Automatización de tareas de QA en localización de videojuegos
Automation of Localization QA tasks in videogames

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Dewei Chen
Nota del TFG: 7

Director
Guillermo Jiménez Díaz

Colaborador
Departamento de Ingeniería del Software e Inteligencia Artificial

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Automatización de tareas de QA en localización de videojuegos
Automation of Localization QA tasks in videogames

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Autor
Dewei Chen
Nota del TFG: 7

Director
Guillermo Jiménez Díaz

Colaborador
Departamento de Ingeniería del Software e Inteligencia Artificial

Convocatoria: *Mayo 2025*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

16 de junio de 2025

Dedicatoria

A Guille por darme la oportunidad de elaborar este trabajo y la ayuda proporcionada.

Agradecimientos

A Guillermo, por el tiempo empleado en dirigirme y la ayuda en la elaboración de este trabajo y por animarme cuando estoy deprimido y asustado por este trabajo.

Resumen

Automatización de tareas de QA en localización de videojuegos

La localización es una tarea imprescindible de los videojuegos hoy en día, ya que los videojuegos se quieren vender y publicar en distintos países y culturas. Para ello, es necesario adaptar los videojuegos a diferentes lenguajes y a diferentes culturas. Ese proceso lo llamamos localización. El trabajo del equipo de localización puede llevar varias iteraciones para evitar errores que puedan aparecer.

Hoy en día no existen programas que verifiquen si un videojuego tiene errores de localización, ya sean de traducción o de internacionalización, por lo que es necesario personal que tenga que hacer esa tarea de verificación, esto supone un alto coste en tiempo y dinero.

El objetivo de este trabajo es tratar de automatizar esas tareas empezando por reconocer y recoger el texto que aparece en el videojuego, seguido de unos tests que verifican si el texto reconocido tiene algún error de localización, generando así un informe que indique los posibles fallos de la localización.

Palabras clave

Videojuegos, Localización, Internacionalización, Automatización, QA, LQA, OCR, Tesseract.

Abstract

Automation of Localization QA tasks in videogames

Localization is an essential task in modern video games, as these products are intended to be sold and published in various countries and cultures. To achieve this, video games must be adapted to different languages and cultural contexts. This process is known as localization. The work of the localization team often involves several iterations to avoid potential errors.

Currently, there are no tools that automatically verify whether a video game contains localization errors, whether related to translation or internationalization. As a result, this verification must be performed manually by specialized personnel, which involves high costs in terms of time and money.

The goal of this project is to automate these tasks, starting with the recognition and extraction of the text that appears in the video game. This is followed by a set of tests that check whether the recognized text contains any localization issues, ultimately generating a report that highlights potential localization errors.

Keywords

Video games, Localization, Internationalization, Automation, QA, LQA, OCR, Tesseract.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hoy en día debido a la gran cantidad de usuarios en los videojuegos en diferentes regiones, surge la necesidad de adaptar el videojuego a otras regiones o países diferentes al origen, para ello existe una serie de pasos o estrategias que se puede seguir, para ello se diseñará una herramienta que ayude a este proceso.

1.1. Motivación

Con el paso de los años, los videojuegos se han convertido en una de las industrias de entretenimiento más grandes del mundo. Según un informe de Newzoo (2024), en 2024, el mercado global de los videojuegos ha alcanzado un valor estimado de 187.7 miles de millones de ingreso como se muestra en la figura 1.1. En el mismo informe aparece el crecimiento jugadores (Figura 1.2), desde 3.14 billones en 2022 a 3.422 billones 2024 y su predicción en 2027 es de 3.759 billones.

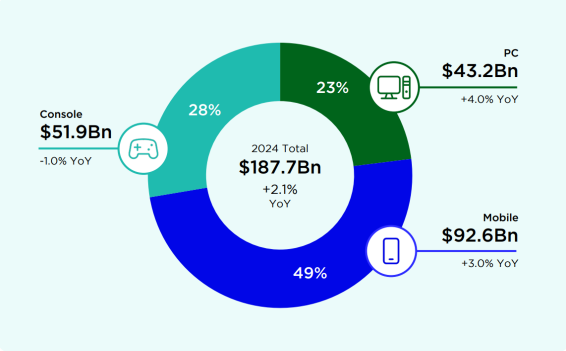

Figura 1.1: Ingreso estimado en la industria de videojuegos 2024.Newzoo

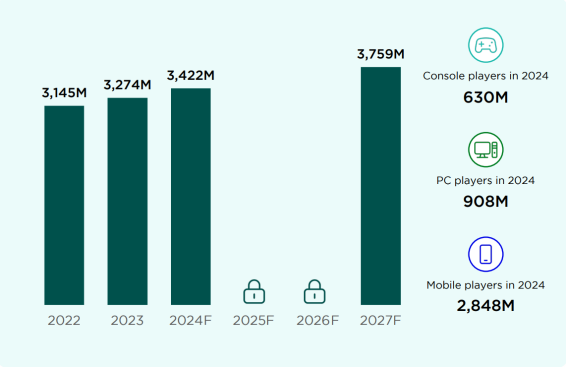

Figura 1.2: Jugadores globales 2022-2024 y predicción en 2027.Newzoo

Además en el mismo informe se comenta el porcentaje de jugadores de cada región ocupando más de la mitad, un 53 % los jugadores pacifico asiáticos como se muestra en la figura 1.3. Con esos crecimientos y estos datos viene la necesidad de adaptar los videojuegos a diferentes lenguajes y culturas a través de los procesos de internacionalización (*I18N*) y localización (*L10N*) para poder ser publicados en otras regiones y países.

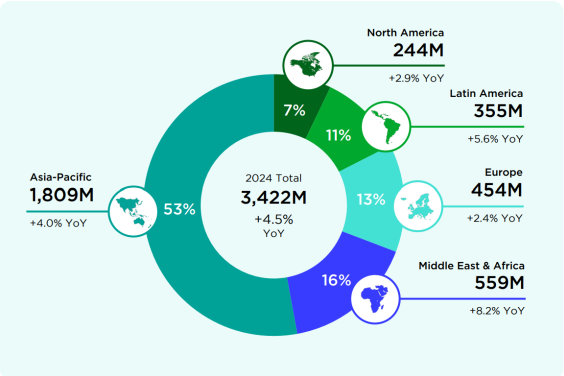

Figura 1.3: Porcentaje de jugadores de cada región. Newzoo

La internacionalización es el proceso por el cual un videojuego se prepara desde sus etapas iniciales de desarrollo para soportar diferentes culturas e idiomas, de tal manera que no haya que realizar grandes modificaciones en el código para introducirlos. Por otra parte, la localización es el proceso por el cual se traducen y adaptan los textos, gráficos y recursos de un videojuego para las diferentes culturas en las que éste se va a comercializar.

Durante ambos de estos procesos se pueden producir distintos tipos de errores (como errores de traducción o errores de visualización). Aquí es donde entra el *Localization Quality Assurance* (o LQA). Este proceso se encarga de revisar y probar las distintas partes del videojuego para asegurarse de que no se produzca ninguno de estos errores y el producto final sea adecuado para las culturas y el público objetivo.

Hasta ahora este proceso ha sido un trabajo manual en el que los probadores comprueban meticulosamente cada texto y como éstos se visualizan dentro del videojuego, lo cual requiere de una gran cantidad de tiempo y recursos. Aunque existen

herramientas para comprobar la corrección de las traducciones y los textos automáticamente, sin embargo, no se puede decir lo mismo en lo referente a comprobar que los textos se visualicen correctamente en el contexto del videojuego.

Teniendo en cuenta lo expuesto en los párrafos anteriores anterior, nuestra motivación para este trabajo es poder automatizar las pruebas sobre los procesos de internacionalización y localización, de forma que se puedan ahorrar tiempo y recursos en estos campos y dedicarlos a otros más importantes. Para conseguir esto, vamos a utilizar técnicas de visión por computador y Reconocimiento óptico de caracteres *(OCR,Optical Character Recognition)* de forma que a partir de una captura de pantalla del videojuego, se pueda extraer el texto y realizar distintas comprobaciones y pruebas sobre él.

1.2. Objetivos

El objetivo principal en este trabajo es el diseño y la implementación de una herramienta que ayude en la automatización del QA en la localización. Para lograr el objetivo principal, se plantea los siguientes objetivos secundarios:

1. Investigar sobre el proceso y las herramientas utilizadas en LQA, además de los errores más comunes en internacionalización y localización.

2. Investigar sobre OCRs y cómo utilizarlas, para elegir una adecuada a nuestras especificaciones, realizando pruebas y comparaciones entre ellas.

3. Diseñar una serie de tests automáticos basado en OCR para los posibles errores lingüísticos.

4. Despliegue de la herramienta en una imagen de Docker para que pueda desplegarse en cualquier equipo.

5. Integrar el OCR para usarlo como entrada de datos a los tests. Hacer mejoras para que el OCR de el máximo acierto posible.

6. Evaluar nuestra herramienta para asegurar de que produce un porcentaje de acierto adecuado.

1.3. Plan de trabajo

Para conseguir nuestros objetivos, seguiremos los siguientes pasos:

1. Investigar y entender como funciona el proceso de internacionalización, localización, el proceso de LQA y los errores de lingüísticos más comunes que se producen a la hora de elaborar un videojuego (Capítulo 2).

2. Investigar sobre distintos OCRs, ver como funcionan, como usarlos para detectar texto en imágenes, como entrenar un modelo sabiendo el idioma y la fuente usada (Capítulo 2).

3. Elegir de los errores más comunes de localización aquellas que estén relacionados con errores de caracteres o posiciones que sea posible ser detectado utilizando una OCR (Capítulo 3).

4. Hacer un diseño de la herramienta detallando los test, el uso de OCR y la salida de la herramienta (Capítulo 3).

5. Elaborar una serie de tests que sean capaz de detectar si existe algún error de localización específico teniendo información de entrada de la OCR (Capitulo 4).

6. Integrar la librería de OCR usada y otras librerías y técnicas auxiliares de la herramienta (Capitulo 4).

7. Hacer una evaluación con los OCRs seleccionados teniendo en cuenta el tiempo y el porcentaje de acierto (Capitulo 5).

8. Hacer una evaluación por separado del funcionamiento de cada módulo (OCR, test) (Capitulo 5).

9. Unificar ambos módulos y hacer una evaluación completa de la herramienta (Capitulo 5).

Capítulo 2

Estado de la Cuestión

Antes de empezar a diseñar la herramienta es necesario conocer lo que usan o lo que hacen normalmente las empresas actuales con la adaptación del videojuego a otros países o regiones. En este capitulo contaré que es la internacionalización, localización, LQA y errores lingüísticos comunes. Además necesitaremos una herramienta que nos permita obtener la información necesaria desde una imagen, para ello esta el OCR (*Optical Character Recognition*), elegiremos y evaluaremos unas de ellas.

2.1. Internacionalización

La internacionalización es el proceso de preparar un producto para que pueda admitir diferentes idiomas y convenciones culturales sin necesidad de volver a rediseñarse.

Según Muñóz Sánchez (2017), la internacionalización de un videojuego implica una serie de decisiones técnicas y de diseño que permiten su adaptación posterior a múltiples idiomas y culturas sin necesidad de modificar su estructura principal. Esto implica separar la lógica de la aplicación de los elementos dependientes del idioma y de la cultura, como los textos, las fechas, las divisas, y otros formatos específicos. A continuación, se detallan algunos de los aspectos más relevantes según Muñóz Sánchez (2017):

- Separación del texto traducible de código fuente. Esto ayuda a que el traductor se centre únicamente en el trabajo de traducción y no tenga que acceder al código fuente.
- Uso de fuentes adecuadas para distintos idiomas. Las fuentes deben mostrar todos los posibles caracteres del idioma seleccionado, el fallo de esto puede conllevar a que en el videojuego no se vea el texto completo.
- Uso de la codificación adecuada para distintos idiomas.Según Cod (2025) la codificación es la forma en que se traduce la información para que pueda ser

interpretada y utilizada por una máquina.La mala elección o uso de la codificación puede provocar que se genere caracteres no legibles como se muestra en la figura 2.1, al igual que el uso incorrecto de la fuente donde la fuente no incluye algún carácter que se esta utilizando ,por tanto es necesario que la codificación pueda interpretar un carácter o símbolo introducido por teclado. Se usa el estándar de Unicode.

- Diseño de interfaces adaptables al texto que se debe mostrar. La misma cadena de texto en distintos idiomas traducidos pueden necesitar distintos espacios para mostrarlo en pantalla, esto puede conllevar a un problema a la hora de diseñar la interfaz y el espacio que tiene en el videojuego para mostrar una cadena. Muñóz Sánchez (2017) plantea las siguientes posibilidades de solucionar el problema:

  - Uso de fuente de ancho variable siempre que sea posible. La fuente elegida afecta mucho a la hora de mostrar más o menos caracteres. Existen las fuentes de ancho fija (todos los caracteres ocupan exactamente el mismo número de píxeles), fuentes de ancho variable (cada carácter ocupa un determinado número de píxeles). Poner ejemplos.
  - Uso de bocadillos, cajas o ventanas de texto adaptables al contenido. El tamaño de las ventanas se adaptará al tamaño de texto que hay dentro.
  - Uso de menús y botones con gran espacio o adaptables al contenido. En muchas ocasiones surge el problema de que el texto traducido no quepa en el espacio del botón o del menú, para ello se recomienda que se hagan con espacio suficiente para abordar cualquier posible tamaño de la traducción o que sean adaptables al contenido.

- Uso de etiquetas especiales para marcar género, sexo o número. Un problema a la hora de traducir reside en la concordancia de género, sexo y número en distintos idiomas, un ejemplo es *"You are so nice"* se puede traducir como "Eres muy simpático" o "Eres muy simpática". Por lo tanto es necesario la existencia de un sistema que cambie la palabra según si se trata de un personaje masculino o femenino.

- Facilitación de un mínimo de información contextual. En distintos idiomas puede resultar que la traducción sea diferente dependiendo del contexto, esto el traductor no lo sabe, por lo tanto es necesario un mínimo de información contextual para agilizar el proceso de traducción.

- Comprobación cultural de imágenes e iconos. Algunos gestos, imágenes o iconos pueden resultar inapropiados para algunas culturas o religiones hay que tener especial cuidado con la zona de localización.

Figura 2.1: Glifos con lo que se representa comúnmente cuando hay error de fuente.

2.2. Localización

Según el articulo de Chatterjee (2023), la **Localización** (o L10N), es el proceso por el cuál un producto se adapta a un idioma y/o cultura diferente del idioma o cultura en el que el producto fue creado satisfaciendo las necesidades. Esto incluye, pero no se limita, a la traducción de los textos y audios.

En el contexto de los videojuegos, la localización es esencial para proporcionar una experiencia inmersiva y auténtica a los jugadores de diferentes regiones. Esto incluye la adaptación de referencias culturales, chistes, juegos de palabras y otros elementos lingüísticos que podrían no tener un significado directo en el idioma de destino. Por ejemplo, si un videojuego contiene juegos de palabras complejos, el localizador debe encontrar equivalentes adecuados en el nuevo idioma que transmitan el mismo efecto o emoción, en lugar de realizar una traducción literal.

Además, la localización puede abarcar aspectos como la adaptación de símbolos, iconos, colores y otros elementos visuales que puedan tener connotaciones diferentes en diversas culturas. También es importante considerar las diferencias en formatos de fecha, hora, monedas y unidades de medida para garantizar que el producto sea completamente funcional y comprensible para el usuario final.

2.3. Localization Quality Assurance

Según Hashemi-Pour y Alexander-S-Gillis (2024), QA (*Quality Assurance*) es un proceso sistemático para garantizar que un producto o servicio cumpla con los requisitos de calidad establecidos. No se trata de probar un producto, sino de prevenir errores y defectos a lo largo del proceso de desarrollo. Esto ayuda al aumento de la confianza de los usuarios, a la vez ayuda al proceso y la eficiencia del desarrollo.

En cuanto LQA (*Localization Quality Assurance*) según Belcic (2022) es un proceso esencial en la localización de contenido que garantiza que el producto adaptado sea preciso, funcional y culturalmente apropiado para su mercado objetivo. A diferencia de la simple corrección de textos, LQA evalúa el contenido en su contexto real, considerando aspectos lingüísticos, visuales, funcionales y culturales.

- Lingüístico: Verifica la precisión de la traducción, la gramática, la ortografía y la coherencia terminológica.

- Visual: Asegura que el texto traducido se visualice correctamente en la interfaz, sin desbordamientos ni problemas de formato.
- Funcional: Comprueba que todas las funciones del producto operen según lo esperado en el idioma de destino.
- Cultural y legal: Evalúa que el contenido sea apropiado culturalmente y cumpla con las normativas legales del mercado objetivo.

Integrar LQA en el flujo de trabajo de localización desde las etapas iniciales permite identificar y corregir problemas oportunamente, mejorando la eficiencia y reduciendo costos. Herramientas como Gridly facilitan este proceso al centralizar la gestión de contenido y permitir colaboraciones en tiempo real entre desarrolladores, traductores y testers. Además proporciona control de versiones, revisión contextual donde puede comprobar como se vería por ejemplo un botón con el texto si encaja o no, validación automática de reglas donde puedes definir validaciones como longitud máximo del texto, uso consistente de mayúsculas, prohibición de ciertos términos.

2.4. Errores de localización comunes

Según Muñóz Sánchez (2017) existe numerosos errores de localización en los videojuegos a continuación se describirá cada uno de ellos.

2.4.1. Problema de fuente

Este error se da comúnmente cuando la fuente utilizada en el videojuego no incluye algunos caracteres especiales de un idioma, como puede ser la ñ en español o caracteres del chino, normalmente se presentan como se muestra en la figura 2.1 Un ejemplo podría ser como se muestra en la figura 2.2 donde algunos caracteres chinos no se muestran correctamente.

Figura 2.2: Twitter de un jugador chino que reporta un error en la fuente del juego.

2.4.2. Implementación de texto incorrecta

Este error se ilustra cuando el texto que debería aparecer en un idioma, aparece en otro diferente o aparece en forma de "placeholder"(Figura 2.3). Este error se produce debido a que hay un error en el documento de localización o porque el programa en sí tiene algún error. Debido a esto, el *tester* tiene que verificar que la celda del documento tiene el texto adecuado.

Figura 2.3: Aparece un placeholder en vez del nivel alcanzado

2.4.3. Cadena no localizada

Este error se produce cuando un texto no está traducido como se muestra en la figura 2.4. Esto puede ocurrir cuando el traductor no haya traducido el texto o que el desarrollador no haya proporcionado el texto.

Figura 2.4: Parte del texto se queda sin ser traducido.

2.4.4. Error tipográfico o de ortografía

Surge cuando existe alguna falta de ortografía en el texto traducido. Por ejemplo en el juego de GhostBusters, los jugadores se encuentran con "Conglatulation" donde lo correcto sería "Congratulations" como se muestra en la figura 2.5

Figura 2.5: Error tipográfico en el juego de GhostBusters.

2.4.5. Error gramatical

Surge cuando existe un fallo gramatical en el texto traducido. Es necesario tener un conocimiento perfecto del idioma al que se traduce.Un ejemplo sería como se muestra en la figura 2.6 donde el personaje dice "Did something happened?", en lugar de "Did something happen?" que sería lo correcto.

Figura 2.6: Error gramatical en ingles del juego.

2.4.6. Error de traducción

Este error se da cuando el texto traducido es errónea(p.ej "Dinero" traducido a "monkey" en lugar de "money"). Este error es uno de los errores más importantes ya que dar información incorrecta al jugador implica dar una experiencia desastrosa al jugador. Suele ocurrir cuando es mal traducido, por falta de información u omisión de palabras o que no tenga sentido en el contexto. Un ejemplo es como se muestra en la figura 2.7 donde chest puede ser traducido como "cofre" o "pecho" dependiendo del contexto, en este caso la traducción al chino ha sido pecho en vez de cofre.

Figura 2.7: Mal traducción de cofre (*chest*) al chino.

2.4.7. Solapamiento de texto

Este error sucede cuando el texto se sale de los limites fijados para él, puede ser un botón o un cuadro de texto. Puede deberse por una mala implementación por parte del desarrollador o que el traductor no haya respetado los límites de espacios proporcionado.(Figura 2.8)

Figura 2.8: El texto sobresale de la caja delimitadora.

2.4.8. Truncamiento de texto

Este error se produce cuando un texto aparece cortado y no se muestra por completo. Puede deberse a que no haya suficiente espacio dentro de los limites fijados. (Figura 2.9)

Figura 2.9: El texto confirm no aparece completo.

2.4.9. Error terminológico o incoherencia

Este error se produce cuando se usa una palabra incorrecta para referirse a una terminología clave (p. ej. utilizar el término "grabar" en lugar de "guardar"). También puede darse cuando el mismo término no está escrito de la misma manera en dos lugares distintos (p. ej. Escribir un término importante en minúsculas en un lugar y en mayúsculas en otro.). Un ejemplo podría ser que en la descripción se describa como un tipo de daño, sin embargo, en el efecto aparezca otro como se muestra en la figura 2.10.

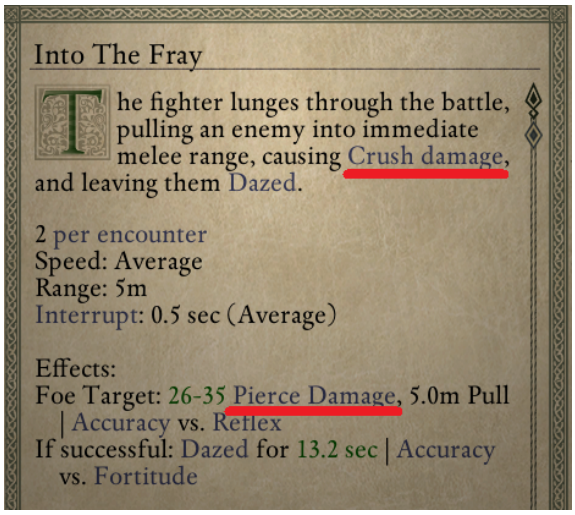

Figura 2.10: Incoherencia entre la descripción y el efecto.

2.4.10. Incumplimiento de directrices

Este error se da cuando una traducción no se cumplen las reglas de estilo que impone un distribuidor o plataforma para los videojuegos que se publiquen en ella. Por ejemplo, los mensajes del sistema en videojuegos en consolas deben de escribirse exactamente como aparecen en las TRC (*Technical Requirements Checklist*) de la

compañía.

2.4.11. Error de subtítulos

Los subtitulos pueden tener diferentes error, los más comunes son los siguientes:

- Los subtitulos no aparecen.
- Los subtitulos aparecen desincronizados
- Subtitulos muy largos por lo que no da tiempo leer.
- No estar bien segmentados como se muestra en la figura 2.11
- No coinciden con el audio doblado.

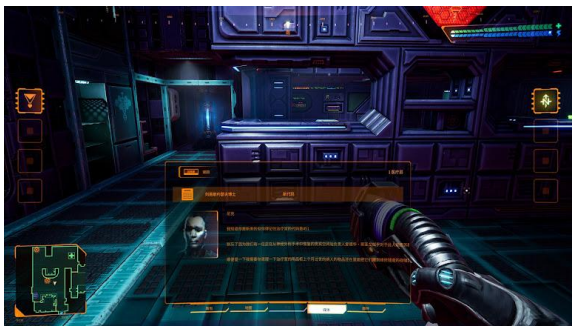

Figura 2.11: Falta de segmentación por lo que dificulta la lectura.

2.4.12. Error de audio

Al igual que los subtitulos, los errores de audio puede ser de varios tipos:

- El dialogo no esta doblado o no se escucha
- El audio esta desincronizado
- El audio no coincide con el contexto.

2.4.13. Problemas culturales

Este error se suele dar cuando en un videojuego se utiliza alguna simbología o gesto que tiene un significado ofensivo en otra cultura. Un ejemplo puede ser la utilización del símbolo manji en un juego japonés, que es muy parecido a la esvástica de la Alemania nazi como aparece en la figura 2.12.

Figura 2.12: Diferencia entre manji y esvástica.

2.5. Reconocimiento Óptico de Caracteres (OCR)

El OCR (Reconocimiento Óptico de Caracteres, por sus siglas en inglés *Optical Character Recognition*) es una tecnología que convierte imágenes de texto manuscrito, impreso o mecanografiado en datos de texto que las computadoras pueden interpretar y manipular según AWS (2024). En otras palabras, puede extraer el texto contenida en una imagen. Según AWS (2024) el OCR se usa en varios sectores:

- El sector bancario utiliza el OCR para procesar y verificar el papeleo de documentos de préstamo, cheques de depósito y otras transacciones financieras.
- El sector de la salud utiliza el OCR para procesar registros de pacientes, incluidos tratamientos, pruebas, registros hospitalarios y pagos de seguros.
- Las empresas de logística utilizan el OCR para rastrear etiquetas de paquetes, facturas, recibos y otros documentos de manera más eficiente.

Según TesseractDoc (2023) y OcropusDoc (2015) para llegar al objetivo de reconocer el texto y extraerlo de la imagen el OCR da una serie de pasos:

1. **Preprocesamiento de la imagen:**

   El propósito de este paso es procesar la imagen para que sea más legible para la computadora y reconocer así de forma mas eficiente los textos.

   - Conversión a escala de grises: La mayoría de los OCR primero convierten la imagen en blanco y negro o escala de grises para simplificar el procesamiento.
   - Reducción de ruido: Se aplican filtros para eliminar manchas, borrones o marcas en la imagen que puedan afectar la precisión del reconocimiento.
   - Binarización: La imagen se convierte a una representación en blanco y negro, donde los píxeles se clasifican como parte del fondo (blanco) o del texto (negro). Este proceso ayuda a aislar los caracteres.

2. **Segmentación:**

El OCR divide la imagen en secciones manejables. Primero separa líneas de texto. Luego separa las palabras y finalmente descompone las palabras en caracteres individuales. Este paso es crucial, ya que el OCR necesita reconocer los caracteres de manera individual, pero considerando también su contexto dentro de una palabra o frase.

3. **Detección de características:**

   Extracción de características de los caracteres: El OCR analiza los caracteres y sus formas, midiendo varios atributos como las líneas, contornos, cruces de líneas, y la disposición de los píxeles. Estos datos son usados para diferenciar letras, números y símbolos similares (como "O" y "0" o "l" y "1"). Se aplican técnicas basadas en modelos geométricos, estructuras de redes neuronales o de aprendizaje automático para identificar patrones comunes en los caracteres.

4. **Reconocimiento del carácter:**

   Una vez identificadas las características, el OCR las compara con una base de datos o "alfabeto" interno de posibles caracteres. Esto puede hacerse de varias maneras según AWS (2024), dependiendo del tipo de OCR:

   - Métodos basados en plantillas: Se compara cada carácter con un conjunto de plantillas predefinidas. Si la forma del carácter coincide con una plantilla, se clasifica como ese carácter.
   - Métodos basados en aprendizaje automático o redes neuronales: Las técnicas modernas de OCR suelen usar redes neuronales entrenadas con miles de ejemplos de texto. El sistema "aprende" a identificar caracteres y a hacer distinciones más sutiles en base a sus experiencias pasadas.

2.5.1. Librerías de OCR

Durante el desarrollo de este trabajo, se evaluarán varias herramientas de OCR para seleccionar la más adecuada para nuestras necesidades. Las herramientas que probaremos serán OCRopus[1], EasyOCR[2] y Tesseract[3]. A continuación, se describe brevemente cada una de ellas.

1. **OCRopus** Es un sistema de OCR basado en redes neuronales con un enfoque modular. Principalmente esta disponible para Linux. Proporciona diferentes módulos para la binarización, segmentación, generación del ground-truth y el entreno del modelo. La última actualización de esta librería fue en 16 de Diciembre de 2017

2. **EasyOCR** EasyOCR es una biblioteca de reconocimiento óptico de caracteres (OCR) desarrollada en Python que facilita la extracción de texto de imágenes y documentos. Diseñada con el objetivo de ser fácil de usar y accesible, una de

[1]Repositorio github de OCRopus: `https://github.com/ocropus-archive/DUP-ocropy`
[2]Repositorio github de EasyOCR: `https://github.com/JaidedAI/EasyOCR`
[3]Repositorio github de Tesseract: `https://github.com/tesseract-ocr/tesseract`

las características de EasyOCR es que es capaz de reconocer texto en más de 80 idiomas y soporta múltiples scripts, lo que la convierte en una herramienta versátil para diversas aplicaciones. Proporciona gran variedad de modelos entrenados y también ofrece la opción de entrenar tu propio modelo.

La herramienta en resumen se adapta muy bien a nuestras necesidades.

3. **Tesseract**

   Tesseract es una biblioteca de (OCR) de código abierto que se utiliza para convertir imágenes de texto en texto editable. Originalmente desarrollado por Hewlett-Packard en la década de 1980, Tesseract fue liberado como software de código abierto en 2005 y ha sido mantenido y mejorado por la comunidad, particularmente por Google, que ha contribuido significativamente a su desarrollo.

   Al igual que EasyOCR una de las características más destacadas de Tesseract es su capacidad para reconocer texto en múltiples idiomas y alfabetos, soportando más de 100 idiomas de forma nativa. Esto lo convierte en una herramienta versátil para aplicaciones globales que requieren la extracción de texto de documentos escritos en diferentes lenguas.

   Tesseract utiliza técnicas avanzadas de aprendizaje profundo y redes neuronales, lo que le permite alcanzar altos niveles de precisión en el reconocimiento de texto. Su arquitectura se basa en el uso de modelos de aprendizaje profundo entrenados con grandes conjuntos de datos, lo que mejora continuamente su rendimiento en diversas condiciones.

   El motor puede integrarse fácilmente en aplicaciones de procesamiento de imágenes y visión por computadora. Tesseract se utiliza comúnmente en combinación con bibliotecas como OpenCV para preprocesar imágenes antes de la etapa de reconocimiento, lo que mejora la precisión de los resultados.

   Además, Tesseract ofrece una API en varios lenguajes de programación, lo que facilita su integración en diferentes entornos de desarrollo.

2.6. Conclusión

En conclusión, la internacionalización trata de preparar el producto para que sea fácilmente admitir diferentes idiomas y para ello es necesario seguir algunas pautas como la separación del texto traducible de código fuente. La localización es el proceso por la cual un producto se adapta a un cierto idioma o cultura, esto es esencial en videojuegos para que los jugadores de otras regiones tengan una experiencia inmersiva y auténtica. QA(*Quality Assurance*) se trata de un proceso sistemático para garantizar que un producto cumpla los requisitos y LQA(*Localization Quality Assurance*) es el mismo proceso centrándose en aspectos de la localización como los aspectos lingüísticos, visuales, funcionales y culturares. El OCR es una tecnología que consigue convertir imágenes con texto en datos de texto que la computadora pueda entender.

Esta sección contribuye a comprender el funcionamiento de un sistema OCR, así como la existencia de errores de localización que deben ser resueltos. Además, se analiza cómo se representan visualmente estos errores en pantalla, lo que resulta fundamental para el diseño de las distintas partes de la herramienta tanto los tests como el propio OCR que se detallan en el siguiente capítulo.

Capítulo 3

Descripción del Trabajo

En este capítulo se describirá el diseño de la herramienta detallando las funcionalidades de los dos módulos que va a tener, el de OCR y el de test. Se contará el formato de la salida de la herramienta y la forma en la que va a ser interpretada.

3.1. Diseño de la herramienta

Se plantea el diseño de una herramienta que, a partir de imágenes capturadas del videojuego en desarrollo, permita realizar pruebas de localización. El usuario debe proporcionar las imágenes, una configuración específica y el texto esperado. A partir de estos datos, la herramienta será capaz de reconocer el texto presente en la imagen y detectar posibles errores de localización entre los previamente definidos.

El objetivo es que esta herramienta sirva de apoyo al personal de LQA (Language Quality Assurance) durante el proceso de desarrollo del videojuego.

El flujo de trabajo comienza con la introducción de los datos por parte del usuario. A continuación, la herramienta procesa estos datos y los envía al sistema OCR, que se encarga del reconocimiento del texto y de la extracción de información relevante, como la posición del mismo. Posteriormente, esta información es transferida al módulo de tests, encargado de identificar posibles errores de localización. Finalmente, los resultados obtenidos se devuelven al programa principal, el cual genera un informe con el diagnóstico completo del análisis (Figura 3.1).

Para ello la herramienta constará de dos módulos:

- Modulo del OCR.
- Modulo del test.

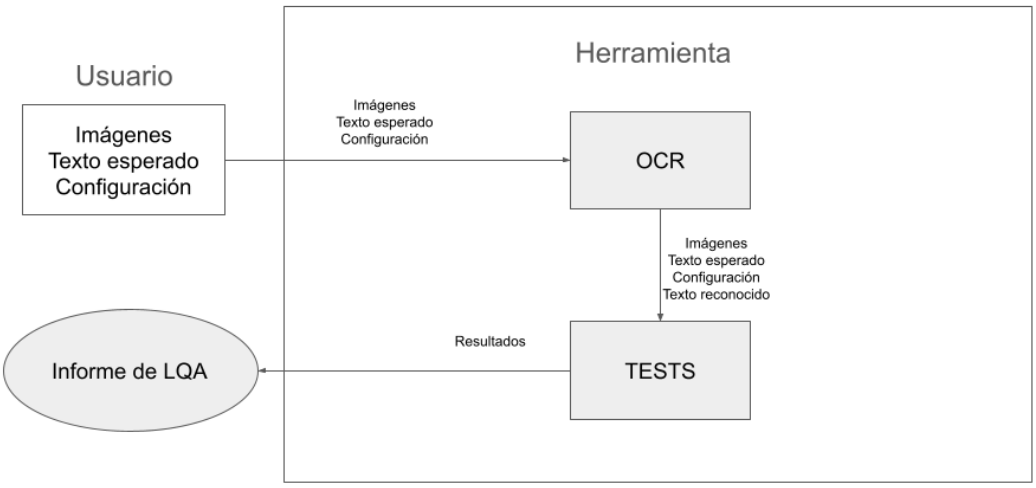

Figura 3.1: Workflow de la herramienta y módulos que la componen.

3.2. Modulo de OCR

El modulo de OCR se encarga de todo lo relacionado con la detección de texto en imágenes y la mejora de la detección.

En este modulo se elegirá una librería de OCR para la detección de texto y tendrá principalmente dos funcionalidades:

- Entrenar modelo: Dado una fuente y un idioma, el OCR entrena un modelo especifico para esa fuente en concreta y ese idioma en concreto, mejorando así el resultado obtenido. Este proceso se hace una sola vez como preparación previa de la herramienta, pero si el usuario desea o necesita varias fuentes o idiomas, puede generarlas las veces que sea necesario.
- Reconocer texto: El OCR reconoce texto de todas la imágenes y se los pasa al módulo de test.

En este módulo, recibirá los datos proporcionado por el usuario y procesará multiples imagenes con un modelo concreto ya sea entrenado o el por defecto de la herramienta. En la configuración es necesario que el usuario proporcione información como:

- La ruta de las imágenes.
- La ruta del texto esperado.
- La ruta del modelo a utilizar.
- Nombre del modelo a utilizar.

En este módulo, cada imagen para por un proceso de preprocesado para mejorar la calidad de la imagen en el sentido de que el OCR reconozca mejor los caractéres y generará un texto. Ese texto antes de ser pasado como entrada al siguiente modulo será procesado otra vez para minimizar el error producido por el OCR. Debido a la necesidad de algunos test, también se ha considerado obtener informaciones como posición del texto en la imagen en este modulo.

3.3. Modulo de test

El modulo de test se cargará de verificar si existe algún error de localización con el texto obtenido del OCR.

Usará como entrada el texto reconocido del modulo de OCR y el texto esperado, proporcionado para validar los tests. Pasará por una serie de tests para verificar si existe o no algún error de localización. La salida de este modulo será el resultado de los tests indicando si ha pasado el tests y en caso negativo indicar (si es posible) donde se produce el fallo.

Con esa salida se generará un archivo con la información de los resultados que luego será transformado en un informe para que sea entendible por un ser humano.

No se van a abordar todos los tests de descritos en la sección 2.4. A continuación se describen los errores para los que se han creado tests, considerando la importancia y la relevancia con nuestra herramienta(p.ej atacar el error de audio no tiene sentido en este trabajo).

3.3.1. Similitud

Este es el test "master", la cual es ejecutado antes de todos los tests que viene a continuación. Su principal funcionalidad es para obviar la ejecución de algunos tests y mejorar el rendimiento del programa obviando ejecuciones innecesarias. En este test se obtiene la cadena reconocida por el OCR y lo compara con la cadena esperada obteniendo un porcentaje de similitud. Si ambas cadenas son iguales, significa que no existe errores en cuanto al contenido del texto, por lo que se puede obviar algunos test como el de truncamiento(subsección 3.3.4) y el de placeholder(subsección 3.3.3). Tests como el de solapamiento(subsección 3.3.2), no tiene que ver con el contenido del texto, sino de la posición, estas se siguen ejecutandose con normalidad.

3.3.2. Solapamiento de texto

Para resolver este problema se plantea usar el OCR para detectar las posiciones de los textos, seguido de una librería gráfica para detectar el contorno de la caja delimitadora guardado para ese texto, y obtener así sus posiciones. Comparando la posición del texto y la posición de la caja delimitadora obtenemos si el texto se está saliendo de la caja o no.

Para simplificar el test, damos algunas restricciones:

- En la imagen, el texto tiene que aparecer completamente en horizontal sin ninguna rotación sobre el eje X.
- Detectamos texto que sobresale por los lados.
- Tiene que estar resaltado y tener un claro contraste con el fondo y el texto para facilitar la detección.
- Debe tener una forma de caja o ventana(cuadriculada, cuadriculada redondeada). No sirve, por ejemplo, un bocadillo en forma de nube.

La salida indicará si existe o no solapamiento en la imagen.

3.3.3. Implementación Incorrecta

Este error aparece cuando por algún error del desarrollador aparecen placeholders en la pantalla del juego en vez del texto (p.ej si en el juego me llamo "pepe", en vez de aparecer en el dialogo "Hola pepe", aparece un "Hola `%*player_name*%`''). Para resolver este problema es necesario especificar como entrada cuál es el formato utilizado para el placeholder indicando su apertura y cierre. Con los datos, hacer una búsqueda del placeholder en el texto reconocido por el OCR.

Se tiene en cuenta las siguientes restrincciones:

- El placeholder es formado por al menos un carácter de apertura y al menos un carácter de cierre. Puede tener más de un carácter en ambos lados.
- Todo lo que este entre la cadena de apertura y cadena de cierre se considera placeholder sin importar el número de palabras que hay dentro.
- Existe un solo tipo de placeholder, de momento no soporta la detección de múltiples tipos de placeholders.

La salida indicará si existe placeholder en la imagen y la posición en la que se encuentra para una mejor búsqueda por parte del personal de QA.

3.3.4. Truncamiento de texto

Para resolver este problema se plantea obtener la cadena esperada y la cadena reconocida por el OCR. Se compara la longitud de ambas cadenas, si la longitud es la misma, entonces no existe truncamiento. En caso contrario, se hará comparaciones palabra a palabra buscando la existencia de subcadenas (p.ej "ment" es subcadena de "mente"), en caso de encontrar subcadenas, entonces existe un truncamiento. Si no se encuentran subcadenas, significa que son diferentes frases por lo que sería otro tipo de error.

La salida indicara si existe o no truncamiento en el texto.

3.4. Informe de resultados

Una vez que se han ejecutado todos los tests sobre cada imagen, se generará un informe con los resultados, donde se especificará:

1. Nombre de la imagen.
2. Texto esperado y texto reconocido.
3. Pass/fail de los tests e información extra de los tests dependiendo de cada test.

Con este informe se generará un html con la información de los resultados como se muestra en la figura 3.2.

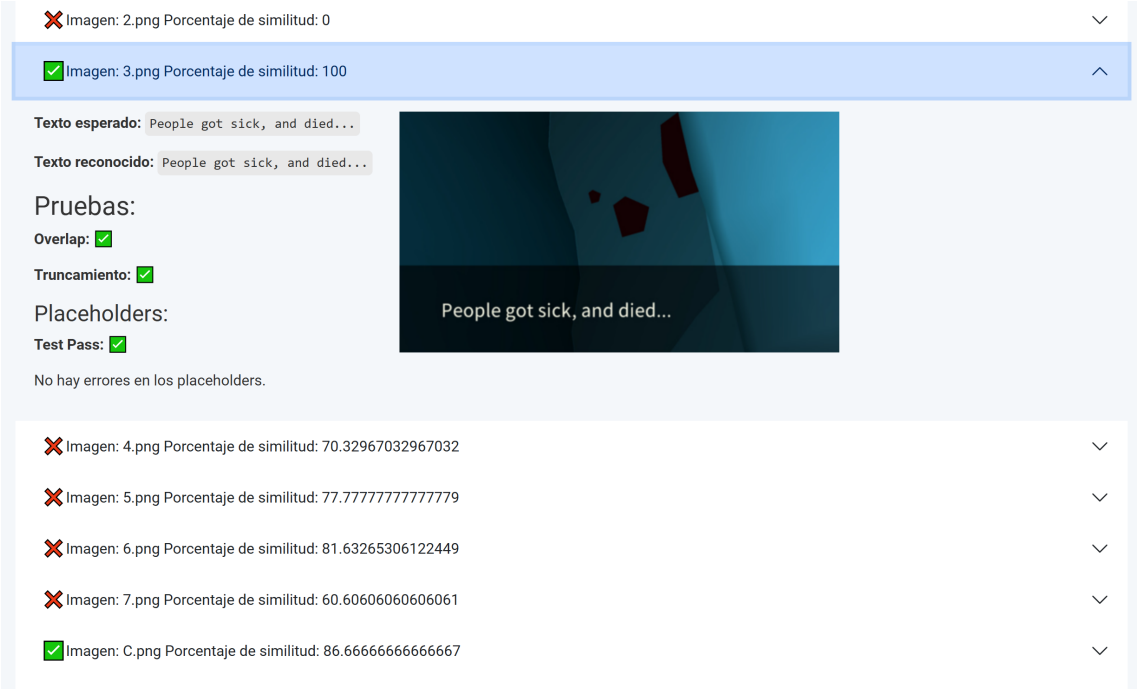

Figura 3.2: Ejemplo de informe.

3.5. Conclusión

En este capítulo se ha descrito las partes de la herramienta detallando cada módulo, uno para el reconocimiento de texto en imágenes usando OCR y otro para la comprobación de errores de localización usando tests. El modulo de OCR recibirá como entrada las imágenes y una determinada configuración, reconocerá el texto y esta será pasado al modulo de tests para comprobar si existe algún error generando así un informe. Una vez realizado el diseño de la herramienta, pasamos a describir como ha sido implementada en el siguiente capítulo.

Capítulo 4

Implementación de la herramienta

En esta sección se detallará la implementación de la herramienta, especificando cómo se ha implementado cada modulo y cada uno de los tests. Para la implementación de la herramienta se ha decidido utilizar *C++* como lenguaje de programación. Para implementar el módulo de OCR, se ha decidido usar la librería Tesseract ya que proporciona api en *C++*. Para facilitar el uso de la herramienta se ha decidido desplegar la aplicación en un contenedor. Como la aplicación se ejecuta dentro del entorno del contenedor, podemos usar la aplicación en cualquier sistema operativo. Para ello se ha decidido utilizar la herramienta de Docker[1].

Todo el material de la herramienta de testing de localización se encuentra en el repositorio de github[2].

La herramienta de testing de localización tiene una arquitectura de clases como se muestra en la figura 4.1.

[1]Docker `https://www.docker.com/`

[2]Repositorio de github de la herramienta: `https://github.com/Dewo2000/TFG-2024-2025`

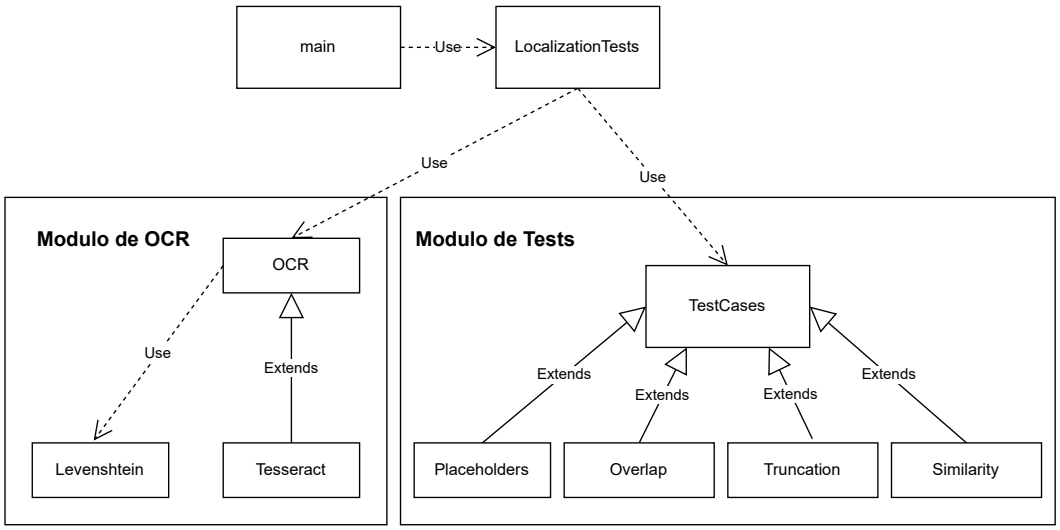

Figura 4.1: Diagrama de clases de la herramienta.

4.1. Módulo de entrada

Este es el módulo donde estará el programa principal donde se encarga de recibir los argumentos del usuario, parsear los argumentos, parsear los datos de configuración y llamar a los módulos de OCR y tests. En el programa principal recibirá como parámetro:

- –train. Para entrenar un modelo y se necesitará otros parametros para el entreno:
  - -f font. Para especificar la fuente.
  - -l lan. Para especificar el lenguaje.
  - -i iter. El número de iteraciones.
  - –clear . Si se elimina el entreno anterior del mismo modelo.
- –test. Para hacer el test dada un archivo de configuración -c config.

Los datos proporcionados por el usuario son gestionados por la clase `LocalizationTests`, la cual se encarga de parsear los argumentos de entrada. En caso de que se indique un proceso de entrenamiento, esta clase invoca al módulo OCR en modo entrenamiento. Si, por el contrario, se trata de una ejecución de testing, se leerán del archivo de configuración los parámetros necesarios para llevar a cabo las pruebas.

`LocalizationTests` actúa como coordinadora entre los módulos principales del sistema: realiza las llamadas al OCR para el reconocimiento de texto y, una vez finalizado este proceso, organiza la ejecución de los tests de localización.

Los resultados de cada test son almacenados en un archivo en formato JSON, el cual se guarda en la ruta de salida especificada. Este archivo JSON puede ser interpretado posteriormente para generar un informe en formato HTML, cuyo proceso de conversión se detalla en la sección 4.4.

A continuación se detalla la implementación de cada módulo.

4.2. Implementación del modulo OCR

En el modulo del OCR, tenemos principalmente dos partes, el entrenamiento del modelo y el reconocimiento de texto. Para la librería de OCR, en este caso se ha elegido Tesseract que cumple nuestras necesidades en tiempo y en precisión. La evaluación de las librerias de OCR se describe en la sección 5.1.

4.2.1. Entrenamiento de modelo

Para el entrenamiento del modelo se ha usado la plantilla proporcionado por Robayo (2023) donde se ha hecho el entreno usando la api de Tesseract en python. Para nuestro caso, hasta el día de la implementación, Tesseract no ha sacado ningún API para el entreno de modelo en *C++*, por lo que en nuestra herramienta haremos llamadas al sistema con comandos proporcionado en la plantilla de Robayo (2023) para el entreno de modelo.

4.2.2. Reconocimiento de texto y mejoras

Este módulo se ha diseñado pensando en su extensibilidad de modo que es posible usar otras librerías de OCR diferentes a la que se está usando la herramienta. En este modulo es posible cambiar de libreria de OCR (p.ej EasyOCR) heredando de la clase virtual pura `OCR` y completando las funciones necesarias con métodos de esa librería sin afectar al funcionamiento completo de la herramienta.

El proceso de reconocimiento y mejoras es el siguiente:

1. Recibidos parámetros del programa principal, este crea las carpetas correspondientes a la salida usando llamadas al sistema.
2. Recorrerá la carpeta de imágenes preprocesando cada imagen con OpenCV.
3. La imagen resultante del preprocesado será pasado a la librería de Tesseract para su reconocimiento de texto.
4. Seguido de una limpieza de la salida de OCR usando la distancia Levenshtein en el caso de que el usuario haya proporcionado la salida esperada.
5. Al finalizar el proceso, escribirá la salida en el directorio de la salida con el formato *nombre_ imagen.txt*.

Para la recogida de parámetros, es necesario un archivo de configuración con el siguiente formato:

- imagePath. Directorio donde se encuentra las imágenes.
- gtPath. Directorio donde se encuentra la cadena esperada de cada imagen.
- model. Nombre del modelo que se va a usar.
- modelPath. Directorio donde se encuentra el modelo que se va a usar.
- outputPath. Directorio de salida de la información.
- OCR. El ocr que se va a usar. Por defecto Tesseract.
- placeholders. El formato de la cadena de placeholder que se usa. Debe tener un campo `begin` con la cadena de apertura del placeholder y un campo `end` con la cadena de cierre.

El preprocesado se ejecuta en el método protegido `preprocessing` de la clase `OCR`, la cual se usan métodos proporcionado por OpenCV de las cuales son:

- `cvtColor` Cambia la imagen a escala de grises.
- `Resize`. Cambia las dimensiones la imagen, lo cual ajusta el tamaño de la imagen para un mejor reconocimiento.
- `Threshold` Aplica simple thresholding la cual asigna un valor binario a cada píxel dependiendo el umbral. Resalta mejor los caracteres del fondo.
- `fastNlMeansDenoising` Aplica reducción de ruido para mejorar la calidad visual y el reconocimiento.
- `medianBlur` y `GaussianBlur` Otra técnica para aplicar reducción de ruido suavizando la imagen.
- `dilate` y `erode` Técnica de dilatar y erosionar que ayuda a reducir ruidos y el cierre de contornos, mejora la detección de los caracteres.
- `createCLAHE` Ecualización adaptativa del histograma que mejora el contraste de la imagen.

La combinación anterior se ha obtenido haciendo estudio y experimento sobre las técnicas de preprocesamiento en la sección 5.2.

El siguiente paso es el reconocimiento de texto utilizando la librería de Tesseract. El método `getDirImgText` para reconocer texto de una carpeta de imágenes se sitúa en la clase `OCR`, la cual es un método virtual puro que en la clase de `Tesseract` debe implementar. En el método se hace lo siguiente:

1. Genera las carpetas de salida relacionado al reconocimiento utilizando llamadas al sistema y dando permisos de escritura a la carpeta.

2. Abre el directorio de carpetas con `opendir` y obteniendo la información en `dirent`, para obtener los nombres de los archivos y filtrar aquellas que son imágenes en formato png.
3. Ejecutará el preprocesamiento de cada imagen con las técnicas anteriores.
4. La imagen resultante será pasado al OCR con `SetImage` de Tesseract.
5. Se obtendrá el texto reconocido con `GetUTF8Text` de Tesseract.
6. Se hace la limpieza de caracteres utilizando distancia Levenshtein y se escribe en la salida el resultado.

La limpieza se hace porque el OCR puede devolver caractéres basura(p.ej reconoce una caja del fondo como un carácter), y esta basura afecta a la detección de errores. Como por ejemplo, la figura 4.2 puede devolver

```
"
|
\ J
-
Agumon /
Still, a ‘‘smahrt fown’’ is cool! It really can
do anything! <
```

donde las primeras líneas son caracteres basura que ha reconocido el OCR.

Figura 4.2: Diagrama de clases de la herramienta.

Para resolver este problema se utiliza la distancia Levenshtein, una métrica que calcula el grado de diferencia entre dos cadenas de texto con la idea de quedarnos

con aquellas líneas reconocidas que más se parezca a nuestra cadena esperada. La limpieza se hace con el método `findMostSimilarLine` de la clase `OCR`, lo que hace es calcular el valor de similitud de las cadenas utilizando la distancia Levenshtein y si supera un cierto umbral que definimos, entonces se considera la línea.

Tesseract también se encarga de devolver las coordenadas del texto en la imagen si es necesario, para ello, la información lo podemos encontrar en el método `BoundingBox` del `ResultIterator` de Tesseract.Para la implementación de este método, se usa `Pix` para leer la imagen, seguido del reconocimiento de texto de Tesseract. Una vez reconocido el texto, se usa el `ResultIterator` de Tesseract para iterar sobre los resultados donde se puede obtener las cajas de los textos utilizando el método `BoundingBox` de la clase `ResultIterator` de Tesseract.

También usamos OpenCV para el reconocimiento del contorno de la caja delimitadora para el error de solapamiento de texto. Utilizando `findContours` para encontrar el contorno y `approxPolyDP` para el filtrado de contornos ya que solo nos quedamos con los rectángulos. Para la implementación de este método se hace uso de OpenCV siguiendo estos pasos:

1. Se lee la imagen con `imread`.
2. Se pasa a grises con `cvtColor`
3. Se detecta los bordes con gradientes utilizando `canny`. Los gradientes es lo que mide cuán rápido cambia la intensidad del píxel respecto a sus vecinos.
4. Se obtiene los contornos con `findContours`
5. Se busca los contornos de forma de cuadrado utilizando `approxPolyDP` y obtenemos las coordenadas de una posible caja delimitadora.

4.3. Implementación del modulo test

Este módulo también se ha diseñado pensando en su extensibilidad. En este modulo, los tests heredan de la clase `TestCases` por lo que es posible implementar más tests de los que hay implementado sin afectar el funcionamiento general de la herramienta.

4.3.1. Test sobre solapamiento de texto

Este test está implementado en la clase `Overlap`. Para la implementación de este test se ha hecho el uso de Tesseract para obtener las coordenadas del texto en la imagen, además se hace el uso de OpenCV para la detección de la caja delimitadora del texto y la obtención de las coordenadas en la imagen.

La implementación de obtención de coordenadas del texto mediante Tesseract se sitúa en el módulo de OCR en la clase `Tesseract` en el método `getBoundingBoxes` la cual devuelve un vector de todas las cajas de texto.

La implementación de obtención de coordenadas de la caja delimitadora también se ha hecho en el módulo de OCR. En la propia clase `OCR` existe un método llamado `getButtonsFromImage` la cual devuelve un vector de todas las cajas delimitadoras.

Se comparan ambas coordenadas comprobando que parte del texto está dentro de la caja delimitadora y parte esta fuera de esa caja, por lo que se considera que existe un solapamiento de texto. En el test se puede especificar un tamaño mínimo de la caja delimitadora para que OpenCV ignore todos los contornos menores a ello.

4.3.2. Test sobre implementación incorrecta

Este test está implementado en la clase `Placeholders`. Para la implementación de este test se obtiene las cadenas delimitadoras de placeholder que se especifica en el archivo de configuración con su apertura y cierre. Después se hace el uso de expresiones regulares para la detección de subcadenas donde existan placeholder. Según Arasa (2022) define las expresiones regulares como una cadena genérica que se usa a modo de patrón y que sirve para localizar texto dentro de un texto mayor.

En nuestro caso usaremos el regex proporcionado en *C++* y el patrón es la siguiente:

```
std::wstring pattern = L"\\" + inicio + L"(.*?)" +L"\\" +  fin;
```

Donde "inicio" es la cadena de apertura del placeholder y "fin" es la cadena de cierre del placeholder.

Dado que los caracteres con tilde ocupan más de un byte en la codificación UTF-8, el cálculo de la posición de ciertos elementos, como los placeholders, puede resultar incorrecto si existen caracteres acentuados antes de dichos elementos. Esto se debe a que el conteo se realiza a nivel de bytes y no de caracteres, lo que provoca desajustes en la localización.

Para resolver este problema, se empleará la clase `wstring`, que utiliza caracteres anchos (wide characters). Esta clase permite representar correctamente cada carácter como una unidad, independientemente de su longitud en bytes, garantizando así una contabilización precisa de las posiciones dentro del texto.

Antes de realizar la búsqueda, se convierte la cadena codificada en UTF-8 a un objeto de tipo `wstring`. Este proceso implica primero la conversión de UTF-8 a Unicode utilizando la biblioteca ICU (International Components for Unicode), y posteriormente la transformación a `wstring`. Del mismo modo, una vez completado el procesamiento, se realiza la conversión inversa para devolver la cadena al formato UTF-8 si es necesario.

4.3.3. Test sobre truncamiento de texto

Este test está implementado en la clase `Truncation`. Para la implementación de este test es necesario tener la cadena esperada. Se compara el número de caracteres de la cadena esperada y la cadena reconocida por el OCR. En el caso de que sean iguales, no hay forma de que exista un truncamiento por lo que pasa el test. Si

la longitud de las dos cadenas son diferentes, se compara palabra a palabra de la cadena esperada y de la cadena reconocida. Si una es subcadena de otra, es decir que una contiene a la otra utilizando el método `substr` del `string`. Cada palabra que coincidan o que se detecte como subcadena de otra serán descartadas del bucle de comparación.

Si al final del bucle aun queda cadenas sin comparar, eso significa que falta palabras, por lo que es un truncamiento a nivel de frase.

4.4. Generación de informe

Una vez terminado el proceso de reconocimiento y detección de errores de localización con los test, la clase `LocalizationTests` genera un Json con el siguiente formato:

```
"nombre_imagen.png": {
 "tests": {
  "overall_pass": false,
  "overlap": {
   "test_pass": false
  },
  "placeholders": {
   "test_pass": false,
   "errors":[{
    "pos":20
    "contenido": &_placeholder_&
   }]
  },
  "similarity": 22.69503546099291,
  "truncamiento": {
   "test_pass": false
  }
 },
 "texto_esperado": texto
 "texto_reconocido": texto
}
```

Donde tendrá:

- Nombre de la imagen.
- Texto esperado y texto reconocido.
- Campo `tests` donde indica el resultado de cada test con un booleano en el campo `test_pass` e información extra si los tiene, como los `pos` y `contenido` cuando hay un error de placeholder.

- Campo `overall_pass` que indica si pasa todos los test o falla en alguno.
- Campo `similarity` que indica el porcentaje de similitud entre la cadena esperada y el reconocido.

Para la generación de informe se ha hecho el uso de handlebars[3] que permite generar html a partir de una plantilla.

4.5. Despliegue

Como se comenta al principio de este capítulo, esta aplicación se despliega en un contenedor de docker pensando que la herramienta se pueda ejecutar en cualquier sistema siempre y cuando se disponga de docker.

Docker es una plataforma de código abierto que permite el despliegue de una aplicación en un contenedor, donde solamente contiene lo necesario para la ejecución de la aplicación. La imagen de docker son construidos mediante un fichero DockerFile. En este documento viene las especificaciones técnicas que debe tener el entorno como las librerías que debe tener y las configuraciones de la imagen.A partir de esa imagen se construye contenedores donde podemos ejecutar el programa.

Existirá dos formas de ejecutar la herramienta dependiendo de la necesidad. Para ello se ha creado dos imágenes diferentes, uno para el desarrollo de la herramienta y otra para el uso de la herramienta.

Si se desea hacer una extensión de la herramienta como añadir más tests o usar una librería de OCR diferente,el proceso será:

1. Construir la imagen usando el DockerFile proporcionado.
2. Arrancar el contenedor con la imagen construida.
3. Abril un editor y conectarse al contenedor de docker.
4. Ejecutar el programa y conseguir los resultados.

Las instrucciones concretas se encuentran en el README.md.

4.6. Conclusión

La herramienta se despliega dentro de un contenedor Docker y está implementada en *C++*. El módulo de OCR utiliza la biblioteca Tesseract para el reconocimiento de texto, junto con OpenCV para el preprocesado de imágenes y la detección de contornos. Este módulo ha sido diseñado para ser extensible, permitiendo la integración de otras bibliotecas OCR mediante la herencia de la clase base `OCR` e implementando las funciones requeridas.

[3]Handlebars `https://handlebarsjs.com/`

Del mismo modo, el sistema de tests también es ampliable: se pueden incorporar nuevos casos de prueba mediante la herencia de la clase `TestCases`.

Una vez implementados tanto el módulo de OCR como el de tests, es necesario llevar a cabo una evaluación que permita determinar su efectividad y precisión. Este proceso de evaluación se describirá en detalle en el siguiente capítulo.

Capítulo 5

Evaluación de la herramienta

En este capítulo se describirá la evaluación de la herramienta separando en una evaluación para el módulo de OCR, otra para el módulo de tests y una evaluación de la herramienta de forma conjunta(end to end testing). En la parte de OCR se evaluará la librería utilizada, la mejora que proporciona la parte de preprocesamiento y la limpieza de caractéres basura utilizando distancia Levenshtein. En la parte de tests se evaluará la precisión de los tests asegurándonos de que no exista casos que de un falso error en el test. De forma genérica se evaluará la herramienta con los resultados obtenidos ejecutando la herramienta midiendo la precisión.

5.1. Pruebas y resultados de los OCR

En esta sección se hará una evaluación de los distintos OCRs para elegir la más conveniente para nuestra herramienta. La evaluación se hará sobre las librerías de Tesseract, EasyOCR y OCRopus, son estas elegidas por su disponibilidad gratuita.

El objetivo de la evaluación es elegir el OCR que mejor se adapte a nuestras necesidades, lo que resalta la importancia de la precisión del OCR. Para poder medir esa precisión, tendremos en cuenta el CER(*Character Error Rate*). Además el tiempo también es un factor importante que tendremos en cuenta.

El CER(*Character Error Rate*) es una métrica usada comúnmente para evaluar la precisión de un sistema de reconocimiento de caracteres u otros sistemas que transcriba texto. El valor se obtiene calculando el número de errores a nivel de carácter dividido por el número total de caracteres en el texto de referencia. Los errores pueden ser debidos a:

1. Sustituciones: Un carácter incorrecto reemplaza a uno correcto.
2. Inserciones: Falta un carácter que debería estar presente.
3. Eliminaciones: Aparece un carácter extra que no debería estar.

La fórmula general para calcular el CER es:

$CER = \frac{S+D+I}{N}$

Donde:

S es el número de sustituciones.

D es el número de eliminaciones.

I es el número de inserciones.

N es el número total de caracteres en el texto de referencia.

El CER se expresa como un valor entre 0 y 1 (o como porcentaje), donde 0 significa que no hubo errores (transcripción perfecta) y 1 (o 100) indica que todos los caracteres son incorrectos.

La metodología seguida para la evaluación es la siguiente:

1. Se hará la evaluación de forma independiente con las 3 librerías de OCR. Tesseract, EasyOCR y OCRopus.

2. Se ha elegido 13 imágenes nombrados del 1 al 13 siendo la imagen 1(figura 5.1) el más simple y el 13(figura 5.2) el más complejo. Para la clasificación de simplificad de las imágenes se ha tenido en cuenta el número de caracteres a reconocer y la complejidad del fondo, entre ellas, el contraste y el número de geometrías. Las imágenes se pueden encontrar en el repositorio del trabajo[1].

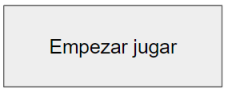

Figura 5.1: Ejemplo de imagen simple.

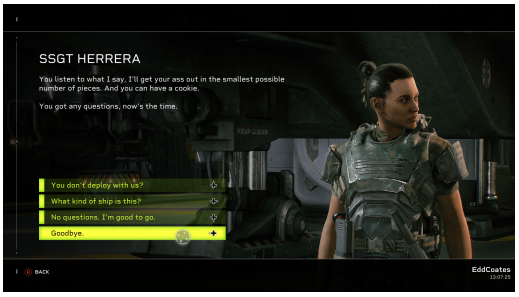

Figura 5.2: Ejemplo de imagen complejo.

3. Cada imagen será procesadas por el OCR utilizando el modelo por defecto de cada una de ellas.

4. Se utiliza la libreria Jiwer[2] en python para los cálculos de CER.

5. Se ejecutará 6 veces el proceso de reconocimiento de cada OCR para sacar el tiempo medio de cada una.

En los resultados de los OCR se encontrará un valor llamado CER medio, este valor se obtiene calculando el número total de errores a nivel de carácter encontrado

[1]Repositorio github con las imágenes de evaluación: `https://github.com/Dewo2000/TFG-2024-2025/tree/main/OCRTests/images`

[2](Jiwer Usage)`https://jitsi.github.io/jiwer/usage/`

en toda la batería de prueba divido entre el número total de caracteres de toda la batería de prueba.

Los resultados de los OCRs son las siguientes:

Tabla 5.1: Resultado de CER de los OCR (Redondeado a 3 decimales)

Imagen	**Tesseract**	**EasyOCR**	**OCRopus**
1	0.000	0.000	0.000
2	0.063	0.281	0.406
3	0.538	0.212	0.538
4	0.043	0.109	0.978
5	0.289	0.086	0.719
6	0.025	0.025	0.700
7	0.750	0.000	1.417
8	0.935	0.652	0.870
9	0.721	0.256	0.744
10	1.000	0.857	8.286
11	0.050	0.150	2.550
12	0.352	0.055	0.834
13	0.230	0.337	0.881
CER medio	**0.384**	**0.232**	**1.456**

Tabla 5.2: Resultado de OCR en tiempo(ms)

Iteración	**Tesseract**	**EasyOCR**	**OCRopus**
1	5539	204625	86213
2	5580	223809	90172
3	5539	216701	88290
4	5720	238900	83278
5	5602	216791	87810
6	5571	204502	88789
Tiempo medio	**5591**	**217554**	**87425**

Como podemos ver en la tabla 5.1, todos los OCRs han podido reconocer de forma exacta la imagen más simple y que todos han caido en la imagén 10. En cuanto las otras imágenes podemos observar que unas lo reconoce mejor Tesseract y otras EasyOCR por lo que ambos producen buenos resultados, en cuanto OCRopus, sus resultados ya no son tan buenos en comparación con las otras dos por lo que queda descartado. El OCR puede reconocer más caracteres del texto esperado por lo que el CER supera a 1 en algunos casos, lo llamamos caracteres basura que se explicará con más detalle en la sección 5.2.1. Aunque EasyOCR obtiene un mejor CER medio en los resultados (0.23) que Tesseract (0.38), Tesseract es muchísimo más rápido que EasyOCR teniendo un 211936 milisegundo de adelanto que equivale a 3 minutos y 50 segundos aproximadamente como podemos observar en la tabla 5.2. Por tanto decidimos seguir en adelante con Tesseract.

5.2. Mejoras en el reconocimiento de texto en imágenes

Como podemos notar en los resultados del apartado anterior, la imagen 10 ha sido un reto para el CER de los OCRs, esto es debido a que el OCR reconoce como carácter geometrías del fondo y produce basura.Para solucionar esto, se plantea aplicar preprocesamiento a las imágenes y la limpieza del resultado usando distancia Levenshtein para mejorar la precisión del texto obtenido por el OCR disminuyendo el CER.

Para obtener más información sobre aquellos factores o características de las imágenes que puedan afectar al rendimiento y efectividad de la herramienta de OCR, se ha clasificado las imágenes en 5 categorías diferentes. La clasificación se ha hecho de forma subjetiva, de acuerdo a ciertas características de imágenes.

1. Fondos simples(F.Simple)(Figura 5.3). Imágenes donde la cantidad de geometrías del fondo son pocas o fáciles de reconocer(no se confunde con los caractéres).

Figura 5.3: Ejemplo imagen fondos simples

2. Fondos complejos(F.Complejo)(Figura 5.4). Imágenes donde la cantidad de geometrías del fondo son muchas o difíciles de reconocer(fácil de confundirse con los caractéres).

Figura 5.4: Ejemplo imagen fondos complejos

3. PixelArt(PixelArt)(Figura 5.5). Imágenes donde el fondo y las letras tiene un estilo pixelart. Como las imágenes son píxeladas(cuadrado de color que forman formas y texto), supone un reto para el OCR.

Figura 5.5: Ejemplo imagen PixelArt

4. Texto en bocadillos(TxtBoc)(Figura 5.6). Imágenes donde el texto a reconocer se situa en un bocadillo y que el color del bocadillo tiene un alto contraste con el fondo.

Figura 5.6: Ejemplo imagen de texto en bocadillo

5. Texto en bocadillos con poca diferenciación con el fondo(TxtBoc2)(Figura 5.7).Imágenes donde el texto a reconocer se situa en un bocadillo y que el color del bocadillo tiene un contraste medio o bajo con el fondo.

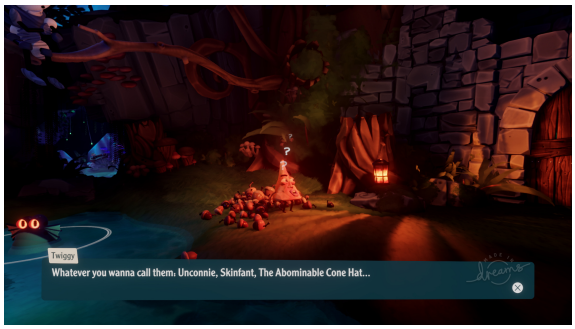

Figura 5.7: Ejemplo imagen de texto en bocadillos con poca diferenciación con el fondo

El preprocesamiento de imágenes consiste en aplicar técnicas que modifican las imágenes para que sea más fácil de reconocer por una OCR el texto existente. Para saber que técnicas mejoran el reconocimiento, se ha ido probando con cada una de ellas en el manual de OpenCV (2025), obteniendo los resultados e identificando aquellas técnicas que ayuda a mejorar el CER medio. Las técnicas probadas son las siguientes:

1. Escala de grises(Figura 5.8): Convierte una imagen a un formato de un solo canal, representando solo la intensidad de luz. Es útil para simplificar y reducir la cantidad de datos cuando el color no es relevante.

Figura 5.8: Imagen a escala de grises

2. Aumentar contraste(Figura 5.9): Mejora la diferencia entre las áreas claras y oscuras en una imagen, lo que facilita la detección de detalles.

Figura 5.9: Imagen con aumento de contraste

3. Ecualización del histograma(Figura 5.10): Ajusta el contraste de la imagen extendiendo la distribución de los niveles de gris, mejorando el rango dinámico y resaltando los detalles.

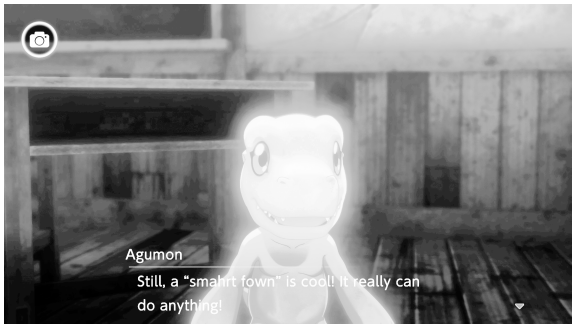

Figura 5.10: Imagen con ecualización de histograma

4. Corrección del gamma(Figura 5.11): Ajusta los valores de intensidad en la imagen para compensar la percepción humana y los errores del sensor, lo que puede hacer que ciertas áreas sean más visibles.

Figura 5.11: Imagen con corrección del gamma

5. Filtro de nitidez(Figura 5.12): Realza los bordes de una imagen para destacar detalles, útil en aplicaciones donde se requiere mayor definición.

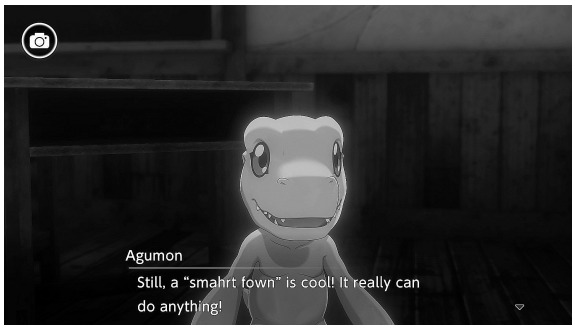

Figura 5.12: Imagen con filtro de nitidez

6. Adaptive Thresholding(Figura 5.13): Segmenta una imagen dividiéndola en áreas claras y oscuras, aplicando un umbral que se ajusta de forma adaptativa a las variaciones locales de luz.

Figura 5.13: Imagen aplicando adaptive thresholding

7. Simple Thresholding(Figura 5.14): Asigna un valor binario a cada píxel dependiendo de si está por encima o por debajo de un umbral específico, útil para crear máscaras y segmentación sencilla.

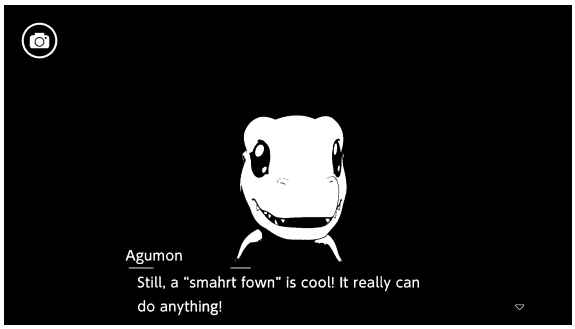

Figura 5.14: Imagen aplicando simple thresholding

8. Image Blurring (Desenfoque de Imagen)(Figura 5.15): Reduce el ruido y los detalles mediante técnicas como filtros Gaussianos o de promediado, comúnmente utilizado para suavizar imágenes antes de un análisis.

Figura 5.15: Imagen aplicando desenfoque de imagen

9. Redimensionar la imagen: Cambia las dimensiones de una imagen, lo que puede ser útil para normalizar entradas a una red neuronal o ajustar el tamaño de una imagen para procesamiento.

10. Dilatar y erosionar(Figura 5.16): Técnicas de morfología matemática que expanden o reducen las regiones blancas (o los objetos) en una imagen binaria, útiles para limpieza de ruido o cierre de contornos.

Figura 5.16: Imagen dilatado y erosionado

11. Denoising (Reducción de ruido)(Figura 5.17): Elimina o reduce el ruido en una imagen para mejorar la calidad visual y el rendimiento de tareas de reconocimiento.

Figura 5.17: Imagen aplicando reducción de ruido

Tipo	F.Complejo	F.Simple	PixelArt	TxTBoc	TxtBoc2
Nada	9.39	1.05	2.51	2.02	5.61
Grises	3.72	0.59	2.69	1.17	2.16
Contraste	5.06	0.81	2.39	1.13	10.07
Ecualización de histograma	5.59	3.60	6.56	2.41	8.67
Gamma	3.72	0.59	2.69	1.17	2.16
Filtro de nitidez	11.06	2.17	8.84	3.29	12.05
Thresholding C	16.80	9.44	10.36	4.08	23.44
Thresholding Gaussian	12.85	9.35	16.16	4.71	22.83
Thresold Binary	4.05	0.49	2.14	1.79	2.04
Redimension x1.5	4.77	0.80	2.86	1.27	3.07
Gaussian Blur	3.02	0.68	2.34	1.14	1.86
Median Blur	2.75	0.51	2.75	1.25	1.37
2 Blur	2.26	0.59	2.15	1.22	1.09
Dilatación y erosión	2.19	0.51	2.48	0.69	1.83
Dilatación	1.80	0.62	1.23	0.80	5.53
Erosión	2.02	1.00	2.57	0.82	1.21
Denoising	3.23	0.57	2.64	1.07	1.70

Tabla 5.3: Tabla con los resultados de CER medio de cada categoría de imágenes después de aplicar un tipo de preprocesamiento.

A continuación se ha hecho el experimento de aplicar un solo tipo de preprocesamiento a todas las imágenes de cada categoría, ejecutando el OCR sobre las imágenes preprocesadas y obteniendo el CER de cada imágen y el CER medio de cada categoría. Este proceso se ha aplicado para cada uno de las posibles técnicas de preprocesamiento obteniendo el resultado en la tabla 5.3. Cada columna representa la categoría de la imagen y cada fila representa el preprocesamiento aplicado a las imágenes. En este caso no se hace distinción de OCR ya que preprocesamiento se aplica directamente en las imágenes antes de ser pasados a la OCR por lo que no importa la librería de OCR que se este usando. Se marca en rojo aquel tipo de preprocesamiento que da peor resultado en la categoría y en verde, aquel que da mejor resultado.

Como podemos observar, en la mayoría de los preprocesamientos, los resultados mejoran en comparación con la de sin aplicar nada. Sin embargo, hay otros que empeoran, esto es debido a que el preprocesamiento ha marcado más las líneas y las geometrías por lo que el OCR reconozca más caracteres “basura”. Esto no significa que el OCR vaya ir a peor, puede que reconozca mejor el texto esperado pero añadiendo más basura que antes, algo que intentaremos resolver en el siguiente apartado.

Obteniendo esta tabla y viendo los resultados de imágenes se ha ido probando distintas combinaciones de preprocesados de imágenes:

1. Experimento 1:
    - Grises
    - Escalado
    - Adaptive Threshold
    - Denoising
    - Blurring
    - Dilate_Erode
2. Experimento 2:
    - Grises
    - Escalado
    - Denoising Blurring
    - Dilate_Erode
    - Ec. Histograma
    - Gamma
    - F.Nitidez
3. Experimento 3:
    - Grises
    - Escalado
    - Simple Threshold
    - Denoising
    - Blurring
    - F.Nitidez
    - Dilate_Erode
4. Experimento 4:
    - Grises
    - Escalado
    - Simple Threshold
    - Denoising
    - Blurring
    - Dilate_Erode

Obteniendo estos resultados:

Experimento	Complejo	Simple	PixelArt	TxTBoc	TxtBoc2
1	18.79	3.96	13.93	5.26	22.01
2	7.23	4.58	14.96	4.94	11.94
3	3.68	0.34	2.30	1.16	1.62
4	2.57	0.29	2.01	1.01	1.49

Tabla 5.4: Tabla con los resultados CER de cada categoría en cada experimento.

Donde podemos ver en la tabla 5.4 que el experimento más destacado es el experimento 4, por lo que en adelante seguiremos el preprocesamiento con las técnicas del experimento 4.

5.2.1. Eliminación de caracteres basura

Obteniendo el resultado de CER medio de los preprocesamientos(tabla 5.3), podemos ver que se produce muchos números que superan al 1, esto significa que el OCR ha reconocido más caracteres de lo que hay en el texto esperado, todos esos caracteres que sobran son caracteres basura y el texto que necesitaremos estará en algunas de esas líneas. En esta sección intentaremos eliminar esos caracteres basura dejando solo lo necesario.

Figura 5.18: Imagen ejemplo.

El texto esperado de la figura 5.18 es la siguiente:

```
Jonny
```

```
If you two hadn't met at the reunion,
you wouldn't be going out right now.
```

Y una posible salida de la OCR con la figura 5.18 puede ser la siguiente:

```
T A W
Wml"lll"|||"lll‘‘1]]‘‘11‘‘‘"1‘‘l‘l]1"]]]"]]1"1V‘‘‘"‘’’V"”””””””“WWHH\HH\HH
G
wiui:lfifl}m;,;r}l’‘‘
Nul1l1lmu11111um111llumI11llumIIIIIIMMI1IllMmlllIIIUU1
Hunum;.x:flwww’M;‘dm‘w‘w‘w‘w‘uHfl‘‘fl‘‘flflmw e 4 ‘ w}\‘s.*"‘r Ll e e p
: L T Q1 A it e
- ’ diiry el R ;4
o wm W el 5
i " :_.‘‘;»"*‘ i i m?‘‘ Q:‘]m}illl}jjw ! ‘‘hu‘11111111{}}‘‘
T i 1‘11‘‘1’”””“”””“““’’ HHHHHHH ‘HHU’”‘HH\HHHHH\HMH\H ‘ (TR
I e > h ¢ 4 ’ i
= 4 o
T
- i e
§ ...
E ‘E | \ i
P S y:
,,|,- |
IR ill ‘
- ’ A 000 Ir yo
```

Donde de forma subjetiva no encontramos ningún trozo del texto que se corresponda o sea parecido al ground-truth.

Después de aplicar el preprocesamiento elegido en el apartado anterior obtenemos el siguiente resultado:

```
T A W
Wml"lll"|||"lll‘‘1]]‘‘11‘‘‘"1‘‘l‘l]1"]]]"]]1"1V‘‘‘"‘’’V"”””””””“WWHH\H
dJonny If you two hadn't met at the reunion
you wouldn't be going out right now
,,|,- |
IR ill ‘
- ’ A 000 Ir yo
```

En esta salida ya podemos reconocer que hay dos líneas que se corresponde con la salida que esperamos. La frase que realmente importa es solamente las dos líneas de la salida.

```
Jonny If you two hadn't met at the reunion
you wouldn't be going out right now
```

Esto es un problema importante que debemos solucionar ya que es imposible saber si un test es correcto o no con esta entrada.

En esta sección se propone una solución a este problema de caracteres "basura" usando el algoritmo de distancia *levenshtein*.

La distancia de Levenshtein(también conocida como distancia de edición) según un articulo de Greenhill (2011) se define como una métrica que mide el número mínimo de operaciones necesarias para transformar una cadena en otra, utilizando tres tipos de operaciones básicas:

- Inserciones: Agregar un carácter.
- Eliminaciones: Eliminar un carácter.
- Sustituciones: Reemplazar un carácter por otro.

Suponiendo que tenemos el texto esperado de la imagen, utilizando esta métrica, podemos obtener la distancia levenshtein entre una línea del texto esperado y una línea del texto reconocido por la OCR. Con la distancia obtenida podemos calcular la similitud de las cadenas y aplicando un cierto umbral, podemos identificar aquellas líneas que más se asimila al texto esperado, obteniendo así las líneas deseadas y descartando aquellas que no cumpla un cierto umbral.

La fórmula general para calcular la similitud es:

$Simulitud = 1 - \frac{d}{max(s1,s2)}$

Donde:

d es la distancia levenshtein.

max(s1,s2) es el máximo entre la longitud de la cadena *s1* y cadena *s2*

Uno de los resultados obtenidos aplicando la distancia levenshtein es la siguiente si aplicamos un umbral de similitud de 0.8 (tiene que ser 80 % de parecido):

- Texto real de OCR:

```
"
|
\ J
-
Agumon /
Still, a ‘‘smahrt fown’’ is cool! It really can
do anything! <
```

- Texto esperado:

```
Agumon
Still,a "smahrt fown" is cool! It really can
do anything!
```

El algoritmo de limpieza sigue los siguientes pasos:

1. Se compara la primera línea obtenida por el OCR con la primera línea del texto esperado.
2. Se calcula la distancia de Levenshtein entre ambas cadenas.
3. A partir de la distancia de Levenshtein y la longitud de las cadenas, se calcula el valor de similitud.
4. Se verifica si la similitud supera el umbral previamente definido.
5. Si la similitud es superior al umbral, se marca la línea como válida. En caso contrario, se continúa con la siguiente línea del texto esperado.
6. Este proceso se repite hasta haber comparado la línea del OCR con todas las líneas del texto esperado. Si durante la iteración se encuentra una línea con una similitud mayor que la obtenida hasta el momento, se actualiza la línea marcada con esta nueva coincidencia.
7. El mismo procedimiento se repite para la segunda línea del OCR y para el resto de líneas subsiguientes.

Después de todo ese proceso obtenemos la cadena limpiada utilizando distancia levenshtein.

- Tras aplicar este algoritmo, la cadena de OCR resultante será la siguiente:

```
Agumon /
Still "smahrt fown" is cool! It really can
do anything! <
```

Aplicando el algoritmo a los resultados de todas las imágenes de cada categoría mencionadas en el apartado anterior después de aplicar preprocesamiento obtenemos estos nuevos resultados del CER medio de cada categoría:

Tipo	Complejo	Simple	PixelArt	TxTBoc	TxtBoc2
OCR	2.57	0.29	2.01	1.01	1.49
Levenshtein	0.63	0.16	0.63	0.55	0.42

El resultado de OCR es el resultado que obtenemos con solamente aplicando el preprocesamiento del apartado anterior. Sobre esa salida, se aplica la distancia levenshtein para la eliminación de basura obteniendo los nuevos resultados. Podemos ver en los números que mejoran casi un 50 % en todas la categorías por lo que es una técnica viable para nuestra herramienta.

5.3. Evaluación de los tests

El objetivo de esta evaluación es asegurar de que los tests implementados sean correctos sin producir ningún falso positivo y falso negativo. La metodología de la evaluación es utilizando test de unidad donde existirá una suite de test para casos positivos de cada test y casos negativos de cada test.

5.3.1. Test de placeholders

Se define en este test un delimitador de placeholder indicado su cadena de apertura y su cadena de cierre. Se comprueba en los test de casos positivos que el texto no tenga ningún placeholder. Se comprueba en los test de casos negativos que reconozca el placeholder del texto en cualquier posición. P.ej se define la cadena de apertura de placeholder "%_" y la de cierre "_%" y se obtiene el resultado de la tabla 5.5 indicando positivo que pasa el test(no hay placeholder) y negativo que no pasa el test de placeholder(hay placeholder).

Texto	Resultado
Sin placeholder	Positivo
%Sin placeholder%	Positivo
%_Placeholders_%	Negativo
hola%_Placeholders_%	Negativo

Tabla 5.5: Ejemplo de algunos test de unidad de placeholder

5.3.2. Test de solapamiento

Para este test, se ha creado imágenes simples donde existen solapamiento de texto y otras en las cuales no hay. El suite test positivo verifica que no existen solapamiento por lo que pasa el test y el suite negativo verifica que existe solapamiento por lo que no lo pasa. Las imágenes positivas y negativas tienen un aspecto similar a la figura 5.19 y la figura 5.20 .

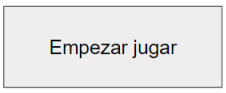

Figura 5.19: Ejemplo de imagen positivo en solapamiento.

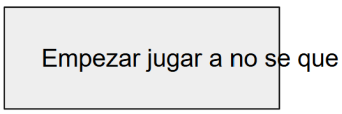

Figura 5.20: Ejemplo de imagen negativo en solapamiento.

5.3.3. Test de truncamiento

En este test, para el suite de test positivos, se define la cadena del texto esperado y el texto reconocido de forma simétrica comprobando que pasa el test de truncamiento. En el suite de negativos, se define la cadena del texto esperado, y las cadenas del texto reconocido son subcadenas del texto esperado.P.ej :

- Texto esperado: Empezar jugar
- Texto reconocido: Empezar jug

5.3.4. Resultado del test de unidad

Se ejecutan:

- Placeholder: 8 casos positivos y 5 casos negativos.
- Solapamiento: 2 casos positivos y 2 casos negativos.
- Truncamiento: 2 casos positivos y 3 casos negativos.

Ejecutando el test de unidad obtenemos que se ha pasado todos los tests, tanto positivos como negativos por lo que podemos llegar a la conclusión de que la implementación de los tests son correctas.

5.4. Evaluación de la herramienta

En esta sección se hará una evaluación de la herramienta comprobando el funcionamiento completo incluyendo los dos modulos de OCR y tests. El objetivo de esta evaluación es asegurar de que nuestra herramienta es preciso y exacto como ayuda de la parte de LQA. La metodología de la evaluación es la siguiente:

1. Se ha creado una batería de pruebas con 30 imágenes en total. Entre las 30 imágenes, existen
   - 7 errores de solapamiento
   - 5 errores de truncamiento
   - 5 errores de placeholders

   Las imágenes de prueba con errores de localización se ha hecho editando sobre imágenes de los juegos sobrescribiendo sobre el contenido original.
2. Se ejecutará la herramienta propocionando información de estas imagenes y configuración.
3. Se obtendrá los resultados de las imágenes.
4. Se generará una matriz de confusión con los resultados obtenidos y esperados.
5. Se obtendrá información de la matriz de confusión como precisión, falsos positivos y falsos negativos.

En la matriz de confusión:

- Positivo representa que existe un error de localización.
- Negativo representa que no existe error de localización.
- El valor esperado es el valor que realmente es.

- El valor real es el valor de la herramienta.
- Verdadero positivo(TP) indica que hay error y detecta error.
- Falso positivo(FP) indica que no hay error pero detecta error.
- Verdadero negativo(TN) indica que no hay error y no detecta error.
- Falso negativo(FN) indica que hay error pero no detecta error.
- Exactitud mide el porcentaje de predicciones correctas sobre el total de predicciones. Indica el porcentaje de acierto sobre si existe o no errores de localización en las imágenes.

  $Exactitud = \frac{TP+TN}{TP+TN+FP+FN}$
- Precisión mide el porcentaje de predicciones positivas correctas sobre el total de predicciones positivas. Indica el porcentaje de acierto de detección de errores de localización.

  $Precision = \frac{TP}{TP+FP}$

el aspecto de la matriz es similar a la tabla 5.6

		Esperado	
		Positivo	Negativo
Real	Positivo	TP	FP
	Negativo	FN	TN
	Exactitud		
	Precisión		

Tabla 5.6: Matriz de confusión ejemplo.

		Esperado	
		Positivo	Negativo
Real	Positivo	14	2
	Negativo	12	2
	Exactitud	53.33	
	Precisión	87.5	

Tabla 5.7: Matriz de confusión del resultado.

Como podemos observar en la tabla 5.7 se producen 12 falsos negativos(se espera que detecte errores de localización pero la herramienta no lo detecta) y 2 falsos positivos(se esperaba que no exista ningún error de localización pero la herramienta los detecta como error). La cantidad de falsos positivos indica que la herramienta

está detectando errores en imágenes donde no debería aparecer errores, esto es un problema para nuestra herramienta ya que nuestro fin es minimizar trabajo de un ser humano y este tipo de error aumenta el trabajo teniendo que evaluar las imágenes cuando en realidad no tiene ningún error. Y los falsos negativos indica que la herramienta no detecta ningún error en la imagen cuando en realidad existen errores lo que también es un problema porque la herramienta trata de reconocer errores para ser corregidos antes de la publicación del juego. Podemos observar que la precisión de la herramienta es alta(87.5 %) pero la exactitud es mediana(53.33 %) por lo cual la herramienta puede estar detectando errores de localización pero mucha de las veces falla en clasificar si una imagen tiene o no error de localización.

Para obtener más detalle de la evaluación, generaremos una matriz de confusión para cada tipo de test.

		Esperado	
		Positivo	Negativo
Real	Positivo	3	4
	Negativo	7	16
	Exactitud	63.33	
	Precisión	42.86	

Tabla 5.8: Matriz de confusión del resultado de solapamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	5	20
	Negativo	0	5
	Exactitud	33.33	
	Precisión	20	

Tabla 5.9: Matriz de confusión del resultado de truncamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	0	0
	Negativo	5	25
	Exactitud	83.33	
	Precisión	0	

Tabla 5.10: Matriz de confusión del resultado de placeholders.

Mirando los resultados de cada test, podemos observar en la tabla 5.8, se producen 11 errores(falso positivo + falso negativo) al detectar solapamiento lo cual ha

conseguido acertar en 19 imágenes(63.33 % de exactitud) y que de los 7 errores que había, ha conseguido detectar 3(42.86 % de precisión). Sin embargo en las otras dos existen errores graves. En la tabla 5.9 podemos observar que se detectan 20 casos de falsos negativos que es un valor bastante alto lo que baja la exactitud a un 33.33 %. En la tabla 5.10 observamos que se produce 25 casos de verdaderos negativos y 5 casos de falsos positivos, aunque parezca que tiene buenos resultados, recordamos que existe 5 errores de placeholders solamente, lo cual significa que no ha conseguido detectar ninguno de ellos(0 % de precisión). Lo que tienen en común estos últimos dos test es que usan el texto reconocido en la salida del OCR lo cual el problema puede encontrarse allí.

Observando el informe los resultados de salida, de las 30 imágenes de prueba, el porcentaje de similitud de 17 imágenes son menores del 50 % y entre ellas, 14 son 0 %, esto significa que no se esta reconociendo de forma correcta el texto de las imágenes produciendo todo ese error en los tests. Por tanto, aunque en la tabla 5.9 obtenga una precisión de 100 %, esto puede ser debido al mal reconocimiento de texto produciendo un error de truncamiento y que "justamente" en esas imágenes existen error de truncamiento.

Haciendo una investigación del problema, se detecta que el problema puede ser producido en el proceso de preprocesamiento, por lo cual mostramos por salida el resultado de la imagen después del preprocesamiento de la figura 5.21 (imagen original) obteniendo el resultado como se muestra en la figura 5.22.

Figura 5.21: Imagen ejemplo de prueba.

Figura 5.22: Resultado de la imagen de prueba después de preprocesamiento.

Podemos ver que la imagen preprocesada se ve borroso en la parte del texto incluso a ojo humano, por tanto es de lo esperado que el OCR no consiga reconocer el texto de la imagen. Este problema es debido a que en la sección 5.2, la evaluación del CER de los preprocesamientos se ha hecho sin tener en cuenta la distancia levenshtein que fue implementado y evaluado posteriormente del preprocesamiento. El bajo CER que se obtuvo es debido a que el OCR reconocía menos caracteres basura pero sin reconocer el texto. Puede darse el caso de que el OCR reconozca el texto pero con mucha basura, lo que aumenta el CER, algo que se soluciona aplicando la distancia levenshtein.

Para intentar resolver este problema, evaluaremos la herramienta quitando los preprocesamientos que hagan que la imagen se vea más borroso que son:

- Denoising
- Blurring
- Dilatar y erosionar.

Figura 5.23: Resultado de la imagen de prueba después del nuevo preprocesamiento.

Se obtiene la imagen que se muestra en la figura 5.23 y los siguientes nuevos resultados.

		Esperado	
		Positivo	Negativo
Real	Positivo	15	1
	Negativo	12	2
	Exactitud	56.67	
	Precisión	93.75	

Tabla 5.11: Matriz de confusión del resultado.

		Esperado	
		Positivo	Negativo
Real	Positivo	3	4
	Negativo	7	16
	Exactitud	63.33	
	Precisión	42.86	

Tabla 5.12: Matriz de confusión del resultado de solapamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	5	20
	Negativo	0	5
	Exactitud	33.33	
	Precisión	20	

Tabla 5.13: Matriz de confusión del resultado de truncamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	1	0
	Negativo	4	25
	Exactitud	86.67	
	Precisión	20	

Tabla 5.14: Matriz de confusión del resultado de placeholders.

Vemos que no ha mejorado mucho en la matriz de confusión, sin embargo si tenemos en cuenta el porcentaje de similitud como se observa en la tabla 5.15, sí que se ha mejorado de forma genérica teniendo 15 imágenes por debajo del 50 % y entre ellas 9 imágenes de 0 %.

También observamos que en algunos casos ha empeorado, como puede ser la imagen 1 que tenía un 85 % y que ahora está a 0 %.

Imagen	Antes	Después
1	85.71	0
2	0	0
3	0	0
4	0	100
5	0	70.32
6	0	70
7	0	81.63
8	91	60.60
9	86.66	86.66
10	53.84	57.69
11	75.67	59.35
12	10.09	35.77
13	54.92	54.92
14	0	0
15	0	0
16	0	0
17	0	0
18	100	100
19	0	0
20	76.92	76.92
21	54.05	51.35
22	0	26.60
23	0	35.77
24	10	34.86
25	69	47.88
26	67	54.92
27	0	0
28	96	96
29	30	30.30
30	96.42	94.64

Tabla 5.15: Comparación de porcentaje de similitud entre los dos preprocesamientos.

Con esto podemos llegar a la conclusión de que es necesario diferente preprocesamiento dependiendo de las características de las imágenes. Para ello, se intentará buscar el mejor preprocesamiento para cada imagen. Se ejecutará los siguientes tipos de preprocesamiento:

- El preprocesado original de la sección 5.2.
- El preprocesado mejorado de esta sección.
- Escala de grises y ajuste adaptativo de contraste.

- Escala de grises y simple thresholding.

- Escala de grises solamente.

Se aplican distintos métodos de preprocesado teniendo en cuenta que, en algunas imágenes, dicho preprocesado puede empeorar la calidad del reconocimiento, mientras que en otras resulta imprescindible para mejorar los resultados. Por ello, para cada imagen se evaluarán las diferentes opciones de preprocesado y se seleccionará aquella que obtenga el mayor porcentaje de similitud en el reconocimiento del texto. Se obtiene los siguientes nuevos resultados:

		Esperado	
		Positivo	Negativo
Real	Positivo	15	1
	Negativo	10	4
	Exactitud	63.33	
	Precisión	93.75	

Tabla 5.16: Matriz de confusión del resultado.

		Esperado	
		Positivo	Negativo
Real	Positivo	4	3
	Negativo	6	17
	Exactitud	70	
	Precisión	57.14	

Tabla 5.17: Matriz de confusión del resultado de solapamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	5	17
	Negativo	0	8
	Exactitud	43.33	
	Precisión	27.77	

Tabla 5.18: Matriz de confusión del resultado de truncamiento.

		Esperado	
		Positivo	Negativo
Real	Positivo	1	0
	Negativo	4	25
	Exactitud	86.67	
	Precisión	20	

Tabla 5.19: Matriz de confusión del resultado de placeholders.

Imagen	Antes	Mejor de cada imagen.
1	85.71	86
2	0	70
3	0	67
4	0	100
5	0	91
6	0	70
7	0	89
8	91	91
9	86.66	86.66
10	53.84	65
11	75.67	75.67
12	10.09	38
13	54.92	83
14	0	8
15	0	73
16	0	88
17	0	76
18	100	100
19	0	65
20	76.92	76.92
21	54.05	70
22	0	29
23	0	38
24	10	34.86
25	69	76
26	67	63
27	0	45
28	96	96
29	30	79
30	96.42	96.42

Tabla 5.20: Comparación de porcentaje de similitud entre el preprocesado original y los mejores de cada imagen.

Como podemos observar en las tablas 5.16, 5.17 y 5.18, se vuelve a mejorar un

poco los resultados en cuanto la exactitud. El porcentaje de similitud en este caso ya no hay ningún valor a 0 % y solamente hay 6 imágenes con valores inferiores al 50 % como podemos ver en la tabla 5.20. Este proceso de preprocesamiento específico para cada imagen no es trivial por lo que es necesario en un futuro obtener características de la imágenes y aplicar determinadas técnicas de preprocesamiento según las características.

Como se puede observar en las Tablas 5.16, 5.17 y 5.18, se aprecia una ligera mejora en los resultados de exactitud. En cuanto al porcentaje de similitud, ya no se presentan valores del 0 %, y únicamente seis imágenes registran valores inferiores al 50 %, como se detalla en la Tabla 5.20. Este resultado indica que el uso de un preprocesamiento específico por imagen contribuye a mejorar el rendimiento del sistema. Sin embargo, este proceso no es trivial, por lo que en trabajos futuros sería recomendable extraer características relevantes de cada imagen y aplicar técnicas de preprocesamiento adaptadas a dichas características.

En conclusión, la precisión y exactitud de la herramienta se sitúan en un nivel intermedio y son claramente mejorables. Una de las principales limitaciones es que el OCR no logra reconocer el texto de forma completamente precisa, lo que sugiere que es necesario optimizar el preprocesamiento para mejorar el rendimiento del reconocimiento. Una posible línea de mejora sería el entrenamiento del modelo OCR con fuentes específicas del videojuego, ya que hasta ahora se ha utilizado únicamente el modelo por defecto de Tesseract.

5.5. Conclusión

En este capítulo se ha llevado a cabo la evaluación del módulo de OCR, del módulo de tests y, finalmente, de la herramienta en su conjunto. En la evaluación del módulo de OCR, Tesseract ha demostrado ser la opción más adecuada para el caso de uso planteado. Con el fin de mejorar el reconocimiento de texto, se han aplicado técnicas como el preprocesamiento de imágenes, que facilita la detección de texto, y el uso de la distancia de Levenshtein para medir la similitud entre cadenas, lo cual permite filtrar caracteres irrelevantes o erróneos.

Para evaluar el módulo de tests se ha desarrollado una prueba unitaria específica, cuyos resultados han sido satisfactorios, demostrando el correcto funcionamiento del sistema de validación.

En cuanto a la evaluación global de la herramienta, los resultados obtenidos reflejan un rendimiento aceptable pero claramente mejorable. La principal limitación detectada radica en el reconocimiento textual por parte del OCR. Para mitigar este problema, se propone como línea de mejora futura la adaptación del preprocesamiento en función de las características de cada imagen, así como el entrenamiento de un modelo OCR específico para la fuente utilizada en el videojuego, en lugar de depender exclusivamente del modelo por defecto.

Capítulo 6

Conclusiones y Trabajo Futuro

Debido a la gran evolución en la industria de los videojuegos, surge la necesidad de adaptar un videojuego a otro idioma/región diferente del que fue creado. Para ello, se siguen los procesos de internacionalización y localización, donde la internacionalización es la realización de soporte para diferentes idiomas y la localización es el proceso la cual se traducen y se adaptan los textos, gráficos, etc.

Para asegurar de que el videojuego está preparado para ser publicado en otras regiones, entra el *Localization Quality Assurance* (o LQA) para asegurar de que no exista ningún error de localización.

Para facilitar el trabajo de LQA, se ha diseñado una herramienta donde el usuario proporciona unas imágenes del juego y una configuración, la herramienta detecte si existe algún error de localización. Esta herramienta constará de dos partes, modulo de OCR y modulo de tests.

Para el modulo de OCR, se evalua diferentes librerías de OCR y para nuestra herramienta, utilizaremos Tesseract. Para mejorar los resultados de reconocimiento, se aplica técnicas como preprocesamiento de imágenes para mejor identificación de texto en imágenes y técnicas como la distancia levenshtein para eliminar caracteres basuras reconocidas por el OCR.

Para el modulo de tests, que diseña y se implementa tests que puedan detectar errores de localización, para nuestro caso, se ha hecho el test de solapamiento de texto, truncamiento de texto y detección de placeholders.

La herramienta generará un archivos con los resultados que será luego interpretado generando así un informe.

Los resultados de la evaluación de la herramienta han sido aceptables, aunque claramente mejorables. La principal limitación observada reside en el reconocimiento de texto por parte del OCR, especialmente en imágenes complejas como las que se generan en un entorno de videojuego.

Para mejorar la herramienta se propone los siguiente trabajos futuros:

1. Clasificación de imágenes para detectar el preprocesamiento necesario según

las características de las imágenes.

2. Árbol de decisión que configure un determinado preprocesamiento según las características de las imágenes.
3. Entreno del modelo con la fuente específica.
4. Integración de otras librerías de OCR como opción al usuario.
5. Implementación de test que solucione otros problemas de localización.
6. Detección de layout en las imágenes para reconocer las zonas de texto y hacer un recorte para eliminar zonas sin texto y mejorar el reconocimiento.
7. Herramienta que recopile dato de pantalla de videojuego y generar los assets necesarios para la herramienta de detección de errores de localización.

Introduction

Nowadays, due to the large number of video game users in different regions, there arises the need to adapt video games to regions or countries other than the original. To achieve this, there are a series of steps or strategies that can be followed. In this work, we propose the design of a tool that supports this process.

6.1. Motivation

Over the years, video games have become one of the largest entertainment industries in the world. According to a report by Newzoo (2024), in 2024 the global video game market reached an estimated revenue of 187.7 billion dollars, as shown in Figure 6.1. The same report shows the growth in the number of players (Figure 6.2), from 3.14 billion in 2022 to 3.422 billion in 2024, with a prediction of 3.759 billion by 2027.

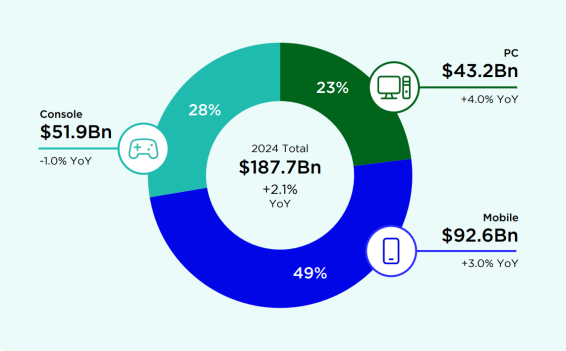

Figure 6.1: Estimated revenue in the video game industry in 2024. Newzoo

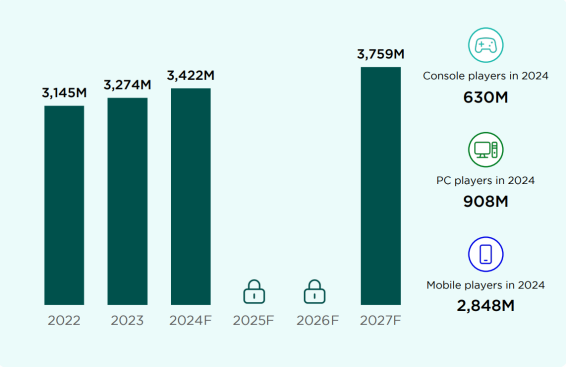

Figure 6.2: Global players 2022–2024 and projection for 2027. Newzoo

Additionally, the same report mentions the percentage of players by region, with the Asia-Pacific region representing more than half (53%) of global players, as shown in Figure 6.3. Given this growth and these figures, there is a need to adapt video games to different languages and cultures through the processes of internationalization (*I18N*) and localization (*L10N*), in order to be published in various regions and countries.

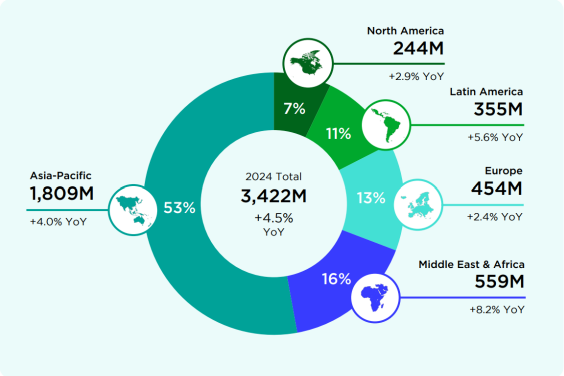

Figure 6.3: Percentage of players by region. Newzoo

Internationalization is the process by which a video game is prepared from its early development stages to support different cultures and languages, so that no major modifications to the code are needed later. On the other hand, localization is the process of translating and adapting the texts, graphics, and resources of a video game to the various cultures in which it will be released.

During both of these processes, various types of errors may occur (such as translation or display errors). This is where *Localization Quality Assurance* (LQA) comes into play. This process involves reviewing and testing different parts of the game to ensure that none of these errors occur and that the final product is appropriate for the target culture and audience.

Until now, this process has been carried out manually, where testers meticulously check each piece of text and how it is displayed within the game, which requires a significant amount of time and resources. While there are tools to automatically

check the correctness of translations and texts, the same cannot be said for verifying how texts are displayed within the context of the game.

Based on the points discussed above, our motivation for this work is to automate testing of internationalization and localization processes in order to save time and resources, which can then be redirected to other more critical areas. To achieve this, we will use computer vision techniques and *Optical Character Recognition* (OCR) so that, given a screenshot of a video game, we can extract the text and perform various checks and tests on it.

6.2. Objectives

The main objective of this project is the design and implementation of a tool to assist in the automation of localization QA. To achieve this main goal, the following secondary objectives are proposed:

1. Research the LQA process and the tools used, as well as the most common errors in internationalization and localization.
2. Investigate OCR technologies and how to use them, in order to choose one that fits our requirements by conducting tests and comparisons.
3. Design a series of OCR-based automated tests to detect common linguistic errors.
4. Deploy the tool using a Docker image so that it can be run on any machine.
5. Integrate the OCR to use its output as input for the tests, and improve the OCR's accuracy as much as possible.
6. Evaluate our tool to ensure it produces an acceptable accuracy rate.

6.3. Work Plan

To achieve our objectives, we will follow the steps below:

1. Research and understand how the internationalization, localization, and LQA processes work, and identify the most common linguistic errors that occur when developing a video game (Chapter 2).
2. Research different OCRs, understand how they work, how to use them to detect text in images, and how to train a model given the language and font used (Chapter 2).
3. Select localization errors that are related to character or positioning issues, which can be detected using OCR (Chapter 3).

4. Design the tool in detail, describing the tests, OCR usage, and output of the tool (Chapter 3).

5. Develop a series of tests capable of detecting whether a specific localization error exists, based on input from the OCR (Chapter 4).

6. Integrate the selected OCR library and other auxiliary libraries and techniques used by the tool (Chapter 4).

7. Evaluate the selected OCRs based on their accuracy and processing time (Chapter 5).

8. Perform separate evaluations of each module (OCR, tests) (Chapter 5).

9. Integrate both modules and conduct a full evaluation of the tool (Chapter 5).

Conclusions and Future Work

Due to the significant growth of the video game industry, there is an increasing need to adapt games to languages and regions different from their original design. This is achieved through the processes of internationalization and localization, where internationalization refers to adding support for multiple languages, and localization is the process of translating and adapting texts, graphics, and other assets.

To ensure that a video game is ready to be released in other regions, the Localization Quality Assurance (LQA) process is applied. LQA is responsible for verifying that there are no localization-related errors.

To assist with the LQA process, a tool has been developed that allows users to provide screenshots from the game along with a configuration file. The tool will then detect whether any localization errors are present. This tool consists of two main modules: the OCR module and the testing module.

For the OCR module, several OCR libraries were evaluated, and Tesseract was selected for use in the tool. To improve text recognition accuracy, techniques such as image preprocessing are applied to enhance text clarity in images. Additionally, methods like the Levenshtein distance are used to filter out noisy characters incorrectly recognized by the OCR.

For the testing module, tests were designed and implemented to detect localization issues. In this project, the following tests were developed: text overlap detection, text truncation detection, and placeholder detection.

The tool generates a result file, which is later interpreted to produce a final report.

The evaluation results of the tool were acceptable, although there is clear room for improvement. The main limitation observed lies in the OCR's ability to recognize text, especially in complex images such as those generated within video game environments.

To enhance the tool, the following future work is proposed:

1. Image classification to determine the most appropriate preprocessing based on image characteristics.
2. A decision tree to apply specific preprocessing steps depending on the input

image's features.

3. Training the OCR model with the game's specific font.
4. Integration of alternative OCR libraries to provide users with more options.
5. Implementation of additional tests to detect other types of localization issues.
6. Layout detection in images to identify text areas and crop out irrelevant zones, improving recognition accuracy.
7. A tool to automatically collect game screen data and generate the necessary assets for the localization error detection tool.

Bibliografía

Codificacion. https://conceptos.es/codificacion, 2025.

Arasa, C. G. ¿que son las expresiones regulares?(regex). https://www.viewnext.com/que-son-las-expresiones-regulares-regex/, 2022.

AWS. ¿qué es el reconocimiento óptico de caracteres (ocr)? https://aws.amazon.com/es/what-is/ocr/, 2024.

Belcic, I. What is localization quality assurance (lqa)?? https://www.gridly.com/blog/what-is-localization-quality-assurance-lqa/, 2022.

Chatterjee, A. Diferencias entre internacionalización y localización (i18n y l10n). https://www.multilocale.com/blog/es/diferencias-entre-internacionalizacion-y-localizacion-i18n-y-l10n, 2023.

Greenhill, S. J. Levenshtein distances fail to identify language relationships accurately. https://doi.org/10.1162/COLI_a_00073, 2011.

Hashemi-Pour, C. y Alexander-S-Gillis. What is quality assurance (qa)? https://www.techtarget.com/searchsoftwarequality/definition/quality-assurance, 2024.

Muñóz Sánchez, P. *Localización de Videojuegos*. Editorial Síntesis, 2017.

Newzoo. Newzoo's global games market report 2024. 2024.

OcropusDoc. Extracting text from an image using ocropus. https://www.danvk.org/2015/01/09/extracting-text-from-an-image-using-ocropus.html, 2015.

OpenCV. Image processing in opencv. https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html, 2025.

Robayo, J. D. R. *Testing Automatizado para la localización en videojuegos*. Proyecto de investigación, Universidad Complutense, 2023. Disponible en https://github.com/JosedaMachine/LocalizationTesting-CollabProject.

TesseractDoc. Improving the quality of the output. https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html, 2023.