pues tras estas comprobaciones comprobaremos previo a invocar a la alarma, comprobaremos si el contador está incrementando o decrementando y según sea el caso decrementaremos o incrementaremos una unidad. En cada ocasión que nos enfrentemos a supermaño tras haber superado las doce fases, se incrementará en una unidad la iteración, pasando a iterar más veces en su ráfaga de saltos.

Al igual que los otros jefes predecesores "SuperMaño" también tendrá su ráfaga especial al llegar su vida a cero. Al igual que el "Alcalde" comenzará a saltar, pero sin lanzar objetos a priori. En su caso, en lugar de lanzar objetos en varios ángulos, haremos que lance armas desde el cielo. Para ello en el evento *Step*, en el que hemos introducido al igual que con el "Alcalde" la ráfaga de saltos. Introduciremos un bucle que comenzará en un número aleatorio entre 0 y 20. Esta será la coordenada -*X*- de la primera arma que lanzará. En cada iteración incrementará esa coordenada en 70

Figura C.6: Estados de “SuperMaño” y sus armas utilizadas.

pixeles hasta llegar a un valor superior a 620 (se saldría de la pantalla). Por otro lado en cada iteración la coordenada -*Y*- será un número aleatorio entre 0 y -250, de este modo conseguimos que cada arma caiga a una altura distinta. El resto de alarmas se mantendrán como en otros jefes (palabra de “peligro”, explosión del jefe, y aumento de tamaño).

C.10. Música del juego

GameMaker nos permite añadir música y sonidos fácilmente. Simplemente deberemos crear un objeto sonido y añadirle un fichero en formato *.mp3, .ogg o .wav*. Una vez elegido el fichero, nos dará opciones para cambiar el *bitrate*[3] y el *Sample Rate*[4]. También nos dará opción de elegir si deseamos escuchar el sonido en "mono" o "estéreo" y si el sonido es de 8 o 16 bits.

Por último nos dará tres opciones:

- Sin compresión y sin transmisión: se aloja en memoria y requiere menos recursos de CPU.
- Comprimido y sin transmisión: se aloja en memoria y requiere más recursos de CPU.
- Comprimido (no en carga) y sin transmisión: alto uso de memoria y poco de CPU.
- Comprimido y con transmisión: se aloja en disco, requiere un alto uso de CPU

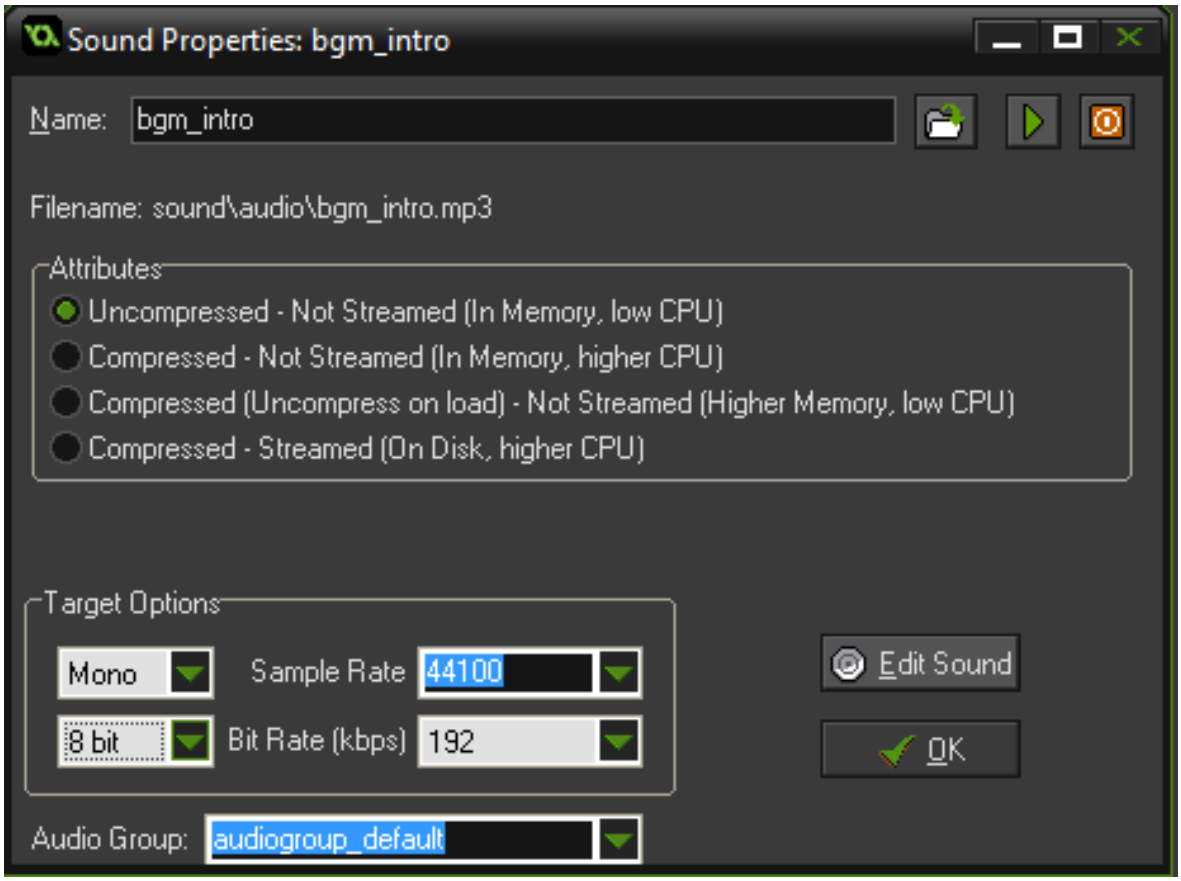

Figura C.7: Inserción de sonidos/músicas en *GameMaker*.

[3]https://en.wikipedia.org/wiki/Bit_rate
[4]https://es.wikipedia.org/wiki/Frecuencia_de_muestreo

C.11. Puntuaciones locales

Como hemos comentado en otras secciones, *GameMaker* nos facilita tanto un vector para almacenar las mejores puntuaciones, como una variable global donde almacenar la puntuación en curso. Al comienzo de las fases de juego, reiniciamos dicha puntuación a cero, para que comience a contar.

Para poder visualizar por pantalla tanto la puntuación máxima como la puntuación en curso añadiremos en el objeto controlador de fase un evento *Draw*, en el mismo definiremos que el color de las letras sea el verde. Y procederemos a insertar mediante código, haciendo uso del método *draw_text_transformed()*, las puntuaciones en pantalla. Como el evento *Draw* se ejecuta en cada *tick*, siempre se mantendrá actualizada la puntuación en curso.

Cada vez que alcancemos un enemigo en pantalla aumentaremos la puntuación, así pues será necesario añadir en los eventos de colisión con los disparos del jugador el incremento en la puntuación.

Las puntuaciones que nos dará cada enemigo serán las siguientes:

Enemigo	Puntuación
Invasores Rojos	40
Invasores Amarillos	30
Invasores rosas	20
Invasores azules	10
Boosters	60
Alcalde	15 cada disparo
Fluvi	20 cada disparo
SuperMaño	25 cada disparo

Tabla C.1: Tabla de puntuaciones.

C.12. Menú de pausa

Al menú de pausa llegaremos añadiendo una interrupción de la tecla escape y retroceso en pantalla (para tratar el caso de *Android*). Este menú estará formado por un objeto controlador que inicializará todo. Así pues en inicio este controlador realizará lo siguiente:

- Creará un recuadro verde en el cual introducirá el título de "Juego en Pausa" y dos objetos botón "Continuar" y "Salir"
- Indicaremos en el atributo del controlador de fase que el juego está pausado

– Invocaremos al evento de pausa en cada uno de los objetos susceptibles de aparecer en las fases de juego.
– Añadiremos un botón igual que el del menú principal para activar o desactivar el sonido, con la misma lógica que éste.

El desplazamiento entre botones lo realizaremos haciendo uso del controlador anterior definido, añadiendo las interrupciones sobre el mismo, al igual que se hizo con el menú principal. Así pues en el caso de que nos desplacemos con flechas direccionales, necesitaremos conocer mediante un atributo en cual de los dos botones nos encontramos. Una vez implementado, si pulsamos salir de cualquiera de las formas invocaremos a una alarma del controlador que destruirá los botones anteriores pintando unos nuevos y cambiando el título del menú de pausa, pasando a contener la pregunta "¿Está seguro?". De este modo, el usuario tendrá una segunda oportunidad, por si pulsó sin querer. Los dos nuevos botones que crearemos (serán dos nuevos ojbetos) "Confirmar" y "Cancelar"; invocarán por un lado a la alarma que nos lleve al menú de fin del juego y por otro lado a una alarma, que destruirá estos botones y nos llevará al menú de pausa original creando los objetos originales del mismo. El botón "Continuar" del menú de pausa, invocará una alarma que al comenzar su evento, destruirá el controlador de pausa, creando uno nuevo en nuestro caso denominado "resume_pause" cuyo objeto es crear una cuenta atrás y pintar por pantalla dicha cuenta atrás comenzando en 3 y llegando hasta 0, momento en el cual saltará la interrupción que destruirá dicho controlador de regreso invocando a su vez el evento en cada uno de los objetos de la fase para volver a su estado anterior, volviendo a suprimir el estado de juego en pausa en el atributo pertinente del controlador.