eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA EN COMPUTACIÓN

TRABAJO DE FIN DE GRADO

Esteganografía en imágenes y sus aplicaciones en videojuegos

Alumno/Alumna: Barruti, Agapios, Agustín

Director/Directora: Pereira, Varela, Juanan

Curso: 2021/2022

Fecha: Donostia, 24, Junio, 2022

Resumen

Muchos videojuegos en línea dependen de puntuaciones, estadísticas, logros y otros tipos de datos para su ecosistema de clasificación, que suele estar generado por la propia comunidad de jugadores. Los jugadores pueden obtener capturas de pantalla del videojuego en el que han conseguido una buena puntuación. Estas capturas de pantalla sirven como reconocimiento.

Sin embargo, existe el peligro de que puedan ser manipuladas o creadas artificialmente (sin haber obtenido la puntuación indicada). Una opción para evitar o al menos mitigar este problema de fraude, consiste en insertar metainformación en la imagen de tal forma que no sea visible a simple vista, pero que pueda obtenerse mediante herramientas adecuadas. Esta es precisamente la base de la esteganografía en imágenes, esconder información en una imagen modificando su composición interna.

Por medio de este proceso podremos dotar a los videojuegos que puedan depender de la verificación visual del logro de un jugador una capa más de confirmación de la información contenida. A través de este sistema, además del nivel de seguridad que añade, ofrecemos la posibilidad de depender únicamente de las propias imágenes para la generación de tablas de clasificación en línea, utilizando puramente los datos contenidos en ellas. De esta forma podemos resolver el problema del posible fraude de puntuaciones y al mismo tiempo facilitar la generación de las tablas clasificatorias.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

La idea de poder ocultar un mensaje para enviarlo de forma segura es un concepto que se ha estudiado y puesto en práctica desde tiempos inmemoriales. Diferentes técnicas se han empleado a lo largo de la historia para poder cumplir este objetivo, quizás siendo la criptografía la técnica más conocida y ampliamente utilizada para ello. Existe otra técnica para lograr esto, la esteganografía. La esteganografía es el arte de ocultar un mensaje mediante un medio para que no se pueda percibir la existencia del mismo, a diferencia de la criptografía que, mediante el cifrado, deja el mensaje a la vista pero de una forma inaccesible.

Esta técnica puede darse en cualquier ámbito y de múltiples formas; sonora, física, digital... Ya desde tiempos de la Grecia antigua se conocen escritos donde se describen técnicas esteganográficas. Se cuenta que, con intención de enviar un mensaje a Esparta (para avisar de la intención de invadir Grecia de Xerxes) que pasara desapercibido, en los tablones de cera donde se escribían mensajes se escribió dicha información en la madera, cubriéndola de cera posteriormente y escribiendo en ella. A simple vista, el único mensaje era el escrito en la cera, pero si se retiraba se podía leer el mensaje oculto.

En tiempos actuales, la técnica de la esteganografía es comúnmente utilizada en formatos audiovisuales, como imágenes o vídeos, normalmente para poder comunicar información sensible de forma segura. Además, con el objetivo de proteger dichos mensajes, en ocasiones se ha optado por mezclar la esteganografía junto con la criptografía, ya que son compatibles la una con la otra. Con el paso del tiempo, gracias a la mejora de la computación y de los formatos de imagen y vídeo, nuevas formas de aplicar la esteganografía y darle diferentes usos se han ido desarrollando, pero siempre manteniendo la misma esencia, ocultar un mensaje mediante un medio para que pase desapercibido.

1.1. Origen del proyecto

El origen de este proyecto nace como una idea propia por un interés personal en la esteganografía y en la escena de los videojuegos competitivos. Por ello, un Trabajo de Fin de Grado (TFG) que tratara estos temas se ofrecía como una posibilidad única y enriquecedora. También el hecho de que fuera una idea propia me motivaba a intentar realizarlo, ya que es un tema que no se ha investigado exhaustivamente en el ámbito de los videojuegos.

Durante todo el grado de Ingeniería Informática se realizan diferentes proyectos, unos más completos que otros, en un contexto académico y de forma prefijada. Normalmente, estos proyectos no son de elección del alumnado, estudian y ponen en práctica tecnologías necesarias para el grado, pero que pueden no suscitar interés a cierta parte del alumnado. Aprovechar dos temas que son de tan alto interés personal para un proyecto en un ámbito diferente al académico, más cercano al ámbito laboral real, y poder investigar y poner en práctica otro tipo de tecnologías no estudiadas durante el grado es una experiencia de mucho valor.

Por lo mencionado acerca de ser unos temas que no se han estudiado exhaustivamente juntos, encontrar un profesor que pudiera ser el tutor de este proyecto fue una tarea ardua, ya que tras contactar a diferentes profesores de múltiples departamentos no parecía que fuera a encontrar a un profesor que tuviera la experiencia necesaria en esteganografía para tutorar este proyecto. Al final, y tras una breve reunión para explicar más detalladamente la idea, el proyecto sale adelante con el profesor Juanan Pereira, miembro del departamento de LSI en la Facultad de Informática de Donostia.

1.2. Descripción y situación del proyecto

Este proyecto se basa en la aplicación de la esteganografía en imágenes en un entorno que no está ampliamente explorado: los videojuegos, en concreto, los videojuegos competitivos.

El sector de los videojuegos cuenta con una amplia y dedicada comunidad de jugadores que exploran los límites de su habilidad y jugabilidad para lograr mayores y mejores logros en el contexto del videojuego que practican. Normalmente, mediante capturas de pantalla o vídeos, estos jugadores demuestran el logro conseguido a la comunidad. Sin embargo, han existido casos de jugadores que han intentado realizar trampas, ya sea mediante un vídeo falso, modificando el juego o modificando digitalmente la imagen que demuestra el mismo.

Con este proyecto se busca dotar a videojuegos del ámbito competitivo de un sistema (o plataforma) que ofrezca la seguridad necesaria para que la veracidad de los logros pueda ser verificada. En concreto, se desarrollará un sistema de capturas de pantalla para un videojuego, al cual se le aplicará la esteganografía para introducir información relevante y verificable sobre la partida.

Además, debido a la naturaleza de competitividad en la que se centrará el proyecto, se creará una plataforma donde podrá aprovecharse la información para generar tablas clasificatorias y contribuir de forma más directa no solo a la seguridad de la escena competitiva, sino también a su desarrollo.

1.3. Motivaciones para la elección del proyecto

Las principales motivaciones que han llevado la elección de este proyecto son las siguientes:

Originalidad de la idea

Una breve investigación sobre el estado del arte de la esteganografía en videojuegos muestra que esta técnica se ha aplicado de forma anecdótica en el sector [21], aplicándose en texturas de modelos 3D en Unity [7] o un videojuego que genera mensajes mediante la distribución del mapa de la partida [6] y que no la aplica en imágenes *per se*.

Por ello, una de las principales razones para realizar un proyecto como este es la originalidad que supone, ya que no existen *a priori* aplicaciones de la esteganografía en imágenes orientadas a los videojuegos competitivos de esta forma. Esto da más libertad a la hora de desarrollarlo y ponerlo en práctica, teniendo mucha más independencia en la toma de ciertas decisiones de diseño, arquitectura, etc.

Retos y aprendizaje autónomo

Otra perspectiva que trae consigo la originalidad de esta idea son los posibles retos que supone un proyecto único. Muchos de los problemas que podrán encontrarse tendrán que resolverse de forma autónoma y con falta de información al respecto, lo que supone una oportunidad de crecimiento y aprendizaje en la materia.

Proyecto con múltiples tecnologías

A diferencia de la universidad, donde los proyectos realizados acostumbran a ser centrados en una tecnología, herramienta o lenguaje de programación concretos, un proyecto de estas características requiere el uso de múltiples tecnologías de forma conjunta para lograr los diferentes objetivos, lo que acerca mucho más al desarrollador al mundo laboral y la forma de

trabajar fuera de lo académico.

De igual manera, al desarrollar con múltiples tecnologías de forma simultánea, el aprendizaje también es múltiple, permitiéndome crecer y mejorar mis habilidades con dichas tecnologías al mismo tiempo.

Cercanía con el sector de los videojuegos

Una gran ventaja de este proyecto es el ámbito en el que está centrado, los videojuegos. En lo personal, la cercanía que tengo con este sector facilita mucho el trabajo, ya que anteriores proyectos personales y en equipo que he desarrollado han sido orientados a videojuegos.

De esta manera, un proyecto de fin de grado centrado en ello me ha proporcionado más motivación, ganas e interés, lo que contribuye a un mejor y más eficiente desarrollo.

2. Planteamiento inicial

En esta sección se presentan los objetivos y las herramientas utilizadas, así como el alcance, la planificación temporal, evaluación económica y de riesgos que se han realizado en este proyecto.

2.1. Objetivos

Este proyecto consta de tres objetivos principales que se explicarán a continuación.

Investigar y probar algoritmos de esteganografía y encriptado para la ocultación de la información

Para explicar este objetivo en detalle, se ha dividido en varios puntos generales:

- **Investigación de algoritmos esteganografía**: Para poder aplicar esta técnica en imágenes correctamente, se han investigado diferentes algoritmos para aplicar la codificación de la información. Tanto el apartado técnico como de seguridad se han evaluado e investigado; dificultad de implementación, seguridad frente a intentos de decodificación, compatibilidad con otras técnicas de ocultación de la información...
- **Pruebas de algoritmos**: Tras la investigación, para familiarizarse con lo aprendido y comprenderlo mejor de cara al desarrollo del producto, se opta por probar el o los algoritmos que más se adecúen al proyecto

a forma de *demo*, para después poder reutilizar el código de forma más eficiente en el producto final.

- **Investigación de técnicas criptográficas**: Como se ha mencionado anteriormente, la ocultación de información mediante técnicas criptográficas es compatible con la esteganografía, por lo que se han investigado diferentes algoritmos y formas de compatibilizar las dos de cara al desarrollo para ofrecer mayor seguridad.

Desarrollar una solución viable para las posibles tablas clasificatorias en línea

De la misma forma que se busca ofrecer más seguridad a la escena competitiva de videojuegos mediante este sistema, también se busca aprovechar el proceso para generar tablas clasificatorias en línea que ayuden a desarrollar a una posible comunidad de videojuegos competitivos.

- **Implementar técnicas para decodificar la información de forma segura**: La decodificación de la información de forma segura es crucial para poder utilizarla *en pos de* la generación de tablas clasificatorias.
- **Dar un sentido al proceso de ocultación**: Simplemente ocultar la información en imágenes no aporta significado al uso de la esteganografía, por lo que es esencial que pueda ser utilizada con un propósito real.
- **Pruebas con usuarios reales**: Tener la oportunidad de desarrollar un producto que pueda ser probado por usuarios reales, obtener feedback y hacer mejoras basándose en ello.

Aplicar las competencias aprendidas en el grado

El objetivo es demostrar la capacidad de las tecnologías que se han estudiado a lo largo de todo el grado para implementarse en un proyecto del mundo real. Además, poniéndolo en práctica en un proyecto es la mejor forma de ser capaz de definir necesidades de *software* que cumplan con los lineamientos de la industria.

Esto requiere aplicar los conocimientos de arquitectura y varias fases en el desarrollo de *software*, incluida la captura de los requisitos del cliente, el análisis y el diseño de acuerdo con esos requisitos, la implementación utilizando lenguajes de programación, optimización de procesos y el testeo mediante pruebas que puedan verificar los resultados.

Por último, todo el proceso debe estar bien documentado para que terceros lo comprendan y colaboren de manera efectiva. El lenguaje científico-técnico que se utiliza debe ser adecuado para el entorno de la informática.

2.2. Planificación

Para la planificación de este proyecto, los diferentes paquetes de trabajo se han distribuido en un diagrama EDT.

Este diagrama EDT cuenta con cinco paquetes de trabajo, de los cuales 4 están distribuidos en dos ramas de trabajo diferentes. La rama de trabajo de la aplicación cuenta con los paquetes de trabajo: estudio de la esteganografía y aplicación en videojuegos. Para la rama de trabajo del sitio web (la plataforma donde se va a hacer uso de la información codificada) tenemos los siguientes paquetes de trabajo: *back-end* y *front-end*. El paquete de trabajo de documentación no pertenece a ninguna rama de trabajo concreta.

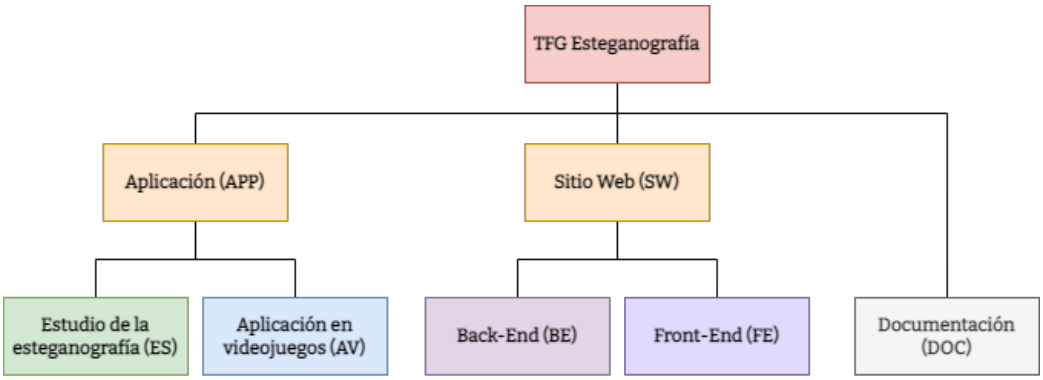


Figura 2.1: Visión general del diagrama EDT.

2.2.1. Rama Aplicación (APP)

Esta rama agrupa todo el proceso de investigación y estudio sobre la esteganografía, así como la puesta en práctica y testeo del desarrollo principal de la aplicación final que la utilizará, en este caso un videojuego compatible con características de la escena competitiva.

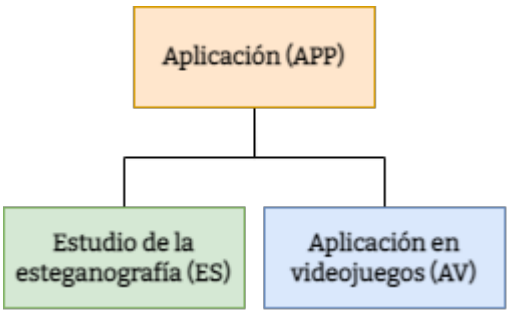


Figura 2.2: Rama **Aplicación** del EDT.

La rama **Aplicación (APP)** agrupa los siguientes paquetes de trabajo:

1. **Estudio de la esteganografía (APP.ES)**:

Estudio de la esteganografía
Rama: «Aplicación» (APP).
Duración estimada: 55 horas.
Descripción: Este paquete de trabajo agrupa todas las tareas necesarias para el estudio, análisis, desarrollo y pruebas de la esteganografía en imágenes.
Salidas/Entregables: Implementación básica del algoritmo de esteganografía a utilizar.
Recursos necesarios: *software* «Visual Studio Code».

Tabla 2.1: Paquete de trabajo «Estudio de la esteganografía» ES.

El paquete de trabajo **Estudio de la esteganografía (ES)** se desglosa en las siguientes tareas:

a) **ES.T1**: Estudio de diferentes algoritmos de esteganografía en imágenes. Con una duración estimada de **20 horas**.

b) **ES.T2**: Elección del algoritmo a implementar. Con una duración estimada de **5 horas**.

c) **ES.T3**: Análisis de los posibles formatos de imagen a utilizar y elección de cuál usar. Con una duración estimada de **10 horas**.

d) **ES.T4**: Pruebas con implementación básica del algoritmo. Con una duración estimada de **20 horas**.

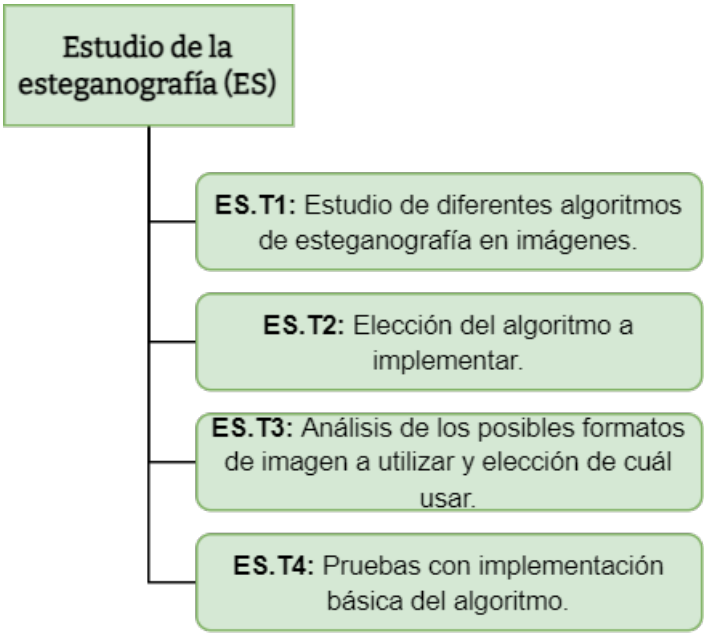


Figura 2.3: Desglose de tareas del paquete ES

2. **Aplicación en videojuegos (APP.AV)**:

Aplicación en videojuegos
Rama: «Aplicación» (APP).
Duración estimada: 130 horas.
Descripción: Este paquete de trabajo agrupa todas las tareas necesarias para el desarrollo de el/los videojuego/s de prueba donde se aplicará la esteganografía.
Salidas/Entregables: Videojuego del ámbito competitivo que utilice la esteganografía de una forma relevante.
Recursos necesarios: *software* «Unity», *git*.

Tabla 2.2: Paquete de trabajo «Aplicación en Videojuegos» AV.

El paquete de trabajo **Aplicación en videojuegos (AV)** se desglosa en las siguientes tareas:

a) **AV.T1**: Elección de los tipos de juegos de demostración para la aplicación. Con una duración estimada de **5 horas**.

b) **AV.T2**: Diseño de la arquitectura de la aplicación. Con una duración estimada de **15 horas**.

c) **AV.T3**: Desarrollo de los juegos de forma básica en Unity. Con una duración estimada de **35 horas**.

d) **AV.T4**: Desarrollo de un sistema de captura de pantalla para las puntuaciones. Con una duración estimada de **10 horas**.

e) **AV.T5**: Aplicar el algoritmo de esteganografía desarrollado en ES.T4 a las imágenes de AV.T4. Con una duración estimada de **40 horas**.

f) **AV.T6**: Aplicación del sistema en el/los videojuego/s de demostración. Con una duración estimada de **25 horas**.

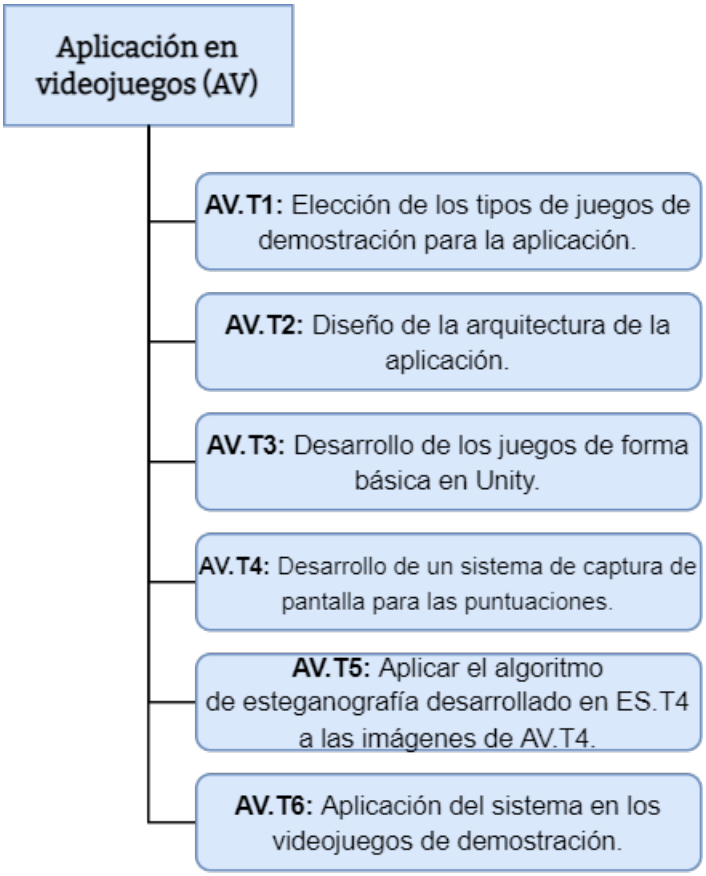


Figura 2.4: Desglose de tareas del paquete AV

2.2.2. Rama Sitio Web (SW)

Esta rama agrupa todo el proceso de desarrollo de la plataforma web que

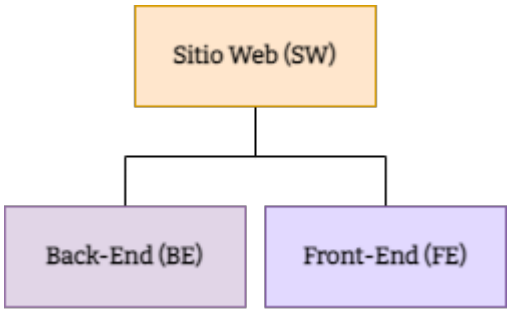


Figura 2.5: Rama **Sitio Web** del EDT.

La rama **Sitio Web (SW)** agrupa los siguientes paquetes de trabajo:

1. Back-end **(SW.BE)**:

back-end
Rama: «Sitio Web» (SW).
Duración estimada: 70 horas.
Descripción: Este paquete de trabajo agrupa las tareas necesarias para el desarrollo del back-end del sitio web encargado de las tablas clasificatorias del videojuego.
Salidas/Entregables: Infraestructura back-end para la plataforma online de tablas clasificatorias.
Recursos necesarios: *software* «Visual Studio Code», Amazon Web Services «AWS» para hosting del servidor, *software* XAMPP para desarrollo web local y **software** «git».

Tabla 2.3: Paquete de trabajo «Back-end» BE.

El paquete de trabajo **Back-end (BE)** se desglosa en las siguientes tareas:

a) **BE.T1**: Configuración inicial del sitio web. Con una duración estimada de **10 horas**.

b) **BE.T2**: Desarrollo de sistema básico de cuentas de usuario y subida de imágenes. Con una duración estimada de **10 horas**.

c) **BE.T3**: Creación de base de datos de tablas clasificatorias de prueba. Con una duración estimada de **5 horas**.

d) **BE.T4**: Desarrollo de sistema de extracción de información esteganografiada de las imágenes. Con una duración estimada de **35 horas**.

e) **BE.T4**: Desarrollo de sistema de guardado per-user de las imágenes y su información. Con una duración estimada de **10 horas**.

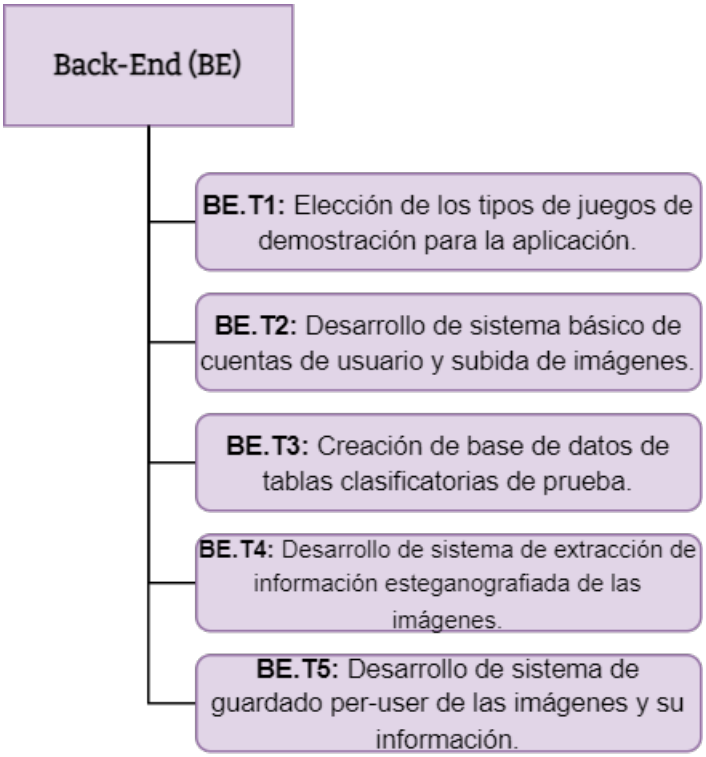


Figura 2.6: Desglose de tareas del paquete BE

2. **Front-end** **(SW.FE)**:

Front-end
Rama: «Sitio Web» SW.
Duración estimada: 20 horas.
Descripción: Este paquete de trabajo agrupa todas las tareas necesarias para el desarrollo del front-end, la interfaz donde podrá visualizarse y explorar la clasificación del sitio web.
Salidas/Entregables: Interfaz/front-end del sitio web.
Recursos necesarios: *software* «Visual Studio Code», Amazon Web Services «AWS» para hosting del servidor, *software* XAMPP para desarrollo web local y **software** «git».

Tabla 2.4: Paquete de trabajo «Front-end» FE.

El paquete de trabajo **Front-end (FE)** se desglosa en las siguientes tareas:

a) **FE.T1**: Creación de estructura HTML básica del sitio web. Con una duración estimada de **10 horas**.

b) **FE.T2**: Diseño de la interfaz. Con una duración estimada de **10 horas**.

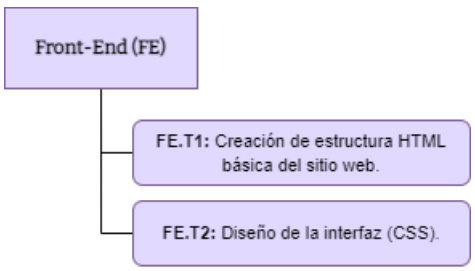


Figura 2.7: Desglose de tareas del paquete FE

2.2.3. Paquete de trabajo Documentación (DOC)

Documentación
Rama: Ninguna.
Duración estimada: 55 horas.
Descripción: Este paquete de trabajo agrupa todas las tareas necesarias para desarrollar la memoria del TFG del proyecto.
Salidas/Entregables: Memoria final del TFG.
Recursos necesarios: Textos, información, tablas, figuras y código que se hayan generado durante el desarrollo del proyecto.

Tabla 2.5: Paquete de trabajo «Documentación» DOC.

El paquete de trabajo **Documentación (DOC)** se desglosa en las siguientes tareas:

1. **DOC.T1**: Recogida periódica de datos e información del desarrollo del TFG. Esta tarea se desarrollará a lo largo del TFG.

2. **DOC.T2**: Elaboración de la memoria. Con una duración estimada de **55 horas**.

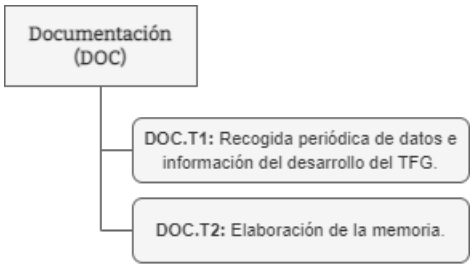


Figura 2.8: Desglose de tareas del paquete DOC

Con esta planificación, la duración total del TFG es la suma de las horas de cada paquete de trabajo, que es de **330 horas**.

2.3. Cronograma

La fecha de comienzo de este proyecto es el 21 de febrero de 2022 y su fecha de cierre es el 21 de junio de 2021.

Algunas tareas del proyecto se realizan a lo largo de todo su desarrollo — como la tarea DOC.T1 «Recogida periódica de datos e información del desarrollo del TFG» — que tiene como fin llevar un registro del trabajo que se realiza cada día, además de sus tiempos. De cara a la realización de la memoria es necesaria para redactar el seguimiento y control del proyecto.

Otras tareas puede ser empezadas antes de la finalización de las previas, además — y esto es importante para poder cumplir las fechas — ciertas tareas se pueden realizar de forma paralela, en este caso los dos paquetes de trabajo back-end y front-end se realizan al mismo tiempo.

Para el proyecto se han definido 3 hitos. Cada uno va asociado a la finalización de determinadas partes importantes de todo el TFG.

- **Hito 1 - Finalización videojuegos con esteganografía**: El grueso del trabajo se sucede en las fechas anteriores a este hito, por lo que para el 16 de mayo de 2022 se marca como objetivo la finalización de todo el trabajo referente a la esteganografía y el videojuego a desarrollar con ella.
- **Hito 2 - Finalización plataforma web para uso de la esteganografía**: Previamente a la finalización del hito 1, se puede comenzar paralelamente la plataforma web que sirva como tabla clasificatoria para poder utilizar la información de las imágenes - y poder probar la extracción de la información esteganografiada -. Se establece que debe estar terminada para el 30 de mayo de 2022.
- **Hito 3 - Finalización de la memoria**: La redacción y finalización de la memoria del TFG marca su fin, por lo que debe estar terminado para el 21 de junio de 2022.

Se ha realizado un diagrama Gantt (Figura 2.9) donde se puede ver la distribución temporal de los paquetes de trabajo con sus respectivas tareas y los hitos que se han detallado en el EDT.

Figura 2.9: Diagrama Gantt de la planificación del proyecto.

2.4. Evaluación económica

Esta sección analiza y evalúa los costeas asociados a la realización del proyecto. En una hipotética comercialización del proyecto que se ha realizado, este análisis puede servir como una referencia de costes.

2.4.1. Mano de obra

En el ámbito laboral, un proyecto de estas características sería realizado por una persona con un puesto de programador, desarrollador de software, o desarrollador de videojuegos. Basándonos en las tablas salariales de la disposición 7829 del Boletín Oficial del Estado número 114 de 2022[1], establece que el puesto de trabajo «programador/analista de aplicaciones informáticas» en la industria textil y de la confección (que utilizaremos como estimación) es de 1.324,34 € mensuales. Este salario está distribuido en 14 pagas, por lo que el salario neto asciende a 1.545,06 €.

Asumiendo que este salario es de una jornada laboral diaria de 8 horas, obtenemos 40 horas semanales. Para poder calcular el sueldo por hora se sigue esta fórmula (Fórmula 2.1):

$$Sueldo_{hora} = \frac{Sueldo_{mes}}{Horas\ de\ trabajo_{día} \times Días\ laborables_{mes}} \tag{2.1}$$

El mes promedio consta de 30 días, de los cuales solo 22 son laborables al ser 8 días festivos por fin de semana. Con esta información, podemos completar la fórmula para obtener el salario por hora (Fórmula 2.2):

$$Sueldo_{hora} = \frac{1.545{,}06\,€}{8\ horas \times 22\ días} = 8{,}77\,€ \tag{2.2}$$

[1]Documento oficial disponible en `https://www.boe.es/boe/dias/2022/05/13/pdfs/BOE-A-2022-7829.pdf`.

De la sección «2.2 Planificación» de este documento sabemos que la duración total de este proyecto es de 330 horas. Conociendo el salario por hora que acabamos de obtener, podemos calcular el coste total del proyecto (Fórmula 2.3):

$$\begin{aligned} Coste\ mano\ de\ obra &= Sueldo\ hora \times Horas\ estimadas \\ &= 8{,}77 € \times 330\ horas = 2.894{,}1 € \end{aligned} \tag{2.3}$$

Así obtenemos un coste de 2.894,1 € por la mano de obra de este proyecto.

2.4.2. Gastos de desarrollo

Sabiendo que este proyecto está pensado para el ámbito de los videojuegos, algunas de las herramientas que se han utilizado (Sección «2.5 Herramientas utilizadas») poseen licencias de pago. Para el desarrollo de este proyecto concreto, ninguna de las herramientas utilizadas han precisado de una licencia de pago al poseer licencias de estudiante y/o personales gratuitas, pero debido a que se está evaluando su coste en un entorno comercial, se realizarán los cálculos con las licencias adecuadas para ello.

El cálculo se dividirá en varias secciones; costes del software, gastos materiales y gastos indirectos.

2.4.2.1. Coste del software

De entre todas las herramientas utilizadas para desarrollar este proyecto, el software Unity, que es lo que se ha utilizado para desarrollar la esteganografía en el videojuego, posee diferentes licencias comerciales dependiendo de las necesidades del usuario; Personal, Plus, Pro y Empresa. En este caso, exceptuando la licencia Personal, se opta por la licencia más económica, Plus, por 399$ al año, que son aproximadamente 379,36 € al año. De los servicios utilizados (GitHub, Overleaf) no se han precisado las licencias con funcionalidades avanzadas.

Sabiendo que la licencia es anual, el gasto de licencia mensual se obtiene de:

$$Licencia_{mes} = \frac{Licencia_{anual}}{12} = \frac{379,36}{12} = 31{,}59 € \tag{2.4}$$

Sabiendo que este proyecto tiene una duración de 4 meses, el coste por la licencia es de 126,36 €.

2.4.2.2. Gastos materiales

Este proyecto ha precisado dos equipos informáticos para el desarrollo: un ordenador personal para el desarrollo local y un servidor para la plataforma web.

A continuación se realiza el cálculo de los gastos de amortización asociados a estos dispositivos.

2.4.2.2.1. Ordenador personal

Este proyecto se ha realizado sobre un ordenador personalizado cuyo valor asciende a 1300 € y tiene una vida útil de 8 años.

La pérdida de valor mensual de este activo se puede obtener mediante la siguiente fórmula (Fórmula 2.5):

$$Amortización\ mensual = \frac{Coste}{Vida\ útil_{meses}} = \frac{1300\,€}{96\ meses} = 13{,}54\,€ \tag{2.5}$$

Sabiendo que el proyecto tiene una duración de 4 meses, se puede obtener la depreciación total con la siguiente fórmula (Fórmula 2.6):

$$\begin{aligned} Amortización\ total &= Amortización\ mensual \times Duración \\ &= 13{,}54\,€ \times 4\ meses = 54{,}16\,€ \end{aligned} \tag{2.6}$$

2.4.2.3. Servidor

Se ha utilizado la plataforma Amazon Web Services (AWS), concretamente el servicio EC2 para la creación de un servidor donde poder alojar y desarrollar la plataforma web.

AWS ofrece un plan gratuito con el cual se consigue acceso a 12 meses gratis de su servicio EC2 - entre otros -. Del servicio EC2 se hará uso de la instancia gratuita de tipo *t2.micro*; con 1 CPU y 1 GB de memoria RAM. Este plan ha sido el escogido para desarrollar y alojar la plataforma web, por lo que no ha supuesto ningún coste extra.

Para el dominio de la plataforma web se ha utilizado Name.com mediante el plan de estudiante de GitHub, GitHub Student Developer Pack[2], que ofrece múltiples servicios durante un periodo de tiempo de forma gratuita, en concreto se hace uso de los 12 meses gratuitos de Name.com. Al hacer uso de esto, no suponen ningún coste extra.

De la misma forma, para obtener el certificado SSL del sitio web, se ha utilizado Cloudflare, concretamente su plan gratuito[3]. Igual que los servicios anteriores, no han supuesto un coste extra.

2.4.3. Gastos indirectos

Esta sección recoge todos los gastos que no están asociados directamente al desarrollo del proyecto, pero que han supuesto un coste extra de forma indirecta; la luz y el internet.

Estos dos gastos se tienen en cuenta únicamente para el ordenador personal, ya que el servidor es gratuito.

2.4.3.1. Luz

Para calcular el gasto de luz del ordenador personal se tiene en cuenta la capacidad de su fuente de alimentación. El ordenador donde se ha desarro-

[2]GitHub Student Developer Pack `https://education.github.com/pack`.

[3]Plan gratuito de Cloudflare `https://www.cloudflare.com/es-es/plans/free/`.

llado cuenta con una fuente de alimentación de 750W de potencia, y se ha utilizado durante todo el proyecto, las 330 horas.

El precio por kilovatio medio en España es de 0,246€. Con la siguiente fórmula, obtenemos el coste total de la electricidad (Fórmula 2.7):

$$\begin{aligned} Coste\ eléctrico &= \sum Horas\ activas \times Consumo\ enérgetico \times Precio \\ &= 330\ horas \times 0,750\ kWh \times 0{,}246€/kWh \\ &= 60{,}88€ \end{aligned} \tag{2.7}$$

2.4.3.2. Internet

Se cuenta con una tarifa de 600Mbps de fibra óptica simétrica contratada por 57€ mensuales. Siendo este proyecto de 4 meses, el coste total del internet es de 228€.

2.4.4. Resumen de los gastos

A continuación se muestra un resumen (Tabla 2.6) de todos los gastos descritos en esta sección.

Concepto	Coste
Mano de obra	2894,1€
Software - Licencia Unity	126,36€
Materiales - Ordenador personal	54,16€
Indirectos - Luz	60,88€
Indirectos - Internet	228€
Total	**3423,5€**

Tabla 2.6: Resumen de los gastos asociados al proyecto.

La estimación final tras la evaluación económica da como resultado unos costes totales de 3423,5€.

2.4.5. Gastos futuros de mantenimiento de la plataforma

De forma adicional, una vez desarrollada y desplegada la plataforma web para el videojuego, el cliente debe hacerse cargo de los gastos de mantenimiento al finalizar el periodo gratuito las herramientas mencionadas previamente.

En concreto, el cliente deberá mantener la instancia de EC2, y puede optar por una instancia de pago con prestaciones superiores [4], ya que la escogida (*t2.micro*) no es escalable ni tiene unas prestaciones altas. Por ejemplo, una instancia de tipo *t3.medium* al ser de tipo T3 ofrece un equilibrio entre recursos de computación, memoria y de red, y están diseñadas para aplicaciones con un uso moderado de la CPU que experimentan picos temporales de uso [5], además es escalable.

Con unas prestaciones de 2 CPUs, 8 GB de memoria RAM y con un costo por hora de 0,040 €, teniendo en cuenta que la plataforma debe estar activa las 24 horas del día, y sabiendo que el año tiene 8760 horas, el costo anual sería de 350,4 €. Dependiendo del tipo (y la cantidad de instancias) que requiera el cliente, el precio variará.

Para mantener el dominio de Name.com pasados los 12 meses gratuitos, el cliente deberá hacerse cargo de los costos. Podrá migrar o mantener el mismo dominio, ya que diferentes dominios conllevan diferentes precios. Como ejemplo, asumiendo que migrará a un dominio más profesional como *.net*, el precio anual ofrecido por Name.com para ese dominio es de 14.86 €.

[4] Precio de las instancias bajo demanda de Amazon EC2: `https://aws.amazon.com/es/ec2/pricing/on-demand/`.

[5] Descripción de tipos de instancias: `https://aws.amazon.com/es/ec2/instance-types/`

2.5. Herramientas utilizadas

Para el desarrollo de este proyecto se han utilizado diferentes herramientas que han facilitado y optimizado su desarrollo. En esa sección se listan las herramientas más relevantes que se han utilizado durante el desarrollo:

- **Unity**: Motor gráfico para el desarrollo de aplicaciones 2D y 3D, normalmente utilizado para el desarrollo de videojuegos.
- **Git**: Sistema de control de versiones. Se ha utilizado GitHub[6] para la gestión de los repositorios.
- **GitKraken**: Software para controlar los diferentes repositorios Git mediante una interfaz de usuario.
- **Visual Studio Code**: El IDE (entorno de desarrollo integrado) utilizado para el desarrollo del código del proyecto.
- **AWS**: Amazon Web Services, colección de servicios de computación de Amazon; se ha utilizado su servicio EC2 para la creación del servidor en la nube que aloja la plataforma web.
- **PuTTY**: Cliente SSH utilizado para conectar con el servidor remoto de AWS.
- **XAMPP**: Paquete de software para desarrollo web local. Entre los que incluye, se han utilizado los siguientes:
  - **Apache**: Servidor web. Es el software utilizado como servidor HTTP para la plataforma web del proyecto.
  - **MySQL**: Sistema de gestión de bases de datos (SGBD) SQL.
- **phpMyAdmin**: Interfaz de usuario para la gestión de bases de datos SQL. El uso de esta herramienta ha facilitado mucho la gestión y pruebas de las bases de dato.
- **LaTeX**: Sistema de composición de textos, utilizado para la creación de documentos. La documentación de este TFG ha sido redactada en el servicio Overleaf[7] el cual utiliza el sistema LaTeX.

[6] GitHub: where the world builds software (`www.github.com`)
[7] Overleaf: online LaTeX Editor (`www.overleaf.com`)

2.6. Análisis de riesgos

Con el fin de poder administrar los imprevistos que puedan surgir durante el desarrollo del proyecto, se ha llevado acabo un análisis que evalúe y gestione los posibles riesgos.

Para ello, se siguen las directrices del estándar para gestión de riesgos ISO 31000, que establece una serie de normas para identificarlos, evaluarlos y mitigarlos.

El proceso de evaluación del riesgo es el siguiente:

- **Prevención**: Pasos a seguir para evitar la posible aparición del riesgo.
- **Plan de contingencia**: En caso de darse, el plan a seguir para poder mitigar su impacto en el desarrollo del proyecto.
- **Probabilidad**: La estimación de la probabilidad de darse.
- **Impacto**: Las consecuencias en caso de darse en el proyecto.

Dado que la materialización de los riesgos supone un impacto directo en las horas, se valorará en base a las horas por día totales que puedan perderse. Si el impacto es de una hora o menos, se considerará **bajo**, si es de 1 hasta 5 horas, el impacto será **intermedio**, si es de entre 5 y 15 horas el impacto será **alto** y finalmente si es superior a 15 horas el impacto será **crítico**.

[7] Publicado por la International Organization for Standardization (ISO): `https://www.iso.org/obp/ui#iso:std:iso:31000:ed-2:v1:es`

A continuación se listan los riesgos identificados junto a su evaluación:

2.6.1. Enfermedad o accidente médico

Cualquier tipo de problema de salud, corroborado por personal sanitario, que impida físicamente continuar con el desarrollo del proyecto.

Prevención

- Seguir hábitos de vida saludables.
- Realizarse pruebas médicas asiduamente.
- Realizar ejercicio físico.

Plan de contingencia

- Acudir al médico para obtener un diagnóstico y tratamiento.
- Si el diagnóstico es favorable y no supone un riesgo para la salud, continuar el trabajo
- Seguir el tratamiento y los tiempos de recuperación marcados por el médico.

Probabilidad

La probabilidad de este riesgo está dividida en dos tipos: enfermedad/lesión que incapacite continuar el trabajo y enfermedad/lesión que lo permita.

- Enfermedad/lesión no incapacitante: 5 % de probabilidades.
- Enfermedad/lesión incapacitante: 2 % de probabilidades.

Impacto

- Enfermedad/lesión incapacitante: 1 hora. Impacto bajo.
- Enfermedad/lesión no incapacitante: Más de 15 horas. Impacto crítico.

2.6.2. Fallo informático

Cualquier tipo de problema que pueda ocurrir al ordenador personal, ya sea virus, funcionamiento incorrecto de algún componente o rotura total.

Prevención

- Realizar análisis de virus periódicamente.
- Realizar análisis de componentes periódicamente.
- Mantener el ordenador en un ambiente que no lo deteriore físicamente.

Plan de contingencia

Dependiendo del tipo de fallo informático, la forma de actuar será diferente.

- En caso de fallo por virus:
  - Hacer uso de un antivirus para intentar eliminarlo/s.
  - Restaurar el ordenador a una copia de seguridad de una fecha anterior.
- En caso de fallo de hardware:
  - Utilizar un ordenador portátil secundario.
  - Reemplazar la pieza dañada.
  - Enviar el ordenador a un técnico.

Probabilidad

La probabilidad de este riesgo está dividida en dos tipos: fallo por virus y fallo de hardware.

- Fallo por virus: 5 % de probabilidades.
- Fallo por hardware: 2 % de probabilidades.

Impacto

- Fallo por virus: Los análisis de antivirus suelen tener una duración aproximada de 1 a 2 horas, por lo que el impacto es intermedio.
- Fallo por hardware: Al contar con un ordenador secundario donde poder trabajar, el impacto solo incurre en el tiempo que lleva configurar todo el proyecto, alrededor de 3 a 4 horas. Impacto intermedio.

2.6.3. Incapacidad de desarrollo

Debido a que se trata de un proyecto donde la esteganografía se aplica en imágenes en videojuegos, se cuenta con muy poca información al respecto, lo que puede incurrir en dificultades o la incapacidad de poder desarrollar lo propuesto.

Prevención

- Estudio e investigación extensos sobre el tema.

Plan de contingencia

En caso de darse el riesgo, la mejor forma de conseguir paliar sus efectos es buscar ayuda de personas más profesionales en el ámbito, como el tutor del TFG o en foros como StackOverflow[8], un foro donde realizar cuestiones sobre programación.

[8]Foro StackOverflow: `https://es.stackoverflow.com/`.

Si esto no ayudara a mitigar el problema, habría que considerar una replanificación de la parte del proyecto que supone una dificultad, intentando simplificarla a modo que sea más sencillo y plausible su desarrollo.

Probabilidad

La probabilidad de este riesgo está dividida en dos tipos: dificultad leve y dificultad grave.

- Dificultad leve: 50 % de probabilidades.
- Dificultad grave: 10 % de probabilidades.

Impacto

- Dificultad leve: Puede variar, pero se estima que menos de 5 horas. Impacto intermedio.
- Dificultad grave: Al suponer una reestructuración de cierta parte del proyecto, se estima una duración de entre 5 y 15 horas. Impacto alto.

3. Ejecución

Esta sección agrupa todo el proceso de estudio de esteganografía, comparativa de algoritmos, estudio de formatos de imagen y la implementación con pruebas de la esteganografía en base a lo estudiado.

3.1. Estudio de la esteganografía

En esta sección se realiza una investigación y análisis sobre la esteganografía. Conociendo el alcance del proyecto, se estudiarán y evaluarán diferentes algoritmos comúnmente utilizados en la industria para aplicar esta técnica en imágenes digitales.

3.1.1. Least Significant Bit (LSB)

Uno de los algoritmos más comunes para aplicar esteganografía en imágenes es el Least Significant Bit (LSB). Este algoritmo consiste en modificar el bit menos significativo de cada píxel de la imagen con un bit del mensaje a esconder de forma secuencial, empezando desde el primer píxel de la imagen hasta que se terminen de codificar todos los bits del mensaje[18] [15] [11].

El bit que es cambiado es el número 24 del píxel, si interpretamos el píxel como un conjunto de 3 bytes, cada uno correspondiendo al rojo, verde y azul (RGB) respectivamente, o el bit número 8 del canal azul. Cabe destacar que no necesariamente hay que seguir la convención de los canales RGB para modificar los bits, es posible optar por cualquier combinación de los canales (por ejemplo GRB o BRG) y modificar los bits consecuentemente.

Al ser el método más conocido y simple, también es el menos seguro, ya que si se presupone correctamente que es los bits se han codificado secuencialmente, obteniendo los bits menos significativos de cada píxel se obtendría el mensaje. Además, si se tuviera la imagen original, se podría obtener el mensaje oculto o partir de ella mediante comparativa de los bits de las imágenes[18].

Esto último tenemos que tenerlo en consideración para nuestra aplicación, ya que nosotros somos los emisores y receptores del mensaje, además de ser los generadores de la imagen, por lo que no se genera ni se provee la imagen original (la imagen sin mensaje) al usuario en ningún momento.

Hay varias formas diferentes de implementar el algoritmo, por ejemplo recurriendo a más bits del píxel, en vez de utilizar únicamente uno. También se puede codificar el mensaje en los píxeles de forma no secuencial; es decir, no empezar desde el primer píxel y continuar progresivamente, utilizando algún método de aleatoriedad conocido entre el emisor y el receptor de la imagen para elegir los píxeles en los que se va a codificar el mensaje. A cualquiera de estos métodos se le puede añadir una capa más de seguridad si se utilizan técnicas de criptografía para encriptar el mensaje a codificar.

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	1	1

Tabla 3.1: Color F30B0B (Hexadecimal).

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	1	**0**

Tabla 3.2: Color F30B0B (Hexadecimal) con su LSB cambiado.

3.1.1.1. Stego 2/3/4 LSB

El LSB, por definición, modifica el bit menos significativo del píxel, pero se puedo aplicar LSB a diferentes umbrales de bits para ganar capacidad de payload [19] [20].

Stego 2 bit LSB

Aplica LSB a los 2 bits menos significativos del píxel. De esta forma ganamos más payload, aunque hay más deterioro visual en la imagen. El deterioro sigue siendo muy pequeño y virtualmente invisible al ojo humano, pero peor al LSB convencional.

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	1	0	**0**	**0**

Tabla 3.3: Color F30B0B (Hexadecimal) de 3.1 con su 2 bit LSB.

Stego 3 bit LSB

Idéntico al 2 bit, pero con los 3 bits menos significativos. El deterioro de la imagen ya es más notable, a pesar de la ganancia de payload.

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	1	**1**	**0**	**0**

Tabla 3.4: Color F30B0B (Hexadecimal) de 3.1 con su 3 bit LSB.

Stego 4 bit LSB

De la misma forma, aplica LSB a los 4 bits menos significativos. El deterioro de la imagen puede llegar a ser visible y notorio.

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	**0**	**1**	**0**	**0**

Tabla 3.5: Color F30B0B (Hexadecimal) de 3.1 con su 4 bit LSB.

3.1.1.2. LSB semi-aleatorio

Para hacerlo más seguro, se puede implementar de forma que los bits no se modifiquen de forma secuencial, sino de forma semi-aleatoria en base a una clave (key) que solo conoce el emisor y el receptor del mensaje, en nuestro caso, nosotros mismos [15] [19].

Considerando esto último, este método nos aportaría más seguridad, ya que la presuposición de que los bits se codifican secuencialmente en los píxeles deja de ser cierta, por lo que es virtualmente imposible obtener el mensaje sin la imagen original.

3.1.1.3. LSB con mensaje encriptado

Otro método que mantiene el uso del sencillo LSB, es encriptar el mensaje a codificar en la imagen mediante algún método de criptografía. Esto no solo proporciona la capa de seguridad del LSB, si no de la criptografía, pero añade trabajo a realizar; qué método de encriptación utilizar, seguridad, complejidad...

Tanto el LSB semi-aleatorio como el LSB con mensaje encriptado pueden ser aplicados con Stego 2/3/4 LSB.

3.1.2. Sequential Colour Cycle (SCC)

Este algoritmo realiza esencialmente el mismo proceso que LSB, cambia el bit menos significativo con un bit del mensaje a codificar, la diferencia radica en tratar los píxeles como tres unidades diferentes a modificar, cambiando el bit menos significativo de cada canal RGB del píxel; LSB al canal rojo (R), LSB al canal verde (G) y LSB al canal azul (B) [19] [20].

Este método proporciona un mayor payload, evita que todo el LSB se le aplique a un solo canal de color; esto no solo proporciona más seguridad de cara a intentos de extracción del mensaje, también proporciona seguridad visual, al no cargar un solo canal de color con el mensaje [20].

Rojo								**Verde**								**Azul**							
1	1	1	1	0	0	1	**0**	0	0	0	0	1	0	1	**0**	0	0	0	0	1	0	0	**0**

Tabla 3.6: Color F30B0B (Hexadecimal) de 3.1 con SCC.

Este algoritmo también se puede adaptar al LSB semi-aleatorio o al Stego 2/3/4 LSB.

3.1.3. Pixel Indicator Technique (PIT)

La técnica de Pixel Indicator Technique utiliza la base de SCC y Stego 2 bit; utiliza los 3 canales RGB con los 2 bits menos significativos en cada uno para aplicar su proceso.

Esta técnica utiliza uno de los canales de color del píxel como indicador; este se encarga de informar si los otros dos canales del píxel contienen bits modificados o no en base a los 2 bits menos significativos del indicador, por lo tanto, con esta técnica 1 canal es utilizado para indicar cuántos bits y en cuántos canales, 1 o 2, los oculta [8].

Canal Indicador	Canal 1	Canal 2
00	No hay datos codificados	No hay datos codificados
01	No hay datos codificados	Bits codificados en los 2 LSB
10	Bits codificados en los 2 LSB	No hay datos codificados
11	Bits codificados en los 2 LSB	Bits codificados en los 2 LSB

Tabla 3.7: Tabla de funcionamiento del canal indicador y los canales de codificación adaptada de [8].

El canal indicador puede ser cualquiera de los 3: rojo, verde o azul. El algoritmo decide qué canal escoger como indicador dependiendo de la longitud (número de caracteres) del mensaje a codificar; si dicha longitud es un número par, utiliza el canal rojo, si es primo utiliza el canal azul y si no se cumple ninguno de los dos, utiliza el canal verde.

La elección de qué canal será el primario y el secundario de los 2 restantes depende de la paridad binaria (cantidad de unos en binario) de la longitud del mensaje.

Tipo de longitud del mensaje	Selección canal indicador (1º fase)	Selección primer y segundo canal (2º fase)	
		Paridad impar	Paridad par
Par	R	GB	BG
Primo	B	RG	GR
Otro	G	RB	BR

Tabla 3.8: Tabla de proceso de selección del canal indicador, primario y secundario adaptada de [8]

A partir de aquí, se itera sobre los píxeles fijándose en los 2 bits menos significativos del canal indicador del píxel, y en base a ellos se codifican los bits del mensaje adecuadamente, hasta que el mensaje esté completamente codificado [8].

3.1.4. Algoritmo Max-Bit

Este algoritmo se basa en las intensidades de los píxeles. Este algoritmo utiliza la criptografía en el mensaje para añadir una capa extra de seguridad. Se encarga de decidir qué píxeles pueden ser susceptibles a contener bits del mensaje codificado en base a las intensidades para minimizar el ruido/deterioro visual en la imagen [3]. El proceso completo del algoritmo se divide en 3 fases; encriptación, cálculo de intensidad de los píxeles y esteganografía.

Tras la fase de encriptación del mensaje, calcula las intensidades todos los píxeles de la imagen. Para ello, convierte la imagen a escala de grises, y obtiene un mapa de los píxeles de la imagen con intensidad alta (que superen un valor determinado por el mensaje a codificar). En la siguiente figura (Figura 3.1) la cuadrícula representa los píxeles de la imagen (cada celda es un pixel) y las celdas con un círculo representan los píxeles seleccionados que superen en intensidad al valor calculado en base al mensaje.

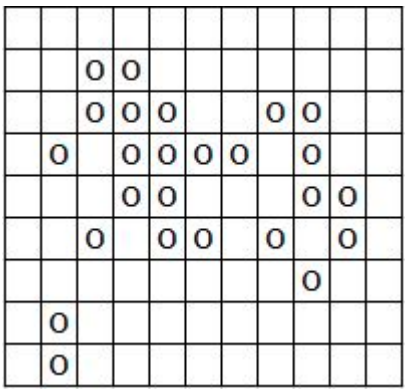


Figura 3.1: Representación del mapa de intensidades de los píxeles adaptado de [3].

Basándose en esa intensidad, también decide cuántos bits de cada canal R, G y B cambia, de 1 a 5 como máximo, para optimizar la cantidad de píxeles modificados sin deteriorar la imagen.

Finalmente, en la fase de codificación del mensaje, utiliza dos semillas aleatorias (dos números); en función del valor de la primera escoge el píxel en el que se va a codificar parte del mensaje (solo de los píxeles de intensidad alta, indicados en el mapa de píxeles seleccionados previamente) y con la segunda semilla (que va de 1 a 5) la cantidad, determinada por la propia semilla (mínimo 1 y máximo 5), de bits a codificar en cada canal de dicho píxel.

3.1.5. Patchwork

Otro sistema diferente a la clásica manipulación de bits no significativos es Patchwork, que comúnmente se utiliza para marcas de agua en imágenes (GPA), aunque se puede aplicar en la esteganografía. Este algoritmo es mucho más complejo; selecciona 2 áreas/grupos de píxeles de la imagen (patch), y modifica las intensidades de cada área, haciendo una más clara y la otra más oscura a partes iguales. El cambio entre estos 2 patches representa codificar un bit en la imagen. Mediante la repetición de este proceso a lo largo de diferentes pares de patches de la imagen, se puede codificar el mensaje completo. [18].

Este método es mucho más seguro, robusto y complejo que LSB, por lo que ofrece mucha más seguridad para ocultar información. A pesar de ello, en comparación al LSB, este método utiliza la imagen completa para ocultar un bit. Se puede dividir la imagen en pequeñas secciones de imagen si es lo suficientemente grande para ocultar más bits y así ocultar el mensaje completo, por lo que es recomendable para mensajes más pequeños y sensibles, además de que el mensaje solo puede codificarse una vez y no repetirse a lo largo de toda la imagen [18].

3.2. Elección del algoritmo

Esta aplicación se basa en las imágenes y en ocultar un mensaje en ellas, por lo que es importante que mantengan su integridad visual al ser codificado un mensaje en ella, y el cambio no sea visible al ojo humano. Relacionado con esto, el propio mensaje tiene que ser codificado de manera que no pueda ser accedido ni manipulado de forma directa por el usuario. En último lugar, la capacidad de la imagen para contener un mensaje tiene que ser acorde a la cantidad de información que queremos codificar; en un videojuego la cantidad de información de una partida puede ser grande, debido a esto tendremos que aprovechar el espacio disponible.

3.2.1. Características a tener en cuenta

Para la elección del algoritmo hay que evaluar los 3 pilares fundamentales de la aplicación; detección visual, seguridad y capacidad de payload.

3.2.1.1. Detección visual

Para la detección visual, el algoritmo o técnica tendrá que ser capaz de codificar el mensaje sin incurrir en un deterioro o variación notable en la imagen original.

Muchos de las técnicas vistas son capaces de lograr esto, algunas con mayor precisión, pero siempre manteniendo la coherencia visual. En este caso el usuario no cuenta con la imagen original y no puede concluir mediante un análisis del histograma de la imagen o de comparativa a nivel de bits entre la imagen original y modificada que hay un mensaje oculto.

Podemos concluir que la única forma que tiene el usuario de detectar que un mensaje está oculto es por inspección visual, por tanto, podremos optar por técnicas que prioricen la ocultación visual y dejen de lado, por ejemplo, la detección por análisis del histograma.

3.2.1.2. Seguridad

En cuanto a la seguridad, hay que tener en cuenta que el algoritmo tiene que lograr, en la medida de lo posible, que el mensaje codificado sea inalcanzable mediante técnicas convencionales de extracción de mensajes esteganografiados.

A priori, si se inserta el mensaje de forma secuencial, un usuario podría obtenerlo simplemente iterando desde el inicio de los píxeles de la imagen hasta que termine encontrándolo. Por ello, hay 2 alternativas para evitar que un usuario analice y extraiga el mensaje completo mediante la extracción secuencial de bits:

- **Repartir los bits semi-aleatoriamente**: Utilizar un algoritmo que inserte los bits en píxeles seleccionados de forma semi-aleatoria y que solo el receptor de la imagen sepa cómo extraer.

- **Utilizar criptografía**: Utilizar técnicas criptográficas para encriptar previamente el mensaje a codificar.

3.2.1.3. Capacidad de payload

Por último, la capacidad de payload es la capacidad máxima que tiene una imagen de esconder un mensaje. Dependiendo del algoritmo/técnica que se utilice de una imagen se podrá aprovechar más el espacio, proporcionando más o menos capacidad de payload.

Nuestros mensajes, dependiendo de la plataforma o videojuego en lo que se apliquen, pueden llegar a contener mucha información, por lo que nuestro algoritmo tendrá que proporcionar una capacidad de payload razonable en base a la imagen.

Tendremos entonces que encontrar un algoritmo que encuentre un equilibrio entre estos 3 factores, sin priorizar ninguno de ellos.

3.2.2. Elección entre los algoritmos estudiados

En primera instancia, el método más robusto y seguro es Patchwork, que divide la imagen en dos secciones (patches), modifica sus intensidades creando un patch más claro y otro más oscuro, y la diferencia entre estas dos secciones codifica un bit.

El problema radica en la capacidad de payload, ya que solo se puede codificar un bit, a no ser que se subdivida la imagen en imágenes más pequeñas para poder codificar más bits. En cualquier caso, la capacidad de payload sigue siendo limitada, y nuestros mensajes pueden llegar a tener una capacidad considerable, por lo que descartamos su uso.

Desechamos la idea de utilizar LSB de forma secuencial, ya que no cumpliría el factor de la seguridad, como se ha mencionado, cualquier persona que itere sobre los píxeles secuencialmente podrá obtener la información. Los algoritmos que utilizan la codificación secuencial, en este caso LSB 1/2/3/4 o SCC, tendrán que implementarse de forma que se apliquen de forma semi aleatoria en los píxeles o con la utilización de técnicas criptográficas, de esta forma no se comprometería la seguridad.

De estos algoritmos sabemos que cuantos más bits sean utilizados por cada byte, más deterioro visual sufrirá la imagen. Basándonos en diferentes estudios de los algoritmos y en la información obtenida, las técnicas que utilizan 3 y 4 bits como LSB 3/4 bit o SCC con 3/4 incurren en un cambio visual de la imagen notable [8], por lo que no serán escogidos.

Entre LSB y SCC sabemos que los dos (aplicados de forma semi-aleatoria o con criptografía) proporcionan seguridad, aunque se ha concluido que SCC es más seguro e indetectable que LSB [20] ya que no se centra en utilizar solo un canal, por lo que prescindiremos de utilizar LSB de 1 y 2 bits. Con SCC al utilizar cada uno de los canales RGB por píxel se cuenta con potencialmente 3 o 6 bits por píxel, dependiendo de si se utiliza 1 o 2 bits por canal. En este caso, la capacidad de payload es suficiente utilizando 1 bit por canal, por lo que se descarta la opción de 2 bits, de esta forma también se utiliza la opción que menos afecta a la detección visual.

Los últimos dos algoritmos que podemos utilizar son Pixel Indicator y Max-Bit. Pixel Indicator ofrece una capacidad de payload similar a SCC pero más seguridad [8]. Max-Bit es la técnica con mejor proporción de capacidad de payload y seguridad[3]. Para la aplicación, cualquiera de estos 3 algoritmos

proporciona la capacidad de payload y seguridad necesarias, además de que ninguno incurre en un deterioro visual notable en la imagen.

Por facilidad de implementación, se descarta el uso de Max-Bit. Con relación a esto, SCC y PIT son los algoritmos más fáciles de implementar. Teniendo en cuenta que SCC de 2 bits y PIT no son susceptibles en cuanto a inspección visual [8] y habiendo decidido que SCC sería de 1 bit por canal y no de 2 - lo que implica que afecta aún menos al deterioro visual -, la aplicación utilizará el algoritmo SCC.

Sabiendo que SCC codifica los bits de forma secuencial, podemos optar por utilizar un sistema de elección de píxeles semi-aleatorio, o utilizar criptografía para el mensaje. Siendo la criptografía la base de la ciberseguridad, se opta por el uso de técnicas criptográficas para añadir la seguridad necesaria al mensaje codificado mediante SCC; de esta forma, si el mensaje es descodificado, deberá ser también descifrado.

Este sería el resultado del proceso con las decisiones tomadas:

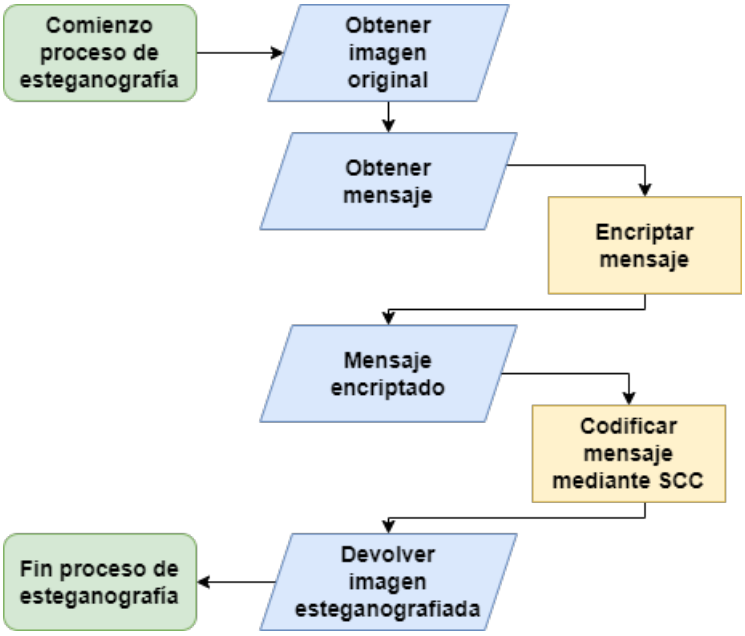


Figura 3.2: Diagrama de flujo del proceso de codificación del mensaje en la imagen.

3.3. Análisis y elección de los formatos de imagen

El formato de imagen a utilizar para nuestra aplicación es esencial, de ello dependen total o parcialmente algunos de los factores a tener en cuenta para nuestro algoritmo, por ejemplo la seguridad o la capacidad de payload. El tipo de compresión que utiliza el formato de imagen puede ser con pérdida (lossy) o sin pérdida (lossless) [18], esta característica es la principal a tener en cuenta para la elección de nuestro formato.

La compresión con pérdida comprime la imagen desechando algunas partes de la misma que pueda encontrar "innecesarias", es decir, que serían indetectables por el ojo humano, para proporcionar tamaños de archivo más pequeños. La imagen resultante, que será de un tamaño más compacto, termina siendo una aproximación extremadamente similar a la original, pero no la misma [18] [17]. El formato más conocido que utiliza este método es JPEG.

La compresión sin pérdida da como resultado la misma imagen que la original, ya que no aproxima partes de la imagen, y aunque realiza compresión, los archivos son de mayor tamaño que los generados con pérdida. Este tipo de compresión se basa en crear un modelo probabilístico [17] de la imagen para poder ser recreado con exactitud sin perder detalle.

Algunos de los formatos más conocidos que utilizan este tipo de compresión son GIF, PNG y WEBP. El formato BMP, que hoy en día aún es utilizado, no utiliza ningún tipo de compresión, por lo que no es ni lossy ni lossless.

3.3.1. JPEG

Para la generación de nuestras imágenes, utilizar JPG nos beneficiaría, ya que estaríamos utilizando posiblemente el formato más utilizado y reconocible en el mundo digital, además de ganar en espacio y seguridad por su tipo de compresión (lossy), lo que lo hace menos propenso a ataques [2].

Precisamente por su tipo de compresión es improbable su uso; las técnicas

esteganográficas existentes con JPG no se basan en la codificación de bits debido a la naturaleza de su compresión, ya que se perdería información irrelevante a la imagen, por lo tanto, perderíamos parcial o totalmente el mensaje codificado.

Para este formato, suelen utilizarse técnicas como Patchwork o Spread Spectrum [16], que no se basan en la modificación de los bits de la propia imagen.

3.3.2. GIF

GIF es uno de los formatos más populares para imágenes, aunque su uso común es en imágenes animadas.

Es un formato que solamente soporta hasta 256 colores mediante una paleta. También soporta transparencia en las imágenes, aunque sin el uso de un canal alfa, únicamente mediante un color completamente transparente en la mencionada paleta.

Para mostrar cada pixel de la imagen, cada uno de ellos está asociado a un color de la paleta (pudiendo repetir el color elegido); ningún píxel puede mostrar un color que no esté en la paleta.

La ventaja de utilizar GIF es que utiliza compresión sin pérdida, por lo que mantenemos la imagen original siempre, con la desventaja de la limitación de colores, además de que otros formatos más modernos cuentan con técnicas de compresión superiores a las de GIF, como PNG [12].

Para la aplicación de técnicas esteganográficas en GIF hay que tener en cuenta la organización de la paleta y de los colores, por lo que no es muy práctico su uso, además de ser un formato reconocible para imágenes animadas y no para imágenes estáticas.

3.3.3. PNG

El formato PNG es muy similar a GIF, pero mejora mayoritariamente todas sus características, exceptuando el soporte para animaciones, con el que no cuenta.

Su soporte de colores es muy superior al de GIF, que era de 8 bits (256 colores), mientras que PNG soporta 8, 16, 24 y hasta 32 [12], siendo la opción de 24 bits la que más colores soporta con un total de 16777216.

Con la opción de 32 bits, las imágenes PNG cuentan con soporte para transparencia mediante el uso de un canal alfa (la imagen pasa de ser RGB a RGBa) [12], por lo que contaríamos con un canal extra a utilizar para esconder nuestro mensaje, lo que podría aumentar nuestra capacidad de payload.

También utiliza un método de compresión sin pérdida, aunque su algoritmo de compresión es superior al de GIF [12], por lo que las imágenes resultantes son de menor tamaño. Es un formato muy reconocible y ampliamente utilizado, a la par con JPEG. Es una de las mejores alternativas para nuestra aplicación.

3.3.4. BMP

BMP es un formato de imagen de mapa de bits introducido para Windows (aunque hoy es soportado por más sistemas operativos).

Como PNG, soporta hasta 32 bits de color; esto incluye canal alfa para transparencia, y como GIF también soporta transparencia por pixel mediante una paleta [5].

Este formato no utiliza ningún tipo de compresión, por lo que es una buena alternativa para codificar información, pero el tamaño de las imágenes generadas será muy superior; por ejemplo, una imagen con una resolución de 1920x1080 píxeles resultaría en un tamaño de aproximadamente 6.2 megabytes.

Con el tiempo este formato está dejando de utilizarse, por lo que no es tan común verlo hoy en día teniendo en cuenta las diferentes y mejores opciones que hay.

3.3.5. WEBP

Una alternativa similar a PNG más moderna es WebP. Este formato está desarrollado por Google ofrece todas las ventajas de PNG, como la transpa-

rencia mediante un canal alfa, además de que tiene soporte para los dos tipos de compresión, con o sin pérdida.

El tamaño de las imágenes WebP con la opción de compresión sin pérdida es menor al de las de PNG, por lo que mejora al formato en dicho aspecto [1].

Esta característica es importante teniendo en cuenta que no solo nuestra aplicación de escritorio (un videojuego) generará imágenes constantemente, sino que nuestra aplicación web tendrá que almacenar y mostrar/cargar esas imágenes, por lo que la carga de trabajo y de almacenamiento es inferior.

3.3.6. Comparativa y elección

En la siguiente tabla se comparan las diferentes características de los formatos analizados, como tipo de compresión, cantidad de compresión y soporte de transparencia.

		JPEG	GIF	PNG	BMP	WEBP
Tipo de compresión	Ninguna				**X**	
	Con pérdida	**X**				**X**
	Sin pérdida		**X**	**X**		**X**
Nivel de compresión	Ninguna				**X**	
	Baja		**X**			
	Media					
	Alta	**X**		**X**		**X**
Soporte de transparecnia	Sin soporte	**X**				
	Paleta		**X**		**X**	
	Canal alfa			**X**	**X**	**X**

Tabla 3.9: Tabla comparativa de formatos de imagen.

Se puede concluir mediante esta tabla que nuestras mejores opciones son PNG y WebP, ya que ofrecen las mismas características. Debido a la cantidad de almacenamiento y carga de imágenes que requiere el apartado web, utilizaremos WebP.

3.4. Desarrollo del algoritmo

En esta sección se detalla todo el proceso que se ha seguido para una primera implementación básica del algoritmo SCC en imágenes. El algoritmo se ha implementado en el lenguaje de programación C#, ya que Unity utiliza el mismo lenguaje, por lo que facilitará mucho el proceso de portabilidad de código cuando sea aplicado al videojuego.

En esta primera implementación del algoritmo no se aplicarán técnicas criptográficas, y si se aplican serán como prueba. Tampoco se utilizarán imágenes con formato WebP, por lo que se ha optado por utilizar imágenes PNG. Se tomará la decisión de qué técnica criptográfica utilizar y se utilizarán imágenes WebP en la implementación final en Unity.

3.4.1. Estructura general

La aplicación cuenta con dos clases principales y una estructura de datos para simular una partida de un videojuego:

- **RunData**: Estructura de datos para simular una partida.
- **ImageManager**: Se encarga de procesar la imagen. También puede generar una nueva imagen dado una serie de píxeles.
- **StegManager**: Se encarga de aplicar la esteganografía a la imagen.

Las siguientes secciones explican detalladamente la estructura de clases de la aplicación.

3.4.1.1. Clase RunData

Para la implementación, se ha creado la estructura de datos *RunData*, que simulará los datos que se pueden guardar en una partida para que la implementación sea más realista.

```
1  public class RunData
2  {
3      public String username {get; set;}
4      public double duracion {get; set;}
5      public long puntuacion {get; set;}
6      public int enemigosEliminados {get; set;}
7      public String nivelHash {get; set;}
8      public String gameVer {get; set;}
9      public bool exito {get; set;}
10
11 }
```

3.4.1.2. Clase ImageManager

La clase ImageManager, dada una imagen, genera un array de píxeles que puede ser devuelto. Además, puede generar una imagen dado otro array de píxeles. Está distribuida de la siguiente manera:

```
1  using System;
2  using System.Drawing;
3
4  namespace TestSteg
5  {
6      public sealed class ImageManager
7      {
8          private Bitmap coverImage;
9          private int coverImage_numBytes;
10         private Color[] coverImage_colors;
11
12         public ImageManager(Image coverImage){}
13
14         public int getNumberOfBytes(){}
15
16         public Color[] getPixels(){}
```

```
17
18          public void SaveNewImage(Color[] newPixeles){}
19
20          private Color[] generateColorArray(){}
21
22      }
23  }
```

A continuación se detalla la estructura de la clase:

- **coverImage**: Imagen a procesar. Se obtiene como bitmap para poder procesar los píxeles con el paquete *System.Drawing* [1].
- **coverImage_numBytes**: Número de bytes de la imagen.
- **coverImage_colors**: Un array de los píxeles de la imagen. Los píxeles se representan mediante la estructura *Color* del paquete *System.Drawing* [2].
- **getNumberOfBytes()**: Calcula el número de bytes de la imagen y se lo asigna a *coverImage_numBytes*.
- **getPixels()**: Procesa la imagen en modo Bitmap y genera el array de píxeles en coverImage_colors.

3.4.1.3. Clase StegManager

La clase StegManager se encarga de realizar todo el proceso de esteganografía a los píxeles de la imagen. Así está organizada:

```
1  using System;
2  using System.Drawing;
3  using System.Text;
4  using System.Text.Json;
5
6  namespace TestSteg
```

[1] Documentación de la clase Bitmap: `https://docs.microsoft.com/es-es/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-6.0`

[2] Documentación de la estructura Color: `https://docs.microsoft.com/es-es/dotnet/api/system.drawing.color?view=net-6.0`

```
7  {
8      public class StegManager
9      {
10         private RunData data;
11         private Color[] pixels;
12         private ImageManager manager;
13         public bool wasDataEncoded = false;
14
15         public StegManager(RunData data, Image coverImage){}
16
17         public StegManager(Image imagen){}
18
19         private Byte[] generateDataBytes(){}
20
21         private bool BitFromByte(int b, int bitNumber){}
22
23         int BoolArrayToInt(bool[] bits){}
24
25         public void encodeData(){}
26
27         public void decodeData(){}
28
29         public void createStegoImage(){}
30
31     }
32 }
```

A continuación, se detalla la estructura de la clase:

- **data**: Una instancia de *RunData*. Esta es la información que se codificará en los píxeles.
- **pixels**: Array de píxeles obtenido del *ImageManager*.
- **manager**: El *ImageManager* encargado de procesar la imagen y devolverla. Con él se generará la nueva imagen con los píxeles modificados.
- **generateDataBytes()**: Convierte el objeto de *RunData* en una string en formato JSON y lo convierte a bytes para que pueda ser codificada.
- **BitFromByte()**: Devuelve el bit en la posición dada del byte como un bool (true: 1, false: 0).

- **BoolArrayToInt()**: Convierte un array de bools a un entero[3].
- **encodeData()**: Realiza el algoritmo SCC para codificar los datos en el array de píxeles *pixels*.
- **decodeData()**: Extrae la información esteganografiada de los píxeles, usado para testeo.
- **createStegoImage()**: Con los píxeles modificados crea, mediante el *manager*, la imagen con la información codificada.

3.4.2. Implementación de SCC

Para la implementación del método *encodeData()*, que es el que realiza el algoritmo SCC en los píxeles, se han seguido los siguientes pasos:

1. Obtener el mensaje en bytes.
2. Obtener el número de bytes del mensaje.
3. Codificar en los primeros 32 bits mediante SCC el número de bytes del mensaje.
4. Codificar el mensaje mediante SCC.

La razón de codificar el número de bytes del mensaje en los primeros 32 bits mediante SCC es facilitar el proceso de extracción de información; si codificamos el número de bytes del mensaje, durante el proceso de extracción de la información se puede indicar hasta qué punto hay que extraer mensaje y dónde parar.

Para obtener el mensaje en bytes, hacemos uso del método *generateDataBytes()*, que genera una string en formato JSON del objeto *RunData*. Del array obtenido, obtenemos su longitud para conocer el número de bytes a codificar en los primeros 32 bits[4].

[3]Obtenido de: `https://www.codeproject.com/Answers/312449/How-to-convert-a-bool-array-to-a-byte-and-further#answer2`

[4]Los primeros 32 bits menos significativos.

```
1 private Byte[] generateDataBytes()
2 {
3     String mensaje = JsonSerializer.Serialize(data);
4     Byte[] mensajeEnBytes = Encoding.ASCII.GetBytes(mensaje);
5     return mensajeEnBytes;
6
7 }
```

```
1 public void encodeData()
2 {
3     int R = 0, G = 0, B = 0;
4     int bit = 0;
5     int cicloRGB = 0;
6     Byte[] mensajeEnBytes = generateDataBytes();
7     int tamanoMensaje = mensajeEnBytes.Length;
8     int pixelActual = 0;
9     bool pixelModificado = false;
10    ...
11 }
```

La siguiente parte del proceso es aplicar el algoritmo SCC para codificar el número de bytes. El proceso es el siguiente:

- Por cada bit desde 0 hasta 31:

  1. Marcar el píxel como no modificado.
  2. Obtener el bit en la posición de la iteración del número de bytes
     - **Si el ciclo RGB está en 0**: Se codifica el bit en el LSB del canal rojo.
     - **Si el ciclo RGB está en 1**: Se codifica el bit en el LSB del canal verde.
     - **Si el ciclo RGB no está ni en 0 ni en 1**: Se codifica el bit del canal azul y se reinicia a 0 el ciclo RGB. Como se han codificado los 3 canales, se inserta el nuevo píxel en el array de píxeles originales sustituyendo al píxel original.

La implementación quedaría de la siguiente manera:

```
for (int i = 31; i >= 0; i--)
{
    pixelModificado = false;
    bit = (BitFromByte(tamanoMensaje, i) == true) ? 1 : 0;
    Console.Write(bit);
    if(cicloRGB == 0) //Codificamos el bit en rojo
    {
        R = pixels[pixelActual].R;
        R = (bit == 1) ? R | 1 : R & ~1;
        cicloRGB++;
    }
    else if (cicloRGB == 1) //Codificamos el bit en verde
    {
        G = pixels[pixelActual].G;
        G = (bit == 1) ? G | 1 : G & ~1;
        cicloRGB++;
    }
    else  //Codificamos el bit en azul
    {
        B = pixels[pixelActual].B;
        B = (bit == 1) ? B | 1 : B & ~1;

        //Sustituimos el pixel y reiniciamos el ciclo
        pixels[pixelActual] = Color.FromArgb(255,R,G,B);
        pixelActual++;
        pixelModificado = true;
        cicloRGB = 0;
    }
}
```

Para la codificación del mensaje el proceso es similar: se realiza para cada carácter (byte), y por cada carácter hay que codificar 8 bits.

```
foreach (Byte b in mensajeEnBytes) //Por cada caracter
{
    for (int i = 7; i >= 0; i--) //8 bits por caracter
    {
    ...   //Algoritmo SCC
    }
}
```

3.4.3. Pruebas en imágenes

Para las pruebas en imágenes se ha creado una imagen PNG de una resolución de 1920 píxeles de ancho y 1080 de alto.

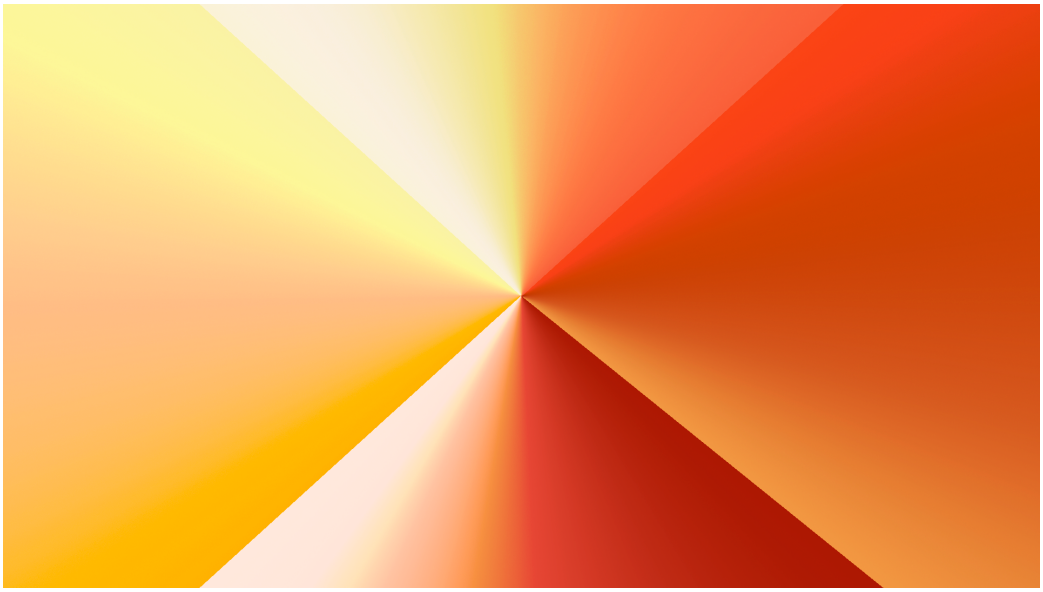


Figura 3.3: Imagen creada para realizar pruebas del algoritmo.

En el programa principal, se ha creado un objeto *RunData* con los siguientes valores:

```
RunData data = new RunData()
{
    username = "Abarruti",
    duracion = 3000,
    puntuacion = 45768,
    enemigosEliminados = 23,
    nivelHash = "027351de5c7ea8e7c5fb602564808b6d",
    gameVer = "2021.1.04f",
    exito = true
};
```

Se lanza el programa para que codifique los datos, genere la imagen esteganografiada y decodifique los datos.

```
1 StegManager manager = new StegManager(data, imagen);
2 manager.encodeData();
3 manager.createStegoImage();
4 manager.decodeData();
```

Tras el proceso, esta es la imagen resultante:

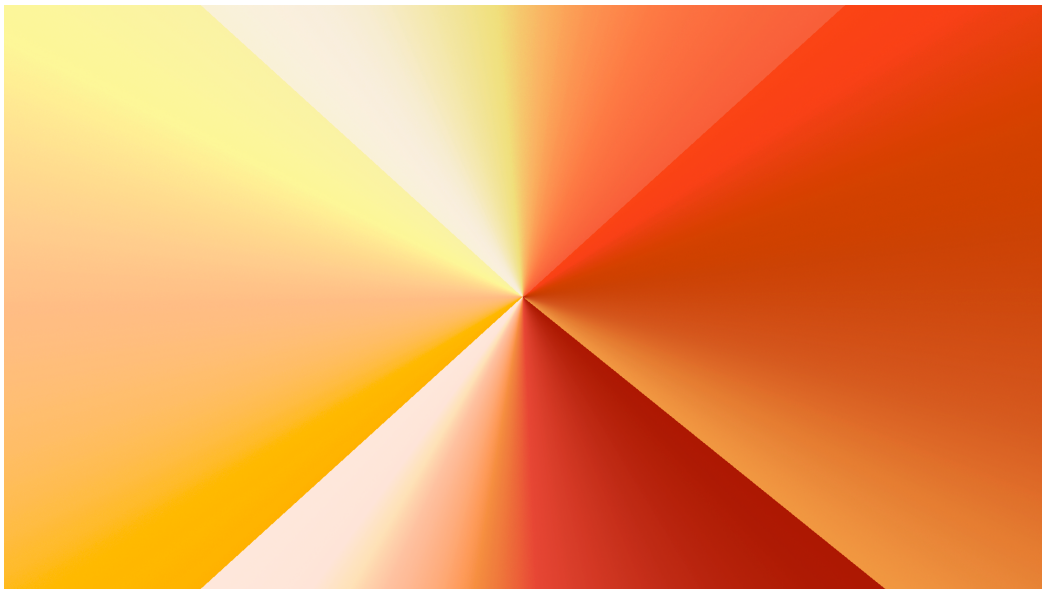


Figura 3.4: Imagen con los datos codificados mediante esteganografía.

Como se puede observar, la imagen es virtualmente idéntica a la original.

Se puede verificar que la información se ha codificado correctamente viendo la salida del programa al realizar el proceso de decodificación.

Figura 3.5: Salida del programa tras el proceso de esteganografía.

También comparando los tamaños de imagen, se puede ver que la imagen esteganografiada tiene un tamaño diferente al modificar los bits de algunos píxeles.

	Imagen original	**Imagen esteganografiada**
Tamaño	2032 KB	2182 KB

Tabla 3.10: Tabla comparativa del peso de las imágenes.

3.4.3.1. Prueba visual de los píxeles modificados

Otra forma que tenemos de confirmar que los píxeles están siendo modificados, es forzar los píxeles que se van a modificar a un color concreto. En este caso se realizará una prueba codificando con color verde (valores RGB: 0, 255, 0) los píxeles que ocultarían el tamaño del mensaje, y de color azul (valores RGB: 0, 0, 255) los píxeles que ocultarían el propio mensaje.

Esta es la imagen resultante de dicho proceso.

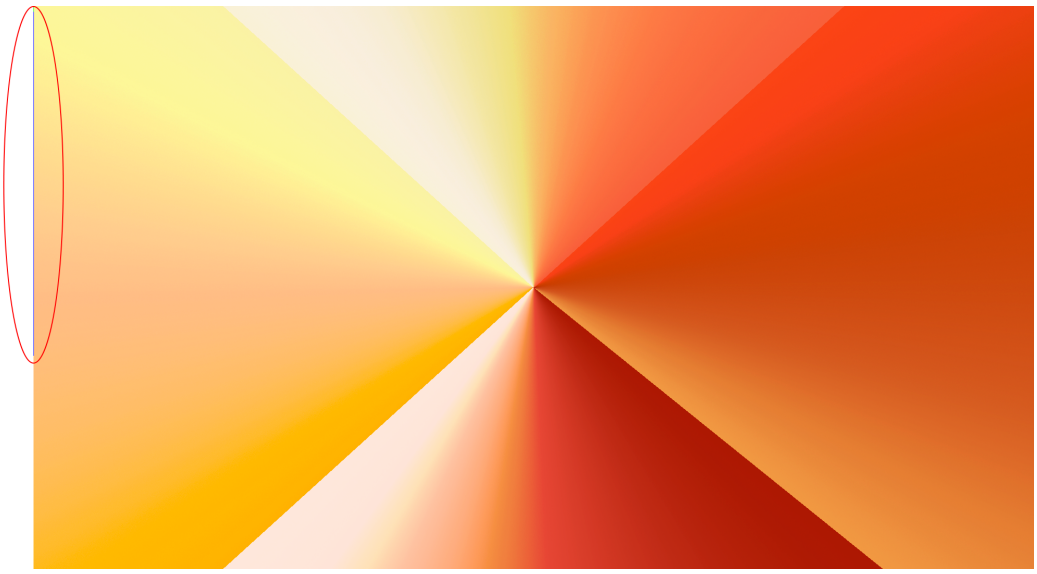


Figura 3.6: Imagen con los píxeles modificados a color.

A primera vista, los píxeles coloreados no son notorios, pero puede verse una línea vertical que empieza desde la esquina superior izquierda y continúa hasta pasada la mitad de la imagen.

Ampliando la imagen (y rotándola), se puede observar con mayor detalle una pequeña línea de píxeles de color verde (el tamaño del mensaje) y otra línea de píxeles mucho más larga de color azul (el mensaje):

Figura 3.7: Ampliación sobre una porción de los píxeles coloreados.

4. Aplicación en videojuegos

En esta sección se detallará el proceso de selección de videojuegos para desarrollar el sistema de esteganografía, así como el desarrollo del mismo.

4.1. Elección del videojuego

Este proyecto busca aplicar todo este sistema de esteganografía al sector de los videojuegos, concretamente al sector de los videojuegos competitivos. Los videojuegos competitivos son aquellos que tienen una o múltiples metas por las cuales varios jugadores compiten. Este tipo de videojuegos son ideales para nuestro proyecto, ya que nos da la posibilidad de sacar capturas cuando se logran esas metas/logros, y esas capturas son las que se pueden utilizar para aplicar el sistema de esteganografía.

4.1.1. Requisitos para el videojuego

El videojuego escogido debe tener ciertas características que lo hagan válido para utilizar el sistema de esteganografía:

- **Ámbito competitivo**: El videojuego debe ser válido en el sector de los videojuegos competitivos. Deber tener un sentido competir en el juego y debe ser posible competir por diferentes objetivos en el mismo juego, a decisión del jugador.
- **Pantallas/Niveles**: Idealmente, el videojuego debe consistir en diferentes niveles o pantallas que sean repetibles.

- **Datos relevantes**: Los datos que genere la propia partida deben ser relevantes y deben tener una necesidad de ser verificados y protegidos contra posibles manipulaciones.

Considerando estos requisitos, una gran parte de los tipos de videojuegos disponibles no serían válidos para el proyecto, aunque la mayoría de estos juegos tienen un alcance de desarrollo muy grande, y no serían realistas para este proyecto. Juegos del estilo *story-driven*; juegos que se centran en la narrativa e historia y su jugabilidad está basada en ella, son los tipos de videojuegos a evitar. Sin embargo, los videojuegos *gameplay-driven*; videojuegos principalmente centrados en la jugabilidad, más rápidos y cortos, normalmente cooperativos, son los tipos de videojuegos que más se ajustan a los requisitos descritos [1].

4.1.2. Elección

Sabiendo qué tipo de videojuegos cumplen los requisitos identificados, hay varias opciones a valorar en esta categoría, ya que hay muchos tipos de juegos de este estilo. Entre los más destacables y populares están:

- **Videojuegos de carreras**: Normalmente juegos puramente competitivos y cooperativos. Podría utilizarse para validar los tiempos de un jugador en un circuito concreto.

- **Videojuegos de acción por niveles**: Aunque videojuegos de acción hay de muchos tipos, los que son centrados en diferentes niveles y se centran en tu rendimiento por partida podrían ser una buena opción, por ejemplo para validar la cantidad de enemigos eliminados, el tiempo en el que se ha realizado, la cantidad de veces que el jugador ha sido eliminado...

- **Videojuegos de plataformas**: Posiblemente, el tipo de juego más conocido y popular, podría usarse por niveles, medir el tiempo por nivel, enemigos eliminados, "monedasrecogidas, cantidad de vidas perdidas...

[1] Story Driven vs. Gameplay Driven Game Design: `https://www.gamedeveloper.com/design/story-driven-vs-gameplay-driven-game-design`

Cualquiera de las opciones sería válida, pero la que mejor cumple con los requisitos es la de un videojuego de plataformas, debido a las posibilidades de guardar diferente información que ofrece e inherentemente ser por niveles.

4.2. Integración en el videojuego

Para integrar el sistema de esteganografía a un videojuego de plataformas se ha utilizado Unity[2], un motor gráfico de desarrollo de aplicaciones 2D y 3D. En esta sección se detalla todo el proceso de creación del proyecto 3D de plataformas y la integración del sistema de esteganografía en él.

4.2.1. Creación del juego de plataformas

Para poder aplicar el sistema de esteganografía, es necesario contar con un proyecto de un videojuego de plataformas existentes, o crear uno. En este caso, Unity ofrece diferentes proyectos de videojuegos ya preparados llamados *Microgames*[3].

Estos proyectos están pensados para aprender a manejar el motor, pero pueden ser utilizados con diferentes propósitos. Entre los *microgames* ofrecidos, nos interesa el *Platformer Microgame*, un videojuego de plataformas básico con todo lo necesario para desarrollar el sistema esteganográfico.

El videojuego ha sido nombrado *Steg Platformer*, haciendo referencia a la esteganografía y al hecho de ser un juego de plataformas.

Figura 4.1: Logo creado para el videojuego

[2]Unity, plataforma de desarrollo en tiempo real: `https://unity.com/es`

[3]Microgames, learn the basics of Unity: `https://learn.unity.com/course/microgames-learn-the-basics-of-unity`

4.2.1.1. Tracking de puntuación

Una vez cargado el proyecto en Unity, contamos con un nivel creado de forma predeterminada. Este es el nivel base que se utilizará, aunque pueden crearse diferentes niveles demo para aprender a manejarse en el proyecto.

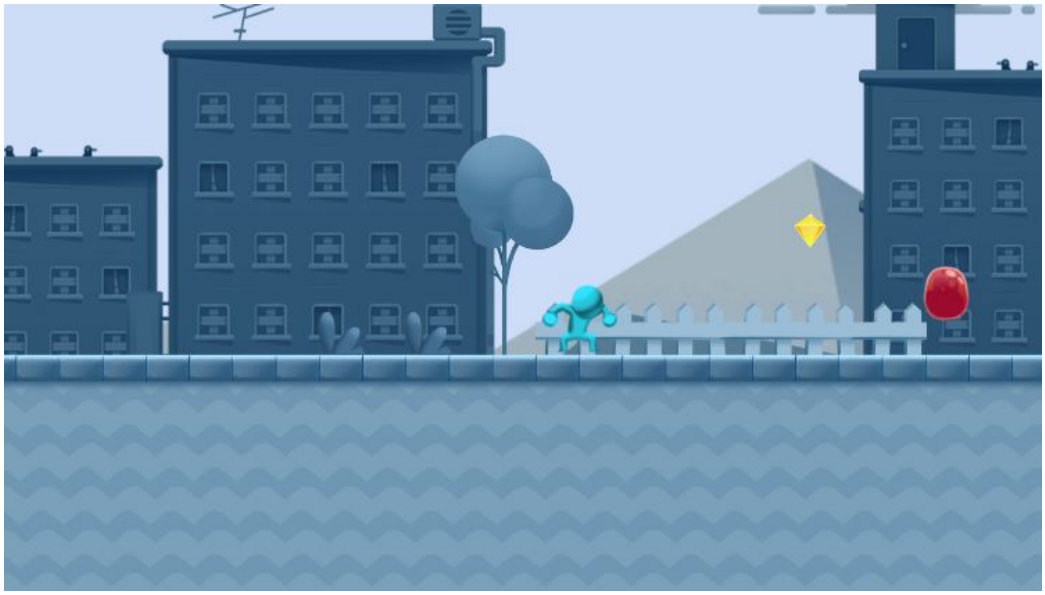


Figura 4.2: Captura del inicio del nivel predeterminado

Como se puede ver en la imagen, el jugador es el personaje azul, las "monedas"son las joyas amarillas y los enemigos los personajes redondos rojos.

Este proyecto *microgame* no cuenta con tracking de puntuación ni de ningún tipo de dato, solamente ofrece una base de jugabilidad para poder ser expandida, por lo que ha sido necesario implementar un sistema de tracking de puntuación y diferentes datos que puedan servir para competir. En concreto, se hace tracking de lo siguiente:

- **Duración de la partida**: Se guarda el tiempo en segundos que ha durado la partida, desde el inicio hasta que se ha completado.
- **Monedas recogidas**: La cantidad de monedas que ha recogido el jugador.

- **Enemigos eliminados**: La cantidad de enemigos que ha eliminado el jugador.

- **Vidas perdidas**: La cantidad de veces que el jugador ha sido eliminado.

- **Puntuación**: En base a las monedas, enemigos y vidas perdidas se calcula una puntuación para la partida en tiempo real, concretamente siguiendo esta fórmula (Fórmula 4.1):

$$Puntuación_{partida} = Monedas_{obtenidas} \times 50 + Enemigos_{eliminados} \times 100 - Vidas_{perdidas} \times 200 \quad (4.1)$$

Una vez añadido el tracking de puntuación al jugador, se muestra por pantalla:

Figura 4.3: Tracking de la partida visible durante la partida.

El videojuego ya recolecta todos los datos de la partida, para poder aplicar el sistema de esteganografía necesita una forma de generar una captura de pantalla a la cual aplicársela, por lo que se ha creado una pantalla de finalización del nivel donde se muestra un resumen de los datos obtenidos. Esta pantalla es la que se usa como base para la generación de una imagen con información esteganografiada.

Figura 4.4: Pantalla de resultados al terminar la partida.

4.2.2. Sistema de capturas de pantalla

Unity ofrece muchas facilidades gracias a su API, lo que facilita mucho el desarrollo de gran parte del proyecto. En el caso del sistema de capturas de pantalla se ha hecho uso de la *ScreenCapture* de la API de Unity[4], que proporciona diferentes funcionalidades para tomar capturas de la pantalla de la aplicación.

[4]Documentación de la clase ScreenCapture: `https://docs.unity3d.com/ScriptReference/ScreenCapture.html`

En concreto, se ha utilizado el método *captureScreenshotAsTexture()*, que toma una captura de pantalla de la aplicación y la devuelve como un objeto de tipo *Texture2D*[5] de Unity, necesario para poder procesar sus píxeles, que se explicará más adelante.

Finalmente, esta funcionalidad se ha añadido al código de la pantalla de resultados, haciendo que sea lanzado una sola vez, tras cargar el primer fotograma.

4.2.2.1. Soporte para imágenes WebP

Con este proceso, se obtiene internamente la imagen como un objeto de tipo *Texture2D*, no se guarda al disco. Para poder guardar la imagen en el disco hay que crear un sistema que procese la Texture2D a una imagen en un formato concreto.

Como se ha mencionado en el análisis de los formatos de imagen (Sección 3.2), el formato que se ha escogido para nuestro proyecto es WebP, sin embargo, Unity no ofrece soporte nativo para este formato[6]. Para solucionar esto, se ha recurrido a una librería externa: *unity.webp* de Eunpyoung Kim[7], que proporciona toda la funcionalidad necesaria para poder procesar y/o generar imágenes WebP desde Unity.

[5]Documentación de la clase Texture2D: `https://docs.unity3d.com/ScriptReference/Texture2D.html`

[6]Unity Asset Types: `https://docs.unity3d.com/Manual/AssetTypes.html`

[7]Librería WebP para Unity: `https://github.com/netpyoung/unity.webp`

Con el uso de esta librería, se ha utilizado el método *EncodeToWebP()*, que devuelve la imagen como un array de bytes en formato WebP. Con ello, se ha creado el método *SaveNewImage()*, que crea la imagen y la guarda al disco.

```
1  public void SaveNewImage(Texture2D p_stegImage)
2  {
3      String newName = DateTime.Now.ToString("yyyyMMddHHmmss")
       + ".webp";
4
5      //Guardar la texture2D como WebP usando la libreria de
       netpyoung para imagenes WebP.
6      byte[] bytes = p_stegImage.EncodeToWebP(-1, out Error
       lError); //-1 para hacer compresion sin perdida.
7      var dirPath = Application.dataPath + "/ScreenShots/";
8      if(!Directory.Exists(dirPath)) {
9          Directory.CreateDirectory(dirPath);
10     }
11     File.WriteAllBytes(dirPath + newName, bytes);
12 }
```

Tras la captura de pantalla, se lanza esta función para que la guarde en el disco.

4.2.3. Integración de la esteganografía

El proyecto ya tiene todos los elementos necesarios para poder aplicar el sistema de esteganografía desarrollado previamente (Sección 3.4).

El sistema previamente desarrollado hacía uso de la librería nativa *System.Drawing* para poder procesar los píxeles de las imágenes, pero tenía varios inconvenientes:

- No es multiplataforma, solo funciona con Windows.
- No proporciona facilidades para procesar píxeles.

Principalmente por no ser multiplataforma se ha adaptado el código para hacer uso de la API de Unity, que es multiplataforma y que de forma nativa proporciona las herramientas para poder procesar las imágenes. En concreto, se han adaptado las siguientes partes del código:

- La clase **Color** de *System.Drawing* por la clase **Color32**[8] de *UnityEngine* para representar los píxeles.
- Se elimina el método *generateColorArray()* ya que Unity proporciona el método *getPixels32()* que devuelve un array de tipo *Color32* de la *Texture2D*.
- Se elimina el método *createStegoImage()* al contar con el nuevo método *SaveNewImage()* mencionado previamente.
- Se puede modificar los píxeles de toda la *Texture2D* mediante el método *SetPixels32()* que proporciona Unity.

Con este sistema preparado y el código ya adaptado, solamente faltaría añadir la seguridad a los datos, es decir, añadir la criptografía.

4.2.4. Sistema de criptografía para las imágenes

El algoritmo utilizado, que es SCC, codifica los bits de forma secuencial en la imagen, lo que supone una brecha de seguridad si alguien decidiera extraer los bits de esa forma, pudiendo obtener los datos y modificarlos.

Para este proyecto, se han probado 2 sistemas diferentes para ofrecer seguridad criptográfica a las imágenes.

[8]Documentación clase Color32: `https://docs.unity3d.com/ScriptReference/Color32.html`

4.2.4.1. Cifrado XOR con clave compartida

En primera instancia, el sistema probado para dar seguridad mediante criptografía a las imágenes fue el uso del cifrado XOR, que aplica la operación XOR a una serie de bits dada una clave de bits.

La operación XOR consiste en la siguiente tabla:

Entrada 1	**Entrada 2**	**Salida**
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Tabla 4.1: Tabla de operación XOR.

El siguiente ejemplo es el resultado de realizar cifrado XOR al carácter 'a' con la clave *10101010*:

CIFRADO XOR DEL CARÁCTER a								
Carácter a	0	1	1	0	0	0	1	1
Clave XOR	1	0	1	0	1	0	1	0
Operador XOR								
Resultado	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**

Tabla 4.2: Cifrado XOR al caracter 'a' con clave *10101010*

Para este proyecto, el videojuego debía conectarse al servidor (la plataforma web), del cual obtendría una clave a utilizar en la operación. Esta clave cambiaba cada semana y las anteriores se iban almacenando en una base de datos junto a la fecha de la semana que fue utilizada para que, llegado el momento de decodificar y desencriptar la información desde el servidor al subir la imagen a la plataforma web, pudiera fijarse en la fecha de realización y desencriptar el mensaje con la clave adecuada.

Este es el diagrama de flujo 3.2 actualizado con el sistema de cifrado XOR:

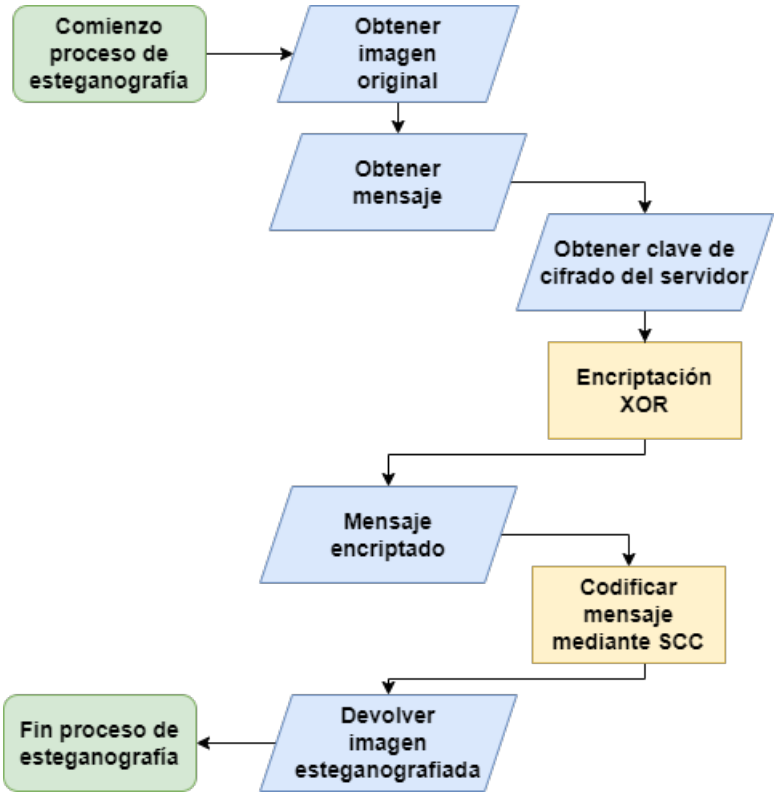


Figura 4.5: Diagrama de flujo actualizado con encriptación XOR.

De acuerdo a lo descrito, este fue el sistema que se implementó inicialmente entre la plataforma web y el videojuego; se obtenía la clave de la semana desde el servidor, se transmitía al juego y con ella se realizaba un cifrado XOR al mensaje que se codificaba en la imagen. Sin embargo, a pesar de la seguridad intrínseca del sistema, suponía un agujero de seguridad considerable, ya que cualquier persona podía capturar la transmisión de la clave al juego cuando se realizaba la petición, por lo que se desechó este sistema.

4.2.4.2. Firma HMAC para validar datos

Tras desechar el sistema de cifrado XOR con claves semanales, se decidió utilizar un sistema similar al de las firmas digitales de archivos, el Hash-based message authentication code (HMAC).

HMAC es un sistema que consiste en crear una función criptográfica que genera un hash en base a ciertos datos que el emisor y el receptor conocen. Si el receptor detecta un hash diferente, implica que los datos han sido modificados. El sistema de funcionamiento es el siguiente:

Echo en falta un ejemplo concreto

1. El emisor genera un hash con los datos mediante cifrado con una clave compartida con el receptor.
2. El emisor envía los datos junto con el hash obtenido.
3. El receptor recibe los datos.
4. El receptor genera un hash con los datos recibidos mediante la clave compartida:
   - Si el hash obtenido es diferente al que ha enviado el emisor, los datos no son válidos.
   - Si el hash obtenido es igual al enviado por el emisor, los datos son válidos.

Para este proyecto, los datos de donde se generará el hash se ha decidido que sean compartidos entre el servidor y el videojuego; por parte del servidor será el nombre de usuario, y por parte del videojuego será la propia cadena en formato JSON de los datos de la partida. Estas 2 cadenas se concatenarán una seguida de la otra de la siguiente forma: *nombre_usuario+:+datos_json*.

Con la cadena resultante, se calcula el hash o resumen criptográfico aplicando la función hash HMAC-SHA256, que devuelve un hash de 256 bits (o 32 bytes).

```
1 private string generateSignatureHMAC()
2     {
3         string checksum = createMD5(mensaje);
4         string username = data.username;
5         string nonHashedSignature = checksum + ':' + username
    ;
6         string hashKey = "-"; //Hash key compartida entre
    servidor y videojuego.
7 
8         //Creamos y devolvemos la firma HMAC
9         HMACSHA256 hashObject = new HMACSHA256(Encoding.ASCII
    .GetBytes(hashKey));
```

```
10          var sign = hashObject.ComputeHash(Encoding.ASCII.
    GetBytes(nonHashedSignature));
11
12          return BitConverter.ToString(sign).Replace("-", "").
    ToLower();
13      }
```

El último paso a realizar es averiguar la forma de enviar junto con la imagen el hash calculado para que el servidor pueda validarla.

Ya que se está utilizando la esteganografía, el hash puede ser codificado mediante SCC seguidamente después de la codificación del mensaje. Del mismo modo que el mensaje, tras codificarlo, se hace lo mismo con la longitud del hash para que el servidor sepa en qué punto parar de decodificar, y a continuación se codifica el hash.

Este sería el diagrama de flujo resultante:

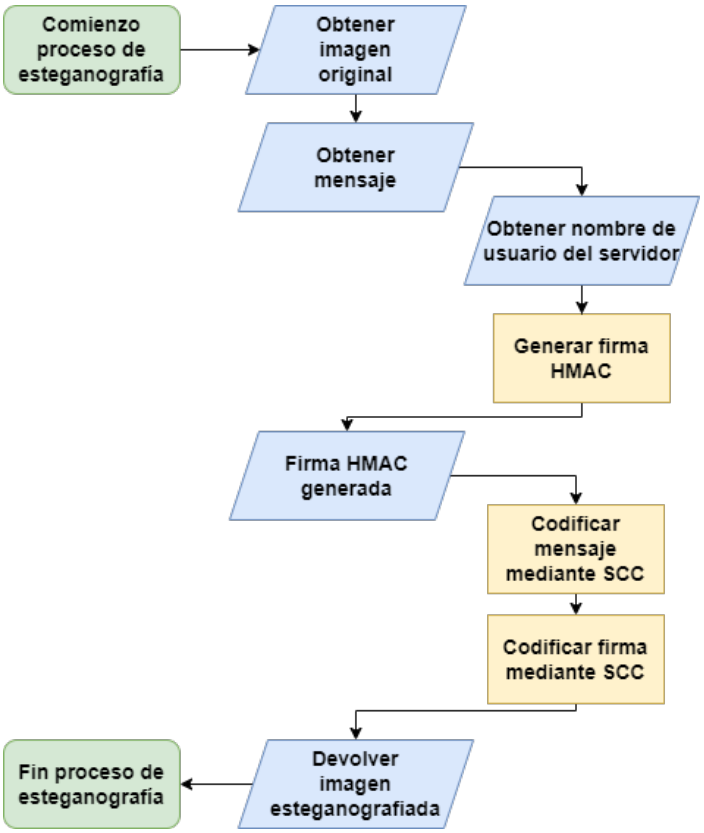


Figura 4.6: Diagrama de flujo con sistema HMAC.

4.2.5. Medidas de seguridad adicionales

También se han añadido dos medidas adicionales de seguridad a las imágenes. Estas medidas no son utilizadas para proteger más los datos, sino que están destinadas a aumentar la seguridad de la plataforma web.

4.2.5.1. Carácter de control

Se codifica, también mediante SCC, un carácter de control, en este caso el carácter 'a', en los primeros 8 bits de la imagen. De esta forma, en la plataforma web si algún usuario intenta subir una imagen que no es válida o aleatoria, el servidor solo decodifica los primeros 8 bits y si el carácter resultante de esos 8 bits no es la 'a', entonces deja de decodificar y manda un aviso de que la imagen no es válida.

4.2.5.2. Patrón de bits en píxeles restantes

Tras realizar absolutamente todo el proceso de codificación (carácter de control, tamaño del mensaje, mensaje, tamaño de firma HMAC y firma HMAC), se codifica mediante SCC en *todos* los píxeles restantes un patrón repetitivo de bits, en este caso un patrón 1/0 (un 1 seguido de un 0 seguido de un 1 seguido de un 0...).

La razón de realizar este patrón en todos los píxeles restantes de la imagen es evitar que una persona modifique visualmente (con algún editor fotográfico) la imagen, ya que aunque la plataforma web verifique la veracidad de los datos de la partida, si se modifican los píxeles que no están codificados se pueden añadir imágenes obscenas o indeseadas, de esta forma se mantiene la integridad original completa de la imagen que genera el videojuego.

4.3. Pruebas en el videojuego

Tras el análisis e implementación de estos sistemas, se han llevado a cabo diferentes pruebas para verificar su correcto funcionamiento.

4.3.1. Prueba de nombre de usuario

Para la firma HMAC, el videojuego necesita conocer el nombre de usuario registrado en la web para poder generar el hash. Esto permite que cada puntuación solamente pueda ser subida estrictamente por la persona que la realizó.

Para ello, el sitio web genera una clave de 32 bytes única por cuenta de usuario. Al iniciar el videojuego pide que se introduzca esta clave, con la que realiza una conexión al servidor y mediante la búsqueda en la base de datos el servidor devuelve el nombre de usuario al que pertenece. Esta clave es única e intransferible.

Para verificar el funcionamiento de este sistema, se ha creado una cuenta de usuario llamada "test".

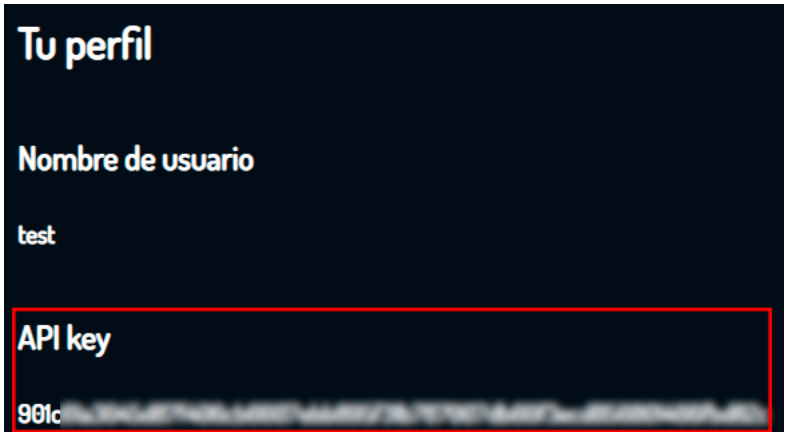


Figura 4.7: Cuenta de usuario con clave (oculta).

Esa clave (API Key en la imagen) es la que se introduce al lanzar el juego, obteniendo el nombre de usuario como respuesta.

Figura 4.8: Clave introducida en el videojuego.

Una vez introducida la clave y escogido el nivel a jugar, al jugador se le muestra su nombre de usuario obtenido del servidor en el menú superior, como se ha mostrado en la figura 4.3 de la sección 4.2.

De igual forma, utilizando el inspector[9] de Unity se puede verificar que el nombre de usuario se ha obtenido correctamente y está almacenado para realizar la futura firma HMAC.

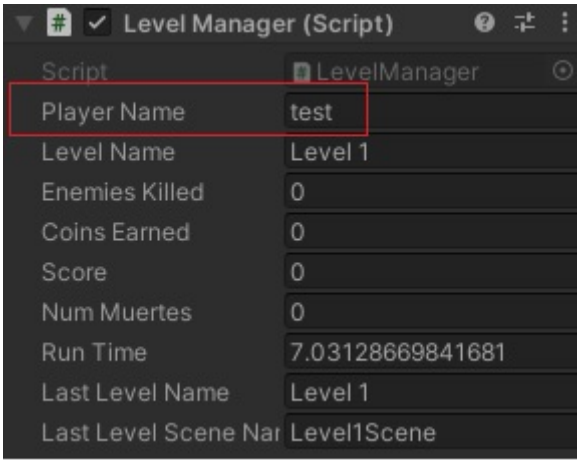


Figura 4.9: Nombre de usuario en el inspector de Unity.

[9]Inspector de Unity: `https://docs.unity3d.com/Manual/UsingTheInspector.html`

4.3.2. Pruebas de formatos de imagen

Tras el análisis de los formatos de imagen realizado en 3.3, se han comparado diferentes formatos, en concreto se han comparado PNG y WebP, ya que son los dos mejores formatos que podían utilizarse y ofrecían virtualmente las mismas funcionalidades.

Para la plataforma web, la parte más importante de cara a las imágenes es el almacenamiento, cuanto menos pesen las imágenes, menos espacio ocupan en el servidor, y menos carga de trabajo para la plataforma. Por lo tanto, se han comparado el peso de las imágenes generadas tanto en PNG y en WebP.

Adicionalmente, también se ha generado la misma imagen en JPEG a pesar de no ser un formato válido para la esteganografía para poner a prueba la compresión sin perdida de WebP.

Esta es la imagen generada:

Figura 4.10: Imagen resultante, idéntica en los 3 formatos.

La cual ha tenido los siguientes pesos en WebP, PNG y JPEG:

	WebP	**JPEG**	**PNG**
Peso	138 KB	144 KB	355 KB

Tabla 4.3: Tabla comparativa de pesos de diferentes formatos.

El formato WebP es el que produce la imagen con menor peso de los 3, incluso mejorando al formato JPG.

4.3.3. Pruebas de las medidas de seguridad

Para la integración de la esteganografía en el videojuego se han añadido diferentes medidas de seguridad; una de cara a proteger los datos y dos de cara a proteger la plataforma web.

Para proteger los datos, se ha desarrollado el sistema de firma HMAC, que devuelve un hash que el servidor debe de validar. Esta cadena se codifica en la imagen de la misma forma que el mensaje con la esteganografía, primero la longitud de caracteres (bytes) de la cadena, y después la cadena en sí.

Para proteger la plataforma web, se han añadido 2 medidas, el carácter de control, y el patrón 1/0 en los píxeles libres. El carácter de control se codifica mediante esteganografía antes que cualquier otro proceso de codificación, y el patrón 1/0 se codifica después de todos los procesos de codificación.

El proceso final resultante es el siguiente:

1. Codificar mediante SCC el carácter de control.
2. Codificar mediante SCC el tamaño del mensaje.
3. Codificar mediante SCC el tamaño de la firma HMAC.
4. Codificar mediante SCC el mensaje.
5. Codificar mediante SCC la firma HMAC.
6. Codificar mediante SCC el patrón 1/0 en los píxeles restantes.

Para comprobar esto, de forma similar a como se comprobó en la sección 3.4, se van a colorear los píxeles que son esteganografiados para poder ver los cambios directamente. En concreto se van a colorear de la siguiente manera:

- **Carácter de control**: Amarillo.
- **Tamaño del mensaje**: Verde.
- **Tamaño de la firma HMAC**: Morado.
- **Mensaje**: Azul.

- **Firma HMAC**: Naranja.

- **Patrón 1/0**: Rojo.

El resultado es una imagen mayoritariamente roja, ya que el patrón 1/0 ocupa la mayoría de los píxeles:

Figura 4.11: Imagen resultante con colores fijados por sección codificada.

A primera vista solo se aprecian los píxeles rojos, pero en la esquina superior izquierda se forma una línea horizontal de diferentes colores, los descritos para cada parte de la esteganografía.

Si ampliamos la imagen a la esquina superior izquierda, se pueden observar las diferentes líneas de color, amarillo para el carácter de control, verde para el tamaño del mensaje y morado para el tamaño de la firma HMAC:

Figura 4.12: Imagen resultante ampliada al inicio de la zona de codificación.

Si continuamos la línea, se pueden ver una larga línea azul, que corresponde al mensaje (recortado ya que es una línea demasiado larga):

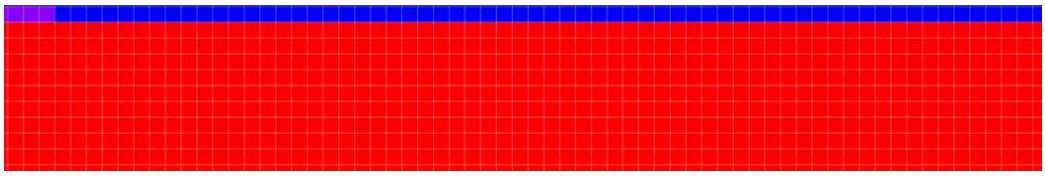


Figura 4.13: Imagen resultante ampliada a píxeles del mensaje (recortada)

Y de la misma forma, la línea naranja para la firma HMAC:

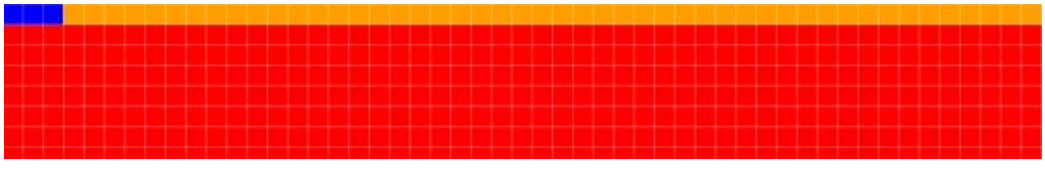


Figura 4.14: Imagen resultante ampliada a píxeles de la firma HMAC (recortada)

5. Plataforma web

Como se ha mencionado al principio, la aplicación en los videojuegos de este sistema de esteganografía tiene 2 objetivos fundamentales: proporcionar un sistema seguro de verificación de logros de jugadores y aprovechar dicho sistema para la generación de tablas clasificatorias en línea.

Con la creación de la plataforma web, se obtiene el medio necesario para finalmente poder cumplir estos 2 objetivos. La plataforma web será el portal de tablas clasificatorias para la hipotética comunidad de jugadores competitivos del videojuego creado y además servirá como herramienta para dar un sentido a todo el proceso de esteganografía desarrollado a lo largo de este proyecto.

En este capítulo detallará el proceso de creación de la web así como el desarrollo del back-end, específicamente, el proceso de decodificación y generación de tablas clasificatorias de la plataforma.

5.1. Creación de la plataforma web

Para la creación del sitio web, se utilizó el servicio gratuito de Amazon Web Services (AWS), donde se generó una instancia Linux gratuita para poder alojar el sitio web al completo.

El sitio web debe de servir como portal para las tablas clasificatorias, es decir, debe ser un hub para la comunidad de jugadores, donde puedan competir entre ellos, subir sus puntuaciones, tener un perfil de usuario... Para ello, se creó la plataforma web con el nombre *StegScored*.

Figura 5.1: Logo creado para el sitio web.

El sitio web está alojado en el dominio `https://stegscored.games/`[1][2].

La plataforma web debe tener ciertas características para que funcione como portal de tablas clasificatorias para el videojuego creado:

- **Perfiles de usuario**: No solo necesarios para dar más sentido de comunidad, sino también para poder utilizar el nombre de usuario durante el proceso de firma HMAC descrito en la sección 4.2.

- **Tablas clasificatorias**: El elemento central, tablas clasificatorias generadas por las imágenes de la pantalla de resultados subidas por los usuarios. Las imágenes deben ser accesibles. Las tablas clasificatorias deben cumplir ciertos aspectos:

  - Filtros por objetivo: Un usuario debe poder filtrar en base al objetivo o logro por el que le interese competir; unos pueden querer ver las puntuaciones más altas, otros las que hayan completado el nivel en menos tiempo, otros las que más monedas hayan obtenido. Incluso múltiples filtros al mismo tiempo.

  - Acceso a usuarios: No únicamente deben existir las tablas clasificatorias, los usuarios deben poder acceder a los perfiles de otros usuarios y ver las puntuaciones subidas por ellos.

[1] Dominio obtenido en: `https://www.name.com/`.
[2] Certificado SSL emitido por Cloudflare: `https://www.cloudflare.com/`

En la página principal se muestran las últimas puntuaciones subidas por los usuarios (en la figura 5.2, la zona derecha, ahora mismo vacía).

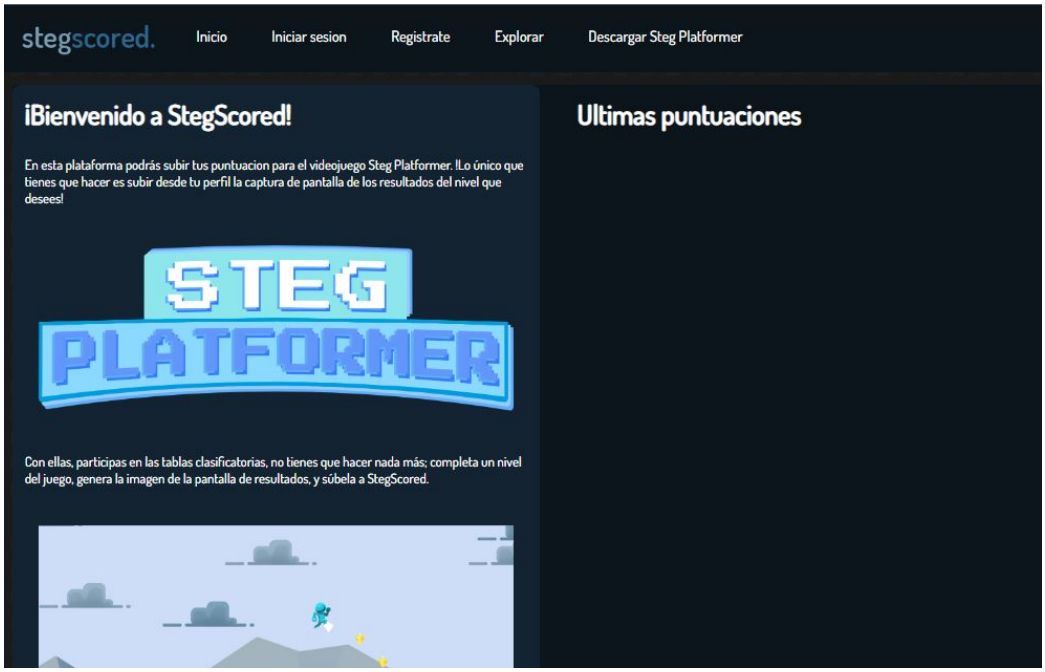


Figura 5.2: Página principal del sitio web.

En el menú de la web se puede acceder a la página *Explorar*, que es en la que se muestra la tabla clasificatoria completa, es el apartado principal de la plataforma.

Figura 5.3: Página principal para las tablas clasificatorias.

Teniendo la plataforma web en marcha, se puede comenzar con el desarrollo del back-end, para finalmente poder subir las imágenes y generar las tablas clasificatorias.

5.2. Back-end de la plataforma web

La plataforma web es la encarga de darle un sentido y poner en práctica el uso de la esteganografía en videojuegos. Para poder generar las tablas clasificatorias, la web debe permitir a los usuarios subir las imágenes y procesarlas para extraer toda la información, es decir, realizar el proceso de decodificación del algoritmo SCC a los píxeles. Todo el back-end del sitio web funciona bajo el lenguaje de programación PHP.

5.2.1. Proceso de decodificación

Para el proceso de decodificación se ha utilizado la librería GD[3] para PHP, una de las librerías para manipulación de imágenes más populares de este lenguaje. Entre otras cosas, ofrece soporte nativo para imágenes WebP.

El proceso de decodificación es sencillo, sigue los mismos pasos que el proceso de codificación:

1. Verificar el carácter de control 'a' en los primeros bits.
2. Obtener el tamaño del mensaje.
3. Obtener el tamaño de la firma HMAC.
4. Obtener el mensaje.
5. Obtener la firma HMAC.
6. Verificar el patrón 1/0 en los píxeles restantes.

[3]Procesamiento de imágenes con GD: `https://www.php.net/manual/es/book.image.php`

El proceso comienza cuando un usuario quiere subir una puntuación en la plataforma:

Figura 5.4: Página para subir la imagen.

El usuario selecciona la imagen con la puntuación que desee subir a su perfil.

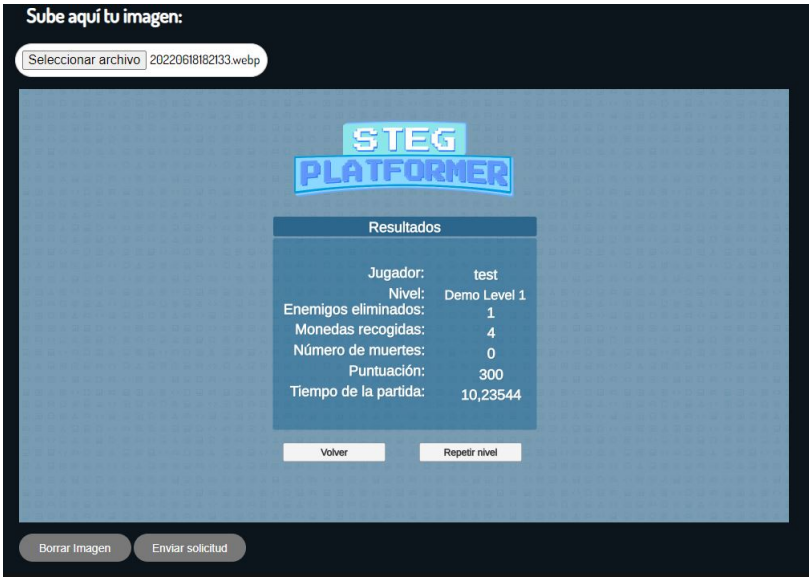


Figura 5.5: Imagen cargada en la plataforma.

Una vez cargada la imagen, el servidor comienza el proceso de decodificación.

Primero se obtiene el alto y el ancho de la imagen para poder iterar sobre todos los píxeles:

```
1 $width = imagesx($tempUploadedImage);
2 $height = imagesy($tempUploadedImage);
```

5.2.1.1. Verificar el carácter de control

PHP ofrece diferentes funciones para poder procesar los píxeles y sus colores; *imagecolorat()* devuelve el índice del píxel que se indique por parámetro mediante los valores **x** e **y**. Con ese índice se puede obtener un array con los valores numéricos de cada canal de color con la función *imagecolorsforindex()*. Con estas funciones se realizará todo el proceso de obtención de valores de cada píxel.

Comenzando con la decodificación del carácter 'a':

```
1 while($indexBit < 32) {
2     $rgb = imagecolorat($tempUploadedImage, $indexPixelX, 0);
3     $colors = imagecolorsforindex($tempUploadedImage, $rgb);
4     ...
5 }
```

Una vez obtenido el array de colores, con la función *decbin()* se puede pasar un valor decimal a binario, y obtener el LSB tratándolo como una cadena de caracteres obteniendo su último carácter, que puede ser un 1 o un 0. Cada LSB obtenido se concatena a una cadena llamada **codigoLetraBin**.

```
1  ...
2  //Obtenemos el LSB del rojo
3  $r = decbin($colors['red']);
4  $rlsb = $r[strlen($r)-1];
5  $indexBit++;
6  $codigoLetraBin = $codigoLetraBin.$rlsb;
7
8  //Obtenemos el LSB del verde
9  $g = decbin($colors['green']);
10 $glsb = $g[strlen($g)-1];
```

```
11 $indexBit++;
12 $codigoLetraBin = $codigoLetraBin.$glsb;
13
14 if($indexBit == 32) break;
15 //Obtenemos el LSB del azul
16 $b = decbin($colors['blue']);
17 $blsb = $b[strlen($b)-1];
18 $indexBit++;
19 $indexPixelX++;
20 $codigoLetraBin = $codigoLetraBin.$blsb;
21 }
```

Una vez finalizado el proceso, se transforma *codigoletrabien* a un valor decimal mediante la función *bindec*, que representa el valor ASCII del carácter encontrado. Con la función *chr* se transforma un valor decimal a su carácter ASCII.

```
1 $codigoLetra = chr(bindec($codigoLetraBin));
2 if ($codigoLetra != 'a'){
3     echo "NOVALID";
4     return;
5 }
```

Si el carácter obtenido no es la 'a', entonces el proceso de decodificación termina y la imagen no es válida.

5.2.1.2. Obtener el tamaño del mensaje y de la firma HMAC

El proceso para obtener el tamaño del mensaje y de la forma HMAC es virtualmente idéntico al del carácter de control, se obtienen los 32 siguientes LSB y se transforma la cadena de unos y ceros obtenida a un valor decimal mediante *decbin()*, esta vez sin utilizar *chr()*.

```
1 while($indexBit < 32) {
2     $rgb = imagecolorat($tempUploadedImage, $indexPixelX, 0);
3     $colors = imagecolorsforindex($tempUploadedImage, $rgb);
4
5     //Obtenemos el LSB del rojo
6     $r = decbin($colors['red']);
7     ...
```

```
8       //Obtenemos el LSB del verde
9       $g = decbin($colors['green']);
10      ...
11      if($indexBit == 32) break;
12      //Obtenemos el LSB del azul
13      $b = decbin($colors['blue']);
14      ...
15  }
16  $indexPixelX++;
17  $indexBit = 0;
18  $tamanoMensaje = bindec($tamanoMensajeBin);
```

El proceso es el mismo para el tamaño de la firma HMAC.

5.2.1.3. Obtener mensaje y firma HMAC

Igualmente que en el proceso anterior se sigue el mismo algoritmo, pero se hace en tantos LSB como el tamaño del mensaje que se ha obtenido (para el mensaje) y en tantos LSB como el tamaño de la firma HMAC que se ha obtenido (para la firma). Además, por cada iteración solo se obtienen 8 LSB, ya que cada carácter son 8 bits (1 byte). Por cada iteración (es decir, por cada carácter obtenido), se concatena cada carácter obtenido a una cadena vacía para que se vaya formando el mensaje completo.

```
1   for ($i=0; $i < $tamanoMensaje; $i++) {
2       $caracter = '';
3       $caracterBits = '';
4       $indexBit = 0;
5       while($indexBit < 8) { //8 bit por caracter ASCII
6
7           $rgb = imagecolorat($tempUploadedImage, $indexPixelX,
        0);
8           $colors = imagecolorsforindex($tempUploadedImage,
       $rgb);
9
10          //Obtenemos el LSB del rojo
11          $r = decbin($colors['red']);
12          $rlsb = $r[strlen($r)-1];
13          ...
14          $caracterBits = $caracterBits.$rlsb;
```

```
15
16          //Obtenemos el LSB del verde
17          $g = decbin($colors['green']);
18          $glsb = $g[strlen($g)-1];
19          ...
20          $caracterBits = $caracterBits.$glsb;
21
22          if($indexBit == 8) break;
23          //Obtenemos el LSB del azul
24          $b = decbin($colors['blue']);
25          $blsb = $b[strlen($b)-1];
26          ...
27          $caracterBits = $caracterBits.$blsb;
28      }
29      $indexPixelX++;
30      $caracterAsciiInt = bindec($caracterBits);
31      $caracter = chr($caracterAsciiInt);
32      $mensajeDecoded = $mensajeDecoded.$caracter;
33  }
```

El mensaje obtenido de la partida es el *mensajeDecoded*, los datos de la partida son una cadena en formato JSON (lo que ayudará más adelante para procesar la puntuación y añadirla a las bases de datos).

Se sigue el mismo proceso para la obtención de la firma HMAC.

5.2.1.4. Verificación del patrón 1/0

Finalmente, para terminar el proceso de decodificación, se verifica que los píxeles no codificados no hayan sido alterados, para ello se verifica el patrón 1/0 en todos los píxeles restantes. Si los LSB que se vayan obteniendo en algún momento rompen el patrón de unos y ceros, la imagen ha sido alterada.

Se comienza con el bit del patrón en 1, en la variable *patternBit*.

```
1  $patternBit = 1;
2      for ($y=0; $y < $height; $y++) {
3          if(!$isCorrectPatern) break;
4          ...
```

Y a continuación se itera sobre todos los píxeles restantes obteniendo sus LSB; por cada LSB obtenido se compara al *patternBit*, si el LSB obtenido es igual entonces es correcto, se actualiza el patternBit y se siguen verificando los LSB. Si en algún momento del proceso algún LSB es diferente al *patternBit*, entonces la imagen ha sido alterada y no es válida.

```
...
while ($indexPixelX < $width) {
    $rgb = imagecolorat($tempUploadedImage, $indexPixelX, $y)
    ;
    $colors = imagecolorsforindex($tempUploadedImage, $rgb);
    $bitObtenido = 0;

    //LSB Rojo
    $r = decbin($colors['red']);
    $bitObtenido = $r & 1;
    if($bitObtenido != $patternBit){
        $isCorrectPatern = false;
        break;
    }
    else{
        $patternBit = ($patternBit == 1) ? 0 : 1;
    }

    //Mismo proceso para el canal verde
    ...

    //Mismo proceso para el canal azul.
    ...

    $indexPixelX++;
}
...
```

Para probar esto, se ha hecho la prueba alterando la imagen original que se pretendía subir a la plataforma. Se han modificado algunos de los píxeles que se sabe que no están modificados por la esteganografía añadiendo diferentes círculos rojos.

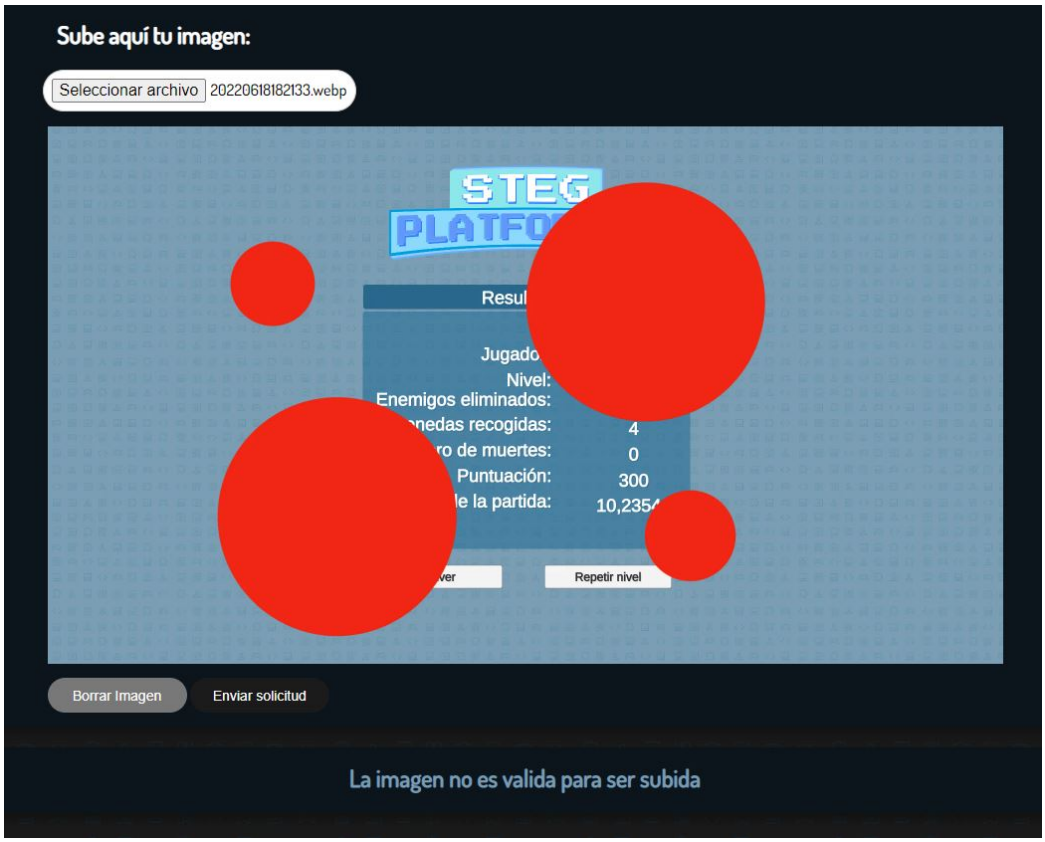


Figura 5.6: Intento de subir la imagen original modificada.

Se han modificado algunos de los píxeles que se sabe que no están modificados, y la plataforma web no permite subir la imagen al no detectarla como válida, ya que el patrón 1/0 de los píxeles se rompe.

5.2.2. Verificación de la firma HMAC

Si la imagen pasa todos los procesos de seguridad visuales (carácter de control y patrón 1/0), entonces el servidor se encarga de generar la firma HMAC y compararla con la obtenida en el proceso de decodificación. La firma HMAC estaba formada por la concatenación de datos del servidor y datos de la partida; el nombre de usuario y la cadena JSON con los datos de la partida, respectivamente.

El proceso de creación del hash es prácticamente idéntico al realizado en el juego, solo que en un lenguaje de programación diferente. Si la firma HMAC es diferente, entonces la plataforma web no acepta la imagen como válida.

```
1 //Verificacion de la firma
2 $imagenValida = false;
3 $checksum = md5($mensajeDecoded);
4 $username = $_SESSION['user'];
5 $nonHashedSignature = $checksum.':'.$username;
6 $hashKey = "-"; //Clave compartida entre el videojuego y el
     servidor
7 $signature = hash_hmac('sha256', $nonHashedSignature,
     $hashKey);
8
9 $imagenValida = ($signature == $firmaDecoded) ? true : false;
```

Si la firma HMAC es igual a la obtenida, entonces la imagen es válida y procesa el JSON para mostrar una vista previa.

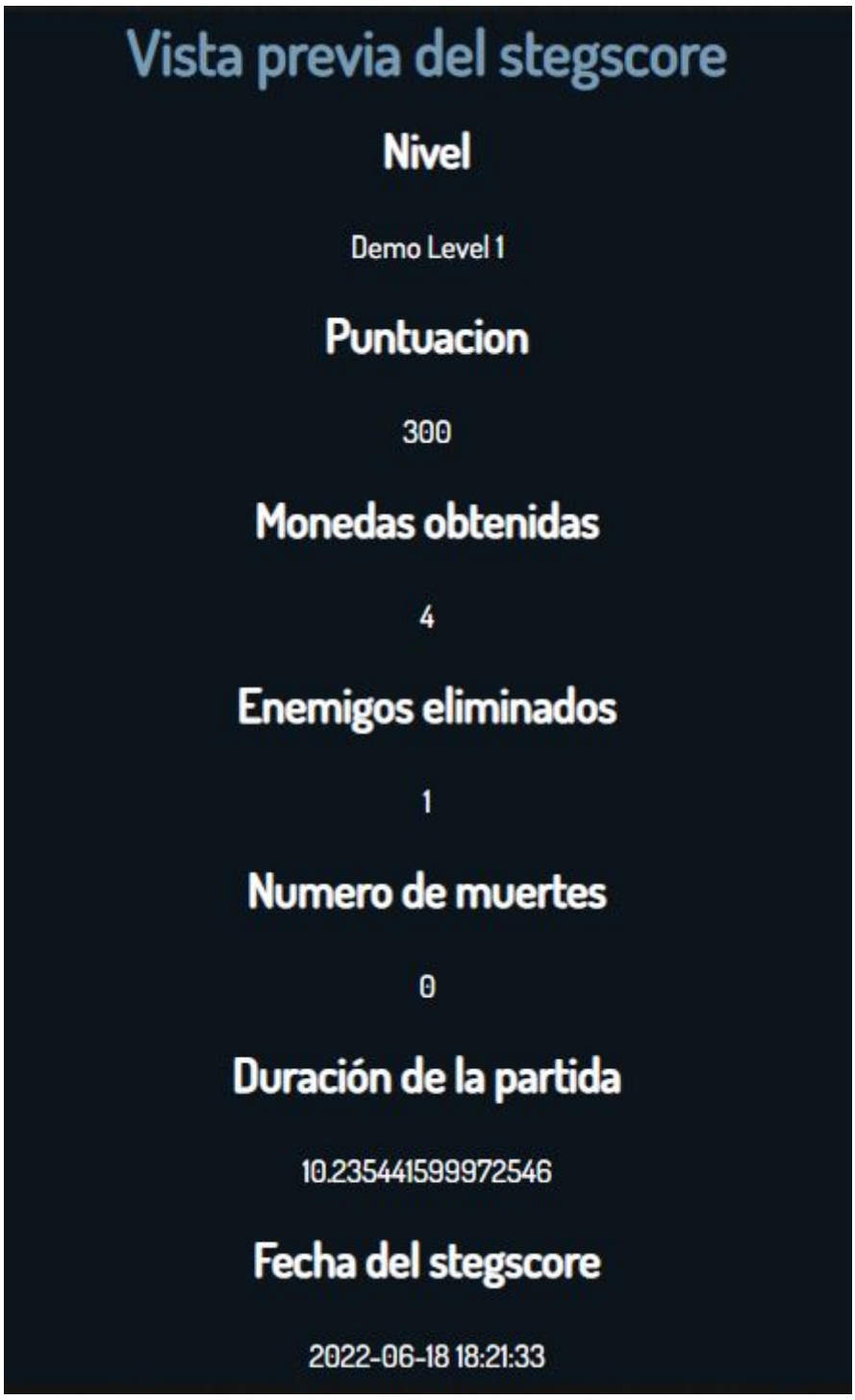


Figura 5.7: Vista previa de los datos de la partida.

5.3. Pruebas con usuarios

Con el videojuego terminado y la plataforma web en funcionamiento, se comenzaron las pruebas en usuarios, que permiten probar todo el proyecto en un entorno real y no controlado.

Las pruebas son simples, invitar a diferentes personas a crear una cuenta y jugar al videojuego y que suban sus propias puntuaciones a la plataforma web.

5.3.1. Tablas clasificatorias

Desde la página *Explorar* del sitio web se llega a las tablas clasificatorias con todas las puntuaciones subidas por los usuarios. En total hay 15 puntuaciones subidas:

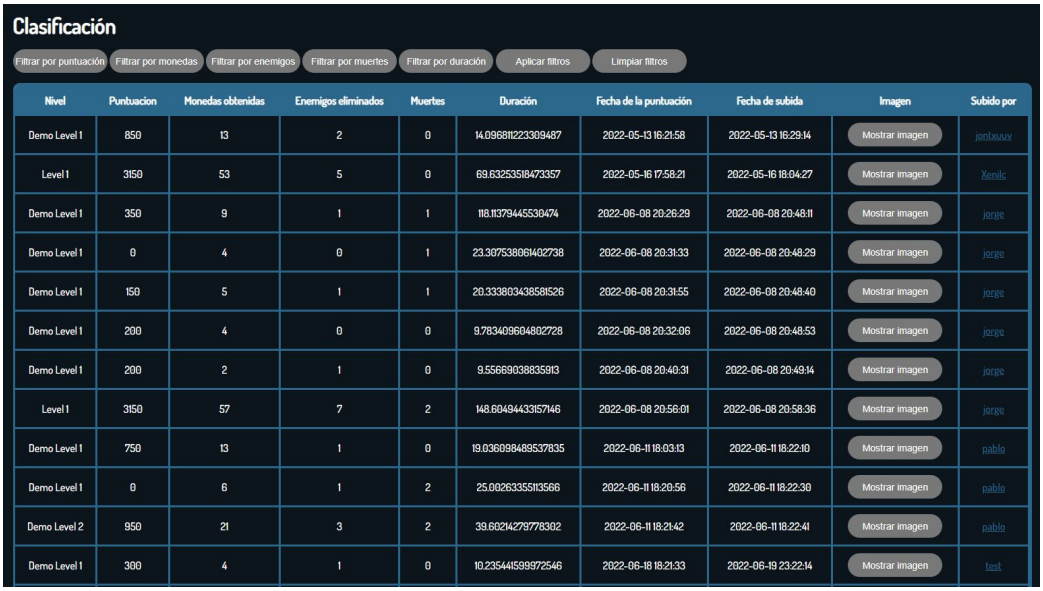


Figura 5.8: Página de clasificación general

Desde aquí hay diferentes parámetros entre los que filtrar la tabla clasificatoria, todo depende de lo que busque la comunidad de jugadores.

Para el nivel principal *Level 1* hay 3 puntuaciones subidas. Si inicialmente filtramos por puntuación, se puede ver que 2 de ellas empatan en puntuación y solo 1 tiene una puntuación superior.

Figura 5.9: Clasificación por puntuación del *Level 1*.

Si los jugadores compitieran por puntuación, la subida por el usuario **abar** sería la más alta, y este jugador sería el vencedor, pero si los jugadores compitieran por el tiempo más corto en completar el nivel, la puntuación del jugador **abar** sería la más baja, y el jugador **Xenilc** sería el vencedor, ya que aunque empate en puntos con el jugador **jorge**, su duración de partida es significativamente inferior.

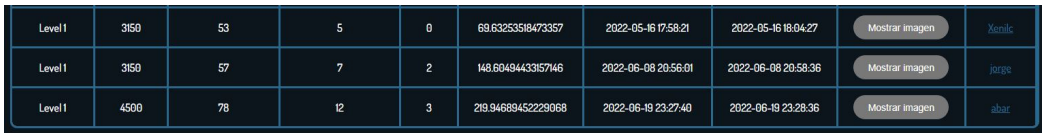


Figura 5.10: Clasificación por tiempo del *Level 1*.

Esta misma puntuación ganaría si se filtrara por vidas perdidas, por ejemplo. Para poder verificar cualquiera de estas puntuaciones, se puede acceder a la propia imagen de la que se han obtenido todos estos datos en cada una de las filas de la tabla clasificatoria.

La comunidad de jugadores puede competir y desarrollarse mediante esta plataforma utilizando únicamente las imágenes generadas por el juego.

5.3.2. Perfiles de usuario

Además de las tablas clasificatorias, desde esta plataforma cada usuario tiene su propio perfil, que puede ser accedido y desde el cual se tiene acceso tanto a las puntuaciones como a las imágenes subidas por el propio usuario.

De esta forma, cualquier usuario puede compararse y competir con otro usuario en concreto, lo que mejora la sensación de ámbito de videojuego competitivo y la comunidad.

Figura 5.11: Perfil de un usuario.

Figura 5.12: Puntuaciones del usuario.

6. Conclusión

6.1. Diferencia entre estimación y tiempo real

Tras el seguimiento y control de las horas y tiempos de trabajo realizados periódicamente durante el desarrollo del proyecto, se ha realizado un gráfico con la comparativa de horas reales a las planificadas para cada paquete de trabajo. (Figura 6.1).

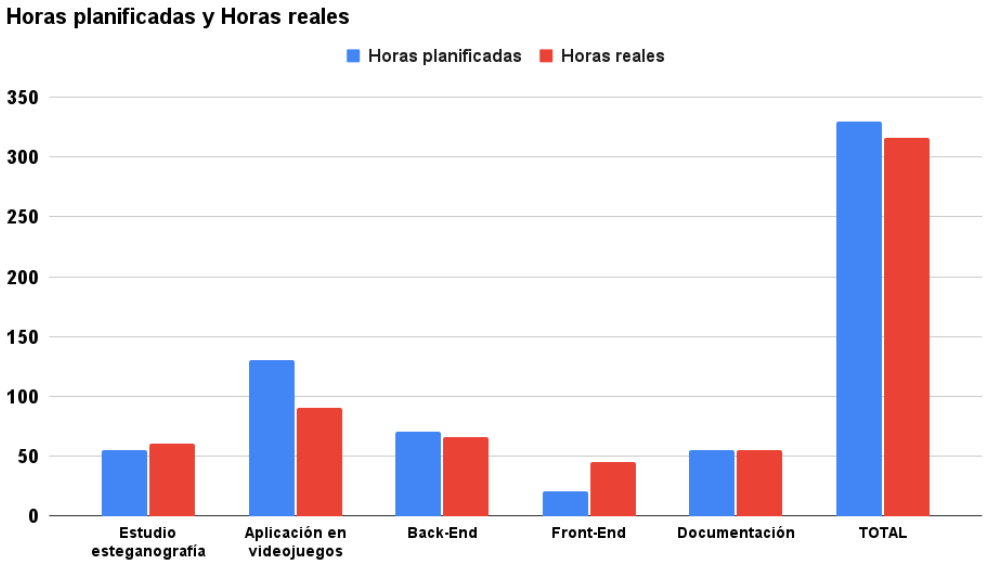


Figura 6.1: Tiempo planificado y tiempo real.

De forma resumida, el proyecto se ha desarrollado en una cantidad de horas similar a lo planificado originalmente. Concretamente, se ha desarrollado completamente en 316 de horas, un 4.3 % menos de las 330 horas planificadas.

La mayor diferencia se ha dado en el paquete de trabajo «Aplicación en videojuegos» (AV), ya que originalmente se planificaron 130 horas, pero se ha desarrollado en 90. Esto se ha debido a que algunas de las tareas originalmente planeadas para este paquete de trabajo se han podido realizar simultáneamente, como la tarea de desarrollo del sistema de capturas de pantalla, y el desarrollo del sistema de esteganografía desarrollado en el paquete «Estudio de la esteganografía» (ES). Además, la planificación original estaba pensada para una cantidad de trabajo diaria de 3 horas, pero para ciertos paquetes de trabajo, como es el caso de AV, se han trabajado más

horas diarias, en concreto 5. Esto ha permitido terminar mucho antes de lo planificado este paquete de trabajo.

El desarrollo del sitio web, que engloba el back-end y front-end, aunque ha sido similar en cuanto a horas planificadas y reales, ha seguido una distribución temporal muy diferente a la planeada. Aunque el front-end sí se ha realizado paralelamente al back-end, en cierto punto se ha parado el desarrollo del back-end para terminar de pulir el front-end y tras ello continuar y finalizar el back-end.

Para el desarrollo de la documentación, aunque se ha terminado en fecha, se ha comenzado más tarde de lo planificado, lo que ha incurrido en más horas diarias.

Se ha recreado el diagrama Gantt considerando las horas reales invertidas (Figura 6.2). Los plazos se han cumplido, ya que se ha finalizado el proyecto en la fecha estipulada.

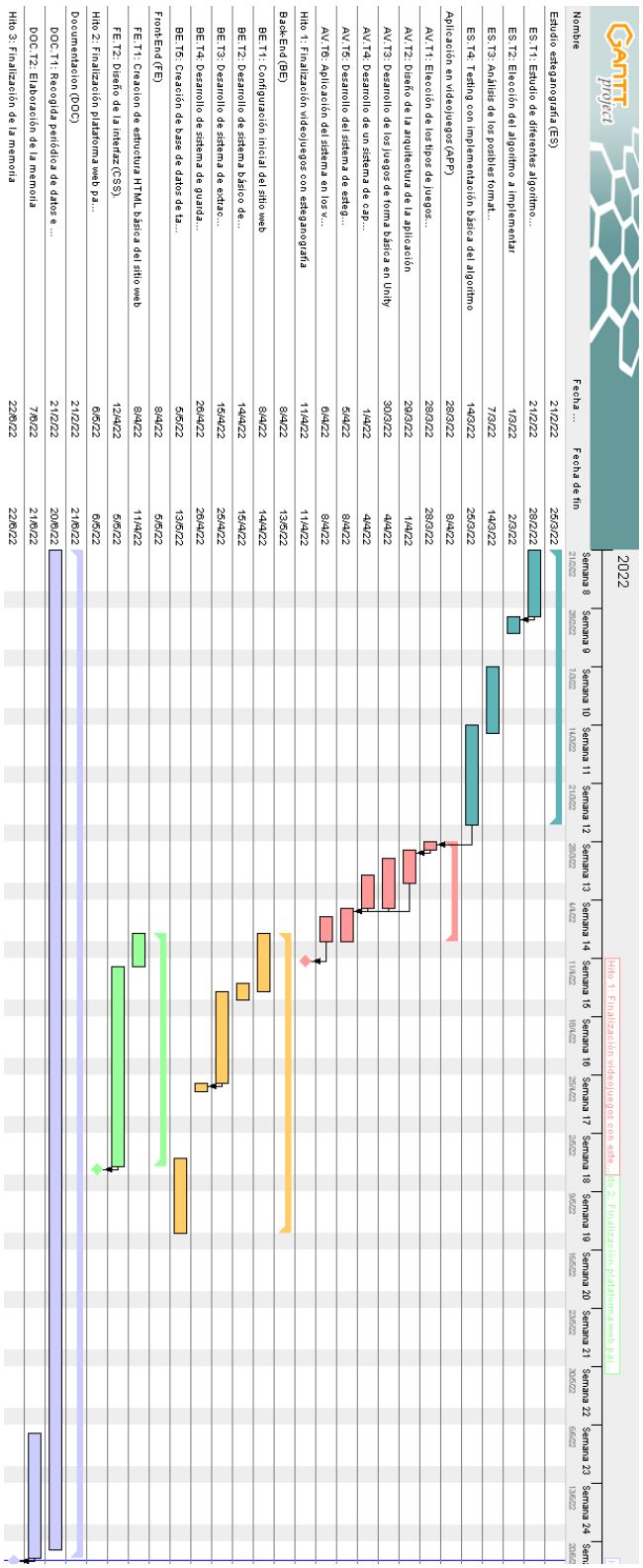


Figura 6.2: Diagrama Gantt del desglose de horas real del proyecto.

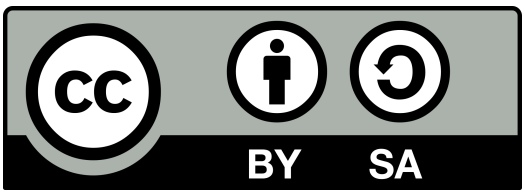
6.2. Futuras líneas de trabajo

Este proyecto tiene muchas vías de trabajo futuro. Idealmente, este sistema de esteganografía y tablas clasificatorias puede desarrollarse para diferentes videojuegos, con diferentes metas/objetivos y datos a codificar; videojuegos de carreras, videojuegos de acción por niveles... Todos ellos pueden ser soportados en la propia plataforma web desarrollada, con sus respectivas secciones.

Se puede expandir y mejorar el sistema de esteganografia con métodos más seguros que se han estudiado en 3.1; como Max-Bit o Pixel Indicator, que requerirían más trabajo pero ofrecerían mayor seguridad. Estos diferentes algoritmos pueden ser mezclados con el sistema de firma HMAC para ofrecer aún más seguridad.

Alternativamente, se puede desarrollar este sistema de esteganografía como un sistema comercial (una Application Programming Interface (API), por ejemplo) para diferentes estudios o compañías de desarrollo de videojuegos, para que pueda ser aplicado en sus propios videojuegos y puedan construir sus tablas clasificatorias con el uso de esta herramienta.

6.3. Licencia



[1]Creative Commons: Attribution-ShareAlike 4.0 International (`www.creativecommons.org/licenses/by-sa/4.0/`)

partir del material para cualquier propósito, incluso comercialmente.

Bajo las condiciones siguientes: (1) Atribución — debe reconocer adecuadamente la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios; (2) compartir igual — si remezcla, transforma o crea a partir del material, deberá difundir sus contribuciones bajo la misma licencia que el original.

6.4. Reflexión personal

Concluyo esta memoria con una reflexión personal y subjetiva sobre este proyecto.

Este proyecto ha sido uno de los más grandes que he realizado, principalmente por ser un proyecto sobre una idea propia y que he desarrollado individualmente. Desde hace tiempo tenía la idea de realizar este proyecto como mi TFG, y fue un gran alivio poder contar con la ayuda de mi tutor Juanan Pereira, que ha contribuido mucho a mejorar el proyecto con su conocimiento sobre la esteganografía.

Desde el inicio del proyecto tenía claro que podía realizarlo íntegramente, a pesar de saber que la cantidad de trabajo que requería era muy grande, ya que se hace uso de muchas tecnologías, sistemas, técnicas y lenguajes de programación diferentes.

Relacionado con esto, poder haber desarrollado este TFG en (mayoritariamente) Unity ha sido de gran ayuda, ya que ya tenía experiencia en el desarrollo de videojuegos con este motor gráfico, por lo que me ha motivado mucho a desarrollar. En gran parte el desarrollo del proyecto lo he encontrado tan interesante hasta el punto de considerarlo como un hobby durante la duración del proyecto.

Como he mencionado, he utilizado diferentes tecnologías conjuntamente, en concreto las relacionadas con el desarrollo web, que han hecho que aprenda y mejore considerablemente mi habilidad en este ámbito, ya que aunque previamente había desarrollado aplicaciones web, ninguna ha sido del tamaño y complejidad de la realizada para este proyecto.

Ha sido una experiencia de mucho aprendizaje y muy fructífera. De algún modo, he querido que fuera de mayor duración, me hubiera gustado implementar más funcionalidades y hacer el proyecto aún más grande y ambicioso.

Acrónimos

API

Application Programming Interface. 103

AWS

Amazon Web Services. 23, 83

EDT

Estructura de descomposición del trabajo. 10, 18

GPA

Generalised Patchwork Algorithm. 39

HMAC

Hash-based message authentication code. 72

IDE

Integrated Development Environment. 26

ISO

International Organization for Standardization. 27

JSON

JavaScript Object Notation. 52, 53, 73

LSB

Least Significant Bit. 33

LSI

Lenguajes y Sistemas Informáticos. 2

SGBD

Sistema de gestión de bases de datos. 26

SSH

Secure SHell. 26

SSL

Secure Socket Layer. 23

TFG

Trabajo de Fin de Grado. 2, 17, 18, 26, 30

Glosario

back-end

Capa lógica de acceso a los datos en un *software*, generalmente se refiere al servidor de la aplicación. 10, 14, 18, 83, 86, 101

demo

Programa informático de demostración que es una versión reducida en prestaciones de un programa para poder utilizarlo y evaluarlo. 8, 65

EC2

Amazon Elastic Compute Cloud (EC2) es un servicio que proporciona capacidad informática en la nube segura y de tamaño modificable. 23, 25, 26

front-end

Capa de presentación de una aplicación *software*. 10, 15, 16, 18, 101

git

Software de control de versiones. 12, 14, 16

hardware

Componentes físicos de un sistema informático, como la unidad central de procesamiento (CPU) y los dispositvos periféricos. 29

hash

Un resumen criptográfico, una cadena generada por la función hash.. 73

hub

En el contexto de los videojuegos, lugar donde se concentra la comunidad de jugadores de un videojuego específico.. 83

payload

En esteganografía, cantidad de bits disponibles en la imagen para insertar el mensaje.. 34–36, 41

píxel

Unidad básica de representación en una imagen digital. 33, 36, 38, 49

software

«Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora». 8, 9, 11, 12, 14, 16, 21, 26, 109

Bibliografía

[1] Jyrki Alakuijala y Vincent Rabaud. «Lossless and transparency encoding in webp». En: *URL https://developers.google.com/speed/webp/docs/webp_-lossless_alpha_study* (2017).

[2] Arshiya Sajid Ansari, Mohammad Sajid Mohammadi y Mohammad Tanvir Parvez. «A comparative study of recent steganography techniques for multiple image formats». En: *International Journal of Computer Network and Information Security* 11.1 (2019). Publisher: Modern Education and Computer Science Press, págs. 11-25.

[3] Nabarun Bagchi. «Secure BMP image steganography using dual security model (IDEA, image intensity and bit randomization) and max-bit algorithm». En: *International Journal of Computer Applications* 1.21 (2010). Publisher: Citeseer, págs. 18-22.

[4] CHKrishna Chaitanya y col. «Enhanced Hash Based Image Steganography Technique to Increase Data Integrity and Confidentiality». En: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. IEEE, 2021, págs. 1456-1461.

[5] E. Eltyeb y A. Elgabar. «Comparison of LSB Steganography in BMP and JPEG Images». En: *Int. J. Soft Comput. Eng* 3.5 (2013). Publisher: Citeseer, págs. 91-95.

[6] Chance Gibbs y Narasimha Shashidhar. «StegoRogue: steganography in two-dimensional video game maps». En: *Advances in Computer Science: an International Journal* 4.3 (2015), págs. 141-146.

[7] Richard Gono. «Steganography in computer graphics». En: *Faculty of Informatics. Masaryk University* (2017).

[8] Adnan Abdul-Aziz Gutub. «Pixel indicator technique for RGB image steganography». En: *Journal of emerging technologies in web intelligence* 2.1 (2010), págs. 56-64.

[9] Ahmed Hambouz y col. «Achieving data integrity and confidentiality using image steganography and hashing techniques». En: *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*. IEEE, 2019, págs. 1-6.

[10] Aye Thu Hlaing y Mya Thidar Myo Win. «Secure Image Steganography using Canny Magic LSB Substitution Method and HMAC Algorithm». En: ().

[11] Neil F. Johnson y Sushil Jajodia. «Exploring steganography: Seeing the unseen». En: *Computer* 31.2 (1998). Publisher: IEEE, págs. 26-34.

[12] Jeff Kabachinski. «TIFF, GIF, and PNG: get the picture?» En: *Biomedical instrumentation & technology* 41.4 (2007), págs. 297-300.

[13] David Kahn. «The history of steganography». En: *International workshop on information hiding*. Springer, 1996, págs. 1-5.

[14] Hugo Krawczyk, Mihir Bellare y Ran Canetti. *RFC2104: HMAC: Keyed-hashing for message authentication*. 1997.

[15] Ashish Kumari, Shyama Sharma y Navdeep Bohra. *Implementation of IMAGE STEGANOGRAPHY Based on Random LSB*. 2011. URL: https://www.semanticscholar.org/paper/Implementation-of-IMAGE-STEGANOGRAPHY-Based-on-LSB-%20Kumari-Sharma/ef8061d9c7310fe30e60dc2e4d5866747bbb4d9e.

[16] Lisa M. Marvel, Charles G. Boncelet y Charles T. Retter. «Spread spectrum image steganography». En: *IEEE Transactions on image processing* 8.8 (1999). Publisher: IEEE, págs. 1075-1083.

[17] Fabian Mentzer, Luc Van Gool y Michael Tschannen. «Learning better lossless compression using lossy compression». En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, págs. 6638-6647.

[18] Tayana Morkel, Jan HP Eloff y Martin S. Olivier. «An overview of image steganography.» En: *ISSA*. Vol. 1. Issue: 2. 2005, págs. 1-11.

[19] J. S. Neenu y Elizabeth B. Varghese. «A novel approach for SCC algorithm using Pattern based Image Steganography». En: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 3. IEEE, 2016, págs. 1-6.

[20] Lip Yee Por y col. «An enhanced mechanism for image steganography using sequential colour cycle algorithm.» En: *Int. Arab J. Inf. Technol.* 10.1 (2013), págs. 51-60.

[21] Yash Sharma. «A Review on Game based Steganography». En: *2019 International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE, 2019, págs. 286-290.

[22] Ali Sheidaee y Leili Farzinvash. «A novel image steganography method based on DCT and LSB». En: *2017 9th International conference on Information and Knowledge Technology (IKT)*. IEEE, 2017, págs. 116-123.

[23] Namita Tiwari y Dr Madhu Shandilya. «Evaluation of various LSB based methods of image steganography on GIF file format». En: *International Journal of Computer Applications* 6.2 (2010). Publisher: Citeseer, págs. 1-4.

[24] James M. Turner. «The keyed-hash message authentication code (hmac)». En: *Federal Information Processing Standards Publication* 198.1 (2008). Publisher: US Department of Commerce, National Institute of Standards and Technology ...

[25] Stuart Wilson. «Unreal Steganography». En: *Forensic Focus* (2019).