Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería de Software

Firework Madness: Aplicando las métodos de un ingeniero de software al desarrollo de videojuegos

---

Rubén Pérez Alonso

Junio 2016

informatika fakultatea
facultad de informática

Agradecimientos

Antes de empezar veo necesario agradecer la ayuda y apoyo recibidos a todos los implicados en el desarrollo de este proyecto y en mi estancia cursando el grado de Ingeniería Informática.

Gracias lo primero y más importante a mi familia por apoyarme estos cinco años que he estado en la universidad, tanto en los buenos como en los malos momentos. Animándome cuando las cosas no iban bien y pidiéndome más cuando no había problemas.

Gracias a mis compañeros de grado, tanto a los que han seguido conmigo estudiando estos años como a los que han tenido que dejarlo, y gracias sobre todo a los que me han aguantado en la vida diaria en mi piso de estudiantes, irse a vivir a una ciudad nueva sin conocer a nadie es duro pero todo ha sido más fácil con amigos como vosotros.

Gracias a los profesores que me han dado clase estos años, incluidos los que más difícil me lo han puesto para acabar el grado. Sin duda estos años me he encontrado con algunos de los mejores docentes que he conocido en toda mi vida e incluso podría decir que al final de la asignatura he considerado a más de uno como un compañero más de la carrera. Comentar como mención especial entre otros a José Ángel Vadillo, cuya actitud con los alumnos, apoyo a éstos y forma de docencia me ha parecido realmente buena y me sorprendió gratamente cuando lo conocí.

Pero si hay que destacar a un profesor que me haya apoyado de entre todos, éste sería José Miguel Blanco. Desde que empezó a darme clase hasta ahora que ha sido mi director de proyecto no ha hecho más que apoyarme en mis decisiones e instarme a dar el 120%. Me empujó a ir de empresa en empresa para encontrar unas prácticas que realmente fuesen de un campo que me gustase y no conformarme con las ofertadas en la universidad y le estaré eternamente agradecido por ello.

Porque gracias a él entré de prácticas en la empresa Nesplora, donde pasé meses trabajando en un campo que realmente me interesa y aprendiendo cosas increíbles todos los días. Donde entré realmente nervioso y fui recibido con una sonrisa y un apoyo constante. Donde al principio no conocía a nadie y cuando me fui me llevé conmigo multitud de amigos y profesionales con los que espero mantener el contacto y poder volver a trabajar en el futuro. De todo corazón gracias Nesplora por la oportunidad, gracias Beñat, gracias Álvaro, gracias Gurutz, gracias a todos en general por todo lo que me habéis enseñado.

Y por último, gracias a mi abuela, que lo último que me pidió fue "Termina bien el último año". Que sepas que todo ha terminado bien y todos te queremos muchísimo y te echamos de menos estés donde estés.

Índice

Índice de Figuras

1

Introducción

Durante este primer capítulo se explican las motivaciones que han dado lugar a la realización de este proyecto y se introduce al lector en el trabajo desarrollado y en la documentación que lo recoge.

1.1. Motivación

En la actualidad, de entre todos los tipos de entretenimiento que hay, la industria del videojuego se ha convertido en la más importante, llegando a movilizar unos ingresos muy superiores al resto de las industrias de este estilo, ya sea cine, televisión o música. Si se quiere más información sobre estos datos se puede leer el artículo "The biggest entertainment markets in the world" [1]

Además, la situación de esta industria en España no para de mejorar. Año a año crece más tanto en empleo como en facturación y se prevé que siga creciendo durante muchos años según los últimos estudios de la Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento, DEV (para más información, consulte el Libro Blanco Del Desarrollo Español de Videojuegos 2015 mediante el enlace que puede encontrar en la sección Bibliografía-Libros).

Sin embargo, esta industria es relativamente joven en comparación al resto y su rápido ascenso no ha estado acompañado de una enseñanza acorde. Esto se traduce en la necesidad de auto-aprendizaje, lo cual lleva a un desarrollo desestructurado y una falta de modelos de desarrollo en los que basarse.

Es en ese punto donde nace este proyecto, intentando unir el aprendizaje de herramientas desconocidas con la aplicación de las técnicas de desarrollo de software impartidas a lo largo de la carrera, adaptando éstas a este tipo de desarrollos y creando así un modelo estructurado de desarrollo de videojuegos y una arquitectura de software apropiada para estos tipo de trabajos, que sea reutilizable tanto en videojuegos simples como éste, como en otros más complejos.

Esto no se limita al desarrollo de un modelo de software, si no al uso de dinámicas de trabajo, gestión de tiempo, diseño de interfaces, etc. en el campo del desarrollo de videojuegos.

1.2. ¿Qué es Firework Madness?

Como resultado de este proyecto nace la idea de Firework Madness, un videojuego para plataformas móviles Android creado haciendo uso de las técnicas anteriormente mencionadas. A lo largo de esta memoria se describe como ha sido el proceso de desarrollo de este producto, desde gestión de tiempo y recursos hasta implementación de código y toma de decisiones, y cómo se han aplicado esas técnicas, de forma que pueda servir de base para futuros desarrollos de este tipo.

[1] http://businesstech.co.za/news/lifestyle/88472/the-biggest-entertainment-markets-in-the-world/

2

Objetivo del proyecto

En este capítulo se explica el objetivo del proyecto y su alcance total, aclarando las exclusiones.

2.1. Objetivo

El objetivo del proyecto es utilizar el motor gráfico Unity para desarrollar el videojuego "Firework Madness" para dispositivos móviles Android y aplicar durante el desarrollo técnicas de desarrollo de software, diseño de interfaces, modelado de datos, diseño de casos de uso... propias de un ingeniero de software.

Dado que el aprendizaje de la herramienta será simultáneo al desarrollo del videojuego, no se busca realizar un videojuego impresionante a niveles técnicos y jugables, sino un videojuego simple que sea utilizable y haya sido realizado aplicando las técnicas comentadas, para que posteriormente pueda ser utilizado como base para el desarrollo de productos más complejos de forma ordenada.

2.2. Alcance

El alcance del proyecto está enfocado a proporcionar una base reutilizable sobre la que poder desarrollar videojuegos de una forma ordenada y estructurada, sin importar la complejidad de éste.

En el alcance del producto se incluyen los siguientes puntos:

- Desarrollo de un videojuego simple mediante el motor de desarrollo de videojuegos multiplataforma Unity, completamente funcional e instalable en cualquier dispositivo móvil Android.
- Un manual de usuario que recoja las instrucciones necesarias tanto para la instalación del juego Firework Madness como para el uso de éste.
- Aplicación de técnicas de desarrollo software en el ámbito del desarrollo de videojuegos profesional.

Además se pueden diferenciar claramente varias fases a lo largo del desarrollo del proyecto:

- Fase de inicio. Esta fase se centra en la planificación del proyecto (gestión de tiempos, recursos y riesgos, definición de hitos, etc.) y en el planteamiento de las bases del videojuego que se quiere crear, características, funcionalidades, estilo...
- Fase de implementación. Es la fase en la que se produce el desarrollo del videojuego, las distintas pruebas sobre éste y el aprendizaje de las distintas herramientas necesarias.
- Fase de pruebas con usuarios. Durante esta fase tiene lugar la aplicación de un plan de entrevistas con distintos tipos de usuarios mientras hacen uso del producto en busca de *feedback* para posibles mejoras.

- Fase de seguimiento y control. Esta fase se produce a lo largo de toda la vida del proyecto y se basa en el seguimiento diario del progreso del proyecto en relación a la planificación planteada en la primera fase.
- Fase de cierre.

2.2.1. Exclusiones

Pese a ser considerados al inicio del proyecto, no han sido incluidos en el alcance ninguno de los siguientes puntos:

- Desarrollo del videojuego desde el punto de vista de gerente de una empresa del sector, incluyendo el desarrollo de distintos análisis de viabilidad de la empresa y tomando el papel de director de empresa en vez de ingeniero desarrollador de software.
- Creación del videojuego desde el punto de vista de director de un equipo, incluyendo la formación y dirección de éste.

3

Herramientas y tecnologías

A continuación se describen las herramientas y tecnologías usadas en el desarrollo del proyecto incluyendo entornos de desarrollo, lenguajes de programación y herramientas complementarias de utilidad en desarrollos de este tipo.

3.1. Herramientas usadas

En este apartado se mencionan las principales herramientas usadas a lo largo del proyecto. Éstas son el motor gráfico Unity y el lenguaje de programación C#.

Unity (https://unity3d.com/es)

Figura 1. Logo de Unity3D

Unity es el motor de desarrollo de videojuegos elegido para el desarrollo del producto. Esta elección viene motivada por tres factores importantes: su coste, su profundidad y su facilidad de aprendizaje.

Pese a que gran cantidad de herramientas de este tipo son gratuitas, muchas de ellas tienen una licencia gratuita limitada y pasado un tiempo o si el producto desarrollado se publica, se debe pagar la licencia avanzada, que no es precisamente barata. Unity en cambio permite el uso totalmente gratuito de la herramienta si se es un desarrollador pequeño, y solo obliga a pagar cuando el producto desarrollado ha sido publicado y ha tenido un mínimo de ventas, es decir, cuando el desarrollador realmente se lo puede permitir.

Respecto al segundo apartado, pese a que Unity es una herramienta relativamente nueva (fue lanzado en 2005 pero no adquirió popularidad hasta 2010) está adquiriendo un gran renombre gracias a su constante evolución. Semanalmente los desarrolladores lo actualizan y añaden nuevos elementos que facilitan desarrollos más complejos y avanzados. Por poner un ejemplo, ya cuentan con un profundo sistema de desarrollo en *VR* (*Virtual Reality*), una tecnología realmente reciente.

Por último, su página web cuenta con gran cantidad de guías de inicio en el desarrollo en Unity y además hay una gran comunidad de usuarios dando apoyo a los nuevos desarrolladores en internet. Adicionalmente, la propia forma en que está concebido Unity facilita el desarrollo, dando la capacidad de realizar gráficamente muchas de las tareas gracias a su interfaz.

C#

Figura 2. Logo de C#

Unity permite utilizar dos lenguajes de programación, C# o Unityscript, que está basado en Javascript. La elección de usar C# se debe a que gran parte de la comunidad lo prefiere y tiene todos sus tutoriales y guías explicados con éste, lo que ha acabado extendiendo el uso de C# sobre Unityscript para aprender. Sin embargo, no deja de ser una elección personal y la API de Unity es igual para cualquiera de los dos lenguajes, por lo que no hay ningún inconveniente en el caso de elegir Unityscript.

MonoDevelop (http://www.monodevelop.com/)

Figura 3. Logo de MonoDevelop©

El entorno de desarrollo utilizado para la implementación del código ha sido MonoDevelop, que es el entorno por defecto que suministra Unity para trabajar.

Además MonoDevelop© es *open source*, soporta varios lenguajes de programación como C#, C, C++ o Visual Basic .NET y cuenta con una herramienta de auto-rellenado de código para C# lo cual ha acelerado en gran medida el desarrollo de la aplicación.

Como alternativa, Unity también permite establecer Visual Studio© como *IDE* (*Integrated Development Enviroment*) por defecto. Ésta es una herramienta más completa pero para su uso hace falta una licencia y es más lenta, por lo que para un desarrollo simple como el propuesto MonoDevelop es más adecuado.

3.2. Tecnologías usadas

En cuanto a las tecnologías usadas cabe destacar MonoDevelop, UML y, en especial, la API de Unity.

La API de Unity (http://docs.unity3d.com/ScriptReference/)

La API de Unity supone la base de la programación en Unity, independientemente del lenguaje de programación. Resulta realmente profunda y permite programar cualquier aspecto del videojuego, por lo que su conocimiento es indispensable para el desarrollador.

UML (http://www.uml.org/)

Figura 4. Logo de UML

Unified Modelling Language. Es el lenguaje de modelado software más conocido y usado actualmente, y junto con la herramienta StarUml© (http://staruml.io/) ha sido base para el desarrollo de todos los modelos de software del proyecto, desde los diagramas de casos de uso hasta el esquema entidad-relacion.

Otras tecnologías y herramientas complementarias

Cabe destacar herramientas complementarias utilizadas para tareas de gestión y documentación, como Microsoft Office Word©, Microsoft Office PowerPoint©, Google Drive© y Google Calendar©, o para el dibujado de los elementos del juego como Adobe Photoshop© o Macromedia FreeHand©.

4

Caracterización del producto y casos de uso

En este capítulo se habla de las características y funcionalidades principales con las que cuenta el producto.

4.1. Características de Firework Madness

Firework Madness es un videojuego que se enmarca en el género "puzzle". Actualmente no cuenta con funcionalidades sociales de ningún tipo, por lo que es un juego sin necesidad de conexión a internet.

Al empezar una partida se creará de forma aleatoria un tablero de juego con cohetes de distintos colores. La cantidad de cohetes que rellenarán a ese tablero al empezar dependerá de la dificultad elegida por el usuario al principio del juego. Una vez rellenado aparecerá un conjunto de dos cohetes de distintos colores unidos en la parte superior de la pantalla.

El objetivo del usuario es mover y rotar esa pieza mientras desciende hasta que no pueda bajar más y esta forme parte del tablero de juego. Si al colocarla el jugador consigue que coincidan tres o más cohetes del mismo color de forma consecutiva, estos explotarán en una serie de fuegos artificiales, dando puntos al jugador. Tras esto, nuevamente aparecerá otra pieza y se repetirá el proceso.

De esta forma, el objetivo del jugador es hacer uso de estas mecánicas para conseguir la mayor puntuación posible y superar su propio record.

La partida acabará cuando el tablero de juego se quede sin cohetes para jugar, cuando éste se llene demasiado y llegue al extremo superior de la tabla (marcado por una línea roja) o cuando acabe el tiempo límite de juego.

La idea principal con la que nace Firework Madness es la de que las partidas sean cortas y rápidas, enfocadas a que el usuario no tenga la necesidad de jugar constantemente para ver progresos, si no que el factor de auto-superación del usuario, hablando en este caso de la superación de su anterior record en el juego, sea el que le haga jugar más.

4.2. Jerarquía de actores y diagrama de casos de uso

En este producto, como en la mayor parte de los productos de este tipo, se distingue un solo tipo de actor, el usuario.

Sin embargo, y aunque no constituye un usuario de por sí, cabe destacar la *inteligencia* como componente esencial en este tipo de desarrollos, ya que contiene gran parte de la jugabilidad programada.

Figura 5. Actor "usuario" para los casos de uso

Figura 6. Actor "Inteligencia" para los casos de uso

El diagrama de casos de uso para cada uno de los actores es el siguiente:

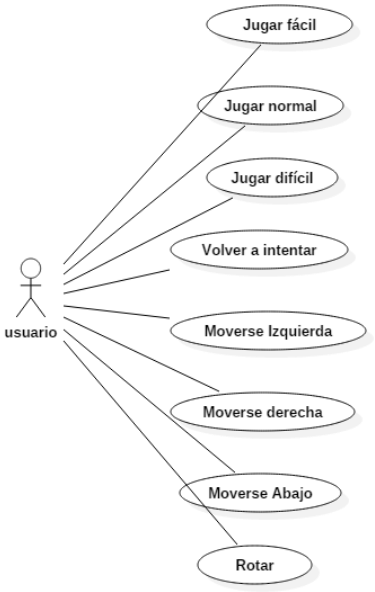


Figura 7. Casos de uso para el actor "usuario"

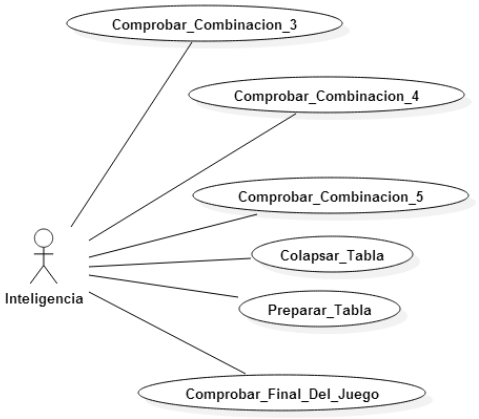


Figura 8. Casos de uso para el actor "Inteligencia"

4.3. Casos de uso extendidos

Nombre: Jugar fácil

Descripción:
El usuario en la escena de introducción elige jugar en modo fácil

Actores:
Usuario

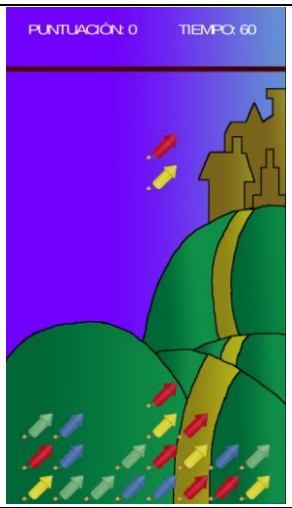
Precondiciones:
Ninguna

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El usuario pulsa el botón “FÁCIL” en el menú de introducción.
2. El sistema carga el juego y establece la dificultad fácil.

Postcondiciones:
Ninguna



Nombre: Jugar normal

Descripción:

El usuario en la escena de introducción elige jugar en modo normal

Actores:

Usuario

Precondiciones:

Ninguna

Requisitos no funcionales:

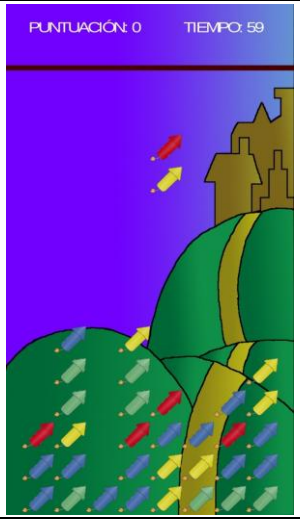
Ninguno

Flujo de eventos:

1. El usuario pulsa el botón “NORMAL” en el menú de introducción.
2. El sistema carga el juego y establece la dificultad normal.

Postcondiciones:

Ninguna



Nombre: Jugar difícil

Descripción:

El usuario en la escena de introducción elige jugar en modo difícil

Actores:

Usuario

Precondiciones:

Ninguna

Requisitos no funcionales:

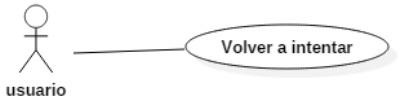
Ninguno

Flujo de eventos:

1. El usuario pulsa el botón “DIFÍCIL” en el menú de introducción.
2. El sistema carga el juego y establece la dificultad difícil.

Postcondiciones:

Ninguna



Nombre: Volver a intentar

Descripción:

El usuario en la escena de puntuación final decide seguir jugando y pulsar "Volver a intentar"

Actores:

Usuario

Precondiciones:

Ninguna

Requisitos no funcionales:

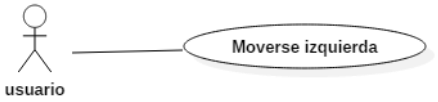
Ninguno

Flujo de eventos:

1. El usuario pulsa el botón "Volver a intentar" en el menú de puntuación final.
2. El sistema carga el menú inicial y le da al usuario la opción de volver a elegir dificultad para jugar.

Postcondiciones:

Ninguna



Nombre: Moverse izquierda

Descripción:

El usuario mientras juega pulsa a la izquierda de la pieza para moverla una unidad a la izquierda.

Actores:

Usuario

Precondiciones:

Ninguna

Requisitos no funcionales:

Ninguno

Flujo de eventos:

1. El usuario pulsa en la mitad superior de la pantalla a la izquierda de la pieza.
2. El sistema comprueba si el movimiento es posible, es decir si al mover la pieza ésta no se sale de los límites de juego o si no hay ningún cohete a la izquierda.
3. En caso afirmativo, el sistema mueve la pieza una unidad a la izquierda. Si no, la pieza no se mueve de su posición.

Postcondiciones:

Ninguna

Nombre: Moverse derecha

Descripción:
El usuario mientras juega pulsa a la derecha de la pieza para moverla una unidad a la derecha.

Actores:
Usuario

Precondiciones:
Ninguna

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El usuario pulsa en la mitad superior de la pantalla a la derecha de la pieza.
2. El sistema comprueba si el movimiento es posible, es decir, si al mover la pieza ésta no se sale de los límites de juego o no hay ningún cohete a la derecha.
3. En caso afirmativo, el sistema mueva la pieza una unidad a la derecha. Si no, la pieza no se mueve de su posición.

Postcondiciones:
Ninguna

Nombre: Moverse abajo

Descripción:
El usuario mientras juega pulsa en la mitad inferior de la pantalla del dispositivo para mover la una unidad hacia abajo.

Actores:
Usuario

Precondiciones:
Ninguna

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El usuario pulsa debajo de la pieza en la pantalla del dispositivo.
2. El sistema comprueba si el movimiento es posible, es decir si al mover la pieza ésta no se sale de los límites de juego o no hay ningún cohete debajo de su posición.
3. En caso afirmativo, el sistema mueve la pieza una unidad hacia abajo y permite al usuario realizar un nuevo movimiento. Si no, el sistema fija la pieza en la tabla e instancia una nueva pieza en la parte superior central de la tabla que será el nuevo bloque del jugador.

Postcondiciones:
Ninguna

Nombre: Volver a intentar

Descripción:

El usuario mientras juega pulsa encima de la pieza en la pantalla de su dispositivo para rotarla 90º a la derecha.

Actores:

Usuario

Precondiciones:

Ninguna

Requisitos no funcionales:

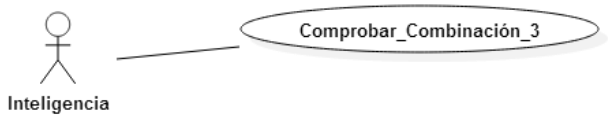
Ninguno

Flujo de eventos:

1. El usuario pulsa encima de la pieza en la pantalla del dispositivo.
2. El sistema comprueba si el movimiento es posible, es decir si al rotar la pieza ésta no se sale de los límites de juego o no pega con ningún cohete.
3. En caso afirmativo, el sistema rota la pieza 90º respecto a su posición actual y permite al usuario realizar un nuevo movimiento. Si no, el sistema no permite la rotación y devuelve el control al usuario para que realice un nuevo movimiento.

Postcondiciones:

Ninguna



Nombre: Comprobar_Combinación_3

Descripción:
El sistema comprueba si en la tabla se produce alguna combinación de tres cohetes del mismo color.

Actores:
Inteligencia

Precondiciones:
La tabla se ha colapsado y ya se ha comprobado si ha habido alguna combinación de cinco o cuatro cohetes del mismo color.

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El sistema comprueba si en la tabla hay alguna combinación de tres cohetes del mismo tipo vertical u horizontalmente.
2. En caso afirmativo, si alguno de los cohetes es de tipo bonus_4, se activa el sistema de partículas contenido en cada uno de los cohetes de la fila en la que se situé ese bonus, creando los fuegos artificiales, se destruyen los cohetes y suma los puntos relativos a una cada cohete dependiendo el tipo de éste. Si algún cohete fuese de tipo bonus_5 además ocurriría lo mismo en la fila inmediatamente inferior ó en la inmediatamente superior si el cohete se encontrase en la fila inferior de la tabla. Si los cohetes fuesen normales, solo se destruirían los cohetes de la combinación.

Postcondiciones:
Ninguna

Nombre: Comprobar_Combinación_4

Descripción:
El sistema comprueba si en la tabla se produce alguna combinación de cuatro cohetes del mismo color.

Actores:
Inteligencia

Precondiciones:
La tabla se ha colapsado y ya se ha comprobado si ha habido alguna combinación de cinco cohetes del mismo color.

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El sistema comprueba si en la tabla hay alguna combinación de cuatro cohetes del mismo tipo vertical u horizontalmente.
2. En caso afirmativo, si alguno de los cohetes es de tipo bonus_4, se activa el sistema de partículas contenido en cada uno de los cohetes de la fila en la que se situé ese bonus, creando los fuegos artificiales, se destruyen los cohetes y suma los puntos relativos a una cada cohete dependiendo el tipo de éste. Si algún cohete fuese de tipo bonus_5 además ocurriría lo mismo en la fila inmediatamente inferior ó en la inmediatamente superior si el cohete se encontrase en la fila inferior de la tabla. Si los cohetes fuesen normales, solo se destruirían los cohetes de la combinación.
3. Por último, en caso afirmativo, el sistema instancia un nuevo cohete del mismo color que la combinación realizada pero de tipo bonus_4 en la primera posición que había en la combinación.

Postcondiciones:
Ninguna

Nombre: Comprobar_Combinación_5

Descripción:

El sistema comprueba si en la tabla se produce alguna combinación de cinco cohetes del mismo color.

Actores:

Inteligencia

Precondiciones:

La tabla se ha colapsado y ya se ha comprobado si ha habido alguna combinación de cinco cohetes del mismo color.

Requisitos no funcionales:

Ninguno

Flujo de eventos:

1. El sistema comprueba si en la tabla hay alguna combinación de cinco cohetes del mismo tipo vertical u horizontalmente.
2. En caso afirmativo, si alguno de los cohetes es de tipo bonus_4, se activa el sistema de partículas contenido en cada uno de los cohetes de la fila en la que se situé ese bonus, creando los fuegos artificiales, se destruyen los cohetes y suma los puntos relativos a una cada cohete dependiendo el tipo de éste. Si algún cohete fuese de tipo bonus_5 además ocurriría lo mismo en la fila inmediatamente inferior ó en la inmediatamente superior si el cohete se encontrase en la fila inferior de la tabla. Si los cohetes fuesen normales, solo se destruirían los cohetes de la combinación.
3. Por último, en caso afirmativo, el sistema instancia un nuevo cohete del mismo color que la combinación realizada pero de tipo bonus_5 en la primera posición que había en la combinación.

Postcondiciones:

Ninguna

Nombre: Colapsar_Tabla

Descripción:
El sistema colapsa la tabla de forma que no queden huecos entre cohetes.

Actores:
Inteligencia

Precondiciones:
La tabla ya se ha preparado correctamente al inicio del juego.

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El sistema recorre toda la tabla en busca de huecos.
2. Si hubiese algún cohete en la misma columna que el hueco pero en una fila superior, este cohete sería colocado en la posición del hueco, dejando su posición libre.
3. Si no hay cohetes encima, eso significa que esa columna se ha colapsado correctamente.

Postcondiciones:
Ninguna

Nombre: Preparar_Tabla

Descripción:
El sistema al inicio del juego, crea la tabla y al rellena de cohetes.

Actores:
Inteligencia

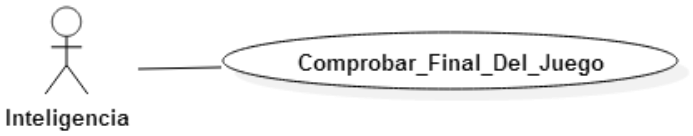
Precondiciones:
Ninguna

Requisitos no funcionales:
Ninguno

Flujo de eventos:

1. El sistema crea la tabla con ocho columnas de ancho y doce filas de altura.
2. Una vez creada la tabla, el sistema la rellena aleatoriamente hasta un número determinado de filas, dependiendo de la dificultad elegida por el usuario en el menú. Si ha elegido fácil, se rellenan cuatro filas, en modo normal, se rellenan siete y en modo difícil diez.
3. El sistema comprueba que al rellenar la tabla, no la ha rellenado con combinaciones, en cuyo caso destruye uno de los cohetes de la combinación y lo sustituye por otro de otro color aleatorio.

Postcondiciones:
Ninguna



Nombre: Comprobar_Final_Del_Juego

Descripción:

El sistema comprueba que no ha terminado el juego.

Actores:

Inteligencia

Precondiciones:

Ninguna

Requisitos no funcionales:

Ninguno

Flujo de eventos:

1. El sistema comprueba las tres condiciones que provocan el final del juego, si la tabla no tiene cohetes, si se ha superado el límite superior de la tabla o si el tiempo ha terminado.
2. En caso de que alguna se cumpla, el sistema guarda la puntuación lograda y carga la escena de puntuaciones, donde si ha conseguido superar el record anterior se le dará la enhorabuena al jugador y se establecerá esa puntuación como nuevo record. Si no lo ha conseguido superar el sistema mostrará un mensaje notificándolo y mostrará al jugador el record anterior.

Postcondiciones:

Ninguna

5

Arquitectura software

Actualmente el producto sigue una arquitectura "compacta", asociada a su instalación y ejecución en modo local, siguiendo el modelo MVC. Sin embargo, las características especiales que tiene el desarrollo de videojuegos han obligado a la adaptación de la arquitectura, pasando a ser un modelo AMVCC (*Application Model View Controller Component*).

5.1. El modelo AMVCC

En el desarrollo de videojuegos sobre la plataforma Unity ocurre que seguir el modelo MVC resulta insuficiente, y da lugar a los siguientes problemas:

- Las referencias a clases MVC se dispersan fácilmente por el código, pudiendo causar errores y *bugs*.
- Algunos elementos encapsulan funcionalidades que pueden ser reutilizadas frecuentemente y que no encajan en ninguna de las categorías del MVC.

Para solucionarlo surgen dos nuevas categorías, Application y Component:

- Application es un punto de entrada único por el que todas las instancias MVC pueden ser alcanzadas y recogidas.
- Component es un script que contiene las funciones reutilizables.

  De esta forma surge el modelo AMVCC (Application, MVC, Component):

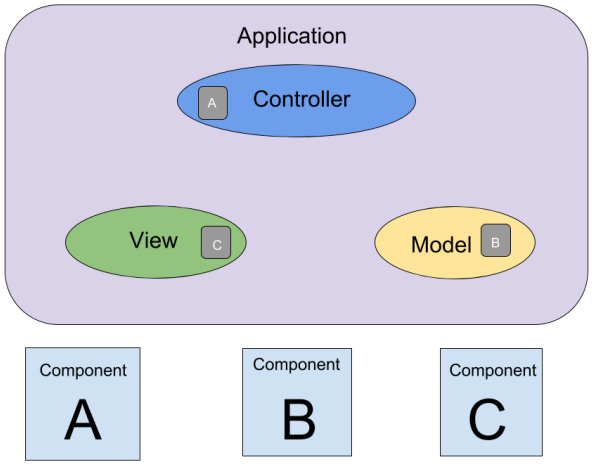


Figura 9. El modelo AMVCC

El modelo AMVCC no se implementa en todos los elementos de la escena y del juego, solo en aquellos que conformen la jugabilidad programada, y en nuestro caso ese elemento es el objeto "Bloque", que controla el jugador, y todos los componentes que forman a éste.

5.1.1. Application

El elemento Application recoge el resto de instancias del modelo MVC, conformando el punto raíz desde el que acceder al resto de funcionalidades.

En el producto, el objeto *Bloque* se encarga de la función de Application dentro del modelo AMVCC y contiene el script *BloqueApp*, que es el encargado de recoger todas las instancias del modelo MVC usado (*BloqueModelo*, *BloqueVista* y *BloqueControlador*).

5.1.2. Model

El elemento Model es el encargado de almacenar todos los datos relativos al jugador y todas las funciones que puedan modificar éstos.

El objeto *BloqueModelo* recoge todos los datos que necesita el bloque y todas las funciones de movimiento del jugador, que se ejecutarán cuando *BloqueController* lo notifique.

5.1.3. View

El elemento View es el que constituye la parte visual del Bloque. Su función es acceder a los datos del Model que haya que visualizar en pantalla, los cohetes y su posición en nuestro caso, y mostrarlos.

Resulta ser realmente corto, dado que Unity es capaz de instanciar un elemento en la escena mediante una sola llamada a la función *Instantiate*. Por ejemplo en nuestro caso solo se realizan dos llamadas a la función instantiate con los cohetes del bloque y la posición que indique el Model como parámetros.

5.1.4. Controller

El elemento Controller es el encargado de recibir los *Inputs* del dispositivo móvil, es decir las pulsaciones en la pantalla, analizar éstos y notificar la acción a realizar al Model. En este caso, todos los Inputs que se analizan dan como resultado la notificación de la necesidad de realizar un movimiento del bloque y el Model será el encargado de realizar ese movimiento si fuese posible.

5.1.5. Component

En este caso concreto no ha sido necesario el uso de un elemento Component, dado que no hay funciones complementarias que no sea posible encuadrar en alguno de los elementos anteriormente descritos o sean usadas en varios de ellos. Sin embargo, cabe destacar la utilidad de estos elementos en desarrollos más complejos que el realizado.

5.2. La arquitectura en el futuro

Los últimos éxitos en el campo de los videojuegos para plataformas móviles han demostrado que para captar al público hace falta un gran componente social.

Ya sea cooperando entre usuarios o compitiendo contra ellos el factor social resulta altamente adictivo. Dos ejemplos claros pueden ser el famosos "Clash of Clans©" donde usuarios de todo el mundo pueden unirse y cooperar para intentar ser el mejor clan de guerreros del ranking mundial o el ya clásico "Candy Crush©", donde ,en vez de cooperar, los usuarios compiten con sus amigos de facebook para ver quién es el que mejor record consigue en determinadas pantallas o el que más avanza en la historia del juego.

Teniendo en cuenta esto, para adaptarse a esta tendencia en un futuro, Firework Madness implementará un ranking online donde competir contra otros jugadores. Esto conllevará un cambio en la arquitectura del software, pasando de ser únicamente local a convertirse en un producto con arquitectura cliente-servidor.

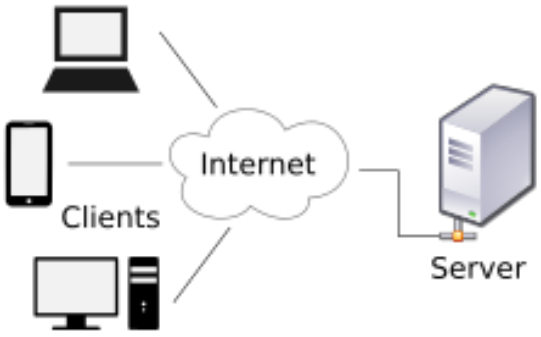


Figura 10. Futura arquitectura de Firework Madness

De esta forma, cada dispositivo móvil de cada usuario será un cliente, que se conectará a un servidor vía internet para recibir los recursos necesarios, en este caso el ranking mundial de usuarios del juego.

6

Diseño de interfaz de usuario y modelo de datos

El diseño de interfaz de usuario supone una parte esencial en cualquier desarrollo software, y en este caso con mayor importancia incluso, dado que es indispensable que la interacción entre el sistema y el usuario sea perfecta. El modelo de datos contiene las distintas entidades existentes en la aplicación y como se relacionan entre sí.

6.1. El diseño de interfaces en videojuegos

Las bases de diseño de interfaces de usuario deben ser adaptadas para que sean útiles en este sector. Así pues, reglas como dar la capacidad al usuario de interrumpir su tarea o deshacer sus acciones, que son básicas en aplicaciones de escritorio normales, resultan imposibles de ser implementadas en determinados tipos de videojuegos.

Nuestro caso es un ejemplo de esto. En ningún momento se le debe dar al usuario el control sobre pausar el juego o deshacer las acciones realizadas, ya que iría en contra del objetivo del juego y le proporcionaría una ventaja injusta.

Sin embargo, muchas otras reglas base si son utilizables, como puede ser la de proporcionar distintos niveles de uso del producto para usuarios con distintos niveles de experiencia (en nuestro caso distintos niveles de dificultad), permitir al usuario una cómoda navegación o hacer transparente la interfaz del usuario. En definitiva, hacer sentir al usuario que tiene control sobre el sistema.

Una de las reglas básicas del diseño de interfaces es la de mantener debidamente informado al usuario del funcionamiento del producto y su control. Sin embargo, al ser éste un producto en desarrollo todavía queda pendiente la implementación de un tutorial dentro del juego antes de permitir al usuario jugar. Para solucionar este problema, se ha creado un manual de usuario de la fase "alpha" del producto.

6.2. Manual de instrucciones de Firework Madness

Introducción

Una vez que entre en Firework Madness, lo primero que se le presenta al usuario es el menú de introducción y el título del juego.

Figura 11. Pantalla de introducción

Desde aquí el usuario puede elegir en que dificultad desea jugar.

Inicio de la partida

Tras elegir el modo, la partida empezará. Al usuario se le presenta un tablero con filas de cohetes de distintos colores, un indicador de la puntuación que está obteniendo y un indicador del tiempo restante de partida.

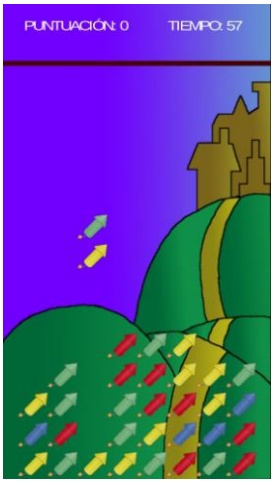


Figura 12. Jugando a Firework Madness

Objetivo

El objetivo es simple, el usuario deberá juntar cohetes del mismo tipo para hacerlos explotar, crear fuegos artificiales y ganar puntos. Para ello controlará distintos bloques de dos cohetes que irán apareciendo y cayendo por la pantalla. Una vez que se han posado en la tabla, si el usuario ha conseguido juntar tres o más cohetes del mismo color los hará explotar.

Movimientos

Antes de explicar los movimientos hay que destacar que el control está basado en pulsaciones en la pantalla, es decir, que mantener pulsada la pantalla o desplazar el dedo no será reconocido como múltiples movimientos simultáneos.

El usuario puede realizar cuatro movimientos distintos con la pieza que esté cayendo en ese momento: moverse a la izquierda, moverse a la derecha, bajar y rotar la pieza.

Para moverse a la izquierda bastará con pulsar a la izquierda de la pieza en los tres cuartos superiores de la pantalla del dispositivo. De esta forma el bloque se desplazará una unidad a la izquierda.

Para moverse a la derecha la pulsación es la misma solo que a la derecha de la pieza.

Automáticamente la pieza baja con el tiempo, pero en caso de querer bajar la pieza más rápido el usuario deberá pulsar en el cuarto inferior de la pantalla. De esta forma la pieza bajará una unidad por pulsación realizada.

Por último, cabe la posibilidad de rotar la pieza 90º a la derecha pulsando encima de ella.

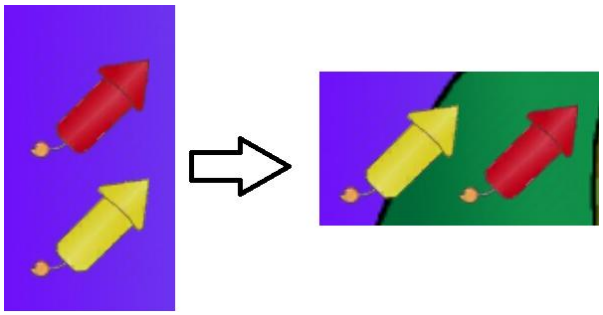


Figura 13. Resultado de la rotación de la pieza

Mecánicas

Mediante los movimientos explicados, el usuario colocará la pieza en el tablero intentando realizar combinaciones de más de dos cohetes del mismo tipo, para que éstas exploten, creando fuegos artificiales y dándole puntos.

Además si la combinación realizada ha sido de cuatro o cinco cohetes, se generará un cohete especial del tipo explotado. Por ejemplo si se ha realizado una combinación de cuatro cohetes rojos se creará un cohete rojo con rallas blancas, que si se combina hará explotar toda la fila en la que se encuentre y si ha sido una combinación de cinco cohetes, se creará un cohete rojo con estrellas blancas, que si se combina hará explotar toda su fila y la inferior.

De esta forma, al usuario se le recompensa si intenta realizar movimientos más difíciles.

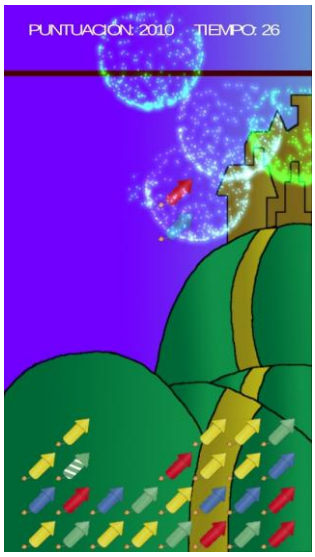


Figura 14. Cohete verde especial rayado resultado de una combinación de 4 cohetes verdes

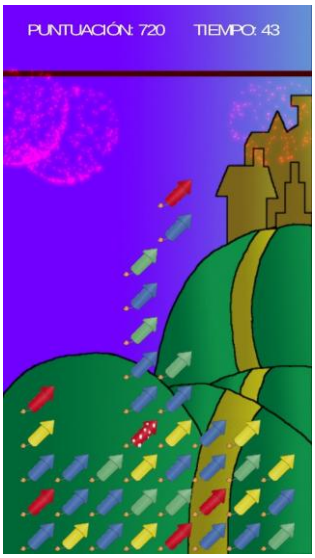


Figura 15. Cohete rojo especial estrellado resultado de una combinación de 5 cohetes rojos.

La partida termina una vez que el indicador de tiempo llega a cero o si el tablero se ha llenado de cohetes rebasando el límite rojo superior o si el usuario ha conseguido que no queden piezas en el tablero. Tras esto al usuario se le lleva a la pantalla de puntuación final.

Puntuación final

En esta pantalla, se abren dos posibilidades, si el usuario ha superado el record establecido en ese dispositivo o si ha fallado en su intento.

Si lo ha conseguido, se le mostrará un mensaje de felicitación por la gesta superada y la nueva puntuación record.

De lo contrario, se le mostrará un mensaje de ánimo y se le recordará cuál es el record establecido.

En ambos casos, se le da al usuario un botón de volver a jugar, llevándolo al menú de inicio para volver a elegir la dificultad de partida deseada.

Figura 16. Pantalla de puntuación final

6.3. Modelo de datos

El esquema entidad/relación que se puede ver en la siguiente figura resume el modelo de datos en el que se basa Firework Madness.

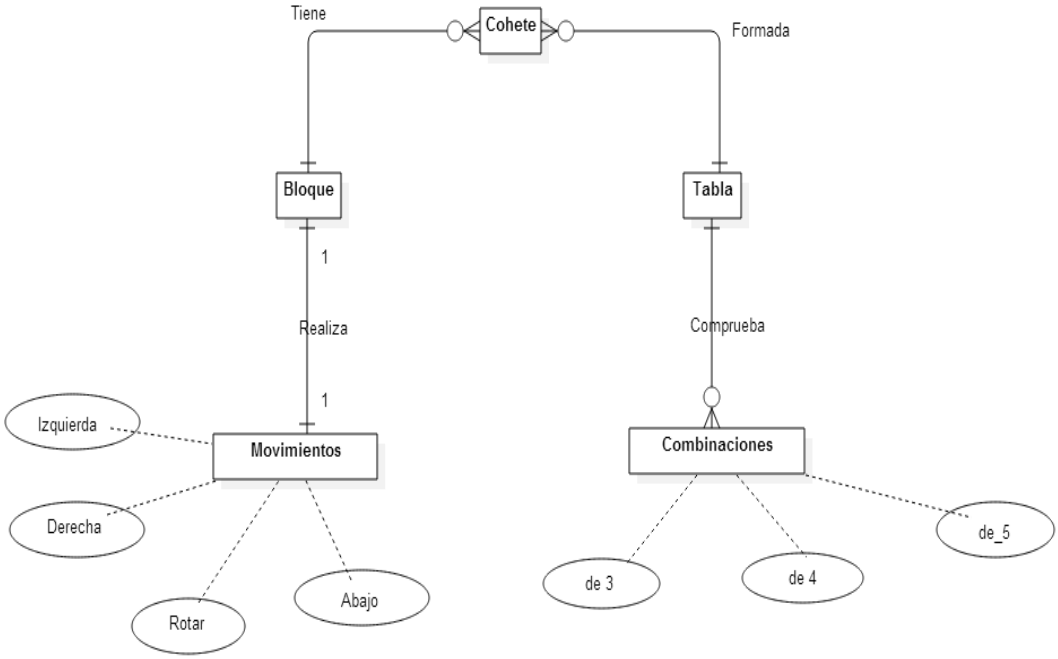


Figura 17. Esquema E/R del modelo de datos de Firework Madness

De esta forma, podemos distinguir tres entidades clave:

- Cohete. Es la entidad base de todo el modelo, ya que sobre ella gira el resto de entidades. Así pues, un "Bloque", esto es la pieza que controla el jugador durante la partida, está formada por dos cohetes, mientras que la "Tabla", es decir el tablero de juego, está formado por una gran cantidad de cohetes, dependiente de la dificultad elegida al empezar la partida.
- Bloque. Es la pieza que controla el usuario durante la partida y puede realizar los movimientos: mover una unidad abajo, izquierda o derecha y rotar la pieza 90º hacia la derecha.
- Tabla. Es el tablero de juego en el que se desarrolla la partida y está constantemente comprobando si ocurre alguna posible combinación de cohetes de un mismo tipo, ya sea una combinación de tres, cuatro o cinco cohetes.

7

Implementación

En este capítulo se trata la implementación de la aplicación desarrollada a lo largo de este proyecto.

7.1. La arquitectura Unity

El motor gráfico Unity sirve de base para el desarrollo del producto y es de obligado cumplimiento explicar su funcionamiento básico de cara a facilitar la comprensión de los siguientes apartados.

7.1.1. La interfaz de Unity

La interfaz de Unity se divide en seis secciones de importancia: la ventana de proyecto, la vista de la escena, la ventana de jerarquía, la ventana del Inspector, la barra de herramientas y la ventana del juego.

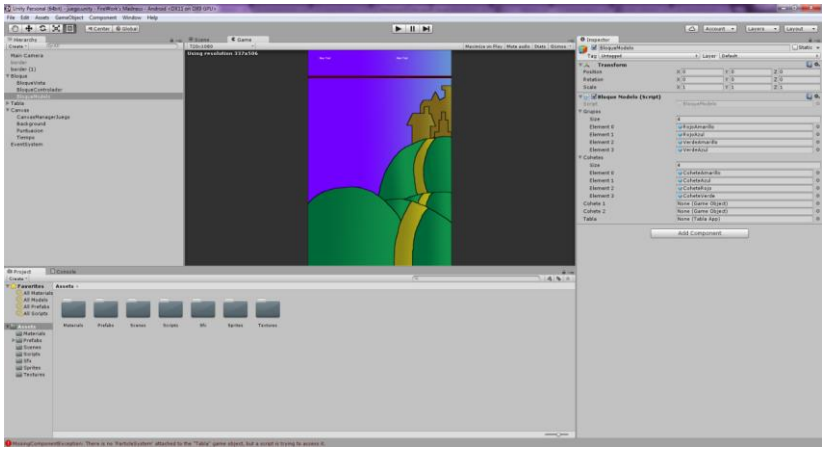


Figura 18. Interfaz general de Unity

La ventana del proyecto

En esta ventana se pueden acceder y gestionar todos los recursos del proyecto.

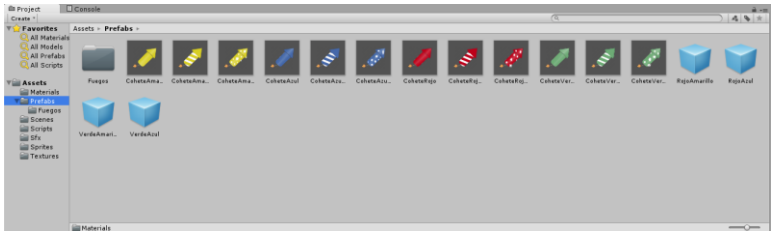


Figura 19. La ventana del proyecto en Unity

La vista de la escena

Es la ventana desde la que se va a poder interactuar con la escena. Con ella se pueden mover y manipular objetos de la escena rápidamente.

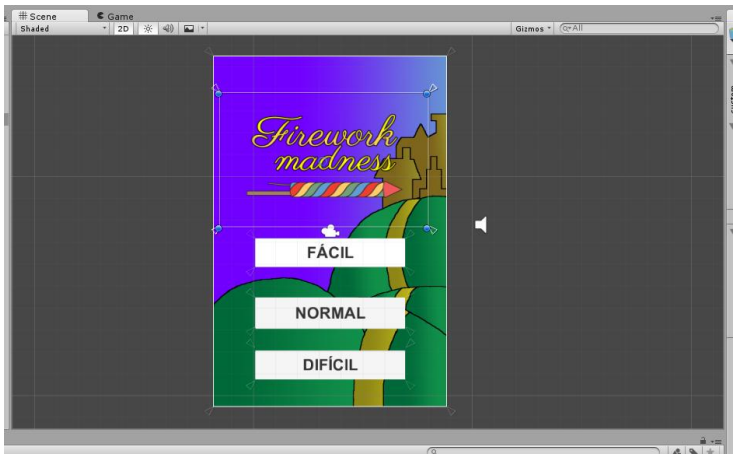


Figura 20. La vista de la escena en Unity

La ventana de la jerarquía

Ésta contiene cada objeto que compone a la escena actual.

Figura 21. La ventana de la jerarquía de Unity

La ventana del Inspector

El Inspector es la ventana desde la que se pueden ver y editar las propiedades de un objeto elegido en la jerarquía o en la vista de la escena. Si el objeto contiene un componente Script personalizado, las variables públicas de este script se mostrarán en el Inspector y pueden ser vistas y editadas fácilmente sin necesidad de código.

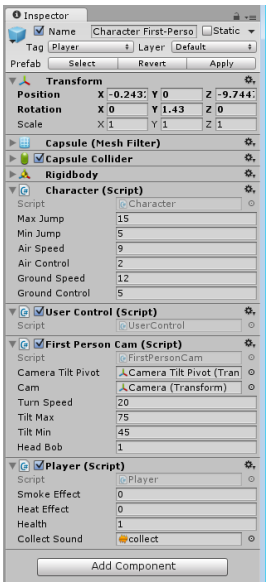


Figura 22. La ventana del Inspector de Unity

La barra de herramientas

La barra de herramientas contiene cinco controles básicos, de los cuales cada uno se relaciona con diferentes partes del editor de la escena, herramientas del *Transform* (conjunto de variables de posicionamiento de un elemento en la escena), tres botones de manejo de la reproducción de la escena, dos opciones de control de servicios online proporcionados por Unity y de gestión de la cuenta con la que se está trabajando, un desplegable que controla que elementos son mostrados en la escena y otro desplegable que controla la disposición de la interfaz gráfica.

Figura 23. La barra de herramientas de Unity

La ventana del juego

Es la ventana en la que se reproduce la representación del juego ya finalizado, visto desde el punto de vista de la cámara o cámaras que forman parte del juego.

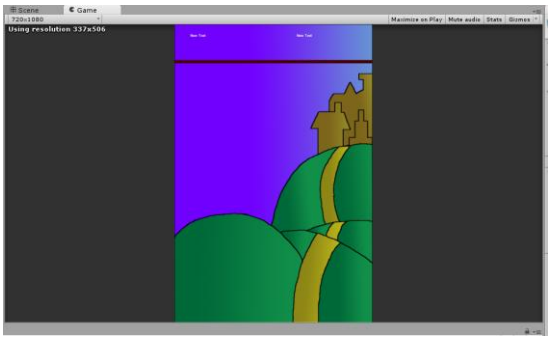


Figura 24. La ventana de juego de Unity

7.1.2. Los componentes de Unity

Una escena puede estar conformada por multitud de elementos distintos: cámaras, luces, objetos, interfaces...

Explicar todos supondría gran cantidad de espacio, así que nos limitaremos a explicar los que se han utilizado en nuestro juego.

Cámara

La cámara conforma el punto de vista del usuario dentro del juego. Cuenta con gran cantidad de opciones personalizables para el desarrollo de todo tipo de juegos distintos. En nuestro caso, como el tipo de juego es 2D con una cámara fija, ha bastado con especificar el tamaño de la cámara deseado y la posición de ésta.

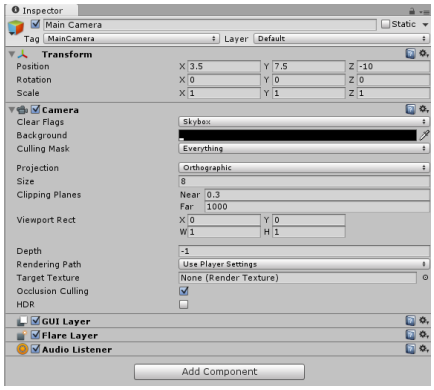


Figura 25. La cámara en Unity

Canvas

El *canvas* es la herramienta de Unity que engloba todo lo relacionado con interfaces, desde imágenes hasta textos o botones. Una de las cualidades destacables del canvas es su capacidad de adaptabilidad a la resolución del dispositivo mediante un sistema de anclajes.

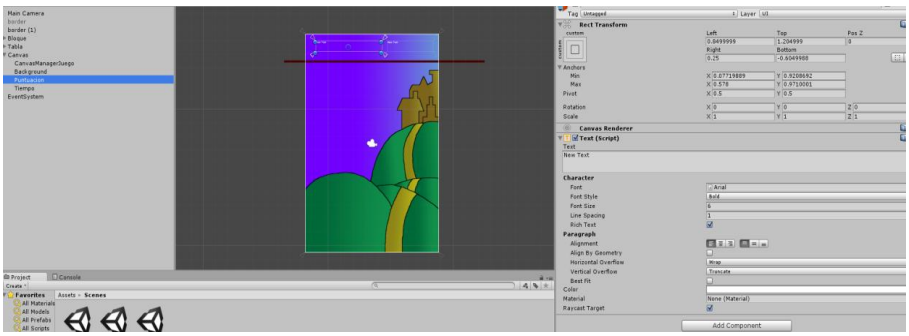


Figura 26. El Canvas en Unity

Objetos

Los objetos son el elemento principal de cualquier juego en Unity. Pueden ser usados para el desarrollo de cualquier elemento de la escena, desde el jugador a un enemigo, decorado o elemento encargado de tareas internas dentro de la inteligencia del juego.

Esto se debe a que a ellos se les pueden añadir gran cantidad de elementos especiales de Unity, sistemas de partículas, controladores de audio, elementos gráficos... y todos éstos pueden ser controlados y programados internamente por los llamados "scripts".

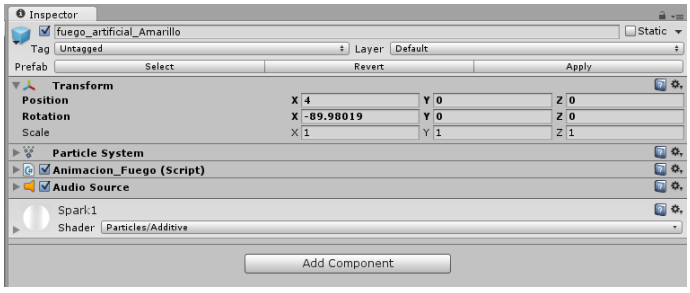


Figura 27. Un objeto en Unity

Prefabs

Es posible que cuando estemos creando una escena reutilicemos un objeto varias veces. Se podría simplemente copiar y pegar ese objeto por la escena, pero aunque son duplicados, éstos se editarán independientemente. Sin embargo, normalmente se quiere que todas las instancias de un objeto tengan las mimas propiedades y cuando éstas se editen es preferible no tener que editar en cada una de ellas.

Para solucionar esto Unity permite hacer un *prefab* que almacene el objeto por completo, incluyendo sus propiedades. El prefab actúa como plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena. Además, aunque cualquier edición hecha en el prefab se reflejará en todas las instancias producidas por él, también se pueden anular componentes y cambiar ajustes de estas instancias individualmente.

Scripts

Son los encargados de controlar el comportamiento del elemento al que estén "atados" y constituyen la parte programable del desarrollo de videojuegos en Unity.

Mediante los *scripts* y el uso de la API propia de Unity puede programarse el comportamiento de todos los elementos que conformen al objeto al que está atado (el controlador del sonido que tenga incorporado el objeto, los datos posicionales del objeto en sí, el funcionamiento del sistema de partículas, etc.).

Unity da la posibilidad de programar estos *scripts* en dos lenguajes, unityscript (basado en javascript) o C# y los complementa con una API propia de gran extensión y con multitud de posibilidades.

Cabe destacar, que en la programación en Unity se pasa de programar un *main* para el script realizado a programar 3 tipos de funciones principales, Awake(), Start() y Update(). Awake() contiene todo lo que se va a realizar antes de empezar a utilizarse el script, Start() lo que se realiza al comienzo del script y Update() lo que ocurre cada *frame* de segundo en el que se ejecuta el script.

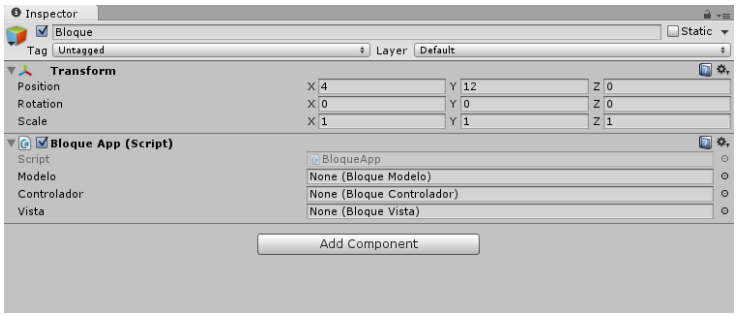


Figura 28. Un objeto en Unity con un script atado a él

Figura 29. Ejemplo de script escrito en C#

7.1.3. Las escenas de Unity

En Unity, el videojuego que se realiza se divide en “escenas” y cada una conforma una parte del juego, por ejemplo la escena de menú, la escena de opciones, la escena de selección de nivel, la escena de juego, etc.

En Firework Madness, tenemos tres escenas: Introducción, juego y puntuación.

Introducción

La escena de introducción es un simple menú con una imagen de fondo, el título del juego y tres botones que permiten al usuario empezar el juego en tres dificultades distintas.

Todos estos componentes conforman la primera interfaz gráfica de usuario, y están realizados mediante la herramienta canvas de Unity, que facilita el desarrollo de este tipo de elementos, permitiendo personalizarlos rápidamente mediante una interfaz propia. Además canvas también permite una implementación rápida del funcionamiento de los botones creados, atando su comportamiento al de un script anteriormente creado por nosotros.

En este caso, el script SceneChanger (atado al objeto GameManager) implementa el paso a la escena de juego con la dificultad indicada en el botón pulsado.

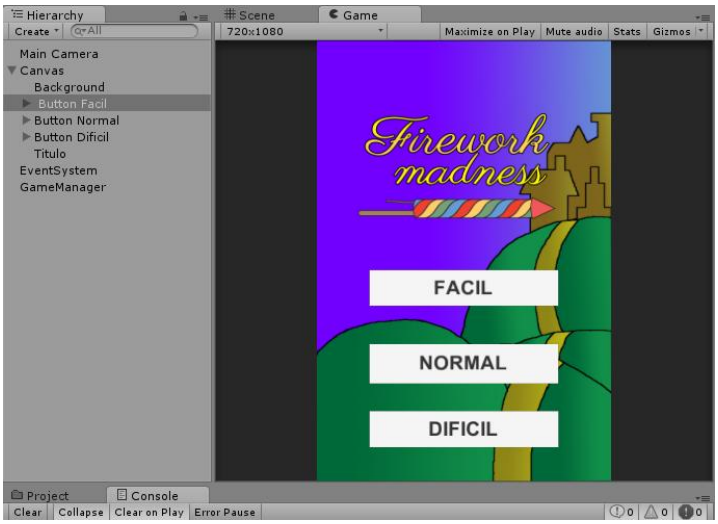


Figura 30. Escena de introducción

Juego

La escena de juego es donde recae la mayor parte del trabajo de implementación realizado.

Entre sus componentes, además del canvas que controla la interfaz de usuario, podemos destacar el objeto "Bloque", referente a la pieza controlada por el usuario y que basa su funcionamiento en el modelo AMVCC.

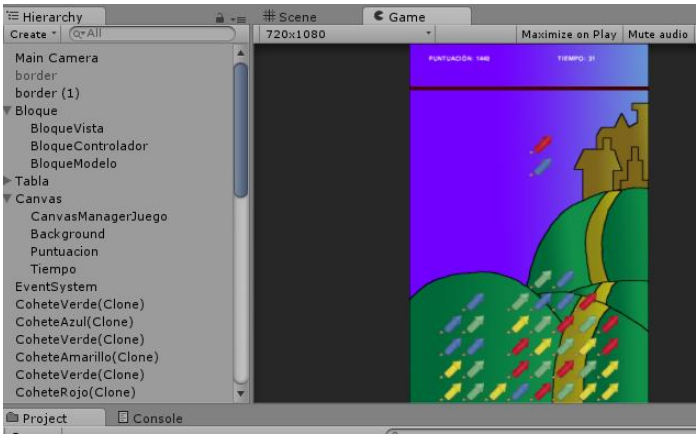


Figura 31. Escena de juego

El objeto "BloqueVista" es el encargado de *renderizar* en pantalla al jugador, es decir, el bloque de cohetes a colocar. Es realmente corto dado que en Unity con una simple llamada a la función Instantiate, ya se instancia en la escena el objeto indicado, en nuestro caso nuestros cohetes, y se visualiza éste.

Puntuación

En la escena de puntuación simplemente se encuentra un canvas con mensajes de felicitación o de ánimo dependiendo de si se ha superado el record anterior. Esta condición está controlada mediante un script *CanvasManagerPuntuación* que se encarga de cambiar el texto dependiendo del caso.

Además se tiene un botón para volver a intentar una nueva partida que lleva al menú de elección de dificultad.

Figura 32. Escena de puntuación

7.2. Implementando Firework Madness

En este apartado se describe en profundidad la implementación de cada una de las escenas que forman Firework Madness y las decisiones de diseño tomadas durante el desarrollo de éstas. Sin embargo primero resulta necesario explicar la implementación de un script que no pertenece a ninguna escena específica, el script *Constants*.

La independencia de este script en cuanto a las escenas del juego es debido a que éste es un script de configuración del juego, es decir, este script guarda todas las variables y datos por defecto con los que se debe empezar cada vez que se abra la aplicación, desde el valor de inicio de *flags* usados en la implementación de códigos usados posteriormente hasta la puntuación que debe darse por cada tipo de combinación.

Esta decisión de separar estas variables en un script independiente se debe a que de esta forma se facilita el cambio de los ajustes del juego en una futura actualización de la jugabilidad. Por ejemplo, si se decide cambiar la puntuación que recibe el jugador al explotar una combinación de tres cohetes al desarrollador le bastará con cambiar el valor de la variable *puntuacionMatch3* en el script Constants.

```
using UnityEngine;
using System.Collections;

public static class Constants {

    public static readonly int  columnas = 8;
    public static readonly int  filas = 13;

    public static string dificultad;

    public static bool spawn;
    public static bool cayendo=true;
    public static bool comprobandoMatchs = false;
    public static bool iniciando=true;
    public static bool hayMatches=true;
    public static bool rellenando = true;
    public static bool tableroListoParaPrimeraComprobacion = false;
    public static bool primerColapso=false;
    public static bool colapsado=false;
    public static bool posado = false;
    public static bool gameOver = false;

    public static int puntuacion=0;
    public static int highScore;
    //public static int tiempoJuego;


    public static float tiempoUltimoDescenso=0.5f;
    public static float tiempoMovimiento = 0.5f;
    //Tiempo hasta el siguiente spawn
    public static float tiempoSiguienteSpawn = 2f;

    public static readonly float duracionAnimacionExplosionNormal = 5f;
    public static readonly float duracionAnimacionMovimiento = 0.05f;
    public static readonly float duracionExplosion = 0.3f;

    public static readonly float esperaAntesDeCheckearPosiblesMatchs = 2f;

    //public static readonly float delayAnimacionOpacidad = 0.05f;

    public static readonly int  minimoMatches = 3;

    public static readonly int  minimoMatchesParaBonus = 4;

    public static readonly int  puntuacionMatch3 = 60;
    public static readonly int  puntuacionMatch4 = 90;
    public static readonly int  puntuacionMatch5 = 120;

    public static readonly int puntuacionSubsecuencias = 1000;
}

```

Figura 33. El script "Constants"

La escena "Introducción"

Esta escena sirve de menú de presentación al juego, y está programada mediante dos scripts que manejan el sonido y el cambio de escenas.

El script *SceneChanger*, atado al objeto *GameManager*, contiene las funciones encargadas de establecer la dificultad del juego dependiendo del botón pulsado y cambiar entre escenas.

De esta forma, por ejemplo la función *ButtonNormal()* establece la dificultad Normal y carga la escena "juego". Contiene además otras funciones que no se usan en esta escena, por lo que se comentarán más tarde.

SceneChanger.cs

```
tion
using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;

public class SceneChanger : MonoBehaviour {

    public void ButtonFacil(){
        Constants.dificultad = "Facil";
        SceneManager.LoadScene ("juego");
    }
    public void ButtonNormal(){
        Constants.dificultad = "Normal";
        SceneManager.LoadScene ("juego");
    }
    public void ButtonDificil(){
        Constants.dificultad = "Dificil";
        SceneManager.LoadScene ("juego");
    }
    public void ButtonVolverAIntentarlo(){
        SceneManager.LoadScene ("inicio");
    }
    public static void FinJuego(){
        SceneManager.LoadScene ("puntuacion");
    }

}
```

Figura 34. El script "SceneChanger"

El script *MusicManager*, atado al objeto *GameMusic*, es el encargado de administrar la música del fondo de juego. La importancia de este script radica en que, dado que se desea que la música se reproduzca en todas las escenas sin interrupción, es necesario asegurarse que la instancia de este administrador no se destruye en el cambio de escena por lo que está implementado mediante el patrón Singleton para evitar la creación de varias instancias de la misma clase.

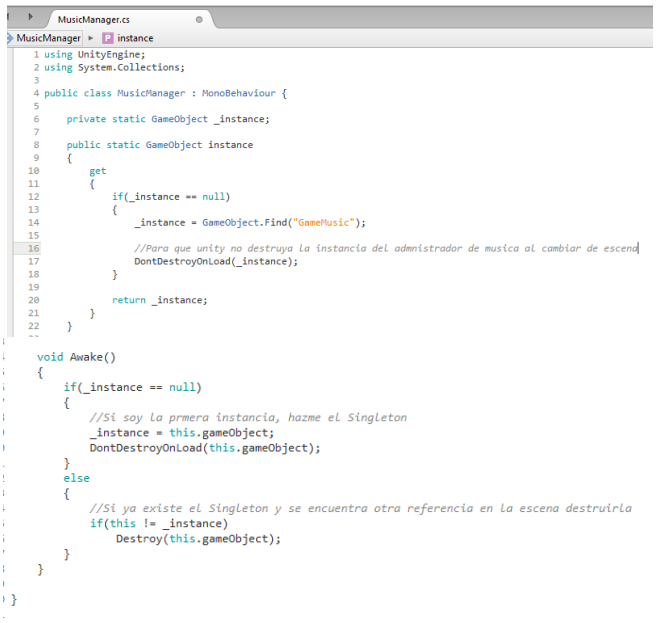


Figura 35. El script "MusicManager"

Además del script, este objeto contiene un elemento *AudioSource*. Éste es uno de los muchos elementos que proporciona Unity y en este caso es una "fuente de sonido", que se encarga de reproducir el sonido que se le indique desde la posición del objeto al que esté atado y con los ajustes de sonido que se le indiquen. En este caso reproduce la música de fondo del juego, añadida mediante la interfaz gráfica.

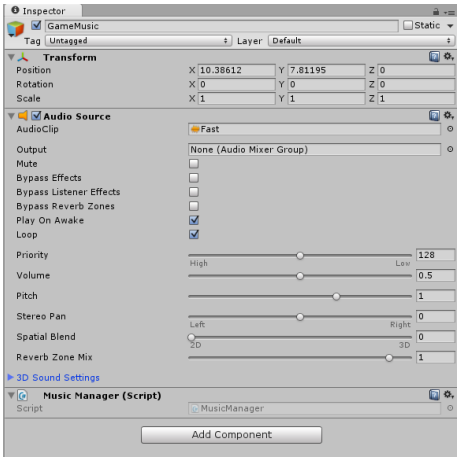


Figura 36. Objeto "GameMusic" con el script "MusicManager" atado y un componente AudioSource

Además de estos dos scripts, completan la escena una cámara por defecto colocada adecuadamente en la escena y un canvas que contiene la imagen de fondo, tres botones y la imagen de título.

La escena "Juego"

La escena de juego es la principal de todo el videojuego y es, en consecuencia, la que mayor carga de implementación ha necesitado.

Cuenta con dos objetos que son los que llevan la carga de la implementación de la jugabilidad, *Bloque* y *Tabla*.

Bloque

Bloque es el objeto que representa la pieza controlable por el jugador en la partida. A él está atado el script *BloqueApp*, que cumple la función Application dentro del modelo AMVCC que se explica en el capítulo "Arquitectura del software". Este script, solo contiene las instancias al resto de elementos del modelo AMVCC.

Además, en su interior cuenta con otros tres objetos: *BloqueVista*, *BloqueControlador* y *BloqueModelo*. Cada uno de éstos conforman el resto de elementos del modelo AMVCC y cuentan con scripts en su interior.

BloqueModelo es el elemento Modelo de la arquitectura AMVCC y es el de más importancia. En él se guardan todos los datos y variables de la pieza, además de las funciones que puedan modificar a ésta. Dado que tiene gran extensión se explicará poco a poco.

Las variables que controla son un array de objetos llamado *piezas* con los prefabs de los posibles bloques de dos cohetes que se pueden crear (en nuestro caso cuatro: RojoAmarillo, RojoAzul, VerdeAmarillo y VerdeAzul), otro array de objetos llamado *cohetes* esta vez con los prefabs de los cohetes con los que se puede formar la pieza (otra vez cuatro: CoheteRojo, CoheteAmarillo, CoheteVerde y CoheteAzul), dos objetos *cohete1* y *cohete2* que representan a los dos cohetes que definitivamente definen a la pieza y una instancia del tablero de juego llamada simplemente *Tabla*.

Las funciones que contiene son las siguientes:

CheckMove(), que se encarga de comprobar si cualquiera de los movimientos es posible.

SpawnJugador(), que se encarga de spawnear un bloque aleatorio de entre los del array *piezas*.

Rotar(), que rota la pieza 90º a la derecha si la llamada a CheckMove() lo permite.

MoverseIzquierda()/MoverseDerecha(), que realiza el movimiento deseado si la llamada a CheckMove() lo permite.

MoverseAbajo(), que realiza el movimiento si la llamada a CheckMove() lo permite. De lo contrario llena la tabla con los cohetes que forman la pieza en la posición actual.

Update(), que llama a SpawnJugador() constantemente.

```
using UnityEngine;
using System.Collections;

public class BloqueModelo : BloqueElement {

    public GameObject[] grupos;
    public GameObject[] cohetes;
    public GameObject cohete1;
    public GameObject cohete2;

    public TablaApp Tabla;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        StartCoroutine ("EsperarYSpawnear");
    }

    public void MoverseIzquierda(){
        if (CheckMove ("izquierda")) {
            Vector3 nuevaPos1 = new Vector3 (cohete1.transform.position.x - 1, cohete1.transform.position.y, cohete1.transform.position.z);
            Vector3 nuevaPos2 = new Vector3 (cohete2.transform.position.x - 1, cohete2.transform.position.y, cohete2.transform.position.z);
            Destroy (cohete1);
            Destroy (cohete2);
            cohete1 = Instantiate (cohetes[0], nuevaPos1, Quaternion.identity) as GameObject;
            cohete2 = Instantiate (cohetes[1], nuevaPos2, Quaternion.identity) as GameObject;


        }
    }



    public void MoverseDerecha(){
        if (CheckMove ("derecha")) {


            Vector3 nuevaPos1 = new Vector3 (cohete1.transform.position.x + 1, cohete1.transform.position.y, cohete1.transform.position.z);
            Vector3 nuevaPos2 = new Vector3 (cohete2.transform.position.x + 1, cohete2.transform.position.y, cohete2.transform.position.z);
            Destroy (cohete1);
            Destroy (cohete2);
            cohete1 = Instantiate (cohetes[0], nuevaPos1, Quaternion.identity) as GameObject;
            cohete2 = Instantiate (cohetes[1], nuevaPos2, Quaternion.identity) as GameObject;

        }
    }
```

Figura 37.1 Extracto del script "BloqueModelo"

Para ver el contenido del script BloqueModelo al completo puede consultarlo en la sección "Anexo: Figuras" figuras 37.1 a 37.5

BloqueControlador es el elemento Controlador del modelo AMVCC y es el encargado de procesar los inputs de la pantalla táctil del dispositivo y notificar al Modelo el movimiento que debe realizar dependiendo de la zona de la pantalla pulsada.

```
using UnityEngine;
using System.Collections;

public class BloqueControlador : BloqueElement {


  private Vector3 touchOrigin;
  private bool moviendo;
  private string movimiento="ninguno";

  void Update () {

      //this.transform.position = cohete2.transform.position;
      int nbTouches = Input.touchCount;
      if(nbTouches > 0)
      {

          Touch myTouch = Input.touches[0];

          if (myTouch.phase == TouchPhase.Began) {
              touchOrigin = myTouch.position;
              moviendo = true;
              Vector3 touchOriginConvertido= Camera.main.ScreenToWorldPoint (touchOrigin);
              if (touchOriginConvertido.x > this.transform.position.x+1 && touchOriginConvertido.y > 4) {
                  movimiento = "Derecha";
              } else if (touchOriginConvertido.x < this.transform.position.x-1 && touchOriginConvertido.y > 4) {
                  movimiento = "Izquierda";
              } else if ((touchOriginConvertido.x < this.transform.position.x+1)&&(touchOriginConvertido.x > this.transform.position.x-1)&&(touchOriginConvertido.y < this.transform.position.y+1)
                  && (touchOriginConvertido.y> this.transform.position.y-1)) {
                  movimiento = "Rotar";
                  //} else if (touchOriginConvertido.y < this.transform.position.y-2) {
              } else if (touchOriginConvertido.y < 4) {
                  movimiento = "Abajo";
              }
          }else if(myTouch.phase == TouchPhase.Ended){
              movimiento = "Ninguno";
              moviendo = false;
          }
      }
      if (moviendo) {
          // Move Left
          if ((Input.GetKeyDown (KeyCode.LeftArrow))||(movimiento=="Izquierda")) {
              app.modelo.MoverseIzquierda ();

          }// Move Right
          else if ((Input.GetKeyDown (KeyCode.RightArrow))||(movimiento=="Derecha")) {
              app.modelo.MoverseDerecha ();
          }// Rotate
```

Figura.38.1. Extracto del script "BloqueControlador"

Para ver el contenido del script BloqueControlador al completo, puede consultarlo en la sección "Anexos: Figuras" figuras 38.1 y 38.2

BloqueVista es el elemento Vista del modelo AMVCC y es el encargado de mostrar la pieza en la escena. Para ello lo único que hace es realizar la instancia de los dos cohetes que forman la pieza a partir del array *piezaFinal*.

Tabla

Tabla es el objeto que representa el tablero de juego. A diferencia del objeto Bloque, éste no obedece al modelo AMVCC, dado que no responde a ningún input del usuario y no es necesario aplicarlo en este caso.

A él está atado el script *Tabla*, que se encarga de manejar y controlar el tablero de juego.

Los datos que contiene y controla son tres arrays que contienen todos los tipos de cohetes que pueden aparecer en la tabla (incluidos los de bonus), y un array con todas las animaciones de fuegos artificiales que existen.

Mediante estos datos, las funciones que realiza este script son las de rellenar el tablero al principio del juego, comprobar constantemente si hay alguna combinación y si se da el caso explotar y activar las animaciones de fuegos artificiales y colapsar la tabla acto seguido y comprobar si se cumple alguna de las condiciones que hacen terminar al juego (sobrepasar los límites del tablero o el tiempo proporcionado para jugar).

```
using UnityEngine;
using System.Collections;

public class Tabla : MonoBehaviour {


    public static int anchura= Constants.columnas;
    public static int  altura=Constants.filas;
    public static Cohete[,] tabla = new Cohete[anchura, altura];
    public static int filasRellenado;

    public GameObject[] animaciones_fuegos;
    public GameObject[] cohetes;
    public GameObject[] cohetesBonus4;
    public GameObject[] cohetesBonus5;

    private Animacion_Fuego anim;

    void Awake (){
        Constants.cayendo=true;
        Constants.comprobandoMatchs = false;
        Constants.iniciando=true;
        Constants.hayMatches=true;
        Constants.rellenando = true;
        Constants.tableroListoParaPrimeraComprobacion = false;
        Constants.primerColapso=false;
        Constants.colapsado=false;
        Constants.posado = false;
        Constants.gameOver = false;

        Constants.puntuacion=0;

        anim=this.GetComponent<Animacion_Fuego>();
    }


    void Start () {
        PlayerPrefs.SetInt ("Puntuacion",0);
        generarTabla ();
        RellenarTabla ();
        Constants.primerColapso = true;

    }
```

Figura 39.1. Extracto del script "Tabla"

Para ver el contenido completo del script Tabla, puede consultarlo en la sección "Anexos: Figuras" figuras 39.1 a 39.14

Por último, dentro del canvas de esta escena, encontramos un objeto llamado *CanvasManagerJuego*, que se encarga, mediante un script de mismo nombre, de adaptar todos los objetos del canvas de esta escena según la resolución de la pantalla del dispositivo móvil y mantener los textos de puntuación y tiempo actualizados.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class CanvasManagerJuego : MonoBehaviour {

    public Text puntuacionText;
    public Text tiempoText;

    void Start(){
        puntuacionText.fontSize = puntuacionText.fontSize * (Screen.height/200);
        tiempoText.fontSize = tiempoText.fontSize * (Screen.height/200);
    }

    void Update(){

        if (puntuacionText.name == "Puntuacion") {

            puntuacionText.text = "PUNTUACIÓN: " + Constants.puntuacion;
        }
        if (tiempoText.name == "Tiempo") {

            tiempoText.text = "TIEMPO: " + (60-(int)Time.timeSinceLevelLoad).ToString();
        }

    }
}

```

Figura 40. Script "CanvasManagerJuego"

La escena "Puntuación"

La escena de puntuación es la escena final del juego y su única función es la de mostrar mediante canvas la puntuación final del usuario y si ha superado el record anterior o no. Además cuenta con un botón para volver a jugar.

Debido a que en esta escena solo se maneja este canvas, los únicos objetos con scripts implementados vuelven a ser uno para adaptar los elementos de éste a la resolución del móvil y actualizar los textos dependiendo del resultado de la partida realizada llamado *CanvasManagerPuntuación*, y otra vez el script *SceneChanger* para controlar el flujo entre escenas al pulsar el botón "Volver a intentar".

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;

public class CanvasManagerPuntuacion : MonoBehaviour {

    public Text mensajeText;
    public Text scoreText;
    public Text tituloText;
    public Text buttonText;
    public int puntuacion;
    private int highScore;


    void Start(){
        tituloText.fontSize = tituloText.fontSize * (Screen.height/200);
        mensajeText.fontSize = mensajeText.fontSize * (Screen.height/200);
        scoreText.fontSize = scoreText.fontSize * (Screen.height / 200);
        buttonText.fontSize = buttonText.fontSize * (Screen.height / 200);

        puntuacion = PlayerPrefs.GetInt ("Puntuacion");
        highScore = PlayerPrefs.GetInt ("HighScorePref");
        if (puntuacion > highScore) {
            highScore = puntuacion;
            if (mensajeText.name == "Mensaje") {

                mensajeText.text = "Has superado tu anterior record. Buen Trabajo!!";
            }
        } else {
            if (mensajeText.name == "Mensaje") {
                mensajeText.text = "Esta vez no has conseguido superarte. Intentalo otra vez!!";
            }
        }

        PlayerPrefs.SetInt ("HighScorePref",highScore);
        if (scoreText.name == "HighScore") {

            scoreText.text = "PUNTUACIÓN RECORD: " + highScore;
        }

    }

}
```

Figura 41. Script "CanvasManagerPuntuacion"

7.3. Pruebas

A la hora de probar el código implementado, se ha seguido una metodología de pruebas de *caja negra* y *caja blanca*, probando cada funcionalidad nueva en el momento en que ésta ha sido implementada.

7.3.1. Pruebas de caja negra

Las pruebas de caja negra se encargan de comprobar el correcto funcionamiento de un módulo o funcionalidad sin conocer el código que lo implementa, es decir, sin conocer a priori los casos que hacen que el código pueda fallar.

Para realizar la prueba, se comprueban las entradas de la funcionalidad que se quiere probar y se comparan el resultado obtenido y el resultado esperado.

Han sido muchas las pruebas de caja negra realizadas sobre las distintas funcionalidades implementadas por lo que simplemente se mostrará un ejemplo de una, en este caso el movimiento hacia la derecha.

Empezamos definiendo las distintas entradas que afectan al resultado obtenido en la funcionalidad, que en este caso sería el estado de la posición a la derecha de la pieza que se está moviendo. Acto seguido, se definen los distintos casos que pueden ocurrir dependiendo del valor de esas entradas y se comprueban los resultados obtenidos en comparación con el resultado esperado. En la siguiente figura se pueden ver los resultados de esta prueba.

Movimiento hacia la derecha		
Estado posición a la derecha	Resultado esperado	Resultado obtenido
LIBRE	ÉXITO	ÉXITO
OCUPADO	FALLO	FALLO

Figura 42. Ejemplo de prueba de caja negra

Si durante la prueba en algún caso no hubiesen coincidido el resultado obtenido con el esperado, entonces habría que volver a la implementación de esa funcionalidad y arreglar la causa del error. En este caso se confirma que la funcionalidad está implementada correctamente.

7.3.2. Pruebas de caja blanca

Las pruebas de caja blanca nos permiten encontrar casos de uso especiales de una funcionalidad que den error mediante el conocimiento del código implementado.

Siguiendo el ejemplo anterior del movimiento hacia la derecha, existe un caso especial en el que la posición a la derecha es una posición que está fuera de la tabla de juego. Así pues se comprueba este caso especial nuevamente comparando el resultado obtenido y el esperado.

Movimiento hacia la derecha		
Estado posición a la derecha	Resultado esperado	Resultado obtenido
LIBRE	ÉXITO	ÉXITO
OCUPADO	FALLO	FALLO
FUERA DE TABLA*	FALLO	FALLO
*= caso especial de caja blanca		

Figura 43. Ejemplo de prueba de caja blanca

Así pues una vez hecha la prueba de caja blanca se ve que en este caso la implementación es totalmente correcta para cualquier caso. De no ser así, habría que corregir el código implementado para cubrir este tipo de casos especiales.

8

Plan de entrevistas

Durante el desarrollo de cualquier producto software es indispensable realizar una fase de testeo con usuarios mediante la cual se hallen posibles correcciones, mejoras o exclusiones. En este caso, esta necesidad se repite, por lo que se han realizado una serie de entrevistas a distintos perfiles de usuarios mientras probaban Firework Madness.

8.1. Alcance

8.1.1. Descripción del alcance

Se han realizado una serie de entrevistas a un grupo de usuarios mientras prueban el producto para el posterior análisis de éstas en busca de posibles correcciones y mejoras.

Para ello se ha hecho una búsqueda de voluntarios que cumplan distintos perfiles de usuario, desde el usuario normal al que tiene conocimientos del desarrollo de videojuegos en plataformas móviles. Dadas las diferencias entre estos perfiles también es necesario definir dos formatos de entrevista distintos, uno básico y otro más técnico para el perfil del desarrollador de este tipo de productos, en el que se profundiza más en elementos y tecnicismos del desarrollo de videojuegos. Sin embargo, pese a las diferencias ambos formatos comparten la estructura de la entrevista, que se basa en distintos apartados que componen el videojuego, desde estilo o jugabilidad hasta el aspecto audiovisual.

Una vez hechas todas las entrevistas se concentra toda la información extraída de estas en una serie de conclusiones que incluyen las posibles mejoras y correcciones que según el *feedback* recibido por los usuarios mejorarían la calidad del producto.

8.1.2. Exclusiones

Las entrevistas se han realizado en un dispositivo móvil suministrado por el entrevistador por comodidad y rapidez (instalar el producto en los móviles de los propios usuarios sería complejo debido a la necesidad de conectarlos al ordenador del desarrollador e instalar manualmente la apk del producto).

8.2. Objetivos

Estas entrevistas se realizan en cualquier desarrollo software de este tipo con el objetivo de hallar posibles mejoras de la calidad del producto antes de su lanzamiento oficial.

De esta forma, tras analizar las entrevistas, se pueden analizar éstas y descubrir:

- “Bugs” o errores que han ocurrido durante el testeo y que se tendrán que solucionar.
- Posibles mejoras a corto plazo.
- Posibles mejoras a largo plazo.
- Mejoras que han sido sugeridas pero que se descartan por tener un coste demasiado alto o por una decisión de negocio.

8.3. Fechas clave

Durante la realización del plan de entrevistas y puesta en marcha de éste tuvieron lugar las siguientes fechas clave:

- 28/IV/2016: Realización del primer formato de entrevista que se usará de cara a los perfiles no técnicos.
- 29/IV/2016: Entrevista con el primer perfil de usuario (persona adulta inexperta), documentación de ésta y mejora del formato usado para entrevistas posteriores.
- 30/IV/2016: Entrevista con el segundo perfil de usuario (joven que hace uso usualmente de este tipo de productos), documentación de ésta y preparación de las entrevistas con el perfil del informático.
- 3/V/2016: Realización de las dos entrevistas pertenecientes al perfil del informático, incluyendo documentación de ésta, y preparación del segundo formato de entrevista más técnico de cara a la entrevista con el perfil del trabajador del sector.
- 11/V/2016: Entrevista con el tercer y último perfil de usuario (trabajador del sector con experiencia en desarrollos de este tipo) haciendo uso del formato de entrevista técnico y documentación de ésta.
- 12/V/2016: Análisis conjunto de todas las entrevistas y extracción de resultados.

8.4. Intervinientes

Los usuarios entrevistados permanecerán anónimos, pero se pueden dividir en estos cuatro perfiles de usuario. Ninguno de estos perfiles refleja aspectos personales de los usuarios entrevistados que no tengan que ver con el carácter de la entrevista.

8.4.1. El adulto inexperto

María tiene 52 años y ahora mismo está en el paro. Dedica su tiempo a cuidar de su familia mientras busca trabajo pero esto le deja bastante tiempo libre. Debido a esto, le ha pedido a su hijo que le descargue algún juego a su nuevo smartphone que le sirva de entretenimiento en esos ratos aburridos.

8.4.2. El joven dado a estos productos

José tiene 20 años y está estudiando la carrera de Arquitectura. Pese a tener que estudiar mucho, siempre encuentra algún momento del día para estar con su móvil, ya sea hablando con sus amigos o jugando, aunque nunca dura más de unas semanas jugando al mismo juego.

8.4.3. El informático con conocimientos base

Juan tiene 26 años y es ingeniero informático. Aunque cuenta con conocimientos de informática debido a sus estudios, desconoce las bases del desarrollo de videojuegos, ya que es un sector al que nunca se ha dedicado, aunque sí es un jugador habitual de este tipo de productos.

Este perfil supone el punto medio entre el usuario normal y el usuario técnico.

8.4.4. El trabajador de este sector con experiencia

Iván tiene 32 años y trabaja en el desarrollo de videojuegos. Lleva 8 años trabajando en empresas de esta industria y cuenta con varios juegos realizados por él en la *market* de Android. Debido a esto se puede confirmar que tiene gran experiencia en desarrollos de este tipo de software y resulta ser una gran ayuda en caso de necesidad.

8.5. Desarrollo de la experiencia

Durante la entrevista, al usuario primeramente se la ha dado una explicación de en qué consiste el juego, el objetivo que tiene y como se controla. Una vez terminada la introducción y que el entrevistador se ha asegurado de que el usuario no tiene ninguna duda pendiente se le ha presentado el producto a éste para su prueba.

Al usuario se le ha dejado probar el producto varias veces, dejándole libertad total y sin hablarle para no afectarle durante el uso que haga de esté.

Una vez terminadas las pruebas, se ha recogida el dispositivo de prueba y se le ha pedido al usuario que responda a una serie de preguntas sobre distintos aspectos del producto.

8.6. Formato para recoger los resultados

Para realizar las entrevistas y recoger los resultados se han desarrollado dos formatos distintos, uno para la entrevista con el desarrollador de este tipo de productos y otro para el resto de perfiles. Se ha tomado esta decisión dado que la entrevista con el desarrollador se realizará profundizando más en tecnicismos del desarrollo de videojuegos y puede aportar un *feedback* mayor de cara a posibles mejoras del producto.

8.6.1. Formato normal

Este formato hace uso de un lenguaje coloquial, de forma que el entrevistado pueda entender todo lo que se le pregunte sin confusiones.

8.6.2. Formato técnico

En el formato técnico se plantean cuestiones más técnicas, profundizando en elementos del desarrollo de videojuegos y en el uso que se ha hecho de ciertas herramientas, por ejemplo hablando del sistema de partículas utilizado para crear los fuegos artificiales.

8.7. Entrevistas

En esta sección se recogen las entrevistas realizadas:

8.7.1. Entrevistas pertenecientes al formato normal

8.7.1.1. Entrevistados

Para referirnos a las respuestas de los distintos entrevistados en cada pregunta se referirá a cada uno de éstos mediante las siguientes abreviaturas:

AI: Se refiere al adulto inexperto.

JD: Se refiere al joven dado a este tipo de productos.

IC1: Es el primer informático con conocimientos base.

IC2: Es el segundo informático con conocimientos base.

8.7.1.2. Entrevista

Estilo

Este punto se centra en saber la opinión del usuario sobre el mundo que se plantea en el juego:

P: **¿Qué te parece el estilo del juego y el mundo que se plantea en él?**

AI: La idea que tiene y el estilo está bien, pero la forma en que está dibujado ese mundo en el juego no me termina de gustar.

JD: Me gusta pero creo que le falta algo que le dé más personalidad.

IC1: No está mal, aunque personalmente no es santo de mi devoción.

IC2: No deja de ser el estilo que se lleva ahora mismo en los juegos que he probado en mi móvil.

P: ¿Te convence la idea de crear fuegos artificiales mientras juegas?

AI: La verdad es que me parece algo bastante bonito y original.

JD: Me gusta bastante porque lo que consigo en el juego se refleja en los fuegos que salen y por ejemplo cuando he conseguido algún combo bueno han aparecido muchísimos fuegos a la vez que quedan chulos.

IC1: Aunque no es el tipo de juegos que suelo jugar, esa parte sí que me gusta.

IC2: Está bien, al menos es algo distinto a lo que ya se ve en el mercado.

P: ¿Qué añadirías o quitarías para hacer el mundo del juego más divertido?

AI: Estaría bien que cuando hagas un fuego artificial salga algo más en pantalla porque ahora mismo está bien pero se queda un poco pobre. Además los fuegos que salen siempre son los mismos, me gustaría que hubiese más variedad.

JD: Me gustaría que hubiese más fuegos diferentes y no siempre el tipo "palmera" que está ahora.

IC1: Estaría bien que hubiese más variedad de dibujos y fuegos.

IC2: Intentaría arreglar el estilo general del juego, añadir algo que le diese un punto original que le distinguiera del resto de videojuegos.

Jugabilidad

La jugabilidad se centra en el tipo de juego que se le propone al usuario, en el caso de este producto puzles:

P: ¿Qué te parece el tipo de puzle que se te plantea en el juego?

AI: Me parece que es entretenido, pero no engancha la verdad. Creo que le falta algo aunque no sé el qué.

JD: No me convence, al final es algo que está muy visto ya y no tiene algo que le diferencie del resto de juegos y te enganche.

IC1: Los puzles nunca me han gustado demasiado pero al menos entretiene.

IC2: He probado unos cuantos juegos de puzles porque son mis preferidos y nunca había visto uno de este tipo, así que al menos es original.

P: ¿Te gustaría que hubiese más variedad de combinaciones posibles o te parecería demasiado complicado?

AI: Aunque a mí me ha costado un poco al principio hacer las más simples, estaría bien que hubiese más cosas que hacer o por ejemplo que se puedan hacer combinaciones en diagonal.

JD: Con más variedad de combinaciones mejoraría. Por ejemplo, pudiendo hacer combinaciones en L o diagonales.

IC1: Estaría bien tener más opciones o que por ejemplo de vez en cuando apareciera algún tipo de comodín que cambiase la dinámica un poco.

IC2: Estaría bien tener más formas de combinar que no fuesen horizontal o vertical.

P: ¿Prefieres que el juego se base en un solo nivel donde tu objetivo es batir tu propio record o te gustaría más que hubiese un mundo con distintos niveles y objetivos a ir superando?

AI: La verdad es que un solo nivel como ahora se hace aburrido al de un rato. La idea de los niveles distintos me gusta más y creo que me "picaría" más a seguir jugando.

JD: Prefiero niveles, así puedo picarme con mis amigos a ver quién lo hace mejor en cada nivel y el juego va cambiando a medida que avanzas.

IC1: Prefiero niveles porque al menos tendría un objetivo más claro que simplemente superar mi puntuación anterior.

IC2: Personalmente me gustan más los juegos que funcionan por niveles.

P: ¿Qué cambiarías para qué fuese más divertido?

AI: Preferiría que el juego te pidiese hacer algo específico, en vez de simplemente hacer la mayor puntuación posible.

JD: Metería más posibilidades, piezas que bloqueen zonas del tablero u objetivos dentro del nivel, por ejemplo hacer 5 combinaciones de 4 cohetes en una partida.

IC1: Metería de alguna forma el poder competir con amigos y rivalizar con otra gente.

IC2: Algún tipo de obstáculos que complicasen la partida y que el tiempo de juego aumentase a medida que puntúas.

Control

En este punto queremos saber que piensa el usuario del control actual del producto:

P: **¿Te sientes cómodo a la hora de jugar?**

AI: Excepto el movimiento de rotar la pieza, el resto se controla bien.

JD: El control va bien, pero no me gusta que cuando la pieza está pegada a una de las paredes, ésta no se pueda girar. En otros juegos que he probado, gira aunque sea moviéndote a la derecha si estás en la pared de la izquierda o al revés.

IC1: Para mí el control está bien, pero creo que en el caso de gente más torpe con el móvil podría liarse entre bajar la pieza y moverla a los lados y rotarla

IC2: Una vez que lo entiendes el control funciona bien pero me he acostumbrado a jugar a otros juegos del estilo deslizando el dedo por la pantalla en vez de pulsando solo y ahora no me acomodo del todo.

P: **¿Si se te diese la opción, qué preferirías este modo de control o una serie de botones en la pantalla?**

AI: Me gusta más el control como ahora pero prefiero que se pueda rotar más fácil. Con botones no sé cómo se vería en la pantalla.

JD: Prefiero como está, no me gusta cuando se controla por botones, no suele ser nada cómodo.

IC1: No creo que poner botones quede bien, sobre todo si el móvil en el que se juegue fuese pequeño.

IC2: Sin botones está bien porque tienes toda la pantalla para jugar.

Aspecto audiovisual

El aspecto audiovisual lo conforman tanto los gráficos del juego y la interfaz, como la música escogida y los sonidos que se producen:

P: ¿Qué te parece el juego gráficamente?

AI: El fondo, los botones y las letras son muy simples, podrían ser algo más bonitos pero los cohetes me gustan como están dibujados.

JD: Excepto el fondo y la interfaz que son bastante simples, los cohetes y el título de entrada están chulos.

IC1: En general creo que es un poco simple y la interfaz es muy estándar.

IC2: Es sencillo en general. Creo que cambiando las letras y la barra de límite por un estilo más animado el juego ganaría bastante.

P: ¿La combinación de colores y la interfaz elegida te parece cómoda?

AI: Los colores de los cohetes si me gustan, porque además se diferencian bien entre ellos y el fondo, pero las letras y el resto de cosas, aunque sí que se diferencian, me siguen pareciendo algo feas.

JD: Los colores están bien porque no se mezclan entre sí, por ejemplo los cohetes se diferencian bien del fondo, pero sigo pensando que la interfaz es algo fea

IC1: Está bien pero tengo curiosidad sobre como funcionaría el juego con gente daltónica por ejemplo.

IC2: A mí no me ocurre, pero no sé si una persona con peor vista podría confundir los tonos verdes con la pradera del fondo o los amarillos con el camino del pueblo.

P: ¿Qué piensas de los fuegos artificiales que se crean?

AI: Están muy chulos pero siempre son los mismos y además podrían tener algo más de color.

JD: Es lo más chulo del juego pero podría haber más tipos distintos.

IC1: Es con diferencia lo más bonito y divertido del juego. Más variedad no le vendría mal.

IC2: Me gustan en general, igual estarían mejor si fuesen un poco más grandes.

P: ¿Qué piensas del sonido del juego?

AI: Respuesta: La música que suena no termina de gustarme, pero el sonido de los cohetes explotando está logrado.

JD: La música de fondo no me gusta del todo, aunque el sonido de los fuegos artificiales está bien.

IC1: Es simple, pero la música pega con el estilo del juego y el sonido del cohete está conseguido.

IC2: Está bien, cumple pero no destaca.

Problemas encontrados y sugerencias importantes

En este punto, el entrevistado tiene libertad total para comentar algún problema que haya encontrado durante la prueba o hacer sugerencias o comentarios que se hayan pasado por alto en los anteriores apartados:

AI: Se me ha hecho corto el tiempo que te da para jugar. Éste debería ser mayor o que se pudiese conseguir más de alguna forma en el juego.

JD: He notado que cuando la pieza está horizontal y uno de los cohetes cae y el otro se posa, el que va cayendo explota en cuanto se cruza con una posible combinación, en vez de solo intentar combinarse cuando esté posado también.

IC1: Creo que se debería dar más tiempo para jugar, se me hace corto así. Además he notado que el límite de altura para jugar y el lugar donde aparecen las piezas están demasiado juntos, estaría bien separarlos un poco.

Por último, el *modo fácil* que he probado resultaba demasiado fácil en mi opinión.

IC2: Se me ocurre que quedaría bien que el contador de tiempo fuese una barra horizontal con forma de mecha que fuese vaciándose hasta explotar y terminar el juego, creo que esto pegaría con el estilo.

Además la barra podría rellenarse y ganar tiempo con las puntuaciones.

8.7.1.3.Observaciones durante la prueba

Durante la prueba y sin que el usuario sea consciente de ello se han tomado las siguientes observaciones sobre el curso de ésta:

P: ¿Qué modos de dificultad ha probado, y en qué orden?

AI: Solo ha probado el fácil.

JD: Solo ha jugado en difícil.

IC1: Ha probado primero el fácil y luego el difícil.

IC2: Ha probado los modos fácil y difícil

P: ¿Ha tardado en comprender el funcionamiento del juego y el control?

AI: Al principio pensaba que se podían hacer combinaciones en diagonal y a la hora de controlar tendía a deslizar el dedo por la pantalla en vez de tocar solo.

JD: Una vez explicado, ha jugado sin problemas excepto el que ha comentado de rotar la pieza en la pared.

IC1: No ha tardado nada tras la explicación.

IC2: No.

P: ¿Ha realizado algún combo especial o se ha limitado a combinaciones simples?

AI: Solo ha hecho combinaciones simples y con dificultad.

JD: Se concentraba más en mirar si podía hacer combos especiales y probarlos que en hacer combos simples.

IC1: Solo ha hecho combinaciones simples.

IC2: Solo ha conseguido combos sencillos.

P: ¿Tras los primeros intentos ha mejorado su puntuación y control del juego?

AI: La puntuación ha sido prácticamente la misma pero poco a poco ha ido mejorando en el control del juego, aunque seguía fallando en el movimiento de rotar la pieza.

JD: La puntuación ha sido prácticamente la misma y en cuanto al control, se ha adaptado a no poder rotar en la pared rápidamente.

IC1: Se ha mantenido prácticamente igual.

IC2: Sí, ha ido mejorando poco a poco.

8.7.2. Entrevistas pertenecientes al formato técnico

8.7.2.1. Entrevistados

Para referirnos a las respuestas del entrevistado en cada pregunta se referirá a éste mediante las siglas **TS**, abreviatura de "trabajador del sector"

8.7.2.2. Entrevista

Estilo

Este punto se centra en saber la opinión del usuario sobre el mundo que se plantea en el juego:

P: **¿Qué te parece el estilo del juego y el mundo que se plantea en él?**

TS: No me parece que tenga un estilo definido en sí, no veo nada que lo haga único en el estilo del juego.

P: **¿Te convence la idea de crear fuegos artificiales mientras juegas?**

TS: Es sin duda el aspecto y la idea más original del juego.

P: **¿Qué añadirías o quitarías para hacer el mundo del juego más divertido?**

TS: Habría que buscar algo que le diese un punto de originalidad y simpatía al juego, por ejemplo añadir una mascota o personajes.

Jugabilidad

La jugabilidad se centra en el tipo de juego que se le propone al usuario, en el caso de este producto puzles:

P: ¿Qué te parece el tipo de puzle que se te plantea en el juego?

TS: El tipo de puzle se ve que está bien implementado y funciona bien, por lo que ya depende de los gustos de las personas. Yo creo que está bien esa mezcla de distintas mecánicas de otros juegos del estilo.

P: Ahora mismo las combinaciones solo se pueden hacer por posición, ya sea una línea recta horizontal o vertical, ¿crees que así es suficiente profundo para jugar o crees que incluir otros tipos de combinaciones posibles mejorarían el juego, por ejemplo combinando por contacto, es decir pudiendo realizar por ejemplo combinaciones en diagonal o en formas complejas?

TS: Creo que sería demasiado para el jugador. Si añades muchas combinaciones puede pasar que el jugador acabe haciendo combinaciones sin querer y no entienda que es lo que ocurre.

P: Actualmente el juego solo cuenta con una escena jugable, en la que el objetivo es superar tu propia puntuación. ¿Te gusta este planteamiento o preferirías tener múltiples escenas con layouts del tablero distinto y objetivos distintos?

TS: Eso depende. En este caso y con el tipo de juego que es creo que pega más la idea de hacer un solo nivel y basar toda la jugabilidad en hacer records en algún tipo de ranking online.

Sin embargo, yo si hiciese un juego de este tipo intentaría adaptarlo a un mundo con distintos niveles y progreso, ya que actualmente este tipo de juegos están más de moda.

P: ¿Qué cambiarías para qué fuese más divertido?

TS: Se me ocurren un par de ideas. Primeramente aumentar la velocidad de caída de la pieza con el tiempo, crear algún tipo de cohete comodín que aparezca de vez en cuando y de algo de variedad, tener piezas de distintos tamaños en vez de siempre la misma pero con distintos cohetes.

Además creo que hace falta "lavar" un poco el estilo y pulir las animaciones del tablero.

Control

En este punto queremos saber que piensa el usuario del control actual del producto:

P: ¿Te sientes cómodo a la hora de jugar realizando pulsaciones sueltas en la pantalla? ¿Qué pensarías de cambiar el análisis de inputs para moverse solo cuando el usuario deslice la pieza a través de la pantalla?

TS: Creo que quedaría mejor mover mediante deslizamientos, calculando la posición de inicio de la pulsación y la posición en la que se deja de pulsar y moviendo entonces.

P: ¿Si se te diese la opción, qué preferirías este modo de control o una serie de botones en la pantalla realizados con el canvas de Unity?

TS: Los botones de canvas nunca quedan bien para controlar el juego, es mucho más limpio hacerlo como ahora, sin interfaz que obstaculice la vista del juego.

Aspecto audiovisual

El aspecto audiovisual lo conforman tanto los gráficos del juego y la interfaz, como la música escogida y los sonidos que se producen:

P: ¿Qué te parece el juego gráficamente?

TS: Excepto los fuegos artificiales, que están bien creados, el resto gráficamente no es bueno.

P: ¿La combinación de colores y la interfaz elegida te parece cómoda?

TS: Creo que quedaría mejor si los cohetes tuviesen un contorno negro a su alrededor que los hiciese destacar un poco más.

P: Los fuegos artificiales están creados con el sistema de partículas de Unity, ¿qué te parecen? ¿Crees que es buena idea hacerlos de esta forma o quedaría mejor haberlos realizado con alguna herramienta de edición de animaciones?

TS: Para este caso, el sistema de partículas cumple de sobra y queda bien. Aunque las herramientas de animación ofrecen más herramientas y posibilidades, éstas son más complejas y no merece la pena usarlas para este caso.

P: ¿Qué piensas del sonido actual del juego? ¿Crees que el uso de los AudioSource dentro de la escena del juego es el correcto?

TS: La música está bien y la idea de hacerla sonar continuadamente a lo largo de las tres escenas es buena.

Problemas encontrados y sugerencias importantes

En este punto, el entrevistado tiene libertad total para comentar algún problema que haya encontrado durante la prueba o hacer sugerencias o comentarios que se hayan pasado por alto en los anteriores apartados:

TS: Creo que sería necesario que se activase algún sonido de alarma cuando el jugador se esté quedando sin tiempo. Por ejemplo, la primera partida no me he dado cuenta cuando estaba terminándose el tiempo. También daría más jugabilidad ganar tiempo con la puntuación.

8.7.2.3. Observaciones durante la prueba

Durante la prueba y sin que el usuario sea consciente de ello se han tomado las siguientes observaciones sobre el curso de ésta:

P: ¿Qué modos de dificultad ha probado, y en qué orden?

TS: Ha jugado tres partidas. La primera y segunda en normal y la tercera en difícil.

P: ¿Ha tardado en comprender el funcionamiento del juego y el control?

TS: No, ha comprendido todo en cuanto la explicación ha terminado.

P: ¿Ha realizado algún combo especial o se ha limitado a combinaciones simples?

TS: Ha probado todos los combos para comprobar todas las posibilidades que ofrece el juego.

P: ¿Tras los primeros intentos ha mejorado su puntuación y control del juego?

TS: Ha ido mejorando la puntuación porque ha empezado a intentar crear combinaciones complejas. El control ha sido bueno desde el principio.

8.8. Conclusiones

Una vez realizadas todas las entrevistas, se deben analizar éstas para sacar conclusiones. Se pueden dividir éstas en cuatro tipos: correcciones, mejoras realizables a corto plazo, mejoras cuyo coste sea alto y se pospongan a una futura continuación del desarrollo del producto y sugerencias descartadas ya sea por decisión propia del desarrollador o por tener un coste de desarrollo demasiado alto.

8.8.1. Correcciones

Durante las pruebas el único error que ha detectado el usuario ha sido el de que una vez la pieza pueda explotar mientras baja en vez de cuando está posado completamente. Este error fue inmediatamente solucionado el mismo día que el entrevistado lo descubrió.

8.8.2. Mejoras a corto plazo

Durante varias de las entrevistas el entrevistado tuvo quejas acerca del tiempo de juego, en concreto pidiendo que la duración de la partida fuese mayor o que ésta se pudiese alargar mediante combinaciones. Se ha decidido que la idea de ganar tiempo a medida que se juega es más divertida, por lo que se ha decidido implementar esto en vez de aumentar el tiempo base.

8.8.3. Mejoras a largo plazo

De entre todas las sugerencias que se han hecho ha destacado principalmente la idea de que hubiese más variedad de combinaciones y fuegos artificiales. Es una idea que puede ser implementada, pero necesita de tiempo para crear las nuevas animaciones de fuegos artificiales y incluir las nuevas combinaciones en el código actual y realizar este cambio antes de la presentación del producto tiene el riesgo de poner en peligro otros aspectos como la documentación o la estabilidad de la aplicación actual, por lo que se ha pospuesto a una futura expansión del juego.

Lo mismo ocurre con la idea de cambiar el control a un control por deslizamiento o las de cambiar el estilo general del juego, como dibujar fondos nuevos, mascotas, mejorar la interfaz, etc. Son tareas que necesitan un tiempo para ser realizadas que actualmente no se tiene, por lo que se posponen también.

8.8.4. Sugerencias descartadas

Pese a que la mayoría de los entrevistados prefería los juegos por niveles en vez de un solo nivel en el que superarte, hacer este cambio en el producto resulta demasiado costoso, dado que la idea de inicio es que el juego sea rápido y no de dedicación continua a través de niveles. Sumado a que el coste de realizar este cambio es muy alto, la decisión de no realizarlo es más de carácter personal, dado que iría en contra de la idea original del desarrollador.

8.9. Otros activos

Durante la realización de las entrevistas se realizaron las siguientes tomas gráficas de las pruebas:

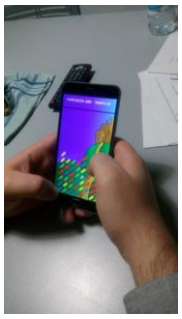


Figura 44. Foto obtenida durante la entrevista

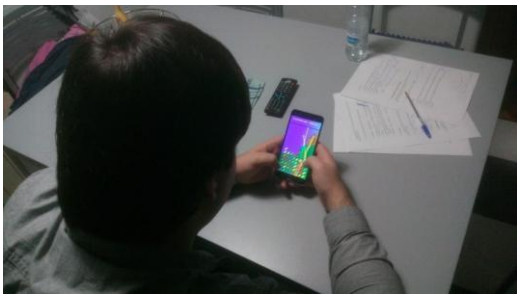


Figura 45. Usuario probando Firework Madness

Figura 46. Durante la entrevista

9

Gestión del proyecto

En este capítulo se define la gestión realizada en las distintas fases del proyecto: Inicio, planificación, seguimiento y control y cierre.

9.1. Fase de inicio

Durante la fase de inicio se constituyó la idea del proyecto a realizar y sus objetivos, es decir la idea de realizar un videojuego para plataformas móviles aplicando técnicas de desarrollo de software propias de un ingeniero de la especialidad de software. Además se definió las características generales que tendría ese videojuego, estilo, jugabilidad, etc. y se eligió las herramientas que se utilizarían para implementarlo, por ejemplo usar Unity como motor de desarrollo.

9.2. Fase de planificación

La fase de planificación incluye la definición del alcance del proyecto, de los entregables que compondrán a éste, la creación de un calendario de trabajo acorde y de un EDT (Estructura de Descomposición del Trabajo) y la gestión general de tiempos, costes y riesgos.

El alcance es el mismo definido en el capítulo 2, Objetivo del proyecto.

Los entregables planificados que componen el proyecto son los siguientes:

- Una aplicación móvil completamente funcional e instalable en cualquier móvil Android para su uso y presentación.
- Una memoria del desarrollo del software donde se describirá la funcionalidad de éste, el proceso de desarrollo, problemas encontrados y soluciones propuestas, etc.

Para la creación del calendario de tiempo se ha hecho uso de la herramienta Google Calendar©, marcando como fechas clave las entregas de los distintos hitos y fechas de riesgo debido a carga lectiva u otros motivos.

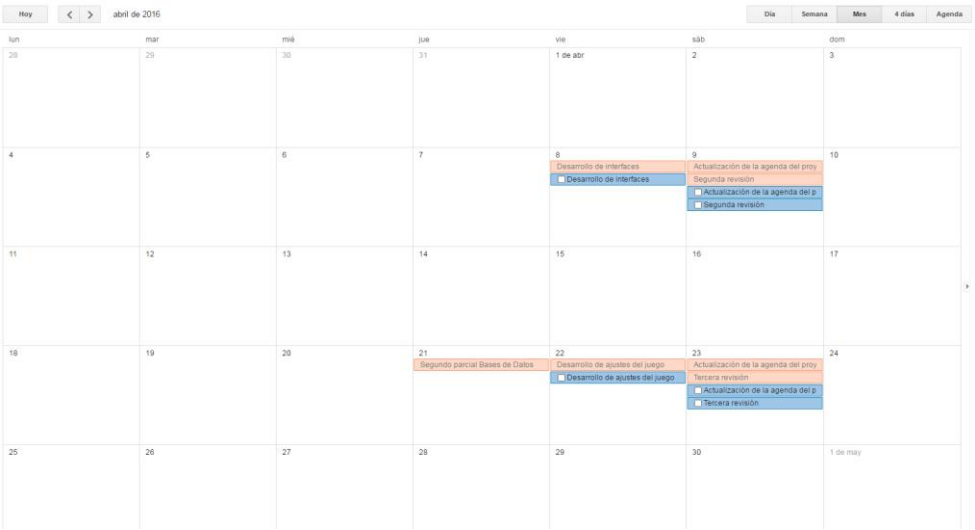


Figura 47. Extracto del calendario del desarrollo del proyecto

Inicialmente el EDT se planificó de la siguiente forma:

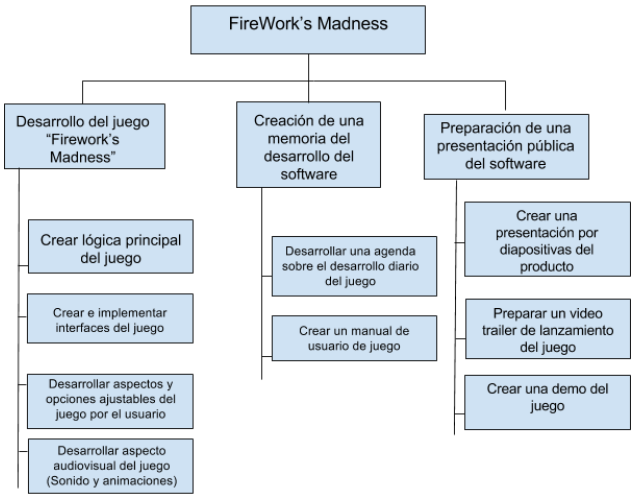


Figura 48. EDT del proyecto

En cuanto a la gestión del tiempo empleado en el desarrollo del proyecto, inicialmente se planteo de forma errónea, pensando en una dedicación de 7 horas semanales en semanas sin otras cargas y de 3 horas semanales en caso de tener otras obligaciones sumando un total de 140 horas que al final no reflejan el tiempo invertido realmente, pero estos cambios se comentarán en la fase de seguimiento y control.

Además para la gestión del momento de realización de las entrevistas con los distintos perfiles de usuario se contactó con éstos mediante correo electrónico o en persona y se acordaron mutuamente las siguientes fechas:

- El 29/04/2016 desde las 17:30 hasta las 18:30 para la entrevista con el perfil del adulto inexperto.
- El 30/04/2016 de 18:00 a 19:00 para la entrevista con el perfil del joven dado a estos productos.
- El 3/05/2016 de 16:00 a 17:00 para la entrevista con el primer usuario perteneciente al perfil del informático con conocimientos base.
- El 3/05/2016 de 17:00 a 18:00 para la entrevista con el segundo usuario perteneciente al perfil del informático con conocimientos base.
- El 11/05/2016 de 11:00 a 12:00 para la entrevista final con el usuario perteneciente al perfil del trabajador de este sector con experiencia.

La gestión de costes ha sido de una única decisión, y ésta es la de hacer uso de herramientas gratuitas o de herramientas con las que ya se contase con una licencia de trabajo anteriormente, evitando de esta forma cualquier tipo de coste en el desarrollo del producto.

Por último en la gestión de riesgos se identificaron los siguientes dos riesgos:

1. La correcta planificación del tiempo disponible y el reparto de éste entre las distintas tareas por hacer.
2. La decisión de las funcionalidades definitivas que tendrá el proyecto, definiendo el punto intermedio entre tener un proyecto pobre y en el que ha sobrado tiempo y un proyecto a medio terminar porque el número de funcionalidades que se ha querido abarcar ha sido demasiado.

A estos dos riesgos se le deben sumar las siguientes fechas críticas identificadas:

- Semana 5 (22/II): Primera semana de horario agrupado y por lo tanto semana de exámenes parciales y entregas de trabajos, por lo que la carga lectiva es mayor y es necesario controlarla.
- Semanas 9 (21/III) y 10 (28/III): Vacaciones de semana santa y momento en el que me voy de viaje de fin de carrera, por lo que la dedicación ese tiempo es nula y conviene llegar a este momento con el trabajo realizado a buen ritmo y sin grandes retrasos.
- Semana 13 (18/IV): Segunda semana de horario agrupado, y nuevamente mayor carga lectiva.
- Semanas 18 (23/V) y 19 (30/V): Últimas semanas de clase y momento de realización de los últimos exámenes parciales y entrega de las prácticas finales.
- Semanas 20(6/VI), 21(13/VI) y 22(20/VI): Últimas semanas antes de entregar el proyecto, por lo que son semanas importantes para perfilar los últimos detalles del proyecto y asegurarse del total funcionamiento correcto.

Para gestionar estos riesgos se decidió dedicar un tiempo mayor antes del viaje y de las semanas de entrega de trabajos y exámenes, para adelantar trabajo y evitar una posible falta de tiempo después.

9.3. Fase de seguimiento y control

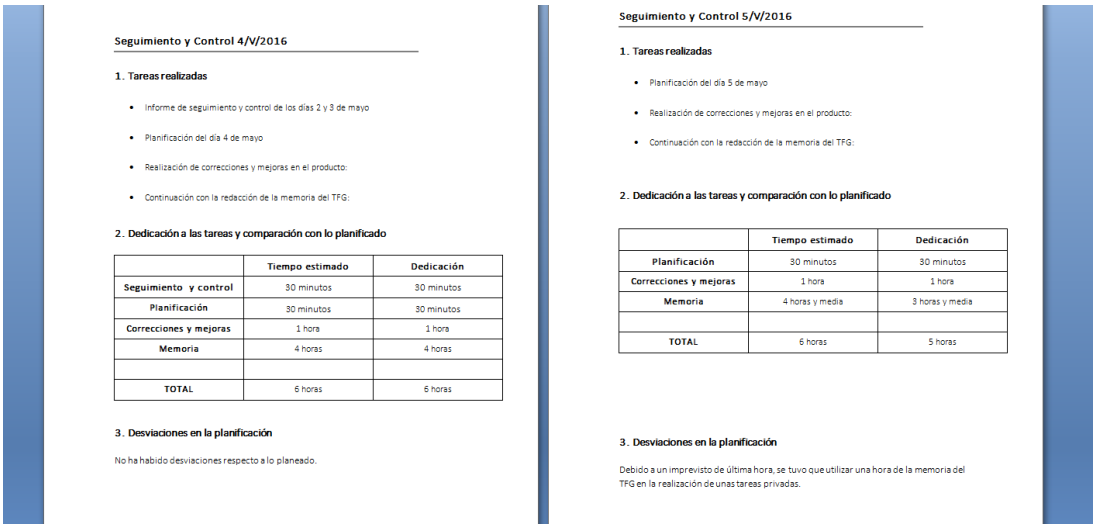
En esta fase se tuvieron que hacer diversos cambios en el calendario planificado y en la gestión de tiempos, así como eliminar ciertos aspectos que iba a incluir el producto, ya sea por falta de tiempo o por tener un costo demasiado alto.

Primeramente la fecha crítica del viaje de fin de carrera se tuvo que desplazar a finales de la semana 16(9/V) hasta principios de la semana 18(23/V) por cambios de última hora en la planificación de éste.

Por otro lado, como se dijo anteriormente, la gestión del tiempo empleado que se planificó era errónea. Mientras que al principio se siguió el plan de realizar 7 horas semanales durante la implementación del producto, más tarde se vio que era insuficiente y que se iba a necesitar más tiempo, por lo que éste se aumentó, pasando a dedicar 6 horas

diarias entre semana y 8 horas los sábados para mejorar el proyecto lo máximo posible. Finalmente la dedicación ha alcanzado las 250 horas en el momento de finalización de esta memoria y teniendo en cuenta que para la preparación de la defensa pública del proyecto se hará uso de aproximadamente 20 horas, lo que suman 270 horas para la realización total del proyecto.

Para controlar el progreso se ha realizado cada dos días un informe de seguimiento y control, como el siguiente:



Seguimiento y Control 4/V/2016

1. Tareas realizadas

- Informe de seguimiento y control de los días 2 y 3 de mayo
- Planificación del día 4 de mayo
- Realización de correcciones y mejoras en el producto:
- Continuación con la redacción de la memoria del TFG:

2. Dedicación a las tareas y comparación con lo planificado

	Tiempo estimado	Dedicación
Seguimiento y control	30 minutos	30 minutos
Planificación	30 minutos	30 minutos
Correcciones y mejoras	1 hora	1 hora
Memoria	4 horas	4 horas
TOTAL	6 horas	6 horas

3. Desviaciones en la planificación

No ha habido desviaciones respecto a lo planeado.

Seguimiento y Control 5/V/2016

1. Tareas realizadas

- Planificación del día 5 de mayo
- Realización de correcciones y mejoras en el producto:
- Continuación con la redacción de la memoria del TFG:

2. Dedicación a las tareas y comparación con lo planificado

	Tiempo estimado	Dedicación
Planificación	30 minutos	30 minutos
Correcciones y mejoras	1 hora	1 hora
Memoria	4 horas y media	3 horas y media
TOTAL	6 horas	5 horas

3. Desviaciones en la planificación

Debido a un imprevisto de última hora, se tuvo que utilizar una hora de la memoria del TFG en la realización de unas tareas privadas.

Figura 49. Ejemplo de informe de seguimiento y control realizado a lo largo del proyecto

Por último, de las funcionalidades planificadas que iba a tener el producto se ha tenido que descartar alguna por falta de tiempo. Las elecciones descartadas han sido la de implementar distintas interfaces y la de implementar ajustes opcionales dentro del juego, como fuese el lenguaje o el volumen del juego.

En una futura continuación del producto se introducirán, pero actualmente la necesidad de aprender la herramienta de desarrollo y distintas mecánicas de desarrollo de videojuegos han llevado más tiempo del planificado y han imposibilitado su implementación.

9.4. Fase de cierre

La fase de cierre del proyecto se basa en la elaboración de esta memoria, donde se recoge toda la información del proceso de desarrollo del proyecto, y en la elaboración de una presentación de éste mediante diapositivas para su posterior uso en la defensa pública.

Una vez realizada con éxito la presentación se dará por cerrado el proyecto.

10

Conclusiones

Aquí se recogen las conclusiones extraídas del trabajo realizado durante este trabajo de fin de grado y el posible futuro de éste.

10.1. Conclusión

Han sido varios meses de duro trabajo y aprendizaje y gran cantidad de horas que he pasado sin progresar debido a errores cuyo origen desconocía. Es lo que tienen estos tipos de trabajos, donde el factor de necesidad de aprender una nueva herramienta y una nueva forma de trabajar consumen gran parte del tiempo.

Sin embargo, Firework Madness resulta finalmente como un proyecto exitoso. Pese a seguir siendo un software en fase "alpha", cumple los objetivos que se pusieron al inicio de la producción. Es un juego utilizable y disfrutable, pese a que haya mejoras posibles pendientes de aplicar que aumentarían la calidad de éste.

Durante el desarrollo se ha cumplido el objetivo de aprendizaje de la herramienta Unity y de las técnicas y dinámicas de desarrollo de videojuegos.

Además se ha aplicado exitosamente técnicas de desarrollo de software aprendidas durante la carrera, como son el patrón MVC adaptado, que se ha enseñado y usado en varias asignaturas como "Ingeniería del Software 1" e "Ingeniería del Software 2" o la técnica de entrevistas con usuarios para localizar errores y mejoras que se enseña en la asignatura de "Interacción Persona Computador".

Por último, Firework Madness cuenta con un manual de usuario completamente funcional donde se explica el proceso de descarga e instalación en el dispositivo móvil y otro en el que se explica el funcionamiento completo de la aplicación.

Así pues, como ya se ha dicho anteriormente, se concluye que el proyecto cumple con los objetivos propuestos al inicio de éste y como resultado se cuenta con un ejemplo que sirve de base para el desarrollo de videojuegos mediante dinámicas y herramientas de trabajo propias de un ingeniero de software, que se puede reutilizar para futuros desarrollos más complejos o para una posible ampliación del mismo.

10.2. Líneas futuras

Debido al contexto del proyecto, es decir, un trabajo de fin de grado con unos recursos y un tiempo finitos, éste ha tenido que llegar a su fase de cierre y al fin del mismo. Sin embargo, el ciclo de vida del proyecto no termina aquí ya que como todo producto software cuenta con un proceso de evolución y mejora continua si es que el desarrollador así lo desea.

De esta forma se abren dos caminos para el futuro: mejorar y extender la calidad de este propio proyecto o reutilizar elementos aprendidos en éste para el desarrollo de un nuevo proyecto si se desea.

Si se decidiese por mejorar este mismo proyecto el plan de entrevistas con usuarios realizado concluyó con una serie de mejoras y sugerencias que se podrían aplicar para aumentar la calidad del producto. Además queda pendiente la transformación de la arquitectura local en la que se basa ahora el producto en una arquitectura cliente-servidor que proporcione un factor social y competitivo a éste.

Por otro lado, si se escogiese la opción de crear un nuevo juego desde cero siempre se podrían reutilizar métodos usados durante este proyecto. Por ejemplo, el modelo AMVCC aplicado en la implementación de este producto o la metodología usada en el plan de entrevistas en busca de *feedback* son elementos perfectamente reutilizables en el desarrollo de cualquier otro tipo de juego.

11

Bibliografía

En este capítulo final se recogen los libros y sitios web usados durante el desarrollo del proyecto.

11.1. Libros

- Power-Ups Conviértete en un profesional de los videojuegos. Juan P. Ordóñez, Plan B, 2013.
- Level Up! The Guide to Great Video Game Design. Scott Rogers, John Wiley & sons, 2014.
- DEV. Libro Blanco Del Desarrollo Español de Videojuegos 2015 http://www.dev.org.es/images/stories/docs/libro%20blanco%20videojuegos%202015%20final%20low.pdf

11.2. Sitios web

- https://unity3d.com/es
- https://unity3d.com/es/learn/tutorials
- http://forum.unity3d.com/#unity-community-support.4
- http://docs.unity3d.com/ScriptReference/
- http://www.uml.org/
- http://www.indiegamemusic.com/
- http://www.freesound.org/
- https://es.wikipedia.org/wiki/Wikipedia:Portada
- https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development
- http://businesstech.co.za/news/lifestyle/88472/the-biggest-entertainment-markets-in-the-world/

Anexo : Instalación

En este anexo se especifica cómo puede el usuario descargar el producto e instalarlo en su propio dispositivo móvil.

Descarga de la APK

El primer paso para el usuario será descargarse la aplicación. La forma oficial y usual de descargar cualquier producto para móviles es a través de la *Market* de Android, que permite además la instalación automática una vez descargada la aplicación.

Sin embargo este método suele reservarse a productos finalizados. Dado que nuestro producto todavía está en fase de desarrollo y mejora no conviene publicarlo en el estado actual, ya que podría dar una mala imagen al público. Así pues, la descarga se hará a través de un repositorio suministrado por el desarrollador personalmente a petición del usuario que quiera probarlo.

Como repositorio de descarga de la apk se ha optado por Mega ©, dado que ya se tiene experiencia en su uso. Para descargar la apk basta con copiar el siguiente enlace en su navegador de internet preferido y clickar en "Descargar a través de su navegador":

https://mega.nz/#!FQhH3CLI!9zwOJCJ0BiVWR2M8O3bZseQHRoQb6PvXhSqB0MVh0yQ

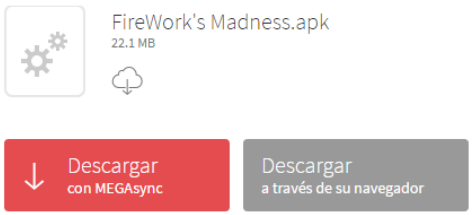


Figura 50. Descarga de la apk

Una vez terminada la descarga, se podrá encontrar la apk en la carpeta para descargas que tenga establecida en su navegador (por defecto en windows: \User\Descargas).

Transferir la APK al dispositivo móvil

Si la APK ha sido descargada directamente desde su smartphone puede pasar al siguiente capítulo directamente. De lo contrario, siga leyendo.

Una vez descargada la APK será necesario transferir ésta a su dispositivo móvil. Para ello necesitará de un cable USB con el que conectar su móvil al PC, en el cual deberán estar instalados los drivers de su móvil. Si no estuviesen instalados, instálelos antes (normalmente viene un Cd con ellos en la caja del móvil).

A continuación conecte el móvil al PC con el cable y en el momento en el que el móvil lo solicite, active el modo de almacenamiento masivo.

Figura 51. Activando el almacenamiento masivo en un móvil Android

El siguiente paso es transferir el archivo APK descargado a su móvil. Para ello arrástrelo mediante el administrador de archivos de Windows. Conviene crear una carpeta dentro de su móvil para recordar más fácilmente su ubicación.

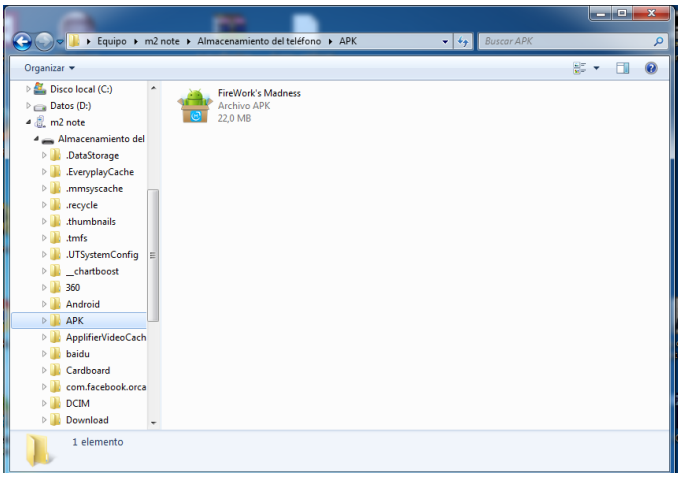


Figura 52. Transfiriendo la APK al móvil.

Instalación de la APK manualmente

El último paso es instalar la APK desde el móvil, pero antes debemos indicarle a Android que permita la instalación de programas fuera de Google Play, es decir fuera de la Market de Android.

Para ello nos desplazamos hasta **Ajustes/Seguridad/Orígenes desconocidos** y activamos esta opción.

Figura 53. Activar la instalación de apks de origen desconocidos

Ahora estamos en condiciones de instalar la aplicación. Abrimos el administrador de archivos que tengamos en el móvil (algunos móviles no cuentan con uno de serie, por lo que se deberá instalar uno de las múltiples opciones que puede encontrar en la Market, recomendamos "ES Explorador de Archivos"), accedemos a la carpeta donde habíamos guardado la APK descargada y la ejecutamos.

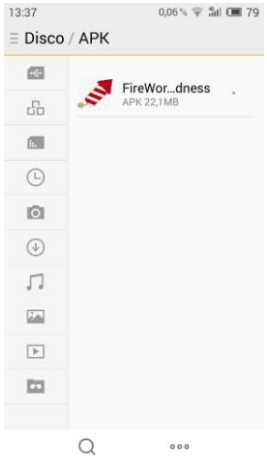


Figura 54. La apk vista desde el explorador de archivos de Android

Cuando el proceso termine, Firework Madness ya estará instalado en su dispositivo para jugar.

```
using UnityEngine;
using System.Collections;

public class BloqueModelo : BloqueElement {

    public GameObject[] grupos;
    public GameObject[] cohetes;
    public GameObject cohete1;
    public GameObject cohete2;

    public TablaApp Tabla;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        StartCoroutine ("EsperarYSpawnear");
    }

    public void MoverseIzquierda(){
        if (CheckMove ("izquierda")) {
            Vector3 nuevaPos1 = new Vector3 (cohete1.transform.position.x - 1, cohete1.transform.position.y, cohete1.transform.position.z);
            Vector3 nuevaPos2 = new Vector3 (cohete2.transform.position.x - 1, cohete2.transform.position.y, cohete2.transform.position.z);
            Destroy (cohete1);
            Destroy (cohete2);
            cohete1 = Instantiate (cohetes[0], nuevaPos1, Quaternion.identity) as GameObject;
            cohete2 = Instantiate (cohetes[1], nuevaPos2, Quaternion.identity) as GameObject;


        }
    }



    public void MoverseDerecha(){
        if (CheckMove ("derecha")) {


            Vector3 nuevaPos1 = new Vector3 (cohete1.transform.position.x + 1, cohete1.transform.position.y, cohete1.transform.position.z);
            Vector3 nuevaPos2 = new Vector3 (cohete2.transform.position.x + 1, cohete2.transform.position.y, cohete2.transform.position.z);
            Destroy (cohete1);
            Destroy (cohete2);
            cohete1 = Instantiate (cohetes[0], nuevaPos1, Quaternion.identity) as GameObject;
            cohete2 = Instantiate (cohetes[1], nuevaPos2, Quaternion.identity) as GameObject;

        }
    }
```

Figura 37.1. Extracto del script "BloqueModelo"

```
    public void MoverseAbajo(){

        if (CheckMove("abajo")) { //Si no pegamos con nada nos movemos hacia abajo

            Vector3 nuevaPos1 = new Vector3 (cohete1.transform.position.x, cohete1.transform.position.y-1, cohete1.transform.position.z);
            Vector3 nuevaPos2 = new Vector3 (cohete2.transform.position.x, cohete2.transform.position.y-1, cohete2.transform.position.z);
            Destroy (cohete1);
            Destroy (cohete2);
            cohete1 = Instantiate (cohetes[0], nuevaPos1, Quaternion.identity) as GameObject;
            cohete2 = Instantiate (cohetes[1], nuevaPos2, Quaternion.identity) as GameObject;


        } else {
            //Estamos pegando con algo, pararse y marcar la localizaciónn del bloque actual como lleno en la tabla, además destruir el pivote de rotacion

            //llenar la tabla con esos bloques FALTA
            Tabla.modelo.tabla[Mathf.RoundToInt(cohete1.transform.position.x),Mathf.RoundToInt(cohete1.transform.position.y)]=cohete1.GetComponent<Cohete>();
            Tabla.modelo.tabla[Mathf.RoundToInt(cohete2.transform.position.x),Mathf.RoundToInt(cohete2.transform.position.y)]=cohete2.GetComponent<Cohete>();

            Destroy (this.gameObject);
            //booleano para indicar que hemos terminado de colocar el bloque y tabla.c debe comprobar posibles matches
            Constants.cayendo=false;

            if(!Constants.gameOver){
                Constants.spawn = true; //Spawnear nuevo bloque
            }
        }

    }
```

Figura 37.2. Extracto del script "BloqueModelo"

```
81  public void Rotar(){
82
83      Vector3 nuevaPos1=new Vector3(0,0,0);
84      bool movimiento = false;
85      if(cohete1.transform.position.y>cohete2.transform.position.y){ //0º->90º
86          if (CheckMove ("rotacion1")) {
87              movimiento = true;
88              nuevaPos1 = new Vector3 (cohete1.transform.position.x + 1, cohete1.transform.position.y - 1, cohete1.transform.position.z);
89          }
90      }else if(cohete1.transform.position.x>cohete2.transform.position.x){//90º->180º
91          if (CheckMove ("rotacion2")) {
92              movimiento = true;
93              nuevaPos1 = new Vector3 (cohete1.transform.position.x - 1, cohete1.transform.position.y - 1, cohete1.transform.position.z);
94          }
95      }else if(cohete1.transform.position.y<cohete2.transform.position.y){//180º->270º
96          if (CheckMove ("rotacion3")) {
97              movimiento = true;
98              nuevaPos1 = new Vector3 (cohete1.transform.position.x - 1, cohete1.transform.position.y + 1, cohete1.transform.position.z);
99          }
100     }else if(cohete1.transform.position.x<cohete2.transform.position.x){//270º->0º
101         if (CheckMove ("rotacion4")) {
102             movimiento = true;
103             nuevaPos1 = new Vector3 (cohete1.transform.position.x + 1, cohete1.transform.position.y + 1, cohete1.transform.position.z);
104         }
105     }
106     if (movimiento) {
107         Destroy (cohete1);
108         cohete1 = Instantiate (cohetes [0], nuevaPos1, Quaternion.identity) as GameObject;
109     }
110
111 }
112
113
```

Figura 37.3. Extracto del script "BloqueModelo"

```
115     public bool CheckMove(string dir){
116
117         int cohete1X = Mathf.RoundToInt( cohete1.transform.position.x);
118         int cohete2X = Mathf.RoundToInt( cohete2.transform.position.x);
119         int cohete1Y =  Mathf.RoundToInt( cohete1.transform.position.y);
120         int cohete2Y = Mathf.RoundToInt( cohete2.transform.position.y);
121
122         if (dir == "izquierda") {
123             cohete1X--;
124             cohete2X--;
125             if ((cohete1X == -1) || (cohete2X == -1)) {
126
127                 return false;
128             } else if ((Tabla.modelo.tabla [cohete1X, cohete1Y] != null)||(Tabla.modelo.tabla [cohete2X, coh
129                 return false;
130         } else if (dir == "derecha") {
131             cohete1X++;
132             cohete2X++;
133             if ((cohete1X == Constants.columnas) || (cohete2X == Constants.columnas)) {
134                 return false;
135             } else if ((Tabla.modelo.tabla [cohete1X, cohete1Y] != null) ||
136                 (Tabla.modelo.tabla [cohete2X, cohete2Y] != null))
137                 return false;
138
139         } else if (dir == "rotacion1") { //0º->90º
140             cohete1X++;
141             cohete1Y--;
142             if ((cohete1X == Constants.columnas) || (cohete1Y == -1)) {
143                 return false;
144             } else if (Tabla.modelo.tabla [cohete1X, cohete1Y] != null)
145                 return false;
146
147
148         }else if (dir == "rotacion2") { //90º->180º
149             cohete1X--;
150             cohete1Y--;
151             if ((cohete1X == -1) || (cohete1Y == -1)) {
152                 return false;
153             } else if (Tabla.modelo.tabla [cohete1X, cohete1Y] != null)
154                 return false;
155
156         }else if (dir == "rotacion3") {//180º->270º
157             cohete1X--;
158             cohete1Y++;
159             if (cohete1X == -1) {
160                 return false;
161             } else if (Tabla.modelo.tabla [cohete1X, cohete1Y] != null)
162                 return false;
163
```

Figura 37.4. Extracto del script "BloqueModelo"

```
                return false;


        }else if (dir == "rotacion4") {//270º->0º
            cohete1X++;
            cohete1Y++;

            if ((cohete1X == Constants.columnas)) {
                return false;
            } else if (Tabla.modelo.tabla [cohete1X, cohete1Y] != null)
                return false;



        }else {

            cohete1Y--;
            cohete2Y--;
            if ((cohete1Y == -1) || (cohete2Y == -1)) {
                return false;
                //ultima fila
            }else if ((Tabla.modelo.tabla [cohete1X, cohete1Y] != null) ||
                (Tabla.modelo.tabla [cohete2X, cohete2Y] != null))
                return false;
        }
        return true;
    }

    public void SpawnJugador(){
        if (Constants.spawn == true) {
            int i = Random.Range (0, grupos.Length);

            Instantiate (grupos [i], transform.position, Quaternion.identity);    //transform.position es la posicion del spawner
        }
        Constants.spawn=false;

    }

    IEnumerator EsperarYSpawnear(){
        yield return new WaitForSeconds (0.5f);
        SpawnJugador();

    }



    IEnumerator esperarMovimiento(){
        yield return new WaitForSeconds (5f);
    }
}
```

Figura 37.5. Extracto del script "BloqueModelo"

```
using UnityEngine;
using System.Collections;

public class BloqueControlador : BloqueElement {


    private Vector3 touchOrigin;
    private bool moviendo;
    private string movimiento="ninguno";

    void Update () {

        //this.transform.position = cohete2.transform.position;
        int nbTouches = Input.touchCount;
        if(nbTouches > 0)
        {

            Touch myTouch = Input.touches[0];

            if (myTouch.phase == TouchPhase.Began) {
                touchOrigin = myTouch.position;
                moviendo = true;
                Vector3 touchOriginConvertido= Camera.main.ScreenToWorldPoint (touchOrigin);
                if (touchOriginConvertido.x > this.transform.position.x+1 && touchOriginConvertido.y > 4) {
                    movimiento = "Derecha";
                } else if (touchOriginConvertido.x < this.transform.position.x-1 && touchOriginConvertido.y > 4) {
                    movimiento = "Izquierda";
                } else if ((touchOriginConvertido.x < this.transform.position.x+1)&&(touchOriginConvertido.x > this.transform.position.x-1)&&(touchOriginConvertido.y < this.transform.position.y+1)
                    && (touchOriginConvertido.y> this.transform.position.y-1)) {
                    movimiento = "Rotar";
                    //} else if (touchOriginConvertido.y < this.transform.position.y-2) {
                } else if (touchOriginConvertido.y < 4) {
                    movimiento = "Abajo";
                }
            }else if(myTouch.phase == TouchPhase.Ended){
                movimiento = "Ninguno";
                moviendo = false;
            }
        }
        if (moviendo) {
            // Move Left
            if ((Input.GetKeyDown (KeyCode.LeftArrow))||(movimiento=="Izquierda")) {
                app.modelo.MoverseIzquierda ();

            }// Move Right
            else if ((Input.GetKeyDown (KeyCode.RightArrow))||(movimiento=="Derecha")) {
                app.modelo.MoverseDerecha ();
            }// Rotate
```

Figura 38.1. Extracto del script "BloqueControlador"

```
            else if (movimiento=="Abajo") {  //pensar forma de que baje segun el tiempo

                app.modelo.MoverseAbajo ();
                Constants.tiempoUltimoDescenso = Time.time;

            } else if (Input.GetKeyDown (KeyCode.Space) || (movimiento=="Rotar")) {
                app.modelo.Rotar ();
            }
            moviendo = false;
            StartCoroutine ("esperarMovimiento");
        }
        if((Input.GetKeyDown (KeyCode.DownArrow) || Time.time - Constants.tiempoUltimoDescenso >=1)){
            app.modelo.MoverseAbajo ();
            Constants.tiempoUltimoDescenso = Time.time;
        }


    }
}
```

Figura 38.2. Extracto del script "BloqueControlador"

```
using UnityEngine;
using System.Collections;

public class Tabla : MonoBehaviour {


    public static int anchura= Constants.columnas;
    public static int  altura=Constants.filas;
    public static Cohete[,] tabla = new Cohete[anchura, altura];
    public static int filasRellenado;

    public GameObject[] animaciones_fuegos;
    public GameObject[] cohetes;
    public GameObject[] cohetesBonus4;
    public GameObject[] cohetesBonus5;

    private Animacion_Fuego anim;

    void Awake (){
        Constants.cayendo=true;
        Constants.comprobandoMatchs = false;
        Constants.iniciando=true;
        Constants.hayMatches=true;
        Constants.rellenando = true;
        Constants.tableroListoParaPrimeraComprobacion = false;
        Constants.primerColapso=false;
        Constants.colapsado=false;
        Constants.posado = false;
        Constants.gameOver = false;

        Constants.puntuacion=0;

        anim=this.GetComponent<Animacion_Fuego>();
    }


    void Start () {
        PlayerPrefs.SetInt ("Puntuacion",0);
        generarTabla ();
        RellenarTabla ();
        Constants.primerColapso = true;

    }
```

Figura 39.1. Extracto del script "Tabla"

```
    void Update(){
        //Comprobar condicion GAMEOVER TODO EL RATO

        if(comprobarGameOver()){
            PlayerPrefs.SetInt ("Puntuacion",Constants.puntuacion);
            SceneChanger.FinJuego();
        }else{
            colapsarTabla ();
            if ((Constants.comprobandoMatchs == false)&&(Constants.iniciando==false)){
                Constants.comprobandoMatchs = true;
                if(Constants.cayendo==false)
                    StartCoroutine (comprobarYEsperar ());
                Constants.comprobandoMatchs = false;

            }
        }
    }


    public  bool DentroTabla(Vector2 pos){
        return ((int)pos.x >= 0 && (int)pos.x < anchura && (int)pos.y >= 0);        //no es necesario comprobar y<altura porque los bloques no suben
    }


    public void Bonus4Fila(Cohete coheteBonus){

        Cohete CoheteAux;
        for(int x=0; x< anchura; x++){
            if(tabla [x, (int)coheteBonus.transform.position.y]!=null){

                CoheteAux = tabla [x, (int)coheteBonus.transform.position.y].GetComponent<Cohete> ();
                anim.ActivarAnimacion (CoheteAux);

                //Por algun motivo comprobar esto supone demasiada carga, pensar otra forma
                /*if (CoheteAux.Bonus == "Bonus4") {
                    StartCoroutine ("esperar");
                    Bonus4Fila (CoheteAux);
                } else if (CoheteAux.Bonus == "Bonus5") {
                    StartCoroutine ("esperar");
                    Bonus5Fila (CoheteAux);
                }*/
                Constants.puntuacion = Constants.puntuacion + CoheteAux.valor;
                Destroy (tabla [x, (int)coheteBonus.transform.position.y].gameObject);  //Sustituir por funcion que destruya y Ademas active la animacion de explosion y cuenta de puntos e
            }
        }
    }
```

Figura 39.2. Extracto del script "Tabla"

```
92    public  void Bonus5Fila(Cohete coheteBonus){
93        Cohete CoheteAux;
94
95        for(int x=0; x< anchura; x++){
96
97            if (tabla [x, (int)coheteBonus.transform.position.y] != null) {
98
99                CoheteAux = tabla [x, (int)coheteBonus.transform.position.y].GetComponent<Cohete> ();
100               anim.ActivarAnimacion (CoheteAux);
101               Constants.puntuacion = Constants.puntuacion + CoheteAux.valor;
102               Destroy (tabla [x, (int)coheteBonus.transform.position.y].gameObject);
103           }
104           if ((int)coheteBonus.transform.position.y - 1 >= 0) {
105               if (tabla [x, (int)coheteBonus.transform.position.y - 1] != null) {
106
107                   CoheteAux = tabla [x, (int)(coheteBonus.transform.position.y - 1)].GetComponent<Cohete> ();
108                   anim.ActivarAnimacion (CoheteAux);
109                   /*if (CoheteAux.Bonus == "Bonus4") {
110                       //Bonus4Fila (CoheteAux);
111                   } else if (CoheteAux.Bonus == "Bonus5") {
112                       //Bonus5Fila (CoheteAux);
113                   }*/
114                   Constants.puntuacion = Constants.puntuacion + CoheteAux.valor;
115                   Destroy (tabla [x, (int)(coheteBonus.transform.position.y - 1)].gameObject);
116               }
117           } else {    //Si el bonus esta en la ultima fila, borrará la superior en vez de la inferior
118               if (tabla [x, (int)coheteBonus.transform.position.y + 1] != null) {
119
120                   CoheteAux = tabla [x, (int)(coheteBonus.transform.position.y + 1)].GetComponent<Cohete> ();
121                   anim.ActivarAnimacion (CoheteAux);
122                   /*if (CoheteAux.Bonus == "Bonus4") {
123                       //Bonus4Fila (CoheteAux);
124                   } else if (CoheteAux.Bonus == "Bonus5") {
125                       //Bonus5Fila (CoheteAux);
126                   }*/
127                   Constants.puntuacion = Constants.puntuacion + CoheteAux.valor;
128                   Destroy (tabla [x, (int)(coheteBonus.transform.position.y + 1)].gameObject);
129               }
130           }
131       }
132   }
133
134
```

Figura 39.3. Extracto del script "Tabla"

```
135    public  void RellenarTabla(){
136        //int filas;
137
138        if (Constants.dificultad == "Facil") {
139            filasRellenado = 4;
140
141        } else if (Constants.dificultad == "Normal") {
142            filasRellenado = 7;
143
144        }else
145            filasRellenado = 10;
146
147        for (int y = 0; y <= filasRellenado; y++) {
148            for (int x = 0; x < anchura; x++) {
149
150                int relleno = Random.Range (0, 2);
151                if(relleno == 1){
152                    int i = Random.Range (0, cohetes.Length);
153                    GameObject coheteCreado = cohetes[i];
154
155                    transform.position = new Vector3 (x, y);
156                    GameObject goAux = Instantiate (coheteCreado, transform.position, Quaternion.identity) as GameObject;
157
158                    goAux.GetComponent<Cohete> ().Assign (goAux.GetComponent<Cohete> ().tipo, y, x, "Ninguno");
159                    tabla [x, y] = goAux.GetComponent<Cohete> ();
160
161                }
162            }
163        }
164
165    }
166
```

Figura 39.4. Extracto del script "Tabla"

```
    public void colapsarTabla(){ //HaciaAbjao

        for (int y = 1; y<altura; y++) {
            for (int x = 0; x <anchura; x++) {
                //if(y+1<altura){
                if ((tabla [x, y-1]==null)&&(tabla[x,y]!=null)) {   //si debajo de un cohete hay hueco vacio

                    tabla [x, y - 1] = tabla [x, y];
                    tabla [x, y] = null;
                    tabla [x, y - 1].transform.position += new Vector3 (0, -1, 0);
                }
                //}
            }
        }
        Constants.colapsado = true;
        if(Constants.primerColapso==true){
            StartCoroutine ("EvitarMatchesInicializando", filasRellenado);
            Constants.primerColapso = false;

        }
    }
```

Figura 39.5. Extracto del script "Tabla"

```
IEnumerator EvitarMatchesInicializando(int filas){
    yield return new WaitForSeconds (0.1f);
    string tipoSuperior;
    string tipoHorizontal;
    if (Constants.colapsado) {
        for (int y = 0; y <= filas; y++) {
            for (int x = 0; x < anchura; x++) {
                bool sustituir = false;

                int i = 0;
                if (tabla [x, y] != null) {
                    GameObject coheteCreado = tabla [x, y].gameObject;
                    tipoSuperior="Vacio1";
                    tipoHorizontal="Vacio2";
                    if ((y + 1 < altura) && (y + 2 < altura)) {

                        if ((tabla [x, y] != null) && (tabla [x, (int)(y + 1.1)] != null) && (tabla [x, (int)(y + 2.2)] != null)) {

                            if (tabla [x, y + 1].GetComponent<Cohete> ().tipo == tabla [x, y + 2].GetComponent < Cohete> ().tipo) {
                                tipoSuperior = tabla [x, (int)(y + 1.1)].GetComponent<Cohete> ().tipo;
                            }
                        }
                    }
                    if ((x + 1 < anchura) && (x + 2 < anchura)) {

                        if ((tabla [x, y] != null) && (tabla [(int)(x + 1.1), y] != null) && (tabla [(int)(x + 2.2), y] != null)) {
                            if (tabla [(int)(x + 1.1), y].GetComponent<Cohete> ().tipo == tabla [(int)(x + 2.2), y].GetComponent < Cohete> ().tipo) {
                                tipoHorizontal = tabla [(int)(x + 2.2), y].GetComponent<Cohete> ().tipo;
                            }
                        }
                    }
                    while ((tabla [x, y].GetComponent<Cohete> ().tipo == tipoHorizontal) || (tabla [x, y].GetComponent<Cohete> ().tipo == tipoSuperior)) {
                        i = Random.Range (0, cohetes.Length);
                        coheteCreado = cohetes [i];
                        if ((coheteCreado.GetComponent<Cohete> ().tipo != tipoHorizontal)&&(coheteCreado.GetComponent<Cohete> ().tipo != tipoSuperior)) {
                            Destroy (tabla [x, y].gameObject);
                            transform.position = new Vector3 (x, y, 0);
                            GameObject goAux = Instantiate (coheteCreado, transform.position, Quaternion.identity) as GameObject;
                            goAux.GetComponent<Cohete> ().Assign (goAux.GetComponent<Cohete> ().tipo, y, x,"Ninguno");
                            tabla [x, y] = goAux.GetComponent<Cohete> ();

                        }
                    }
                }
            }
        }
    }
    Constants.rellenando = false;
    Constants.iniciando = false;
    Constants.spawn = true;
}
```

Figura 39.6. Extracto del script "Tabla"

```
240
241     private void generarTabla(){  //vacia, simplemente transparente
242
243         for (int y = 0; y < altura; y++) {
244             for (int x = 0; x < anchura; x++) {
245                 tabla[x,y]=null;
246             }
247         }
248     }
249
250     IEnumerator comprobarYEsperar(){
251         if (comprobarMatch5()==false) {
252             if (comprobarMatch4() == false) {
253                 comprobarMatch3 ();
254             }
255         }
256         yield return new WaitForSeconds (1);
257
258
259     }
260
261
```

Figura 39.7. Extracto del script "Tabla"

```
263 private bool comprobarMatch3(){
264     bool match = false;
265
266     for (int y = 0; y < altura; y++) {
267         for (int x = 0; x < anchura; x++) {
268
269             if ((y + 1 < altura) && (y + 2 < altura)) {
270
271                 if ((tabla [x, y] != null) && (tabla [x, y + 1] != null) && (tabla [x, y + 2] != null)) {
272
273                     if (mismoTipo3 (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2])) {//comprobacion vertical
274                         destruirMatch3YPuntuar (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2]);
275
276                         match = true;
277
278                     }
279                 }
280             }
281             if ((x + 1 < anchura) && (x + 2 < anchura)) {
282
283                 if ((tabla [x, y] != null) && (tabla [x + 1, y] != null) && (tabla [x + 2, y] != null)) {
284
285                     if (mismoTipo3 (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y])) {//comprobacion horizontal
286                         destruirMatch3YPuntuar (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y]);
287
288                         match=true;
289                     }
290                 }
291             }
292         }
293     }
294     return match;
295
296 }
297
```

Figura 39.8. Extracto del script "Tabla"

```
298     private bool comprobarMatch4(){
299         bool match = false;
300 
301         for (int y = 0; y < altura; y++) {
302             for (int x = 0; x < anchura; x++) {
303 
304                 if ((y + 1 < altura) && (y + 2 < altura)&& (y + 3 < altura)) {
305 
306                     if ((tabla [x, y] != null) && (tabla [x, y + 1] != null) && (tabla [x, y + 2] != null)&& (tabla [x, y + 3] != null)) {
307 
308                         if (mismoTipo4 (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2], tabla [x, y + 3])) {//comprobacion vertical
309                             destruirMatch4CrearBonusYPuntuar (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2], tabla [x, y + 3]);
310 
311                             match = true;
312 
313                         }
314                     }
315                 }
316                 if ((x + 1 < anchura) && (x + 2 < anchura)&& (x + 3 < anchura)) {
317 
318                     if ((tabla [x, y] != null) && (tabla [x + 1, y] != null) && (tabla [x + 2, y] != null) && (tabla [x + 3, y] != null)) {
319 
320                         if (mismoTipo4 (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y], tabla [x + 3, y])) {//comprobacion horizontal
321                             destruirMatch4CrearBonusYPuntuar (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y], tabla [x + 3, y]);
322 
323                             match=true;
324                         }
325                     }
326                 }
327             }
328         }
329         return match;
330 
331     }
332 
```

Figura 39.9. Extracto del script "Tabla"

```
333     private bool comprobarMatch5(){
334         bool match = false;
335 
336         for (int y = 0; y < altura; y++) {
337             for (int x = 0; x < anchura; x++) {
338 
339                 if ((y + 1 < altura) && (y + 2 < altura)&&(y + 3 < altura)&&(y + 4 < altura)) {
340 
341                     if ((tabla [x, y] != null) && (tabla [x, y + 1] != null) && (tabla [x, y + 2] != null)&& (tabla [x, y + 3] != null)&& (tabla [x, y + 4] != null)) {
342 
343                         if (mismoTipo5 (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2], tabla [x, y + 3], tabla [x, y + 4])) {//comprobacion vertical
344                             destruirMatch5CrearBonusYPuntuar (tabla [x, y], tabla [x, y + 1], tabla [x, y + 2], tabla [x, y + 3], tabla [x, y + 4]);
345 
346                             match = true;
347 
348                         }
349                     }
350                 }
351                 if ((x + 1 < anchura) && (x + 2 < anchura)&& (x + 3 < anchura)&& (x + 4 < anchura)) {
352 
353                     if ((tabla [x, y] != null) && (tabla [x + 1, y] != null) && (tabla [x + 2, y] != null) && (tabla [x + 3, y] != null) && (tabla [x + 4, y] != null)) {
354 
355                         if (mismoTipo5 (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y], tabla [x + 3, y], tabla [x + 4, y])) {//comprobacion horizontal
356                             destruirMatch5CrearBonusYPuntuar (tabla [x, y], tabla [x + 1, y], tabla [x + 2, y],tabla [x + 3, y], tabla [x + 4, y]);
357 
358                             match=true;
359                         }
360                     }
361                 }
362             }
363         }
364         return match;
365 
366     }
367 
368     private static bool mismoTipo3(Cohete cohete1, Cohete cohete2, Cohete cohete3){
369 
370         if ((cohete1.tipo == cohete2.tipo) && (cohete2.tipo == cohete3.tipo))
371             return true;
372         else
373             return false;
374 
375     }
```

Figura 39.10. Extracto del script "Tabla"

```
376     private static bool mismoTipo4(Cohete cohete1, Cohete cohete2, Cohete cohete3,Cohete cohete4){
377
378         if ((cohete1.tipo == cohete2.tipo) && (cohete2.tipo == cohete3.tipo)&& (cohete3.tipo == cohete4.tipo))
379             return true;
380         else
381             return false;
382
383     }
384     private static bool mismoTipo5(Cohete cohete1, Cohete cohete2, Cohete cohete3,Cohete cohete4,Cohete cohete5){
385
386         if ((cohete1.tipo == cohete2.tipo) && (cohete2.tipo == cohete3.tipo)&& (cohete3.tipo == cohete4.tipo)&& (cohete4.tipo == cohete5.tipo))
387             return true;
388         else
389             return false;
390
391     }
392
393
394     private  void destruirMatch3YPuntuar(Cohete cohete1, Cohete cohete2, Cohete cohete3){
395
396
397         if (cohete1.Bonus == "Bonus4") {
398             Bonus4Fila (cohete1);
399         } else if (cohete1.Bonus == "Bonus5") {
400             Bonus5Fila (cohete1);
401         } else if (cohete2.Bonus == "Bonus4") {
402             Bonus4Fila (cohete2);
403         } else if (cohete2.Bonus == "Bonus5") {
404             Bonus5Fila (cohete2);
405         } else if (cohete3.Bonus == "Bonus4") {
406             Bonus4Fila (cohete3);
407         } else if (cohete3.Bonus == "Bonus5") {
408             Bonus5Fila (cohete3);
409         } else {
410             anim.ActivarAnimacion (cohete1);
411             anim.ActivarAnimacion (cohete2);
412             anim.ActivarAnimacion (cohete3);
413             Constants.puntuacion = Constants.puntuacion + cohete1.valor + cohete2.valor + cohete3.valor;
414             Destroy (cohete1.gameObject);
415             Destroy (cohete2.gameObject);
416             Destroy (cohete3.gameObject);
417         }
418
419     }
```

Figura 39.11. Extracto del script "Tabla"

```
420     private  void destruirMatch4CrearBonusYPuntuar(Cohete cohete1, Cohete cohete2, Cohete cohete3,Cohete cohete4){
421         Vector3 posBonus = cohete1.transform.position;
422
423         GameObject coheteBonus4;
424         if (cohete1.tipo == "Rojo") {
425             coheteBonus4 = cohetesBonus4 [2];
426         } else if (cohete1.tipo == "Amarillo") {
427             coheteBonus4 = cohetesBonus4 [0];
428         } else if (cohete1.tipo == "Azul") {
429             coheteBonus4 = cohetesBonus4 [1];
430         } else {
431             coheteBonus4 = cohetesBonus4 [3];
432         }
433
434         if (cohete1.Bonus == "Bonus4") {
435             Bonus4Fila (cohete1);
436         } else if (cohete1.Bonus == "Bonus5") {
437             Bonus5Fila (cohete1);
438         } else if (cohete2.Bonus == "Bonus4") {
439             Bonus4Fila (cohete2);
440         } else if (cohete2.Bonus == "Bonus5") {
441             Bonus5Fila (cohete2);
442         } else if (cohete3.Bonus == "Bonus4") {
443             Bonus4Fila (cohete3);
444         } else if (cohete3.Bonus == "Bonus5") {
445             Bonus5Fila (cohete3);
446         } else if (cohete4.Bonus == "Bonus4") {
447             Bonus4Fila (cohete4);
448         } else if (cohete4.Bonus == "Bonus5") {
449             Bonus5Fila (cohete4);
450         } else {
451             anim.ActivarAnimacion (cohete1);
452             anim.ActivarAnimacion (cohete2);
453             anim.ActivarAnimacion (cohete3);
454             anim.ActivarAnimacion (cohete4);
455             Constants.puntuacion = Constants.puntuacion + cohete1.valor + cohete2.valor + cohete3.valor + cohete4.valor;
456             Destroy (cohete1.gameObject);
457             Destroy (cohete2.gameObject);
458             Destroy (cohete3.gameObject);
459             Destroy (cohete4.gameObject);
460             GameObject goAux = Instantiate (coheteBonus4, posBonus, Quaternion.identity) as GameObject;
461             goAux.GetComponent<Cohete> ().Assign (goAux.GetComponent<Cohete> ().tipo, (int)posBonus.y, (int)posBonus.x, "Bonus4");
462             tabla [(int)posBonus.x, (int)posBonus.y] = goAux.GetComponent<Cohete> ();
463             Constants.puntuacion = Constants.puntuacion + Constants.puntuacionMatch4;
464         }
465
466     }
```

Figura 39.12. Extracto del script "Tabla"

```
    }
    private  void destruirMatch5CrearBonusYPuntuar(Cohete cohete1, Cohete cohete2, Cohete cohete3,Cohete cohete4,Cohete cohete5){
        Vector3 posBonus = cohete1.transform.position;
        GameObject coheteBonus5;
        if (cohete1.tipo == "Rojo") {
            coheteBonus5 = cohetesBonus5 [2];
        } else if (cohete1.tipo == "Amarillo") {
            coheteBonus5 = cohetesBonus5 [0];
        } else if (cohete1.tipo == "Azul") {
            coheteBonus5 = cohetesBonus5 [1];
        } else {
            coheteBonus5 = cohetesBonus5 [3];
        }

        if (cohete1.Bonus == "Bonus4") {
            Bonus4Fila (cohete1);
        } else if (cohete1.Bonus == "Bonus5") {
            Bonus5Fila (cohete1);
        } else if (cohete2.Bonus == "Bonus4") {
            Bonus4Fila (cohete2);
        } else if (cohete2.Bonus == "Bonus5") {
            Bonus5Fila (cohete2);
        } else if (cohete3.Bonus == "Bonus4") {
            Bonus4Fila (cohete3);
        } else if (cohete3.Bonus == "Bonus5") {
            Bonus5Fila (cohete3);
        } else if (cohete4.Bonus == "Bonus4") {
            Bonus4Fila (cohete4);
        } else if (cohete4.Bonus == "Bonus5") {
            Bonus5Fila (cohete4);
        } else if (cohete5.Bonus == "Bonus4") {
            Bonus4Fila (cohete5);
        } else if (cohete5.Bonus == "Bonus5") {
            Bonus5Fila (cohete5);
        } else {
            anim.ActivarAnimacion (cohete1);
            anim.ActivarAnimacion (cohete2);
            anim.ActivarAnimacion (cohete3);
            anim.ActivarAnimacion (cohete4);
            anim.ActivarAnimacion (cohete5);
            Constants.puntuacion = Constants.puntuacion + cohete1.valor + cohete2.valor + cohete3.valor + cohete4.valor + cohete5.valor;
            Destroy (cohete1.gameObject);
            Destroy (cohete2.gameObject);
            Destroy (cohete3.gameObject);
            Destroy (cohete4.gameObject);
            Destroy (cohete5.gameObject);
            GameObject goAux = Instantiate (coheteBonus5, posBonus, Quaternion.identity) as GameObject;
            goAux.GetComponent<Cohete> ().Assign (goAux.GetComponent<Cohete> ().tipo, (int)posBonus.y, (int)posBonus.x, "Bonus5");
            tabla [(int)posBonus.x, (int)posBonus.y] = goAux.GetComponent<Cohete> ();
            Constants.puntuacion = Constants.puntuacion + Constants.puntuacionMatch5;
        }
    }
```

Figura 39.13. Extracto del script "Tabla"

```
    public void ActivarAnimacion(Cohete coheteExplotado){
        GameObject animacion;
        float VelocidadDeSerie=animaciones_fuegos [0].GetComponent<ParticleSystem> ().startSpeed;
        for (int i = 0; i < 4; i++) {
            animaciones_fuegos [i].GetComponent<ParticleSystem> ().startSpeed=animaciones_fuegos [i].GetComponent<ParticleSystem> ().startSpeed-coheteExplotado.transform.position.y;
        }

        if (coheteExplotado.tipo == "Amarillo") {

            animacion=Instantiate (animaciones_fuegos[0], coheteExplotado.transform.position,Quaternion.Euler(270,0,0)) as GameObject;
        //  Debug.Log (animacion.name);
        } else if (coheteExplotado.tipo == "Rojo") {
            animacion=Instantiate (animaciones_fuegos[2], coheteExplotado.transform.position, Quaternion.Euler(270,0,0)) as GameObject;
        }else if (coheteExplotado.tipo == "Azul") {
            animacion=Instantiate (animaciones_fuegos[1], coheteExplotado.transform.position, Quaternion.Euler(270,0,0)) as GameObject;
        }else{
            animacion=Instantiate (animaciones_fuegos[3], coheteExplotado.transform.position, Quaternion.Euler(270,0,0)) as GameObject;
        }
        for (int i = 0; i < 4; i++) {
            animaciones_fuegos [i].GetComponent<ParticleSystem> ().startSpeed = VelocidadDeSerie;
        }
        StartCoroutine ("esperarAnimacion");
    }

    IEnumerator esperarAnimacion(){
        yield return new WaitForSeconds (1f);

    }

    IEnumerator esperar(){
        yield return new WaitForSeconds (5f);

    }

    public bool comprobarGameOver(){
        bool gameover=false;
        for (int x = 0; x < anchura; x++) {
            if (tabla [x, 12] != null) {
                //hemos llegado al limite de la tabla y perdemos
                Debug.Log("GameOver");
                gameover = true;
                return gameover;
            }
        }
        if(60-(int)Time.timeSinceLevelLoad==0){
            gameover = true;
            return gameover;
        }
        return gameover;
    }

}
```

Figura 39.14. Extracto del script "Tabla"