FACULTAD DE CIENCIAS

Desarrollo de un juego educativo sobre gestión de riesgos

(Development of an educational game about risk management)

Trabajo de Fin de Grado
para acceder al

Grado en Ingeniería Informática

Autor: Javier Paz Torcida

Director: Carlos Blanco Bueno

Septiembre - 2019

Agradecimientos

Una vez en este punto, lo que se debe de hacer es agradecer a todas aquellas personas que siempre han confiado en mí y me han dado lo mejor de ellas.

Lo primero de todo, y casi una obligación, es dar las gracias a mi familia por haberme permitido estudiar lo que quería, por no dejar que me desanimará cuando tenia que ir a Septiembre y sobre todo por hacerme levantar siempre que caía.

A mis amigos, ya sean los viejos amigos como los conocidos en la universidad, por ser una fuente constante de apoyo y de transmitir alegría y felicidad. Por estar siempre cuando se os necesita y por las cañas que nos quedan. Y sobretodo por ser insistentes y no dejar que me rindiera, sobretodo por eso.

Otro agradecimiento va por supuesto para mis compañeros de universidad, que tantos buenos momentos me han dado a lo largo de todos estos años. Me llevo el que fuerais tan abiertos y seguramente tambien me llevo la amistad de muchos de ellos.

Por último pero no menos importante, agradecer a todos los profesores que han pasado por mi vida, unos mejores que otros si, pero siempre se saca una enseñanza de cada uno de ellos y con eso es lo que me quedo.

Índice

Índice de tablas

Índice de figuras

Resumen

La gestión de riesgos es una parte muy importante dentro de la gestión de proyectos, no solo usada dentro del campo de la informática sino en cualquier área. Todo proyecto esta sujeto a ciertos riesgos que pueden o no materializarse provocando una cantidad de daño al proyecto. Todos estos riesgos deben de identificarse, analizarlos y planificar una respuesta para evitarlos o minimizarlos. Por lo tanto, la gestión de riesgos es necesaria si se quiere reducir al máximo la posibilidad de que el proyecto sufra algún daño por algún factor no controlado.

Por otro lado, tenemos la industria de los videojuegos. Un mercado que está asentando como uno de los mayores proveedores de entretenimiento para jóvenes y mayores, y el cuál no deja crecer cada año aumentando el número de juegos distribuidos y el número de personas que juegan a ellos cada año.

La idea clave de todo esto es juntar un mundo llamativo y popular como el de los videojuegos con un mundo de gran importancia en el día a día como la gestión de riesgos. El objetivo de este videojuego, por lo tanto, es el de educar a través de una herramienta vistosa y con la que se disfrute el aprendizaje.

Palabras Clave: Gestión de riesgos, Unity, videojuego, educar.

Abstract

Risk management its an essential part inside every project. Nowadays, it's not only used in the computer science field, but in any area. Every project is subject to some kind of risk that can or cannot materialize, causing damage to the project. All of this risk should be indentify and analized. This way, an answer in order to avoid or minimize any possible damage can be planify. Therefore, risk management is necessary if we want to decrease to minimum the possibility of the project suffering any harm by some uncontrolled factor.

Furthermore, we have the video games industry, a market that nowadays is becoming one of the biggest providers of entertainment for both young and elder. Every year, the amount of games distributed and the number of players in them is increasing hevily, which is opening the opportunity for new ideas and companies to use them as tools for further proposes.

The key idea of this project is to merge a popular world as video games with an important part of every work, such as risk management. The main objetive of this video game is to educate through an enjoyable tool that provides both, learning new skills and fun.

Key words: Risk management, Unity, video games, education.

1. Introducción

El objetivo que persigue la gestión de riesgos es ofrecer la posibilidad de identificar las posibles variables que puedan poner en riesgo y afectar un proyecto, así como ofrecer una base sólida para la toma de dichos riesgos y facilitar la planificación de acciones que eviten o minimicen el daño que estos pueden producir si acaban ocurriendo. En este campo, la principal fuente de conocimientos es el PMBOK, donde se muestran aquellos procesos que deben de seguirse si el objetivo es realizar una buena gestión de riesgos, con el fin de mantener el proyecto lo más protegido posible. Por otro lado, tenemos el enorme mercado de los videojuegos. Desde su creación, la sociedad ha entendido los videojuegos como algo con carácter lúdico cuyo único fin era entretener. Poco después, los desarrolladores llegaron a la conclusión de que el videojuego es algo más, y que se puede considerar como una herramienta capaz de llevar al usuario a una mejora de sus habilidades, y que a la vez de jugar, se pueden aprender y fijar nuevos conocimientos. Este tipo de videojuegos se conocen como serious games (juegos serios)[6], y cada vez son mas aceptados por centros educativos como escuelas o universidades, que comienzan a utilizarlos para enseñar a sus alumnos distintos conocimientos.

1.1. Contenido educativo

El contenido educativo ayuda a aprender sobre la gestión de riesgos, lo cual implica conocer como anticipar riesgos que puedan alterar el calendario del proyecto o la calidad de un software a entregar, y posteriormente tomar acciones para evitar dichos riesgos. [1]

Para la planificación de la gestión de riesgos, se deben de seguir una serie de procesos. El primero de ellos es la identificación de los riesgos, donde se trata de sacar averiguar todos los riesgos que pueden afectar al proyecto [4].

El segundo proceso consisten en realizar un análisis cualitativo de los riesgos que previamente se han identificado con el objetivo de priorizarlos. Para ello se debe calcular la probabilidad de que el riesgo se materialice, la pérdida que conllevaría si el riesgo llegara a darse. Y por último calcular con estos dos datos, la exposición al riesgo que nos servirá para priorizarlos. La Tab.1 muestra la tabla que se debe de obtener como resultado de aplicar éste análisis.

El último proceso a llevar a cabo es la planificación de las respuestas ante esos riesgos. La estrategia para responder a cada riesgo depende esencialmente de la medida en la que afecte al proyecto en caso de su materialización. Cómo puede observarse en la Fig.1, los riesgos con poco impacto suelen aceptarse, mientras que para los de mayor impacto se establecen estrategias (de mitigación o eliminación).

Riesgo	Probabilidad de pérdida	Magnitud de la pérdida (semanas)	Exposición a riesgo (semanas)
Añadir nuevas caracteristicas desde marketing (sin conocer las caracteristicas especificas)	35%	8	2.8
Planificación demasiado optimista	50%	5	2.5
Diseño inadecuado (hay que volver a diseñar)	15%	15	2.2
Las nuevas herramientas de programación no producen el ahorro prometido	30%	5	1.5
Añadir un requisito para la actualización automática desde el servidor	5%	20	1.0
Interfaz del subsistema de formato de gráficos inestable	25%	4	1.0
La aprobación del proyecto tarda más de lo esperado	25%	4	1.0
El personal contratado se retrasa en la entrega del subsistema encargado de formatear los gráficos	20%	4	0.8
Los recursos no están disponibles en su momento	10%	2	0.2
Los informes de estado a nivel de directiva necesitan más tiempo del previsto	10%	1	0.1

Tabla 1: Proceso cualitativo

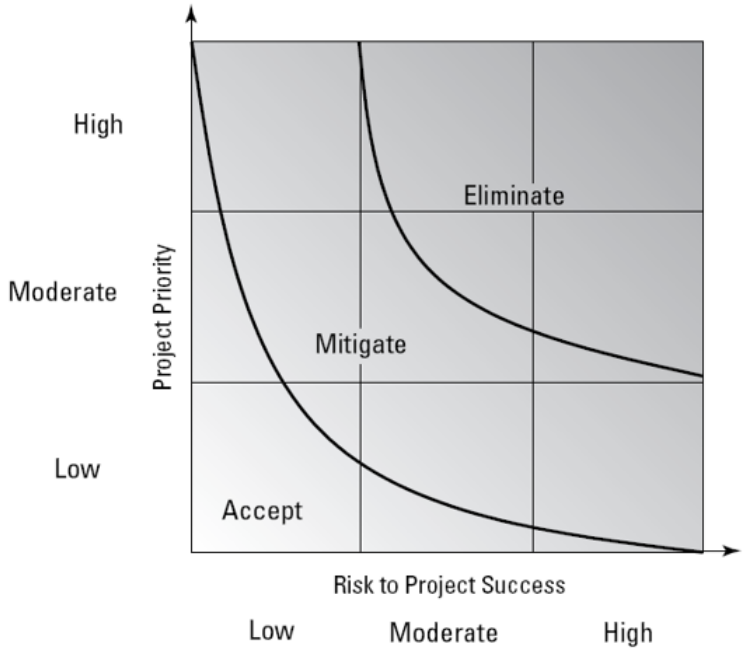


Figura 1: Estrategia para responder a cada riesgo

1.2. Motivación

La gestión de riesgos es algo primordial para asegurar el éxito de un proyecto y que no sufra una cancelación debido a factores no controlados. La motivación principal de este proyecto no es otra que la creación de un juego serio el cual pueda ser utilizado por cual-

quier persona interesada en aprender sobre como identificar riesgos y planificar un modo de actuación en base a éstos.

La motivación personal es comenzar a conocer el mundo de la programación de los videojuegos, lo cual siempre me ha llamado la atención. Con este trabajo, me enfrento a un reto tanto a nivel programático como de organización, que me permiten conocer mis virtudes y mis carencias, y aprender de todas ellas para futuros proyectos.

1.3. Objetivo

Los juegos serios utilizados en las clases tienden a ser poco llamativos para el alumno, haciendo el tiempo que se invierte en jugarlo de poco valor para esa persona. Un juego, que a vista del usuario, proporcione entretenimiento aunque su objetivo principal sea el educar sobre una materia, siempre resultará más atractivo al usuario y existirán más posibilidades de que vuelva a jugarlo, lo que llevará a un aumento de la fijación de dicho conocimiento.

Por lo tanto el objetivo principal de este proyecto es realizar un juego con el fin de educar sobre la gestión de riesgos que permita al usuario tener una experiencia agradable en el proceso, lo cual invite a volver a jugar no por obligación, sino por disfrute en el aprendizaje.

Como objetivo personal está el poner en práctica todo lo aprendido durante el grado, tanto a nivel software (patrones, buenas prácticas, arquitecturas) como a nivel organizativo (metodologías, división de tareas).

1.4. Idea del juego

Antes de comenzar con todo lo relacionado con el proyecto, explicaré de una manera clara y concisa el funcionamiento del videojuego a realizar, acompañado de su relación con los conceptos vistos anteriormente sobre la gestión de riesgos.

La idea básica reside en que el usuario deberá moverse por una pirámide de piezas hexagonales, las cuales representarán un riesgo para el jugador. Estos riesgos son los mismos que el usuario definiría en la etapa de identificación de riesgos. El jugador al comienzo de cada nivel deberá decidir, valorando tanto la probabilidad de que el riesgo se materialice como el daño que pueda causar, que contramedidas son las mejores/necesarias para completar ese nivel, mitigando al máximo todos los posibles riesgos presentes. Estas contramedidas hacen referencia a los posibles planes para mitigar los riesgos identificados, y por lo tanto es el jugador el que ha de saber cuáles le convienen más y cuáles se puede permitir no seleccionar.

Una partida típica comienza con el jugador en la posición inicial del tablero, donde antes de comenzar a jugar deberá visualizar el mapa y ver como se distribuyen los diferentes riesgos por este. Una vez controlados y vista la contramedida con la cual se mitiga ese riesgo, el usuario deberá elegir que contramedidas son las mas necesarias, siempre controlando el peso de éstas para no pasarse del peso máximo. Una vez están seleccionadas y el jugador lo desea podrá comenzar la partida. El jugador comenzará a moverse, cayendo de forma aleatoria en alguna de las piezas vecinas, donde existirá un riesgo y se comprobará si se tiene la contramedida o no. En caso de no tenerla perderá vida. Si llega al final con salud, se le permitirá avanzar al siguiente nivel donde se repetirá el proceso.

Una vez explicado como es una partida tipo, se mostrará cómo los conceptos de gestión de riesgos que se quieren transmitir se mapean en el juego. La identificación de los riesgos es la parte más sencilla para el jugador ya que solo necesita ver la leyenda para comprobar

cuales y cómo se sitúan los riesgos en el tablero. El análisis cualitativo se llevará a cabo visualizando visualizando el tablero de juego y la información de los riesgos. Mediante las casillas de juego podemos saber la probabilidad de ocurrencia de un determinado riesgo, observando el numero de veces que aparece y el nivel de fila en el que aparece. Por otro lado el impacto del riesgo la podemos conocer mediante la descripción del mismo, en el que se observa el numero de puntos de salud que perderían, en caso de que se materializara el riesgo por no disponer de la contramedida.

Por ultimo el proceso de planificar las respuestas a los riesgos se realiza cuando el jugador planifica su estrategia a la hora de escoger que contramedidas va a utilizar y cuales no. De esta forma está distinguiendo entre riesgos leves, los cuales debería aceptar, y riesgos que afectarían mucho al proyecto (quitándole mucha vida y con una alta probabilidad de ocurrir) sobre los que debería seleccionar su contramedida. Dependiendo de la estrategia seleccionada, conseguirá avanzar más niveles en el juego o por consiguiente perderá antes.

2. Herramientas, tecnologías y materiales utilizados

En este apartado se describirán lasa diferentes herramientas, tecnologías y materiales que se han ido utilizando a lo largo del proyecto.

2.1. Herramientas

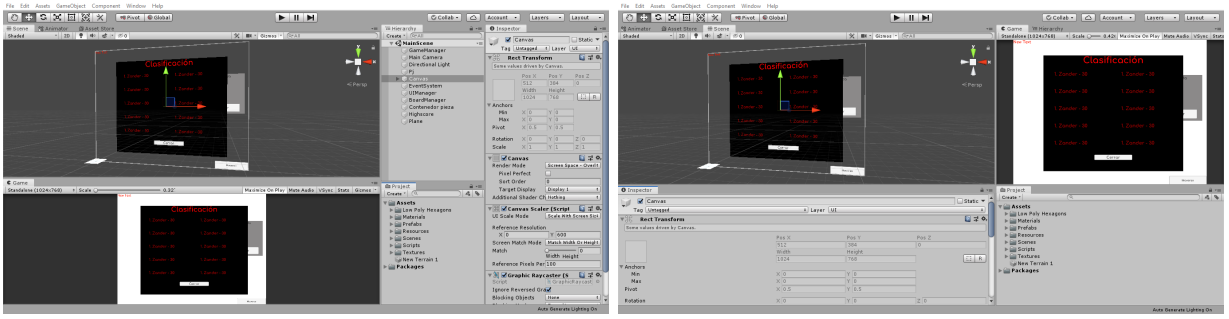
Las herramientas utilizadas para el desarrollo del videojuego han sido las siguientes.

2.1.1. Unity

Unity es un motor utilizado para el desarrollo de videojuegos, tanto 2D como 3D, creado por Unity Technologies. Sus inicios están datados en 2005, siendo lanzado exclusivamente para MacOS, y gracias al éxito que cosecho prosiguieron mejorando este motor permitiendo que a día de hoy también pueda ser ejecutado en otras plataformas como Windows o Linux.

Algo que destacar de Unity es que no solo se limita a la creación de videojuegos sino que también puede ser usada para desarrollar aplicaciones interactivas, como la realidad aumentada, o el desarrollo de aplicaciones para móviles, que no tienen por que ser videojuegos.

Entrando a la funcionalidad de la herramienta, se nos ofrecen muchas facilidades en forma de una interfaz sencilla de entender y muy personalizable a la vez. Tenemos la posibilidad de mover cada componente a nuestro gusto, como se muestra en la Fig.2, con el objetivo de tener nuestro entorno de trabajo lo más cómodo posible. Aparte soporta distintos lenguajes de programación de fácil aprendizaje como JavaScript, C# o Boo y un mercado de recursos, donde se puede consultar los trabajos de otros desarrolladores o incluso volcar los tuyos propios. La disponibilidad de la mayoría de estos recursos es gratuita, lo que permite consultar este material y utilizarlo de una manera muy sencilla.



(a) (b)

Figura 2: Personalización de la interfaz de Unity

Lo último a destacar es la amplia documentación con la que cuenta Unity, siendo esta muy intuitiva y muy explícita para que el usuario pueda consultar y entender de manera rápida para seguir con su proyecto.

Cabe añadir que Unity dispone de distintas licencias que se adaptan a las necesidades de cada uno de sus usuarios. Una de estas es ofrecida gratuitamente a cualquier tipo de público. A pesar de su fácil disponibilidad, esta version permite la realización de trabajos de muy

alto nivel profesional, no presentando apenas características inferiores a otras. Dicha version es utilizada en este proyecto.

2.1.2. Visual Studio

Visual Studio es un entorno de desarrollo multilenguaje (C++, C#, Java, Python, etc) así como multiplataforma (Microsoft, Linux, MacOS). La primera versión de Visual Studio fue lanzada en 1998 de la mano de Microsoft, pero la versión que propició un gran cambio y que sentó las bases para que este programa sea la herramienta que es ahora fue en 2002.

Este entorno nos proporciona una gran cantidad de facilidades dentro de él. A la hora de codificar, nos proporciona gran cantidad de atajos de teclado, así como la funcionalidad del auto completado y una gran variedad de recomendaciones para solucionar cualquier fallo que aparezca. En cuanto a la depuración, permite el libre movimiento por el código aún encontrándonos en modo debug, sin necesidad de reiniciar la ejecución, así como la facilidad para conocer el valor de cualquier variable [13].

Mencionar por último, la posibilidad de incluir pluggins para hacer más sencilla la utilización de esta herramienta. Además de poder comunicarse con otras aplicaciones, como en este caso que con el *plugin* de Unity podemos trabajar con todas sus librerías así como vincular ambos programas para poder realizar la tarea de *debug* cuando se ejecute el juego en Unity [14].

Aunque Unity ofrece un entorno de desarrollo gratuito, en este proyecto se ha decidido trabajar con Visual Studio debido a que, además de ser gratuito también, es una herramienta utilizada durante el grado, lo que proporciona ventajas tanto en funcionalidad como en conocimiento y manejo sobre esta.

2.2. Tecnologías

En este apartado se explican las tecnologías utilizadas para la realización del proyecto abarcando los distintos lenguajes que se hayan utilizado así como librerías que tengan una importancia reseñable.

2.2.1. Lenguaje C#

El lenguaje C# es un lenguaje de programación orientado a objetos cuyo diseño y estandarización corre a cargo de Microsoft.Es considerado una evolución de los lenguajes C y C++ y surge con la idea de ser utilizada en .NET, plataforma creada por Microsoft.

Este lenguaje dispone de varias características que muestran su potencial. Nos encontramos ante un lenguaje más sencillo que sus antecesores, que dispone de una alta seguridad, además de ser extensible. La disponibilidad de distintas versiones hace que se mantenga actualizado constantemente y que incorpore novedades en cada version nueva. Además, es compatible tanto con los lenguajes que precede como con Java, un lenguaje muy usado actualmente.[5]

No hay que olvidar que existen más lenguajes de programacion para el desarrollo de videojuegos, pero en este caso se utiliza C# ya que el juego a desarrollar no necesita tener en cuenta una gestión de los recursos óptima.

2.2.2. XML

XML es un lenguaje de marcado de propósito general (es decir, tu eres quien define las etiquetas ya que no hay predeterminadas). Se trata de un subconjunto de SGML (*Standard Generalized Markup Language*)[2]. La funcionalidad de este lenguaje reside en poder almacenar información de forma estructurada y permitir que ésta misma sea almacenada, transmitida, procesada y visualizada por diferentes tipos de aplicaciones. Este tipo de aplicaciones suelen ser dispositivos móviles, servicios web (SOAP, UDDI), sistemas de información, etc.

Las ventajas de este lenguaje están en que es muy fácil de procesar, no pertenece a ninguna empresa ergo es libre su uso, separa el contenido de la presentación y además soporta cualquier lenguaje. Mencionar que existen mecanismos para comprobar si un XML es valido como son el XSD o el DTD los cuales definen esqueletos que el XML debe cumplir.

2.2.3. Android SDK

SDK(*Software Development Kit*) es un kit de desarrollo software el cual dispone de un grupo de herramientas que permiten la programación de aplicaciones móviles .Al ser un kit de Android nos indica que sirve para el desarrollo de aplicaciones para dispositivos de esta misma marca. Gracias a este kit se ha desarrollado el juego tanto para móvil como para tablet.

2.3. Materiales

Los materiales utilizados en este proyecto (entendiéndose materiales como los diferentes *assets* gráficos o de estilo, así como los objetos que se han utilizado en el videojuego) provienen del mercado que proporciona Unity y los cuales son de libre acceso.

En este caso, en la estética de las piezas del tablero se hace uso de los *prefabs* creados por BLACKDRAGONBE, quien no pide ningún tipo de mención pero me parece licito mencionar su autoría.

3. Metodología

En este apartado se aborda la metodología que se ha utilizado para realizar el proyecto. La metodología elegida ha sido la iterativa incremental.

3.1. Metodología iterativa incremental

La idea básica de la metodología iterativa incremental es desarrollar el sistema siguiendo etapas incrementales caracterizadas por generación de sucesivas versiones que van abarcando requerimientos hasta completar el sistema.

Esta metodología ofrece numerosas ventajas con respecto a otras metodologías. Las expectativas del cliente son gestionadas de manera regular, ya que puede ir tomando decisiones en cada iteración, la introducción de nuevos requisitos tiene un coste reducido y además ofrece gran flexibilidad ante posibles cambios. Algo importante es que permite conocer el progreso real del proyecto y permite tras las primeras iteraciones, ver si se van a cumplir los plazos o por otro lado el cliente necesita priorizar requisitos.

Por último, se recomienda el uso de esta metodología cuando puedes hablar con el cliente de manera frecuente. En este caso, se utiliza esta forma ya que el trato con el director de proyecto cumple estas expectativas.

Para llevar acabo esta metodología lo primero que hay que realizar es una planificación del número de iteraciones que se van a realizar a lo largo del proyecto, con su determinada duración y con las funcionalidades que se van a implementar.

Una vez hecha la planificación, cada iteración va a funcionar como si se estuviese desarrollando un sistema software común, es decir en cada iteración se va a realizar las tareas de análisis, diseño, implementación y pruebas. Fig.3 muestra el funcionamiento de esta metodología.

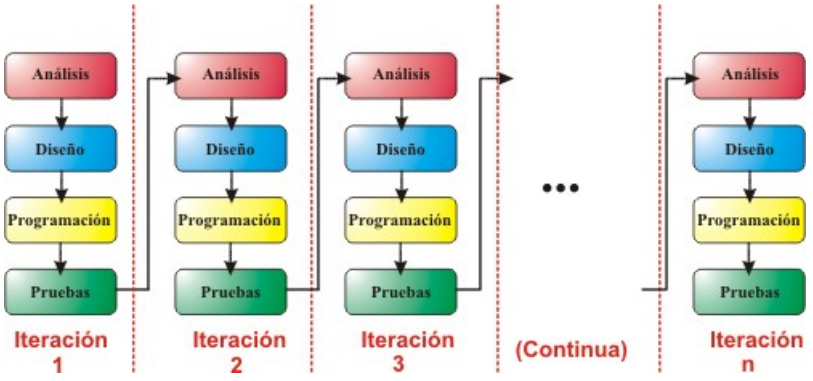


Figura 3: Funcionamiento de la metodología iterativa incremental

3.2. Planificación

En la Tab. 2 se muestra el plan acordado para la realización del proyecto donde se muetran las fechas de inicio y de fin de todas las iteraciones así como de la etapa de integración.

Etapas	Fecha de Inicio	Fecha de Fin
Iteración 1	11/07/2019	18/07/2019
Iteración 2	19/07/2019	26/07/2019
Iteración 3	27/07/2019	05/08/2019
Iteración 4	06/08/2019	13/08/2019
Iteración 5	14/08/2019	21/08/2019
Integración	22/08/2019	05/09/2019

Tabla 2: Plan del proyecto

3.2.1. Etapa de formación

Aunque en la planificacion no venga reflejada una etapa de formación esta fue realizada antes de realizar la planificacion del proyecto.

En esta etapa se buscó familiarizarse con la herramienta de Unity, para ello se realizaron una serie de tutoriales proporcionados por la misma página de Unity. La idea fue crear varios juegos sencillos y de distinta índole (2D y 3D) con el fin de tener una visión sencilla pero amplia para abordar este proyecto [3,8-12].

3.2.2. Etapa de desarrollo

Esta etapa esta formada por todas y cada una de las diferentes iteraciones planificadas y las cuales siguen la misma estructura que se ha explicado anteriormente.

1. **Análisis:** En esta etapa se deben especificar los hitos que tienen que cumplirse al final de la iteracion para considerarla como completada o no. Además tambien se tienen que definir los medios para alcanzar esos hitos.

2. **Diseño:** Etapa donde se deben adoptar decisiones cuya finalidad tiene el cumplimiento de los hitos definidos en la etapa de análisis. Este tipo de decisiones abarcan tanto a nivel de diseño (es decir que patrones usar o como distribuir las clases) como a nivel visual (todo lo relacionado con la interfaz del juego).

3. **Implementación:** Etapa donde se implementa la logica del juego, la cual ha sido diseñada previamente.

4. **Pruebas:** Última etapa de la iteración donde se comprueba que las nuevas implementaciones no han alterado el juego y todo mantiene un correcto funcionamiento.

En el apartado de análisis, se explica que los hitos son los que permiten ver si una iteración ha sido exitosa o no al comprobar si estos se han completado. Procedo a realizar un breve resumen de los hitos planificados y cumplidos en las distintas iteraciones del proyecto.

- **Iteración 1:** En la primera iteración se realizo un primer diseño del juego completo, tanto sus pantallas como su mapa principal. Además se definió la funcionalidad general del juego siendo esta la de moverse por las plataformas de manera aleatoria, incluir la existencia de riesgos (para las piezas del tablero) y contramedidas (para el usuario), añadir un efecto a los riesgos (traducido en una perdida de una cantidad), que el usuario pudiese seleccionar esas contramedidas para contrarrestar los riesgos y por último limitar esa selección introduciendo el concepto de peso. En el área visual la interfaz era lo más arcaica posible con el fin del probar todo esto de la manera más sencilla.

- **Iteración 2:** En esta iteracion se incidió sobre la interfaz de hacerla más amigable al usuario respetando todas las funcionalidades anteriores. Por lo tanto se realizaron distintos mock ups de los menús para introducirlos al juego más tarde cambiando los que ya existían por unos más actualizados y más intuitivos.

- **Iteración 3:** El objetivo de esta iteración fue el de realizar la generación automática del mapa. Esto se basa en generar el mapa cada con un numero mayor o menor de piezas totales. Esto se utiliza cada vez que el usuario completa un nivel, para que al siguiente se encuentre con más piezas y sus riesgos correspondientes cambiados de lugar para no acostumbrarse.

- **Iteración 4:** Es esta iteración se realizo la creación de estructuras de datos para almacenar tanto los riesgos como las contramedidas, en ficheros XML. Además se trabajo un poco más en las piezas asignándoles un color dependiendo del riesgo que tuviesen para facilitar al usuario su identificación así como la introducción de una leyenda para hacerlo mas sencillo.

- **Iteración 5:** En esta última iteracion el hito a cumplir consistía en la introducción de un sistema de clasificación online de la puntuación utilizando la pagina web Dreamlo y añadiendo a la interfaz el soporte para mostrar esta puntuación así como un pequeño formulario pidiendo al usuario un nombre para guardarlo.

3.2.3. Etapa de integración

Se trata de la última etapa del proyecto donde se busca que todo el sistema final funcione correctamente, pulir aquellos detalles de ultima hora que le den un mejor aspecto al juego y por último generar toda la documentación relacionada con el proyecto, es decir esta memoria.

4. Análisis de Requisitos

En este apartado se detallan tanto los requisitos funcionales como los requisitos no funcionales.

4.1. Requisitos funcionales

En la Tab. 3 se muestran con detalle los requisitos funcionales que el juego debe de cumplir

ID	Descripción
RF01	El jugador se moverá aleatoriamente.
RF02	Las piezas solo tendrán 1 riesgo asignado.
RF03	El jugador tendrá un valor máximo para controlar la selección de contramedidas.
RF04	El juego acabará cuando el jugador llegue a 0.
RF05	Al completar un nivel, se generará un nuevo nivel aleatorio.
RF06	Al pasar de nivel el usuario podrá seleccionar nuevas contramedidas.
RF07	Al terminar la partida el jugador puede volver a jugar si lo desea.
RF08	Los riesgos no estarán en la misma posición en cada nivel.
RF09	Las contramedidas tendrán un valor que le servirá de referencia al usuario.
RF10	Cada vez que caes en un riesgo, se mostrará información al usuario de en que riesgo ha caído, de si dispone o no de la contramedida y si ha perdido puntos de vida en caso de no disponer de ella.
RF11	Cuando se complete un nivel se mostrará información resumen del nivel como son los puntos iniciales, cuantos ha perdido y los puntos actuales.
RF12	El juego tendrá un sistema de clasificación online
RF13	El jugador no podrá coger una contramedida que exceda su peso máximo.
RF14	Solo se guardará la máxima puntuación de cada jugador
RF15	El jugador podrá deseleccionar aquellas contramedidas que haya seleccionado previamente.

Tabla 3: Requisitos funcionales

4.2. Requisitos no funcionales

En la Tab. 4 se muestran con detalle los requisitos relacionados con el funcionamiento del juego y no con la funcionalidad de este [7].

ID	Tipo	Descripción	Relevancia
RNF01	Rendimiento	El juego deberá superar en todo momento los 30 frames por segundo.	Media
RNF02	Rendimiento	El juego deberá hacer un uso máximo de memoria de 2GB.	Alta
RNF03	Portabilidad	El juego debe poder ejecutarse tanto para escritorios (Windows, Linux, MacOS) como en móviles y tabletas Android.	Muy Alta
RNF04	Usabilidad	El juego será apto para todos los públicos.	Media

Tabla 4: Requisitos no funcionales

5. Diseño e Implementación

En este apartado se explicará el diseño arquitectónico utilizado así como los aspectos que se consideren más relevantes sobre el diseño y la implementación de cada una de las partes involucradas en el proyecto.

5.1. Arquitectura del sistema

La arquitectura elegida es la arquitectura en tres capas. Es el patrón más típico a la hora de desarrollar un sistema software. Consiste en separar el sistema en distintas capas, teniendo como objetivo la separación de responsabilidades entre las distintas capas. Las tres capas de esta arquitectura son: Presentación, negocio y datos. Más tarde se explicará la función de cada una, acompañado de ejemplos dentro del juego.

Las ventajas que nos puede ofrecer este tipo de arquitectura residen en el hecho de separarlas, ya que nos ofrece aislamiento, nos permite realizar cambios en la tecnología en alguna de estas sin que el sistema completo se vea afectado, un mayor rendimiento procedente de poder aumentar la escalabilidad y la reutilización de código basándonos en que las capas inferiores no dependen de las superiores por lo que pueden ser usadas en otro tipo de escenarios.

5.2. Capa de presentación

En esta capa se realiza la tarea de llevar a cabo la interacción entre el usuario y el sistema. Por tanto, aquí se presentarán y administrarán las diferentes pantallas del sistema. Con esta capa el usuario podrá visualizar el mapa por el que va a moverse, gestionar las contramedidas y ver los riesgos a los que se enfrenta. Debido a la arquitectura escogida, esta capa solo debe comunicarse con la capa de negocio.

Dentro de esta capa vamos a distinguir entre dos componentes distintos: la interfaz de usuario y las escenas del juego.

5.2.1. Interfaz de usuario

La idea principal de la interfaz del usuario es que este se sienta cómodo a la hora de interactuar y le resulte sencilla. En este caso se puede hacer una distinción entre dos partes.

Figura 4: Ventana de selección de las contramedidas

Por una lado tenemos la parte de la interfaz pre-movimiento, la cual se puede ver en la Fig.4, donde el usuario podrá consultar cuales son los riesgos que hay en el tablero, ofreciéndole un pequeño resumen con la descripción de este, que efecto tienen sobre el y que contramedida le anula tal efecto. Además se le incluye un botón para ver el mapa, ya que los riesgos son visuales a través de colores. Por último, se acompaña de una leyenda para facilitar la identificación de dichos riesgos, y poder ver donde se ubica cada uno y su cantidad para ver que contramedida es mas interesante de señalar.

Una vez se han consultado los riesgos y ha visto cual necesita evitar más, puede pasar a seleccionar las contramedidas. Están acompañadas de un peso a través del cual el usuario debe guiarse para decidir cual elegir o no, ya que este tiene un peso máximo y no puede pasarse de el. Este se le indica justo encima de la lista de las contramedidas, cambiando si se selecciona o deselecciona una contramedida. La lista de la izquierda muestra las contramedidas que el usuario ha seleccionado para jugar en ese nivel, pudiendo quitarlas si lo desea. Estas contramedidas no son eternas, es decir cuando se cambien de nivel la lista de seleccionadas volver a estar vacía teniendo el usuario que volver a decidir cual es la mejor opción ahora.

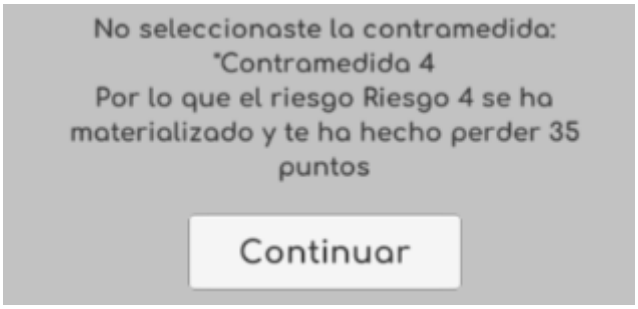


Figura 5: Ventana con la información del riesgo

Por otro lado tenemos la parte de la interfaz donde el usuario comienza el juego realmente. Se buscaba que esta fuera lo más sencilla posible por eso el usuario con la menor cantidad de elementos posibles. En este punto únicamente aparecerán ventanas con información para el usuario hasta que llega al final del nivel. Estas ventanas, como la de la Fig5 que le aparecen al usuario muestran información básica de la pieza. En ella se puede distinguir el nombre del riesgo que la afecta, si se dispone de la contramedida para mitigarle o en caso contrario mostrará la cantidad de puntos que ha perdido el usuario.

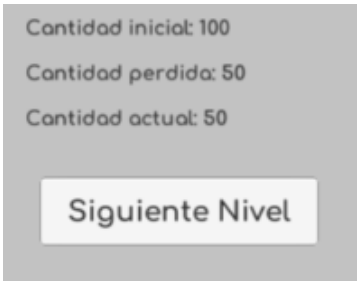


Figura 6: Ventana con la información resumen del tramo

La otra pantalla de información, la representada en la Fig.6 aparece siempre al final del nivel donde le indica al usuario un pequeño resumen de cómo le ha ido el nivel, mostrando la cantidad de puntos con los que empezaba, la cantidad de puntos que ha perdido a lo largo del camino y por último la cantidad actual de puntos que le quedan.

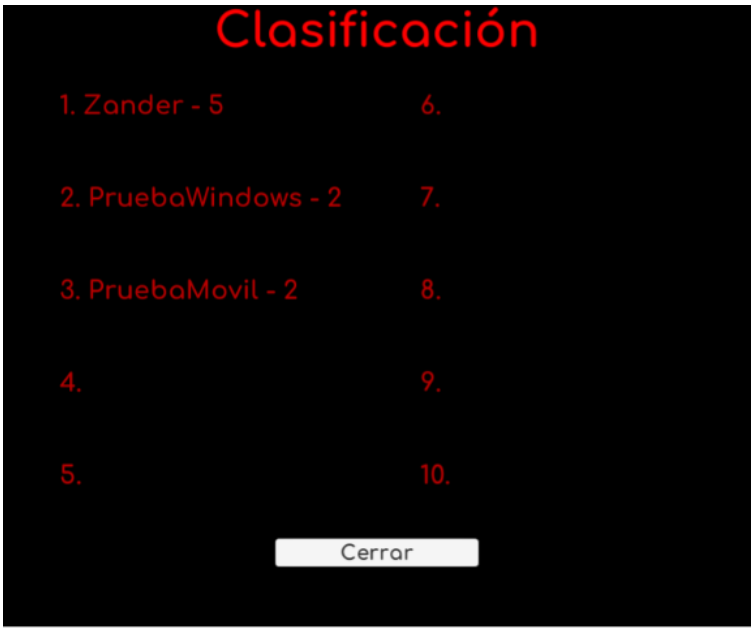


Figura 7: Clasificación

Comentar tambien la Clasificación , mostrada en la Fig.7, donde el usuario puede visualizar al final de la partida si lo desea, donde se ha buscado darle un toque sencillo que simplemente cumpla la función de mostrar quienes son los mejores. Se ha buscado mostrar un numero amplio de jugadores con el fin de que al haber más jugadores se intente potenciar el hecho de querer aparecer en ella ya que hay mas posibilidades. Además el pequeño formulario en la pantalla de carga, antes de arrancar el juego, nos ayuda a obtener el nombre de usuario para guardar en la clasificación.

Comentar que los colores utilizados para las distintas piezas tiene como objetivo poder distinguir los riesgos, son una medida temporal pero que cumplen su función muy bien. Este punto será desarrollado en mayor profundidad más tarde en las conclusiones.

Por último, dar una explicación sobre el módulo, a nivel de código, encargado de gestionar todo lo relacionado con la interfaz. Este modulo contiene todo lo relacionado con manejar los distintos eventos de la interfaz. La clase principal es UIManager, y en ella reside toda la logica para administrar la interfaz del juego, tanto la funcionalidad de cada botón, como los eventos que surjan a medida que el jugador va avanzando. La clase PantallaCargaManager hace la misma función que la clase descrita anteriormente pero actuando solo en la pantalla de carga, por lo que su complejidad es mucho menor. Por último la clase EfectoEscritura permite escribir un texto letra a letra.

5.2.2. Escenas

Las escenas en Unity hacen de contenedor de todos los menús y entornos, objetos del juego que se han explicado en el punto anterior, por lo tanto se podría considerar que cada escena forma un nivel único y ahí reside la idea fundamental de utilizarlas y es hacer que el desarrollo del juego sea mediante unir "pedazos" con el fin de hacer más sencillo cada una de las distintas escenas.

En este caso el juego dispone de dos escenas:

- **MainScene:**Es la escena donde se desarrolla todo el juego, en esta escena se encuentra la mayor parte de la interfaz descrita en el punto anterior.
- **PantallaCarga:** Es la escena que gestiona el inicio del juego y donde si es la primera vez que accedes, se muestra el formulario pequeño para introducir el nombre.

5.3. Capa de Negocio

Esta es la capa se almacena toda la lógica que hay detrás del programa, es decir, toda la funcionalidad del programa y las reglas que han de cumplirse. Se comunica con la capa de presentación, recibiendo solicitudes y presentando los datos, y con la capa de datos solicitando datos o para almacenarlos en esta. Mencionar que en esta capa se mantendrá una comunicación con un servicio externo (Dreamlo) el cual se encarga de gestionar la clasificación y cuyo funcionamiento será explicado más tarde.

En esta sección se comentará tanto la estructura completa del proyecto como centrarnos en aquellos aspectos funcionales que he creído que disponen de un mayor interés y de una mayor dificultad, aunque puntualizar que todas aquellas funcionalidades que no comente, no quiere decir que su programación sea trivial sino que tienen un impacto menor en el juego final.

5.3.1. Estructura del proyecto

El código del proyecto está formado por un único módulo. Este contiene toda la funcionalidad del juego en cuanto a jugabilidad se refiere. En la Fig. 8 se puede apreciar cómo se distribuyen jerárquicamente cada uno de los *scripts* creados.

Antes de pasar con los elementos más relevantes, conviene comentar brevemente que función tiene cada uno de los *scripts* dentro de su módulo:

- **Juego:** En este modulo se encuentra toda la logica del juego. El *GameManager* es la clase que se encarga de la gestión del juego. El modulo de la clasificación contiene toda la comunicación con el servicio externo (dreamlo) para mantener una clasificación online de la puntuación. Dentro de los elementos del tablero encontramos las partes necesarias para tener un tablero completo. El tablero propiamente dicho contiene una clase Tablero donde se realiza el cargado de estructuras internas para controlar toda la información referida a cómo se relacionan unas piezas con otras; y la clase BoardManager que se encarga de crear en cada nuevo nivel un tablero nuevo. El modulo de los riesgos y contramedidas se encarga de gestionar su asignación y distribución tanto a las piezas como al propio jugador. La clase Pieza sirve para identificar a cada pieza del tablero y poder guardar el riesgo que tiene cada una; la cámara cuyo *script* nos ayuda

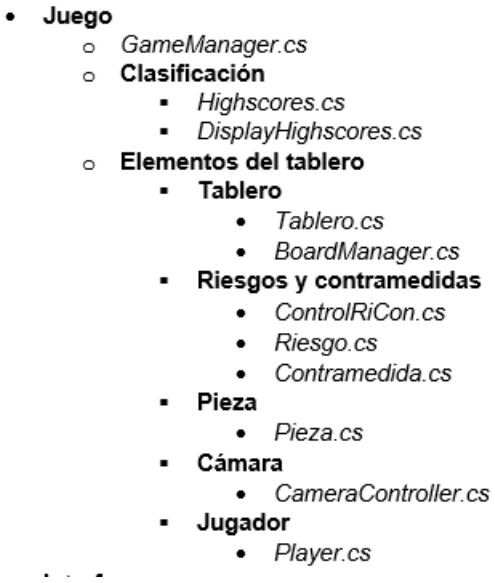


Figura 8: Estructura del proyecto.

a seguir el movimiento del jugador y no mantener la cámara fija. Y por último la clase Jugador que en ella se almacena la información referente al jugador y la gestión de su movimiento.

5.3.2. Gestor del tablero (BoardManager)

El BoardManager es una de las clases más importantes del juego ya que gracias a ella, el mapa se genera de nuevo cada vez que el jugador completa un nivel. En este apartado voy a ir explicando los métodos que contiene esta clase, para asegurar que el tablero se construye de la manera correcta:

- **creaPiezas:** El objetivo de este método es el de, en base a la posición de una pieza de la fila superior, crear las piezas que se denominarían vecinos (piezas que se encuentran en el nivel inferior a una dada y a las cuales puede llegar con tan solo avanzar). Para todo esto recibe la posición de la pieza, se explicará más adelante que es esto, y en base a eso sabe que factores de distancia debe aplicar a esa pieza para crear a sus vecinos (siempre comprobando que no hay una pieza ya creada en esa posición debido a que las piezas comparten un vecino). Una vez se les aplica esos factores, tenemos como resultado posiciones nuevas donde crearemos las nuevas piezas y por lo tanto se procede a su creación.

- **posiciónPieza:** Este método es el que proporciona en que lugar de la fila se encuentra una posición dada y la cual es usada por el método anterior. Los factores que se les aplica a las piezas se basas en si se encuentran a la derecha, a la izquierda o en medio de la fila. Por lo tanto aquí se calcula en que posición numérica están y se les asigna uno de los nombres dichos anteriormente.

- **cargaColorRiesgo:** Este método es primordial para el buen funcionamiento del juego ya que se encarga de asignar un color a un riesgo determinado para que cuando se creen

estas piezas tengan este color y permitan al usuario distinguir los diferentes riesgos que se encuentran en el tablero. Básicamente su funcionamiento reside en realizar parejas clave, valor, donde la clave es el nombre del riesgo y el valor la pieza que debe de crearse. Es un método sencillo a nivel de código pero de suma importancia para el programa.

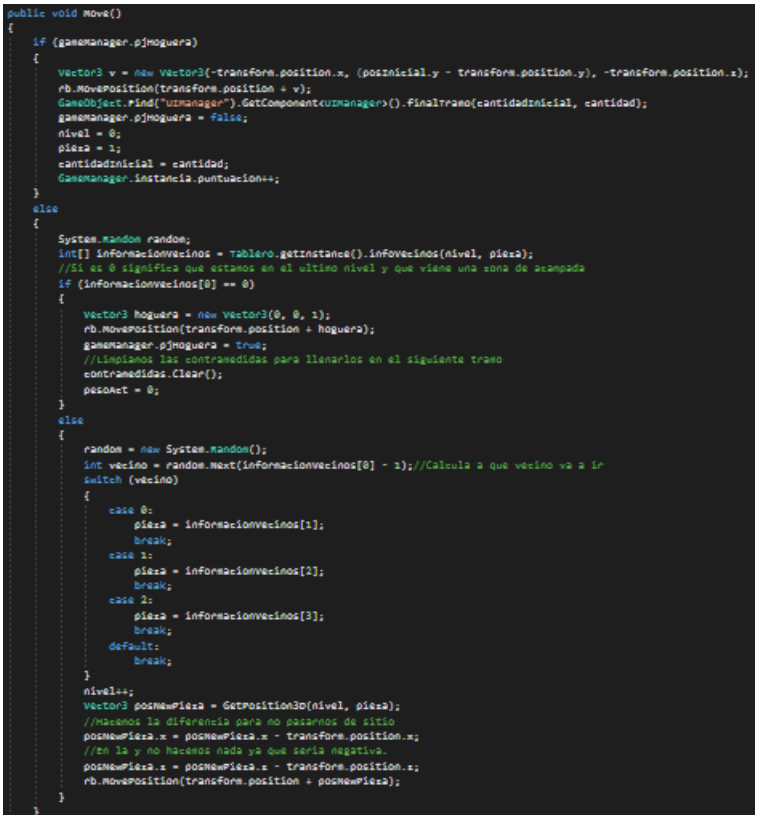
- **creaTablero:** Por último esta el método principal de esta clase con el que, utilizando el resto de métodos mencionados, se permite la generación del tablero con un numero de niveles dado. En la Fig.9 se muestra el código asociado a este método. Destacar el bucle for donde se produce la creación de los distintos niveles completos utilizando la posición 3D de las piezas de la fila anterior para sumarles los factores previamente comentados.

```
public void creaTablero(int niveles)
{
    cargaColorRiesgo();
    Tablero.getInstance().cargaVecinos(niveles);
    tablero = new GameObject("Tablero").transform;
    piezasNivel  = Tablero.getInstance().piezasNivel;
    int altura = niveles * 2;
    string posicion;

    //Creamos la primera pieza
    GameObject instance = Instantiate(riesgoColor["Primera pieza"], new Vector3(0f, 0f, 0f), Quaternion.identity) as GameObject;
    instance.transform.localScale += new Vector3(0, altura, 0);
    instance.transform.localEulerAngles = new Vector3(0, 90, 0);
    instance.transform.SetParent(tablero);



    //Guardaremos la posicion de las piezas del nivel anterior
    Vector3[] posNivelAnterior = new Vector3[1];
    posNivelAnterior[0] = new Vector3(0f, 0f, 0f);
    Vector3[] posNivelActual;

    int contador = 1;

    //Y ahora el resto de piezas
    for (int i = 1; i < piezasNivel.Length; i++)
    {
        int hueco = 0;
        posNivelActual = new Vector3[piezasNivel[i]];
        altura -= 2;
        while (contador <= piezasNivel[i-1])
        {
            posicion = posicionPieza(contador,piezasNivel[i-1]);
            hueco = creaPiezas(posicion, posNivelAnterior[contador-1], posNivelActual,hueco,altura);
            contador++;
        }
        posNivelAnterior = posNivelActual;
        contador = 1;
    }
}
```

Figura 9: Código del método creaTablero.

5.3.3. Jugador

En este apartado se va a explicar y mostrar la lógica del jugador, ya que el movimiento de este tiene una gran importancia dentro del juego, debido a que es necesario que el movimiento sea aleatorio con el fin de que el usuario nunca sepa de manera segura en que riesgo va a caer y por lo tanto pueda utilizar este modo aleatorio para valorar los diferentes riesgos.



```
public void Move()
{
    if (gameManager.pjHoguera)
    {
        Vector3 v = new Vector3(-transform.position.x, (posInicial.y - transform.position.y), -transform.position.z);
        rb.MovePosition(transform.position + v);
        GameObject.Find("UIManager").GetComponent<UIManager>().FinalTramo(cantidadInicial, cantidad);
        gameManager.pjHoguera = false;
        nivel = 0;
        pieza = 1;
        cantidadInicial = cantidad;
        GameManager.instancia.puntuacion++;
    }
    else
    {
        System.Random random;
        int[] informacionVecinos = Tablero.getInstance().infoVecinos(nivel, pieza);
        //Si es 0 significa que estamos en el ultimo nivel y que viene una zona de acampada
        if (informacionVecinos[0] == 0)
        {
            Vector3 hoguera = new Vector3(0, 0, 1);
            rb.MovePosition(transform.position + hoguera);
            gameManager.pjHoguera = true;
            //Limpiamos las contramedidas para llenarlos en el siguiente tramo
            contramedidas.Clear();
            pesoAct = 0;
        }
        else
        {
            random = new System.Random();
            int vecino = random.Next(informacionVecinos[0] - 1);//Calcula a qué vecino va a ir
            switch (vecino)
            {
                case 0:
                    pieza = informacionVecinos[1];
                    break;
                case 1:
                    pieza = informacionVecinos[2];
                    break;
                case 2:
                    pieza = informacionVecinos[3];
                    break;
                default:
                    break;
            }
            nivel++;
            Vector3 posNewPieza = GetPosition3D(nivel, pieza);
            //Hacemos la diferencia para no pasarnos de sitio
            posNewPieza.x = posNewPieza.x - transform.position.x;
            //En la y no hacemos nada ya que sería negativa.
            posNewPieza.z = posNewPieza.z - transform.position.z;
            rb.MovePosition(transform.position + posNewPieza);
        }
    }
```

Figura 10: Código del método Move.

- **GetPosition3D:** Este método nos devolverá la posición exacta en la que se encuentra una pieza dada. Resulta necesario ya que a la hora de moverse el personaje necesita conocer la posición a la que va a ir.

- **Move:** Método principal de la clase y primordial en el juego. La Fig.10 muestra el código asociado al método. Lo que se busca en este método es, desde la pieza actual, ver quienes son tus vecinos y mediante la probabilidad moverte a uno de ellos. Para ello utilizando la ayuda de la clase Tablero que conoce toda la estructura de los vecinos, te dirá cuantos tienes y quienes son, para que al sacar un numero aleatorio ir a ese vecino. Una vez tenemos la posición del vecino al que ir, gracias al método explicado en el punto anterior, tan solo hay que desplazar al jugador hasta esa pieza. Añadir que tambien se tiene en cuenta al principio del método la situación del jugador antes de moverse. El jugador puede estar en tres situaciones distintas, una es la descrita hasta ahora la cual tiene un movimiento normal. La segundo es cuando se encuentra en la ultima fila y ya no tiene más vecinos por lo que su siguiente movimiento va a ser a caer al suelo. Al llegar a este punto se aprovecha para liberar al personaje de las contramedidas seleccionadas y reiniciar por tanto su peso. Y la última situación es la de encontrarse en el suelo (o como lo llamo dentro estar en la hoguera, haciendo una analogía de como si estuviera descansando) donde la acción de moverse le hará volver a la posición inicial pero de un nuevo tablero ya que, aquí se aprovecha para generar uno nuevo como consecuencia de haber completado el nivel.

- **Trigger:** Esta parte es para comentar un poco como actúa el personaje al caer sobre una pieza. Cuando este entra en contacto con una, automáticamente saltará el trigger ya que se trata de una pieza con un riesgo y debe comprobarse las consecuencias de esto. Cuando esto ocurre debe primero comprobarse si el usuario dispone de la contramedida adecuada para ver si consigue mitigar el riesgo, en caso negativo habrá que calcular el daño que le hace este riesgo a la cantidad de puntos del jugador. Por último activa la ventana con la información referida a lo que acaba de ocurrir.

5.4. Capa de datos

Esta capa almacena los datos y se encarga de acceder a ellos. La forman uno o más gestores de datos siendo estos los que realizan el almacenamiento de la información y el procesado de llamadas desde la capa de negocio para solicitar o almacenar información. En este proyecto esta formada por los ficheros XML los cuales almacenan los riesgos con sus contramedidas y la clasificación online la cual esta almacenada por el servicio dreamlo.

Antes de pasar a explicar más en profundidad cada una de las partes de esta capa, hay que mencionar que existe un tercer elemento de almacenamiento de datos. Este es proporcionado por Unity y se basa en ficheros de configuración los cuales van almacenando a medida que se van creando.

5.4.1. Ficheros XML

Como este tipo de tecnología ya se ha explicado anteriormente me ahorrare más datos sobre ella y pasaré directo a hablar de los ficheros creados para este proyecto.

Los únicos elementos almacenados a través de esta tecnología en el proyecto son los riesgos y las contramedidas. Ante esto surge una duda y es si realizar todo en el mismo o separarlos en dos diferentes. El hecho de separarles traía una dificultad extra y era la de tener que comprobar, a la hora de cargarlos, que los Id de la contramedida cargada y la contramedida que mitiga su riesgo eran los mismos. No representaba un problema enorme, pero se prefirió hacerlo los dos en el mismo archivo por comodidad y por el hecho de que cada riesgo solo va a tener una contramedida por lo que separarlos no tendría sentido.

Antes de hablar del archivo XML, hay que comentar su esqueleto, que en este caso se escogió hacerse en XSD por las ventajas que ofrece frente a la otra opcion, el DTD. En la Fig11 se muestra el utilizado.

En la primera parte definimos la lista de riesgos, la cual nos servirá como contenedor de todos los riesgos a almacenar. La información que se decide guardar del riesgo en la mínima y necesaria, un nombre para identificarlo, una pequeña descripción para que dentro del juego tenga un poco más de inmersión el jugador al leer riesgos que se pueden entender, su efecto (la cantidad de puntos que hará perder al jugador) y por último la contramedida.

Esta contramedida tiene los campos casi iguales que el riesgo, solo difieren en que ésta no tiene efecto sino peso, esto es la medida para que el jugador no pueda coger todas las contramedidas.

Del XML no hay mucho que decir, simplemente mostrar en la Fig.12 un ejemplo de riesgo.

En lo referente a la carga de datos, el código de la Fig.13 realiza esta función. Lo primero que se debe de hacer es volcar los datos del xml dentro de una variable para luego ir sacando la información que nosotros deseemos. Debemos coger toda la información relativa a los

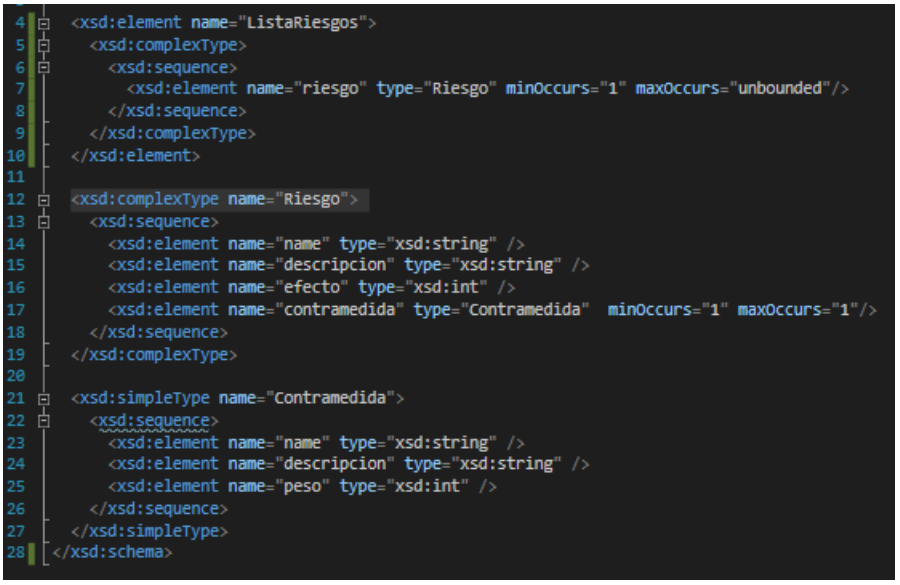


Figura 11: XSD de la lista de riesgos.

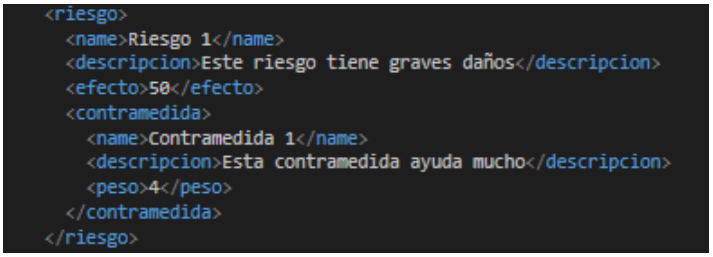


Figura 12: Ejemplo XML de un riesgo.

riesgos, para eso utilizamos el método GetElementByTagName(), con el que obtendremos todos los elementos que tengan ese nombre. Una vez tenemos los elementos que queremos, basta con ir guardando en variables la información deseada para luego crear el objeto con esas variables. La peculiaridad aquí reside en que tenemos que realizar una doble búsqueda, es decir, dentro de los riesgos hay contramedidas luego tenemos que encontrar todos los elementos cuyo nombre sea contramedida para poder obtener su información. Un proceso análogo al de los riesgos. Cuando tenemos ambos objetos simplemente los guardamos en las listas y se finalizaría su carga.

5.5. Clasificación online

Al querer mostrar al usuario una clasificación online, era necesario tener una base de datos donde poder ir guardando los datos de todas las personas que jugases al juego. Normalmente habría que realizar una búsqueda del mercado viendo y comparando que base de datos se adapta más a las necesidades de nuestro proyecto pero en este caso no fue necesaria ya que se propuso utilizar Dreamlo, debido a que no era una herramienta nueva sino que ya había sido usada en otros proyectos cuyas necesidades eran similares a la de este, y por la sencillez para trabajar con ella.

```
private void cargaRiesgosContra()
{
    contramedidas = new List<Contramedida>();
    riesgos = new List<Riesgo>();

    string nameR;
    string descripcionR;
    int efecto;
    string nameC;
    string descripcionC;
    int peso;

    XmlDocument xmlDoc = new XmlDocument();
    ficherosDatos = Resources.Load<TextAsset>("xml/listaRiesgos");
    xmlDoc.LoadXml(ficherosDatos.text);
    XmlNodeList nodos = null;

    nodos = xmlDoc.GetElementsByTagName("riesgo");

    //recorro
    for(int i = 0; i < nodos.Count; i++)
    {
        //extraigo info del xml
        //Primero del riesgo
        nameR = nodos[i].SelectSingleNode("name").InnerText;
        descripcionR = nodos[i].SelectSingleNode("descripcion").InnerText;
        efecto = int.Parse(nodos[i].SelectSingleNode("efecto").InnerText);
        //Despues de la contramedida
        XmlNodeList nodosContr = xmlDoc.GetElementsByTagName("contramedida");
        nameC = nodosContr[i].SelectSingleNode("name").InnerText;
        descripcionC = nodosContr[i].SelectSingleNode("descripcion").InnerText;
        peso = int.Parse(nodosContr[i].SelectSingleNode("peso").InnerText);

        //Se procede a crearse los objetos
        Contramedida c = new Contramedida(nameC, peso, descripcionC);
        contramedidas.Add(c);

        riesgos.Add(new Riesgo(nameR, efecto, descripcionR, c));
    }
}
```

Figura 13: Método con la carga de los datos del XML (ControlRiCon.cs)

Esta pagina web, podría llegar a considerarse como una API REST, proporciona una serie de métodos tanto para el almacenamiento de los datos como para su solicitud. A la hora de almacenar, solo necesitábamos uno de los métodos, el cual limita la aparición del nombre en la tabla a sola vez, mostrando su puntuación más alta por lo que es perfecto para nuestras necesidades. A la hora de solicitar los datos podemos recibirlos en el formato que elijamos (JSON, XML, pipe) en este caso, lo más sencillo era solicitarlo en forma de pipe y una vez recibido encargarnos nosotros de darle el formato adecuado a los datos recibidos como se puede ver la Fig.14.

```
void FormatHighscores(string textStream)
{
    string[] entries = textStream.Split(new char[] { '\n' }, System.StringSplitOptions.RemoveEmptyEntries);
    highscoresList = new Highscore[entries.Length];

    for (int i = 0; i < entries.Length; i++)
    {
        string[] entryInfo = entries[i].Split(new char[] { '|' });
        string username = entryInfo[0];
        int score = int.Parse(entryInfo[1]);
        highscoresList[i] = new Highscore(username, score);
    }
}
```

Figura 14: Método con el tratamiento de los datos (Highscores.cs)

6. Evaluación y pruebas

En este apartado se describirán las pruebas realizadas durante el desarrollo del juego. El hecho de realizar pruebas tiene como objetivo verificar que el sistema cumple con las especificaciones marcadas.

6.1. Pruebas unitarias y de integración

Por una lado las pruebas unitarias tiene como finalidad verificar que la funcionalidad de una pieza de código aislado cumple con lo esperado. Por otro lado, las pruebas de integración tienen como objetivo comprobar que diferentes módulos tienen un comportamiento válido cuando se ejecutan juntos.

Las pruebas unitarias en Unity son difíciles de realizar ya que en programas así es muy complicado aislar piezas de código para ver su funcionamiento, aun así existen programas que permiten acercarse casi a ejecutar pruebas unitarias en las piezas de código deseadas. En este caso no ha sido así, y lo más cercano a una prueba unitaria ha sido la utilización del Log para encontrar fallos durante el desarrollo.

Para las pruebas de integración se ha ejecutado el juego en el editor de Unity. Esto tiene su explicación y es que debido a las características de Unity, cada vez que se pulsa el botón de play en el editor se ejecuta la escena en la que nos encontramos, teniendo que interactuar todos los objetos existentes en ella entre sí. Por lo tanto cuanto se introdujera un objeto nuevo al programa bastaría con comprobar que cuando se ejecuta el juego, no se reportara ningún error al intentar hacer las funcionalidades que implementara ese objeto nuevo. Por lo tanto en este proyecto cuando se introducía un objeto nuevo, las pruebas de integración consistían en probar dentro del propio juego si su comportamiento era el adecuado al interactuar con los demás objetos. En caso afirmativo, se daba por concluida la prueba y en caso negativo, tienes pruebas visuales de la interacción que ha provocado el fallo en el objeto introducido.

6.2. Prueba de sistema

Estas pruebas consisten en verificar el sistema en su conjunto y sobre todo se utilizan para verificar que se cumplan los requisitos no funcionales por lo que en este apartado se explicarán todas las pruebas realizadas para cumplir con los que fueron especificados en la planificación.

6.2.1. Pruebas de rendimiento

Para este tipo de requisitos teníamos definido dos peticiones. Antes de abordar las pruebas para cada requisito, explicar que se va a utilizar la misma herramienta para comprobar ambos. Esta es proporcionada por Unity y recibe el nombre de Profiler. Es una herramienta que tiene como objetivo ayudar al desarrollador a optimizar su juego. Cuando esta está activa y ejecutamos el juego, nos arroja un reporte en función del tiempo de ejecución sobre distintas áreas del juego (CPU, GPU, Memoria, etc). Puede verse los distintos reportes que lanza en la Fig.15.

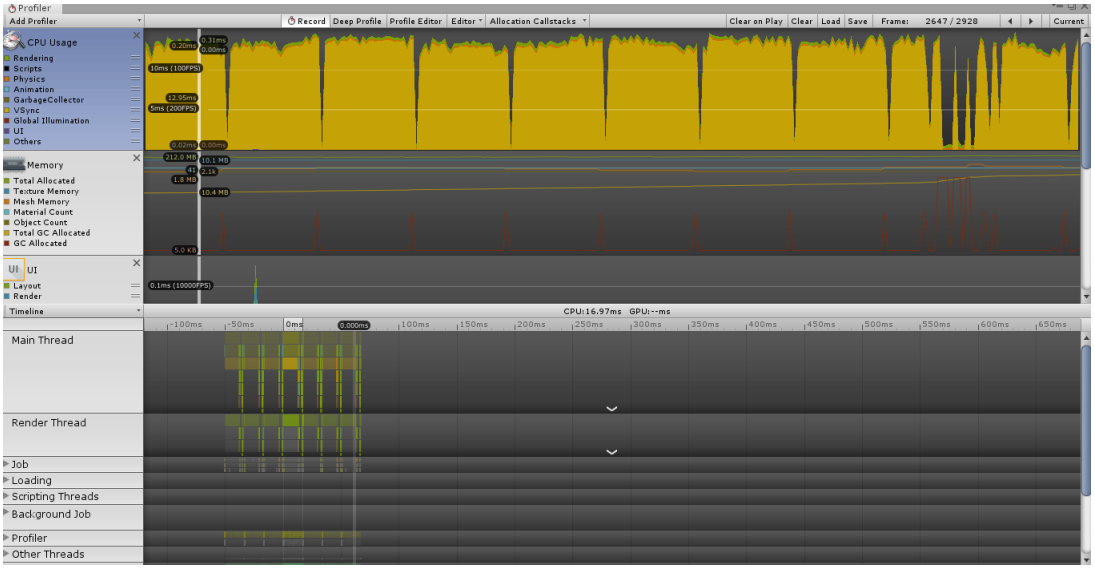


Figura 15: Unity Profiler

Entrando en las pruebas concretas a los requisitos, uno de ellos era que el juego no podía caer de los 30 fps (fotogramas por segundo). Esta es la medida mínima que un videojuego debería de respetar para proporcionarle al usuario una sensación de movimiento natural y suave. Para comprobar que el juego no baja de esta cantidad tenemos que ver en que números se mueve la tasa de refresco de la CPU, para a continuación realizar una sencilla formula, siendo esta fps = 1/T(s), para transformar ese refresco en fotogramas por segundo. Comprobando la gráfica del uso de la CPU vemos que se mueve en un rango de entre 10 y 16 milisegundos (este dato se encuentra justo debajo de la gráfica de la UI), por lo que aplicando la formula concluimos que nos movemos en una cantidad de fps entre 60 y 100. El requisito está más que cumplido pero además el ser capaces de jugar a esta cantidad de fotogramas le brindan al usuario una experiencia gratificante, hablando a nivel de pantalla.

El segundo requisito por comprobar decía que el juego deberá hacer un uso máximo de memoria de 2GB. Para comprobar esto, estando en la pantalla de la Fig.15 debemos seleccionar en *memory* y nos aparecerá un desglose de la memoria como el que aparece en la Fig.16.

Figura 16: Información detallada del uso de la memoria

Nos proporciona datos muy interesantes como saber la memoria que esta siendo usada y quien esta haciendo uso de ella, pero esto no es lo que nos ocupa. Debemos fijarnos en el valor de la tercera fila el cual nos da el valor total que el sistema de memoria esta usando. En este caso nos reporta un valor de 0.95GB, inferior a los 2GB que se nos pedía que no alcanzará por lo tanto se puede dar por cumplido este requisito también.

6.2.2. Pruebas de usabilidad

En este tipo de pruebas teníamos únicamente un requisito, el cual definía que el juego debía de ser apto para todos los públicos. Para probar este punto, se han ido respondiendo a cuestiones típicas que debe cumplir todo juego para poder aparecer como una aplicación para todos los públicos (si existe violencia del juego, si hay imágenes de contenido sexual, si existe humor negro) en todas las especificaciones internacionales (PEGI europeo, ESRB norteamericano). Por lo que se considera para todos los públicos ya que todas las respuestas son negativas.

6.2.3. Pruebas de portabilidad

En cuanto a la portabilidad se pedía que el juego fuera capaz de correr tanto en las principales plataformas de escritorio (Windows, Linux, MacOS) así como en dispositivos android (móviles y tabletas). La prueba de este requisito consistía principalmente en ejecutar el juego en los distintos dispositivos mencionados y verificar que este funcionaba en todos. Una vez se vio que el juego podía ejecutarse en todos los entornos pedidos, se dio la prueba por finalizada y por superada.

6.3. Pruebas de aceptación

En estas pruebas se busca validar que el sistema final cumple con las necesidades y los requerimientos del usuario. Para la realización de estas pruebas se le suministro el producto final al director de proyecto, que en parte actuaría como potencial o posible cliente.

Tras estar jugando un periodo de tiempo aceptable y probar todas las posibles funcionalidades que marcamos en la planificación, se pidió su valoración sobre el juego y la cual concluyo en que cumplía con lo establecido, lo que permitió dar por cerrado el desarrollo del mismo.

Además, cabe mencionar que tambien fue distribuido a conocidos y amigos con el fin de obtener un poco más de feedback, explicándoles antes de todo la idea del juego y que lo exploraran antes de dar una opinión. La gran mayoría aseguraba que el juego funcionaba como era esperado que lo hiciese.

7. Conclusiones y Trabajos Futuros

Una vez llegado a este punto, es necesario valorar como se ha llevado a cabo el trabajo, ver aquello en lo que se ha fallado y por qué, y mirar al futuro cercano para ver que se puede esperar de él.

7.1. Conclusiones

En primer lugar, considero adecuado recordar el objetivo que se planteo al hacer este videojuego. Un juego educativo con el fin de enseñar a aquellos que lo jueguen sobre la gestión de riesgos, añadiendo el entretenimiento a la ecuación. De esta manera no resultaría otro "típico"juego educativo, que termina haciéndose pesado para el jugador.

Puedo dar por cumplido este proyecto, ya que todas las funcionalidades planificadas (como la generación aleatoria del mapa, el almacenamiento en ficheros XML, la utilización de sistemas externos) ha sido realizada e implementada. La parte visual además cubre todas las necesidades acordadas permitiendo al usuario interactuar con el juego de manera sencilla e intuitiva.

El proyecto me ha parecido muy interesante y me permitido trabajar en el campo de los videojuegos, el cual me ha llamado siempre la atención, además de probar la experiencia de enfrentarme al desarrollo de uno yo solo. Se busco realizar un juego con una estética comercial, pero por falta de tiempo se pasó a realizar una parte gráfica no con el aspecto deseado pero que satisface con los objetivos propuestos. Por otro lado, con la parte programática también estoy satisfecho, ya que se ha conseguido realizar todo lo acordado de una manera clara y correcta.

De la realización de este proyecto he aprendido muchas cosas. Por la magnitud de este, me he visto poniendo en practica todas las habilidades que he ido aprendiendo a lo largo del grado, así como descubriendo otras nuevas, ya sea tanto a nivel de software (conceptos relacionados con la programacion y con el uso de programas) como a nivel personal.

Para acabar, me gustaría resaltar lo que este proyecto me ha aportado a nivel personal. El mundo del desarrollo de videojuegos es algo que siempre me ha llamado la atención, pero nunca he sabido si es a donde debía enfocar mi futuro. Tras la realización de este trabajo, tengo la sensación de que quiero seguir formándome y aprendiendo todo lo posible sobre esto. Además, creo que he mejorado el aspecto organizativo. He aprendido a realizar una mejor distribución del trabajo, una mejor planificación de mi tiempo y he desarrollado la capacidad de comprometerme a cumplir unos objetivos previamente determinados.

7.2. Trabajos futuros

Aunque se haya completado de manera satisfactoria este proyecto, siempre existe el margen de mejora, por eso se van a seguir introduciendo nuevas mejoras al juego con el objetivo de incrementar su valor.

Una de las cosas con mayor interés a introducir al juego seria el concepto de temáticas. El fundamento de esto reside en que el usuario pueda escoger antes de jugar una temática especifica, por ejemplo Ïnvernalz toda la apariencia del juego se cambiará automáticamente para darle esa sensación al jugador.

Además, la gestión de riesgos esta dentro de la gestión de proyectos por lo que una mejora bien aceptada consistiría en permitir utilizar a los jugadores sus riesgos y contramedidas personalizadas con el fin de abarcar cualquier área educativa, por ejemplo que el mismo juego pueda ser utilizado en informática pero tambien en arquitectura y cada uno de ellos con su personalización.

Otro aspecto en cual seria positivo trabajar es la introducción de elementos sonoros en el videojuego. Todo videojuego debe de tener sonidos ya que estos dan la posibilidad de crear un ambiente idóneo para las temáticas anteriormente comentadas.

Otro objetivo de futuro sería añadir una funcionalidad que le permitiese al usuario ver una tabla con toda la información relativa al análisis cualitativo de los riesgos existentes en el tablero. Implementar esto tiene como fin ayudar al usuario en el aprendizaje de los análisis cualitativos, mostrándole ejemplos de como deben de realizarse para que así, en siguiente niveles puede desarrollar su propio análisis de manera independiente.

Como ultima mejora, seria interesante añadir un sistema de logros al juego, con el fin de motivar al usuario a conseguir todos ellos y prolongar su estancia en el juego buscando ser aquel que los tenga todos.

Referencias

[1] Sommerville I. *Ingeniería del software*. Addison-Wesley, ninth edition, 2012.

[2] Rusty Harold E. & Scott Means W. *XML in a Nutshelll*. O'Reilly, 2004.

[3] Arrioja Landa N. *Unity: diseño y programación de videojuegos.* RedUSERS., 2013.

[4] Pablo Lledó and Gustavo Rivarola. *Gestión de proyectos*. Pearson Educación, 2007.

[5] Robert C. Martin and Micah Martin. *Agile Principles, Patterns, and Practices in C# (Robert C. Martin)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[6] David R. Michael and Sandra L. Chen. *Serious Games: Games That Educate, Train, and Inform*. Muska & Lipman/Premier-Trade, 2005.

[7] ISO25000. ISO/IEC 25010, 2017. https://iso25000.com/index.php/normas-iso-25000/iso-25010.

[8] Unity Technologies. Profiler, 2018. https://docs.unity3d.com/Manual/Profiler.html.

[9] Unity Technologies. Scene, 2018. https://docs.unity3d.com/Manual/CreatingScenes.html.

[10] Unity Technologies. Prefabs, 2018. https://docs.unity3d.com/Manual/Prefabs.html.

[11] Unity Technologies. Uicanvas, 2018. https://docs.unity3d.com/Manual/UICanvas.html.

[12] Unity Technologies. Editor, 2018. https://unity3d.com/es/unity/editor.

[13] Microsoft. Develop, 2019. https://visualstudio.microsoft.com/es/vs/features/develop/.

[14] Microsoft. Testing-tools, 2019. https://visualstudio.microsoft.com/es/vs/features/testing-tools/.