UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

título del TFG

Presentado por Nombre del alumno
en Universidad de Burgos — 3 de junio de 2021
Tutor: nombre tutor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Nombre del alumno, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de junio de 2021

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice general iii

Índice de figuras iv

Índice de tablas v

Introducción 1

Objetivos del proyecto 3

Conceptos teóricos 5
- 3.1. Secciones . . . 5
- 3.2. Referencias . . . 5
- 3.3. Imágenes . . . 6
- 3.4. Listas de items . . . 6
- 3.5. Tablas . . . 7

Técnicas y herramientas 9

Aspectos relevantes del desarrollo del proyecto 11

Trabajos relacionados 13

Conclusiones y Líneas de trabajo futuras 15
- 7.1. ¿Qué he aprendido sobre el desarrollo de videojuegos (como producto software)? . . . 15
- 7.2. Opinión sobre Unity . . . 17
- 7.3. Líneas de trabajo futuras . . . 20

Índice de figuras

3.1. Autómata para una expresión vacía 6

Índice de tablas

3.1. Herramientas y tecnologías utilizadas en cada parte del proyecto 8

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

Conceptos teóricos

En aquellos proyectos que necesiten para su comprensión y desarrollo de unos conceptos teóricos de una determinada materia o de un determinado dominio de conocimiento, debe existir un apartado que sintetice dichos conceptos.

Algunos conceptos teóricos de LaTeX[1].

3.1. Secciones

Las secciones se incluyen con el comando section.

Subsecciones

Además de secciones tenemos subsecciones.

Subsubsecciones

Y subsecciones.

3.2. Referencias

Las referencias se incluyen en el texto usando cite [**?**]. Para citar webs, artículos o libros [**?**].

[1] Créditos a los proyectos de Álvaro López Cantero: Configurador de Presupuestos y Roberto Izquierdo Amo: PLQuiz

3.3. Imágenes

Se pueden incluir imágenes con los comandos standard de LaTeX, pero esta plantilla dispone de comandos propios como por ejemplo el siguiente:

Figura 3.1: Autómata para una expresión vacía

3.4. Listas de items

Existen tres posibilidades:

- primer item.
- segundo item.

1. primer item.
2. segundo item.

Primer item más información sobre el primer item.

Segundo item más información sobre el segundo item.

-

3.5. Tablas

Igualmente se pueden usar los comandos específicos de LaTeXo bien usar alguno de los comandos de la plantilla.

Herramientas	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
JavaScript	X			
AngularJS	X			
Bower	X			
PHP		X		
Karma + Jasmine	X			
Slim framework		X		
Idiorm		X		
Composer		X		
JSON	X	X		
PhpStorm	X	X		
MySQL			X	
PhpMyAdmin			X	
Git + BitBucket	X	X	X	X
MikTEX				X
TEXMaker				X
Astah				X
Balsamiq Mockups	X			
VersionOne	X	X	X	X

Tabla 3.1: Herramientas y tecnologías utilizadas en cada parte del proyecto

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

El proyecto ha sido llevado a cabo satisfactoriamente. Se han implementado todas las mecánicas de juego que se había especificado en el documento de diseño del juego y todos los elementos propios de un videojuego (como menú y transiciones entre estos y sistemas controladores del juego y el estado del nivel). Es cierto que a nivel artístico puede ser un poco simple, pero como no era el objetivo de este proyecto afrontar el aspecto artístico de un videojuego sino su parte funcional como producto software, se considera un mal menor.

Este proyecto a resultado propio para aplicar todos los conocimientos aprendidos durante la carrera sobre gestión de proyectos y buenas prácticas para el desarrollo del software.

7.1. ¿Qué he aprendido sobre el desarrollo de videojuegos (como producto software)?

Un videojuego es un producto software muy particular por las siguientes razones:

Pruebas

Es un producto del que hacer pruebas es muy complejo, pues todos sus elementos están diseñados para funcionar como conjunto y no como elementos independientes. En el caso de este trabajo hacer pruebas unitarias

ha resultado imposible (al menos con el tiempo con el que se disponía) debido a que los elementos de Unity son un lastre a la hora de hacer pruebas. Todos los elementos de Unity están pensados para interactuar juntos hasta el punto de que no hay ninguna forma de simular frame a frame el flujo del videojuego. Además, intentar capturar momentos del juego que se desean evaluar (por ejemplo la colisión con una pared) y tratarlos como pruebas unitarias es imposible debido a que hay muchos eventos (concretamente de las clases que heredan de MonoBehaviour) cuyo funcionamiento esta íntimamente ligado a los mensajes de Unity y no pueden ser probados sin hacer uso de estos, lo cual es imposible si no se esta ejecutando el juego en el entorno de Unity.

Adicionalmente hay variables a las que no se puede acceder desde un entorno que no sea Unity (como lo es un framework de pruebas). Un ejemplo sería la variable Time.deltaTime. Esto se debe a que son variables que el entorno de Unity van variando durante la ejecución del programa (en el caso del Time.deltaTime antes de que se inicie la fase de llamadas a los métodos Update) y no funcionarán ni tendrán los valores esperados.

Para colmo, la principal característica de los videojuegos es la interacción, con lo que si no se hace uso de inteligencias artificiales que apoyen el proceso de pruebas es imposible su automatización.

Por último, y asumiendo que realizar pruebas fuese posible, los videojuegos se apoyan fuertemente en la concurrencia, por lo que sería muy recomendable hacer uso de frameworks especializados en concurrencia o ejecutar un número muy alto de veces la prueba que se desea hacer buscando posibles caminos alternativos en la ejecución que provoquen fallos en las pruebas (este método no es el más eficiente).

Tiempo de desarrollo

El tiempo de desarrollo de un videojuego para ordenador normalmente puede llevar entre 9 y 24 meses. Puede parecer un intervalo de tiempo muy amplio, como un videojuego no consiste solo en el desarrollo del producto software, sino que consta de más elementos como el desarrollo artístico, el histórico o el desarrollo de todos los elementos sonoros y musicales de los que se va a hacer uso.

El videojuego que se ha desarrollado en este TFG no se ha profundizado en todos estos elementos ajenos al desarrollo del software que consumen tiempo de desarrollo del proyecto, de manera que no se ha implementado

una historia al juego y todos los elementos artísticos se han obtenido de páginas que ofrecen recursos artísticos gratuitamente.

Son estas pequeñas diferencias (además de que el videojuego lo ha desarrollado una sola persona) las que hacen que este proyecto no represente de manera totalmente fiel el proceso de desarrollo de un videojuego en términos tanto de tiempo como de presupuesto.

Calidad del código

Los videojuegos son productos cuyos requisitos cambian continuamente y la implementación de las mecánicas también para adecuarse a estos cambios. Es por ello que es importantísimo que la calidad del código sea la mejor posible y plantearse seriamente si realizar refactorizaciones antes de añadir cada neueva funcionalidad.

Mencionar además que al comienzo del proyecto no se le dio la suficiente importancia a la implementación de las mecánicas como conjunto en vez de como elementos independientes entre sí. Esto ha generado que el producto arrastre una serie de problemas que han dificultado la implementación de las mecánicas y son el origen de la mayoría de bugs del juego. Se debería haber planeado el flujo de ejecución de las físicas y mecánicas correspondientes a la ejecución de un frame que les diese estabilidad. Este error de planificación es la causa de la mayoría de las líneas de trabajo futuras.

7.2. Opinión sobre Unity

Después de haber desarrollado un videojuego en Unity se considera que se posee un dominio suficiente sobre esta herramienta como para ofrecer una opinión solida sobre Unity y lo que ofrece.

Unity es una herramienta que ofrece todos los elementos a muy bajo nivel que se van a necesitar para el desarrollo de un videojuego (tales como reproductores de audio, elementos UI, o la arquitectura de flujo de ejecución del videojuego). Esto hace Unity una herramienta muy útil que ahorra mucho trabajo y tiempo al desarrollador. Unity también se encarga de la exportación del juego a una carpeta con un ejecutable haciendo instantáneo el proceso de creación del ejecutable del juego y facilitando mucho la instalación y ejecución del juego para el usuario (que solo tendrá que descargar la carpeta y ejecutar el fichero .exe dentro de esta).

Sin embargo, este planteamiento tan guiado de Unity resta mucha flexibilidad al desarrollo hasta el punto de ser contraproducente. Esta rigidez presenta los siguientes inconvenientes:

Pruebas

Como se mencionó anteriormente realizar pruebas en Unity es una labor si no imposible, muy difícil, costosa y que consume mucho tiempo, lo cual no esta al alcance de todos los equipos de desarrollo de videojuegos.

GameObject

Los prefabs son realmente útiles para generar GameObjects con la misma configuración, resultando muy útil para asegurar que todos los GameController de todas las escenas sean iguales o que todos los menús de pausa sean el mismo. Sin embargo es una ventaja un poco anecdótica teniendo en cuenta que con una buena implementación de clases esa igualdad entre GameControllers se da por hecho. Pero al tener que ser todos los objetos que se usarán en la escena GameObjects y ser esta clase una suerte de base sobre la que añadir todos los componentes que necesitarán los objetos de la escena, utilizar los prefabs es la mejor forma de asegurar que los GameObjects específicos (como el GameController o el menú de pausa) sea siempre iguales.

Pero los prefabs resultan ideales solo para los GameObjects que están implementados por defecto en las escenas. Durante el desarrollo del videojuego se ha implementado una fábrica de obstáculos que hace uso de los prefabs asociados a los obstáculos para instanciar nuevos obstáculos. Esto funciona perfectamente, pero al haber usado prefabs (y sobre todo estar trabajando con Gameobjects) esa fábrica no esta limitada a instanciar obstáculos, sino cualquier GameObject. Como fábrica general eso esta bien, pero si la intención es instanciar un tipo de GameObject concreto solo (como son los obstáculos) no es la solución ideal, pues nada garantiza que el GameObjects que se va a instanciar sea un obstáculo.

Se pensó en generar una clase hija de GameObject exclusiva para los obstáculos haciendo así que la fábrica instancie esa clase hija y no GameObjects, pero no se puede heredar de GameObject.

MonoBehaviour

La clase MonoBehaviour es la clase general que ofrece Unity para los objetos afectados por el flujo de ejecución del videojuego. Esta clase es gigantesca además de que las clases hijas ni usan ni implementan el 90 % de los métodos que ofrece. Es entendible que el tamaño de esta clase sea tan grande una vez se comprende la cantidad de responsabilidades que maneja esta esta clase (lo cual es una violación del principio de responsabilidad única del principio S de los principios SOLID). Además de la responsabilidad anteriormente explicada MonoBehaviour se encarga de controlar los mensajes que afectan al GameObject al que están asociados y se encarga de responsabilidades menores como representar los Gizmos. Son muchas responsabilidades y de muy distinta índole que probablemente hagan esta clase más compleja de lo que debiera.

Otra de las características de MonoBehaviour es que su constructor funciona de forma anómala llamándose varias veces a pesar de haberse construido (además de la llamada al constructor que se da cada vez que se entra al editor o se vuelve a este desde la ejecución de prueba del juego), convirtiendo al constructor en un método poco fiable el cometido que le corresponde. Este comportamiento tan extraño se soluciona llamando a los métodos Awake y Start en vez de al constructor, pero a costa de dejar de lado el constructor. En lineas generales estos métodos solucionan por completo el problema, pero ha habido clases en las que se ha intentado aplicar el patrón de diseño Singelton resultando imposible su implementación de manera limpia. Debido a esto las clases a las que se aplicó el patrón de diseño Singleton si que pueden tener varias instancias de esa clase pero se fuerza que solo se pueda tener acceso a una de ellas. Esto provoca que haya varias instancias de una clase realizando operaciones pero que solo una de ellas tenga relevancia en la ejecución del programa. Es una solución poco eficiente y no completamente segura, pero es la única solución viable si se desea implementar un Singleton.

¿Se recomienda el uso de Unity?

Unity es una herramienta cuyo principal atractivo es la cantidad de elementos que te ofrece por defecto. Sin embargo, como ya se ha mencionado, no ofrece flexibilidad alguna.

El fuerte de esta herramienta es la cantidad de tiempo que ahorra ofreciendo una implementación a todos los elementos a bajo nivel necesarios para el funcionamiento de un videojuego. Es por ello que es el tiempo de

desarrollo el que decidirá si es recomendable usar esta herramienta o no. Si se tiene poco tiempo para desarrollar el producto Unity da los medios para desarrollar un videojuego muy digno en poco. Sin embargo la poca flexibilidad de Unity es tal, que si se tiene el tiempo suficiente como para desarrollar los elementos a bajo nivel, se recomienda encarecidamente implementarlos por tu cuenta y no hacer uso de Unity.

7.3. Líneas de trabajo futuras

El videojuego desarrollado se podría considerar actualmente terminado (como producto software) salvo por un par de bugs que quedan por solucionar. Sin embargo como videojuego y producto de entretenimiento puede ser un poco escaso. En este aspecto sería recomendable:

- Sustituir todos los sprites y elementos artísticos por unos propios que le den personalidad y estabilidad estética al juego.
- Plantear a necesidad de añadir una historia al juego (aunque en un juego de este estilo la historia no es un elemento especialmente relevante).
- Crear más niveles jugables ya que 6 no son contenido suficiente para considerar el videojuego terminado.
- Barajar la necesidad de generar variantes sencillas de las mecánicas ya implementadas que den variabilidad al juego (como por ejemplo que los obstáculos móviles puedan ir en más direcciones que de derecha a izquierda como de arriba a abajo o que puedan realizar movimientos sinusoidales que hagan más difícil de predecir la ruta que siguen).

A pesar de lo dicho anteriormente si que hay una serie de líneas de trabajo futuras en base a mejorar el producto y reducir la probabilidad de aparición de bugs:

- Como se ha mencionado en el apartado de "¿Qué he aprendido sobre el desarrollo de videojuegos?.[Es]te producto adolece de un planteamiento a nivel global del funcionamiento del videojuego. Sería muy recomendable planear e implementar un sistema de jerarquía de llamadas que ordene y desacoplar la influencia que tienen distintos elementos entre sí.

  Un primer acercamiento a esta jerarquía podría ser:

1. Aplicar gravedad
2. Aplicar el movimiento del Player
3. Aplicar mecánicas del Player
4. Aplicar efectos generados por la colisión entre los objetos

Este cambio solucionaría bugs muy complejos de solucionar actualmente como el hecho de que cuando se entra en una zona de tiempo reducido no se puede saltar, que es provocado debido al solapamiento de las mecánicas que modifican la velocidad del Player.

- Estudiar la posibilidad delegar la reproducción de canciones a elementos externos a Unity como la librería System.Media y no a GameOjects implementados en las escenas que ofrecen un funcionamiento válido pero con defectos importantes como que las canciones que suenan en varias escenas se paran y luego reanudan generando una retroalimentación sonora desagradable al cambiar entre escenas.
- Continuar solucionando bugs, los cuales son una actividad que consume mucho tiempo del desarrollo de un videojuego.
- Discutir la posibilidad de separar la clase TimeManager en dos distintas: una que escale el tiempo global y otra que escale el tiempo por zonas. Son dos responsabilidades distintas y con implementaciones muy diferentes que se están combinando en una sola clase. Se teme que esto pueda generar conflictos que resulten en bugs.
- Crear una estructura de clases con la intención de instanciar objetos en tiempo de ejecución (que apoyen el propósito de las fábricas de objetos) y sean más restrictivos que los prefabs. Es entendible que esta tarea responde a una deseo personal de sustituir una implementación que ya funciona, así que es lógico que la prioridad de esta tarea sea baja.