Desarrollo de una herramienta colaborativa con entorno físico interactivo y visualización 3D

Development of a collaborative tool with an interactive physical environment and 3D visualization

---

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela Politécnica Superior de Zamora

Resumen

En el ámbito de la interacción humano-computador (HCI), las herramientas digitales predominantes (ratón, teclado, pantallas táctiles, etc.) limitan la interacción física y la colaboración espontánea entre los usuarios. Existen pocas soluciones que integren la manipulación tangible de objetos con la visualización digital de forma accesible, reutilizable y adaptable a diferentes contextos.

Para dar respuesta a esta necesidad, este Trabajo de Fin de Grado presenta el desarrollo de ChromaVision, una herramienta modular basada en visión artificial, que permite detectar piezas físicas por color y posición, y visualizarlas en un motor de videojuegos de forma sincronizada. El color de cada pieza actúa como un identificador que permite sustituirla por el tipo de objeto que el usuario desee en el motor.

El sistema se compone de dos módulos independientes que se comunican mediante el protocolo UDP. El módulo de detección está implementado en Python con la librería OpenCV. Captura las imágenes de una webcam estándar, localiza la base, identifica las piezas por su color y calcula su posición mediante una cuadrícula ajustable. A continuación, envía los datos de las piezas (color y posición) por UDP en formato JSON.

El módulo de visualización está desarrollado en C++ para Unreal Engine. Recibe e interpreta la información del módulo de detección. Luego instancia los actores 3D que el usuario ha asignado a cada color, situándolos en las posiciones donde se han detectado las piezas. La herramienta es configurable mediante un archivo JSON y en el editor del propio motor. Además, su diseño modular permite adaptarla a otros motores de videojuegos.

Como demostración de su versatilidad, en este TFG se ha desarrollado una aplicación de prototipado de niveles, alternando entre modos de construcción y juego. No obstante, ChromaVision es una herramienta flexible que permite crear aplicaciones colaborativas basadas en la manipulación de objetos tangibles, con potencial en otros ámbitos, como la educación, la arquitectura, el diseño de videojuegos, etc.

El sistema emplea hardware estándar que lo hace accesible: un ordenador, una webcam, una base y piezas LEGO. Por último, los resultados obtenidos demuestran que la herramienta cumple los objetivos funcionales planteados.

Palabras clave: colaboración, OpenCV, prototipado, TUI, Unreal Engine, visión artificial

Abstract

In the field of human-computer interaction (HCI), the predominant digital tools (mouse, keyboard, touch screens, etc.) limit physical interaction and spontaneous collaboration among users. Few solutions integrate the tangible manipulation of objects with digital visualization in an accessible, reusable, and adaptable way for different contexts.

To address this need, this Bachelor's Thesis presents the development of ChromaVision, a modular tool based on computer vision, which detects physical pieces by colour and position and visualises them synchronously in a game engine. The colour of each piece acts as an identifier, allowing it to be replaced by any type of object the user desires in the engine.

The system consists of two independent modules that communicate using the UDP protocol. The detection module is implemented in Python with the OpenCV library. It captures images from a standard webcam, locates the base, identifies pieces by their colour, and calculates their position using an adjustable grid. It then sends the piece data (colour and position) via UDP in JSON format.

The visualization module is developed in C++ for Unreal Engine. It receives and interprets the information from the detection module. It then instantiates the 3D actors assigned by the user to each colour, placing them at the positions where the pieces have been detected. The tool is configurable through a JSON file and within the engine's editor. Moreover, its modular design allows adaptation to other game engines.

As a demonstration of its versatility, this thesis has developed a level-prototyping application, switching between construction and game modes. Nevertheless, ChromaVision is a flexible tool that enables the creation of collaborative applications based on the manipulation of tangible objects, with potential in other fields such as education, architecture, video game design, etc.

The system uses standard hardware, making it accessible: a computer, a webcam, a base, and LEGO pieces. Finally, the results obtained demonstrate that the tool meets the established functional objectives.

Keywords: collaboration, computer vision, OpenCV, prototyping, TUI, Unreal Engine.

Glosario

K-Means Algoritmo de agrupamiento (clustering) que divide datos en k grupos según su similitud. 41, 42

Saliencia Característica de un estímulo que lo hace destacar sobre los demás y llamar la atención. 6

Studs Los pequeños salientes cilíndricos de las piezas LEGO que permiten encajarlas. 43

Transformada de Hough Técnica de procesamiento de imágenes para detectar líneas rectas a partir de bordes, representándolas en coordenadas polares. 40–42, 44, 47, 51

Usabilidad Se refiere a la eficiencia, satisfacción y facilidad con la que el usuario puede alcanzar su objetivo. 7

Visión artificial Campo de la inteligencia artificial que permite a las computadoras interpretar y procesar imágenes y vídeos. 2, 9, 11–14, 19, 20, 48, 50, 51

Visión por computador Véase visión artificial. 3, 15, 23, 24

Siglas

3D tres dimensiones. 1–4, 9, 10, 13–15, 18–24, 26–28, 30, 33–35, 37, 42, 46, 50, VIII

AEVI Asociación Española de Videojuegos. 1

BGR Blue, Green, Red. 43, 44, 51

BLE Bluetooth Low Energy. 23, 24, 26

BP Blueprint. 27, 34, 42, 44

CLI Interfaz de Línea de Comandos. 7

CUI Interfaz de Usuario por Conversación. 7

EMR Resonancia Electromagnética. 11, 23, 24, 26, 27

ESA Entertainment Software Association. 1

FPS fotogramas por segundo. 3, 17–20, 40, 44, 46, 47, 50, 51

GDC Games Developer Conference. 1

GDPR Reglamento General de Protección de Datos. 18, 28

GUI Interfaz Gráfica de Usuario. 7, 9

HCI Interacción Humano-Computador. 5, 6

HSV Hue, Value, Saturation. 25, 31, 40, 41, 43–45, 51

IA inteligencia artificial. 7, 26, 27

JSON JavaScript Object Notation. 32, 41–45, 51, 52

LOPDGDD Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales. 18, 28

NFC Near-Field Communication. 23–25

NUI Interfaz Natural de Usuario. 7

RA Realidad Aumentada. 1

RFID Identificación por Radiofrecuencia. 9, 23–25

RGB Red, Green, Blue. 43

RV Realidad Virtual. 1

TCP Protocolo de Control de Transmisión. 43

TUI Interfaz de Usuario Tangible. 5, 7–13, 30

UDP Protocolo de Datagramas de Usuario. 14, 18, 19, 21, 28, 31–35, 41–47, 49, 50, 52, VIII

VFX efectos visuales. 23

VUI Interfaz de Usuario por Voz. 7

Agradecimientos

En primer lugar agradecer a mis padres y a mi hermano por su apoyo incondicional.
A mis amigos por hacer la carrera más llevadera.
Y a mis tutores por sus consejos.

Índice general

Resumen ..... **I**

Abstract ..... **II**

Glosario ..... **III**

Siglas ..... **IV**

Agradecimientos ..... **VI**

1 Introducción **1**

2 Objetivos **3**

2.1 Objetivo principal 3

2.2 Objetivos funcionales 3

2.3 Objetivos personales 4

3 Antecedentes **5**

3.1 Interacción Humano-Computador 5

3.2 Interfaces de usuario 6

3.3 Interfaces de Usuario Tangibles 8

4 Estado del arte **10**

4.1 Orígenes de las interfaces tangibles 10

4.2 Aplicaciones comerciales 11

4.3 LEGO como sistema tangible modular 13

5 Normas y referencias **14**

5.1 Métodos 14

5.2 Herramientas 15

5.2.1 Herramientas de implementación 15

5.2.2 Herramientas metodológicas 15

5.2.3 Herramientas para la elaboración de la memoria 15

5.3 Prototipos 16

5.4 Métricas 17

5.5 Normativa 18

6 Requisitos iniciales **19**

7 Restricciones y alcance **20**

7.1 Restricciones 20

7.2 Alcance e impacto 21

8 Estudio de alternativas y viabilidad **23**

8.1 Viabilidad Técnica 23

8.1.1 Tecnologías para la detección de las piezas 23
8.1.2 Motores para la visualización 3D 27
8.2 Viabilidad Legal 28
8.3 Viabilidad Económica 28

9 Descripción de la solución propuesta 30
9.1 Arquitectura general del sistema 30
9.2 Funcionalidades de la herramienta 31
9.2.1 Módulo de detección 31
9.2.2 Módulo de visualización 33
9.3 Aplicación de ejemplo 34
9.3.1 Modo construcción 34
9.3.2 Modo juego 36
9.4 Manuales de usuario y programador 37
9.5 Demostración 37

10 Análisis de Riesgos 39

11 Aspectos relevantes del desarrollo 41
11.1 Arquitectura de la implementación 41
11.2 Módulo de detección 42
11.2.1 Algoritmo de detección de la base 42
11.2.2 Detección de color y posición 43
11.2.3 Comunicación UDP 43
11.2.4 Configuración externa 43
11.2.5 Optimización 44
11.3 Módulo de visualización 44
11.3.1 Receptor UDP 44
11.3.2 Instanciación de actores 45
11.3.3 Calibración dinámica de colores (descartada) 45
11.3.4 Integración en motores de videojuegos 45
11.3.5 Aplicación de demostración (prototipado de niveles) 46
11.4 Resultados de las métricas y pruebas 46
11.4.1 Rendimiento conjunto del sistema 46
11.4.2 Otras pruebas realizadas 47

12 Gestión del proyecto 48
12.1 Recursos 48
12.2 Planificación temporal 48

13 Conclusiones y trabajo futuro 50
13.1 Conclusiones 50

13.2 Reflexión final 51
13.3 Limitaciones y trabajo futuro 51

14 Bibliografía 53

Índice de anexos

A Especificaciones del sistema

B Módulo de detección

C Módulo de visualización (Unreal Engine)

D Aplicación de prototipado de niveles (Unreal Engine)

E Manual de usuario

F Manual de programador

Índice de figuras

1 Flujo de la información en la memoria humana 6
2 Ilustración de interacción tangible (Fitzmaurice et al., 1995) 9

3 Urp, sistema tangible de planificación urbana con simulación de sombras y viento (Ishii et al., 2012) 10
4 Makey Makey (JoyLabz LLC., 2026) 11
5 Juego de combinación de colores con Sifteo Cubes (Merrill et al., 2012) . 11
6 Ilustración de Osmo funcionando (Tangible Play, Inc., 2026) 12
7 Dos personas interactuando con una Reactable (Jordà et al., 2007) 12
8 Bit Planner (Special Projects, 2013) 13
9 Videojuego espacial de puzzles (Liermann, 2025) 13

10 Boceto de la interfaz del módulo de detección de piezas 16
11 Prototipo visual de detección de piezas 16
12 Wireframes de las interfaces de la aplicación de prototipado de niveles . . 17
13 Prototipo funcional del modo construcción de la app de prototipado de niveles 17

14 Logotipo de la herramienta desarrollada, ChromaVision 30
15 Arquitectura del sistema 31
16 Módulo de detección: el contorno naranja delimita la base y los puntos azules marcan las esquinas 32
17 Módulo de detección: los círculos señalan las piezas detectadas y su color. La cuadrícula ayuda a colocarlas aunque puede ocultarse 33
18 Módulo de visualización: parámetros del receptor UDP en el editor de Unreal Engine 34
19 Módulo de visualización: parámetros del componente generador de piezas en el editor de Unreal Engine 34
20 Aplicación de prototipado de niveles: modo construcción 36
21 Aplicación de prototipado de niveles: modo juego 36
22 Ventana «Base» con la base detectada y las piezas colocadas sobre ella . 37
23 Ventana «Piezas» con las piezas detectadas y la cuadrícula oculta 38
24 Nivel generado en la aplicación de demostración a partir de las piezas detectadas 38

25 Diagrama de Gantt, planificación temporal del proyecto 49

Índice de tablas

1 Tabla comparativa de tecnologías para la detección de piezas 25
1 Tabla comparativa de tecnologías para la detección de piezas 26
2 Registro de riesgos del proyecto 40
3 Rendimiento conjunto del sistema 47

1. Introducción

La industria del videojuego ha demostrado ser un sector en auge estos últimos años. Según el anuario de la Asociación Española de Videojuegos (AEVI, 2024), 22 millones de personas jugaron a videojuegos en 2024, dedicando 8,2 horas semanales. Además, en Estados Unidos aproximadamente 205 millones de personas (el 64 % de la población) juegan al menos una hora a la semana. La mayoría de los adultos estadounidenses cree que los videojuegos alivian el estrés, fomentan la socialización y tienen valor educativo (Entertainment Software Association [ESA], 2025).

Más allá de los videojuegos, las aplicaciones en tres dimensiones (3D) interactivas tienen un amplio uso en sectores como la medicina, la ingeniería o la educación, con ejemplos como simuladores quirúrgicos, gemelos digitales o entornos de Realidad Virtual (RV) para la formación. Estos datos demuestran la importancia de ambos sectores y su popularidad entre personas de todas las edades.

Sin embargo, tanto en el desarrollo como en el uso de videojuegos y aplicaciones 3D, las herramientas empleadas son mayoritariamente digitales: software, motores gráficos, interfaces de ratón y teclado, o pantallas táctiles. Este enfoque limita la colaboración e interacción física entre varias personas.

Existen videojuegos que incorporan elementos tangibles, como Nintendo Labo (kits de cartón que se acoplan a los mandos), el sistema de control de la Wii, Just Dance (seguimiento de movimiento), gafas de RV, etc. En el ámbito educativo, los juegos buscan captar la atención de los alumnos y fomentar el aprendizaje, pero la parte tangible suele estar basada en la Realidad Aumentada (RA), con ejemplos como El Shemy et al. (2025) de juegos para niños en el espectro autista.

También pertenecen a esta categoría los videojuegos de control alternativo, que utilizan periféricos no convencionales o personalizados. A menudo son creados en game jams, que son eventos donde desarrolladores individuales o equipos crean un videojuego en un tiempo limitado y con una temática específica, como la Game Developer Conference (GDC, 2026). No obstante, estos ejemplos están diseñados como experiencias cerradas, los accesorios son específicos para un único juego o ecosistema y no permiten que el usuario cree sus propias reglas o modifique la interacción. Además, la mayoría de ellos se centran en la jugabilidad, no en proporcionar una herramienta para el diseño colaborativo.

La motivación de este trabajo es cubrir la carencia de este tipo de herramientas tangibles, que favorecen la generación espontánea de ideas en el trabajo en equipo. Se desea emular esos momentos creativos que ocurren en el mundo físico, cuando se emplea cualquier objeto a mano para expresar una idea, como cuando alguien representa una situación mediante botes de especias.

Se propone el desarrollo de una herramienta colaborativa que permita interactuar

con el entorno físico y visualizarlo en 3D. El sistema consiste en una base en la que los usuarios colocan piezas de construcción de colores. Mediante visión artificial se reconocen los colores y su posición, y esta información se transmite durante la ejecución a un motor de videojuegos para su posterior integración en una aplicación o videojuego. De este modo, se logra una interacción tangible que permite a varias personas trabajar de forma simultánea, combinando la fluidez del mundo físico con el amplio abanico de posibilidades de los motores de videojuegos.

Esta solución es extrapolable a diversos ámbitos: prototipado colaborativo de niveles en videojuegos, aplicaciones didácticas, organización de estructuras en arquitectura, pixel art, etc. Su versatilidad no solo permite crear diferentes aplicaciones, sino también emplearla en distintos motores de videojuegos gracias a una programación modular. Además, este proyecto emplea elementos asequibles y no personalizados para la experiencia, como piezas de LEGO y una webcam, con el objetivo de que sea práctico y accesible.

La estructura de esta memoria se divide en varios capítulos. Inicialmente se exponen los objetivos del proyecto y se presentan los antecedentes y el estado del arte. Más adelante, se indican las herramientas empleadas, las limitaciones y las alternativas estudiadas. Se detallará el desarrollo de la solución propuesta analizando los riesgos y aspectos relevantes. Para finalizar con la gestión del proyecto, las conclusiones y el trabajo futuro, acompañado de los correspondientes anexos.

2. Objetivos

En este apartado se presentan los objetivos de este Trabajo de Fin de Grado, organizados en diferentes categorías.

2.1. Objetivo principal

El principal objetivo de este proyecto es desarrollar una herramienta colaborativa que permita a los usuarios interactuar con un entorno físico (compuesto por un tablero y piezas) que puedan manipular y visualizar en un espacio 3D virtual.

2.2. Objetivos funcionales

Se han determinado los siguientes objetivos funcionales:

- Implementar un sistema de identificación y detección de la posición de las piezas sobre un tablero empleando visión por computador (OpenCV).
- Establecer una comunicación entre el motor de videojuegos y el sistema de detección, que transmita los datos obtenidos del entorno físico (identificadores y posición).
- Diseñar el módulo de detección como un sistema externo e independiente, que use un protocolo genérico de comunicación que permita su integración con distintos motores (Unreal Engine, Godot, Unity, etc.) sin modificar su lógica de detección.
- Garantizar que la interacción con las piezas se refleje en el entorno 3D con una respuesta suficientemente rápida para no entorpecer la colaboración en tiempo real, midiendo de forma básica los fotogramas por segundo (FPS) como indicador de rendimiento.
- Desarrollar al menos una aplicación, con modelos 3D propios, para demostrar su funcionamiento y uso en distintos campos, como el didáctico o el diseño de videojuegos.

2.3. Objetivos personales

Además, los objetivos personales fijados son:

- Adquirir nuevos conocimientos, no impartidos en el grado, sobre la visión por computador y el uso de la librería OpenCV mediante una aplicación práctica.
- Integrar los conocimientos adquiridos durante el grado sobre motores de videojuegos y modelado 3D con un entorno físico interactivo, combinando ambos ámbitos.

3. Antecedentes

En este capítulo se presentan los fundamentos teóricos que sustentan este trabajo. Primero se introduce la Interacción Humano-Computador (HCI), analizando las capacidades y limitaciones de los usuarios y de los sistemas. A continuación, se clasifican los diferentes tipos de interfaces de usuario. Finalmente, se profundiza en la Interfaz de Usuario Tangible (TUI), marco en el que se encuadra la solución propuesta.

3.1. Interacción Humano-Computador

La HCI es la doctrina que se centra en diseñar y comprender la interacción entre las personas y los ordenadores (Rapp, 2023). Está estrechamente relacionada con la ergonomía y la psicología, como Dix et al. (2004) analizan en su libro. En esta comunicación intervienen dos agentes: el humano y el ordenador.

El ser **humano** es el usuario al que los sistemas interactivos deben asistir. Es por ello que sus capacidades y limitaciones cognitivas condicionan el diseño de estos sistemas.

Para entender estas limitaciones, se debe tener en cuenta que los humanos reciben estímulos e información principalmente a través de los canales visual, auditivo y táctil, en el caso de uso de un ordenador. Según Roda (2011) la atención tiene un rol muy importante en la HCI, además, es selectiva y limitada. Es por ello que los sistemas interactivos deben evitar una sobrecarga cognitiva del usuario y priorizar la información relevante. Como indica Dix et al. (2004), las personas constan de tres tipos de memoria (Figura 1):

- **Memoria sensorial**: actúa como regulador de los estímulos recibidos de los sentidos. Solo envía la información relevante a la memoria a corto plazo, ejerciendo de filtro.
- **Memoria a corto plazo o de trabajo**: guarda información temporal y tiene una capacidad limitada, de 7±2 elementos según el psicólogo cognitivo George A. Miller.
- **Memoria a largo plazo**: tiene una capacidad en teoría ilimitada que guarda toda la información que conocemos. Se divide en episódica (sucesos experimentados personalmente) y semántica (conceptos y habilidades adquiridas).

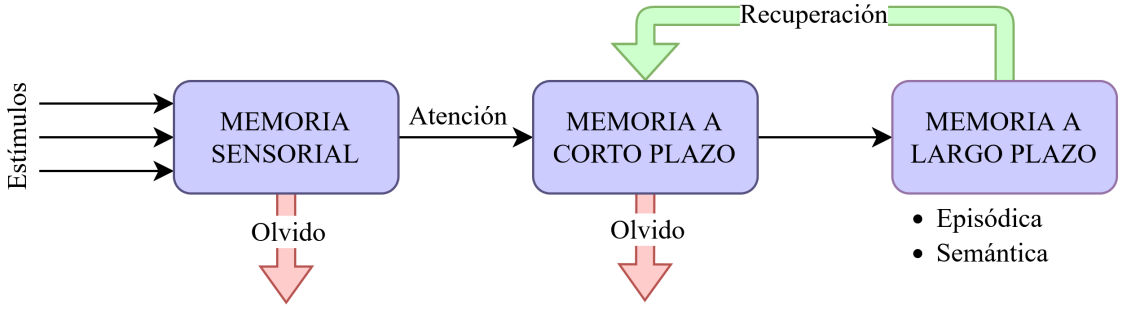

Figura 1: Flujo de la información en la memoria humana

Esta memoria nos ayuda a resolver problemas, razonar y adquirir habilidades pero puede llevar a errores si el contexto cambia. En el diseño de sistemas interactivos se deben tener en cuenta estas capacidades y sus limitaciones, por ejemplo, la información se retiene mejor si se organiza en grupos (chunking), la memoria por reconocimiento es mucho más eficaz que la memoria por recuperación o la percepción visual se centra en características con mayor saliencia. Sin embargo, no se puede olvidar que cada persona es distinta y tiene aptitudes individuales a tener en cuenta.

Por otro lado, la **computadora** envía información al usuario a través de sus dispositivos de salida (output) como pantallas o periféricos, y recibe la respuesta del usuario a través de dispositivos de entrada (inputs) como el teclado, la pantalla táctil, el ratón, etc. Existen diferentes dispositivos para que una persona interactúe con la computadora. En este caso las restricciones son técnicas. La velocidad de procesamiento puede impactar negativamente en la comunicación con el usuario, por ejemplo el retardo de movimiento del cursor. Esta velocidad se puede ver limitada por varios factores, como la memoria (principal y secundaria), los gráficos o el retardo en las redes. Aunque también puede verse limitado por aspectos de los periféricos, como la resolución de las pantallas. Por lo tanto, se debe tener en cuenta las propiedades de los dispositivos de los que se compone el sistema (Dix et al., 2004).

3.2. Interfaces de usuario

Las interfaces de usuario son la parte del sistema interactivo que se encarga de la HCI. Según Weiser (1991, como se cita en Hornbæk et al., 2025), las interfaces de usuario deben pasar desapercibidas. Sus objetivos y diseño deben:

- Adaptarse a las necesidades y capacidades de los usuarios que las utilizan.
- Permitir la interacción con el sistema.
- Ayudar a los usuarios a completar sus objetivos de forma eficiente y sin entorpecer el proceso.

Un concepto muy importante es la usabilidad, Hornbæk et al. (2025) explica que se refiere a la eficiencia, satisfacción y facilidad con la que el usuario puede alcanzar su objetivo. Mientras que la accesibilidad se centra en obtener niveles similares de usabilidad en grupos con diferentes capacidades (Hornbæk et al., 2025). Es muy importante que los tiempos de respuesta sean cortos y que se proporcione una retroalimentación constante al usuario.

Las interfaces de usuario se pueden clasificar en numerosos tipos, entre ellas las siguientes (Chopra et al., 2023; Koçer & Alaçam, 2023):

- **Interfaz de Línea de Comandos (CLI)**: surgieron en la década de 1960. En ellas el usuario se comunica con el ordenador a través del teclado, escribiendo comandos en una terminal. La máquina los ejecuta y responde mediante texto en el monitor.

- **Interfaz Gráfica de Usuario (GUI)**: se originaron a finales de la década de 1970. En este tipo de interfaces no se escriben líneas de comando, sino que el usuario interactúa con elementos visuales. La información está organizada en pestañas, menús o ventanas. Además, estos componentes junto con botones e iconos permiten al usuario navegar a través de la interfaz.

- **Interfaz de Usuario por Voz (VUI)**: la interacción entre el humano y el aparato es mediante comandos de voz. El sistema usa reconocimiento de voz para interpretar la pregunta o el comando del usuario y responde también de forma auditiva. Un uso común son los asistentes de voz como Siri o Alexa, o su uso en la conducción.

- **Interfaz de Usuario por Conversación (CUI)**: el usuario se comunica con el dispositivo de forma similar a una conversación con un humano. Algunos ejemplos actuales son los asistentes de inteligencia artificial (IA) como ChatGPT o Gemini, con los que se puede mantener una interacción natural, incluso por voz.

- **Interfaz Natural de Usuario (NUI)**: los humanos interactúan con estas interfaces de una forma natural, reconociendo acciones comunes del día a día. Las NUI combinan diversas tecnologías de interacción, como el reconocimiento de voz, el reconocimiento de gestos, el seguimiento de ojos, táctil o biométrica. Se diferencian de las VUI o las interfaces por gestos en que estas suelen basarse en un único método de entrada.

Existen numerosos tipos de interfaces aparte de los mencionados. El diseño y la experiencia de usuario variará también dependiendo del dispositivo empleado como: móviles, relojes inteligentes, etc. Entre todos estos tipos, las TUI son las que más se

aproximan a la propuesta de este trabajo, por lo que se analizan con mayor detalle en el siguiente apartado.

3.3. Interfaces de Usuario Tangibles

El proyecto presentado en este Trabajo de Fin de Grado se sitúa bajo la categoría de TUI. Según Ullmer e Ishii (2000), «las TUI dan forma física a la información digital, empleando objetos físicos como representaciones y controles para los medios computacionales».

Este tipo de interfaces busca ofrecer una forma física de acceder a la información, cerrando la brecha entre lo digital y lo tangible (Ishii & Ullmer, 1997). Permiten un control directo del dispositivo mediante objetos físicos, aprovechando la destreza natural de los humanos en tareas de manipulación en el entorno real.

En el artículo «Bricks» de Fitzmaurice et al. (1995) realizaron varias pruebas para analizar la manipulación de piezas similares al presente trabajo. Observaron el comportamiento de los sujetos efectuando tres tareas: separar piezas de LEGO por colores lo más rápido posible, ordenar piezas de dominó de forma descendiente con distintos tamaños de espacio de trabajo, manipular objetos (mover, rotar y escalar) o dar forma a un curvígrafo flexible. Obtuvieron los siguientes resultados:

1. **Separación de piezas**: los usuarios usaban ambas manos con movimientos rápidos y un amplio vocabulario de gestos, llegando incluso a cruzar los brazos. En ocasiones deslizaban las piezas en lugar de cogerlas y colocaban varias simultáneamente. Otro detalle es el uso de un gran área de trabajo abarcable con sus brazos.

2. **Ordenar piezas de dominó**: los sujetos ocasionalmente usaban el tacto para coger piezas, mientras su atención visual estaba dirigida a otra tarea. Empleaban la mano no dominante para alinear o guardar piezas temporalmente, mientras con la dominante manipulaban y cogían otras. Además, cuando el espacio de trabajo era reducido apilaban piezas para aprovecharlo.

3. **Manipulación de objetos**: los objetos físicos suelen permitir varios puntos de anclaje para manipularlos. Habitualmente los usuarios los movían, rotaban y escalaban simultáneamente. Contrasta con el software de dibujo digital, que requiere realizar acciones secuenciales seleccionando herramientas de distintos menús.

4. **Curvígrafo flexible**: los usuarios comprendían rápidamente sus propiedades y lo manipulaban usando todos los dedos y la palma.

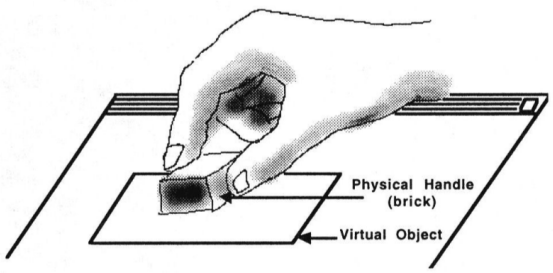

Figura 2: Ilustración de interacción tangible (Fitzmaurice et al., 1995)

Este trabajo fue uno de los primeros en utilizar objetos físicos para manipular información digital (Fitzmaurice et al., 1995), aunque las TUI se consolidaron como concepto con el trabajo de Ishii y Ullmer (1997), titulado «Tangible Bits». Como indican Shaer y Hornecker (2009) en su monografía, Hiroshi Ishii y sus estudiantes del Tangible Media Group, perteneciente al MIT Media Lab, son pioneros y referentes en esta área.

Todas las características de las TUI mencionadas con anterioridad las distinguen de las GUI. Las interfaces gráficas son las más comunes y la información se encuentra confinada en una pantalla mientras que las TUI permiten una interacción intuitiva y natural en un espacio real.

En el ámbito educativo, las TUI tienen potencial para hacer el aprendizaje más tangible y motivador. Sin embargo, como señala Marshall (2007), la evidencia empírica sobre su efectividad real es aún limitada y a veces contradictoria. Aunque se ha observado que las TUI mejoran la implicación y la colaboración, no siempre superan a las interfaces gráficas en resultados de aprendizaje concretos (Zhou & Wang, 2015). Es por ello que el valor educativo de una TUI depende en gran medida del diseño, el contexto de uso y la tarea específica.

Finalmente, no hay un estándar definido de qué dispositivos de entrada y salida usan las TUI. No obstante, es necesario un sistema de detección que reconozca los objetos físicos y su posición. Según Shaer y Hornecker (2009) algunas tecnologías comunes de uso son etiquetas de Identificación por Radiofrecuencia (RFID) y visión artificial. En esta última tecnología está basado el presente trabajo, orientado a la colaboración y el prototipado rápido. El sistema desarrollado emplea visión artificial para identificar piezas de colores y su posición, después esta información se transmite a un motor de videojuegos para mostrar una visualización 3D.

4. Estado del arte

En este capítulo se realiza un estado del arte sobre las TUI, analizando desde sus orígenes en el MIT Media Lab hasta aplicaciones comerciales actuales, así como proyectos que emplean piezas LEGO como plataforma tangible.

4.1. Orígenes de las interfaces tangibles

Como se ha mencionado en el apartado anterior, los orígenes de las TUI se remontan a los trabajos realizados por Hiroshi Ishii y el Tangible Media Group, perteneciente al MIT Media Lab. Su proyecto «Bricks» (Fitzmaurice et al., 1995), exploró la manipulación de objetos virtuales mediante bloques físicos acoplables y sentó las bases de las TUI. Aunque este concepto se consolidó con «Tangible Bits» (Ishii & Ullmer, 1997) y prototipos como metaDESK, una mesa que detectaba la manipulación de objetos físicos (maquetas de edificios) y mostraba información digital en respuesta (un mapa 2D proyectado y modelos 3D).

Una aplicación práctica de esta visión fue Urp (Underkoffler & Ishii, 1999), un simulador de planificación urbana. Permitía mover maquetas de edificios sobre una superficie mientras el sistema proyectaba las sombras que estos generarían a una hora determinada del día, la dirección del viento o los reflejos de las ventanas (Figura 3).

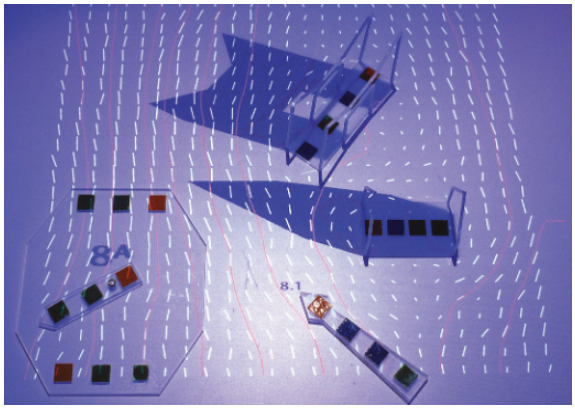

Figura 3: Urp, sistema tangible de planificación urbana con simulación de sombras y viento (Ishii et al., 2012)

Urp ejemplifica una posible aplicación de la herramienta propuesta en este TFG. Gracias a la flexibilidad del sistema y usando Unreal Engine como motor de simulación, las piezas de LEGO podrían representar edificios o carreteras en el motor de videojuegos. De este modo, se podrían explorar distribuciones urbanísticas de forma tangible, sin necesidad del hardware especializado que requería Urp (I/O Bulb, cámaras y proyectores). Cabe destacar que Urp mostraba los resultados de las si-

mulaciones en el entorno real mediante proyectores, mientras que la herramienta propuesta lo hace en un entorno digital, dentro del motor de videojuegos.

Estos trabajos sentaron las bases conceptuales de las TUI e inspiraron el desarrollo de nuevas aplicaciones.

4.2. Aplicaciones comerciales

Los sistemas comerciales basados en las TUI se pueden agrupar en dos grandes grupos: objetos electrónicos y visión artificial.

En el **primer grupo** se encuentra Makey Makey (Silver & Rosenbaum, 2012), un kit que convierte objetos cotidianos en entradas de teclado o ratón. Como se muestra en la Figura 4, una placa base conecta los objetos con el ordenador mediante pinzas de cocodrilo; al cerrar el circuito se envía la señal. Tiene un amplio uso en educación porque permite crear juegos y enseñar programación con Scratch.

Otra aplicación son los Sifteo Cubes (Merrill et al., 2012), herederos del prototipo académico Siftables del MIT Media Lab (Merrill et al., 2007). Como se observa en la Figura 5, eran pequeños cubos con pantallas táctiles y sensores de movimiento, diseñados como un sistema de videojuegos para niños. La interacción requería varios cubos, cada uno con su propia batería y pantalla. El producto fue descatalogado en 2014.

Finalmente, otro ejemplo de aplicación tangible electrónica son los tableros de ajedrez Chessnut (Pendo Technology Ltd., 2025). Se basan en la tecnología de Resonancia Electromagnética (EMR) desarrollada originalmente por Wacom. Emplea bobinas dispuestas en dos ejes perpendiculares que detectan los campos electromagnéticos de las piezas para identificar su posición con precisión. Su principal funcionalidad es jugar partidas online sin pantallas, ya que los movimientos del rival se muestran mediante luces o incluso moviendo las piezas de forma remota.

Figura 4: Makey Makey (JoyLabz LLC., 2026)

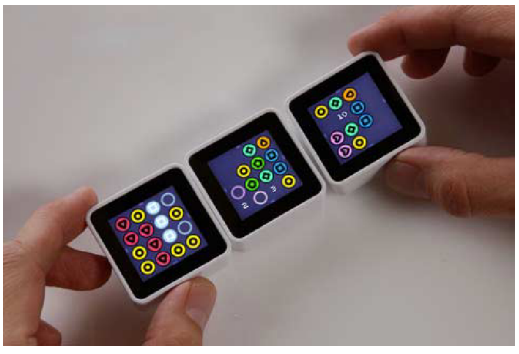

Figura 5: Juego de combinación de colores con Sifteo Cubes (Merrill et al., 2012)

Como se ha podido observar estos ejemplos se basan en el uso de dispositivos electrónicos especializados para su aplicación. A diferencia de los siguientes ejemplos que emplean **visión artificial**, un enfoque diferente y más cercano al de este trabajo.

En el ámbito educativo, Osmo (Tangible Play, Inc., 2026) es un sistema de juegos tangibles que usa un espejo acoplado a la cámara de un iPad para reconocer y seguir objetos delante del dispositivo (dibujos o fichas). Demuestra que la visión artificial puede crear experiencias tangibles de bajo coste y comercialmente exitosas (Figura 6).

Por último, las TUI también están presentes en otras áreas como la música. Reactable (Jordà et al., 2007) era una mesa musical colaborativa. Permitía crear música electrónica mediante objetos combinables que simulaban sintetizadores (Figura 7). Identificaba sus posiciones mediante visión artificial (software ReacTIVision), utilizando una cámara y un proyector situados debajo de la superficie. Fue un éxito comercial y académico, desarrollado por el Grupo de Tecnología Musical de la Universidad Pompeu Fabra, aunque actualmente ha dejado de operar.

Figura 6: Ilustración de Osmo funcionando (Tangible Play, Inc., 2026)

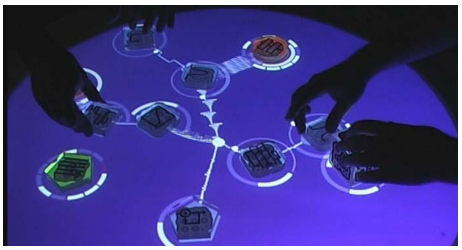

Figura 7: Dos personas interactuando con una Reactable (Jordà et al., 2007)

Como se ha demostrado, los enfoques basados en objetos electrónicos (Makey Makey, Sifteo o Chessnut) requieren hardware especializado, baterías o tecnologías propietarias, lo que incrementa el coste y la complejidad. Por el contrario, los sistemas de visión artificial (Osmo o Reactable) logran experiencias tangibles con una cámara estándar y objetos pasivos. Es por ello que el presente trabajo opta por la visión artificial, buscando un equilibrio entre flexibilidad, coste reducido y facilidad de adaptación a distintos motores de videojuegos. El Capítulo 8 (Estudio de alternativas y viabilidad) profundizará en esta comparativa.

4.3. LEGO como sistema tangible modular

Las piezas de LEGO son un sistema de construcción ampliamente conocido y accesible, está compuesto por bloques que encajan entre sí con precisión. Gracias a su popularidad y simplicidad, la curva de aprendizaje es muy reducida, la mayoría de las personas entienden intuitivamente cómo funcionan y cómo interactuar con ellas. Por lo tanto, no es casualidad que sean utilizadas tanto en el ámbito académico como en proyectos personales de TUI.

Un referente educativo es la línea de robótica LEGO Mindstorms que permitía construir robots sin requerir conocimientos sobre electrónica o programación.

Otro ejemplo es Bit Planner (Special Projects, 2013), una agenda tangible a gran escala que se sincroniza con la nube. Consiste en varias bases segmentadas en meses y días sobre las que se colocan piezas de LEGO de colores para representar proyectos (Figura 8). Una simple foto tomada con el móvil sincroniza el estado de la agenda con la nube empleando visión artificial para reconocer el color y la posición de las piezas. De esta manera la información está accesible de forma física y digital.

Por último, en el ámbito de los videojuegos Liermann (2025) desarrolló un prototipo de juego de puzzles en el que el jugador debe colocar piezas de LEGO para resolverlos (Figura 9). Tiene un enfoque similar, empleando visión artificial (OpenCV) para la identificación de las piezas y Unreal Engine.

Figura 8: Bit Planner (Special Projects, 2013)

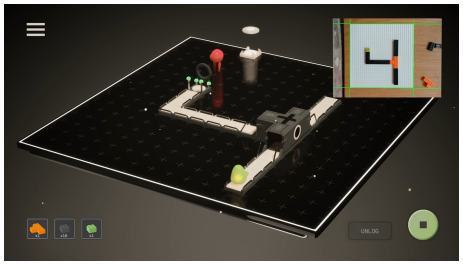

Figura 9: Videojuego espacial de puzzles (Liermann, 2025)

A diferencia de este tipo de proyectos, que se centran en una aplicación o videojuego específicos, este trabajo presenta una herramienta genérica de prototipado colaborativo. Su objetivo es ofrecer una base modular y reutilizable que permita a los usuarios visualizar en 3D la disposición de piezas físicas, independientemente del motor de videojuegos o el área de la aplicación (prototipado de niveles de videojuegos, pixel art, urbanismo, etc.). Además, este sistema aprovecha la amplia gama de colores de las piezas LEGO para identificarlas fácilmente como diferentes tipos objetos dentro del motor de videojuegos.

5. Normas y referencias

Este capítulo detalla la metodología de desarrollo, las herramientas utilizadas, los prototipos elaborados, las métricas de evaluación y las normativas legales aplicables al proyecto.

5.1. Métodos

El desarrollo de este proyecto se ha organizado de forma secuencial e incremental, siguiendo una metodología similar al modelo de cascada. El trabajo se ha agrupado en cinco fases:

1. **Fase inicial**: estudio de las alternativas tecnológicas para la identificación y localización de piezas, explicado en profundidad en el Capítulo 8 (Estudio de alternativas y viabilidad). Una vez seleccionado OpenCV, se dedicó tiempo a aprender su funcionamiento básico y su sintaxis en Python.

2. **Módulo de visión artificial**: implementación en Python con OpenCV. Incluye detección de colores, detección de la base, cálculo de las posiciones de las piezas y envío de la información por sockets UDP (Protocolo de Datagramas de Usuario).

3. **Módulo de visualización**: implementación en el motor de videojuegos Unreal Engine con C++. Incluye un receptor por sockets UDP y sistema de instanciación de actores según los colores y posiciones recibidas.

4. **Aplicación de ejemplo**: desarrollo de una aplicación para prototipado de niveles con modo construcción y modo juego. Incluye la creación de interfaces de usuario, objetos 3D, etc.

5. **Revisión final**: inspección del código y sus comentarios para comprobar si se requiere refactorizar alguna funcionalidad. Además de realizar la comprobación de los rangos de colores implementados y el empaquetado final de las aplicaciones.

Estas fases se han ejecutado en orden, completando cada una antes de iniciar la siguiente. Este enfoque secuencial permitió aislar la complejidad de cada subsistema y verificar su correcto funcionamiento antes de proceder a la siguiente etapa, reduciendo los problemas de integración. Por ejemplo, los módulos de detección y de visualización son independientes, pero la visualización requiere los datos generados por la detección. Sin embargo, dentro de la fase de desarrollo de la aplicación de

ejemplo las tareas no siguen un orden secuencial tan estricto. La gestión de todas estas fases se detalla en el Capítulo 12 (Gestión del proyecto).

5.2. Herramientas

En este subapartado se enumeran las herramientas empleadas en la realización de este TFG, clasificadas según su ámbito de uso: implementación, metodológicas y elaboración de la memoria.

5.2.1. Herramientas de implementación

Para la programación del módulo de detección de las piezas se ha utilizado el lenguaje de programación **Python**, junto con la librería de visión por computador **OpenCV**. Esta biblioteca aporta las funciones necesarias para el procesamiento de imágenes, como: lectura de la webcam, conversión de espacio de color, detección de aristas, etc. Esta parte se ha programado empleando el editor de código **Visual Studio Code** y se ha empaquetado en un ejecutable (.exe) mediante la librería **PyInstaller**.

En cuanto al módulo de visualización, se escogió como motor de videojuegos 3D de ejemplo **Unreal Engine**, en concreto la versión 5.7.4. Cabe destacar que la comunicación con el módulo de detección se ha diseñado de forma versátil, permitiendo su adaptación a otros motores.

Finalmente, para modelar los objetos que se utilizan en la conversión de las piezas de colores, se ha empleado el software de gráficos 3D multiplataforma **Blender**. Además, para el texturizado de los objetos y la creación de los iconos implementados en las interfaces se ha utilizado el editor de imágenes rasterizadas **Krita**.

5.2.2. Herramientas metodológicas

Para el control de versiones del código de los módulos de detección de piezas y de visualización, incluyendo la aplicación de ejemplo, se ha empleado **Git** y **GitHub**.

5.2.3. Herramientas para la elaboración de la memoria

Para la redacción y edición de la memoria se ha empleado **Overleaf**, un editor **LaTeX** basado en la nube, ampliamente utilizado para la composición de artículos académicos y científicos. Además, las referencias bibliográficas se han gestionado con **Zotero**.

5.3. Prototipos

Durante el desarrollo del proyecto se realizaron bocetos en papel y prototipos de las interfaces de la aplicación. Estos prototipos sirvieron como base para el diseño definitivo. Su implementación se describe en el apartado «Interfaces» del Anexo B (Módulo de detección), y también del Anexo D (Aplicación de prototipado de niveles).

En la Figura 10 se puede observar un boceto sobre las interfaces del módulo de detección de las piezas. En él se plantea la toma de decisiones sobre qué ventanas era relevante mostrar y cómo estructurar la información en las correspondientes a la detección de la base y de las piezas. En la Figura 11 se presenta el prototipo inicial de la ventana que muestra la detección de piezas.

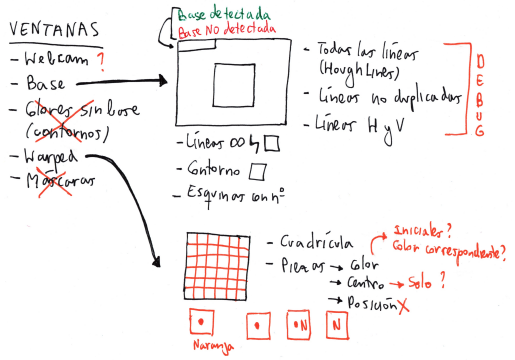

Figura 10: Boceto de la interfaz del módulo de detección de piezas

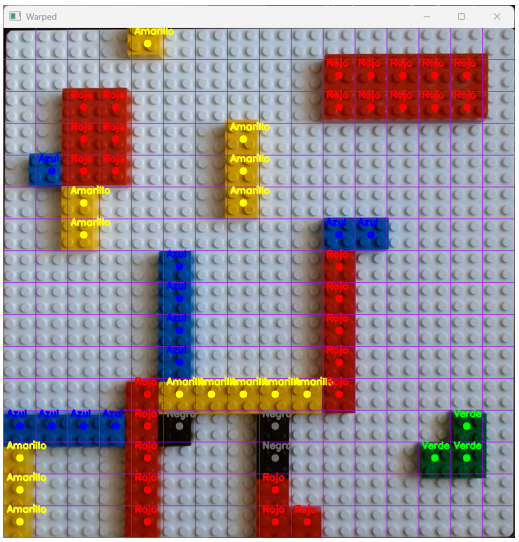

Figura 11: Prototipo visual de detección de piezas

En cuanto a la aplicación de ejemplo, en la Figura 12 se pueden encontrar los wireframes y bocetos de sus diferentes interfaces. Junto con un análisis de las funcionalidades que se pueden realizar en el modo construcción y el de juego.

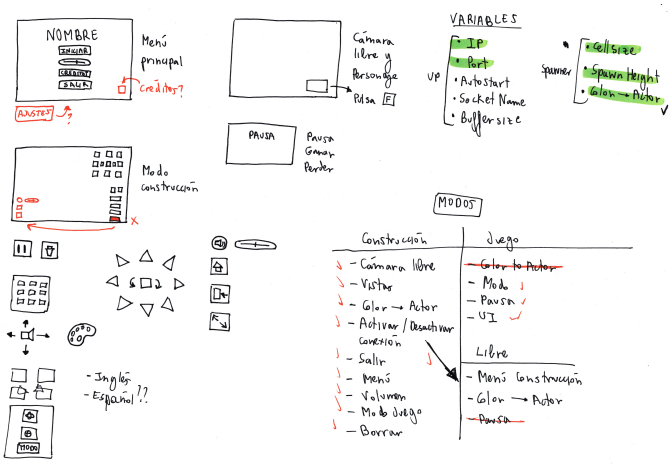

Figura 12: Wireframes de las interfaces de la aplicación de prototipado de niveles

En la Figura 13 se muestra el prototipo funcional del modo de construcción de la aplicación de prototipado de niveles.

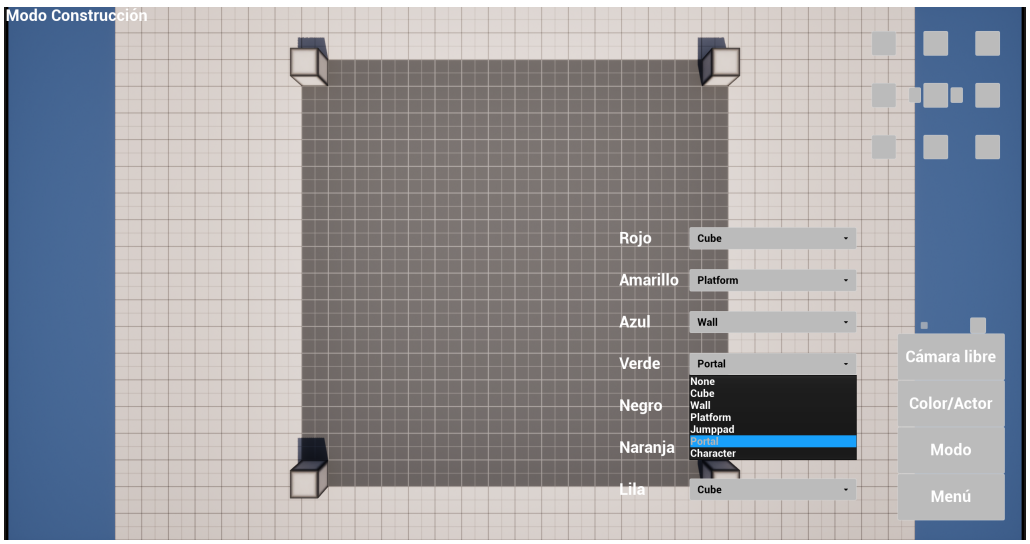

Figura 13: Prototipo funcional del modo construcción de la app de prototipado de niveles

5.4. Métricas

Para evaluar el rendimiento del sistema, se medirán los FPS alcanzados por cada módulo por separado:

- **Módulo de detección de piezas**: rendimiento del procesamiento de OpenCV para la detección de la base y de las piezas. Se debe tener en cuenta que la cámara utilizada captura vídeo a 30 FPS, lo que establece el límite superior teórico.

- **Módulo de visualización**: rendimiento de la aplicación de Unreal Engine.

El objetivo es comprobar que la interacción con las piezas se refleje en el entorno 3D con una respuesta suficientemente rápida para no entorpecer la colaboración en tiempo real.

El test consistirá en medir los FPS durante una sesión de uso de 1 minuto, manipulando diversas piezas de distintos colores y tamaños sobre la base. Se registrarán los valores promedio, máximos y mínimos.

No se ha considerado necesario medir la latencia de la comunicación UDP, ya que ambos módulos se ejecutan en la misma máquina y su contribución al retardo total es despreciable. En un futuro, si el sistema se distribuyera en red, esta métrica cobraría relevancia.

5.5. Normativa

El presente proyecto utiliza una webcam para capturar y procesar imágenes de una base sobre la que los usuarios colocan piezas de colores y con las que interactúan. Estas imágenes pueden contener datos de carácter personal por lo que es necesario considerar la normativa de protección de datos.

El marco legal aplicable en España es el Reglamento (UE) 2016/679 (GDPR) (Parlamento Europeo y Consejo de la Unión Europea, 2016) y su adaptación española, la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD) (Jefatura del Estado, 2018). Ambas normativas tienen por objetivo proteger los derechos de las personas en relación con el tratamiento de sus datos personales.

Sin embargo, este trabajo se acoge a las exenciones recogidas en dichas normas. En primer lugar, el artículo 2.2.c) del GDPR excluye del ámbito de aplicación a los tratamientos efectuados por una persona física en el ejercicio de actividades exclusivamente personales o domésticas (Parlamento Europeo y Consejo de la Unión Europea, 2016, pág. 32). Esta misma exclusión se recoge en el artículo 2.2.a) de la LOPDGDD (Jefatura del Estado, 2018, pág. 15). Por otro lado, el artículo 89 del GDPR establece garantías y excepciones para el tratamiento de datos con fines de investigación científica (Parlamento Europeo y Consejo de la Unión Europea, 2016, pág. 85-86).

Además, el sistema se ha diseñado bajo el principio de «minimización de datos» del artículo 5.1.c) del GDPR (Parlamento Europeo y Consejo de la Unión Europea, 2016, pág. 35). La aplicación únicamente procesa la información estrictamente necesaria (detección de la base, colores y posición de las piezas), descartando el resto de la imagen capturada. Asimismo, estos datos no se almacenan ni registran de forma permanente.

6. Requisitos iniciales

En este capítulo se presentan de forma resumida los requisitos funcionales y no funcionales de este proyecto.

Los **requisitos funcionales** se agruparon según los tres módulos del proyecto: detección de piezas mediante visión artificial, visualización 3D en Unreal Engine y la aplicación de ejemplo que permite prototipar niveles. Con respecto al módulo de detección, lo más relevante es la detección automática de la base y la identificación de los colores y posiciones de las piezas. En cuanto al módulo de visualización, lo más importante es la correcta recepción de los datos de detección enviados por sockets UDP desde el módulo de detección. Además, debe instanciar los actores 3D correspondientes a los colores y posiciones recibidas. Por último, la aplicación de prototipado de niveles debe tener la posibilidad de alternar entre un modo construcción con diversas herramientas y un modo de juego para probar el nivel diseñado.

En cuanto a los **requisitos no funcionales**, se ha prestado especial atención al rendimiento de la aplicación, que debe mantener al menos 15 FPS estables. También se ha tenido en cuenta la usabilidad de las interfaces del módulo de detección, así como el uso de hardware estándar. Por otro lado, se ha buscado la compatibilidad con diferentes motores de videojuegos, gracias a la independencia de los módulos de código.

La especificación completa, incluyendo el conjunto de historias de usuario y las tablas de requisitos detalladas, se pueden encontrar en el Anexo A (Especificaciones del sistema).

7. Restricciones y alcance

En este capítulo se describen las principales limitaciones técnicas identificadas durante el desarrollo del proyecto, así como el impacto esperado de la herramienta y las ventajas que aporta en distintos ámbitos de aplicación.

7.1. Restricciones

La solución propuesta está basada en visión artificial utilizando una webcam convencional, lo que supone ciertas limitaciones inherentes.

- **Iluminación**: la detección por color es sensible a las condiciones lumínicas. Por un lado, las sombras oscurecen los colores, pero pueden compensarse ajustando los rangos de detección. Por otro lado, los reflejos especulares saturan el color convirtiéndolo en blanco y dificultando la identificación. Es por ello que se recomienda utilizar una fuente de luz difusa natural o artificial, aunque en estas últimas la temperatura de color puede alterar la percepción cromática.

- **Gama de colores**: se puede detectar cualquier rango de colores pero es recomendable usar una gama de primarios, ya que las similitudes cromáticas pueden dar lugar a confusiones.

- **Detección de la base**: la base sobre la que se colocan las piezas debe contrastar con la superficie sobre la que se encuentre. Por ejemplo, utilizar una base blanca sobre fondo del mismo color dificulta su detección.

- **Hardware**: la detección se puede ver influida por el hardware utilizado. En este caso la webcam tiene una tasa de 30 FPS máximos, por lo que la detección no superará este límite.

- **Orientación y altura de las piezas**: la identificación de las piezas se realiza mediante color para que el sistema sea simple y accesible. Por este motivo, las piezas no tienen orientación, es decir, rotarlas no afecta al modelo 3D. Además, la visión cenital de la cámara solo reconoce una capa, por lo que no tiene sentido apilar piezas. Sin embargo, se pueden superar estas limitaciones físicas en el motor de videojuegos: rotación inteligente mediante programación, usar varias piezas de colores para representar las rotaciones de un mismo modelo o ajustar la altura de instanciación de los actores.

7.2. Alcance e impacto

En este proyecto se ha desarrollado una herramienta que identifica la posición y el color de piezas de construcción sobre una base. Entonces las sustituye por los objetos 3D asignados a cada color y las muestra en un entorno 3D. Como demostración, se ha desarrollado una aplicación de prototipado de niveles que permite construirlos de forma tangible y jugarlos.

Ventajas para los usuarios

Este trabajo presenta varias ventajas y mejoras para sus usuarios:

- **Accesibilidad**: funciona con hardware estándar, una webcam convencional y piezas de LEGO, que mucha gente tiene en su casa. No se necesitan componentes electrónicos especiales o calibración de la cámara.
- **Compatibilidad**: se trata de un sistema modular, es decir, el módulo detector de piezas es independiente del motor de videojuegos gracias a la conexión UDP. Esta versatilidad permite a los desarrolladores de aplicaciones y videojuegos emplear el motor que deseen.
- **Curva de aprendizaje baja**: el uso de piezas de LEGO, un sistema de construcción muy conocido, hace que la mayoría de los usuarios sean capaces de manipularlas de manera intuitiva.
- **Colaboración tangible**: varias personas pueden manipular piezas simultáneamente y ver los cambios reflejados en el entorno 3D, fomentando la generación de ideas y el trabajo en equipo.
- **Flexibilidad**: permite reasignar fácilmente qué objeto 3D corresponde a cada color y ajustar otros parámetros desde el editor del motor.

Impacto en diversos sectores

Por lo tanto la herramienta tiene un impacto en diversos sectores:

- **Desarrollo de videojuegos**: la aplicación de prototipado de niveles demuestra el uso de la herramienta en esta área. Facilita el diseño de mapas gracias a su interfaz tangible y agiliza la colaboración en equipo. Además, permite alternar entre la construcción del nivel y un modo de juego para comprobar su jugabilidad y realizar ajustes fácilmente.

- **Educación**: la tangibilidad del sistema supone una gran ventaja didáctica. Puede emplearse para enseñar conceptos espaciales (como coordenadas y escalas) o programación por bloques. También puede servir para que los alumnos creen sus propios niveles de videojuegos o circuitos de carreras. Además, el bajo coste facilita su implementación en entornos educativos.

- **Arquitectura y urbanismo**: adaptando los modelos 3D a elementos urbanísticos, el sistema podría utilizarse para planificar distribuciones de espacios (como ejemplifica Urp, Underkoffler e Ishii (1999)).

- **Otros sectores**: la herramienta podría adaptarse a juegos de mesa, actividades interactivas o incluso a la creación de pixel art, permitiendo generar imágenes 2D de forma tangible.

En resumen, la herramienta desarrollada ofrece una forma accesible, tangible y colaborativa de realizar diversas actividades en el ámbito del desarrollo de videojuegos, educativo o incluso urbanístico. Su bajo coste y su independencia de hardware especializado la hacen adecuada para entornos con recursos limitados, como estudios indie o centros educativos.

8. Estudio de alternativas y viabilidad

En este capítulo se analizan las diferentes alternativas tecnológicas consideradas para la detección de piezas y la visualización 3D. Además, se evalúa la viabilidad legal y económica del proyecto.

8.1. Viabilidad Técnica

Una vez determinada la idea de proyecto y los objetivos, se investigaron y analizaron posibles tecnologías para realizarlo. La herramienta tiene dos partes claramente distinguidas: la detección de las piezas, y su visualización y manipulación en un entorno 3D. Es por ello que el estudio de las alternativas se ejecutó en dos fases, que se detallan a continuación.

8.1.1. Tecnologías para la detección de las piezas

La herramienta se basa en el seguimiento de piezas sobre una base, es decir, se deben obtener dos variables para cada pieza: un identificador y su posición en el entorno real. Este identificador no es único para cada pieza, sino para un grupo de ellas. Esto permite sustituirlas por cualquier tipo de objetos en el motor de videojuegos, por ejemplo: puentes, plataformas, enemigos, VFX, etc. Teniendo en cuenta esta necesidad y los requisitos mencionados en el Capítulo 6 (Requisitos iniciales), se han considerado diversas tecnologías, agrupadas en dos conjuntos: inalámbricas y visión por computador. Se empleó una clasificación similar para exponer los ejemplos comerciales en el Capítulo 4 (Estado del arte).

Tecnologías inalámbricas

Se han considerado cuatro tecnologías inalámbricas: RFID, NFC, BLE y EMR. A continuación, se describe brevemente cada una.

Primero, la tecnología de **Identificación por Radiofrecuencia (RFID)** es un sistema de comunicación inalámbrica que permite identificar de forma única objetos sin contacto físico ni línea de visión directa. La antena capta las radiofrecuencias enviadas por las etiquetas (transpondedores) y el lector interpreta la información. Las etiquetas pueden ser activas, con batería y mayor alcance; o pasivas, que obtienen energía del lector. Según Greene (2025), el alcance se clasifica en tres categorías: baja frecuencia (LF), alta frecuencia (HF) y ultra alta frecuencia (UHF), con distancia de detección desde unos pocos centímetros hasta varios metros. Es muy utilizada en control de inventario y logística.

Por otro lado, como explica el NFC Forum (2026), la **Near-Field Communication (NFC)** es una tecnología de comunicación inalámbrica de corto alcance que permite un intercambio de datos seguro entre dispositivos. Opera mediante ondas de radio y requiere de una proximidad de hasta 2 cm. Similar al RFID, consta de dos modos de comunicación: activo, cuando ambos dispositivos generan campos para comunicarse; y pasivo, en el que uno se alimenta del otro para transmitir información. Sus principales modos de operación son: lectura/escritura de etiquetas, emulación de tarjetas o intercambio de datos entre dos dispositivos (peer-to-peer) (MathWorks, 2025). Esta tecnología está muy extendida en dispositivos móviles, control de accesos y pagos con tarjeta.

Otra tecnología de comunicación inalámbrica que se ha tenido en cuenta es **Bluetooth Low Energy (BLE)**. Está diseñada para un consumo de energía muy bajo, ideal para dispositivos que funcionan con baterías pequeñas durante largos periodos de tiempo. En BLE un dispositivo (periférico) anuncia su presencia, mientras que otro escanea y se conecta a él. Según Bluetooth SIG (2026), tiene un alcance de hasta 100 m y sus tres principales topologías son: point-to-point (dispositivo a dispositivo), broadcast (un dispositivo envía datos a muchos) y mesh (red de muchos a muchos). Además, permite estimar la distancia y dirección de otros dispositivos, lo que ayuda a determinar su posición relativa. Se emplea a menudo en domótica, wearables y balizas de proximidad.

La última tecnología inalámbrica que se ha tenido en cuenta es la **Resonancia Electromagnética (EMR)**. Fue comercializada por primera vez en 1987 por la empresa de tabletas gráficas Wacom. Consiste en un sistema que detecta la posición del lápiz, su inclinación, presión y botones sin necesidad de que esté alimentado por una batería. Según Wacom (2021) y la patente expirada US5028745A (Yamanami et al., 1991), la tableta tiene una serie de bobinas dispuestas en dos ejes perpendiculares. Alternan entre un modo de carga, que alimenta el circuito oscilador del lápiz; y un modo de escucha que detecta su campo magnético. Aunque su principal aplicación son las tabletas gráficas, la expiración de la patente ha permitido que surjan proyectos de código abierto, como Patchouli (Yukidama, 2024). También a facilitado la expansión de la tecnología a otros mercados, como el ajedrez con el tablero Chessnut (Pendo Technology Ltd, 2025).

Visión por computador

Además de las tecnologías inalámbricas, se han evaluado varios enfoques basados en visión por computador, que permiten detectar objetos mediante una o varias cámaras. A continuación se describen los principales.

Primero, la **fotogrametría** y el **escaneado 3D**, que reconstruyen modelos 3D a partir de múltiples imágenes. Aunque ofrecen una representación detallada, su

procesado es lento y no son adecuadas para aplicaciones que requieren respuestas en tiempo real.

Otro método son las **redes neuronales** como YOLO o TensorFlow. Son capaces de reconocer e identificar objetos, pero necesitan ser entrenadas con grandes conjuntos de datos y elevados recursos computacionales.

Por otro lado, se ha considerado la biblioteca de visión artificial **OpenCV** que permite procesar imágenes y reconocer objetos, entre otras funcionalidades. En primer lugar, los **marcadores fiduciales (ArUco)**, son patrones cuadrados impresos, que colocados en el campo de visión, se utilizan como referencia para realizar mediciones o identificar objetos y estimar su orientación.

Como cada pieza tendría que llevar uno y depender de la iluminación, se consideró la posibilidad de utilizar marcadores infrarrojos menos invasivos o incluso situar la cámara debajo de una base traslúcida. Otra alternativa es la **detección por color** mediante la conversión al espacio Hue, Value, Saturation (HSV) y el uso de umbrales para segmentar las piezas según su color.

Se valoraron otras alternativas, como sensores de profundidad o el uso de varias cámaras con un filtro de profundidad, pero estas opciones se descartaron rápidamente por su complejidad innecesaria para el alcance de este proyecto.

A modo de resumen, la Tabla 1 recoge las ventajas e inconvenientes de cada tecnología analizada, evaluadas según su idoneidad para el proyecto.

Tabla 1: Tabla comparativa de tecnologías para la detección de piezas

Tecnología	Ventajas	Inconvenientes
Inalámbricas		
RFID	• Etiquetas pasivas • Bajo coste de etiquetas • Inalámbrico • Etiquetas pequeñas • Alcance amplio • Detección rápida • Posible detección simultánea	• Una etiqueta por pieza • Hardware especializado (antenas y lector) • Posibles interferencias entre etiquetas
NFC	• Etiquetas pasivas • Bajo coste de etiquetas • Extendido en móviles (asignación de IDs)	• Alcance muy corto • Una etiqueta por pieza

Tabla 1: Tabla comparativa de tecnologías para la detección de piezas

Tecnología	Ventajas	Inconvenientes
BLE	• Bajo consumo • Inalámbrico • Alcance amplio	• Cada pieza necesita un emisor con batería • Alto coste • Baja precisión • Alta complejidad
EMR	• Alta precisión • Detección simultánea • Las piezas no requieren batería • Inalámbrico	• Hardware especializado • Superficie digitalizadora • Elevado coste • Componentes electrónicos en cada pieza
Visión Artificial		
Fotogrametría / Escaneado 3D	• Modelos 3D detallados	• Procesamiento lento (no apto para tiempo real) • Gran cantidad de imágenes • Alto coste computacional
IA	• Capacidad de identificar distintos objetos	• Alto coste computacional • Entrenamiento con grandes conjuntos de datos
Marcadores ArUco (OpenCV)	• Identificación única de piezas • Permite estimar orientación	• Marcadores visibles en cada pieza (invasivo) • Sensible a oclusión parcial
Detección por color (OpenCV)	• Baja carga de procesamiento • Ligero • Rápido • Fácil de implementar • Piezas sin componentes adicionales	• Sensible a la iluminación • Gama de colores diferenciada • No detecta orientación de piezas • No distingue piezas del mismo color si están próximas
Otras alternativas (varias cámaras, infrarrojos, etc.)	• Posible detección de apilamiento de piezas (con complejidad añadida)	• Alto coste • Alta complejidad • Montaje complejo

Decisión final

Tras el análisis comparativo recogido en la Tabla 1, se ha escogido la detección por color empleando OpenCV como tecnología más adecuada para la realización del proyecto. Esta opción no requiere hardware especializado ni componentes adicionales en las piezas, lo que la hace accesible y de bajo coste. Además, su integración es sencilla y el procesamiento rápido, lo que permite una comunicación ágil y compatible con diferentes motores de videojuegos. Por lo tanto, cumple con los requisitos establecidos en el Capítulo 6 (Requisitos iniciales).

Las tecnologías inalámbricas se descartan por la necesidad de hardware especializado, coste y complejidad técnica. Cabe destacar que se le dedicó especial atención a investigar la tecnología EMR durante la fase inicial, debido a su alta precisión de detección de posición. En cuanto al resto de alternativas de visión por computador: la fotogrametría no permite detección inmediata y, al igual que la IA, son computacionalmente pesadas. Los marcadores ArUco obligan a adherir patrones visibles, y las configuraciones con infrarrojos o varias cámaras suponen un montaje más complejo y caro.

Es por todo ello que OpenCV con detección por color resulta la solución más práctica, accesible y alineada con los requisitos de rápida detección, bajo coste y facilidad de uso que persigue este TFG.

8.1.2. Motores para la visualización 3D

Una vez decidida la tecnología aemplear para detectar las piezas, el siguiente paso fue elegir el motor de videojuegos encargado de la visualización y la interacción en 3D. Los tres motores más relevantes en la actualidad son Unreal Engine, Godot y Unity.

Unreal Engine es el motor líder en gráficos 3D realistas. Ofrece altas capacidades de renderizado en tiempo real, iluminación dinámica gracias a Lumen y geometría virtualizada con Nanite. Se programa utilizando el lenguaje C++ o mediante un sistema de scripting visual con nodos llamado Blueprint (BP).

Por otro lado, **Godot** es un motor de código abierto que destaca por ser ligero e intuitivo. Permite crear aplicaciones 2D y 3D, aunque tiene ciertas limitaciones gráficas. En los últimos años ha ganado popularidad porque ofrece un sistema de escenas basado en nodos, amplia documentación y su lenguaje principal es GDScript, similar a Python.

Por último, **Unity** es un motor multiplataforma que permite desarrollar tanto en 2D como en 3D. Emplea C# como lenguaje principal y dispone de una amplia comunidad y recursos de aprendizaje. Es muy popular para la creación de videojuegos para móvil, pero también se utiliza para consola y PC.

Tanto Unreal Engine como Unity son gratuitos para pequeños desarrolladores que no superen una cierta cantidad de ingresos. Por el contrario, Godot es completamente gratuito.

Se ha escogido Unreal Engine porque la aplicación de ejemplo se centra en el prototipado de niveles 3D, además es el motor enseñado en la titulación. No obstante, dado que el sistema de detección se ha diseñado de forma modular y envía los datos de las piezas mediante un protocolo estándar (UDP), el módulo de visualización podría haberse implementado en cualquiera de estos tres motores sin modificar el código del detector. De hecho, la compatibilidad con distintos motores de videojuegos se estableció como uno de los objetivos del proyecto. Se planteó el desarrollo de una segunda aplicación 2D de ejemplo en Godot para crear pixel art, con el objetivo de demostrar la versatilidad de la herramienta, en cuanto a motores y gráficos. Sin embargo, finalmente se priorizó pulir la aplicación 3D de ejemplo.

8.2. Viabilidad Legal

Como se ha explicado en la Sección 5.5, el desarrollo de este proyecto se acoge a las exenciones de las normativas de protección de datos GDPR (Parlamento Europeo y Consejo de la Unión Europea, 2016) y LOPDGDD (Jefatura del Estado, 2018). Se trata de una actividad de desarrollo personal y con fines de investigación, además, respeta el principio de minimización de datos porque solo procesa la información estrictamente necesaria sin almacenarla.

Asimismo, todas las herramientas empleadas para el desarrollo de este trabajo (detalladas en la Sección 5.2) disponen de licencias de código abierto o gratuitas para usos académicos y no comerciales. Por tanto, no existen restricciones legales que impidan la realización del proyecto ni su posterior difusión como trabajo académico.

8.3. Viabilidad Económica

El desarrollo del proyecto ha requerido hardware convencional. Se ha empleado un ordenador personal (sin coste adicional), una webcam (aproximadamente 60 €), una base blanca de LEGO (9 €) y piezas de LEGO (ya se disponía de ellas, pero se puede estimar un coste de unos 30 €). Por lo tanto, el coste asciende a alrededor de 100 €. Hay que tener en cuenta que la mayoría de los usuarios ya poseen ordenador personal y que las piezas de LEGO son muy comunes, por lo que se pueden reutilizar las que ya se poseen.

En cuanto al software, todas las herramientas empleadas son de código abierto o gratuitas para uso académico (OpenCV, Python, Unreal Engine, Blender, Krita, etc.), por lo que no han supuesto ningún desembolso en licencias.

La herramienta desarrollada en este trabajo tiene aplicaciones y beneficios en diversos ámbitos, como la educación o el desarrollo de videojuegos (como se analizó en la Sección 7.2). Al tratarse de un sistema de código abierto basado en hardware estándar, las posibles vías de monetización podrían ser: formación, personalización de la herramienta, venta de kits (webcam, base y piezas) o desarrollo de aplicaciones a medida. La baja inversión inicial necesaria hace que el sistema sea económicamente viable. Sin embargo, se trata de un trabajo académico cuya finalidad es la demostración de un prototipo funcional de bajo coste.

9. Descripción de la solución propuesta

ChromaVision (Figura 14) es una herramienta modular que permite a los usuarios manipular piezas físicas simultáneamente sobre una base y visualizar su disposición en un entorno 3D a medida que interactúan con ellas. Esta herramienta sirve para desarrollar aplicaciones y videojuegos colaborativos con TUI, ofreciendo la capacidad de modificar los colores de piezas detectadas y los objetos 3D asignados. Se trata de un sistema accesible, basado en visión artificial, que únicamente requiere un ordenador, una webcam, piezas y una base (en este caso de LEGO).

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

Figura 14: Logotipo de la herramienta desarrollada, ChromaVision

A continuación se describe la arquitectura general del sistema, sus funcionalidades principales y la aplicación de ejemplo desarrollada para demostrar su utilidad.

9.1. Arquitectura general del sistema

La herramienta desarrollada se compone de dos módulos independientes:

1. **Módulo de detección de piezas**: implementado en Python con OpenCV. Detecta el color y la posición de las piezas sobre la base.

2. **Módulo de visualización**: script integrado en el proyecto del motor de videojuegos. Recibe una lista con las piezas detectadas e instancia los objetos que el usuario ha asignado a cada color, en la posición correspondiente a la pieza física.

En la Figura 15 se muestra un diagrama del flujo de datos del sistema. Primero el **módulo de detección** captura y procesa las imágenes de la webcam para localizar la base. A continuación, identifica el color y la posición de las piezas situadas dentro de la base. Luego envía esta información al módulo de visualización mediante el

protocolo UDP. Este último interpreta el mensaje e instancia los actores en las posiciones correspondientes.

El **módulo de visualización** se añade al proyecto del motor de videojuegos y forma parte de la aplicación o videojuego desarrollado, que puede comunicarse con él para gestionar la conexión UDP, la asignación de actores, etc. En el presente trabajo, se ha demostrado el funcionamiento del sistema mediante una aplicación de prototipado de niveles, descrita en la Sección 9.3.

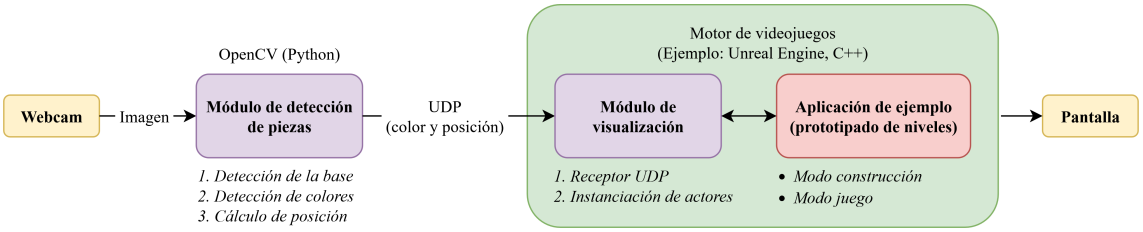

Figura 15: Arquitectura del sistema

La separación lógica entre ambos módulos, junto con la comunicación mediante UDP, permiten que la herramienta sea independiente del motor de videojuegos. Cualquier motor que incorpore en su proyecto el módulo de visualización correspondiente podrá recibir e interpretar los mensajes del módulo de detección sin necesidad de modificar su lógica.

En este trabajo se ha desarrollado el módulo de visualización para Unreal Engine. Es reutilizable, es decir, puede integrarse en cualquier proyecto de Unreal Engine sin modificar su código fuente, simplemente añadiendo los scripts al proyecto.

9.2. Funcionalidades de la herramienta

A continuación se enumeran las funcionalidades de la herramienta.

9.2.1. Módulo de detección

El módulo de detección, implementado en Python con OpenCV, es el encargado de procesar las imágenes de la webcam. Sus principales funcionalidades son:

- Detección automática de la base de trabajo.
- Trasformación de la perspectiva de la base a vista cenital (corrigiendo inclinaciones de la cámara o de la base).
- Identificación de las piezas dentro de la base por su color (en el espacio HSV).
- Cálculo de la posición de cada pieza sobre la base.

- Envío de los datos (color y posición) al módulo de visualización mediante el protocolo UDP.

El procesamiento se ejecuta de forma continua, actualizando la información a medida que se reciben nuevos fotogramas y los usuarios manipulan las piezas sobre la base.

Además, el módulo está empaquetado como ejecutable independiente, Al ejecutarlo, genera automáticamente un archivo JavaScript Object Notation (JSON) donde el usuario puede configurar los rangos de color, el tamaño de las ventanas, la dirección UDP, etc. Para recargar la configuración sin reiniciar el programa, basta con pulsar la tecla `R`. Si el usuario introduce valores incorrectos o elimina alguna parte de la configuración, el programa restaura los valores por defecto para evitar errores.

El módulo de detección muestra dos ventanas: la interfaz que presenta el campo de visión de la webcam con la base detectada, y otra que muestra las piezas detectadas junto con su color.

En la Figura 16 se puede observar la detección de la base de LEGO, recuadrada en naranja y con las esquinas marcadas en azul.

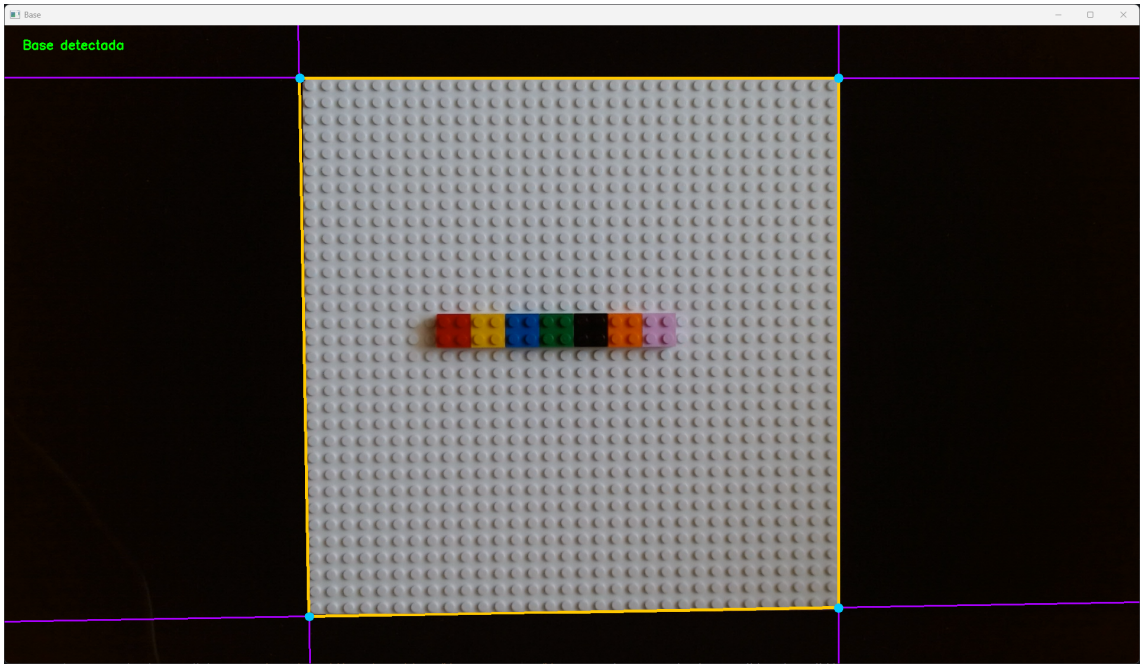

Figura 16: Módulo de detección: el contorno naranja delimita la base y los puntos azules marcan las esquinas

En la Figura 17 se puede observar la detección de las piezas ubicadas dentro de la base. La interfaz indica cuáles están siendo detectadas mediante círculos rellenos del color identificado. Además, la cuadrícula sirve de guía para posicionar las piezas, pero el usuario puede ocultarla en el archivo de configuración.

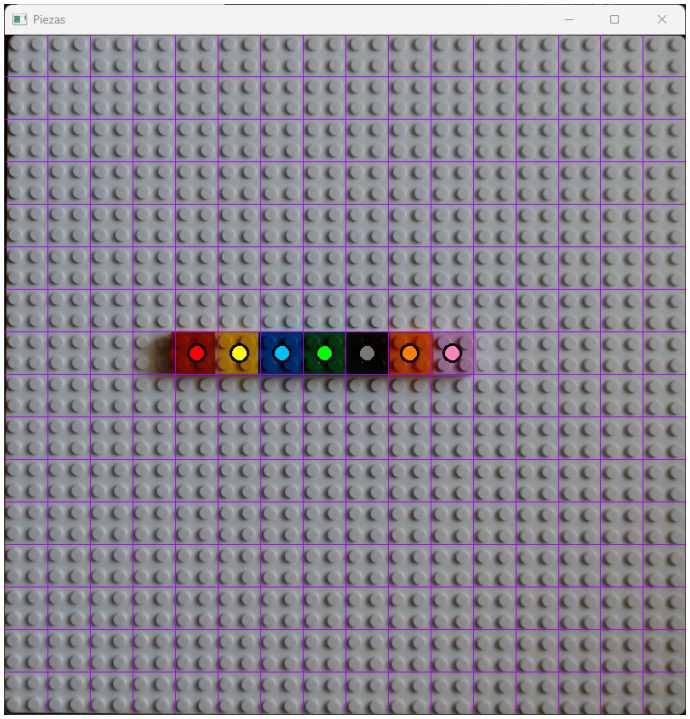

Figura 17: Módulo de detección: los círculos señalan las piezas detectadas y su color. La cuadrícula ayuda a colocarlas aunque puede ocultarse

9.2.2. Módulo de visualización

El módulo de visualización se integra en el proyecto del motor de videojuegos (en este trabajo, Unreal Engine). Sus funciones principales son:

- Recepción continua de datos a través de UDP y procesamiento sin interrumpir el flujo del juego.
- Interpretación de los mensajes para obtener el color y la posición de cada pieza.
- Instanciación dinámica de actores 3D en la escena, asignando a cada color la clase configurado por el usuario.
- Actualización eficiente de la escena: solo se añaden, actualizan o eliminan los actores que han cambiado.
- Limpieza completa de la escena cuando se recibe una lista vacía.

- Configuración desde el editor del motor el tamaño de celda, altura de instanciación, asignación color/actor y dirección UDP, sin necesidad de modificar el código fuente.
- Gestión de la recepción UDP, borrado manual de actores y notificaciones desde BPs, permitiendo lógica personalizada sin modificar el código fuente.

En resumen, este módulo transforma los datos de detección en una representación 3D dinámica. Su diseño modular y la exposición de parámetros en el editor (Figura 18 y Figura 19) permiten a los desarrolladores ajustar el comportamiento sin modificar el código fuente, facilitando su integración en distintos proyectos.

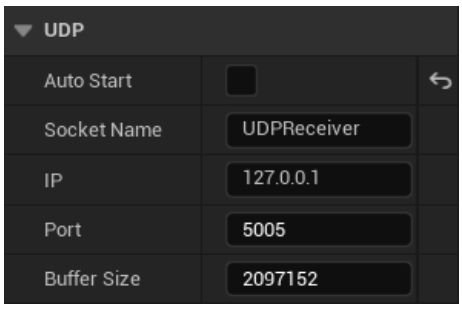

Figura 18: Módulo de visualización: parámetros del receptor UDP en el editor de Unreal Engine

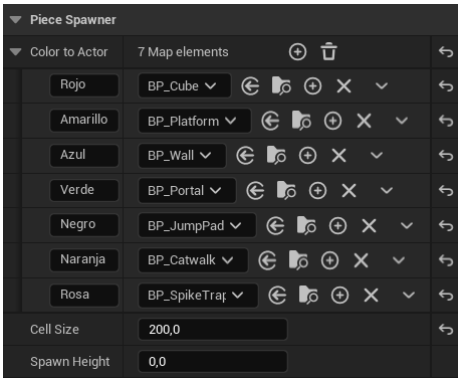

Figura 19: Módulo de visualización: parámetros del componente generador de piezas en el editor de Unreal Engine

9.3. Aplicación de ejemplo

Para demostrar el potencial de la herramienta, se ha desarrollado una aplicación de ejemplo en Unreal Engine que permite prototipar niveles de forma tangible. La aplicación se divide en dos modos principales: construcción y juego.

9.3.1. Modo construcción

En el modo construcción, el usuario coloca piezas LEGO sobre la base y la aplicación muestra los objetos 3D correspondientes (plataformas, muros, trampas, etc.) de forma sincronizada. Desde la interfaz se pueden realizar las siguientes acciones:

- Cambiar la vista de la escena mediante una cruceta, pudiendo rotar la vista cenital para alinearla con el ángulo de la cámara (que puede variar de posición).
- Cambiar a una cámara libre para observar el nivel desde cualquier ángulo.

- Pausar o reanudar la conexión UDP (la recepción de datos de las piezas).
- Borrar manualmente los actores generados.
- Reasignar qué clase de objeto se asocia a cada color de pieza.
- Cambiar al modo de juego.

Además, la interfaz incluye opciones generales como ajustar el volumen del audio, volver al menú principal o salir de la aplicación.

En la Figura 20 se puede observar el modo construcción con los actores asignados a cada color dispuestos según la posición de las piezas de la Figura 16 y Figura 17. Para esta aplicación de ejemplo se han identificado siete colores, asociados a los siguientes elementos:

- **Rojo**: bloques que el jugador no puede saltar.
- **Amarillo**: plataformas que se mueven verticalmente, permitiendo al jugador alcanzar zonas más altas.
- **Azul**: muros de doble altura que el jugador no puede saltar, ni siquiera usando otros modelos 3D.
- **Verde**: portal que, al tocarlo, completa el nivel (objetivo del juego).
- **Negro**: impulsores que propulsan al jugador hacia arriba cuando se sitúa sobre ellos.
- **Naranja**: pasarelas que permiten al jugador caminar por debajo y por encima de ellas.
- **Rosa**: trampas de pinchos que suben y dañan al jugador si pasa por encima de ellas.

Como se ha mencionado anteriormente, el actor correspondiente a cada color se puede asignar en la interfaz. Además, es posible eliminar la representación de un color seleccionando la opción «Nada» en el desplegable. También se puede asignar a un color el personaje, permitiendo al usuario colocarlo en la posición que desee, en lugar de usar la ubicación por defecto en el modo juego.

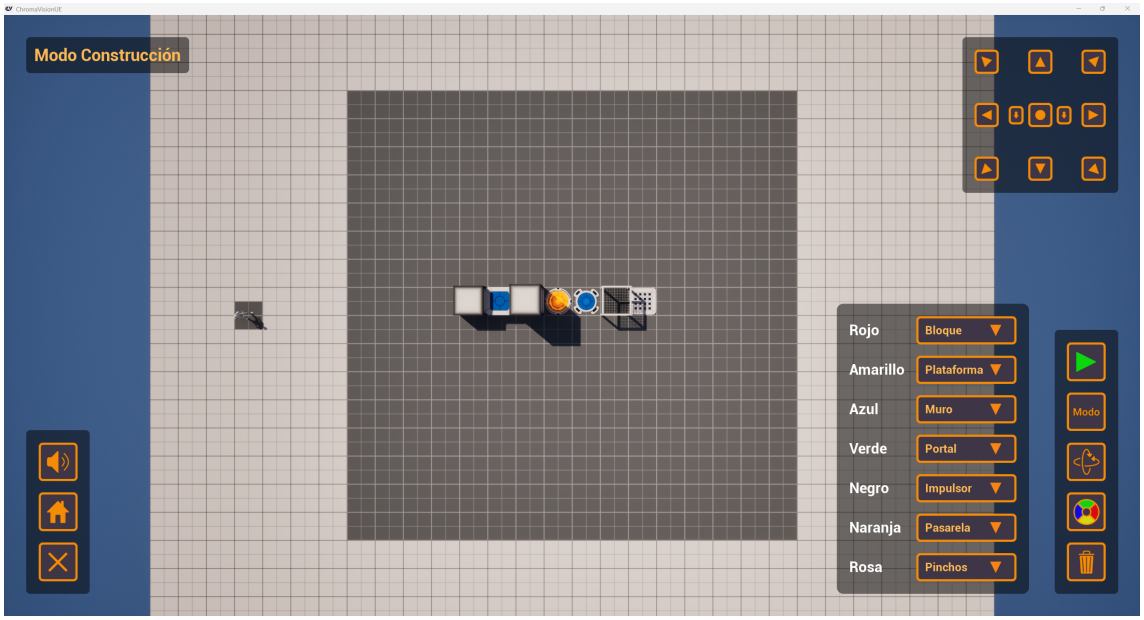

Figura 20: Aplicación de prototipado de niveles: modo construcción

9.3.2. Modo juego

Una vez generado el nivel en el modo construcción, se puede cambiar al modo juego. En este modo, el usuario controla un personaje en primera persona capaz de recorrer el nivel, interactuar con los objetos, recibir daño y alcanzar un objetivo, (llegar al portal para ganar). La tecla `F` permite volver al modo construcción.

En la Figura 21 se muestra el modo juego con los actores ubicados según las piezas detectadas en la Figura 17. Además, la barra de vida del jugador aparece en la esquina superior derecha.

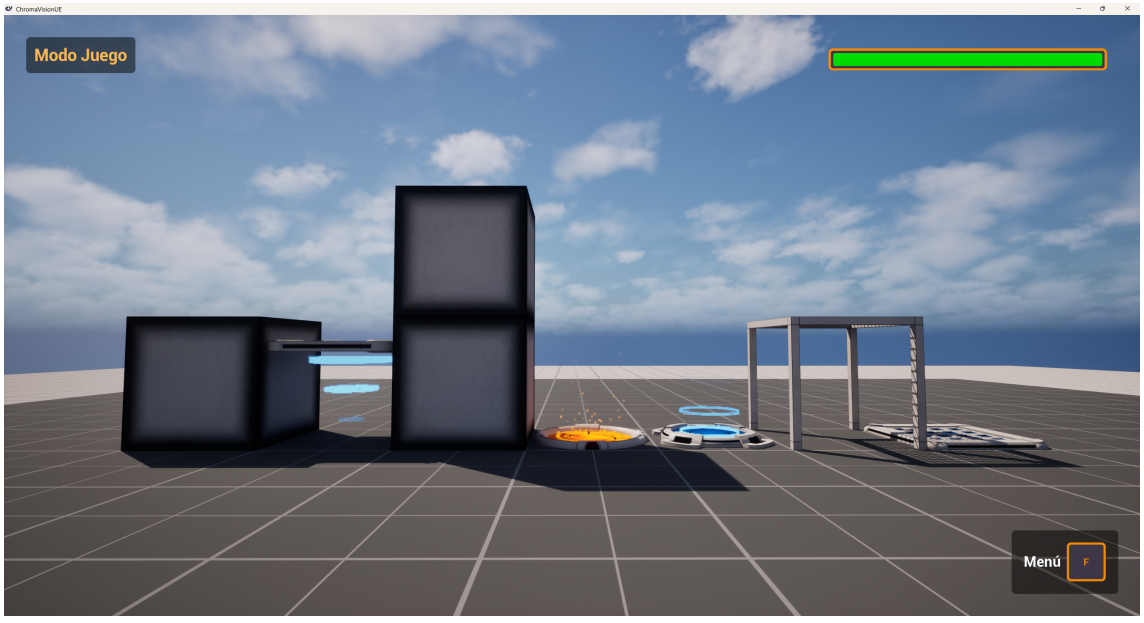

Figura 21: Aplicación de prototipado de niveles: modo juego

9.4. Manuales de usuario y programador

Los manuales de usuario y programador se incluyen en los Anexos E e F, respectivamente. El manual de usuario proporciona instrucciones detalladas para la ejecución y configuración del módulo de detección, la integración del módulo de visualización en Unreal Engine y el uso de la aplicación de demostración. El manual del programador ofrece una documentación técnica del código.

9.5. Demostración

Para ilustrar el funcionamiento de la herramienta y de la aplicación de ejemplo, se ha grabado un vídeo demostrativo accesible en el siguiente enlace: *enlace al vídeo*. En él se muestra el proceso completo de prototipado de un nivel: desde la colocación de las piezas sobre la base hasta la generación del nivel 3D y su exploración en modo juego.

La Figura 22 corresponde a la ventana «Base», que muestra la imagen capturada por la webcam con las piezas físicas colocadas sobre la base para diseñar el nivel.

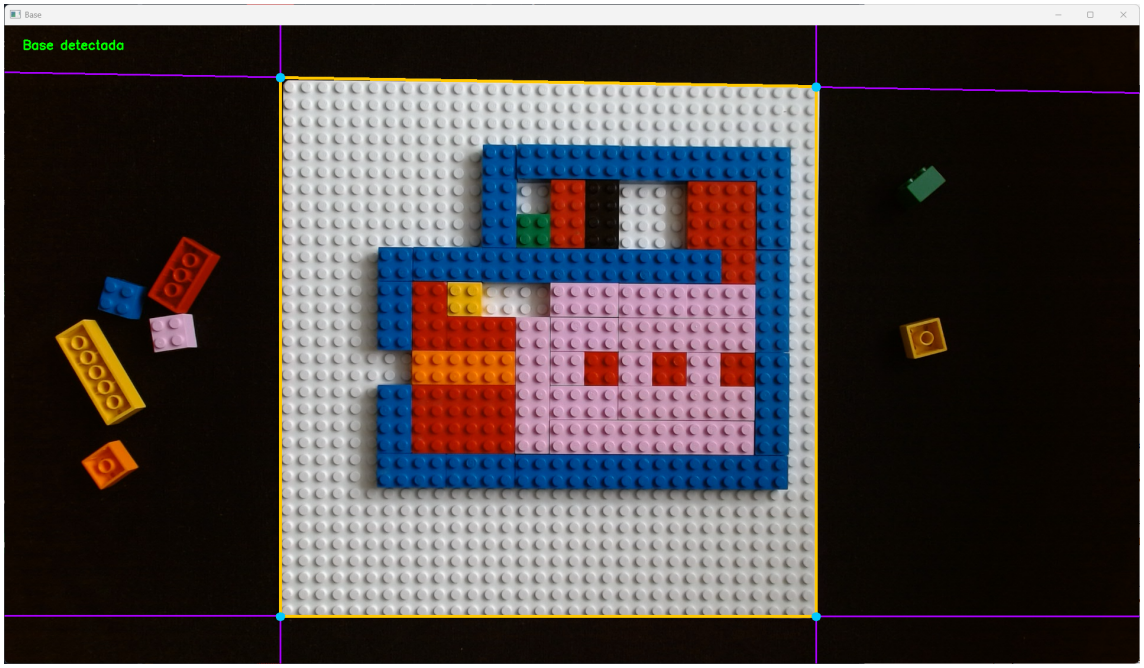

Figura 22: Ventana «Base» con la base detectada y las piezas colocadas sobre ella

La Figura 23 presenta la ventana «Piezas», que indica las piezas detectadas, en este caso con la cuadrícula oculta.

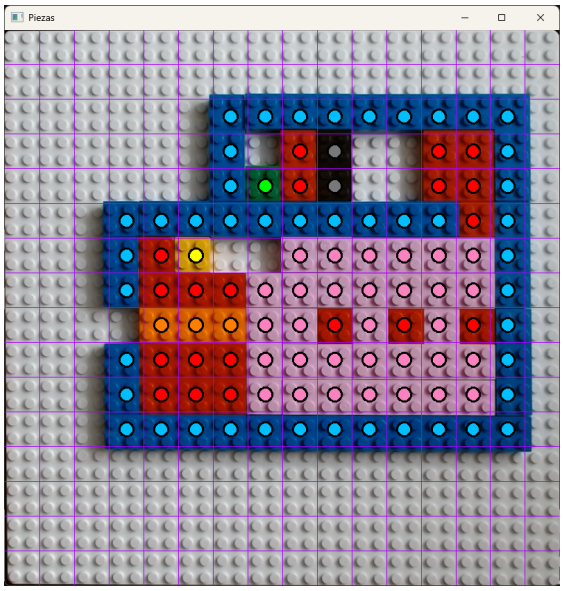

Figura 23: Ventana «Piezas» con las piezas detectadas y la cuadrícula oculta

Finalmente, la Figura 24 muestra el nivel generado en la aplicación de demostración a partir de las piezas detectadas.

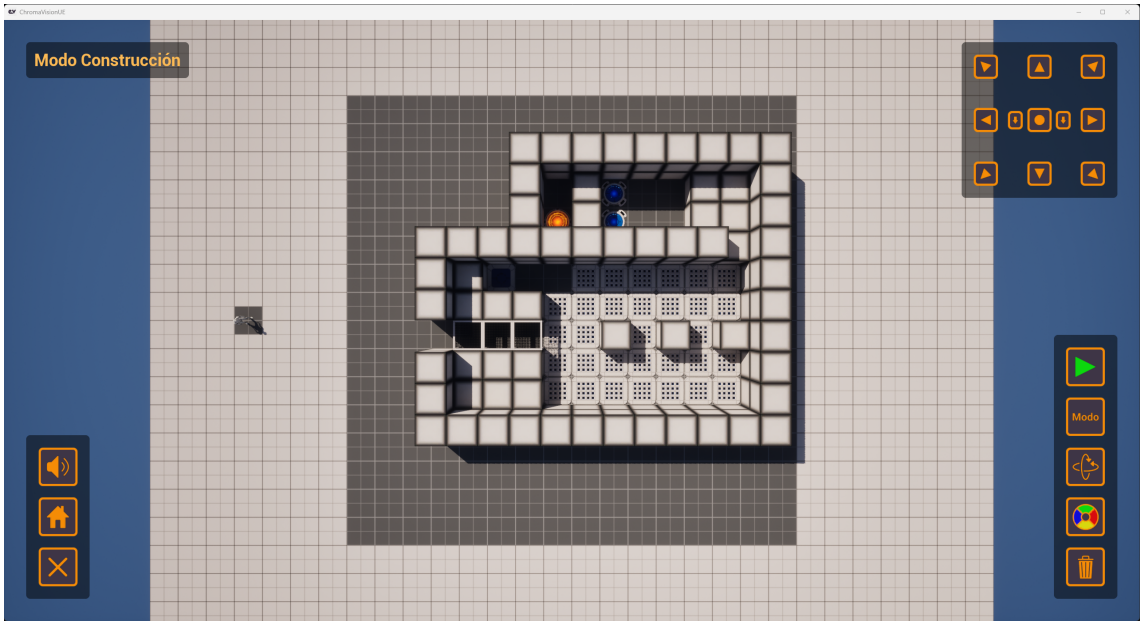

Figura 24: Nivel generado en la aplicación de demostración a partir de las piezas detectadas

10. Análisis de Riesgos

Durante el desarrollo del proyecto han surgido varios problemas que han condicionado algunas decisiones de diseño. En este apartado se recogen los principales riesgos identificados, tanto los que se previeron al principio como los que aparecieron durante la implementación. Para cada uno se indica su prioridad, su complejidad y las medidas que se tomaron para resolverlos o minimizar su impacto.

Dado que el proyecto tiene un carácter tanto técnico como orientado al usuario final, se han considerado riesgos tecnológicos y de aceptación.

La Tabla 2 resume estos riesgos. La mayoría de ellos se han resuelto o minimizado durante la implementación, como se ha descrito en los capítulos anteriores.

Tabla 2: Registro de riesgos del proyecto

Riesgo	**Descripción**	**Prioridad**	**Complejidad**	**Medidas correctoras**
R-01	Sensibilidad a la iluminación en la detección de color (sombras, reflejos y temperatura de color)	Alta	Media	• Ajustar rangos HSV • Configurar durante la ejecución (config.json) • Utilizar luz difusa/indirecta para minimizar reflejos
R-02	Rendimiento inferior a 15 FPS en el módulo de detección	Alta	Media	• Reducir la resolución de la imagen al 50 % antes de aplicar la transformada de Hough en la detección de la base
R-03	Oclusión de la base (manos del usuario y piezas en los bordes)	Media	Alta	• Utilizar la transformada de Hough (líneas infinitas) en lugar de contornos para identificar la base
R-04	Imposibilidad de modificar parámetros una vez empaquetado el ejecutable	Baja	Media	• Archivo «config.json» editable con cualquier editor de texto • Recarga de la configuración durante la ejecución
R-05	Contraste insuficiente entre la base y la superficie de la mesa	Alta	Baja	• Utilizar una base de color distinto al de la mesa • Si la base es del mismo color que la mesa, usar una tela de contraste
R-06	Curva de aprendizaje en la manipulación de las piezas por parte del usuario	Media	Media	• Emplear un sistema conocido e intuitivo (piezas LEGO) • Mostrar la cuadrícula en la interfaz de Python para guiar la colocación
R-07	Necesidad de calibración manual de la cámara o de la perspectiva por parte del usuario	Media	Media	• Usar una webcam estándar que no requiere calibración previa • Corrección automática de la perspectiva (vista cenital) • Umbral de ocupación de celda que compensa pequeños desajustes de perspectiva y evita falsos positivos
R-08	Aceptación de la interfaz por parte del usuario	Media	Media	• Detección: retroalimentación visual clara (círculos y contorno) para verificar la detección • Demo: interfaz con cruceta de cámaras, iconos intuitivos y diversas funcionalidades

11. Aspectos relevantes del desarrollo

Se ha presentado de forma general la herramienta en el Capítulo 9 (Descripción de la solución propuesta). En este capítulo se profundiza en los detalles de implementación de cada módulo, las decisiones de diseño adoptadas, los problemas técnicos encontrados durante el desarrollo y las soluciones aplicadas.

Los diagramas de flujo, explicaciones y capturas del módulo de detección, módulo de visualización y aplicación de prototipado de niveles se incluyen en los Anexos B, C y D, respectivamente. Por lo tanto, este capítulo se centrará en los aspectos relevantes del desarrollo y expondrá de manera conjunta los resultados de las métricas obtenidas del módulo de detección y de la aplicación de demostración.

11.1. Arquitectura de la implementación

La arquitectura general del sistema ya se presentó en el Capítulo 9 (Descripción de la solución propuesta). En este apartado se profundiza en la organización interna de los dos módulos principales.

El módulo de detección, escrito en Python implementando la librería OpenCV, se estructura en seis archivos independientes:

- **main.py**: punto de entrada del programa, contiene el bucle principal de procesado de fotogramas.
- **config.py**: centraliza los parámetros y gestiona la configuración externa mediante un archivo JSON.
- **base_detector.py**: se encarga de la detección de la base mediante la transformada de Hough y K-Means. Además, de una serie de procesamientos de los fotogramas capturados por la webcam.
- **base_warp.py**: corrección de la perspectiva de la base para compensar posibles inclinaciones suyas o de la webcam.
- **color_detector.py**: identificación de piezas por color con máscaras HSV y una cuadrícula.
- **udp_sender.py**: envío de los datos de las piezas (color y posición) en formato JSON por protocolo UDP.

El módulo de visualización, implementado en C++ para Unreal Engine, se divide en dos clases:

- **UDPReceiver**: clase de tipo actor. Actúa como receptor asíncrono de datagramas UDP en un hilo secundario. Parsea el archivo JSON recibido del módulo de detección y notificación al componente y a los BPs.
- **PieceSpawnerComponent**: contiene el mapa configurable que relaciona colores y actores. Actualiza de forma diferencial la escena, convirtiendo las coordenadas de celda a posiciones 3D en el mapa.

Los detalles completos de estos archivos y clases, además de sus diagramas de flujo se encuentran en los Anexos B (Módulo de detección) y C (Módulo de visualización).

11.2. Módulo de detección

A continuación se describen los aspectos más relevantes de la implementación del módulo de detección, desarrollado en Python con la librería OpenCV. Se explica el algoritmo de localización de la base, la identificación de piezas por color y posición, la comunicación UDP, el sistema de configuración externa, así como las optimizaciones de rendimiento y los problemas de iluminación encontrados. El diagrama de flujo y otras capturas del proceso pueden consultarse en el Anexo B.

11.2.1. Algoritmo de detección de la base

Localizar la base de forma fiable, incluso cuando manos o piezas ocultan parte de su contorno, fue uno de los primeros desafíos. El algoritmo implementado identifica los bordes de la imagen recibida de la webcam, para después comprobar su perpendicularidad y área, y así obtener la base.

Para detectar los bordes se escogió la transformada de Hough estándar (HoughLines) porque proporciona líneas infinitas, lo que permite intersectar los bordes aunque no sean continuos. Para separar las líneas en dos grupos (horizontales y verticales) sin umbrales fijos de ángulo, se aplicó el algoritmo K-Means con k=2. Esto permite que el sistema funcione con cualquier orientación de la base. A partir de las líneas agrupadas, se generan todas las combinaciones de dos líneas de cada grupo y se comprueba su perpendicularidad. A continuación se calculan las intersecciones para obtener las cuatro esquinas, descartando aquellos rectángulos que tengan al menos una fuera de la imagen. Entonces se selecciona el cuadrilátero de mayor área que supere un umbral mínimo.

11.2.2. Detección de color y posición

La identificación de las piezas se realiza en el espacio HSV, más robusto frente a cambios de iluminación, y sobre una imagen de la base recortada y con la perspectiva cenital corregida. En el archivo de configuración se define una lista «COLORS» que contiene información para cada color: nombre, rango inferior de detección HSV, rango superior HSV y su color representativo en la interfaz Blue, Green, Red (BGR), OpenCV emplea este formato en lugar de RGB.

Para cada color definido en la configuración se genera una máscara binaria en la que los píxeles blancos representan zonas en las que se ha detectado el color. Esta detección tiene en cuenta la naturaleza circular del tono (H) en el espacio de color, los valores extremos comparten color. Colores como el rosa tienen un rango circular donde el límite inferior es mayor que el superior. El programa divide el intervalo en dos máscaras que luego combina.

Para poder de distinguir los contornos de las piezas, incluso de un mismo color, se decidió subdividir la base en una cuadrícula de 16x16 celdas. Teniendo en cuenta que la base de LEGO tiene 32x32 studs, cada celda se corresponde con una pieza de 2x2 studs, las más comunes.

En cada celda se cuenta el número de píxeles de cada máscara y se selecciona el color que supere un umbral de ocupación del 60 %. Este umbral filtra ruido y compensa pequeños desajustes de perspectiva, evitando falsos positivos y garantizando que se detecta el color adecuado. El resultado es una lista de piezas con su color y coordenadas de celda.

11.2.3. Comunicación UDP

Para comunicarse con el motor de videojuegos, emplea el protocolo UDP porque se prioriza la velocidad de envío por encima de la fiabilidad, como sería el caso del Protocolo de Control de Transmisión (TCP). Primero se serializan los mensajes en formato JSON, contienen el nombre del color y las coordenadas x e y de la celda. A continuación se envía, incluso cuando haya piezas (lista vacía), esto permite que el motor de videojuegos gestione la falta de piezas en la base y pueda sincronizar correctamente la escena al mundo real.

11.2.4. Configuración externa

El módulo tiene varios parámetros ajustables: la lista de colores, el número de divisiones de la cuadrícula, la dirección IP, etc. Todos ellos se guardan en un archivo «config.json» que el programa crea automáticamente si no existe. Esto permite configurar las variables del ejecutable (.exe) sin necesidad de modificar el código o

recompilar. Además, la configuración puede recargarse durante la ejecución pulsando la tecla `R`, sin necesidad de reiniciar el programa. Antes de implementar los cambios valida el contenido del JSON para evitar errores de sintaxis, falta de campos o tipos de datos equivocados, en estos casos recarga los valores por defecto.

Para facilitar la definición de los rangos de color, se desarrolló una herramienta auxiliar de cuentagotas (descrita en el Anexo B). Muestra los valores BGR y HSV del píxel seleccionado sobre la imagen de la webcam, simplificando el ajuste de los parámetros de detección.

11.2.5. Optimización

Al realizar las pruebas de rendimiento se observó un cuello de botella en la detección de la base. La transformada de Hough tiene un coste computacional elevado, que aumenta considerablemente con la resolución del fotograma. Esto provocaba un bajo rendimiento de FPS que se solventó redimensionando la imagen procesada a la mitad de su tamaño y realizar la detección de la base sobre ella. Con esta optimización, la tasa de FPS se multiplicó aproximadamente por tres.

En cuanto a la iluminación, las sombras oscurecen el color y se compensaron ajustando los rangos. Los reflejos especulares, sin embargo, resultan más problemáticos porque convierten el color de la pieza en blanco y dificultan la detección en esas zonas. Esta limitación es inherente a la superficie brillante de las piezas de LEGO y se mitiga mediante una iluminación difusa.

11.3. Módulo de visualización

El módulo de visualización se ha implementado en C++ para Unreal Engine. Se compone de dos clases principales: «UDPReceiver» (receptor UDP) y «PieceSpawnerComponent» (instanciador de actores). A continuación se describe el funcionamiento de cada una. El diagrama de flujo y los detalles del código se encuentran en el Anexo C.

11.3.1. Receptor UDP

La clase «UDPReceiver» abre un socket en un hilo secundario y espera datagramas. Cuando llega un mensaje, un callback limpia la cadena (elimina caracteres residuales) y programa una tarea en el hilo principal para parsear el JSON. Si el parseo es correcto y el objeto contiene un array llamado «pieces», se lanza el evento «OnPiecesReceived» para notificar a los BPs y se invoca al componente «PieceSpawnerComponent» para actualizar la escena. Esta arquitectura asíncrona evita bloquear el hilo principal del juego con la escucha continua del socket.

11.3.2. Instanciación de actores

«PieceSpawnerComponent» contiene el mapa «ColorToActor» configurable desde el editor, donde cada color puede asociarse a una clase de actor (o a «None» para desactivarlo). Se pueden añadir, modificar o eliminar elementos a este mapa fácilmente desde el editor. Cuando recibe un nuevo array de piezas, compara con el estado actual y determina qué celdas han cambiado. Solo destruye actores en celdas que han quedado vacías, y para las piezas nuevas o modificadas consulta la clase correspondiente. Si la clase es distinta, destruye el actor anterior e instancia un actor en la posición calculada a partir de las coordenadas de celda (convertidas a coordenadas del mundo). En cambio, si la clase es la misma, no hace nada. Esta actualización diferencial minimiza el coste computacional. Además, expone los parámetros «CellSize» y «SpawnHeight» para que el desarrollador pueda configurar el tamaño de la celda y la altura de spawneo desde el editor, sin necesidad de modificar el código fuente.

11.3.3. Calibración dinámica de colores (descartada)

Se desarrolló un prototipo que permitía ajustar los rangos HSV de detección de colores desde el editor de Unreal Engine y enviarlos a Python mediante comunicación UDP bidireccional. Esto permitía añadir, modificar y eliminar colores desde el editor, además de usar un selector de colores para asignar los rangos. Aunque funcionaba, se descartó por la complejidad añadida, la posible limitación de flexibilidad de la herramienta y que la selección visual de los colores no era intuitiva porque Unreal Engine trabaja en el espacio de color RGB. Los detalles de este prototipo se exponen en el Anexo C.

11.3.4. Integración en motores de videojuegos

El módulo de detección envía los datos mediante UDP en formato JSON, un estándar ligero y fácil de parsear. Cualquier motor de videojuegos (Godot, Unity, etc.) que pueda abrir un socket UDP e interpretar JSON puede implementar un módulo de visualización equivalente. Por tanto, el sistema es independiente del motor utilizado. En este trabajo se ha desarrollado para Unreal Engine como prueba de concepto.

Además, el módulo de visualización de Unreal Engine es fácilmente reutilizable en otros proyectos del motor, se detalla el proceso que se debe seguir en el Anexo E (Manual de usuario). En el editor se exponen ciertos parámetros modificables de conexión UDP e instanciación de actores, que permite a diseñadores sin conocimientos de C++ adaptar la herramienta a sus necesidades.

11.3.5. Aplicación de demostración (prototipado de niveles)

Para demostrar la utilidad de la herramienta se desarrolló una aplicación en Unreal Engine que permite construir un nivel de forma tangible. En el Anexo D se pueden encontrar el diagrama de navegación, las capturas de la aplicación y el logotipo de la herramienta, que ha sido nombrada «ChromaVision». La aplicación tiene dos modos.

En el modo construcción, el usuario coloca piezas LEGO y la aplicación muestra los actores 3D correspondientes. Desde la interfaz se pueden cambiar las vistas de cámara (incluyendo una vista cenital rotable para alinear la orientación con la webcam real), pausar la recepción UDP, borrar todos los actores, reasignar la conversión color/actor y ajustar el volumen.

En el modo juego, el usuario controla un personaje en primera persona que puede recorrer el nivel, interactuar con los objetos (plataformas móviles, impulsores, trampas, etc.), recibir daño y alcanzar el portal para ganar. La tecla `F` permite volver al modo construcción.

Entre las funcionalidades más relevantes destaca el spawneo del personaje mediante una pieza, es posible asociar la clase del personaje a un color determinado. Entonces al colocar en la base una pieza de ese color, la celda correspondiente se convierte en el punto de inicio del nivel en el modo juego. Además, la conexión UDP se detiene automáticamente al entrar en el modo juego para evitar modificaciones no deseadas del nivel, y permanece desactivada al regresar al modo construcción, hasta que el usuario la reactive manualmente.

Por otro lado se modelaron, texturizaron y programaron siete tipos de actores de ejemplo para prototipar niveles: bloque, muro, plataforma móvil, portal, impulsor, trampa de pinchos y pasarela.

11.4. Resultados de las métricas y pruebas

En este apartado se presentan los resultados de las pruebas de rendimiento (FPS) del módulo de detección y de la aplicación de demostración, así como la validación de la robustez del sistema frente a oclusiones parciales y diferentes condiciones lumínicas. Los imágenes de las pruebas lumínicas pueden consultarse en el Anexo B.

11.4.1. Rendimiento conjunto del sistema

Se midió la tasa de FPS del módulo de detección y de la aplicación de ejemplo en una sesión de un minuto con manipulación normal de las piezas. Los resultados conjuntos se muestran en la Tabla 3.

Tabla 3: Rendimiento conjunto del sistema

Elemento	FPS promedio	FPS máximo	FPS mínimo
Módulo de detección (Python y OpenCV)	21,72	29,51	16,72
Aplicación de demostración (Unreal Engine)	115,27	120,05	98,74

El módulo de detección supera el mínimo de 15 FPS establecido como aceptable en los requisitos no funcionales del Anexo A. Por lo tanto, la actualización de los datos de las piezas es suficientemente rápida. Además, el pico máximo alcanza los 29,51 FPS, acercándose al límite teórico de la webcam (30 FPS), aunque la media se sitúa en 21,72 FPS.

Por otro lado, la aplicación de ejemplo alcanza una media de 115 FPS, demostrando que la comunicación UDP y la actualización de actores no introducen una carga significativa en el motor. El mínimo registrado de 98,74 FPS sigue siendo elevado y no afecta a la experiencia de usuario. Cabe destacar que ambas métricas se evaluaron en el mismo dispositivo pero pueden variar los resultados dependiendo del hardware empleado.

11.4.2. Otras pruebas realizadas

Se verificó la correcta detección de la base incluso con oclusiones parciales (manos o piezas en los bordes), gracias al uso de líneas infinitas en la transformada de Hough. También se calibraron y probaron los rangos de color bajo tres condiciones lumínicas: luz natural, luz artificial de techo y luz indirecta con tono cálido procedente de un flexo (Anexo B). Todos los colores (rojo, amarillo, azul, verde, negro, naranja y rosa) se detectaron correctamente, aunque los colores adyacentes (rojo, naranja y rosa) resultaron más conflictivos con luz cálida. La detección de la base se mostró igualmente robusta frente a sombras y reflejos.

12. Gestión del proyecto

En este capítulo se describe la gestión del proyecto, incluyendo la distribución temporal real de las tareas y los recursos empleados. Para el control de versiones del código se ha utilizado Git y GitHub, lo que ha permitido mantener un registro ordenado de los cambios y facilitar la trazabilidad del desarrollo.

12.1. Recursos

Los recursos utilizados en el proyecto se pueden agrupar en tres categorías:

- **Recursos software**: las herramientas empleadas (Python, OpenCV, Unreal Engine, Blender, Krita, etc.) se describen en la Sección 5.2.
- **Recursos materiales**: el hardware y materiales necesario (ordenador personal, webcam, base y piezas LEGO) se detalla en la Sección 8.3, donde también se indica su coste aproximado.
- **Recursos personales**: el tiempo dedicado al proyecto ha sido un recurso fundamental. Se han invertido nueve meses en el desarrollo, distribuidos entre la investigación de alternativas tecnológicas, la implementación de los módulos de la herramienta y la aplicación de demostración. Además, ha sido necesario dedicar tiempo y esfuerzo al aprendizaje de tecnologías que no se dominaban al principio, como OpenCV (visión artificial) y LaTeX (para la redacción de la memoria). Este aprendizaje ha sido paralelo al desarrollo y ha condicionado el ritmo de las primeras fases del proyecto.

12.2. Planificación temporal

La distribución temporal del trabajo se ha organizado siguiendo las fases descritas en la Sección 5.1. A continuación se detallan las principales fases, que se presentan en el diagrama de Gantt de la Figura 25.

- **Fase inicial (octubre – febrero)**: estudio de alternativas tecnológicas y aprendizaje de la sintaxis básica de OpenCV en Python. Esta fase sentó las bases para la elección e implementación de la solución final.
- **Módulo de detección (febrero – abril)**: Implementación secuencial de las distintas partes de la detección de piezas de colores, incluyendo la base. Esta fase fue extensa debido a los ajustes y pruebas de precisión necesarios para garantizar una detección robusta.

- **Módulo de visualización (abril – mayo)**: desarrollo del receptor UDP y del componente de instanciación de actores en Unreal Engine (C++). Esta fase se solapó parcialmente con el diseño de la aplicación de ejemplo.
- **Aplicación de demostración (mayo)**: creación de los tipos de actores, implementación de los modos construcción y juego, y diseño de las interfaces. Fue una fase intensa debido a la combinación de programación y del trabajo artístico.
- **Revisión final y empaquetado (junio)**: refactorización de código, revisión de comentarios, ajuste de rangos de color, empaquetado del módulo de detección con PyInstaller y de la aplicación.
- **Memoria**: se tomaron notas de forma continua a lo largo el proyecto. En el mes de abril se dedicó un periodo a la preparación de la memoria en LaTeX. La redacción y revisión final se concentró en mayo y junio, una vez completada la mayor parte del desarrollo.

El diagrama de Gantt (Figura 25) muestra la distribución temporal de estas tareas desde octubre hasta junio. Se siguió un desarrollo secuencial con solapamientos puntuales entre fases.

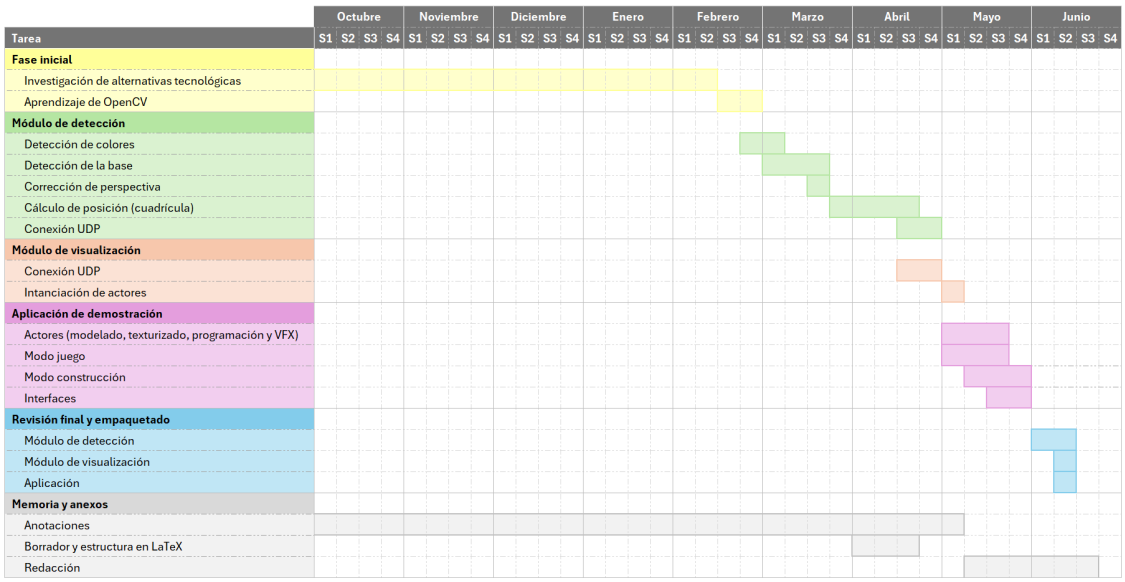

Figura 25: Diagrama de Gantt, planificación temporal del proyecto

13. Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones del trabajo, evaluando el cumplimiento de los objetivos planteados y reflexionando sobre el proceso de desarrollo. También se exponen las principales limitaciones del sistema y se proponen líneas de trabajo futuro para su mejora y ampliación.

13.1. Conclusiones

Se puede concluir que el presente Trabajo de Fin de Grado ha cumplido satisfactoriamente los objetivos planteados en el Capítulo 2.

Se ha desarrollado «ChromaVision», una herramienta modular basada en visión artificial que permite detectar piezas LEGO por color y posición, y visualizarlas en el entorno de un motor de videojuegos gracias a una comunicación UDP. Además, para demostrar su funcionamiento se ha creado una aplicación de prototipado de niveles en Unreal Engine.

El sistema cumple todos los objetivos funcionales:

- La herramienta es capaz de identificar las piezas mediante su color y calcular su posición utilizando visión artificial (OpenCV).
- El módulo de detección y el motor de videojuegos (módulo de visualización) establecen una conexión mediante el protocolo UDP, que utilizan para transmitir información de las piezas identificadas (color y posición).
- El módulo de detección es independiente del motor de videojuegos, gracias a la conexión mediante UDP no es necesario modificar su lógica para que funcione con diversos motores. Cualquier motor que permita integrar un receptor UDP podrá escuchar los mensajes, solo requiere crear un módulo de visualización personalizado. En este trabajo se programó el correspondiente a Unreal Engine, reutilizable en distintos proyectos.
- La manipulación de las piezas se refleja con una velocidad adecuada en el entorno 3D. El módulo de detección tiene un promedio de 21,72 FPS, llegando incluso a alcanzar 29,52 FPS, casi el límite de 30 FPS de la webcam. Además la aplicación de Unreal Engine alcanza un promedio de 115,27 FPS.
- Se ha desarrollado una aplicación de prototipado de niveles que demuestra el funcionamiento de la herramienta. Si bien se planteó la posibilidad de desarrollar una aplicación de pixel art en Godot (entorno 2D) para demostrar la

compatibilidad de la herramienta con otros motores, debido al tiempo limitado se priorizó la demostración de Unreal Engine.

También se han completado los requisitos funcionales y no funcionales expuestos en el Anexo A: la detección de la base es automática sin requerir calibración, se puede asignar qué actor corresponde a cada color, el módulo de detección supera los 15 FPS mínimos establecidos, etc.

13.2. Reflexión final

El cumplimiento de ciertos objetivos supuso un reto. Por ejemplo, inicialmente el módulo de detección no tenía el rendimiento mínimo necesario, pero se solventó optimizando y refactorizando su código. Además, ofrecer la posibilidad de configurar los rangos de detección de los colores se planteó originalmente mediante la exposición de los rangos en el propio motor de videojuegos con una comunicación bidireccional. Sin embargo, al final resultó más eficiente y flexible la creación automática de un archivo JSON en el propio módulo de detección. De esta forma no depende del motor, está accesible en el programa empaquetado y permite la recarga durante la ejecución (la explicación de esta decisión se puede leer en el Anexo B). Para facilitar esta configuración durante el desarrollo, se creó una herramienta auxiliar de cuentagotas que mostraba los valores BGR y HSV del píxel seleccionado.

El principal obstáculo fue la detección robusta de la base, especialmente cuando manos o piezas ocultaban parte de su contorno. La solución adoptada, basada en la transformada de Hough con líneas infinitas, requirió varias iteraciones de prueba y error hasta lograr un resultado fiable. además, los rangos de color se ajustaron a lo largo del desarrollo hasta obtener unos valores con los que se estuviera conforme.

En cuanto al aprendizaje, este TFG me ha servido para adentrarme en el campo de la visión artificial y, en particular, en la librería OpenCV, que no había utilizado previamente. También he aprendido a utilizar LaTeX para la redacción de la memoria, lo que ha facilitado la organización del documento y la gestión de las referencias bibliográficas. Me ha resultado especialmente interesante combinar estos conocimientos con el desarrollo de videojuegos y aplicaciones, aplicándolo a un proyecto que permite tanto una interacción tangible como digital.

13.3. Limitaciones y trabajo futuro

A pesar de que el sistema cumple con los objetivos marcados, presenta ciertas limitaciones. En primer lugar, la detección por color es sensible a cambios de iluminación. Aunque se han ajustado los rangos para obtener una detección fiable, los

reflejos especulares en superficies brillantes siguen siendo problemáticos. Por otro lado, el sistema no detecta la orientación de las piezas ni permite apilarlas en varias alturas, lo que restringe su uso en ciertos escenarios. Asimismo, la gama de colores está restringida a primarios bien diferenciados aunque se ha demostrado la correcta distinción en valores cercanos como rojos y naranjas.

A partir de estas limitaciones y de la experiencia adquirida, se plantean las siguientes líneas de trabajo futuro:

- Explorar la detección de orientación de las piezas y la posibilidad de apilarlas en varias alturas, utilizando por ejemplo la variable «SpawnHeight» para ajustar la altura de instanciación desde la interfaz.
- Desarrollar una interfaz gráfica que permita ajustar los rangos de color de forma intuitiva y visual, evitando la edición manual del archivo JSON.
- Implementar módulos de visualización para otros motores de videojuegos como Godot o Unity, aprovechando que el sistema ya envía los datos en un formato estándar (JSON) a través de UDP.
- Explorar otras aplicaciones de la herramienta, como el ámbito educativo o la creación de pixel art en Godot, una posibilidad que se planteó inicialmente como otra aplicación de demostración.
- Explorar el uso colaborativo a distancia mediante comunicación en red, aprovechando las capacidades del protocolo UDP.
- Empaquetar el módulo de visualización de Unreal Engine como un plugin para facilitar su distribución e integración en otros proyectos.

14. Bibliografía

Asociación Española de Videojuegos. (2024, 30 de junio). *La industria del videojuego en España, Anuario 2024* (Anuario). Asociación Española de Videojuegos. España. Consultado el 10 de mayo de 2026, desde https://aevi.org.es/wp-content/uploads/2025/05/VDIGITAL-ANUARIO2024-comprimido.pdf

Bluetooth SIG. (2026). *Bluetooth Technology Overview* [Bluetooth Technology Website]. Consultado el 15 de abril de 2026, desde https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Chopra, S., Kumar, D., Singh, G., & Nayyar, A. (2023, julio). USER INTERFACE: AN OVERVIEW. En *Research Trends in Science and Technology* (pp. 1-9, Vol. 1). Bhumi Publishing. Consultado el 12 de mayo de 2026, desde https://www.researchgate.net/publication/373196945_USER_INTERFACE_AN_OVERVIEW

Dix, A., Finlay, J., Abowd, G. D., & Beale, R. (2004). *Human-Computer Interaction* (3rd Edition). Pearson Education.

El Shemy, I., Jaccheri, L., Giannakos, M., & Vulchanova, M. (2025). Participatory design of augmented reality games for word learning in autistic children: the parental perspective. *Entertainment Computing*, *52*(100756). https://doi.org/10.1016/j.entcom.2024.100756

Entertainment Software Association. (2025). *Essential Facts About the U.S. Video Game Industry 2025* (Anuario). Entertainment Software Association. Consultado el 10 de mayo de 2026, desde https://www.theesa.com/resources/essential-facts-about-the-us-video-game-industry/2025-data/

Fitzmaurice, G. W., Ishii, H., & Buxton, W. A. S. (1995). Bricks: laying the foundations for graspable user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 442-449. https://doi.org/10.1145/223904.223964

Game Developer Conference. (2026). *Alt.ctrl.GDC | GDC festival of gaming.* Consultado el 11 de mayo de 2026, desde https://gdconf.com/alt-ctrl-gdc/

Greene, J. (2025). Radio-frequency identification (RFID). En *EBSCO Research Starters*. EBSCO Information Services. Consultado el 14 de abril de 2026, desde https://www.ebsco.com/research-starters/communication-and-mass-media/radio-frequency-identification-rfid

Hornbæk, K., Kristensson, P. O., & Oulasvirta, A. (2025, 6 de junio). Introduction to user interfaces. En K. Hornbæk, P. O. Kristensson & A. Oulasvirta (Eds.), *Introduction to Human-Computer Interaction* (pp. 389-404). Oxford University Press. https://doi.org/10.1093/oso/9780192864543.003.0023

Ishii, H., Lakatos, D., Bonanni, L., & Labrune, J.-B. (2012). Radical atoms: beyond tangible bits, toward transformable materials. *Interactions*, *19*(1), 38-51. https://doi.org/10.1145/2065327.2065337

Ishii, H., & Ullmer, B. (1997). Tangible bits: towards seamless interfaces between people, bits and atoms. *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, 234-241. https://doi.org/10.1145/258549.258715

Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales (2018, 7 de diciembre). Consultado el 31 de mayo de 2026, desde https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf

Jordà, S., Geiger, G., Alonso, M., & Kaltenbrunner, M. (2007). The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 139-146. https://doi.org/10.1145/1226969.1226998

JoyLabz LLC. (2026). *Makey Makey* [Makey Makey]. Consultado el 14 de mayo de 2026, desde https://makeymakey.com/

Koçer, F., & Alaçam, S. (2023). AN EVALUATION OF AUGMENTED REALITY-BASED USER INTERFACES IN THE DESIGN PROCESS. *Architecture and Planning Journal (APJ)*, *28*(3). https://doi.org/10.54729/2789-8547.1234

Liermann, E. (2025, 7 de mayo). *Spatial Interface using Open CV and Lego*. Consultado el 4 de octubre de 2025, desde https://www.erikliermann.org/project/spatial-lego

Marshall, P. (2007). Do tangible interfaces enhance learning? *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 163-170. https://doi.org/10.1145/1226969.1227004

MathWorks. (2025). *Near Field Communication (NFC)* [MATLAB & Simulink]. Consultado el 14 de abril de 2026, desde https://www.mathworks.com/help/comm/ug/near-field-communication-nfc.html

Merrill, D., Kalanithi, J., & Maes, P. (2007). Siftables: towards sensor network user interfaces. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 75-78. https://doi.org/10.1145/1226969.1226984

Merrill, D., Sun, E., & Kalanithi, J. (2012). Sifteo cubes. *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, 1015-1018. https://doi.org/10.1145/2212776.2212374

NFC Forum. (2026). *NFC Forum* [NFC Forum]. Consultado el 14 de abril de 2026, desde https://nfc-forum.org/learn/

REGLAMENTO (UE) 2016/ 679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO - de 27 de abril de 2016 - relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/ 46/ CE (Reglamento general de protección de datos) (2016, 25 de mayo). Consultado el 31 de mayo de 2026, desde https://www.boe.es/doue/2016/119/L00001-00088.pdf

Pendo Technology Ltd. (2025). *Chessnut Move – Fastest Self-Moving Robotic Chessboard* [Chessnut]. Consultado el 14 de mayo de 2026, desde https://www.chessnutech.com/pages/chessnut-move

Rapp, A. (2023). Human–Computer Interaction. En F. C. Worrell (Ed.), *Oxford Research Encyclopedia of Psychology*. Oxford University Press. https://doi.org/10.1093/acrefore/9780190236557.013.47

Roda, C. (Ed.). (2011). *Human Attention in Digital Environments*. Cambridge University Press. https://doi.org/10.1017/CBO9780511974519

Shaer, O., & Hornecker, E. (2009). Tangible User Interfaces: Past, Present and Future Directions. *Foundations and Trends in Human-Computer Interaction*, *3*(1), 1-137. https://doi.org/10.1561/1100000026

Silver, J., & Rosenbaum, E. (2012). Makey Makey: improvising tangible and nature-based user interfaces. *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, 367-370. https://doi.org/10.1145/2148131.2148219

Special Projects. (2013). *Bit Planner* [Special Projects]. Consultado el 10 de marzo de 2026, desde https://specialprojects.studio/project/bit-planner/

Tangible Play, Inc. (2026). *PlayOsmo* [PlayOsmo]. Consultado el 14 de mayo de 2026, desde https://www.playosmo.com/

Ullmer, B., & Ishii, H. (2000). Emerging frameworks for tangible user interfaces. *IBM Systems Journal*, *39*(3), 915-931. https://doi.org/10.1147/sj.393.0915

Underkoffler, J., & Ishii, H. (1999). Urp: a luminous-tangible workbench for urban planning and design. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 386-393. https://doi.org/10.1145/302979.303114

Wacom. (2021, 8 de abril). *What is the EMR (Electro - Magnetic Resonance) method incorporated in a pen tablet?* [Wacom Support]. Consultado el 14 de octubre de 2025, desde https://support.wacom.com/hc/en-us/articles/1500006270401-What-is-the-EMR-Electro-Magnetic-Resonance-method-incorporated-in-a-pen-tablet

Yamanami, T., Funahashi, T., & Senda, T. (1991, 2 de julio). *Position detecting apparatus* (pat. estadounidense N.º 5028745A). Consultado el 12 de febrero de 2026, desde https://patents.google.com/patent/US5028745A

Zhou, Y., & Wang, M. (2015). Tangible User Interfaces in Learning and Education. En J. D. Wright (Ed.), *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)* (2nd Edition, pp. 20-25). Elsevier. https://doi.org/https://doi.org/10.1016/B978-0-08-097086-8.92034-8

Anexo A

Especificaciones del sistema

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela **Politécnica Superior de Zamora**

Siglas

3D tres dimensiones. 2–5

FPS fotogramas por segundo. 6

UDP Protocolo de Datagramas de Usuario. 4, 5

Índice general

Siglas . . . . . **I**

1 Introducción **1**

2 Historias de usuario **2**

2.1 Módulo de detección de piezas 2

2.2 Módulo de visualización 2

2.3 Aplicación de ejemplo (prototipado de niveles) 3

2.4 General 3

3 Requisitos **4**

3.1 Requisitos funcionales 4

3.2 Requisitos no funcionales 6

Índice de tablas

1 Requisitos funcionales del proyecto 4
1 Requisitos funcionales del proyecto 5
2 Requisitos no funcionales del proyecto 6

1. Introducción

Este anexo contiene la documentación completa sobre los requisitos del sistema desarrollado en este TFG. Se incluye el conjunto completo de historias de usuario y las tablas detalladas con los requisitos funcionales y no funcionales.

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Historias de usuario

Las historias de usuario son una representación breve y por escrito de un requisito. Dado que el sistema desarrollado es modular, se han creado las historias de usuario de manera acorde.

2.1. Módulo de detección de piezas

- **HU-01**: Como usuario, quiero que el sistema detecte automáticamente la base sobre la que coloco las piezas, para no tener que calibrar manualmente la cámara.
- **HU-02**: Como usuario, quiero que el sistema distinga diferentes colores de piezas, para poder asignar distintos tipos de objetos a cada color.
- **HU-03**: Como usuario, quiero que el sistema calcule la posición exacta de cada pieza sobre la base, para que los objetos 3D aparezcan en el lugar correcto.
- **HU-04**: Como usuario, quiero que la interfaz sea intuitiva, para reconocer a simple vista si el sistema está detectando correctamente la base, las piezas y sus colores.
- **HU-05**: Como desarrollador, quiero que el módulo de detección sea compatible con distintos motores de videojuegos, para poder elegir el motor que prefiera.

2.2. Módulo de visualización

- **HU-06**: Como desarrollador, quiero que el módulo de visualización sea fácilmente integrable y modificable, para poder adaptarlo a la aplicación o videojuego que este creando.
- **HU-07**: Como usuario, quiero que al colocar una pieza sobre la base, el objeto 3D asignado aparezca en la posición correcta, para ver una representación fiel de lo que he construido.
- **HU-08**: Como diseñador de niveles, quiero que el objeto 3D siga el movimiento de la pieza sin latencia molesta, para poder modificar la disposición del nivel dinámicamente.
- **HU-09**: Como desarrollador, quiero poder añadir nuevos colores de pieza y ajustar sus rangos de color desde el editor del motor de videojuegos, para adaptar el sistema a mis necesidades sin modificar el código.

- **HU-10**: Como desarrollador, quiero poder redefinir qué tipo de objeto 3D se asocia a cada color de pieza, para adaptar el sistema a mis propios modelos sin tener que modificar el código fuente.

2.3. Aplicación de ejemplo (prototipado de niveles)

- **HU-11**: Como diseñador de niveles, quiero poder visualizar fácilmente y desde distintos ángulos el nivel que estoy creando, para poder tomar decisiones sobre su diseño.
- **HU-12**: Como diseñador de niveles, quiero poder jugar el nivel que he diseñado, para comprobar su jugabilidad y detectar errores de diseño.
- **HU-13**: Como jugador, quiero poder jugar el nivel diseñado con las piezas, para probar la experiencia de juego y dar retroalimentación al diseñador.
- **HU-14**: Como diseñador, quiero poder alternar entre la creación del nivel mediante piezas y un modo de juego, para probar el nivel y realizar cambios de forma ágil.
- **HU-15**: Como profesor, quiero que los alumnos puedan construir mapas con piezas y visualizarlos en 3D, para hacer tangible la enseñanza de conceptos espaciales (por ejemplo, coordenadas).

2.4. General

- **HU-16**: Como usuario, quiero que el sistema funcione con una webcam estándar y piezas comunes, para no tener que adquirir hardware especial ni caro.

3. Requisitos

Las historias de usuario presentadas en el apartado anterior han servido como base para identificar los requisitos del sistema. En los siguientes apartados se definen los requisitos funcionales y no funcionales, junto con su nivel de prioridad.

3.1. Requisitos funcionales

La Tabla 1 recoge los requisitos funcionales, organizados según el módulo al que pertenecen: detección, visualización y aplicación de ejemplo. Para cada requisito se especifica un identificador, una descripción concisa y su prioridad (baja, media o alta). Además, las abreviaturas «(M.C.)» y «(M.J.)» se refieren a los modos construcción y juego de la aplicación de ejemplo, respectivamente.

Tabla 1: Requisitos funcionales del proyecto

Requisitos Funcionales			
ID	**Módulo**	**Requisito**	**Prioridad**
RF-01	Detección	Detectar automáticamente la base	Alta
RF-02	Detección	Identificar los colores de varias piezas simultáneamente	Alta
RF-03	Detección	Calcular la posición de todas las piezas detectadas	Alta
RF-04	Detección	Transformar las coordenadas de píxeles a coordenadas relativas a la base	Media
RF-05	Detección	Enviar datos (color y posición) mediante socket UDP al motor de videojuegos	Alta
RF-06	Detección (configuración)	Gestionar los colores y sus rangos mediante un archivo de configuración, editor del motor o una interfaz	Media
RF-07	Visualización	Recibir los datos mediante socket UDP	Alta
RF-08	Visualización	Instanciar actores 3D según el color y posición recibidas	Alta
RF-09	Visualización	Actualizar la posición de los actores cuando se mueven las piezas	Alta

Tabla 1: Requisitos funcionales del proyecto

Requisitos Funcionales			
ID	**Módulo**	**Requisito**	**Prioridad**
RF-10	Visualización (configuración)	Controlar desde el editor del motor la conexión UDP y parámetros de la instanciación de los objetos 3D	Alta
RF-11	Aplicación	Crear dos modos: construcción y juego	Alta
RF-12	Aplicación (M.C.)	Visualizar los objetos 3D generados a partir de las piezas	Alta
RF-13	Aplicación (M.C.)	Ofrecer diferentes vistas de cámara (cenital, libre, etc.) para visualizar el nivel desde distintos ángulos	Media
RF-14	Aplicación (M.C.)	Permitir pausar y reanudar la conexión UDP	Baja
RF-15	Aplicación (M.C.)	Herramienta para borrar los objetos 3D generados	Baja
RF-16	Aplicación (M.C.)	Menú para poder reasignar qué objeto 3D se asocia a cada color	Alta
RF-17	Aplicación (M.J.)	Controlar un personaje en primera o tercera persona para recorre el nivel e interactuar con los actores	Alta
RF-18	Aplicación (M.J.)	Incluir condiciones de victoria y de derrota	Media
RF-19	Aplicación	Alternar entre el modo construcción y el modo juego con facilidad	Alta

3.2. Requisitos no funcionales

De forma análoga, la Tabla 2 presenta los requisitos no funcionales, clasificados por categoría: rendimiento, usabilidad, accesibilidad, compatibilidad y mantenibilidad. Además, para cada requisito se especifica su identificador, descripción y prioridad.

Tabla 2: Requisitos no funcionales del proyecto

Requisitos no funcionales			
ID	Categoría	Requisito	Prioridad
RNF-01	Rendimiento	La aplicación debe mantener al menos 15 FPS estables en condiciones normales de uso (manipulación de piezas y modo juego)	Alta
RNF-03	Usabilidad	Las interfaces del módulo de detección deben permitir reconocer visualmente la correcta detección de la base, las piezas y sus colores	Media
RNF-04	Accesibilidad	El sistema debe funcionar con hardware estándar (webcam convencional y piezas de LEGO comunes)	Media
RNF-05	Compatibilidad	El módulo de detección debe poder comunicarse con distintos motores de videojuegos	Alta
RNF-06	Mantenibilidad	El módulo de detección debe ser independiente del módulo de visualización	Alta

Anexo B

Módulo de detección

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela **Politécnica Superior de Zamora**

Glosario

Canny Algoritmo de detección de bordes ampliamente utilizado en visión artificial. 12–14, III, V

K-Means Algoritmo de agrupamiento (clustering) que divide datos en k grupos según su similitud. 2, 16, 17, III, V

Método de Otsu Algoritmo para calcular automáticamente un umbral de binarización en una imagen. 13, V

Studs Los pequeños salientes cilíndricos de las piezas LEGO que permiten encajarlas. 5, 13, 24

Transformada de Hough Técnica de procesamiento de imágenes para detectar líneas rectas a partir de bordes, representándolas en coordenadas polares. 2, 11, 14–16, 18–20, 32, III, V

Visión artificial Campo de la inteligencia artificial que permite a las computadoras interpretar y procesar imágenes y vídeos. 1, I

Siglas

3D tres dimensiones. 27

BGR Blue, Green, Red. 5, 22, 26, V

FPS fotogramas por segundo. 9, 32, 33

HSV Hue, Value, Saturation. 2, 5, 22, 26, V

IP Internet Protocol. 2, 6

JSON JavaScript Object Notation. 2, 5–7, 27, V

RGB Red, Green, Blue. 5

ROI región de interés. 24

TCP Protocolo de Control de Transmisión. 27

UDP Protocolo de Datagramas de Usuario. 1–3, 6, 8–10, 24, 27, IV, V

Índice general

Glosario ..... **I**

Siglas ..... **II**

1 Introducción **1**

2 Organización de los archivos **2**

3 Flujo general del módulo **3**

4 Sistema de configuración **5**

4.1 Parámetros configurables 5

4.2 Funcionamiento del código 7

4.2.1 Recarga durante la ejecución 7

5 Main y bucle principal **9**

5.1 Inicialización 9

5.2 Bucle principal 9

5.3 Visualización 10

5.4 Finalización 10

6 Detección de la base **11**

6.1 Proceso de detección 11

6.1.1 Fotograma inicial 11

6.1.2 Reducción de resolución 12

6.1.3 Escala de grises y filtro bilateral 12

6.1.4 Binarización de Otsu y cierre morfológico 13

6.1.5 Detección de bordes con Canny y dilatación 14

6.1.6 Transformada de Hough 15

6.1.7 Filtrado de líneas duplicadas 16

6.1.8 Agrupación por ángulo con K-Means 16

6.1.9 Combinaciones perpendiculares y esquinas 18

6.1.10 Dibujo del resultado 18

6.2 Método de identificación de la base 19

7 Corrección de perspectiva **21**

7.1 Funcionamiento del código 21

8 Detección de color y posición **22**

8.1 Máscaras de color 22

8.1.1 Rangos circulares 22

8.2 Detección de la posición 24

8.2.1 Elección del método de detección 24

8.3 Visualización de los resultados . . . 25
8.4 Herramienta auxiliar de calibración . . . 26

9 Envío UDP . . . 27
9.1 Formato del mensaje . . . 27
9.2 Gestión de la ausencia de piezas . . . 27

10 Empaquetado . . . 28

11 Interfaces . . . 29
11.1 Ventana «Base» . . . 29
11.2 Ventana «Piezas» . . . 29
11.3 Redimensionado de las ventanas . . . 31

12 Métricas y pruebas realizadas . . . 32
12.1 Medición del rendimiento . . . 32
12.1.1 Método de medición . . . 32
12.1.2 Resultados . . . 32
12.2 Pruebas de iluminación . . . 33

Índice de figuras

1 Diagrama del flujo general del módulo de detección 4

2 Parámetros configurables en el archivo «config.json» con los valores por defecto 6

3 Fotograma inicial capturado por la webcam sin postprocesado 11
4 Fotograma en escala de grises y con un filtro bilateral 12
5 Fotograma binarizado con el método de Otsu y con una operación morfológica de cierre 13
6 Fotograma con los bordes detectados mediante Canny y posteriormente dilatados 14
7 Líneas detectadas por la transformada de Hough 15
8 Líneas únicas obtenidas tras eliminar las duplicadas de la Figura 7 16
9 Líneas agrupadas en dos clústeres (verde y rojo) según su orientación mediante K-Means 17
10 Evolución del número de líneas a lo largo de las etapas de filtrado 18
11 Resultado final de la detección de la base sobre el fotograma original, con las esquinas numeradas según el orden del algoritmo. 19
12 Detección de la base con rotación y oclusión parcial causada por piezas en el borde y las manos del usuario 20

13 Corrección de perspectiva. Izquierda: fotograma con la base inclinada. Derecha: imagen con la perspectiva corregida y la detección de piezas . . . 21

14 Máscaras de color generadas a partir de la imagen con la perspectiva corregida para los siete colores 23
15 Detección de piezas con cuadrícula de 32x32 celda, opción descartada . . . 25
16 Ventana «Piezas» mostrando la imagen con la cuadrícula de 16×16 celdas y las piezas detectadas 25
17 Herramienta auxiliar de cuentagotas. Muestra los valores BGR y HSV del píxel seleccionado sobre la pieza roja 26

18 Ejemplo del mensaje JSON enviado por UDP con la información de las piezas detectadas en la Figura 16 27

19 Ventana «Base» cuando la base ha sido detectada 29
20 Ventana «Piezas» en ambos estados de detección de la base 30
21 Ventana «Piezas» sin la cuadrícula, acompañada del mensaje de confirmación de la recarga de la configuración 30

22 Detección de piezas bajo las tres condiciones lumínicas evaluadas. De izquierda a derecha: luz natural, luz artificial de techo y luz artificial indirecta 34
23 Efecto de las sombras y los reflejos sobre la detección de piezas 34

Índice de tablas

1 Comparativa de rendimiento antes y después de la optimización 33

1. Introducción

El módulo de detección se encarga de capturar y procesar las imágenes de la webcam para localizar la base y las piezas situadas dentro de ella, identificando su color. Posteriormente envía la posición y color de las piezas mediante el protocolo UDP al motor de videojuegos (módulo de visualización). Está desarrollado en Python y utiliza la librería de visión artificial OpenCV.

En este anexo se expone en profundidad su implementación, detallando los pasos de procesado, las decisiones de diseño adoptadas y las métricas evaluadas.

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Organización de los archivos

El código fuente del módulo de detección está compuesto por seis archivos Python. Cada uno tiene una responsabilidad claramente definida para favorecer la modularidad y el mantenimiento.

- **config.py**: centraliza los parámetros configurables del sistema (rangos de color Hue, Value, Saturation (HSV), factor de escala de las ventanas, divisiones de la cuadrícula, conexión UDP, etc.) en un archivo JavaScript Object Notation (JSON). Se encarga de crear el archivo «config.json» al ejecutar la aplicación para que el usuario pueda editar estos ajustes durante la ejecución, sin necesidad de recompilar. Además de cargar la configuración, valida los datos introducidos y la estructura del archivo; en caso de que no sean válidos o que no sea un diccionario JSON, carga los valores por defecto.

- **main.py**: es el punto de entrada del programa. Inicializa la webcam y gestiona el bucle principal de captura de fotogramas y llamada a las funciones del resto de archivos. Este bucle captura el fotograma, detecta la base, aplica la corrección de perspectiva, detecta las piezas (color y posición) y finalmente envía estos datos por protocolo UDP. Además, gestiona la visualización de las ventanas que muestran la base y las piezas detectadas. Permite cerrarlas pulsando la tecla `Escape` y recargar la configuración pulsando `R`.

- **base_detection.py**: contiene la lógica para identificar la base en la imagen. Implementa un algoritmo basado en la transformada de Hough para detectar líneas rectas, agruparlas (algoritmo K-Means) y encontrar el rectángulo de mayor área.

- **base_warp.py**: contiene una función auxiliar que aplica una transformación de perspectiva a la imagen. Recibe las cuatro esquinas de la base detectada y genera una nueva imagen con vista cenital. Eliminando distorsiones causadas por una posible inclinación de la webcam o de la base.

- **color_detection.py**: se encarga de la detección y clasificación de las piezas. Recibe la imagen corregida de la base y, mediante máscaras de color en el espacio HSV y una cuadrícula fija, determina el color predominante en cada celda. Devuelve una lista de piezas detectadas con su color y coordenadas.

- **udp_sender.py**: implementa una clase para la comunicación por sockets UDP. Su función es formatear la lista de piezas detectadas (color y posición) en un diccionario JSON. Para enviarla a la dirección Internet Protocol (IP) y puerto especificados en la configuración.

3. Flujo general del módulo

El diagrama de flujo de la Figura 1 ilustra el pipeline completo del módulo de detección. Se ha utilizado notación estándar: óvalos para el inicio y fin del programa (amarillo), rectángulos para los procesos (morado) y rombos para las decisiones (verde). Las referencias en cursiva indican el archivo fuente donde se implementa cada etapa.

El flujo comienza con la carga de la configuración desde «config.json» (si el archivo no existe, se crea automáticamente con los valores por defecto). A continuación, se inicializa la webcam, en caso de que no esté conectada, el programa finaliza. Si el acceso es correcto, se instancia el socket UDP.

Se entra entonces en el bucle principal, donde cada fotograma capturado se somete al algoritmo de detección de la base. Si la base es detectada, se aplica la corrección de perspectiva y se generan las máscaras de color. Con ellas, las piezas se clasifican y localizan mediante la cuadrícula. Acto seguido, se envían los resultados al motor de videojuegos mediante el socket UDP. Si no se detecta la base, la ventana «Base» muestra la visión de la webcam con un mensaje de aviso y la ventana «Piezas» muestra una imagen de aviso.

En ambos casos, las ventanas «Base» y «Piezas» se actualizan antes de comprobar las teclas pulsadas. El programa finaliza al presionar `Esc` o cerrar las ventanas, y permite recargar la configuración desde «config.json» con la tecla `R`. Antes de finalizar libera la cámara, cierra el socket y destruye las ventanas.

En los siguientes capítulos se detalla el funcionamiento interno de cada una de las etapas clave del proceso.

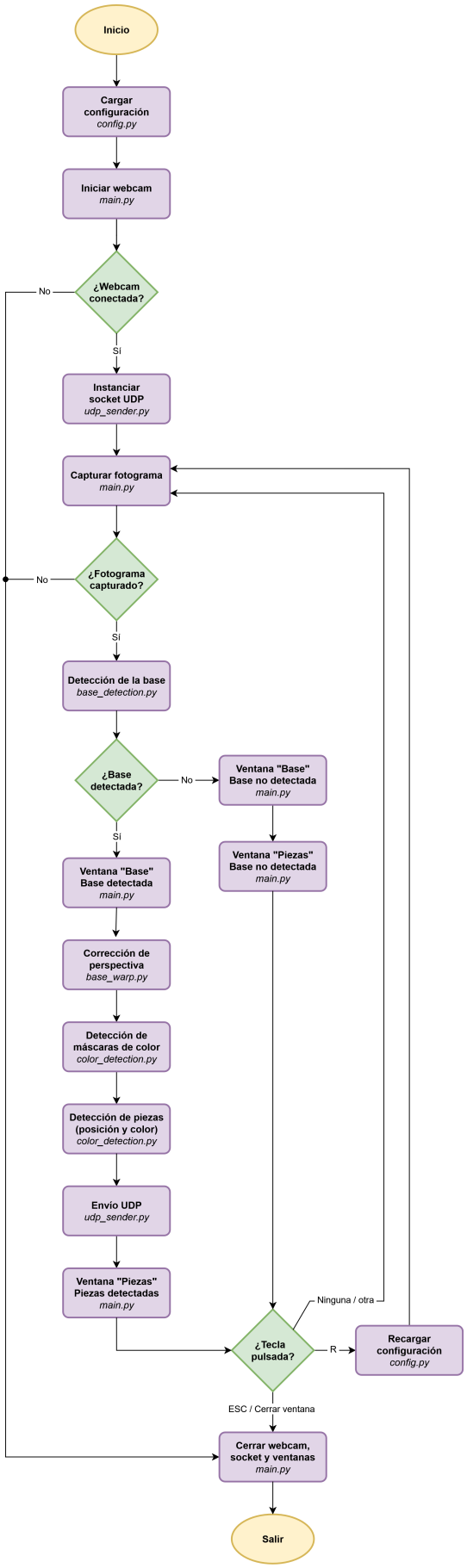

Figura 1: Diagrama del flujo general del módulo de detección

4. Sistema de configuración

Uno de los requisitos funcionales de este módulo fue la capacidad de poder modificar sus parámetros de configuración sin necesidad de alterar el código fuente. Para ello, se implementó un sistema de configuración externa basado en un archivo «config.json» gestionado por «config.py».

Este archivo centraliza todos los valores ajustables del sistema en un formato estándar, legible y editable con cualquier editor de texto. El formato JSON cumple estas necesidades y, al tener una estructura definida, facilita la comprobación de errores de escritura por parte del usuario o parámetros faltantes. De esta forma, con solo recargar la configuración durante la ejecución, se pueden modificar las variables para ajustar los rangos de color o adaptar la herramienta a diferentes condiciones de uso.

4.1. Parámetros configurables

El archivo «config.json» contiene un diccionario con los parámetros configurables del módulo de detección y los valores por defecto mostrados en la Figura 2:

- **COLORS**: lista de colores detectables. Cada color se define por un nombre, los límites inferior y superior en el espacio HSV y el valor Blue, Green, Red (BGR) utilizado para dibujarlo en la interfaz (OpenCV utiliza BGR en lugar de RGB). Gracias al soporte de rangos circulares (explicado en el Capítulo 8), es posible definir colores cuyo tono se sitúa en ambos extremos del rango 0-179, por ejemplo el color rosa.

- **WINDOW_SCALE**: factor de escala para redimensionar las ventanas de visualización de la base y las piezas detectadas. Un valor de 1.0 muestra las imágenes a su tamaño original, mientras que valores inferiores las reducen.

- **WARP_OUTPUT_SIZE**: tamaño en píxeles del lado más largo de la imagen resultante de la corrección de perspectiva de la base. En este trabajo se emplea una base cuadrada de LEGO pero puede tener otra relación de aspecto y ajustarse las divisiones de la cuadrícula modificando GRID_WIDTH y GRID_HEIGHT.

- **GRID_WIDTH y GRID_HEIGHT**: número de divisiones de la cuadrícula para la detección de piezas. Con valores 16×16, cada celda representa 2×2 studs de la base real.

- **PERCENT_FILLED_CELL**: porcentaje mínimo de píxeles de un color en una celda para considerarla ocupada (por defecto 0.6, es decir, el 60 %). Este umbral compensa pequeños desajustes de perspectiva y evita falsas detecciones por ruido en las máscaras de color.
- **MIN_BASE_AREA_RATIO**: porcentaje mínimo del área total de la imagen que debe ocupar el rectángulo de la base detectada. Filtra detecciones de objetos pequeños. El sistema siempre selecciona, de entre los válidos, el rectángulo de mayor área.
- **SHOW_PIECES_GRID**: booleano que controla la visualización de la cuadrícula en la ventana «Piezas».
- **UDP_IP y UDP_PORT**: dirección IP y puerto de destino para el envío de datos al motor de videojuegos mediante UDP.

Al estar en formato JSON, el archivo puede editarse con cualquier editor de texto, lo que permite ajustar los parámetros sin necesidad de instalar un entorno de desarrollo ni recompilar el ejecutable.

config.json > ...

```json
{
    "COLORS": [
        ["Rojo", [0, 200, 20], [9, 255, 255], [0, 0, 255]],
        ["Amarillo", [19, 120, 50], [38, 255, 255], [0, 255, 255]],
        ["Azul", [100, 150, 20], [120, 255, 255], [255, 190, 0]],
        ["Verde", [40, 130, 20], [100, 255, 255], [0, 255, 0]],
        ["Negro", [0, 0, 0], [179, 255, 35], [120, 120, 120]],
        ["Naranja", [9, 150, 120], [19, 255, 255], [0, 125, 255]],
        ["Rosa", [125, 40, 125], [15, 150, 255], [190, 130, 250]]
    ],

    "WINDOW_SCALE": 1.0,
    "WARP_OUTPUT_SIZE": 800,
    "GRID_WIDTH": 16,
    "GRID_HEIGHT": 16,
    "PERCENT_FILLED_CELL": 0.6,
    "MIN_BASE_AREA_RATIO": 0.05,
    "SHOW_PIECES_GRID": true,
    "UDP_IP": "127.0.0.1",
    "UDP_PORT": 5005
}
```

Figura 2: Parámetros configurables en el archivo «config.json» con los valores por defecto

4.2. Funcionamiento del código

Al iniciar el programa, «config.py» carga la configuración guardada en el archivo «config.json». Primero busca el archivo «config.json» en la misma carpeta del ejecutable, pueden darse tres situaciones:

- **El archivo no existe**: el sistema crea automáticamente el archivo con los valores por defecto definidos en el diccionario de la Figura 2. Estos valores se aplican inmediatamente para que el sistema los use y el usuario pueda editarlos.
- **El archivo existe y es válido**: se lee su contenido y se comprueba que todos los parámetros esperados están presentes y que los tipos de datos sean correctos. En este caso se aplican los valores.
- **El archivo existe pero no es válido**: puede deberse a un error de sintaxis JSON, a la falta de algún parámetro o a que los valores no sean del tipo esperado. En este caso, el sistema ignora por completo el archivo, muestra una advertencia en las ventanas de visualización y carga los valores por defecto. Además, regenera el archivo con los valores correctos para que el error no persista en la siguiente ejecución.

Los parámetros se almacenan en variables globales que se actualizan validando que el tipo del nuevo dato coincida con el correspondiente. El resto de archivos importan «config.py» y acceden a las variables directamente. Esta arquitectura permite que todos los componentes compartan la misma configuración y que cualquier cambio se refleje instantáneamente en todo el sistema.

Además de leer y validar, el sistema también se encarga de escribir el archivo «config.json» cuando este no existe y debe crearse, y cuando se ha detectado un error de validación y es necesario regenerarlo. En ambos casos, los parámetros se guardan en formato JSON estándar y, para facilitar su lectura y edición manual, cada color de la lista se formatea en una sola línea.

4.2.1. Recarga durante la ejecución

Durante la ejecución, el usuario puede editar «config.json» con cualquier editor de texto. Al pulsar la tecla `R`, el bucle principal invoca de nuevo la función de carga del archivo, que repite el proceso de lectura, validación y actualización de las variables. Los nuevos valores surten efecto en la siguiente iteración del bucle, sin necesidad de reiniciar el programa. Al completarse la recarga, se muestra un breve mensaje en las ventanas de visualización para confirmar que la operación se ha realizado correctamente.

Cabe destacar que los valores «UDP_IP» y «UDP_PORT» también se actualizan durante la ejecución al recargar la configuración. Sin embargo, estos cambios no se reflejan en la conexión actual porque el socket UDP se inicializa una sola vez al inicio del programa. Por tanto, es necesario reiniciar el ejecutable del módulo de detección para que surtan efecto los cambios.

5. Main y bucle principal

El script «main.py» constituye el punto de entrada del módulo de detección. Se encarga de inicializar los recursos y de gestionar el bucle principal de captura y procesado de los fotogramas, coordinando las llamadas al resto de scripts. Su flujo general ya ha sido descrito en el Capítulo 3, por lo que este apartado se centra en los detalles de implementación más relevantes.

5.1. Inicialización

Para evitar la larga latencia que OpenCV puede experimentar en Windows al acceder a la webcam, se han desactivado las transformaciones de aceleración por hardware de Microsoft Media Foundation antes de cualquier llamada a la librería. Una vez inicializada la cámara se configura su resolución, la tasa de FPS y el tamaño del buffer de fotogramas; este último se reduce a uno para minimizar la latencia al evitar la acumulación de imágenes en cola.

5.2. Bucle principal

El bucle principal implementa un procesamiento en serie con bifurcación condicional:

- Si la base es detectada, se corrige la perspectiva, se generan las máscaras de color y se clasifican y localizan las piezas a partir de ellas. A continuación, se envían estos datos por UDP.
- Si la base no es detectada, se omite el procesamiento posterior y la ventana «Piezas» muestra una imagen por defecto.

Esta arquitectura separa las responsabilidades de cada script. Cabe destacar que la configuración se carga automáticamente al importar el archivo «config.py», por lo que todos los parámetros ajustables están disponibles antes de entrar en el bucle principal.

5.3. Visualización

El programa muestra dos ventanas, cuyo contenido y apariencia se describen en detalle en el Capítulo 11:

- **Base**: el archivo «base_detection.py» gestiona internamente su representación gráfica, que incluye el contorno de la base, sus esquinas y el aviso de detección de la base.
- **Piezas**: «color_detection.py» crea la imagen de las piezas detectadas junto con los indicadores de color, mientras que «main.py» se encarga de mostrar la imagen por defecto cuando la base no ha sido detectada.

Además, las imágenes se redimensionan antes de mostrarse, aplicando el factor «WINDOW_SCALE» definido en la configuración. Cuando se produce una recarga de la configuración, se superpone un mensaje de confirmación o error en ambas ventanas durante unos segundos.

5.4. Finalización

Al salir del bucle, por cierre voluntario de las ventanas o error de captura, el programa cierra el socket UDP, libera la cámara y destruye todas las ventanas creadas. De este modo se garantiza que no queden recursos bloqueados tras la ejecución.

6. Detección de la base

El primer paso del bucle principal consiste en localizar con precisión las cuatro esquinas de la base de LEGO sobre la que se colocan las piezas. Es importante una detección fiable para la identificación de piezas posterior. En este capítulo se explicará el algoritmo desarrollado y los motivos por los que se emplea la transformada de Hough.

6.1. Proceso de detección

A continuación se detallan los pasos de procesado y las operaciones aplicadas al fotograma para identificar la base. Se ilustrará el proceso con imágenes de un mismo posicionamiento para facilitar la comprensión.

6.1.1. Fotograma inicial

La función que gestiona la detección de la base recibe una imagen del fotograma capturado por la webcam. La Figura 3 muestra el ejemplo que se va a emplear para explicar el proceso.

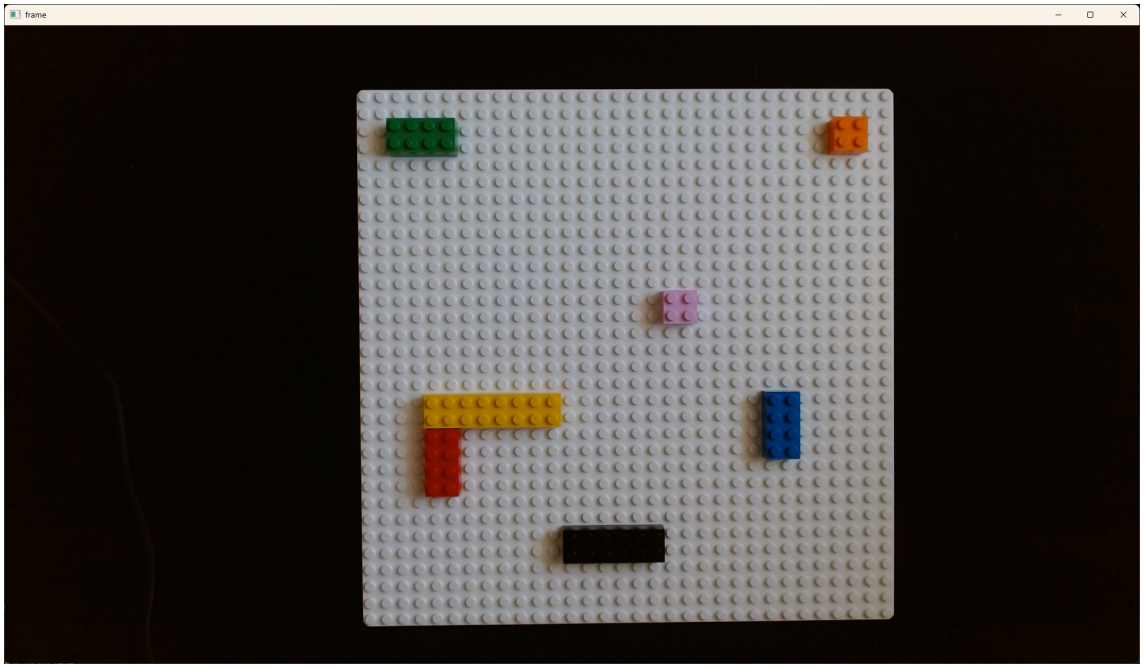

Figura 3: Fotograma inicial capturado por la webcam sin postprocesado

6.1.2. Reducción de resolución

Antes de realizar cualquier análisis, el fotograma original (1920×1080) se redimensiona al 50 % (960 × 540) del tamaño. Esta reducción disminuye drásticamente el número de píxeles que se deben procesar, por lo que reduce el coste computacional de las posteriores etapas. Sin embargo, no se pierde la capacidad de detección, ya que la base sigue ocupando una porción significativa de la imagen.

Para facilitar la visualización en esta memoria, las capturas de pantalla de las siguientes fases se han realizado sobre la imagen a resolución completa, aunque el sistema trabaja con la versión reducida.

6.1.3. Escala de grises y filtro bilateral

La imagen reducida se convierte a escala de grises porque el color no aporta información relevante para la detección de bordes y trabajar con un solo canal simplifica el procesamiento. A continuación se aplica un filtro bilateral para eliminar ruido que pueda generar bordes falsos. A diferencia de un desenfoque gaussiano, el filtro bilateral suaviza las zonas homogéneas pero preserva los bordes nítidos, lo que resulta esencial para la posterior identificación de las aristas mediante Canny. Se puede apreciar el resultado en la Figura 4.

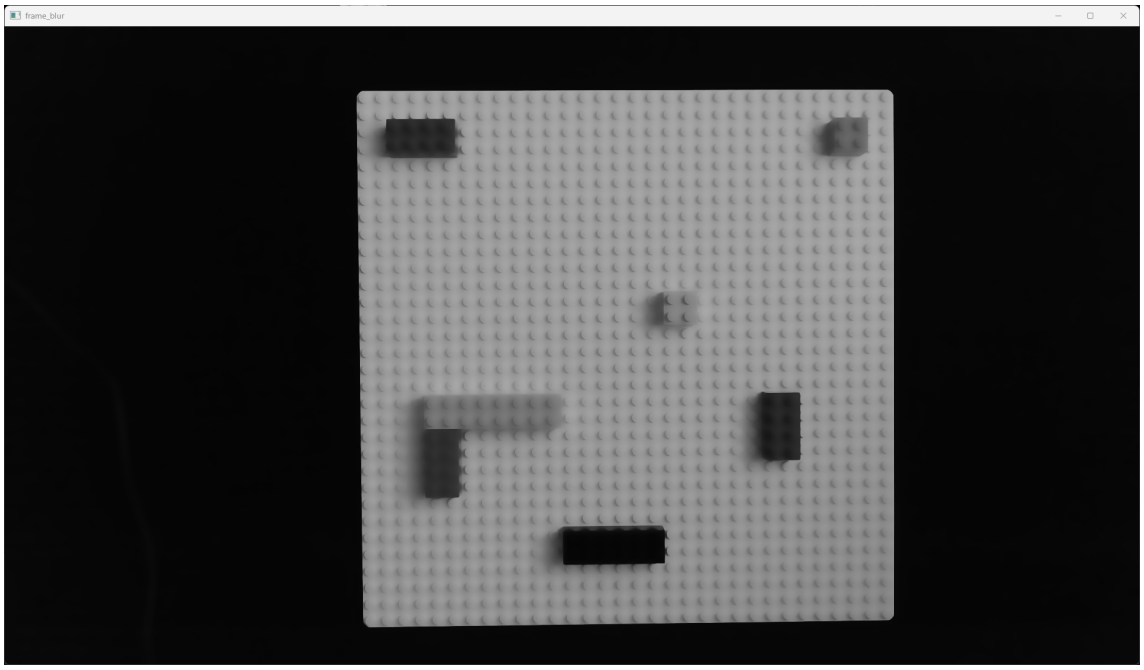

Figura 4: Fotograma en escala de grises y con un filtro bilateral

6.1.4. Binarización de Otsu y cierre morfológico

La imagen filtrada se binariza con el método de Otsu, que calcula automáticamente el umbral óptimo para separar la base del fondo. A continuación se aplica una operación morfológica de cierre, que consiste en una dilatación seguida de una erosión. Esto rellena huecos como los provocados por las sombras que proyectan las piezas, los reflejos o los studs. Además, suaviza los contornos y elimina ruido que, de no ser filtrado, generaría bordes falsos en la etapa de Canny. El resultado se muestra en la Figura 5.

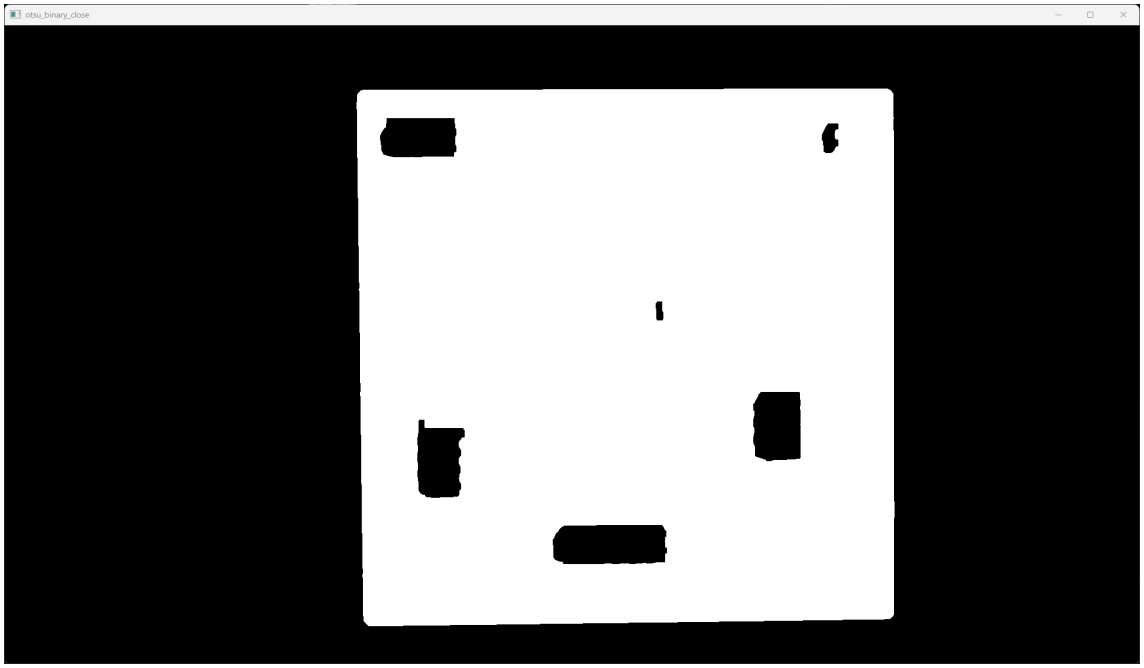

Figura 5: Fotograma binarizado con el método de Otsu y con una operación morfológica de cierre

6.1.5. Detección de bordes con Canny y dilatación

Sobre la imagen binaria se aplica el detector de bordes de Canny (cv2.Canny), que produce una imagen binaria donde los píxeles blancos representan los bordes encontrados. A continuación, los bordes se dilatan para aumentar el grosor de las líneas. Este paso ayuda a que la transformada de Hough, en la etapa siguiente, pueda identificar cada arista como una alineación continua de píxeles y trazar una única línea recta a partir de ella. Sin este engrosamiento, algunos bordes débiles podrían aparecer fragmentados y Hough generaría múltiples líneas casi coincidentes para una misma arista. El resultado se muestra en la Figura 6.

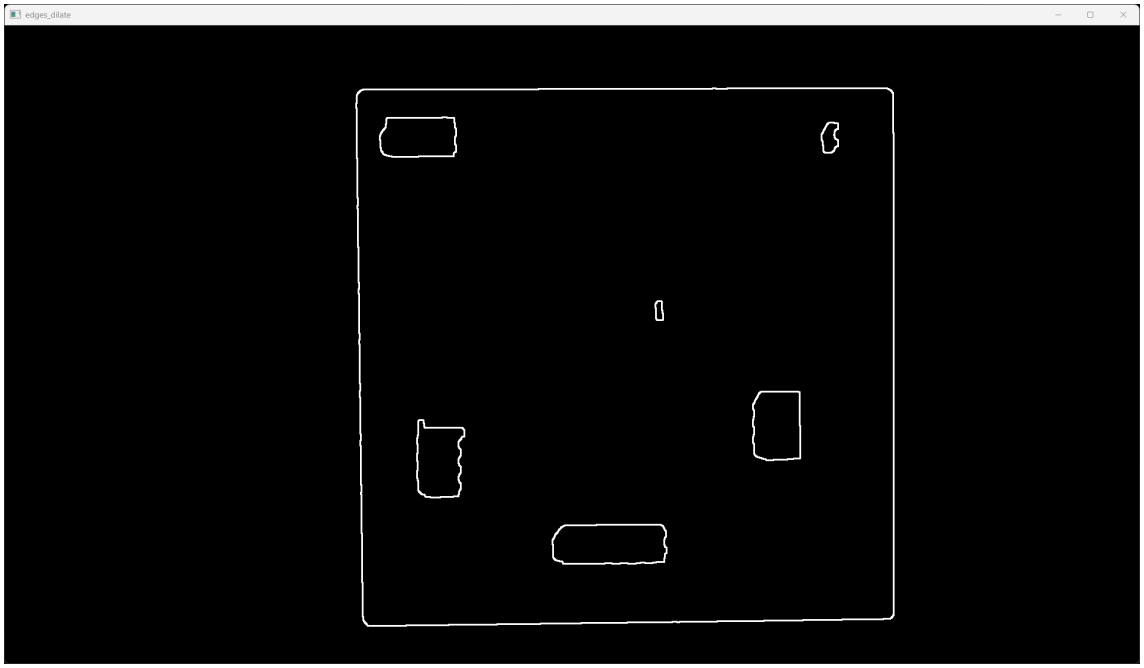

Figura 6: Fotograma con los bordes detectados mediante Canny y posteriormente dilatados

6.1.6. Transformada de Hough

La imagen de bordes dilatados se procesa con la función «cv2.HoughLines», que implementa la transformada de Hough para la detección de líneas rectas. Devuelve un conjunto de líneas en coordenadas polares (ρ, θ), donde ρ es la distancia perpendicular de la recta al origen y θ el ángulo de su normal. Se considera que una línea existe cuando al menos 150 píxeles de borde comparten los mismos parámetros; este es el umbral establecido en el código.

Cada línea es infinita, es decir, se extiende a lo largo de toda la imagen y no está limitada por los extremos del borde que la originó. Esta propiedad es clave para la etapa posterior, ya que permite intersectar líneas aunque los bordes originales no se toquen (por ejemplo, si una esquina de la base está oscurecida por una sombra). La Figura 7 muestra todas las líneas detectadas superpuestas sobre el fotograma.

Los motivos por los que se eligió este método frente a otras alternativas se analizan en la Sección 6.2.

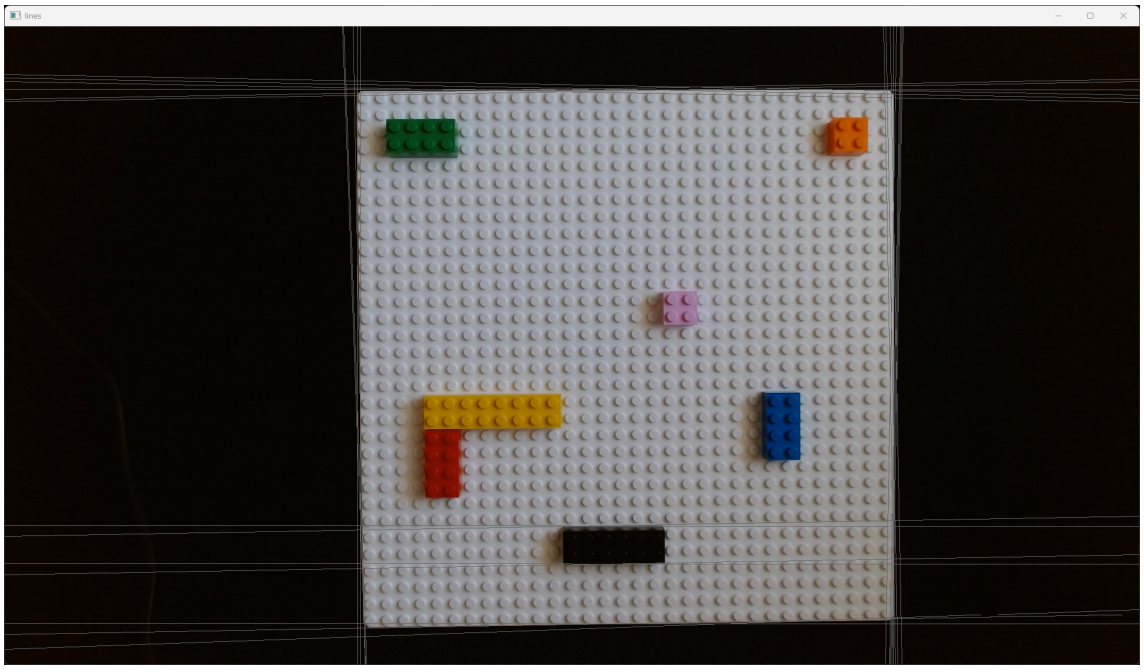

Figura 7: Líneas detectadas por la transformada de Hough

6.1.7. Filtrado de líneas duplicadas

La transformada de Hough tiende a devolver múltiples líneas muy próximas entre sí para un mismo borde, especialmente si es grueso o ligeramente irregular. Para evitar que estas duplicidades interfieran en las etapas posteriores se eliminan mediante un filtrado propio.

Se recorre la lista de líneas detectadas y, para cada una, se comprueba si ya existe una línea similar entre las que se han conservado. Se consideran similares aquellas cuyos parámetros (ρ, θ) difieren en menos de 50 píxeles y 10 grados, respectivamente. Si la línea es similar a alguna de las ya guardadas, se descarta; en caso contrario, se añade a la lista de líneas únicas. De este modo se conserva la primera línea de cada grupo de duplicadas que, por haber superado el umbral de votos de Hough, ya es una representante fiable del borde real.

Se ajustaron los umbrales de filtrado para asegurar la eliminación de líneas duplicadas sin perder aquellas válidas. La Figura 8 muestra las líneas resultantes tras este filtrado.

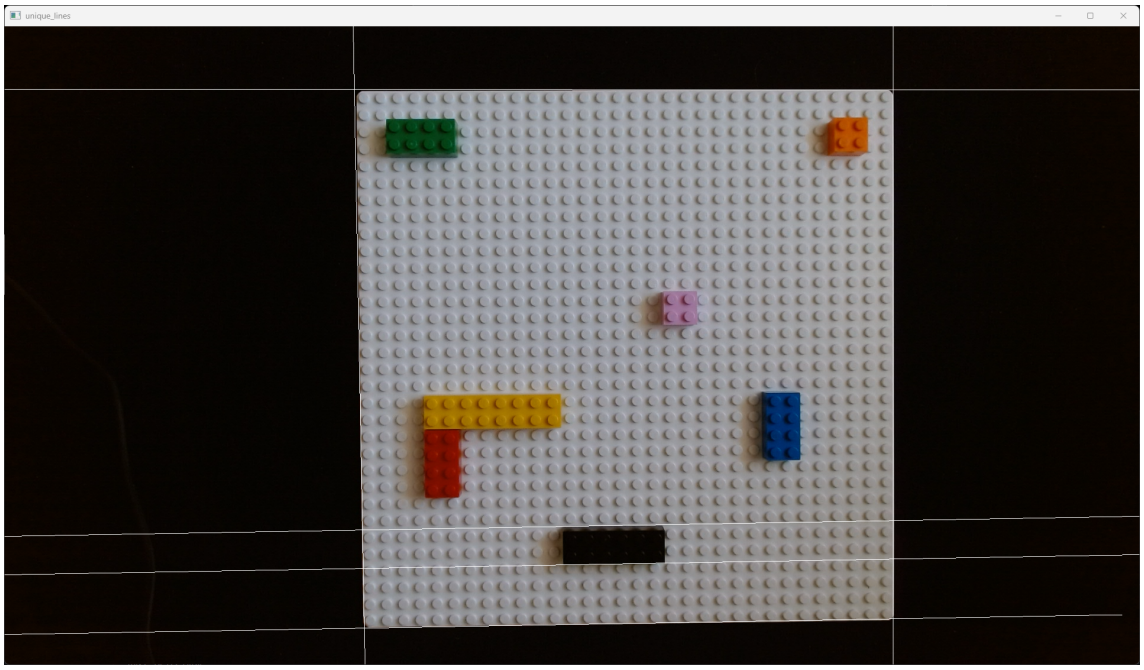

Figura 8: Líneas únicas obtenidas tras eliminar las duplicadas de la Figura 7

6.1.8. Agrupación por ángulo con K-Means

Una vez filtradas las líneas duplicadas, es necesario separarlas en dos grupos según su orientación aproximada: horizontales y verticales. Puesto que se desea detectar la base independientemente de su rotación, para no obligar al usuario a colocarla perfectamente paralela a la webcam, se emplea el algoritmo de agrupa-

miento K-Means (cv2.kmeans) con k=2 en lugar de umbrales fijos de grados. De este modo, el sistema se adapta automáticamente a cualquier orientación de la base sin necesidad de calibrar intervalos.

OpenCV devuelve θ en radianes dentro del rango $[0, \pi)$. Este rango circular supone un desafío para trabajar con ángulos: 0^{o} y 179^{o} son ángulos muy distintos numéricamente, pero ambos corresponden a líneas casi horizontales. Estos ángulos tan separados, pero que corresponden a líneas similares, pueden confundir al algoritmo. Para evitarlo se multiplica el ángulo por 2 antes de calcular el seno y el coseno. Esto hace que las líneas con la misma orientación, separadas originalmente 180^{o}, produzcan el mismo par de valores (punto). De este modo, el algoritmo percibe como cercanas las líneas que realmente tienen la misma dirección, y K-Means puede agruparlas correctamente en dos clústeres de líneas aproximadamente perpendiculares.

El resultado se muestra en la Figura 9, donde las líneas de un clúster aparecen en verde y las del otro en rojo.

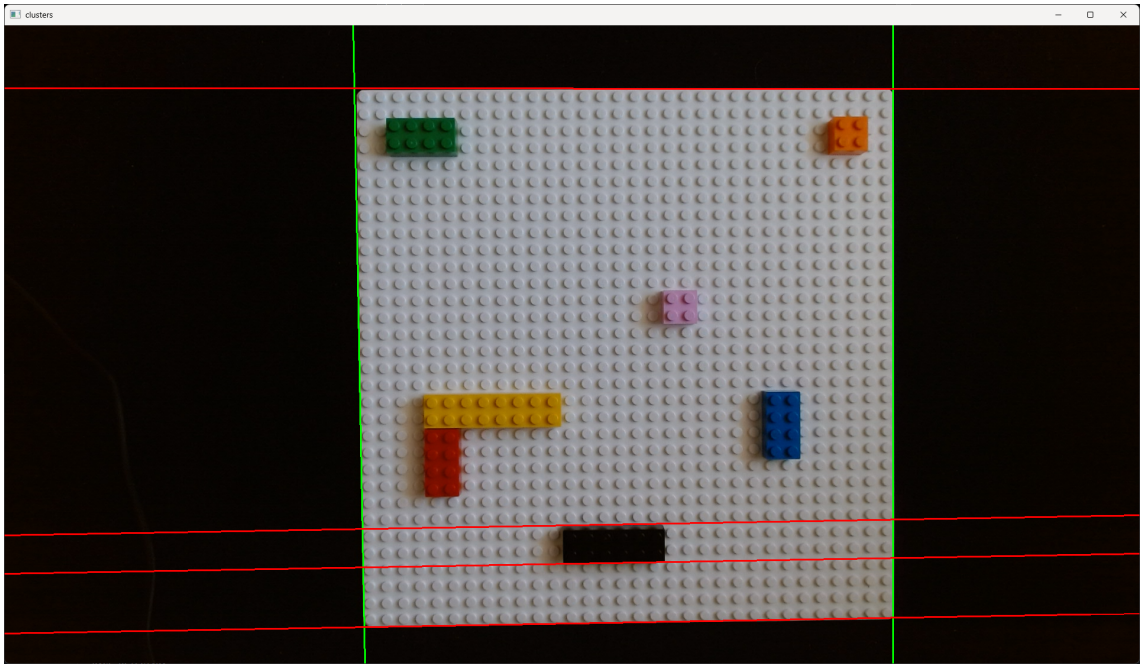

Figura 9: Líneas agrupadas en dos clústeres (verde y rojo) según su orientación mediante K-Means

La Figura 10 muestra la evolución del número de líneas detectadas a lo largo de las etapas de filtrado. La transformada de Hough detectó 26 líneas, que tras eliminar las duplicadas se redujeron a 6, agrupadas en dos clústeres.

```
BASE DETECTADA
Líneas detectadas: 26
Líneas unicas: 6
Cluster 1: 2 líneas, Cluster 2: 4 líneas
```

Figura 10: Evolución del número de líneas a lo largo de las etapas de filtrado

6.1.9. Combinaciones perpendiculares y esquinas

Una vez separadas las líneas en dos clústeres, se generan todas las combinaciones posibles tomando dos líneas de cada clúster mediante «combinations» de la biblioteca «itertools». Para cada combinación de cuatro líneas se comprueba que las dos líneas de un clúster sean perpendiculares a las dos del otro, con una tolerancia de $\pm 5°$. Esta condición garantiza que las cuatro líneas formen un cuadrilátero con ángulos rectos.

A continuación, se calculan las cuatro esquinas del cuadrilátero intersectando las líneas dos a dos y resolviendo el sistema de ecuaciones lineales de cada par de rectas. Se descartan aquellas combinaciones cuyas esquinas estén fuera del fotograma, ya que eso indicaría que la base no está completamente visible y no se podría garantizar una detección fiable de todas las piezas.

Finalmente, las esquinas resultantes se ordenan en sentido horario y se calcula el área del cuadrilátero. Entre todos los rectángulos válidos, se selecciona el de mayor área que supere el umbral mínimo definido por «MIN_BASE_AREA_RATIO» (por defecto, un 5 % del área total de la imagen). Este filtro evita que objetos pequeños con forma rectangular, como una pieza suelta, sean confundidos con la base. Todo este proceso ha sido implementado específicamente para este proyecto.

6.1.10. Dibujo del resultado

El algoritmo descrito hasta aquí se ha ejecutado sobre la imagen reducida al 50 % (960×540 píxeles). Una vez encontrado el mejor rectángulo, sus coordenadas deben trasladarse a la resolución original (1920×1080 píxeles). Para ello, las esquinas y las líneas se multiplican por el factor inverso (2), recuperando así su posición en el fotograma de alta resolución.

Finalmente, se dibuja sobre el fotograma original las líneas infinitas moradas del rectángulo, su contorno naranja y las cuatro esquinas en azul. Además, se añade el texto verde «Base detectada». La Figura 11 muestra el resultado final, donde

también se han numerado las esquinas tal como se describió en el apartado anterior. Si no se ha encontrado ningún rectángulo válido, se muestra únicamente el texto rojo «Base NO detectada».

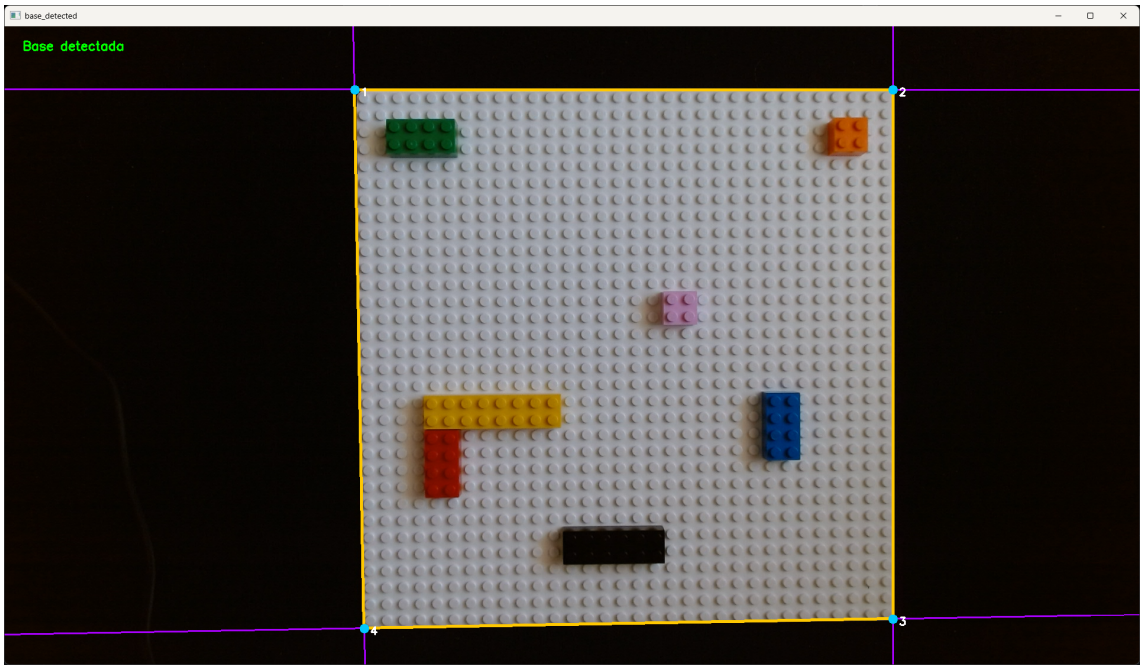

Figura 11: Resultado final de la detección de la base sobre el fotograma original, con las esquinas numeradas según el orden del algoritmo.

6.2. Método de identificación de la base

Durante el desarrollo del algoritmo se valoraron cuatro métodos para localizar la base en la imagen:

- **Contours**: tras binarizar la imagen, se busca el contorno de mayor área que forme un rectángulo. Es simple y rápido, pero poco fiable porque si el contorno está parcialmente tapado la base no se detectará. Por lo tanto es sensible a sombras, reflejos, piezas en el borde o las manos del usuario.

- **HoughLines**: aplica la transformada de Hough clásica que devuelve líneas infinitas. Es capaz de detectar las líneas incluso con oclusiones parciales, no necesita que sean completamente visibles. Además, las líneas infinitas permiten calcular sus intersecciones y perpendicularidad. Sin embargo, tiene mayor coste computacional.

- **HoughLinesP**: variante probabilística de la transformada de Hough que devuelve los segmentos de recta concretos. Es más eficiente que HoughLines

estándar, pero si el borde no es perfectamente continuo puede detectar numerosas líneas fragmentadas que obliga a un procesado complejo para detectar la base.

- **Combinación**: se plantearon diversas combinaciones emparejando los métodos previos con distinto orden pero esta complejidad no aportaba mejorías significativas frente a las opciones simples.

Finalmente se ha escogido la transformada de Hough estándar (HoughLines) por su resistencia a oclusiones parciales y capacidad de trabajar con líneas infinitas para detectar la base. Se compensó el coste computacional reduciendo el tamaño del fotograma. El umbral de detección de líneas se mantuvo en 150 votos para no limitar la detección al variar la altura de la cámara. La Figura 12 muestra un ejemplo de detección con la base parcialmente tapada y rotada, donde el algoritmo la identifica correctamente.

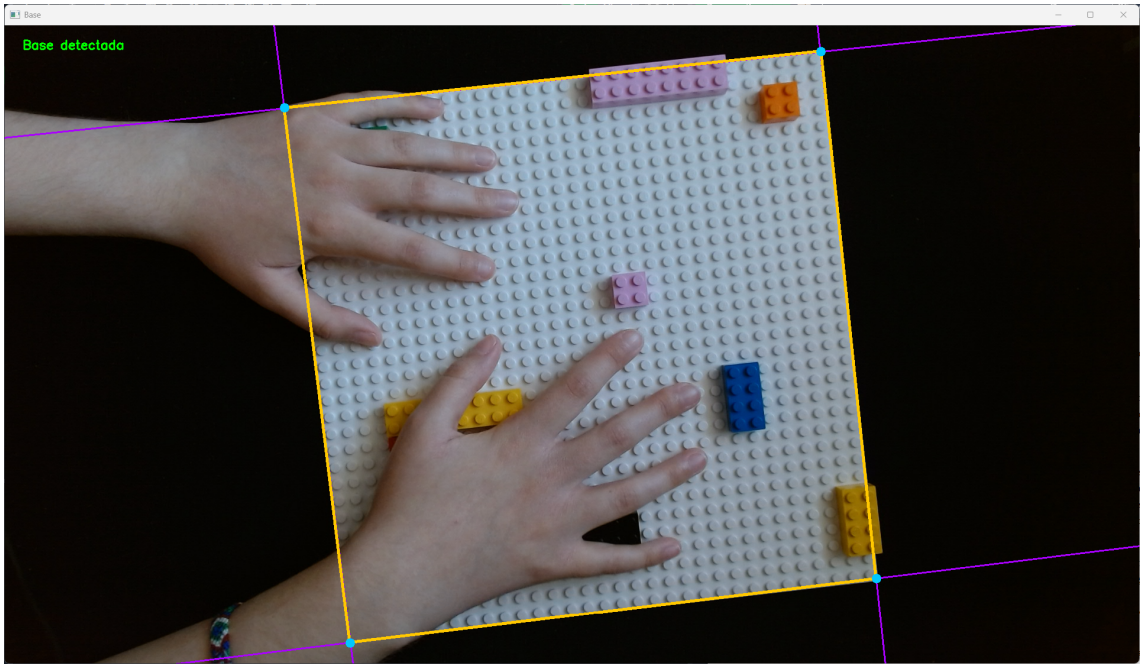

Figura 12: Detección de la base con rotación y oclusión parcial causada por piezas en el borde y las manos del usuario

7. Corrección de perspectiva

Una vez detectadas las cuatro esquinas de la base, se aplica una corrección de perspectiva para obtener una vista cenital de la superficie sobre la que se colocan las piezas. Esta transformación, implementada en el archivo «base_warp.py», permite compensar posibles inclinaciones de la base o de la webcam para poder analizar de forma fiable la posición de las piezas con una cuadrícula.

7.1. Funcionamiento del código

Este script recibe el fotograma original a máxima resolución y las cuatro esquinas de la base, ordenadas en sentido horario. A partir de ellas, calcula el ancho y alto promedio del rectángulo detectado y los multiplica por el factor de escala para que el lado más largo coincida con «WARP_OUTPUT_SIZE». La imagen se obtiene recortando la región de la base y aplicando una transformación de perspectiva que corrige las deformaciones, dejándola lista para la posterior identificación de las piezas.

La Figura 13 muestra, a la izquierda, el fotograma original con la base considerablemente inclinada y el rectángulo de detección dibujado. A la derecha se presenta la imagen con la perspectiva corregida y sobre ella el script de «color_detection» ha añadido la cuadrícula y los indicadores de las piezas detectadas.

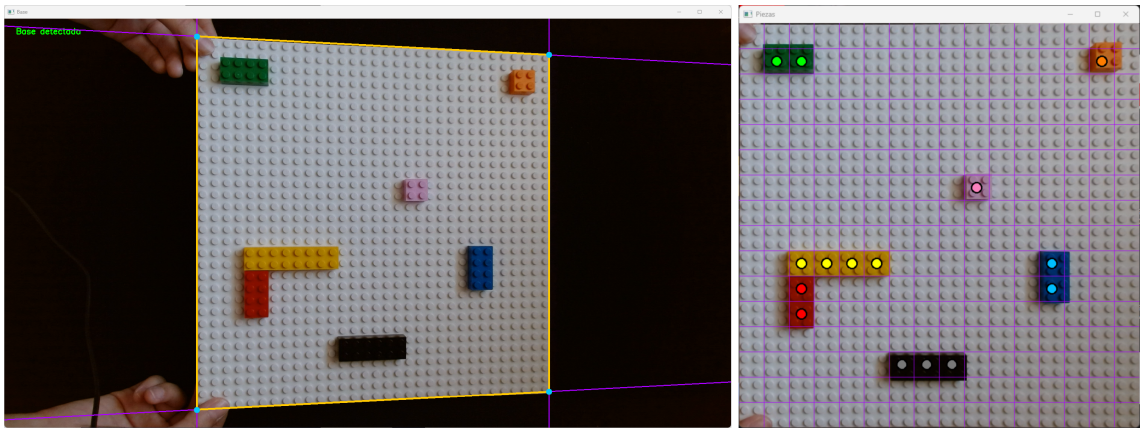

Figura 13: Corrección de perspectiva. Izquierda: fotograma con la base inclinada. Derecha: imagen con la perspectiva corregida y la detección de piezas

8. Detección de color y posición

Una vez corregida la perspectiva de la base, el siguiente paso es identificar las piezas LEGO situadas sobre ella, determinando tanto su color como su posición. Esta tarea se realiza en el archivo «color_detection.py» y se basa en la generación de máscaras de color en el espacio HSV para su posterior análisis según la división de la base en una cuadrícula. Este capítulo describe el proceso y las decisiones de diseño adoptadas.

8.1. Máscaras de color

El primer paso para identificar las piezas consiste en determinar qué píxeles de la imagen pertenecen a cada uno de los colores definidos en la configuración.

Para ello se convierte la imagen resultante de la corrección de perspectiva previa, de BGR (formato nativo de OpenCV) al modelo de color HSV. Este modelo de color separa el tono del color (H), la saturación (S) y el brillo (V). Es más robusto para trabajar porque cuando la iluminación varía, el tono de un objeto tiende a mantenerse estable aunque cambie la intensidad de la luz.

A continuación, se aplica un filtro cv2.inRange para cada color almacenado en la lista «COLORS» del archivo «config.json». Se generan máscaras binarias en las que los píxeles cuyo valor HSV se encuentra dentro del rango especificado se vuelven blancos, y el resto, negros. Es decir, las zonas blancas representan regiones de la imagen que contienen ese color.

La Figura 14 muestra las máscaras de color generadas para los siete colores detectados: rojo, amarillo, azul, verde, negro, naranja y rosa

8.1.1. Rangos circulares

En el modelo de color HSV el tono (H) es circular. OpenCV lo representa en el rango [0, 179], donde los 0 y 179 son valores adyacentes en la realidad, que corresponden a los rojos. Para colores que se sitúan en ambos extremos de este rango, como el rosa, no es posible definir un único intervalo continuo con «cv2.inRange», ya que esta función espera que el límite inferior sea menor que el superior.

Para resolver este problema sin necesidad de duplicar límites en la configuración, se implementó una lógica de rango circular. Si se detecta que el límite inferior del tono es mayor que el superior, la función divide automáticamente el rango en dos partes: una desde el límite inferior hasta 179, y otra desde 0 hasta el límite superior. Ambas máscaras se combinan posteriormente mediante una operación lógica OR, obteniendo así una única máscara que cubre todo el espectro deseado. El color rosa

en la Figura 2 del Capítulo 4 y la Figura 14 ejemplifica esta situación: se optó por un rango circular que, aunque introduce ruido en la máscara, detecta de forma más fiable las piezas rosas en distintas condiciones lumínicas.

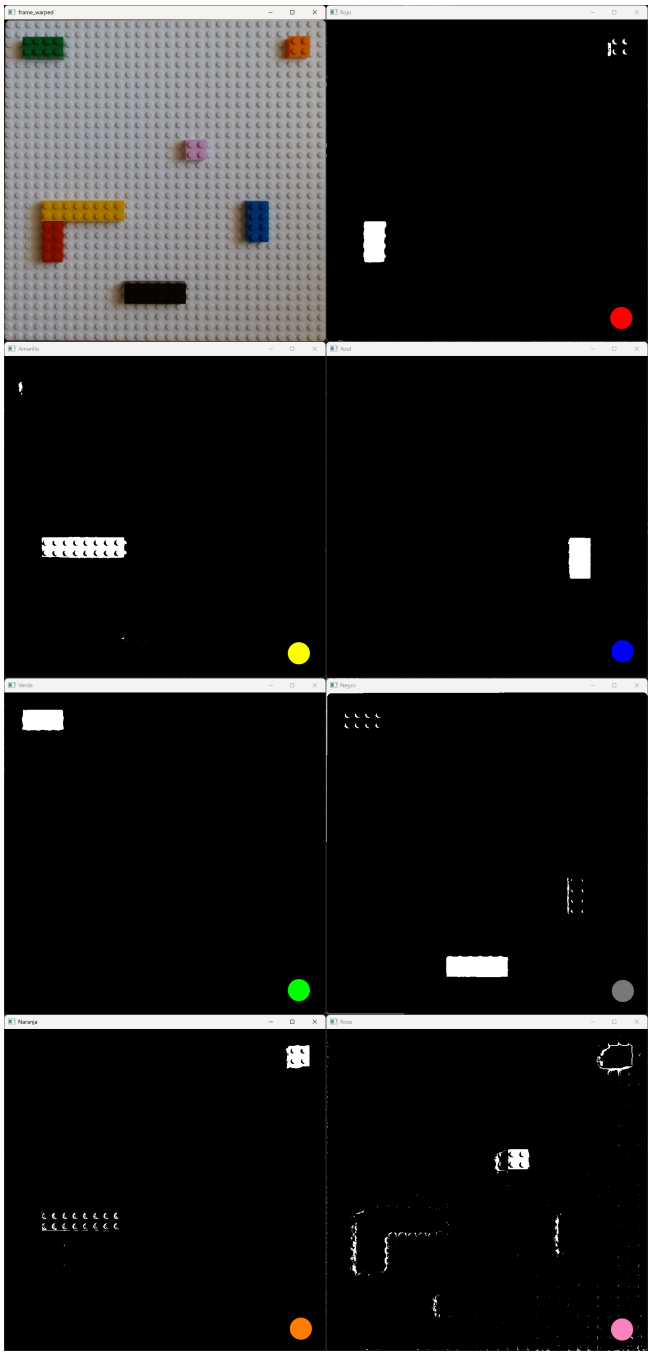

Figura 14: Máscaras de color generadas a partir de la imagen con la perspectiva corregida para los siete colores

8.2. Detección de la posición

Una vez generadas las máscaras de color, el sistema debe determinar la posición de las piezas sobre la base. Para ello, la imagen con la perspectiva corregida se divide en una cuadrícula de 16x16 celdas. Teniendo en cuenta que la base tiene 32x 32 studs, esta división se corresponde con las piezas más comunes de LEGO de 2x2. Se consideró realizar una división de 32 x32 pero las piezas de LEGO de 1x1 studs son pequeñas y difíciles de manipular (Figura 15). Sin embargo se pueden cambiar los valores por los necesarios en «config.json».

Para cada celda, el sistema recorre la región de interés (ROI) correspondiente en cada una de las máscaras de color y cuenta el número de píxeles blancos. Se selecciona el color que tenga el mayor número de píxeles, siempre que estos superen un umbral mínimo de ocupación definido por el parámetro «PERCENT_FILLED_CELL» (por defecto 60 % del área de la celda). Si ninguna máscara alcanza el umbral, la celda se considera vacía.

Este umbral filtra el ruido presente en las máscaras, como el mencionado en la Subsección 8.1.1 para el color rosa, ya que los píxeles aislados o las pequeñas agrupaciones difícilmente llegarán a superar el 60 % de una celda. Además, compensa los pequeños desajustes de la corrección de perspectiva al evitar que píxeles de piezas situadas en celdas contiguas produzcan un falso positivo.

El resultado del proceso es una lista de piezas, donde cada una queda definida por su color y sus coordenadas de celda (columna, fila). Esta lista es la que posteriormente se envía al motor de videojuegos a través del socket UDP.

8.2.1. Elección del método de detección

Inicialmente se implementó una detección basada en contornos, donde se buscaban las regiones blancas de cada máscara y se calculaba su centroide. Este método detectaba bien piezas aisladas, pero cuando varias piezas del mismo color se colocaban juntas, sus contornos se fusionaban en uno solo, impidiendo la identificación individual de cada una.

Para solventar estas limitaciones, se optó por una cuadrícula, descrita en apartado anterior. Este enfoque aprovecha la naturaleza de las piezas de LEGO que encajan alineadas gracias a los studs. Cabe destacar que, debido al tamaño de celda elegido (2×2 studs), las piezas deben colocarse en posiciones pares de la cuadrícula de studs para que cada una ocupe una única celda.

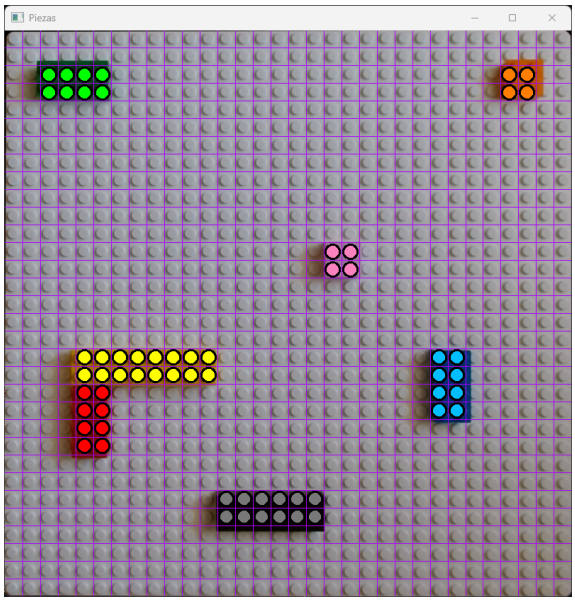

Figura 15: Detección de piezas con cuadrícula de 32x32 celda, opción descartada

8.3. Visualización de los resultados

Además de devolver la lista de piezas, la función se encarga de dibujar sobre la imagen con la perspectiva corregida la cuadrícula (líneas moradas) y un círculo en el centro de cada celda ocupada con el color detectado. Esta visualización se muestra en la ventana «Piezas» y permite al usuario verificar de un vistazo la correcta detección. La Figura 16 muestra la visualización de las piezas del ejemplo empleado a lo largo de toda la explicación.

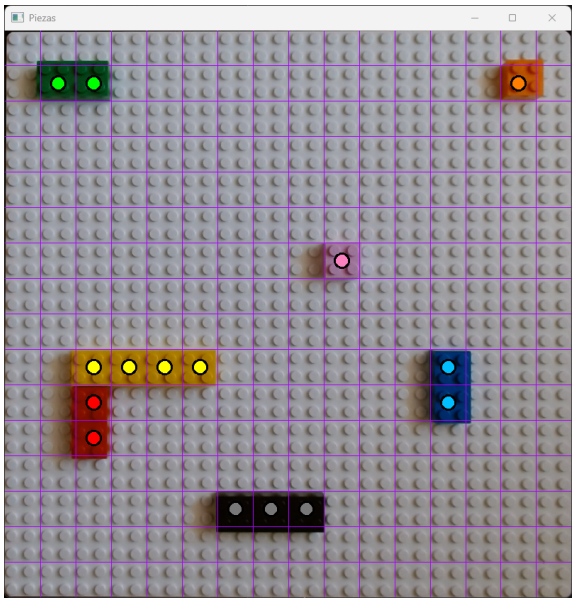

Figura 16: Ventana «Piezas» mostrando la imagen con la cuadrícula de 16×16 celdas y las piezas detectadas

8.4. Herramienta auxiliar de calibración

Ajustar los rangos de color en el espacio HSV no es intuitivo porque los valores numéricos del tono, la saturación y el brillo no se corresponden de forma evidente con la percepción visual de un color. Para facilitar esta labor durante el desarrollo, se creó una pequeña herramienta auxiliar de cuentagotas.

Se ejecuta de forma independiente al sistema principal y muestra en tiempo real la imagen capturada por la webcam. Al hacer clic con el ratón sobre cualquier píxel de la imagen, imprime en la consola sus valores en formato BGR y HSV, como se puede observar en la Figura 17. De este modo, fue posible muestrear directamente las piezas LEGO de cada color bajo diferentes condiciones de iluminación, obteniendo los valores con los que se definieron los rangos de detección presentados en el Capítulo 4.

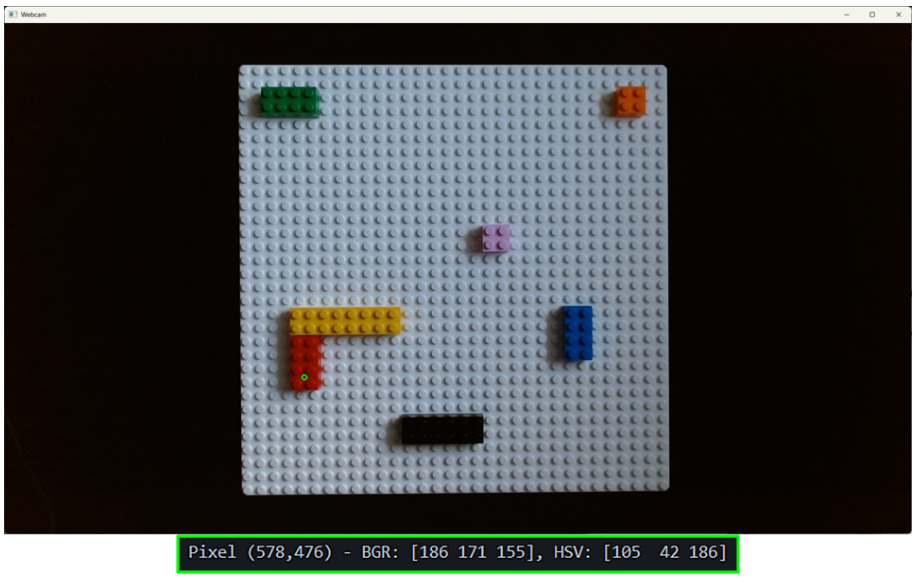

Figura 17: Herramienta auxiliar de cuentagotas. Muestra los valores BGR y HSV del píxel seleccionado sobre la pieza roja

9. Envío UDP

Una vez detectadas y clasificadas las piezas, los resultados deben transmitirse al módulo de visualización para que el motor de videojuegos pueda representarlas de forma sincronizada. Esta comunicación se realiza a través del protocolo UDP mediante la clase implementada en el archivo «udp_sender.py». Se optó por UDP en lugar del Protocolo de Control de Transmisión (TCP) porque la aplicación prioriza la velocidad y la fluidez frente a la fiabilidad de entrega, evitando así las latencias por retransmisión propias de TCP.

9.1. Formato del mensaje

En cada iteración del bucle principal en la que se detecta la base, la lista de piezas se serializa en formato JSON. Se ha elegido este formato por su ligereza, su legibilidad y su facilidad de parseo desde cualquier motor de videojuegos. El mensaje está compuesto por la lista de objetos «pieces». Cada objeto tiene tres campos: color (cadena de texto) y coordenadas x e y (números enteros), que indican la celda en la que se ha detectado la pieza. En la Figura 18 se muestra el mensaje enviado con los datos de las piezas del ejemplo empleado a lo largo de la explicación del proceso.


```
{"pieces": [{"color": "Verde", "x": 1, "y": 1}, {"color": "Verde", "x": 2, "y": 1}, {"color": "Naranja", "x": 14, "y": 1
}, {"color": "Rosa", "x": 9, "y": 6}, {"color": "Amarillo", "x": 2, "y": 9}, {"color": "Amarillo", "x": 3, "y": 9}, {"col
or": "Amarillo", "x": 4, "y": 9}, {"color": "Amarillo", "x": 5, "y": 9}, {"color": "Azul", "x": 12, "y": 9}, {"color": "R
ojo", "x": 2, "y": 10}, {"color": "Azul", "x": 12, "y": 10}, {"color": "Rojo", "x": 2, "y": 11}, {"color": "Negro", "x":
6, "y": 13}, {"color": "Negro", "x": 7, "y": 13}, {"color": "Negro", "x": 8, "y": 13}]}
```


Figura 18: Ejemplo del mensaje JSON enviado por UDP con la información de las piezas detectadas en la Figura 16

9.2. Gestión de la ausencia de piezas

El mensaje UDP se envía en todas las iteraciones, incluso cuando no se detecta ninguna pieza. En ese caso, el campo «pieces» se envía como una lista vacía: `{"pieces": []}`. De esta forma el módulo de visualización conoce el estado real de la base en todo momento y puede eliminar todos los actores de la escena cuando quede vacía, manteniendo la representación 3D siempre sincronizada con lo que ve la webcam.

Además, se capturan las excepciones de red para evitar que un error puntual en el socket interrumpa la ejecución del bucle principal. Si se produce un fallo, se imprime una única advertencia en la consola y el programa continúa normalmente, garantizando que la detección no se detenga por problemas de conectividad.

10. Empaquetado

Para facilitar la distribución y el uso de la herramienta sin necesidad de instalar Python ni sus dependencias, el módulo de detección se empaqueta en un único archivo ejecutable. La generación del ejecutable independiente se realizó con la herramienta «PyInstaller» y el siguiente comando:

```
pyinstaller main.py --onefile --noconsole --name=ChromaVision
--icon=IconChromaVision.ico
```

Este comando genera un solo archivo ejecutable (.exe) evitando que se muestre una ventana terminal junto a las ventanas de la aplicación. Además se le ha asginado el nombre «ChromaVision» y el icono personalizado de la marca. No es necesario incluir el archivo «config.json» porque el propio programa lo genera automáticamente en su primera ejecución si no existe. De este modo, el ejecutable es completamente autónomo y, al mismo tiempo, permite al usuario modificar la configuración sin necesidad de reempaquetar.

11. Interfaces

La interfaz de la herramienta se compone de dos ventanas principales: «Base» y «Piezas». Si bien es cierto que también está la parte configurable de los parámetros del archivo «config.json», esto se modifican en cualquier editor de texto.

11.1. Ventana «Base»

Muestra el fotograma capturado por la cámara en tiempo de ejecución. Sobre él se superponen los resultados de la detección de la base: las líneas infinitas de los bordes de la base (morado), su contorno (naranja), las cuatro esquinas (azul) y el texto «Base detectada» (verde). Cuando la base no se detecta, solo aparece el texto «Base NO detectada» en rojo. Esta ventana permite al usuario verificar de un vistazo si la cámara está correctamente enfocada y si la base ha sido identificada. La Figura 19 muestra el estado en el que la base ha sido detectada.

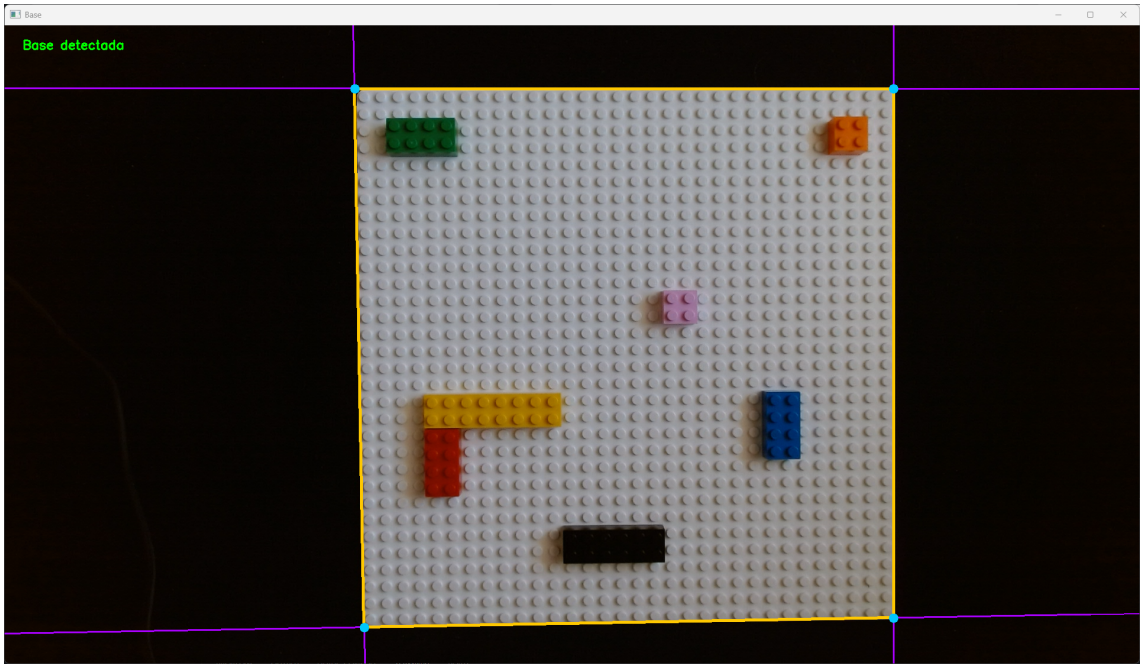

Figura 19: Ventana «Base» cuando la base ha sido detectada

11.2. Ventana «Piezas»

Muestra la imagen de la base con la perspectiva corregida. Sobre ella se superpone la cuadrícula de 16×16 celdas en color morado y círculos centrados en cada celda ocupada, rellenos del color correspondiente a la pieza detectada. Esta visualización permite al usuario comprobar en un vistazo rápido que las piezas se están localizando

y clasificando correctamente. Si la base no se detecta, se muestra una imagen negra con el texto «Base NO detectada» en rojo. La Figura 20 muestra ambos estados.

Figura 20: Ventana «Piezas» en ambos estados de detección de la base

La cuadrícula sirve de referencia visual para que el usuario coloque correctamente las piezas pero puede desactivarla en el archivo «config.json» mediante la variable «SHOW_PIECES_GRID». La Figura 21 muestra la ventana «Piezas» sin cuadrícula y con el aviso que confirma la recarga de la configuración; este mismo mensaje aparece también en la ventana «Base.»

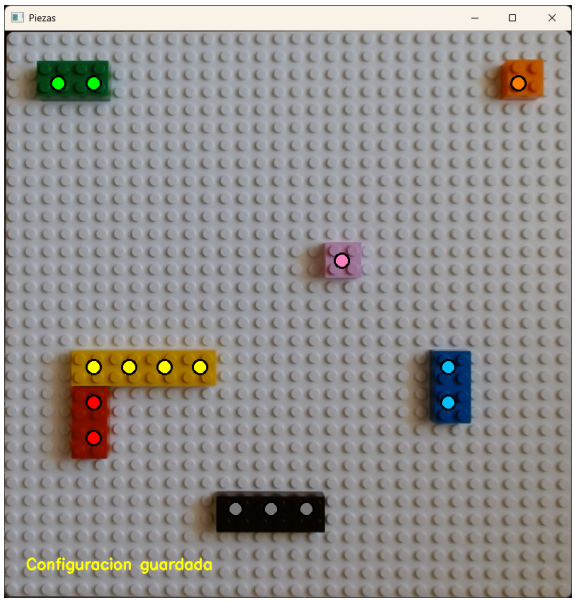

Figura 21: Ventana «Piezas» sin la cuadrícula, acompañada del mensaje de confirmación de la recarga de la configuración

11.3. Redimensionado de las ventanas

OpenCV solo permite redimensionar las ventanas sin conservar la relación de aspecto. Esto puede provocar que el usuario deforme la visualización al redimensionar las ventanas. Por este motivo se decidió que el tamaño de las imágenes únicamente fuese controlado por el parámetro «WINDOW_SCALE» del archivo de configuración. Este aplica un escalado proporcional y mantiene la relación de aspecto para garantizar que la interfaz siempre se vea correctamente.

12. Métricas y pruebas realizadas

Para evaluar el rendimiento del módulo de detección y validar su funcionamiento, se llevaron a cabo dos tipos de pruebas: medición de FPS y pruebas bajo distintas condiciones lumínicas. Este capítulo describe la metodología empleada y los resultados obtenidos.

12.1. Medición del rendimiento

Para evaluar el rendimiento del módulo de detección se midió su tasa de FPS durante una sesión de uso real. Conocer este valor es fundamental para verificar que el sistema procesa los fotogramas con la fluidez suficiente para una colaboración ágil.

12.1.1. Método de medición

Se ha medido el promedio, máximo y minimo de FPS registrados durante un minuto en el que se han manipulado la base y piezas de diversos tamaños y colores, simulando el uso normal de la aplicación.

Cabe destacar que el límite teórico superior lo establece la propia cámara, ya que alcanza un máximo de 30 FPS. Los valores cercanos a este dato reflejan una carga computacional óptima y un tiempo de respuesta adecuado, mientras que valores por debajo de 15 FPS se considerarán inadecuados.

12.1.2. Resultados

En la primera versión del sistema, el algoritmo de detección de la base procesaba el fotograma completo a resolución 1920×1080 píxeles. Con esta configuración, la medición obtuvo una media de 7,57 FPS, con un máximo de 8,34 FPS y un mínimo de 5,05 FPS. Este rendimiento, aunque suficiente para validar el funcionamiento del algoritmo, quedaba lejos de los 30 FPS de la cámara y de los 15 FPS mínimos establecidos en los requisitos.

El principal cuello de botella estaba situado en la detección de la base, cuyo coste computacional crece con el número de píxeles de la imagen de entrada. Para reducir este coste, se introdujo una reducción de resolución interna en el propio script «base_detection.py»: el fotograma original se redimensiona al 50 % (960×540 píxeles) antes de aplicar la transformada de Hough y el resto de operaciones. Las coordenadas resultantes se reescalan posteriormente a la resolución original, por lo que la precisión de la detección no se ve afectada.

Tras aplicar esta optimización la tasa aumentó a 21,72 FPS promedio, 29,51 FPS máximos y 16,72 FPS mínimos. Como se puede observar en la Tabla 1, la

optimización multiplicó aproximadamente por tres el rendimiento inicial. El pico máximo casi alcanza el límite teórico de la webcam y el mínimo supera los 15 FPS establecidos. La media de 21,72 FPS supone un rendimiento adecuado que permite observar el resultado de la detección de las piezas con una sincronización fluida. La diferencia entre ambas configuraciones era visible a simple vista y el tiempo de reacción disminuyó notablemente.

Tabla 1: Comparativa de rendimiento antes y después de la optimización

Resolución	**FPS promedio**	**FPS máximo**	**FPS mínimo**
Original (1920 x 1080)	7,57	8,34	5,05
Mitad (960 x 540)	21,72	29,51	16,72

12.2. Pruebas de iluminación

La detección de color es sensible a las condiciones de iluminación, ya que los valores de los colores detectados varían significativamente. Para garantizar la robustez del sistema, se calibraron los rangos de color y se validó su funcionamiento en tres escenarios controlados:

- Luz natural procedente de una ventana.
- Luz artificial de techo con temperatura de color neutra.
- Luz artificial procedente de un flexo orientado indirectamente. La iluminación es pobre y con tonos muy cálidos.

Durante las pruebas se observó que los colores más conflictivos eran los adyacentes en el espectro (rojo, naranja y rosa), especialmente bajo luz de temperatura cálida. Mientras que los más fiables son las piezas verdes, azules y negras. La calibración final, recogida en los valores por defecto de «config.json» del Capítulo 4, ha demostrado la correcta detección de todos los colores bajo las tres condiciones lumínicas evaluadas.

La Figura 22 muestra la detección en cada condición, de izquierda a derecha: luz natural, luz artificial de techo y luza rtificial indirecta.

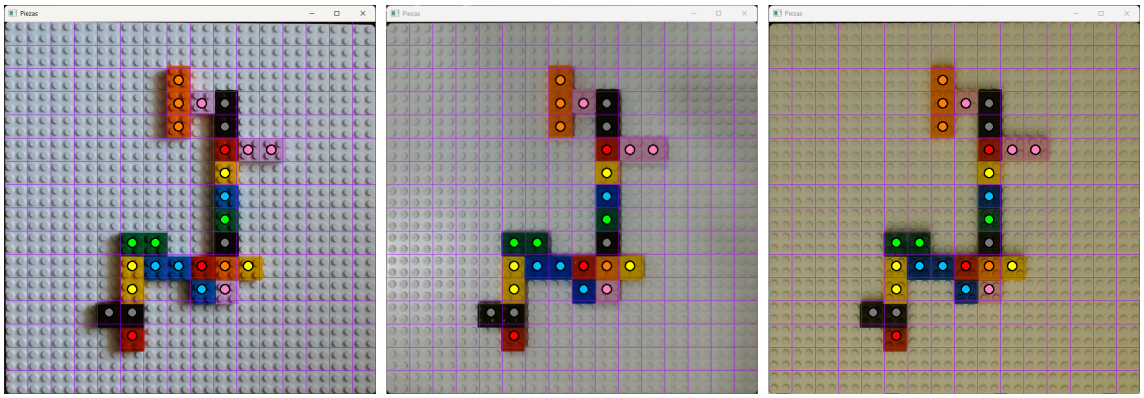

Figura 22: Detección de piezas bajo las tres condiciones lumínicas evaluadas. De izquierda a derecha: luz natural, luz artificial de techo y luz artificial indirecta

Otro aspecto importante son las sombras y los reflejos. Como se puede observar en la Figura 23, las sombras oscurecen los colores pero pueden compensarse en el rango de detección. En cambio, los reflejos convierten el color en blanco, lo que impide la detección de esas zonas. Esta limitación es inherente al uso de una cámara y una superficie brillante, y puede mitigarse ajustando la posición de la luz para minimizar los reflejos directos.

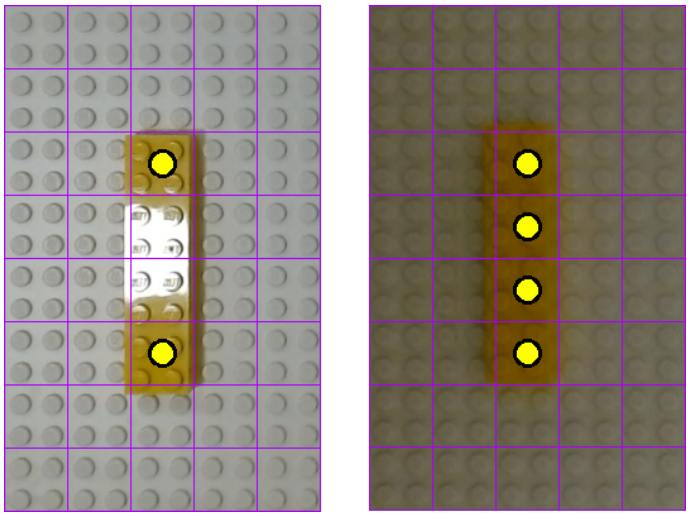

Figura 23: Efecto de las sombras y los reflejos sobre la detección de piezas

Anexo C

Módulo de visualización
(Unreal Engine)

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos
Dpto. de Informática y Automática

Trabajo de Fin de Grado
Octubre 2025 - Junio 2026

Autora
Lucía García Bobillo

Tutor
Juan Alberto García Esteban
Dpto. de Informática y Automática

Cotutor
Roberto José García Martín
Dpto. de Ingeniería Mecánica

Universidad de Salamanca
Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela Politécnica Superior de Zamora

Siglas

3D tres dimensiones. 1, 8, 9

BP Blueprint. 2, 6–10, 13, III

HSV Hue, Value, Saturation. 14, 15

IP Internet Protocol. 6

JSON JavaScript Object Notation. 1–3, 7, 8, 12, 14, II

RGB Red, Green, Blue. 14, 15

UDP Protocolo de Datagramas de Usuario. 1–3, 6–8, 12–15, II, III

UE Unreal Engine. 1–4, 6, 10, 12–15, II

Índice general

Siglas . . . . . **I**

1 Introducción **1**

2 Organización de los archivos **2**

3 Flujo general del módulo **3**

4 Receptor UDP **6**

4.1 Parámetros configurables en el editor 6

4.2 Funcionamiento del código 7

4.2.1 Inicialización de la recepción 7

4.2.2 Recepción y filtrado iniciales 7

4.2.3 Transferencia al hilo principal 7

4.2.4 Parseo del JSON y extracción de datos 8

4.2.5 Delegación en el componente de spawneo 8

4.2.6 Detención de la recepción 8

5 Instanciación de actores **9**

5.1 Parámetros configurables en el editor 9

5.2 Funcionamiento del código 10

5.2.1 Conversión de coordenadas de celda a mundo 10

5.2.2 Actualización de la escena 10

5.2.3 Limpieza total de la escena 11

6 Integración en motores de videojuegos **12**

6.1 Compatibilidad con diferentes motores 12

6.2 Reutilización en proyectos de UE 12

6.3 Configuración en el editor 13

7 Calibración dinámica de colores (descartada) **14**

7.1 Planteamiento 14

7.2 Motivos de descarte 15

Índice de figuras

1 Diagrama del flujo del módulo de visualización 5

2 Parámetros del receptor UDP configurables en el editor 7
3 Funciones del receptor UDP disponibles en el BP 7

4 Parámetros del instanciador de actores configurables en el editor 10
5 Funciones del instanciador de actores disponibles en el BP 10

6 Prototipo del sistema de calibración dinámica de colores 15

1. Introducción

Este anexo describe la implementación del módulo de visualización desarrollado para Unreal Engine (UE). Se encarga de recibir los datos enviados por el módulo de detección (Python) a través del protocolo UDP e interpretar el JSON con la lista de piezas detectadas. Para entonces generar los correspondientes actores 3D en el nivel, manteniendo la escena sincronizada con el estado real de la base física.

Se explican la organización de los archivos, el flujo general de procesamiento y el funcionamiento del código. También se analiza la integración del módulo en otros proyectos de UE y su portabilidad a otros motores de videojuegos. Por último, se presenta el prototipo del sistema de calibración dinámica de colores que finalmente se descartó.

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Organización de los archivos

El módulo de visualización se implementa en C++ dentro del motor UE y se compone de dos clases principales, cada una con una responsabilidad claramente definida e independiente:

- **UDPReceiver (actor)**: se encarga de toda la comunicación por red. Abre un socket UDP por el que recibe datagramas con información de las piezas de forma asíncrona en un hilo secundario. Luego convierte los datos a texto, parsea el JSON y notifica a otros sistemas cuando se recibe un nuevo conjunto de piezas.
- **PieceSpawnerComponent (componente de actor)**: se encarga de instanciar los actores correspondientes a las piezas recibidas por el receptor UDP. Almacena un mapeo entre el nombre de un color (cadena de texto) y una clase de actor (Blueprint (BP)). Cuando recibe el array de piezas actualiza la escena destruyendo los actores que ya no están presentes, añadiendo los nuevos y reemplazando aquellos cuyo color ha cambiado. También es responsable de convertir las coordenadas de celda recibidas a posiciones absolutas en el mundo, usando parámetros configurables como «CellSize» y «SpawnHeight».

Como se puede apreciar, la lógica de ambos scripts es dependiente ya que el receptor UDP únicamente se encarga de gestionar la conexión de red y obtener el array de piezas detectadas por el módulo de detección. Entonces se llama al instanciador de actores que gestiona la construcción del nivel. Esta modularidad favorece el mantenimiento del código.

3. Flujo general del módulo

El diagrama de flujo de la Figura 1 ilustra el pipeline completo del módulo de visualización. Se ha utilizado notación estándar: óvalos para el inicio y fin del programa (amarillo), rectángulos para los procesos (morado) y rombos para las decisiones (verde). Las referencias en cursiva indican el archivo fuente donde se implementa cada etapa.

El módulo de visualización recibe los datos enviados por el módulo de detección (Anexo B) a través del protocolo UDP. La recepción y el procesamiento de estos datos se reparten entre dos clases principales: «UDPReceiver», que gestiona la comunicación de red y el parseo del JSON, y «PieceSpawnerComponent», que gestiona los actores en la escena. El diagrama de flujo representa este proceso desde la llegada de un datagrama hasta la actualización de la escena.

La llegada de un datagrama UDP al socket ejecuta el proceso en un hilo secundario para no interrumpir el flujo principal. Primero «UDPReceiver» convierte los bytes recibidos del mensaje en una cadena de texto y elimina los caracteres residuales que pueden aparecer tras la última llave JSON durante el envío. A continuación se transfiere el mensaje al hilo principal de UE donde se parsea como un objeto JSON y se extrae un array de piezas. Cada elemento contiene una estructura que almacena el color y las coordenadas de celda de cada pieza detectada.

Una vez «UDPReceiver» ha gestionado la comunicación y obtenido la lista de piezas, se solicita al componente «PieceSpawnerComponent» que actualice la escena con el array de piezas recibido.

Esta actualización de los actores instanciados según las piezas recibidas sigue un proceso secuencial y diferencial, es decir, en lugar de borrar todos los actores previos de la escena e instanciar los nuevos, se realiza de forma optimizada con condicionales.

Primero «PieceSpawnerComponent» comprueba si el array recibido estaba vacío, es decir, no hay piezas en la base real. En ese caso destruye todos los actores del anterior mensaje que estaban en la escena y sale del programa.

Si se reciben piezas primero se destruyen aquellos actores de las celdas que han pasado a estar vacías en esta ejecución. A continuación, se recorre cada pieza de la lista. Para cada una se consulta el mapa que asocia nombres de color con clases de actor, si el color no está mapeado, se ignora la pieza y se pasa a la siguiente.

Si el color está mapeado pero la clase asignada es «None» (nada), se destruye cualquier actor que pudiese existir en esa celda, terminando la iteración. Si el color está mapeado y la clase asignada es válida, se comprueba si ya existe un actor en la celda correspondiente. Si no existe, se instancia uno nuevo con la clase correspondiente y en la posición calculada. Si existe y es de la misma clase, se mantiene sin cambios. Por el contrario, si existe pero es de distinta clase, se destruye el actor

antiguo y se genera uno nuevo de la clase correcta.

Todas las ramas del bucle convergen en un punto común, que da paso a la siguiente pieza hasta completar el array.

Este flujo garantiza que la escena de UE refleje fielmente el estado de la base física en cada actualización, incluso cuando está vacía o se producen cambios en el mapeo de colores desde el editor. La correspondencia entre colores detectados y clases de actor es totalmente configurable desde el editor sin modificar el código fuente (junto con otras variables), permitiendo incluso asignar una clase vacía a un color para que éste no genere actores en el nivel. Estos detalles de implementación se ampliarán en los apartados correspondientes a cada script.

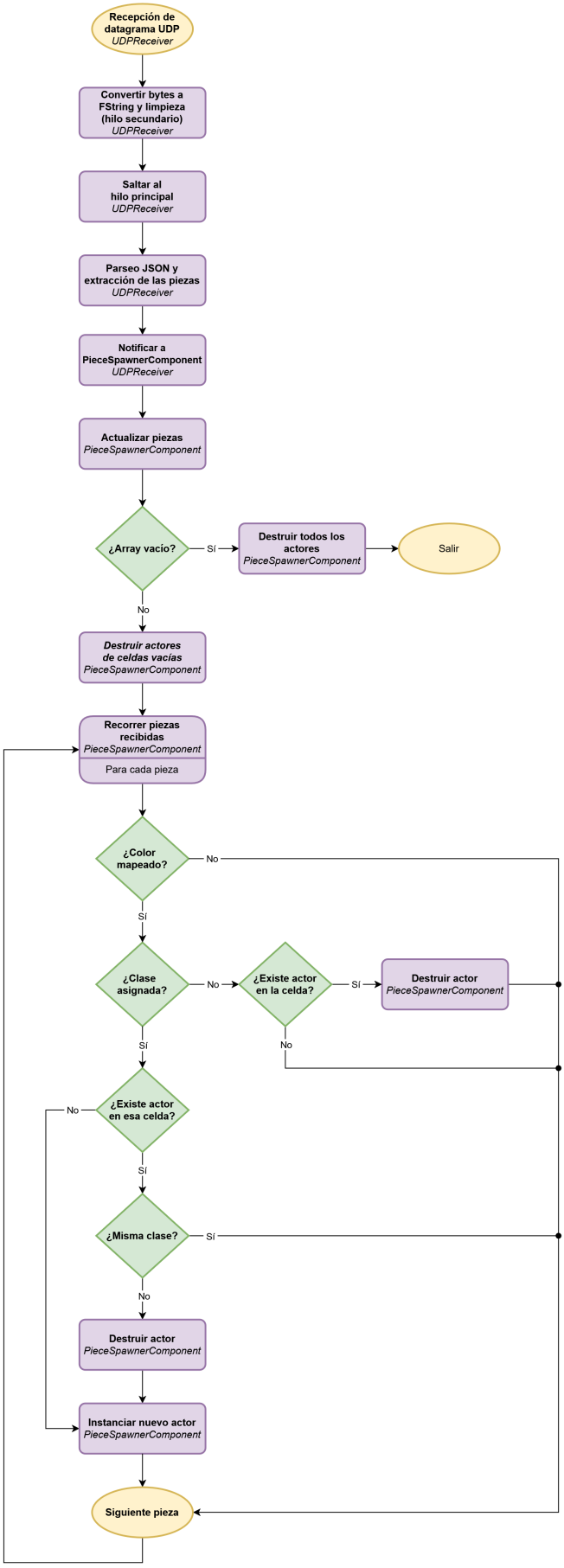

Figura 1: Diagrama del flujo del módulo de visualización

4. Receptor UDP

El actor «UDPReceiver» implementa toda la lógica de comunicación mediante el protocolo UDP. Se encarga de la recepción de los datagramas enviados por el módulo de detección (Python) y transformarlos en datos comprensibles para UE. Opera de forma asíncrona en un hilo secundario para no bloquear el juego, y expone parámetros y funciones configurables desde el editor para adaptar la conexión sin modificar el código fuente. A continuación se describen sus parámetros públicos y su funcionamiento interno.

4.1. Parámetros configurables en el editor

Este script expone varias propiedades de forma pública para que el usuario pueda ajustarlas sin tener que modificar el código fuente:

- **bAutoStart**: booleano que gestiona el inicio automático del socket del receptor UDP al comienzo.
- **SocketName**: cadena de texto que identifica al socket.
- **IP**: cadena de texto de la dirección IPv4 del emisor (módulo de detección).
- **Port**: número entero del puerto de escucha.
- **BufferSize**: tamaño del buffer de recepción, por defecto 2 MB.

Además, las siguientes funciones están accesibles en los BPs:

- **StartUDPReceiver**: control manual del inicio de la recepción UDP.
- **StopUDPReceiver**: control manual de la finalización de la recepción UDP.
- **IsReceiving**: propiedad de solo lectura para consultar el estado actual del receptor UDP.
- **OnPiecesReceived**: evento que se ejecuta al recibir y procesar las piezas, notifica al BP.

Esta arquitectura permite que un desarrollador sin conocimientos de programación por C++ pueda configurar la recepción de piezas.

En la Figura 2 se pueden observar los parámetros configurables en el editor con los valores por defecto. Mientras que la Figura 3 muestra los nodos de las funciones disponibles en el BP.

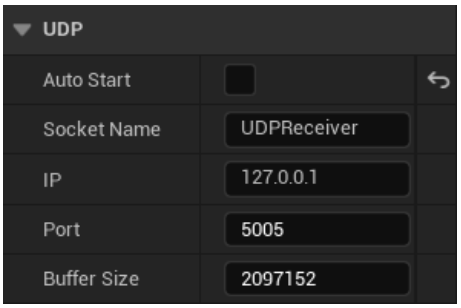

Figura 2: Parámetros del receptor UDP configurables en el editor

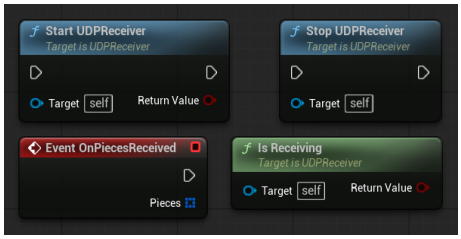

Figura 3: Funciones del receptor UDP disponibles en el BP

4.2. Funcionamiento del código

A continuación se explica el funcionamiento de los métodos del receptor UDP, incluyendo la recepción de datos, el parseo del JSON y la notificación a BP y al componente de spawneo.

4.2.1. Inicialización de la recepción

En el evento «BeginPlay» del actor, se crea un socket UDP si la propiedad «bAutostart» está activa o si se llama manualmente a «StartUDPReceiver». Este escucha permanentemente (hasta que se detiene manualmente la conexión) en un hilo secundario para no bloquear el hilo principal de juego y que pueda encargarse de la lógica de la aplicación.

4.2.2. Recepción y filtrado iniciales

Cuando el socket recibe un datagrama, se ejecuta el método «OnDataReceived» como callback. Este método construye una cadena «FString» usando la conversión de bytes a UTF-8. Entonces elimina todos los caracteres posteriores a la última llave «}» que pueden haber aparecido durante la comunicación por UDP. Si no encuentra ninguna llave de cierre, el mensaje se descarta inmediatamente.

4.2.3. Transferencia al hilo principal

Una vez que se dispone de una cadena limpia, se programa una tarea asíncrona que ejecutará el procesado del mensaje en el hilo principal. Este salto es indispen-

sable porque las operaciones de creación de actores y comunicación con BPs deben realizarse en el hilo principal para evitar comportamientos impredecibles.

4.2.4. Parseo del JSON y extracción de datos

Antes de procesar el mensaje se comprueba que el actor «UDPReceiver» y el mundo no se estén destruyendo, esto evita llamadas durante el cambio de nivel que pueden resultar en cierres inesperados de la aplicación.

Una vez realizada esta comprobación de seguridad, se deserializa la cadena de texto a un objeto JSON, si falla se aborta el procesamiento. Este objeto JSON debe tener un campo llamado «pieces» que contenga el array de piezas, en caso de no encontrarlo se aborta el resto del procesado. Para cada elemento del array se lee un nuevo objeto con los campos: «color» (cadena de texto), «x» y «y» (números enteros). Estos valores se almacenan en una estructura «FPieceData» definida en la clase. Por lo tanto, el resultado es un array de estructuras de cada pieza que representa el estado completo de la base en ese instante.

4.2.5. Delegación en el componente de spawneo

Una vez obtenido el array, se lanza el evento «OnPiecesReceived», accesible mediante Blueprints. Esto permite que los desarrolladores agreguen lógica personalizada si lo desean, sin necesidad de modificar el código C++.

Finalmente, si el actor «UDPReceiver» dispone de un componente «PieceSpawnerComponent» (debe estar añadido al actor en el editor), se invoca su método «UpdatePieces» que se encarga de instanciar los actores correspondientes a las piezas recibidas para que la escena 3D refleje el nuevo estado de la base.

4.2.6. Detención de la recepción

Cuando se destruye el actor «UDPReceiver» (por ejemplo al cambiar de nivel) o se llama manualmente a «StopUDPReceiver» se detiene el hilo secundario del receptor y se cierra y destruye el socket UDP. De esta forma se garantiza la liberación de memoria al finalizar la conexión.

5. Instanciación de actores

El componente «PieceSpawnerComponent» es el responsable de traducir los datos de las piezas (color y coordenadas de celda) a actores 3D en el nivel. Su lógica se basa en un mapeo configurable entre nombres de color y clases de actor (BP), lo que permite al desarrollador decidir qué tipo de elemento aparece para cada color sin modificar el código fuente. Además, gestiona la posición de los actores mediante una cuadrícula cuyas dimensiones (tamaño de celda y altura de spawneo) también son ajustables desde el editor.

5.1. Parámetros configurables en el editor

Este script expone las siguientes propiedades de forma pública para que el usuario pueda ajustarlas sin tener que modificar el código fuente:

- **ColorToActor**: mapa que asocia el nombre de cada color (clave) a una clase de actor (valor). El desarrollador puede añadir, modificar o eliminar entradas directamente en el editor o durante la ejecución mediante un desplegable de clases. Si se asigna «None» a la clase, ese color no generará ningún actor (útil para desactivar temporalmente un color).
- **CellSize**: distancia entre los centros de dos celdas adyacentes. Por defecto 100 y su valor no puede ser inferior a 0. Permite adaptar la herramienta a diferentes tamaños de los modelos 3D.
- **SpawnHeight**: altura Z a la que se instanciarán los actores. Por defecto es 0, es decir, el suelo. Puede ser negativa para que los actores aparezcan por debajo del nivel del suelo o positivo para elevarlos. Permite adaptar la herramienta al mapa del nivel.

Además, las siguientes funciones están accesibles en los BP:

- **UpdatePieces**: es la función que actualiza los actores del nivel a la situación actual de la base, con las piezas recibidas.
- **ClearAllPieces**: borra todos los actores instanciados en la escena.

Esta arquitectura permite que un desarrollador sin conocimientos de programación por C++ pueda configurar la instanciación de actores. En la Figura 4 se pueden observar los parámetros configurables en el editor con los valores por defecto. Mientras que la Figura 5 muestra los nodos de las funciones disponibles en el BP.

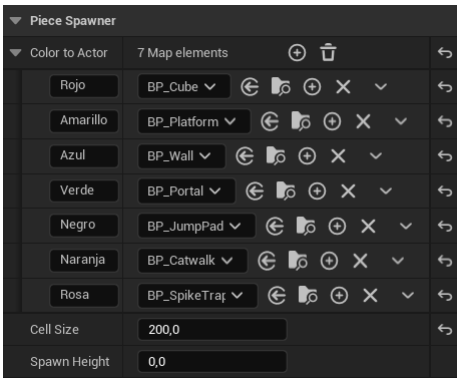

Figura 4: Parámetros del instanciador de actores configurables en el editor

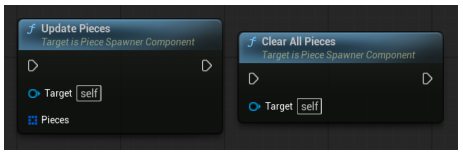

Figura 5: Funciones del instanciador de actores disponibles en el BP

5.2. Funcionamiento del código

A continuación se explica el funcionamiento de los métodos auxiliares del componente: la conversión de coordenadas, la actualización de la escena y la limpieza total de los actores.

5.2.1. Conversión de coordenadas de celda a mundo

El método auxiliar «CellToWorld» calcula la posición absoluta del centro de una celda. Convierte las coordenadas de celda recibidas a posiciones en la escena de UE. Además, asigna al eje Z la altura especificada por la variable «SpawnHeight». Entonces el actor instanciado se situará en el centro de la celda y a la altura configurada.

5.2.2. Actualización de la escena

El flujo general de actualización de la escena se puede consultar en el diagrama de flujo de la Figura 1 del Capítulo 3. En este apartado se describen únicamente los aspectos de implementación más relevantes.

El método «UpdatePieces» recibe el array de piezas y actualiza el nivel de forma diferencial en lugar de destruir y recrear todos los actores cada vez. Primero comprueba que el array no está vacío, si lo está elimina todos los actores generados.

A continuación construye un conjunto (set) con las coordenadas de las celdas activas (las que aparecen en el nuevo estado). Este conjunto sirve para comprobar fácilmente qué actores previos no se conservan en el nuevo estado y por lo tanto deben eliminarse.

Luego se recorren todas las piezas del array, realizando las siguientes comprobaciones:

- Se consulta el mapa «ColorToActor» con el color de la pieza. Si no existe la entrada, el color no está mapeado y se ignora esta pieza.
- Si la clase asociada es nula, se destruye cualquier actor que hubiera en esa celda. Esto permite que un color no genere actor (útil para desactivar temporalmente un tipo de pieza).
- Si la clase es válida, se comprueba si ya existe un actor en la celda:
  - Si existe y es de la misma clase, se conserva sin cambios.
  - Si existe pero es de una clase distinta, se destruye el actor antiguo y se instancia con la clase correspondiente.
  - Si no existe, se spawnea un nuevo actor con la clase correspondiente y la posición devuelta por «CellToWorld».
- Añade el nuevo actor instanciado (si se ha creado) al mapa «CurrentActorsByCell». La clave es las coordenadas de la celda y el valor la clase instanciada.

Este algoritmo garantiza que la escena refleje fielmente el estado de la base física con el mínimo coste computacional, y permite cambiar la asignación de colores a actores durante la ejecución sin afectar a la lógica del componente.

5.2.3. Limpieza total de la escena

El método «ClearAllPieces» itera sobre todos los actores almacenados en «CurrentActorsByCell», los destruye y vacía el mapa «CurrentActorsByCell». Se utiliza principalmente cuando se recibe un array vacío o cuando se desea resetear manualmente el nivel.

6. Integración en motores de videojuegos

El módulo de visualización se ha diseñado siguiendo una arquitectura desacoplada, lo que facilita su traslado a otros motores de videojuegos o su reutilización en distintos proyectos dentro del mismo motor. A continuación se describen los aspectos clave de esta integración y se proporciona un breve manual de uso para UE.

6.1. Compatibilidad con diferentes motores

La herramienta se compone del módulo de detección y el módulo de visualización expuesto en el presente anexo. Ambos módulos son independientes entre sí, el módulo de detección se encarga de localizar y clasificar piezas de colores sobre una base, mientras que el módulo de visualización se encarga de recibir las piezas detectadas e instanciarlas en el mapa.

El módulo de visualización se encuentra integrado en el proyecto del motor de videojuegos. Debido a que la comunicación entre ambos módulos se realiza mediante el protocolo estándar UDP es posible emplear cualquier motor de videojuegos capaz de manejar un socket UDP. Además, el mensaje enviado que contiene las piezas detectadas está escrito en formato JSON, que tiene una estructura claramente definida y manejable. En este caso se desarrolló el módulo de visualización para UE, se compone de dos partes bien diferenciadas:

- **Receptor UDP**: encargado de gestionar la comunicación UDP y el procesamiento del mensaje JSON.
- **Instanciador de actores**: encargado de gestionar la instanciación de actores en el nivel según el color asignado y detectado en cada pieza recibida.

Ambos códigos se pueden adaptar a otros motores de videojuegos replicando la lógica de los scripts. Se debería ajustar la recepción UDP al método de gestión de sockets en el motor de videojuegos y la instanciación de actores al sistema de creación de objetos. Por lo tanto, este módulo tiene una alta portabilidad hacia otros motores, aunque la parte de edición visual en el editor requeriría adaptarse a las herramientas de cada motor.

6.2. Reutilización en proyectos de UE

El módulo está preparado para ser copiado e integrado en cualquier proyecto de UE sin necesidad de modificar su código interno. Los pasos concretos se detallan en el Anexo E (Manual de instrucciones).

De forma resumida, el proceso consiste en:

1. En el proyecto de UE se deben crear las clases C++ con los nombres y tipo exactos: «UDPReceiver» de tipo Actor y «PieceSpawnerComponent» de tipo ActorComponent. Compilar el proyecto para que UE reconozca las nuevas clases C++.

2. En los archivos generados se debe copiar el código desarrollado para el módulo de visualización. Otra opción es reemplazar estos archivos generados.

3. Se debe sustituir el proyecto API en las declaraciones de las clases (CHROMAVISIONUE_API) por el del proyecto de destino en mayúsculas (EJEMPLO_API).

4. Añadir las dependencias «Sockets», «Networking», «Json» en el archivo «Build.cs» del proyecto.

5. Crear un BP a partir de la clase «AUDPReceiver» y agregarle un componente «PieceSpawnerComponent».

6. Configurar en el componente el mapa «ColorToActor» asignando a cada color un BP.

7. Colocar una instancia del actor «BP_UDPReceiver» en el nivel. La conexión se inicia automáticamente (aunque se puede cambiar a realizarlo de forma manual) y sus valores por defecto coinciden con la configuración del módulo de detección.

Una vez realizado esto, al ejecutar el proyecto junto con el script de Python, las piezas detectadas aparecerán en la escena. Después el desarrollador puede configurar el resto de parámetros según sus necesidades.

6.3. Configuración en el editor

Como se ha mencionado con anterioridad el módulo expone parámetros y funciones relevantes en el editor y son accesibles desde BPs. Esto permite a un diseñador de niveles o a un artista adaptar el comportamiento de la herramienta sin necesidad de escribir código C++. Se pueden configurar parámetros de: la conexión UDP, la asignación de colores a actores, el tamaño de las celdas o la altura de spawneo. Además el evento «OnPiecesReceived», implementable en BPs, permite reaccionar a la llegada de piezas, por ejemplo añadiendo efectos de sonido o cualquier otra lógica adicional.

7. Calibración dinámica de colores (descartada)

Durante el desarrollo se exploró una funcionalidad avanzada que permitiera modificar en tiempo de ejecución los rangos de detección de color desde el editor de UE, sin necesidad de reiniciar el script Python o modificar su código. El objetivo era que un desarrollador pudiera ajustar los umbrales HSV de cada color visualmente (mediante un selector de color) y que los cambios se aplicaran al instante en el módulo de detección. Aunque se llegó a implementar un prototipo funcional, finalmente se decidió no incluirlo en la versión final de la demo por las razones que se exponen a continuación.

7.1. Planteamiento

El sistema de calibración dinámica pretendía superar la limitación de tener que editar manualmente el archivo de configuración de Python permitiendo ajustar los rangos de forma visual en UE. Para ello se diseñó una arquitectura bidireccional:

- **UE**: se creó un componente C++ llamado «ColorCalibrationComponent» que exponía en el editor una lista de colores. Cada color disponía de un selector RGB (color representativo en la interfaz del módulo de detección) y rangos ajustables HSV. UE trabaja en el espacio de color RGB por lo que convertía el color seleccionado por el usuario a un rango HSV. Cada vez que se detectaba un cambio en la configuración se enviaba un comando JSON a un segundo puerto UDP (5006).
- **Python**: se añadió un hilo receptor en un nuevo archivo «udp_receiver.py» que escuchaba los comandos de UE, los procesaba y actualizaba la lista global «COLORS» gestionando su disponibilidad para el hilo principal y secundario. Los rangos HSV de OpenCV (H: 0-180, S: 0-255, V: 0-255) son distintos a los de UE (H: 0-360, S: 0-1, V: 0-1) por lo que se realizaba una conversión antes de actualizar la lista «COLORS».

7.2. Motivos de descarte

A pesar de que el sistema funcionaba correctamente, se optó por no integrarlo en la aplicación final por los siguientes motivos:

- El sistema pasaba a tener una comunicación UDP bidireccional entre Python y UE, en lugar de la comunicación unidireccional original (solo Python → UE). Esto añadía complejidad de gestión de puertos, hilos adicionales, etc.
- La conversión de rangos HSV entre UE y OpenCV añadía complejidad y restaba flexibilidad, además de dificultar la posible portabilidad del sistema a otros motores, ya que cada motor puede manejar rangos HSV distintos.
- Debido a que OpenCV trabaja en el espacio de color HSV y UE en RGB escoger rangos de color no resultaba visualmente intuitivo.

Por estos motivos se descartó implementar esta funcionalidad en la aplicación final, priorizando otros aspectos del desarrollo. En la Figura 6 se puede observar el prototipo desarrollado, en el editor estaba disponible una lista llamada «Colors» que permitía añadir, modificar y eliminar colores, viéndose esto reflejado en Python. En este ejemplo se modificó el valor representativo del color rojo mediante un selector, afectando a su representación en la interfaz de Python.

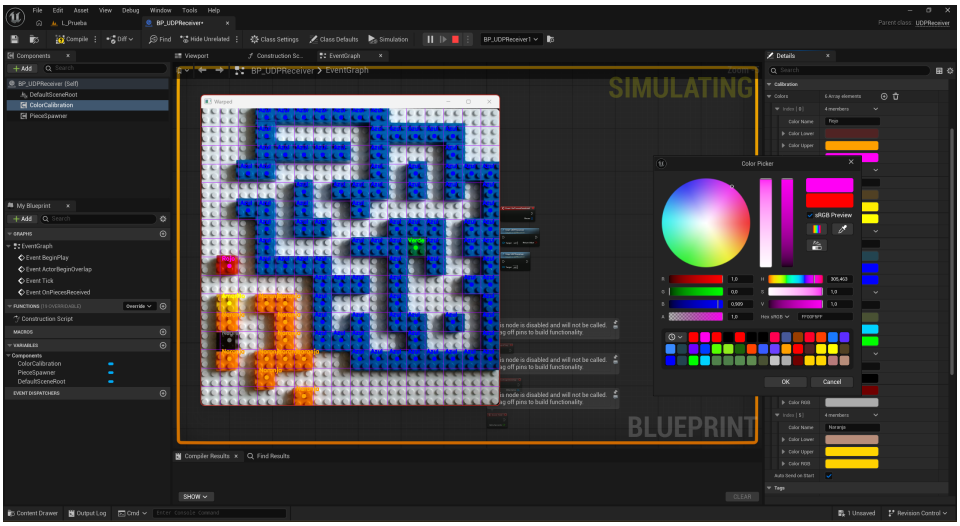

Figura 6: Prototipo del sistema de calibración dinámica de colores

Finalmente se ofreció al usuario la posibilidad de modificar, añadir o eliminar colores desde el archivo «config.json» incluso en tiempo de ejecución. De esta forma se cumple el requisito funcional RF-06 del Anexo A.

Anexo D

Aplicación de prototipado de niveles (Unreal Engine)

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela **Politécnica Superior de Zamora**

Glosario

Visión artificial Campo de la inteligencia artificial que permite a las computadoras interpretar y procesar imágenes y vídeos. 1, 14

Siglas

3D tres dimensiones. 1, 2, 7, 8, 10–12, III, IV

BP Blueprint. 3, 5, 7, 8, 12, 18, 19, III, IV

FPS fotogramas por segundo. 22, V

IP Internet Protocol. 21

RGB Red, Green, Blue. 14

UDP Protocolo de Datagramas de Usuario. 2, 3, 22, IV

UE Unreal Engine. 1, 3, 7, 10, 11, 14, IV

VFX efectos visuales. 3, 9, 12, III

Índice general

Glosario ..... **I**

Siglas ..... **II**

1 Introducción **1**

2 Modos de la aplicación **2**

2.1 Modo construcción 2

2.2 Modo juego 2

3 Aspectos relevantes **3**

3.1 Gestión de modos y control de la conexión UDP 3

3.2 Vistas de la cámara 3

3.3 Mapa 4

3.4 Respawn del personaje al caer del mapa 5

3.5 Spawneo del personaje mediante piezas 5

3.6 Volumen y personaje 7

4 Actores **8**

4.1 Bocetos 9

4.2 Modelos 3D 10

4.3 Texturas y materiales 10

4.4 VFX 12

4.5 BPs 12

5 Interfaces **13**

5.1 Diagrama de navegación 13

5.2 Marca 14

5.3 Menú principal 15

5.4 Modo construcción 16

5.5 Asignación de colores a actores 18

5.6 Modo juego 19

5.7 Menú de pausa 20

5.8 Pantalla de victoria 20

5.9 Pantalla de derrota 21

5.10 Variables accesibles desde la interfaz 21

6 Métricas de rendimiento **22**

Índice de figuras

1 Botón para activar o desactivar la conexión UDP en el modo construcción 3
2 Cámaras situadas en el mapa y visibles en el Outliner 4
3 Mapa de la aplicación rodeado de «BlockingVolumes» y un «PostProcessVolume» 5
4 Nivel de ejemplo (laberinto) con el personaje en el centro 6
5 Disposición real de las piezas LEGO sobre la base para generar un laberinto 6
6 Mensaje de advertencia al colocar múltiples piezas que representen el personaje 7
7 Actores creados para la demostración, cada uno asociado a un color de pieza 8
8 Bocetos iniciales de los actores 9
9 Mallas 3D modeladas en Blender e importadas a UE 10
10 Texturas de los modelos 3D de los actores 10
11 Todas las texturas importadas en el editor de UE 11
12 Todos los materiales creados en el editor de UE 11
13 BPs de los objetos para el prototipado de niveles 12
14 Diagrama de navegación entre las interfaces y modos de la aplicación . . . 13
15 Logotipo de la herramienta ChromaVision 14
16 Icono de la herramienta ChromaVision 14
17 Paleta de colores utilizada para las interfaces y el logotipo 14
18 Interfaz del menú principal 15
19 Modo construcción con la vista cenital rotada 16
20 Disposición de las piezas LEGO sobre la base para diseñar un nivel de ejemplo 16
21 Vista del nivel desde una cámara fija situada en la esquina 17
22 Cámara libre con la interfaz de construcción oculta 17
23 Cámara libre con la interfaz de construcción visible 18
24 Menú selector de actores en el BP 19
25 Menú selector de actores relleno con las opciones disponibles 19
26 Interfaz del modo de juego 19
27 Menú de pausa 20
28 Pantalla de victoria cuando el jugador alcanza el portal 20
29 Pantalla de derrota cuando el jugador muere 21

Índice de tablas

1 Resultados de evaluación de rendimiento (FPS) de la aplicación 22

1. Introducción

Para demostrar la utilidad de la herramienta, se ha desarrollado una aplicación de ejemplo en Unreal Engine (UE) que permite construir un nivel en un entorno 3D mediante la colocación de piezas LEGO físicas sobre una base y de forma sincronizada. Las piezas detectadas mediante visión artificial utilizando una webcam se traducen automáticamente al actor 3D correspondiente a su color. El usuario puede escoger qué elementos asigna a cada color (plataforma, bloque, muro, portal, etc.) pudiendo incluso desactivar temporalmente la detección de un color. Una vez construido el nivel, es posible cambiar a un modo juego para recorrerlo con un personaje, demostrando así la utilidad de la herramienta para el prototipado rápido de niveles.

Este anexo describe los aspectos más relevantes de la aplicación: los modos de funcionamiento, las funcionalidades destacadas, los assets creados, las interfaces de usuario y las métricas de rendimiento.

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Modos de la aplicación

La aplicación se organiza en dos modos: construcción y juego. A continuación se detallan las funcionalidades de cada uno. Las interfaces correspondientes a cada modo se muestran en el Capítulo 5.

2.1. Modo construcción

En el modo construcción, el usuario coloca piezas LEGO sobre la base y la aplicación muestra los objetos 3D correspondientes (plataformas, muros, trampas, etc.) de forma sincronizada. Desde la interfaz se pueden realizar las siguientes acciones:

- **Cambiar la vista de la escena**: una cruceta permite seleccionar entre varias vistas fijas y rotar la cenital para alinearla con el ángulo real de la cámara. Otro botón permite activar una cámara libre para inspeccionar el nivel desde cualquier ángulo.
- **Control de la conexión UDP**: pausar o reanudar la recepción de datos de piezas.
- **Borrado manual**: eliminar todos los actores generados.
- **Reasignación de actores a colores**: modificar la clase de objeto asociada a cada color. Se puede incluso asignar «Nada» para desactivar la detección del color temporalmente.
- **Cambio al modo juego**: transición al modo de prueba del nivel diseñado.
- **Opciones generales**: ajuste de volumen de audio, retorno al menú principal y salida de la aplicación.

2.2. Modo juego

Una vez construido el nivel, se puede cambiar al modo juego. En este modo, el usuario controla un personaje en primera persona capaz de recorrer el nivel, interactuar con los objetos, recibir daño y alcanzar un objetivo (llegar al portal para ganar). La tecla `F` permite volver al modo construcción.

Durante la partida, se puede acceder a un menú de pausa pulsando la tecla `ESC`, que ofrece las opciones de reanudar, ajustar el volumen, volver al menú principal o cerrar la aplicación. Al alcanzar el portal se muestra una pantalla de victoria. Si la salud del jugador llega a cero, una pantalla de derrota. Desde estas pantallas se puede regresar al modo construcción, al menú principal o salir de la aplicación.

3. Aspectos relevantes

Esta aplicación de ejemplo desarrollada en UE es extensa e incluye numerosas funcionalidades (menús, interfaces, VFX, gestión de modos, etc.). No obstante, el foco principal de este TFG es el desarrollo de la herramienta colaborativa (detección de piezas, comunicación UDP y visualización en un motor de videojuegos, en este caso UE). Por ello, en este anexo no se pretende describir la aplicación en su totalidad ni detallar el código completo, sino demostrar la utilidad de la herramienta mediante un caso práctico.

A continuación se mencionan algunos aspectos y funcionalidades relevantes sin entrar en profundidades técnicas ni en el código fuente, desarrollado mediante «BPs».

3.1. Gestión de modos y control de la conexión UDP

Cuando se cambia del modo de construcción al modo de juego se detiene automáticamente la recepción UDP para evitar que el nivel se modifique mientras el jugador recorre la escena. Al volver al modo construcción, la conexión permanece desactivada a menos que el usuario la reactive manualmente con el botón, porque de lo contrario se sobrescribiría el estado del nivel que el jugador acaba de probar por el actual de la base. Esta decisión garantiza una experiencia estable y predecible para el usuario, evitando modificaciones del nivel no deseadas. La Figura 1 muestra el botón que permite activar y desactivar la recepción de piezas del módulo de detección.

Figura 1: Botón para activar o desactivar la conexión UDP en el modo construcción

3.2. Vistas de la cámara

En el modo construcción se puede cambiar de vista mediante una cruceta en la interfaz. Esto se logra colocando nueve cámaras fijas en el mapa y cambiando entre ellas mediante el nodo «Set View Target With Blend» que logra un movimiento suave en la transición. Además, la cámara cenital se puede rotar en incrementos de 90 grados para alinearla con la orientación de la webcam real y facilitar la orientación

espacial del usuario al manipular piezas o exponer el nivel a otras personas. En la Figura 2 se puede observar el mapa en el editor, con las cámaras resaltadas en amarillo y el Outliner que muestra los elementos instanciados en él.

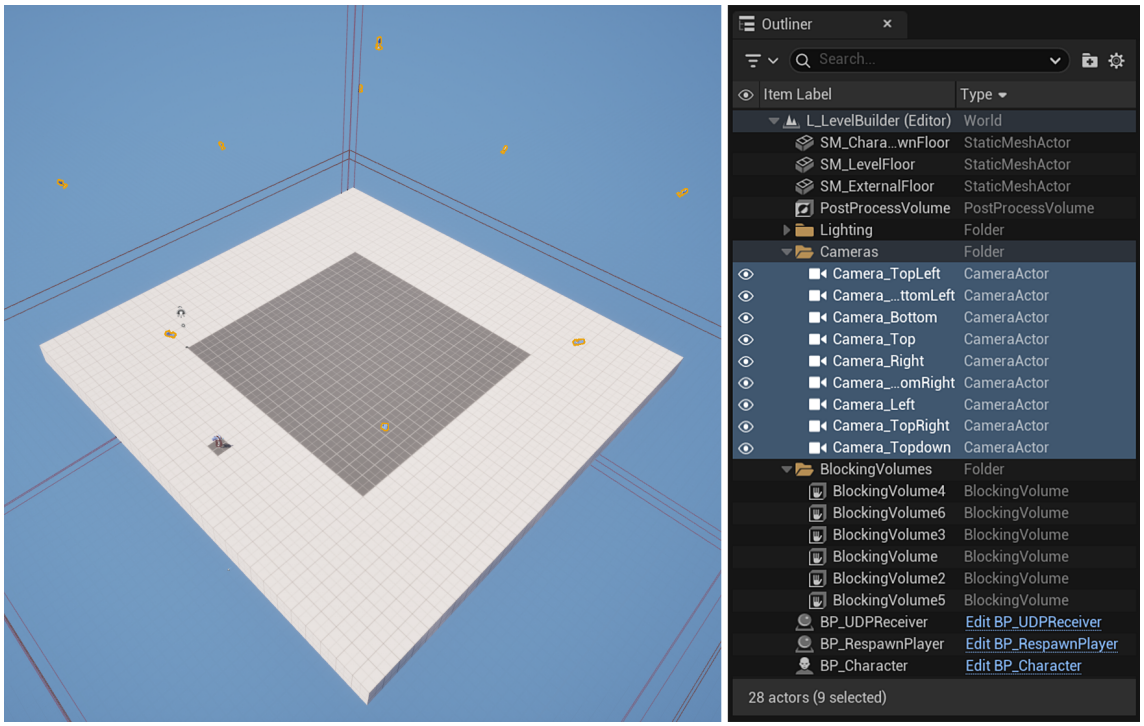

Figura 2: Cámaras situadas en el mapa y visibles en el Outliner

3.3. Mapa

A parte de las cámaras mencionadas en el apartado anterior, también se implementó una cámara libre (espectador) que el usuario puede usar para observar el nivel desde la perspectiva que desee. Se activa mediante un botón en la interfaz del modo construcción y puede desplazarse con las teclas `WASD`, subir o bajar con `Q/E` y rotar con el ratón. Para evitar que el usuario pueda alejarse indefinidamente del mapa, se ha rodeado el área con actores de tipo «BlockingVolumes» (Figura 3).

Además el mapa se ha englobado en un «PostProcessVolume» para reducir el contraste y saturación lumínica en el nivel.

Por otro lado, en la Figura 3 se puede observar que el mapa consiste en un bloque que representa el suelo. Tiene dos materiales, ambos consisten en una cuadrícula configurable del mismo tamaño que las celdas de spawneo. La parte gris delimita la zona de generación de piezas, es decir, la base real de LEGO. Mientras que la zona gris más pequeña indica la posición del personaje por defecto que el usuario controla en el modo juego.

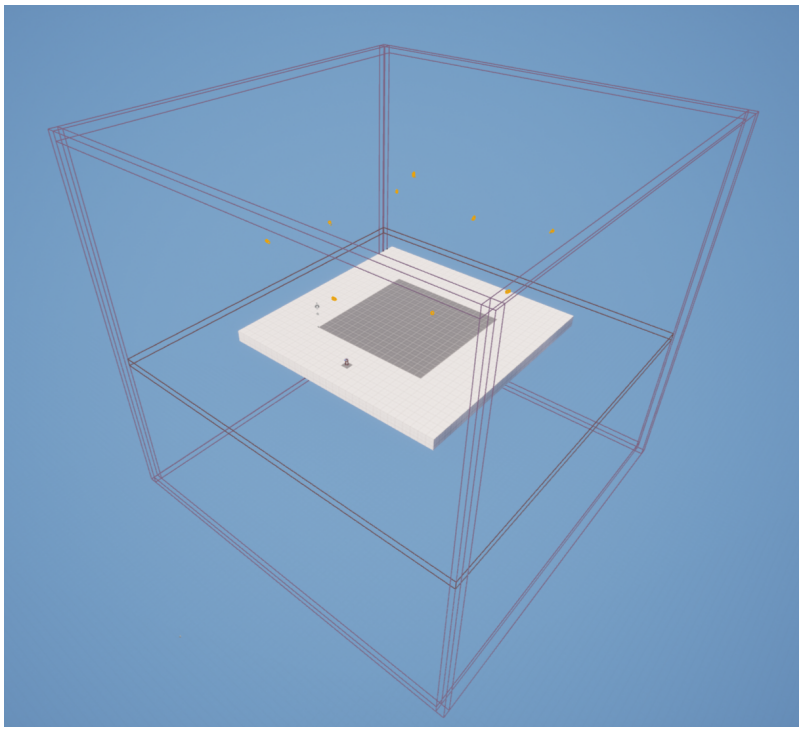

Figura 3: Mapa de la aplicación rodeado de «BlockingVolumes» y un «PostProcessVolume»

3.4. Respawn del personaje al caer del mapa

Como se ha podido observar en la Figura 3, el mapa consiste en un bloque suspendido en el vacío, que para la demostración de la herramienta es suficiente. Sin embargo, el jugador puede caer al vacío, es por ello que se ha añadido una colisión debajo del suelo (Figura 3) que detecta al personaje. Entonces lo teletransporta a la posición desde la que cayó. Para ello, se utiliza un «Line Trace» programado en el BP del personaje que registra la última posición válida en la que ha estado.

3.5. Spawneo del personaje mediante piezas

Una de las funcionalidades más interesantes es la posibilidad de ubicar el punto de inicio del personaje mediante una pieza de color específico. En la Figura 4 se puede ver, en el menú de conversión color/actor, que es posible escoger el personaje. Por defecto, el personaje aparece fuera de la zona de generación (en una celda fija) pero si se coloca una pieza con él asignado, esa celda se convierte en el punto de inicio del nivel. Al cambiar al modo juego, el jugador poseerá automáticamente el personaje ubicado en esa celda.

La Figura 4 muestra un nivel de ejemplo, un laberinto, en el que el personaje spawnea en el centro y debe encontrar la salida. En esta demostración la clase del

personaje está asignada al color naranja.

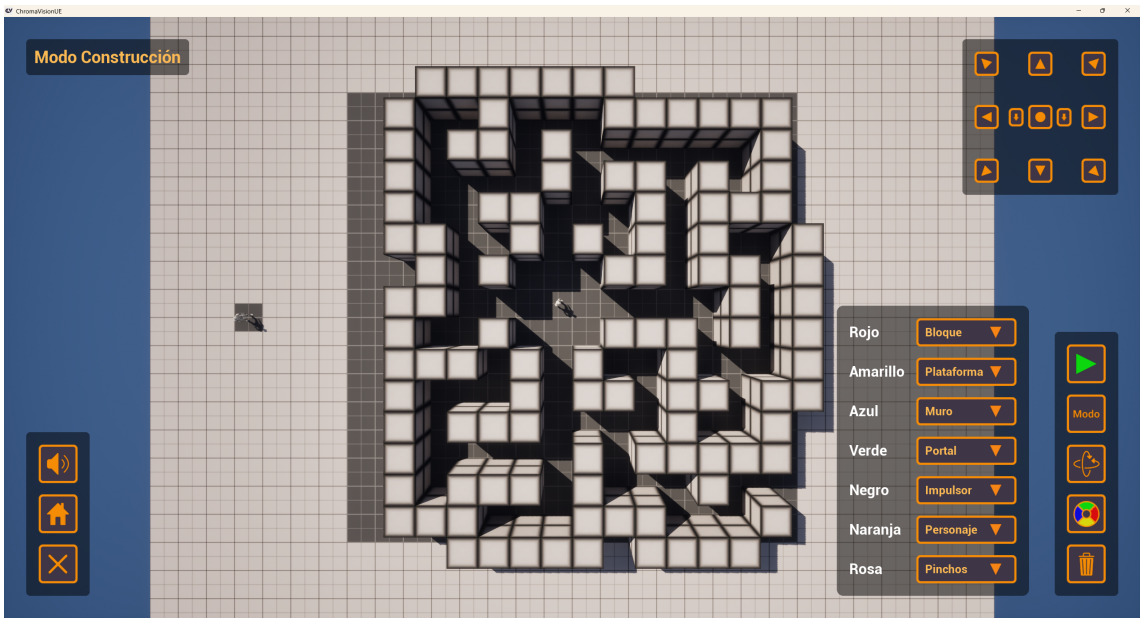

Figura 4: Nivel de ejemplo (laberinto) con el personaje en el centro

La Figura 5 muestra la disposición real de las piezas sobre la base. Se ha posicionado un gran número de piezas, especialmente en los bordes.

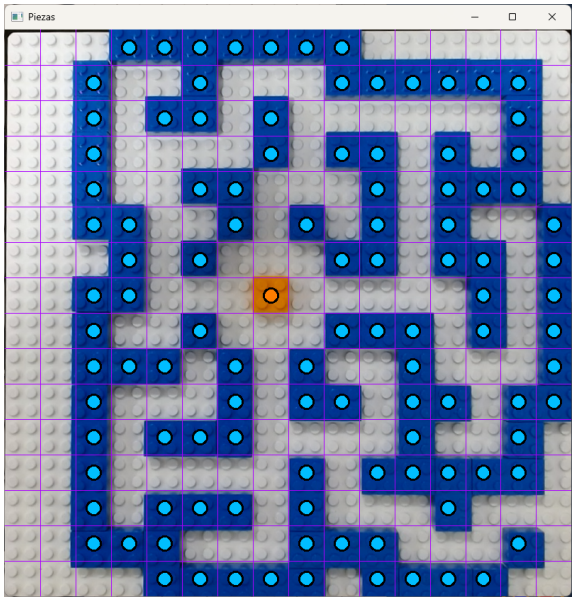

Figura 5: Disposición real de las piezas LEGO sobre la base para generar un laberinto

En caso de que el usuario coloque más de una pieza que instancie al personaje, se muestra un mensaje de advertencia en la interfaz (Figura 6) indicando que solo se usará la primera detectada.

Figura 6: Mensaje de advertencia al colocar múltiples piezas que representen el personaje

Esta característica demuestra la versatilidad del sistema: es posible asignar cualquier actor a cualquier color detectado.

3.6. Volumen y personaje

Se ha programado una barra de sonido que permite al usuario modificar el volumen de todos los sonidos (master) de la aplicación. Hay diversos sonidos implementados: click de botones, ganar, perder y efectos de sonido de los actores 3D. El valor del volumen es persistente a lo largo de los niveles gracias a un «GameInstance», por lo que la interfaz y el sonido siempre coincidirán en el menú principal y el nivel.

Por otro lado, cabe destacar que se ha utilizado el modelo y animaciones del personaje en primera persona estándar de UE para poder centrarse en el desarrollo de la aplicación. El movimiento y la lógica del personaje (salto, correr, daño, etc.) han sido implementados en su BP. No se ha modificado el modelo ni las animaciones por defecto, pero toda la mecánica jugable ha sido desarrollada específicamente para esta demostración.

4. Actores

Para esta aplicación de demostración se crearon varios actores (BP) que se asocian a los colores detectados por el módulo de visión. Cada uno cumple un papel específico dentro del prototipado de niveles y aparecen en la Figura 7:

- **Bloque (rojo)**: bloques que el jugador no puede saltar.
- **Plataforma (amarillo)**: se mueven verticalmente, permitiendo al jugador alcanzar zonas más altas.
- **Muro (azul)**: muros de doble altura que el jugador no puede saltar, ni siquiera usando otros objetos 3D.
- **Portal (verde)**: al tocarlo el jugador completa el nivel (objetivo del juego).
- **Impulsor (negro)**: propulsa al jugador hacia arriba cuando se sitúa sobre él.
- **Pasarela (naranja)**: permite al jugador caminar por debajo y por encima de ella.
- **Trampa de pinchos (rosa)**: los pinchos suben y dañan al jugador si pasa por encima de ella.

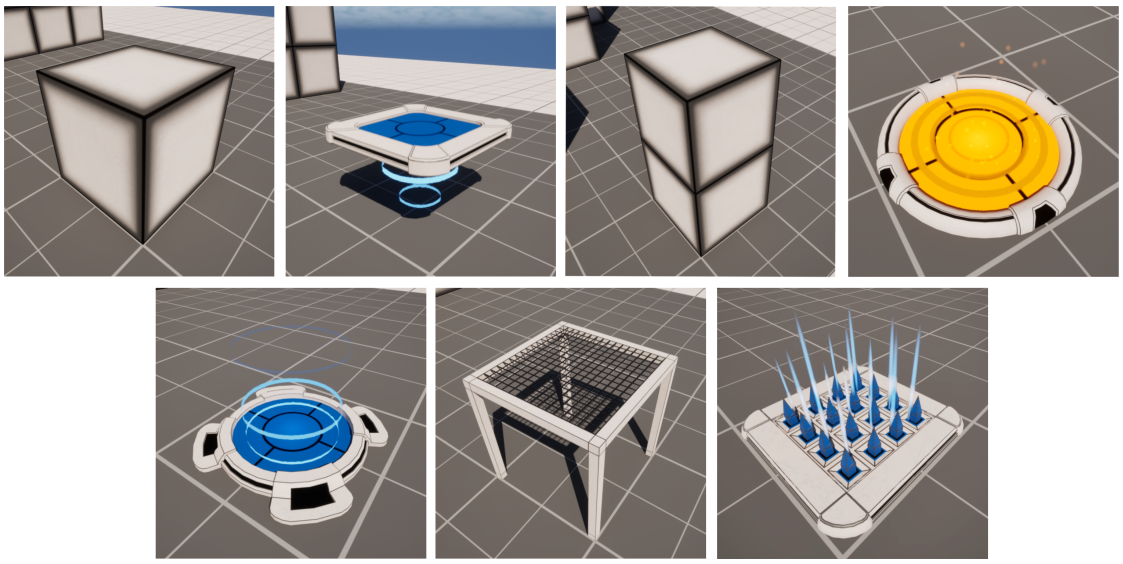

Figura 7: Actores creados para la demostración, cada uno asociado a un color de pieza

4.1. Bocetos

Primero se realizaron bocetos en papel de los diferentes tipos de actores con el objetivo de que fueran fácilmente distinguibles. Se emplea el color azul en los materiales y VFX para indicar que el objeto es interactuable y no estático. En el caso del portal, en la implementación final la parte central es naranja para distinguirlo del resto de elementos ya que es el objetivo final del juego, que da la victoria al jugador. La Figura 8 contiene los bocetos iniciales, se pueden observar ideas no implementadas como puertas, torretas o cubos desplazables.

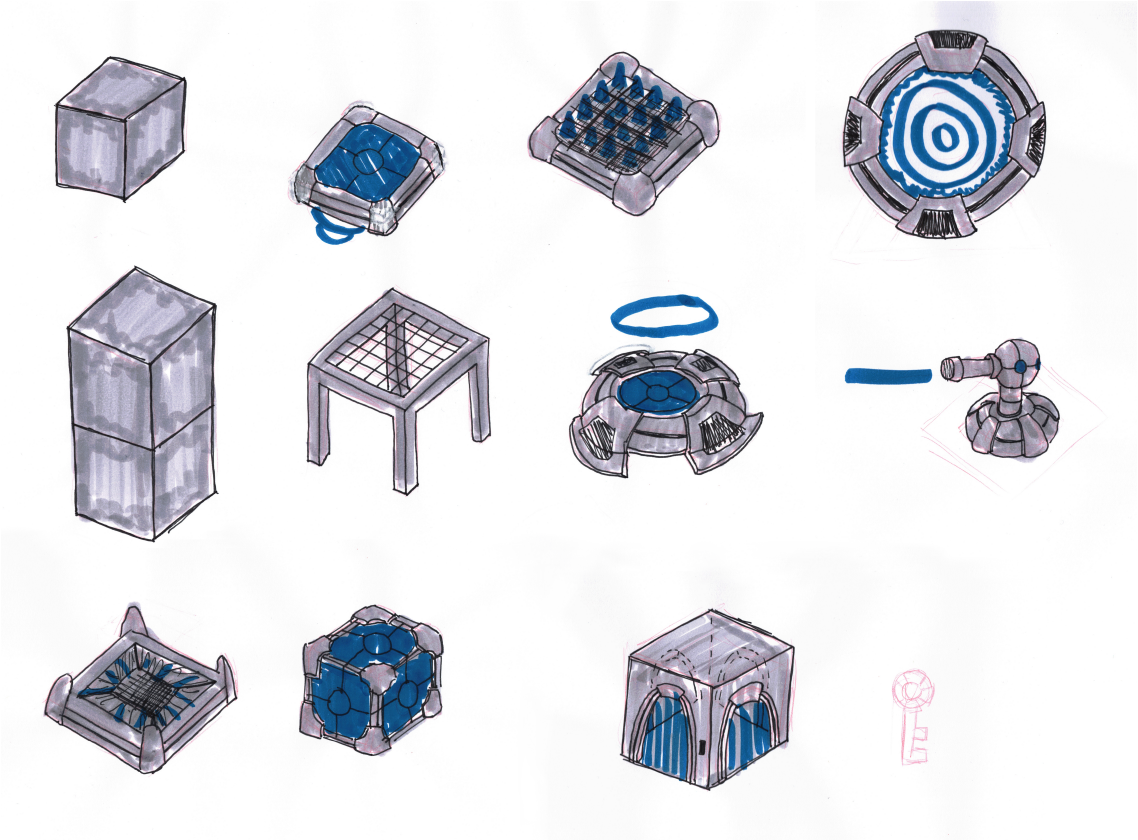

Figura 8: Bocetos iniciales de los actores

4.2. Modelos 3D

Se modelaron las mallas empleando el software Blender y posteriormente se importaron al motor UE, retocando sus colisiones. En la Figura 9 se pueden observar las mallas modeladas importadas en UE.

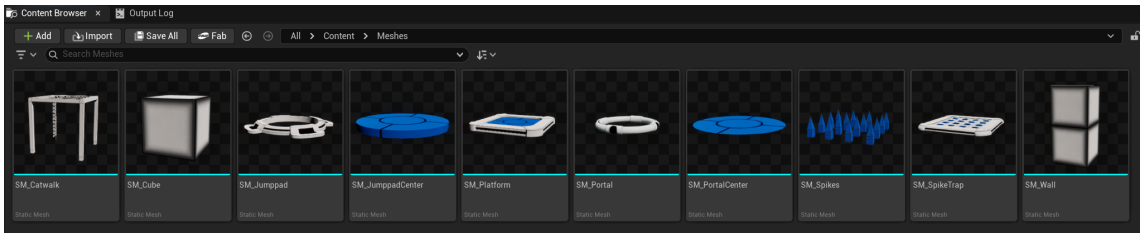

Figura 9: Mallas 3D modeladas en Blender e importadas a UE

4.3. Texturas y materiales

Se crearon texturas para cada objeto 3D usando el software Krita. Además, se realizó el unwrap de las UVs de los modelos. En la Figura 10 se pueden observar las principales texturas diseñadas.

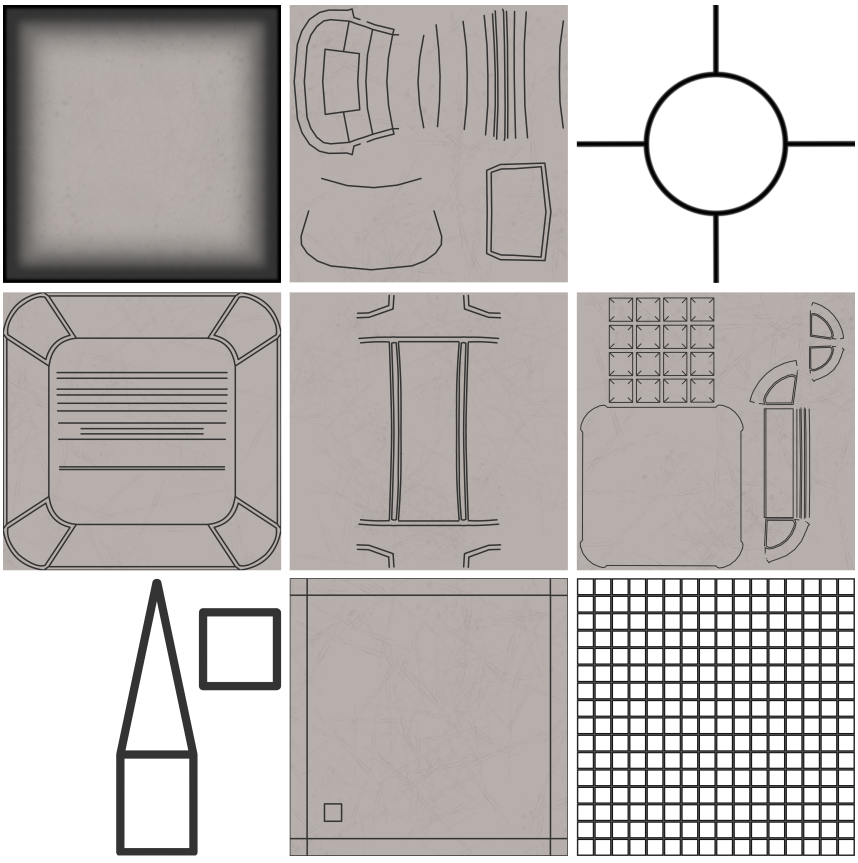

Figura 10: Texturas de los modelos 3D de los actores

Todos los objetos tienen efectos de arañazos y manchas que se ven acentuados por el mapa de normales y el resto de texturas creadas a partir de las anteriores. En la Figura 11 aparecen todas las texturas creadas importadas en UE.

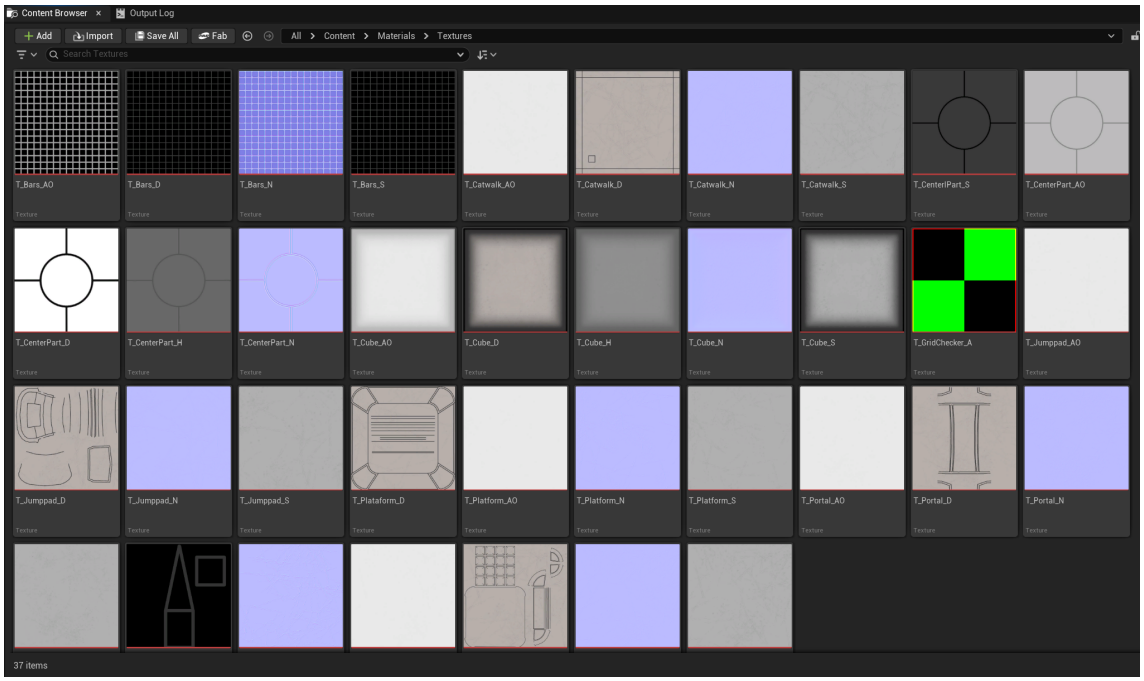

Figura 11: Todas las texturas importadas en el editor de UE

Estas texturas se integraron en materiales que se asignaron a las mallas de los objetos. En la Figura 12 se muestran todos los materiales creados para los modelos 3D implementados. Cabe destacar que el material y las texturas de la cuadrícula del suelo son una modificación de los existentes por defecto en UE.

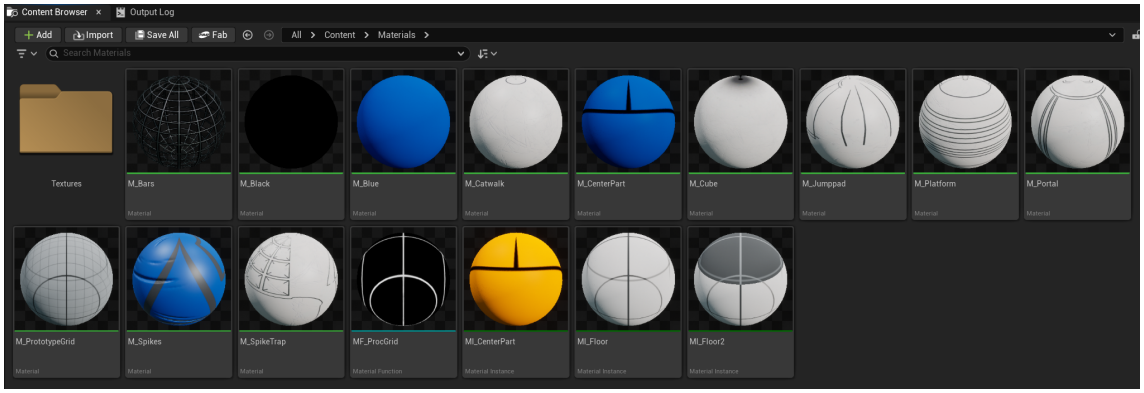

Figura 12: Todos los materiales creados en el editor de UE

4.4. VFX

También se crearon VFX que aportan dinamismo a los objetos. Como se puede observar en la Figura 7 tienen VFX: el impulsor, la plataforma, el portal y la trampa de pinchos. Todos los efectos tienen color azul siguiendo la temática de los materiales.

4.5. BPs

Para poder asignar los modelos 3D a los colores de las piezas, programar su comportamientos e integrar las mallas modeladas se crearon BPs de tipo actor para cada objeto. En la Figura 13 se pueden observar los BPs creados.

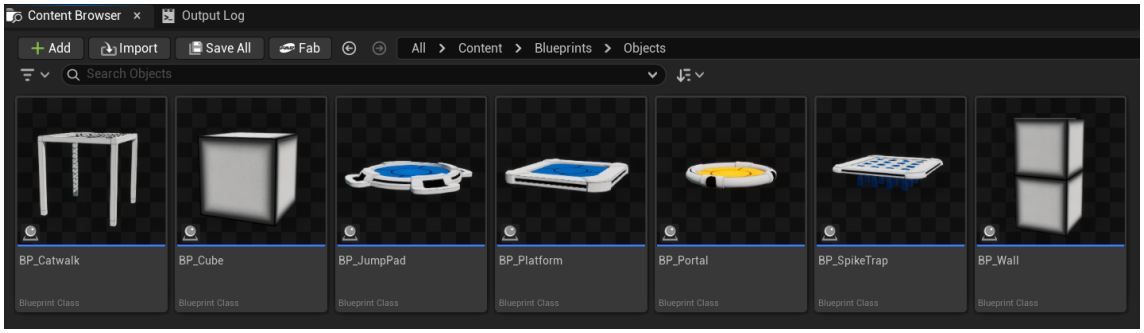

Figura 13: BPs de los objetos para el prototipado de niveles

5. Interfaces

En este apartado se describen las interfaces de usuario de la aplicación junto con un diagrama de navegación entre las distintas pantallas y modos.

5.1. Diagrama de navegación

La Figura 14 ilustra la navegación entre el menú principal, el modo construcción, el modo juego, las pantallas de victoria/derrota y el menú de pausa. Las flechas indican la acción que provoca cada transición (pulsar un botón, tecla o evento de juego). En los siguientes apartados se mostrará cada una de las interfaces.

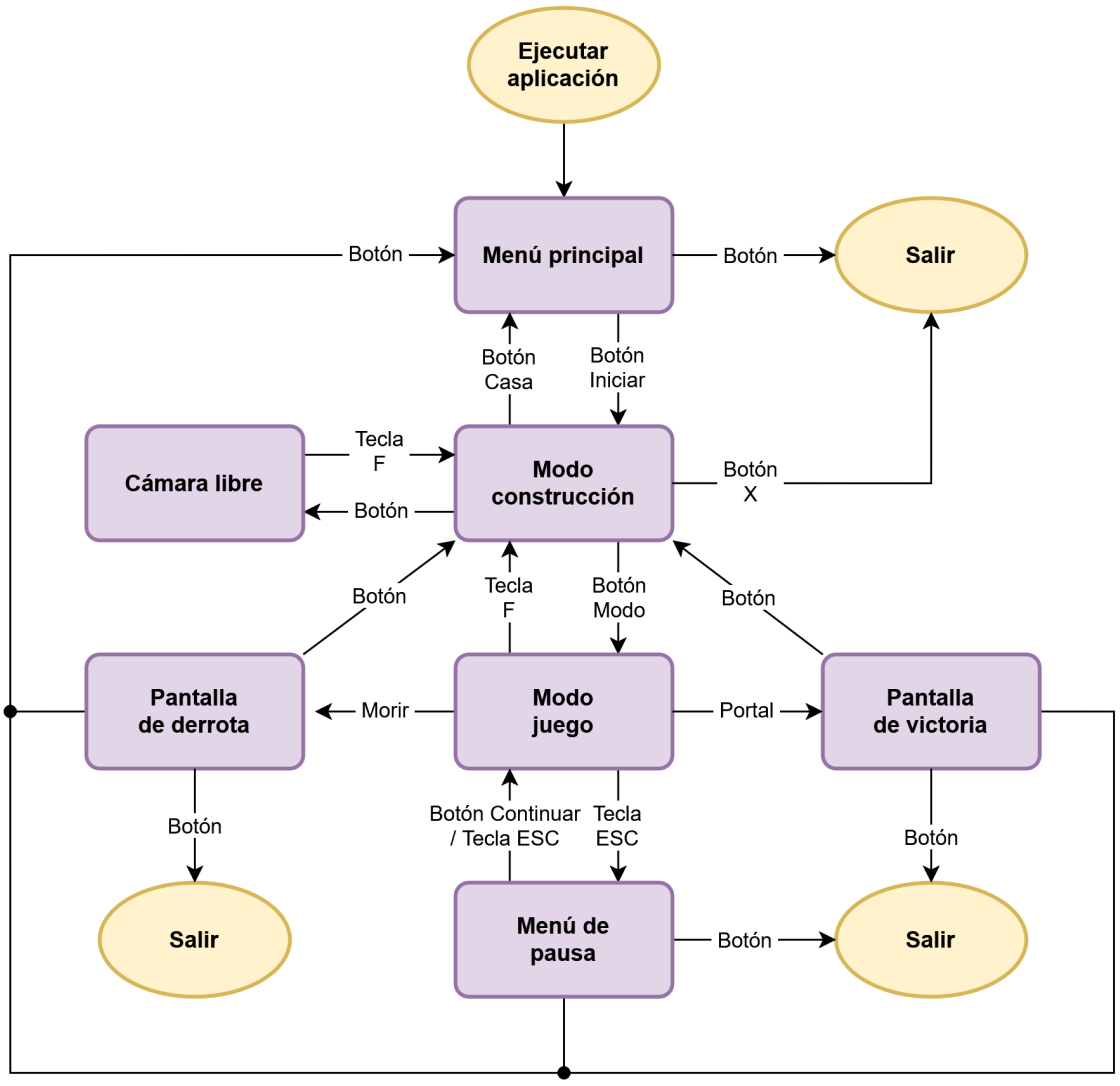

Figura 14: Diagrama de navegación entre las interfaces y modos de la aplicación

5.2. Marca

Se ha diseñado un logotipo (Figura 15) para la herramienta que se implementa en el menú principal, así como un icono (Figura 16) que se emplea en los ejecutables de la aplicación de Python y de UE. Se escogió el nombre «ChromaVision» para la herramienta porque combina la visión artificial y el método de detección mediante colores. Es por ello que también tiene el objetivo de la webcam integrado en el diseño. Además, sus iniciales (CV) son las mismas que las de «Computer Vision», (visión artificial) en inglés.

Figura 15: Logotipo de la herramienta ChromaVision

Figura 16: Icono de la herramienta ChromaVision

Para las interfaces se emplearon principalmente cuatro colores, como se pueden observar en la Figura 17 con sus códigos RGB. Los tonos oscuros son los principales, mientras que los claros se emplearon para botones deshabilitados, cuando se superpone el cursor encima o títulos informativos. El logotipo tiene el color morado oscuro manteniendo la coherencia visual en la aplicación.

Figura 17: Paleta de colores utilizada para las interfaces y el logotipo

5.3. Menú principal

La Figura 18 muestra el menú principal de la aplicación. Contiene el logotipo de «ChromaVision» y los botones para iniciar la aplicación, ajustar el volumen y cerrar la aplicación. Además, se incluyen los créditos.

Figura 18: Interfaz del menú principal

5.4. Modo construcción

La Figura 19 muestra el modo construcción con la recepción de piezas activada y un nivel de ejemplo diseñado. En ella se aprecian la barra de volumen y el menú de conversión color/actor, aunque habitualmente estos paneles permanecen ocultos hasta que el usuario pulsa el botón correspondiente.

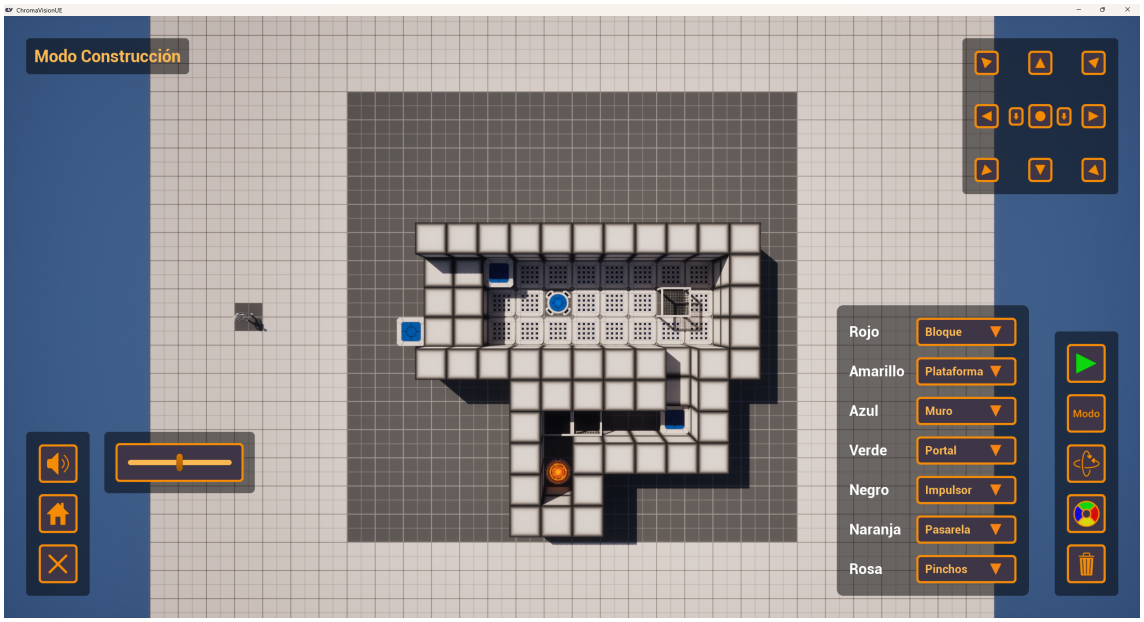

Figura 19: Modo construcción con la vista cenital rotada

La vista cenital de la Figura 19 se encuentra rotada 90 grados hacia la derecha para igualar la rotación de la cámara física, haciendo más intuitiva la visualización. En la Figura 20 puede observarse la disposición real de las piezas sobre la base.

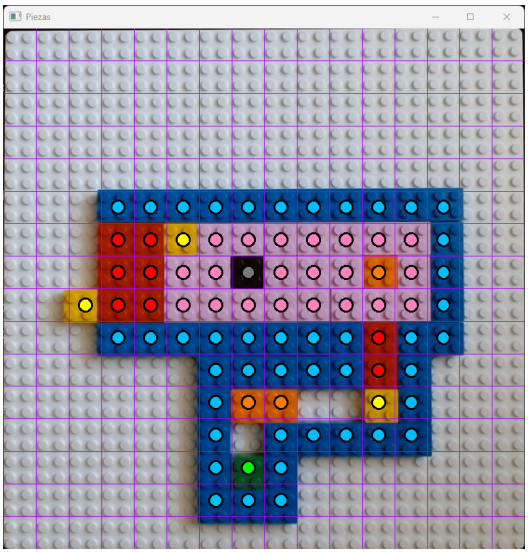

Figura 20: Disposición de las piezas LEGO sobre la base para diseñar un nivel de ejemplo

En la Figura 21 se muestra una vista fija desde la esquina del nivel. Como se puede observar los botones de rotación de la vista cenital en la cruceta aparecen deshabilitados (grises) porque esta vista no permite rotación, solo la vista cenital ofrece esta funcionalidad.

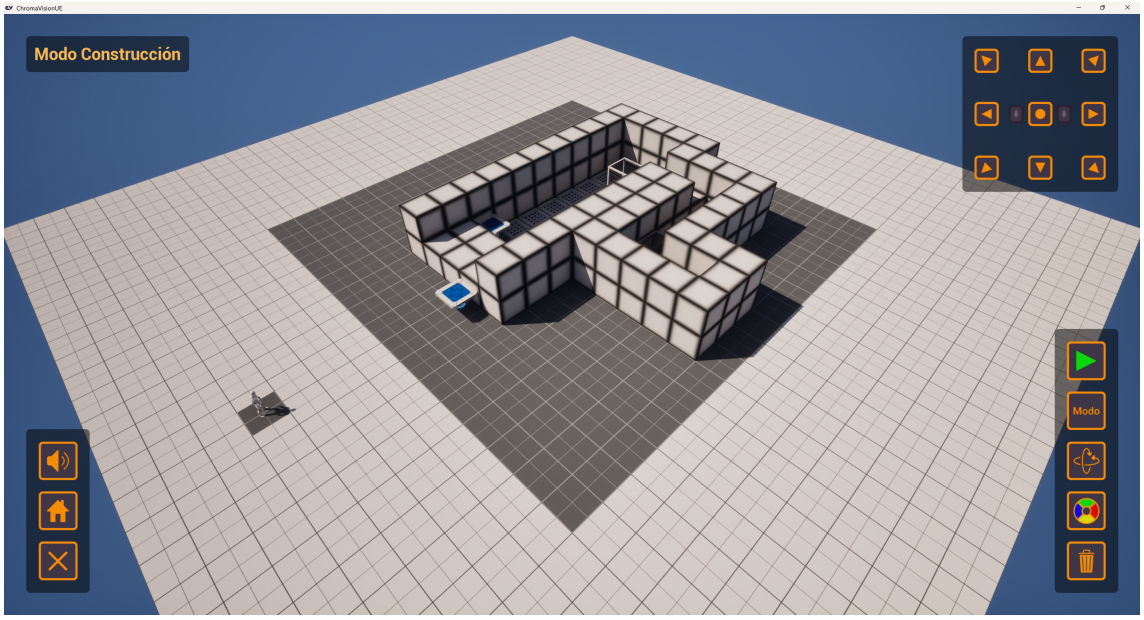

Figura 21: Vista del nivel desde una cámara fija situada en la esquina

La Figura 22 presenta la interfaz de la cámara libre con los paneles de control del modo construcción ocultos, lo que permite una visión despejada del nivel.

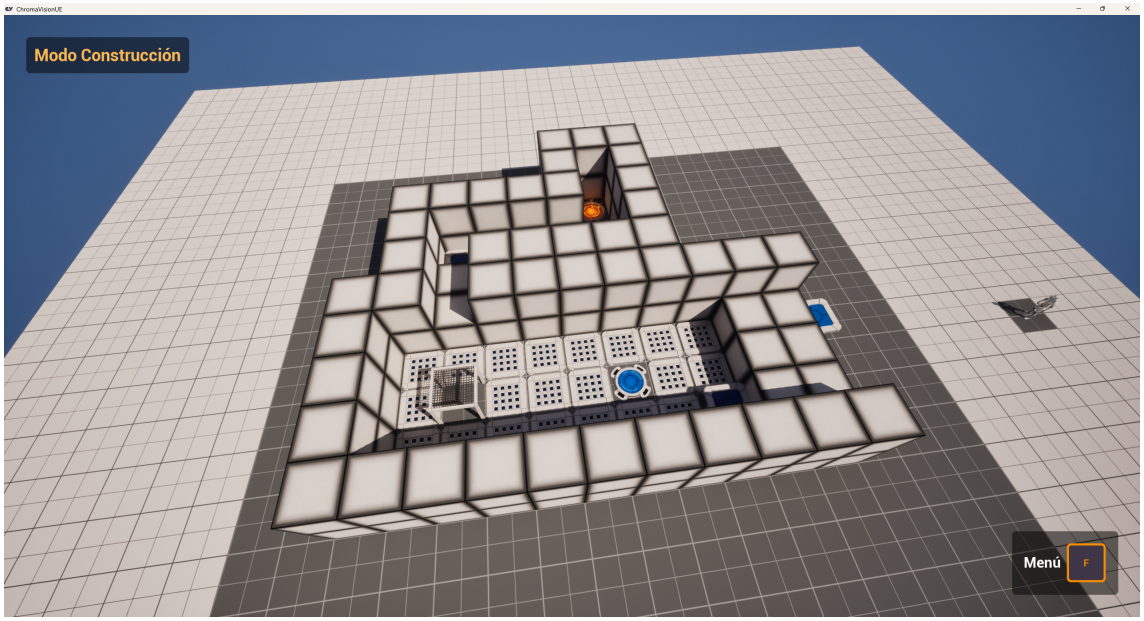

Figura 22: Cámara libre con la interfaz de construcción oculta

La Figura 23 muestra la misma cámara libre tras pulsar la tecla `F`. Entonces

aparece la interfaz completa del modo construcción y libera el cursor del ratón para interactuar con los botones de la interfaz (el movimiento de la cámara con `WASD`, `Q/E` y ratón sigue activo). El botón de activación de la cámara libre aparece deshabilitado, ya que el usuario ya se encuentra en modo espectador. Puede volver a ocultar esta interfaz pulsando la tecla `F`.

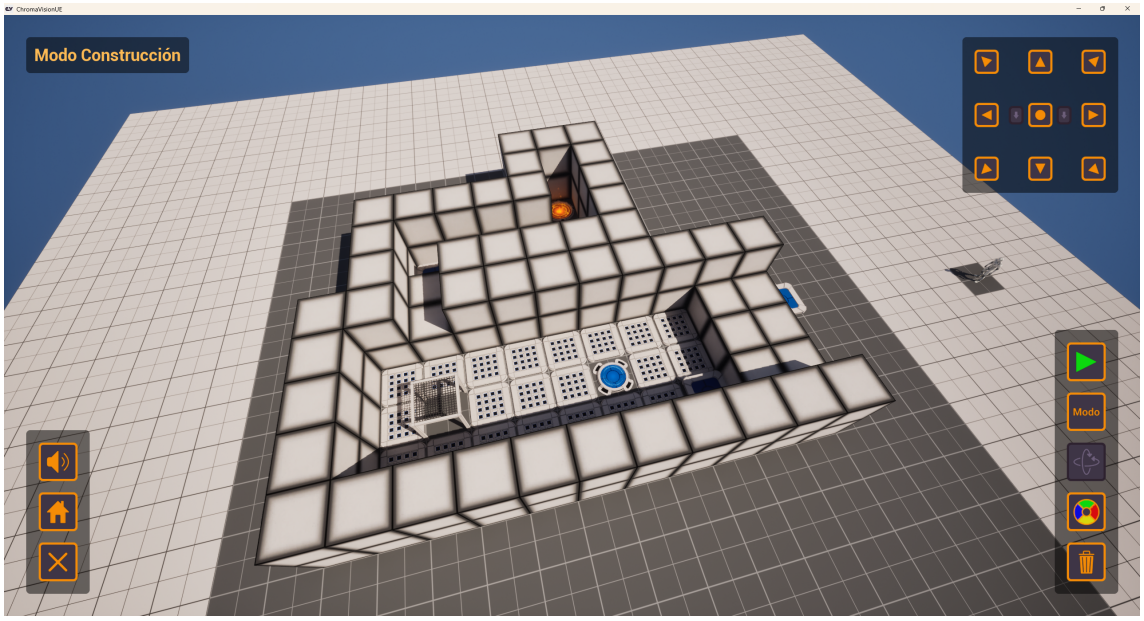

Figura 23: Cámara libre con la interfaz de construcción visible

5.5. Asignación de colores a actores

El menú desplegable de asignación de colores a actores (visible en la Figura 19) es una interfaz independiente insertada en el modo construcción. En la Figura 24 se muestra su apariencia en el BP, originalmente los desplegables de selección de actores están vacíos porque no son valores fijos. El usuario asigna los actores en el componente «PieceSpawnerComponent» (perteneciente al actor «UDPReceiver») mediante la variable «ColorToActor». Es por ello que los campos y el valor seleccionado se configuran al comienzo del nivel, antes de mostrar la interfaz. De esta forma, el menú se rellena con los valores seleccionados en el editor, como se puede observar en la Figura 25.

Figura 24: Menú selector de actores en el BP

Figura 25: Menú selector de actores relleno con las opciones disponibles

5.6. Modo juego

La Figura 26 muestra el modo juego. Se puede observar la barra de vida del jugador que está probando el nivel de diseñado. El usuario puede regresar al modo construcción pulsando la tecla `F`. Además, cuando el jugador recibe daño aparece un filtro rojo para indicarlo.

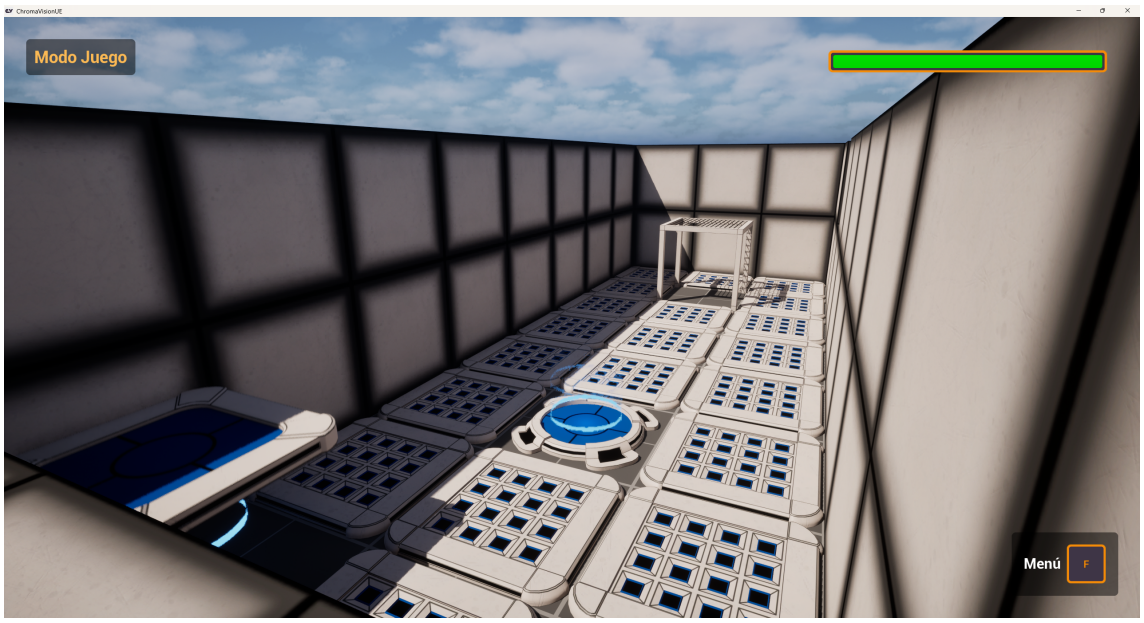

Figura 26: Interfaz del modo de juego

5.7. Menú de pausa

Se puede observar la interfaz del menú de pausa en la Figura 27. En ella se puede regresar al modo juego, configurar el volumen, ir al menú principal o cerrar la aplicación.

Figura 27: Menú de pausa

5.8. Pantalla de victoria

Cuando el usuario alcanza el portal, ha ganado y se muestra una pantalla de victoria (Figura 28). En ella puede volver al modo construcción por si desea modificar el nivel. También puede ir al menú principal o cerrar la aplicación.

Figura 28: Pantalla de victoria cuando el jugador alcanza el portal

5.9. Pantalla de derrota

Cuando el usuario ha perdido toda su vida al recibir daño, se muestra una pantalla de derrota (Figura 29). En ella puede volver al modo construcción por si desea modificar el nivel. También puede ir al menú principal o cerrar la aplicación.

Figura 29: Pantalla de derrota cuando el jugador muere

5.10. Variables accesibles desde la interfaz

En las primeras versiones de la aplicación se planteó exponer en la interfaz de usuario ciertos parámetros configurables del módulo de visualización: IP, puerto, tamaño de celda, altura de instanciación y la conversión color/actor. Sin embargo, finalmente se decidió dejar accesible únicamente la asignación de actores a colores, ya que se trata de una aplicación de demostración en la que los usuarios no necesitarán modificar estos valores. Por lo tanto, no se muestran dichos parámetros en la interfaz, aunque siguen siendo modificables en el editor del proyecto para un desarrollador. En la versión final, la interfaz solo incorpora los controles indispensables para la construcción y el juego.

6. Métricas de rendimiento

Se ha evaluado el redimiento de la aplicación midiendo el promedio, máximo y minimo de FPS registrados durante un minuto en el que se han manipulado la base y piezas de diversos tamaños y colores, simulando el uso normal de la aplicación.

Tabla 1: Resultados de evaluación de rendimiento (FPS) de la aplicación

FPS promedio	**FPS máximo**	**FPS mínimo**
115,27	120.05	98,74

La Tabla 1 recoge los resultados obtenidos. Se ha observado una media de 115,27 FPS, que demuestra que la aplicación se ejecuta de forma fluida y estable. El valor mínimo observado fue 98,74 FPS, este pico ocasional no afecta a la experiencia de usuario ni a su percepción visual. La aplicación ha llegado a alcanzar 120,05 FPS. Por lo tanto, el rendimiento es más que suficiente para el prototipado de niveles en tiempo de ejecución.

En cuanto al módulo de detección (Python), su actividad no afecta directamente a los FPS de la aplicación, ya que es un ejecutable independiente y la comunicación UDP es asíncrona. Los FPS de la aplicación Python afectan a la frecuencia de envío de datos que, si es baja, puede verse reflejada en la velocidad de actualización de los actores. Sin embargo, también se midió el rendimiento del módulo de detección y se obtuvieron resultados adecuados que no afectan visiblemente al rendimiento. Estos resultados pueden verse en el Anexo B.

Anexo E

Manual de usuario

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela **Politécnica Superior de Zamora**

Glosario

Visión artificial Campo de la inteligencia artificial que permite a las computadoras interpretar y procesar imágenes y vídeos. 2

Siglas

3D tres dimensiones. 2, 12, 14, IV

BGR Blue, Green, Red. 6

BP Blueprint. 8, 10–13, III, IV

FPS fotogramas por segundo. 3, 17

HSV Hue, Value, Saturation. 6

IP Internet Protocol. 7, 10, 12

JSON JavaScript Object Notation. 2

UDP Protocolo de Datagramas de Usuario. 2, 7, 10, 12, 13, 17, IV

UE Unreal Engine. 1–3, 8, 10–12, 14, 17, IV

Índice general

Glosario ..... **I**

Siglas ..... **II**

1 Introducción **1**

2 Presentación de la herramienta **2**

3 Requisitos del sistema **3**

4 Módulo de detección (Python) **4**

4.1 Ejecución 4

4.2 Ventana «Base» 4

4.3 Ventana «Piezas» 5

4.4 Configuración mediante «config.json» 6

5 Módulo de visualización (Unreal Engine) **8**

5.1 Integración en un proyecto 8

5.2 Configuración en el editor 10

5.3 Uso en el nivel 11

5.4 Funciones disponibles en BPs 12

6 Aplicación de demostración **14**

6.1 Modo construcción 14

6.2 Modo juego 15

6.3 Menús y pantallas 16

7 Consejos y solución de problemas **17**

Índice de figuras

1 Ventana «Base»: el contorno naranja delimita la base y los puntos azules marcan las esquinas 4
2 Ventana «Piezas»: los círculos señalan las piezas detectadas y su color. La cuadrícula ayuda a colocarlas, aunque puede ocultarse 5
3 Parámetros configurables en el archivo «config.json» con los valores por defecto 6
4 Mensajes de aviso tras recargar la configuración 7
5 Carpeta «C++ Classes» en el Content Browser con las clases «UDPReceiver y PieceSpawnerComponent» 8
6 Línea «CHROMAVISIONUE_API» en «PieceSpawnerComponent.h», que debe reemplazarse en ambas clases 9
7 Dependencias añadidas en el archivo «[NombreProyecto].Build.cs» 9
8 Parámetros del actor «UDPReceiver» 10
9 Jerarquía del BP «BP_UDPReceiver» con el componente «PieceSpawnerComponent» añadido 10
10 Parámetros del componente «PieceSpawnerComponent» 11
11 Ventana «Piezas» del módulo de detección 12
12 Actores 3D instanciados en UE a partir de las piezas detectadas 12
13 Funciones del receptor UDP disponibles en el BP 13
14 Funciones del instanciador de actores disponibles en el BP 13
15 Modo construcción de la aplicación de prototipado de niveles 15
16 Modo juego de la aplicación de prototipado de niveles 16

1. Introducción

Este anexo contiene las instrucciones necesarias para utilizar la herramienta «ChromaVision» y la aplicación de ejemplo de prototipado de niveles.

El manual explica cómo ejecutar el módulo de detección, interpretar sus ventanas, configurar los parámetros mediante el archivo «config.json». Además, explica cómo integrar el módulo de visualización en un proyecto de Unreal Engine (UE) y utilizar la aplicación de demostración. Al final se incluye una sección con consejos para resolver los problemas más comunes.

Los detalles técnicos de implementación se encuentran en los Anexos B (Módulo de detección), C (Módulo de visualización) y D (Aplicación de prototipado de niveles).

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Presentación de la herramienta

«ChromaVision» es una herramienta modular que permite detectar piezas por color y posición sobre una base mediante visión artificial, y visualizarlas en un motor de videojuegos de forma sincronizada. El color de cada pieza actúa como un identificador que permite sustituirla por el tipo de objeto que el usuario desee en el motor. Se compone de dos partes independientes que se comunican entre sí mediante el protocolo UDP:

- **Módulo de detección (Python y OpenCV)**: es la parte encargada de identificar las piezas y enviar su información al motor de videojuegos. Captura las imágenes de la webcam, identifica la base, y el color y posición de cada pieza situada dentro de ella. Envía los datos de las piezas (color y posición) por UDP en formato JavaScript Object Notation (JSON). Se trata de un archivo ejecutable (.exe) que permite configurar ciertos parámetros y editar los rangos de color.

- **Módulo de visualización (UE, C++)**: es la parte encargada de recibir la información del módulo de detección, interpretarla e instanciar los actores asignados a cada color por el usuario, en las posiciones correspondientes a las piezas detectadas. Debido a la conexión estándar por UDP, es compatible con cualquier motor de videojuegos que pueda integrarla. Sin embargo, para este proyecto se ha programado el módulo correspondiente a UE, el cual es reutilizable en cualquier proyecto de este motor.

La aplicación de ejemplo de prototipado de niveles es una demostración de lo que se puede hacer con la herramienta, pero no es necesaria para el funcionamiento de los módulos, pueden utilizarse para otro tipo de aplicaciones y sectores.

Ambos módulos pueden ejecutarse de forma independiente y en cualquier orden, siempre que estén funcionando simultáneamente para que la comunicación UDP sea efectiva. La base y las piezas se detectan automáticamente, y sus datos se envían sin intervención del usuario. Si la recepción está activada en el módulo de visualización, la escena 3D se actualiza al manipular las piezas sobre la base.

3. Requisitos del sistema

A continuación se enumeran los requisitos necesarios para utilizar la herramienta. Se indican tanto los componentes hardware como el software necesario, así como algunas consideraciones sobre la configuración.

Hardware

- **Ordenador** con sistema operativo Windows (el módulo de detección y la aplicación de demostración se han empaquetado y probado en este sistema).
- **Webcam** estándar (resolución recomendada: 1920×1080, 30 FPS).
- **Base**: se recomienda una base LEGO blanca de 32×32 studs. El sistema detecta cualquier rectángulo, por lo que se puede usar otra base, siempre que contraste con la mesa. Las divisiones de la cuadrícula se pueden ajustar en la configuración.
- **Piezas**: se recomiendan piezas LEGO de al menos 2×2 studs (dimensiones pares) en los colores rojo, amarillo, azul, verde, negro, naranja y rosa. Se pueden añadir, modificar o eliminar en la configuración.

El sistema es flexible, detecta un rectángulo como base y clasifica las piezas por color. Por tanto, no es necesario utilizar LEGO, pero se recomienda por su sistema de encaje que facilita la colocación. La base puede ser de otro material (incluso papel) y las piezas de cualquier tipo, siempre que tengan colores sólidos y forma regular. En ese caso, solo hay que ajustar los rangos de color y las divisiones de la cuadrícula en la configuración (Sección 4.4).

Software

- **Módulo de detección**: archivo ejecutable «ChromaVision.exe». No requiere instalación de Python ni de dependencias adicionales.
- **Módulo de visualización**: archivos C++ para integrar en un proyecto de UE 5.7.
- **Aplicación de prototipado de niveles (opcional)**: carpeta con el ejecutable «ChromaVisionUE.exe» y los archivos necesarios para su funcionamiento. Muestra la integración del módulo de visualización en UE y su comunicación con el módulo de detección. No es necesaria para el funcionamiento independiente de los módulos.

4. Módulo de detección (Python)

4.1. Ejecución

El módulo de detección consiste en un ejecutable llamado «ChromaVision.exe». Al ejecutarlo, se abrirán dos ventanas: «Base» y «Piezas».

Además, la primera vez que se ejecuta, el programa genera automáticamente un archivo de configuración llamado «config.json» en la misma carpeta. Este archivo contiene todos los parámetros ajustables.

4.2. Ventana «Base»

La ventana «Base» (Figura 1) muestra la imagen capturada por la webcam. Sobre ella se dibuja:

- Un **contorno naranja** que delimita la base detectada, y líneas moradas que se extienden sobre sus bordes.
- **Puntos azules** en las cuatro esquinas de la base.
- Un texto verde con el mensaje **«Base detectada»** si la base se ha localizado correctamente, o un texto rojo con **«Base NO detectada»** en caso contrario.

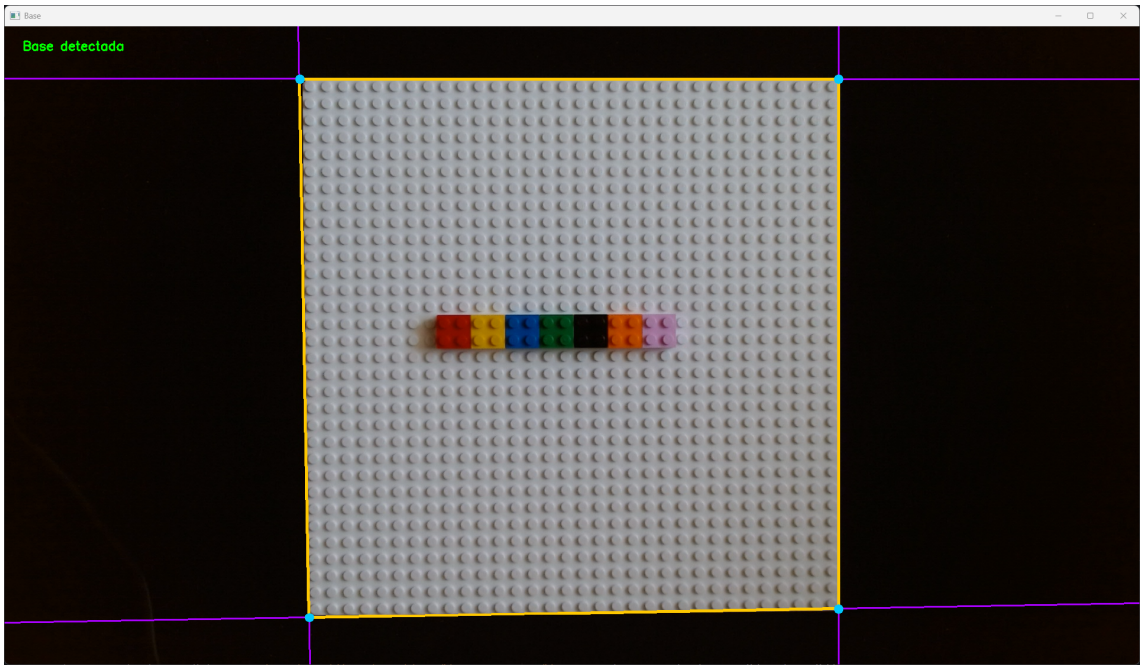

Figura 1: Ventana «Base»: el contorno naranja delimita la base y los puntos azules marcan las esquinas

Para que el sistema funcione correctamente, la base debe estar colocada dentro del campo de visión de la webcam (si alguna esquina queda fuera, se descartará). Además, es recomendable que la base se coloque sobre una superficie de color diferente para que contraste adecuadamente.

4.3. Ventana «Piezas»

La ventana «Piezas» (Figura 2) muestra la imagen de la base con la perspectiva corregida (vista cenital) para compensar posibles inclinaciones de la webcam y la base. Sobre ella se superpone:

- Una **cuadrícula morada** que divide la base en celdas (16×16 por defecto).
- **Círculos de colores** en el centro de cada celda ocupada, indicando el color detectado.

Figura 2: Ventana «Piezas»: los círculos señalan las piezas detectadas y su color. La cuadrícula ayuda a colocarlas, aunque puede ocultarse

Esta ventana permite comprobar de un vistazo si las piezas se están detectando correctamente. Si la base no ha sido detectada, la ventana mostrará un mensaje de aviso en lugar de la cuadrícula y los círculos. La cuadrícula se puede ocultar en el archivo de configuración.

4.4. Configuración mediante «config.json»

El archivo «config.json» se puede editar con cualquier editor de texto (Figura 3) y contiene todos los parámetros ajustables del sistema:

- **COLORS**: lista de colores que el sistema puede detectar. Cada color se define por un nombre, los límites inferior y superior en el espacio HSV (H: 0-179, S: 0-255, V: 0-255) y el valor Blue, Green, Red (BGR) que se muestra en la interfaz.
- **WINDOW_SCALE**: factor de escala para redimensionar las ventanas de visualización. Un valor de 1.0 muestra las imágenes a su tamaño original, mientras que valores inferiores las reducen.
- **WARP_OUTPUT_SIZE**: tamaño de la imagen después de corregir la perspectiva (píxeles del lado más largo).
- **GRID_WIDTH y GRID_HEIGHT**: número de divisiones de la cuadrícula para la detección de piezas (por defecto 16x16). Si se cambia el tamaño de la base, estos valores se deben ajustar.

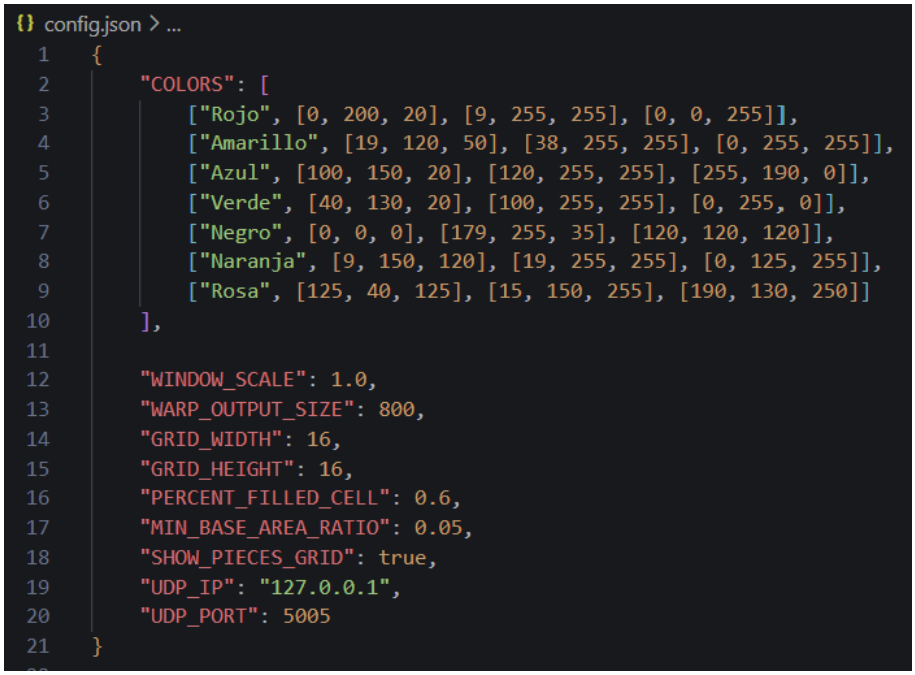

Figura 3: Parámetros configurables en el archivo «config.json» con los valores por defecto

- **PERCENT_FILLED_CELL**: porcentaje mínimo de ocupación de una celda por el color para considerar que hay una pieza (por defecto 60 %). Este um-

bral ayuda a evitar detecciones falsas por contaminación de color o desajustes de perspectiva.

- **MIN_BASE_AREA_RATIO**: porcentaje mínimo que debe ocupar la base en la imagen, filtra objetos pequeños.
- **SHOW_PIECES_GRID**: si es `true`, se muestra la cuadrícula en la ventana «Piezas»; si es `false`, se oculta.
- **UDP_IP y UDP_PORT**: dirección IP y puerto para enviar los datos al motor de videojuegos. Deben coincidir con los configurados en el módulo de visualización.

Tras modificar el archivo, se puede recargar la configuración sin reiniciar el programa pulsando la tecla `R`. Las ventanas mostrarán un mensaje de confirmación si la recarga ha sido correcta (Figura 4), o un mensaje de error si el archivo no es válido (contiene errores de sintaxis o valores incorrectos). En ese caso, el programa regenerará el archivo con los valores por defecto.

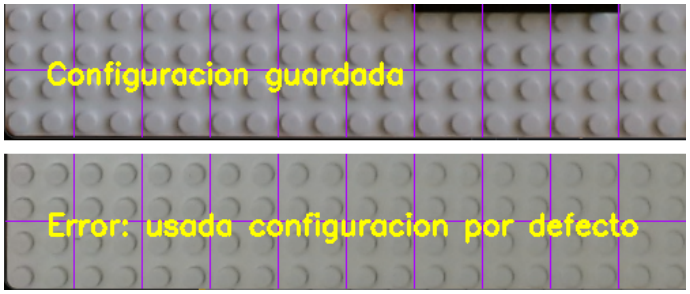

Figura 4: Mensajes de aviso tras recargar la configuración

Nota: Si se modifican la dirección IP o el puerto UDP en el archivo «config.json», estos cambios no se aplicarán hasta que se reinicie el módulo de detección (el ejecutable «ChromaVision.exe»). La recarga con la tecla **R** actualiza el resto de parámetros (colores, cuadrícula, umbrales, etc.) sin necesidad de reiniciar.

5. Módulo de visualización (Unreal Engine)

Esta sección explica cómo integrar el módulo de visualización en un proyecto de UE. El módulo consta de dos scripts en C++ que se añaden al proyecto y se configuran desde el editor.

5.1. Integración en un proyecto

Para añadir el módulo a un proyecto de UE (nuevo o existente), se deben seguir los siguientes pasos:

1. Crear un proyecto en UE 5.7 (puede ser de BPs o C++) o abrir uno ya existente.

2. Crear dos clases de C++ con los nombres y tipos exactos:

   - «UDPReceiver» de tipo Actor.
   - «PieceSpawnerComponent» de tipo ActorComponent.

   Se guardan en la carpeta «Source/[NombreProyecto]/» por defecto.

3. Cerrar el editor de UE.

4. Abrir el proyecto en Visual Studio 2022 y compilar para que UE reconozca las nuevas clases.

5. Volver a abrir UE. Debería aparecer la carpeta «C++ Classes» con ambas clases (Figura 5).

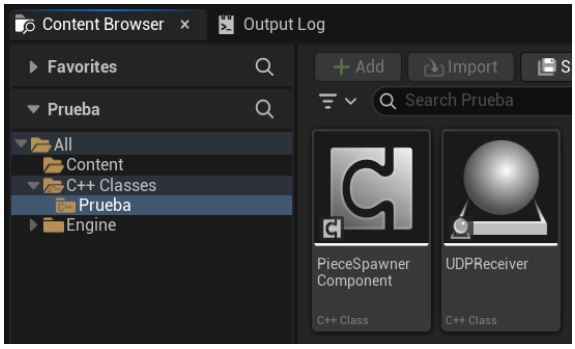

Figura 5: Carpeta «C++ Classes» en el Content Browser con las clases «UDPReceiver y PieceSpawnerComponent»

6. La forma más rápida es copiar los archivos «.h» y «.cpp» de los scripts en la carpeta «Source/[NombreProyecto]/», sobrescribiendo los archivos generados.

También se puede hacer manualmente, abriendo cada archivo y pegando el contenido.

7. En los archivos «.h» de ambas clases, se debe reemplazar «CHROMAVISIONUE_API» (Figura 6) por el nombre del proyecto en mayúsculas seguido de «_API» (por ejemplo, «NOMBREPROYECTO_API»)

```
// PieceSpawnerComponent.h: gestiona el spawneo de actores a partir de piezas recibidas
#pragma once

#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "UDPReceiver.h"
#include "PieceSpawnerComponent.generated.h"


UCLASS( ClassGroup=(Custom), meta=(BlueprintSpawnableComponent) )
class CHROMAVISIONUE_API UPieceSpawnerComponent : public UActorComponent {
    GENERATED_BODY()

public:
    UPieceSpawnerComponent();

public:
    //----------------------- VARIABLES -----------------------
    /* Mapeo de color (string) de la pieza a clase actor que se spawnea (configurable en editor) */
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Piece Spawner")
    TMap<FString, TSubclassOf<AActor>> ColorToActor;
```

Figura 6: Línea «CHROMAVISIONUE_API» en «PieceSpawnerComponent.h», que debe reemplazarse en ambas clases

8. En el archivo «[NombreProyecto].Build.cs» (ubicado en «Source/[NombreProyecto]/»), se deben añadir las dependencias «"Sockets", "Networking", "Json"» en la sección «PublicDependencyModuleNames» (Figura 7).

```
PublicDependencyModuleNames.AddRange(new string[] {
    "Core", "CoreUObject", "Engine", "InputCore", "EnhancedInput",
    "Sockets", "Networking", "Json"
});
```

Figura 7: Dependencias añadidas en el archivo «[NombreProyecto].Build.cs»

9. Compilar el proyecto de nuevo desde Visual Studio.

Una vez completados estos pasos, los scripts estarán listos para ser utilizados en el editor.

5.2. Configuración en el editor

Una vez compilados los scripts, se pueden configurar desde el editor de UE:

1. Crear un Blueprint (BP) a partir de la clase UDPReceiver (por ejemplo, BP_UDPReceiver).

2. Configurar el actor «UDPReceiver», los valores por defecto ya están ajustados para conectarse al módulo de detección:

   - **AutoStart**: si es «true» el socket de recepción UDP se inicia automaticamente.
   - **IP**: dirección IP (127.0.0.1).
   - **Port**: puerto de escucha (por defecto 5005).
   - **BufferSize**: tamaño del buffer de detección (por defecto 2 MB).

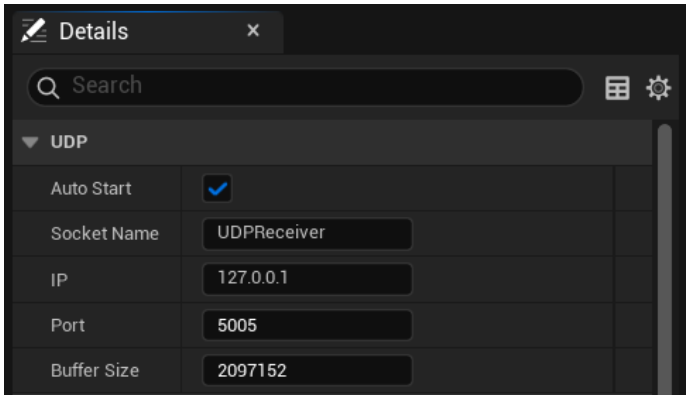

Figura 8: Parámetros del actor «UDPReceiver»

3. En el BP, añadir un componente «PieceSpawnerComponent» (Figura 9).

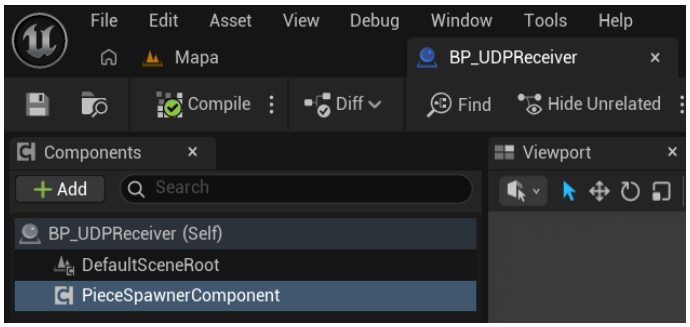

Figura 9: Jerarquía del BP «BP_UDPReceiver» con el componente «PieceSpawnerComponent» añadido

4. Configurar el componente «PieceSpawnerComponent»:

   - **ColorToActor**: asigna cada color a una clase de actor (BP). Se pueden añadir, modificar o eliminar entradas directamente en el editor. En la Figura 10 se muestra un ejemplo con tres colores (azul, rojo, verde) asignados a actores básicos (cubo, cubo, cilindro).
   - **CellSize**: distancia entre el centro de dos celdas (por defecto 100).
   - **SpawnHeight**: altura a la que se instancian los actores (por defecto 0, el suelo).

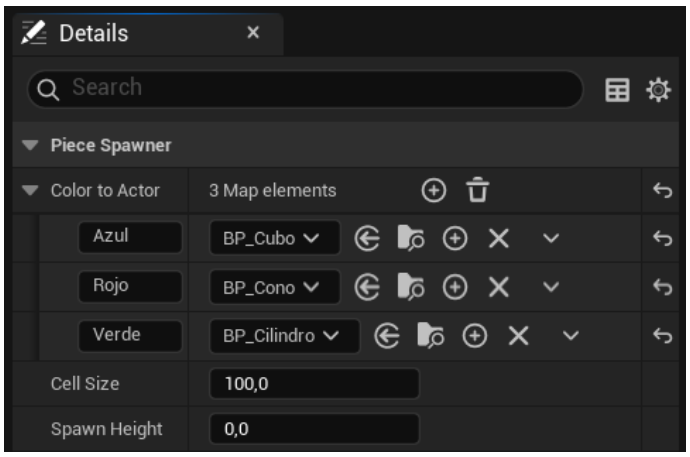

Figura 10: Parámetros del componente «PieceSpawnerComponent»

5.3. Uso en el nivel

Una vez configurado el componente, se debe colocar el BP en el nivel:

1. Colocar una instancia del BP «BP_UDPReceiver» en el nivel. Por defecto, la conexión se inicia automáticamente al comenzar la partida.

2. Ejecutar el módulo de detección («ChromaVision.exe») con la webcam conectada y las piezas colocadas sobre la base.

3. Iniciar la partida en UE. Los actores (Figura 12) deben aparecer en las posiciones correspondientes a las piezas detectadas (Figura 11).

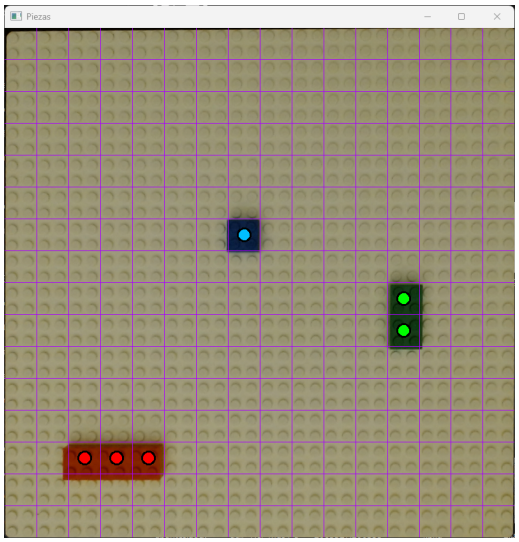

Figura 11: Ventana «Piezas» del módulo de detección

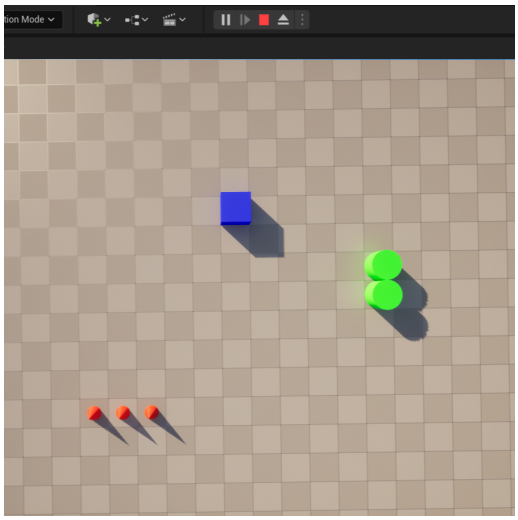

Figura 12: Actores 3D instanciados en UE a partir de las piezas detectadas

Si los actores no se ven correctamente (por ejemplo, dentro del suelo), ajustar el parámetro «SpawnHeight» en el componente. Además, asegurarse de que la dirección IP y el puerto configurados en el «UDPReceiver» coinciden con los definidos en el «config.json» del módulo de detección.

5.4. Funciones disponibles en BPs

El módulo de visualización expone varias funciones y eventos que se pueden utilizar desde los BPs para controlar su comportamiento. Se dividen según la clase a la que pertenecen.

UDPReceiver (Figura 13)

- **StartUDPReceiver**: inicia la recepción UDP manualmente.
- **StopUDPReceiver**: detiene la recepción UDP.
- **IsReceiving**: devuelve si el receptor está activo (booleano).
- **OnPiecesReceived**: evento que se dispara al recibir datos, permite añadir lógica personalizada.

PieceSpawnerComponent (Figura 14)

- **UpdatePieces**: actualiza la escena con el array de piezas recibido (normalmente se llama automáticamente desde el receptor).
- **ClearAllPieces**: elimina todos los actores generados en la escena.

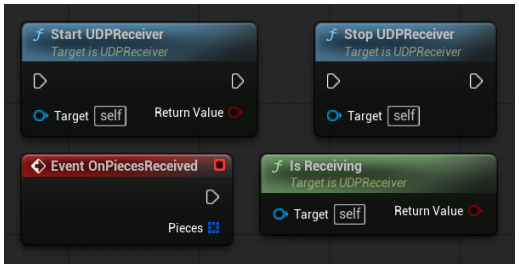

Figura 13: Funciones del receptor UDP disponibles en el BP

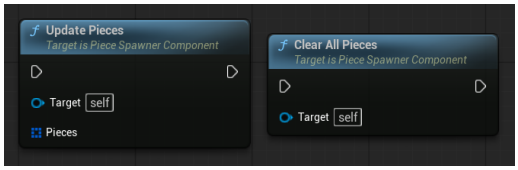

Figura 14: Funciones del instanciador de actores disponibles en el BP

6. Aplicación de demostración

La aplicación de ejemplo es una demostración del uso de la herramienta en un caso práctico, el prototipado de niveles. Se trata de un proyecto empaquetado de UE que ya incluye el módulo de visualización y una serie de actores predefinidos. Para iniciarla se debe ejecutar el archivo «ChromaVisionUE.exe». La aplicación se divide en dos modos principales: construcción y juego.

6.1. Modo construcción

En el modo construcción, el usuario coloca piezas sobre la base y la aplicación muestra los actores 3D correspondientes de forma sincronizada. Desde la interfaz se pueden realizar las siguientes acciones:

- Cambiar la vista de la escena mediante una cruceta, pudiendo rotar la vista cenital para alinearla con el ángulo de la cámara.
- Cambiar a una cámara libre para observar el nivel desde cualquier ángulo.
- Pausar o reanudar la conexión (la recepción de datos de las piezas).
- Borrar manualmente los actores generados.
- Reasignar qué clase de objeto se asocia a cada color de pieza. Incluye la opción de asignar «Nada» para desactivar la detección del color temporalmente, o el personaje para escoger el punto de inicio del nivel.
- Ajuste de volumen de audio, retorno al menú principal y salida de la aplicación.

En la Figura 15 se puede observar el modo construcción con los actores asignados a cada color, dispuestos según la posición de las piezas de la Figura 1. Para esta aplicación de ejemplo se han identificado siete colores, asociados a los siguientes elementos:

- **Rojo**: bloques que el jugador no puede saltar.
- **Amarillo**: plataformas que se mueven verticalmente, permitiendo al jugador alcanzar zonas más altas.
- **Azul**: muros de doble altura que el jugador no puede saltar, ni siquiera usando otros modelos 3D.
- **Verde**: portal que, al tocarlo, completa el nivel (objetivo del juego).

- **Negro**: impulsores que propulsan al jugador hacia arriba cuando se sitúa sobre ellos.
- **Naranja**: pasarelas que permiten al jugador caminar por debajo y por encima de ellas.
- **Rosa**: trampas de pinchos que suben y dañan al jugador si pasa por encima de ellas.

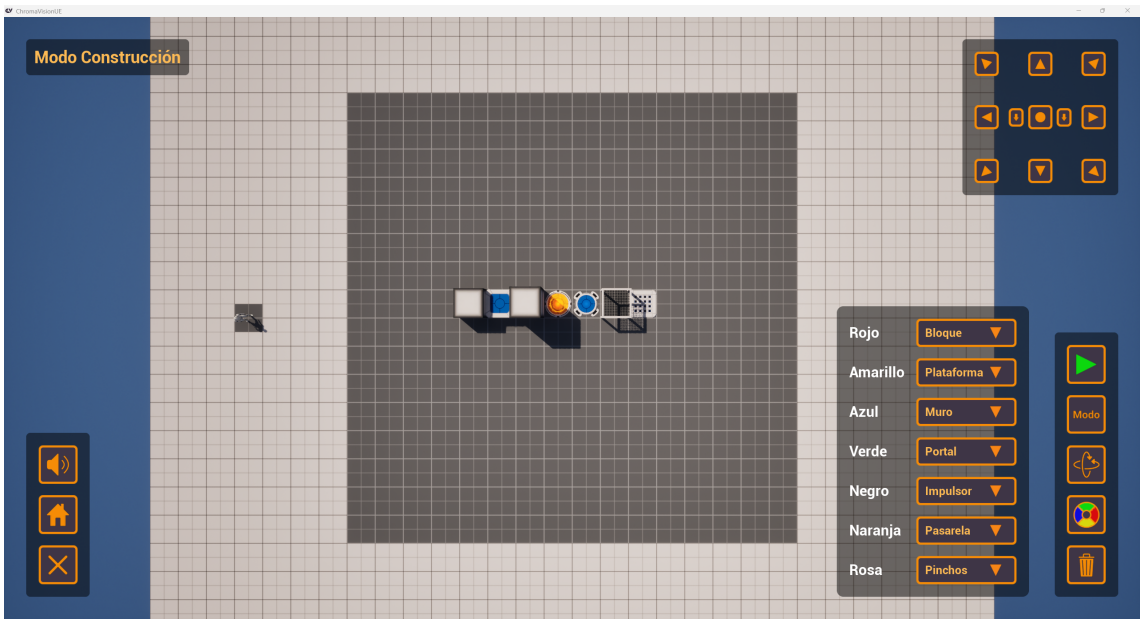

Figura 15: Modo construcción de la aplicación de prototipado de niveles

6.2. Modo juego

En el modo juego, el usuario controla un personaje en primera persona que puede recorrer el nivel, interactuar con los objetos y alcanzar el portal para ganar. La tecla **F** permite volver al modo construcción.

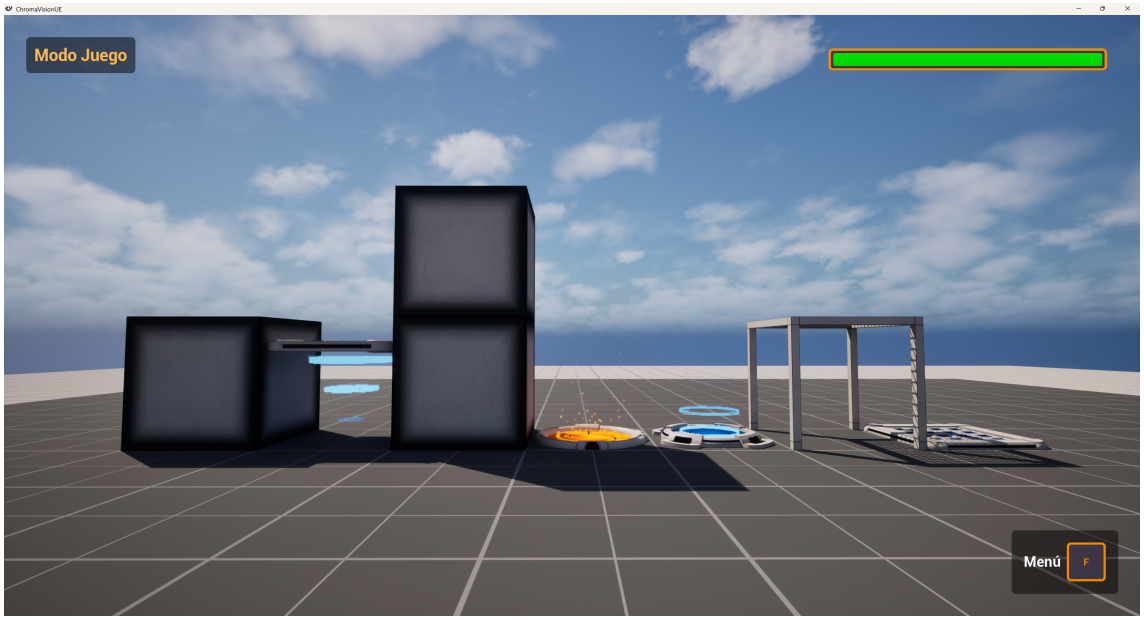

Figura 16: Modo juego de la aplicación de prototipado de niveles

En la Figura 16 se muestra el modo juego con los actores ubicados según las piezas detectadas en la Figura 1. Además, la barra de vida del jugador aparece en la esquina superior derecha.

6.3. Menús y pantallas

La aplicación incluye un menú principal, un menú de pausa (tecla `Esc`), y pantallas de victoria y derrota.

7. Consejos y solución de problemas

A continuación se indican los posibles problemas y sus soluciones.

- **La webcam no se abre:** asegúrate de que la cámara esté conectada y no esté siendo usada por otra aplicación.
- **La base no se detecta:** coloca la base sobre una superficie de color contrastado y asegúrate de que esté completamente visible en la imagen de la webcam.
- **Los colores no se detectan correctamente:** ajusta los rangos de color en el archivo «config.json».
- **La conexión UDP no funciona:** verifica que la IP y el puerto coinciden en «config.json» y en el `UDPReceiver` de UE. Asegúrate de que el módulo de detección y UE se estén ejecutando simultáneamente.
- **Los actores no aparecen en UE:** comprueba que el «PieceSpawnerComponent» tiene asignada una clase de actor para cada color en el mapa «ColorToActor».
- **Consejo**: se recomienda no minimizar las ventanas «Base» y «Piezas» del módulo de detección durante la ejecución. Al minimizarlas, el sistema operativo reduce la prioridad del proceso, lo que puede disminuir su rendimiento (FPS) y, en consecuencia, la fluidez de la detección y del envío de piezas al motor de videojuegos. Para trabajar con otras aplicaciones es preferible situar las ventanas de «ChromaVision» en un segundo monitor o simplemente colocar la ventana activa por encima, sin minimizarlas.

Anexo F

Manual de programador

Grado en Desarrollo de Aplicaciones 3D Interactivas y Videojuegos

Dpto. de Informática y Automática

Trabajo de Fin de Grado

Octubre 2025 - Junio 2026

Autora

Lucía García Bobillo

Tutor

Juan Alberto García Esteban

Dpto. de Informática y Automática

Cotutor

Roberto José García Martín

Dpto. de Ingeniería Mecánica

Universidad de Salamanca

Escuela Politécnica Superior de Zamora

VNiVERSiDAD D SALAMANCA | Escuela **Politécnica Superior de Zamora**

Glosario

Canny Algoritmo de detección de bordes ampliamente utilizado en visión artificial. 10

K-Means Algoritmo de agrupamiento (clustering) que divide datos en k grupos según su similitud. 2, 9, 10, 12, 13

Método de Otsu Algoritmo para calcular automáticamente un umbral de binarización en una imagen. 10

Studs Los pequeños salientes cilíndricos de las piezas LEGO que permiten encajarlas. . 19

Transformada de Hough Técnica de procesamiento de imágenes para detectar líneas rectas a partir de bordes, representándolas en coordenadas polares. 2, 9, 10, 13

Visión artificial Campo de la inteligencia artificial que permite a las computadoras interpretar y procesar imágenes y vídeos. I

Siglas

3D tres dimensiones. 1, 22, 27

BGR Blue, Green, Red. 6, 8, 10, 14, 16–18

BP Blueprint. 22, 23, 25, 27, 30, 31

FPS fotogramas por segundo. 3

HSV Hue, Value, Saturation. 2, 6, 8, 16–19

IP Internet Protocol. 8, 9, 19–24, 31

JSON JavaScript Object Notation. 1, 2, 5–9, 19–23, 25–27, 31

ROI región de interés. 17

UDP Protocolo de Datagramas de Usuario. 1–3, 7, 9, 18–28, 31

UE Unreal Engine. 1, 22, 23, 26–28, 30, 31, IV

Índice general

Glosario . . . . . **I**

Siglas . . . . . **II**

1 Introducción **1**

2 Módulo de detección (Python) **2**

2.1 Estructura general del módulo 2

2.2 main.py 3

2.2.1 Descripción general 3

2.2.2 Funciones principales 3

2.2.3 Variables globales utilizadas 4

2.2.4 Dependencias 4

2.2.5 Notas de implementación 4

2.3 config.py 5

2.3.1 Descripción general 5

2.3.2 Funciones principales 5

2.3.3 Variables globales 7

2.3.4 Dependencias 8

2.3.5 Notas de implementación 9

2.4 base_detection.py 9

2.4.1 Descripción general 9

2.4.2 Funciones principales 10

2.4.3 Funciones auxiliares 11

2.4.4 Variables globales utilizadas 13

2.4.5 Dependencias 13

2.4.6 Notas de implementación 13

2.5 base_warp.py 14

2.5.1 Descripción general 14

2.5.2 Función principal 14

2.5.3 Variables globales utilizadas 15

2.5.4 Dependencias 15

2.5.5 Notas de implementación 15

2.6 color_detection.py 16

2.6.1 Descripción general 16

2.6.2 Funciones principales 16

2.6.3 Variables globales utilizadas 18

2.6.4 Dependencias 19

2.6.5 Notas de implementación 19

2.7 udp_sender.py 19
2.7.1 Descripción general 19
2.7.2 Clase `UDP_socket` 20
2.7.3 Variables globales utilizadas 21
2.7.4 Dependencias 21
2.7.5 Notas de implementación 21

3 Módulo de visualización (Unreal Engine) 22
3.1 Estructura general del módulo 22
3.2 UDPReceiver 22
3.2.1 Descripción general 22
3.2.2 Variables principales 23
3.2.3 Estructura de datos 23
3.2.4 Funciones principales 24
3.2.5 Dependencias 26
3.2.6 Notas de implementación 27
3.3 PieceSpawnerComponent 27
3.3.1 Descripción general 27
3.3.2 Variables principales 28
3.3.3 Funciones principales 28
3.3.4 Dependencias 30
3.3.5 Notas de implementación 30
3.4 Integración en proyectos de UE 30
3.4.1 Creación de las clases en el editor 30
3.4.2 Sustitución por los archivos del módulo 31
3.4.3 Configuración en el editor 31
3.4.4 Portabilidad a otros motores 31

1. Introducción

Este anexo contiene la documentación técnica del código fuente de la herramienta ChromaVision. A diferencia del Manual de usuario (Anexo E), que se centra en el uso de la herramienta, este manual describe la estructura interna del código, el flujo de procesamiento y los detalles de implementación de cada módulo.
El código se divide en dos grandes bloques:

- **Módulo de detección (Python y OpenCV)**: se encarga de capturar las imágenes de la webcam, detectar la base, identificar las piezas por color y posición, y enviar los datos mediante UDP. Se describen los archivos que lo componen, sus funciones principales y otros detalles.
- **Módulo de visualización (Unreal Engine / C++)**: recibe los datos UDP, interpreta el JSON e instancia actores 3D en la escena. Se documentan las clases `UDPReceiver` y `PieceSpawnerComponent`, sus métodos y su integración en proyectos de UE.

Los diagramas de flujo y las explicaciones detalladas del proceso se encuentran en los Anexos B (Módulo de detección) y C (Módulo de visualización).

El código desarrollado para este TFG está disponible en este repositorio de GitHub.

2. Módulo de detección (Python)

2.1. Estructura general del módulo

El módulo de detección es el encargado de procesar las imágenes capturadas por la webcam para identificar la base, detectar las piezas por su color y calcular su posición. Está implementado en Python y utiliza la librería OpenCV para el procesamiento de imágenes, así como NumPy para operaciones matriciales.

El flujo de procesamiento se organiza en varias etapas secuenciales:

1. **Captura de la imagen**: el módulo lee los fotogramas de la webcam a través de OpenCV.

2. **Detección de la base**: se detectan las líneas rectas que delimitan la base mediante la transformada de Hough. Estas líneas se agrupan por su orientación con K-Means y se selecciona el rectángulo de mayor área que cumple condiciones de perpendicularidad y visibilidad.

3. **Corrección de perspectiva**: una vez obtenidas las esquinas de la base, se aplica una transformación de perspectiva para obtener una vista cenital, eliminando distorsiones por inclinación de la cámara o de la base.

4. **Detección de piezas**: sobre la imagen corregida, se generan máscaras de color en el espacio HSV para cada color configurado. La imagen se divide en una cuadrícula ajustable, y cada celda se evalúa según un umbral de ocupación para determinar la presencia y el color de cada pieza.

5. **Envío de datos**: la lista de piezas detectadas (color y coordenadas de celda) se serializa en formato JSON y se envía mediante el protocolo UDP al motor de videojuegos.

El módulo está diseñado para ser configurable y autónomo. Los parámetros ajustables (rangos de color, número de divisiones de la cuadrícula, dirección UDP, etc.) se centralizan en un archivo `config.json`, que puede editarse manualmente y recargarse durante la ejecución. Todo el módulo se distribuye como un único ejecutable empaquetado con PyInstaller, lo que facilita su uso sin necesidad de instalar Python ni dependencias.

2.2. main.py

2.2.1. Descripción general

Este archivo constituye el punto de entrada del módulo de detección. Su función principal es inicializar la webcam, gestionar el bucle de captura de fotogramas, orquestar la detección de la base, la corrección de perspectiva, la detección de piezas y el envío de los datos mediante UDP. También gestiona la visualización de las ventanas `Base` y `Piezas`, así como la recarga de la configuración durante la ejecución (tecla `R`) y la finalización del programa (tecla `Escape` o cierre de ventanas).

2.2.2. Funciones principales

`main()`

- **Descripción**: ejecuta el flujo principal del programa. Configura la webcam, instancia el socket UDP, procesa fotogramas en bucle, y libera recursos al finalizar.
- **Entrada**: ninguna.
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Inicializa la webcam y ajusta sus propiedades (resolución, FPS, buffer).
  - Crea dos ventanas OpenCV: `Base` y `Piezas`.
  - En cada iteración del bucle:
    - Detecta la base mediante la función `detect_base` del módulo `base_detection`.
    - Si la base es detectada, aplica la corrección de perspectiva, genera máscaras de color, detecta piezas y envía datos por UDP.
    - Si no, muestra una imagen de aviso en la ventana `Piezas`.
    - Muestra mensajes de estado de configuración si existen.
  - Gestiona la tecla `R` para recargar la configuración llamando a `config.load_config`.
  - Gestiona la tecla `Escape` o el cierre de ventanas para finalizar el bucle.
  - Al finalizar, cierra el socket UDP, libera la cámara y destruye las ventanas.

- **Excepciones**: no captura excepciones explícitas. Si la webcam no se abre, imprime un error y finaliza el programa.

2.2.3. Variables globales utilizadas

El archivo depende de las siguientes variables definidas en el módulo `config`:

- `WINDOW_SCALE`: factor de escala para redimensionar las ventanas.
- `WARP_OUTPUT_SIZE`: tamaño del lado más largo de la imagen tras la corrección de perspectiva.
- `status_config_msg`: mensaje de estado de la configuración (por ejemplo, «Configuración recargada»).
- `status_expire`: tiempo de expiración del mensaje de estado.

2.2.4. Dependencias

- `config`: proporciona parámetros de configuración.
- `color_detection`: funciones `get_masks` y `detect_pieces_grid`.
- `base_detection`: función `detect_base`.
- `base_warp`: función `base_warp`.
- `udp_sender`: clase `UDP_socket`.
- `cv2` (OpenCV): captura de imágenes, procesamiento y visualización.
- `numpy`: operaciones con arrays.
- `time`: gestión de mensajes de estado.
- `os`: configuración de variables de entorno.

2.2.5. Notas de implementación

- Para evitar la latencia de inicialización de la webcam en Windows, se desactiva la aceleración por hardware de Microsoft Media Foundation estableciendo la variable de entorno `OPENCV_VIDEOIO_MSMF_ENABLE_HW_TRANSFORMS` a `"0"` antes de importar OpenCV.
- El buffer de fotogramas de la webcam se reduce a 1 para minimizar la latencia.

- La ventana `Base` muestra siempre el fotograma con la detección de la base (si existe) y el mensaje de estado.
- La ventana `Piezas` muestra la imagen con la perspectiva corregida y las piezas detectadas, o una imagen por defecto con el texto `Base NO detectada` cuando no se detecta la base.
- La recarga de configuración se activa con la tecla `R` (mayúscula o minúscula), llamando a `config.load_config` con el parámetro `show_success_message=True`.
- El programa finaliza al pulsar `Escape` o al cerrar cualquiera de las dos ventanas OpenCV.

2.3. config.py

2.3.1. Descripción general

Este archivo gestiona toda la configuración del módulo de detección. Proporciona un mecanismo para almacenar, cargar y validar los parámetros ajustables del sistema mediante un archivo `config.json` en formato JSON. Si el archivo no existe o no es válido, el sistema lo crea automáticamente con los valores por defecto. Además, permite la recarga de la configuración durante la ejecución, lo que facilita la adaptación a diferentes condiciones sin necesidad de reiniciar el programa.

El módulo expone las variables de configuración como variables globales, accesibles desde el resto de archivos del proyecto. También incluye un sistema de mensajes de estado para notificar al usuario si la carga ha sido exitosa o ha fallado.

2.3.2. Funciones principales

`save_config(config_path, data)`

- **Descripción**: escribe un archivo JSON con los parámetros de configuración en la ruta especificada, formateando la lista de colores de forma legible (un color por línea).
- **Entrada**:
  - `config_path`: ruta donde se guardará el archivo.
  - `data`: diccionario con la configuración (estructura similar a `DEFAULT_CONFIG`).
- **Salida**: ninguna.

- **Acciones realizadas**:
  - Formatea la lista `COLORS` con un color por línea para facilitar la edición manual.
  - Combina la lista de colores con el resto de parámetros en un único objeto JSON.
  - Escribe el archivo en codificación UTF-8.
- **Excepciones**: puede lanzar excepciones de E/S si no se puede escribir en el archivo; estas se capturan en `load_config`.

`apply_config(config_dict)`

- **Descripción**: convierte un diccionario de configuración (leído del JSON) en las variables globales del módulo, realizando las conversiones de tipo necesarias (arrays a `np.array`, cadenas a enteros, etc.).
- **Entrada**: `config_dict` (diccionario con los parámetros).
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Para cada color, convierte los rangos HSV a `np.array` y el valor BGR a tupla.
  - Asigna el resto de parámetros a las variables globales correspondientes (con conversión de tipo).
- **Excepciones**: si algún valor no tiene el tipo esperado o falta una clave, lanza excepciones que se capturan en `load_config`.

`load_config(show_success_message=False)`

- **Descripción**: carga la configuración desde el archivo `config.json` ubicado en el directorio del ejecutable. Si el archivo no existe, lo crea automáticamente con los valores por defecto (comportamiento esperado en la primera ejecución). Si el archivo existe pero no es válido (sintaxis JSON, tipos incorrectos, campos faltantes), carga los valores por defecto, regenera el archivo y muestra un mensaje de advertencia. Al recargar la configuración (tecla `R`), puede mostrar un mensaje de confirmación en las ventanas si `show_success_message` es `True`.
- **Entrada**: `show_success_message` (booleano, opcional): si es `True` muestra un mensaje de confirmación en las ventanas al recargar la configuración.

- **Salida**: `True` si la carga fue exitosa (archivo válido), `False` si se usaron los valores por defecto.

- **Acciones realizadas**:

  - Determina la ruta del archivo `config.json` en función de si el programa se ejecuta como script o como ejecutable empaquetado (PyInstaller).
  - Si el archivo no existe, lo crea con los valores por defecto.
  - Si existe, intenta leerlo y parsearlo. Si es válido, aplica la configuración con `apply_config`, mostrando una advertencia.
  - Si ocurre cualquier error (archivo corrupto, JSON mal formateado, tipos incorrectos), carga los valores por defecto y regenera el archivo, mostrando una advertencia.
  - Si `show_success_message` es `True`, establece un mensaje de confirmación que se muestra en las ventanas durante unos segundos.

- **Excepciones**: captura cualquier excepción durante la lectura o aplicación de la configuración, y maneja el fallo de forma controlada.

`set_status_config_msg(message)`

- **Descripción**: establece un mensaje de estado y su tiempo de expiración para ser mostrado en las ventanas de visualización.

- **Entrada**: `message` (cadena de texto con el mensaje a mostrar).

- **Salida**: ninguna.

- **Acciones realizadas**: actualiza las variables globales `status_config_msg` y `status_expire`.

2.3.3. Variables globales

Configuración por defecto

- `DEFAULT_CONFIG`: diccionario que contiene los valores por defecto de todos los parámetros, incluyendo la lista de colores, el factor de escala, el tamaño de la corrección de perspectiva, las divisiones de la cuadrícula, umbrales, dirección UDP, etc.

Parámetros configurables (asignados desde JSON)

- `COLORS`: lista de colores detectables. Cada color es una lista con nombre, límite inferior HSV (`np.array`), límite superior HSV (`np.array`), y color BGR (tupla) mostrado en la interfaz.
- `WINDOW_SCALE`: factor de escala para las ventanas.
- `WARP_OUTPUT_SIZE`: tamaño en píxeles del lado más largo de la imagen tras la corrección de perspectiva.
- `GRID_WIDTH`, `GRID_HEIGHT`: número de divisiones de la cuadrícula.
- `PERCENT_FILLED_CELL`: umbral de ocupación de una celda (0-1) según el porcentaje de píxeles rellenos de un mismo color.
- `MIN_BASE_AREA_RATIO`: porcentaje mínimo de área que debe ocupar la base en la imagen.
- `SHOW_PIECES_GRID`: booleano que controla la visualización de la cuadrícula.
- `UDP_IP`, `UDP_PORT`: dirección IP y puerto para el envío de datos.

Variables de estado para mensajes en ventanas

- `status_config_msg`: cadena con el mensaje actual (vacío por defecto).
- `status_expire`: instante de tiempo (en segundos) en el que el mensaje expira.
- `STATUS_CONFIG_MSG_SUCCESS`: mensaje de éxito («Configuración guardada»).
- `STATUS_CONFIG_MSG_ERROR`: mensaje de error («Error: usada configuración por defecto»).
- `STATUS_CONFIG_MSG_TIME`: duración del mensaje en segundos (2.0).

2.3.4. Dependencias

- `sys`: para detectar si el programa se ejecuta como ejecutable empaquetado (PyInstaller).
- `os`: para manejar rutas de archivos.
- `json`: para leer y escribir el archivo de configuración.
- `time`: para gestionar la expiración de los mensajes de estado.
- `numpy` (como `np`): para convertir los rangos HSV a arrays y facilitar operaciones.

2.3.5. Notas de implementación

- El archivo `config.json` se busca en el directorio del ejecutable (cuando está empaquetado con PyInstaller) o en el directorio del script (cuando se ejecuta desde código fuente).

- La lista de colores se guarda en el JSON con un formato de un color por línea para facilitar la edición manual.

- Los valores se validan durante la asignación (`apply_config`): si un parámetro no tiene el tipo esperado, se lanza una excepción y se cargan los valores por defecto.

- La recarga de la configuración (desde `main.py` con la tecla `R`) invoca `load_config` y permite actualizar los parámetros en tiempo de ejecución, mostrando un mensaje de confirmación en las ventanas.

- Los mensajes de estado se muestran en ambas ventanas (Base y Piezas) durante un tiempo limitado, gestionado por `status_expire`.

- Cabe destacar que los valores `UDP_IP` y `UDP_PORT` también se actualizan durante la recarga, pero no afectan a la conexión actual porque el socket UDP se abre una sola vez al inicio del programa. Para aplicar cambios en la IP o el puerto, es necesario reiniciar el ejecutable del módulo de detección.

2.4. base_detection.py

2.4.1. Descripción general

Este archivo contiene la lógica para detectar la base en la imagen capturada por la webcam. Utiliza la transformada de Hough estándar (`cv2.HoughLines`) para detectar líneas rectas, las agrupa según su orientación mediante el algoritmo K-Means (`cv2.kmeans`) y selecciona el rectángulo de mayor área que cumple condiciones de perpendicularidad y visibilidad. El archivo también incluye funciones auxiliares para filtrar líneas duplicadas, calcular intersecciones, dibujar resultados, etc. La detección se realiza sobre una imagen reducida al 50 % de resolución para optimizar el rendimiento, y las coordenadas se escalan posteriormente a la resolución original.

2.4.2. Funciones principales

`detect_base(frame)`

- **Descripción**: detecta la base en el fotograma recibido. Inicialmente aplica una reducción de resolución, convierte a escala de grises, filtra ruido con un filtro bilateral, binariza con el método de Otsu, aplica un cierre morfológico, detecta bordes con Canny, aplica dilatación, y ejecuta la transformada de Hough. A continuación, filtra líneas duplicadas, las agrupa por ángulo con K-Means, y busca combinaciones de cuatro líneas que formen un rectángulo (dos por cada grupo) que estén dentro del fotograma y tengan el área máxima posible. Devuelve un diccionario con las esquinas, las líneas y el área, o `None` si no se detecta la base.
- **Entrada**: `frame` (imagen BGR capturada por la webcam, resolución completa).
- **Salida**: diccionario con las claves «`corners`» (lista de cuatro puntos (x,y)), «`lines`» (lista de líneas en formato `(rho, theta)`), «`area`» (área del rectángulo en píxeles), o `None`.
- **Acciones realizadas**:
  - Reduce la resolución del fotograma al 50 % para acelerar el procesamiento (`DETECT_FRAME_SCALE = 0.5`).
  - Convierte la imagen a escala de grises.
  - Aplica un filtro bilateral para suavizar preservando bordes.
  - Binariza con el método de Otsu y aplica un cierre morfológico (kernel 5x5, 2 iteraciones).
  - Detecta bordes con Canny y los dilata para reforzarlos.
  - Aplica la transformada de Hough (umbral de 150 votos).
  - Si no se detectan líneas, llama a `draw_base` con `None` y retorna `None`.
  - Elimina líneas duplicadas con `remove_line_duplicates`.
  - Si quedan menos de 4 líneas, retorna `None`.
  - Agrupa líneas por orientación con K-Means (`cluster_lines_by_angle`).
  - Si no hay exactamente 2 clusters o algún cluster tiene menos de 2 líneas, retorna `None`.
  - Itera sobre todas las combinaciones de dos líneas de cada cluster, comprueba perpendicularidad y que las esquinas estén dentro del frame.

- Selecciona el rectángulo de mayor área que supere `MIN_BASE_AREA_RATIO`.
- Escala las coordenadas a la resolución original.
- Dibuja el resultado con `draw_base`.
- Retorna el rectángulo seleccionado (o `None`).

- **Excepciones**: no captura excepciones explícitas. Si alguna operación falla (por ejemplo, `cv2.HoughLines` no encuentra líneas), retorna `None` de forma controlada.

2.4.3. Funciones auxiliares

`draw_base(frame, best_rectangle)`

- **Descripción**: dibuja sobre el fotograma el resultado de la detección. Si la base ha sido detectada, muestra el texto verde `"Base detectada"`, dibuja las líneas infinitas en morado, el contorno del rectángulo en naranja y las esquinas en azul. Si no se detecta, muestra el texto rojo `"Base NO detectada"`.
- **Entrada**: `frame` (imagen donde dibujar), `best_rectangle` (diccionario con «`lines`» y «`corners`» o `None`).
- **Salida**: ninguna (modifica la imagen directamente).

`draw_line(frame, line, color=(0,0,255), thickness=1)`

- **Descripción**: dibuja una línea infinita en el fotograma a partir de sus parámetros `(rho, theta)`. La línea se extiende a lo largo de toda la imagen.
- **Entrada**: `frame`, `line` (tupla `(rho, theta)`), `color`, `thickness`.

`remove_line_duplicates(lines, rho_tolerance=50, theta_tolerance=np.radians(10))`

- **Descripción**: elimina líneas duplicadas o muy próximas entre sí. Normaliza cada línea a `rho >= 0` y `theta` en el rango `[0, π)`. Compara la diferencia de `rho` y `theta` con los umbrales; si son similares, descarta la línea nueva.
- **Entrada**: lista de líneas en formato `(rho, theta)`.
- **Salida**: lista de líneas únicas.

`cluster_lines_by_angle(lines, k=2)`

- **Descripción**: agrupa las líneas en `k` clusters según su orientación (ángulo `theta`). Para evitar el problema de la circularidad de los ángulos (0 y π son la misma orientación), transforma cada ángulo en coordenadas $(cos(2 * theta), sin(2 * theta))$ y aplica K-Means con OpenCV.
- **Entrada**: lista de líneas `(rho, theta)`, número de clusters `k` (por defecto 2).
- **Salida**: lista de `k` listas, cada una con las líneas de ese cluster.

`are_lines_perpendicular(line1, line2, tolerance=np.radians(5))`

- **Descripción**: comprueba si dos líneas son perpendiculares dentro de una tolerancia de $\pm 5^{o}$. Para ello, calcula la diferencia de sus ángulos, la normaliza a `[0, π/2]` y la compara con $\pi/2$.
- **Entrada**: dos líneas `(rho, theta)`.
- **Salida**: `True` si son perpendiculares, `False` en caso contrario.

`get_line_intersection(line1, line2)`

- **Descripción**: calcula la intersección de dos líneas resolviendo un sistema de ecuaciones lineales a partir de sus parámetros `(rho, theta)`.
- **Entrada**: dos líneas `(rho, theta)`.
- **Salida**: punto `(x, y)` o `None` si las líneas son paralelas.

`is_point_inside_frame(point, frame)`

- **Descripción**: comprueba si un punto está dentro de los límites de la imagen.
- **Entrada**: `point` (tupla `(x, y)`), `frame` (imagen).
- **Salida**: `True` si el punto está dentro, `False` en caso contrario.

`sort_corners_clockwise(corners)`

- **Descripción**: ordena las cuatro esquinas de un rectángulo en sentido horario, comenzando por la esquina superior izquierda. Utiliza la suma y la diferencia de coordenadas para determinar el orden.
- **Entrada**: lista de cuatro puntos `(x, y)`.
- **Salida**: lista de cuatro puntos ordenados: `[top-left, top-right, bottom-right, bottom-left]`.

`rectangle_area(corners)`

- **Descripción**: calcula el área de un rectángulo a partir de sus cuatro esquinas. Usa la distancia entre el primer y segundo punto (ancho) y entre el primero y el cuarto (alto).
- **Entrada**: lista de cuatro puntos ordenados en sentido horario.
- **Salida**: área (float).

2.4.4. Variables globales utilizadas

- `DETECT_FRAME_SCALE`: factor de reducción de resolución (0.5) para optimizar el rendimiento.
- `MIN_BASE_AREA_RATIO`: umbral mínimo de área (definido en `config`) para filtrar detecciones incorrectas.

2.4.5. Dependencias

- `cv2` (OpenCV): procesamiento de imagen, transformada de Hough, K-Means, operaciones morfológicas, dibujo.
- `numpy`: operaciones con arrays, álgebra lineal.
- `itertools.combinations`: generar todas las combinaciones posibles de dos líneas de cada cluster para evaluar su perpendicularidad y calcular las esquinas del posible rectángulo.
- `config`: proporciona parámetros de configuración (umbral de área mínima).

2.4.6. Notas de implementación

- La detección se realiza sobre una imagen reducida al 50 % para acelerar el procesamiento; todas las coordenadas se escalan posteriormente a la resolución original.
- La transformada de Hough se ejecuta con umbral de 150 votos (balance entre sensibilidad y falsos positivos).
- El filtrado de líneas duplicadas usa tolerancias de 50 píxeles en `rho` y 10° en `theta`.
- La agrupación por ángulo con K-Means utiliza la técnica de duplicar el ángulo para resolver la circularidad.

- Las líneas deben ser perpendiculares con una tolerancia de $\pm 5°$.
- El rectángulo seleccionado debe tener un área superior al 5 % del área total de la imagen (valor configurable en `config.json`).
- Si la base no se detecta, la función retorna `None` y la ventana `Base` muestra el mensaje correspondiente.

2.5. base_warp.py

2.5.1. Descripción general

Este archivo contiene la función encargada de aplicar la corrección de perspectiva a la imagen de la base detectada. A partir de las cuatro esquinas de la base (detectadas en `base_detection.py`), calcula una transformación que convierte la región de la base en una imagen rectificada con vista cenital, eliminando distorsiones causadas por la inclinación de la cámara o de la base. El tamaño de la imagen de salida se escala para que su lado más largo sea igual a `WARP_OUTPUT_SIZE`, manteniendo la relación de aspecto original.

2.5.2. Función principal

`base_warp(frame, corners, output_longest_side=config.WARP_OUTPUT_SIZE)`

- **Descripción**: aplica una transformación de perspectiva al fotograma original para obtener una vista cenital de la base. Utiliza las cuatro esquinas de la base (ordenadas en sentido horario) para calcular la matriz de transformación y recorta la región de la base, devolviendo una imagen rectificada.
- **Entrada**:
  - `frame`: imagen BGR original, resolución completa.
  - `corners`: lista de cuatro puntos `(x, y)` ordenados en sentido horario (superior-izquierda, superior-derecha, inferior-derecha, inferior-izquierda).
  - `output_longest_side` (int, opcional): tamaño en píxeles del lado más largo de la imagen de salida. Por defecto usa `config.WARP_OUTPUT_SIZE`.
- **Salida**: imagen BGR con la perspectiva corregida (vista cenital), de tamaño `(output_width, output_height)`.

- **Acciones realizadas**:
  - Calcula el ancho y alto promediando las distancias entre las esquinas correspondientes para estimar las dimensiones reales de la base.
  - Si el ancho o alto son menores que 1 píxel, retorna el fotograma original (medida de seguridad para evitar división por cero).
  - Determina la escala necesaria para que el lado más largo de la imagen de salida sea igual a `output_longest_side`.
  - Calcula las dimensiones de salida (`output_width`, `output_height`) manteniendo la relación de aspecto.
  - Define las cuatro esquinas de la imagen de salida (esquina superior-izquierda en (0,0), etc.).
  - Calcula la matriz de transformación de perspectiva con `cv2.getPerspectiveTransform`.
  - Aplica la transformación con `cv2.warpPerspective` y devuelve la imagen corregida.
- **Excepciones**: la función no maneja excepciones internamente, ya que se presupone que la entrada ha sido correctamente obtenida. Los errores se propagan al llamador.

2.5.3. Variables globales utilizadas

- `config.WARP_OUTPUT_SIZE`: tamaño del lado más largo de la imagen de salida (por defecto 800 píxeles).

2.5.4. Dependencias

- `cv2` (OpenCV): funciones de transformación de perspectiva (`getPerspectiveTransform`, `warpPerspective`).
- `numpy`: operaciones matemáticas con arrays (`np.linalg.norm` para distancias).
- `config`: proporciona el parámetro `WARP_OUTPUT_SIZE`.

2.5.5. Notas de implementación

- El promedio de las distancias entre esquinas opuestas se usa para obtener un ancho y alto más robustos, compensando posibles errores de detección en las esquinas.

- La imagen de salida mantiene la relación de aspecto original de la base, evitando deformaciones.
- El tamaño de salida está controlado por `WARP_OUTPUT_SIZE`, que es configurable en el archivo `config.json`.
- La transformación de perspectiva se aplica sobre el fotograma completo, pero como la matriz está definida por las esquinas de la base, solo la región de la base se recorta y endereza.
- Si las dimensiones calculadas son menores que 1 píxel (lo que podría ocurrir con entradas inválidas), la función retorna el fotograma original sin modificar como medida de seguridad.

2.6. color_detection.py

2.6.1. Descripción general

Este archivo contiene la lógica para identificar las piezas sobre la base tras la corrección de perspectiva. Utiliza máscaras de color en el espacio HSV para segmentar cada color definido en la configuración, y una cuadrícula ajustable para determinar la posición de cada pieza. La detección se basa en un umbral de ocupación por celda que evita falsos positivos y compensa pequeños desajustes de la transformación de perspectiva. También incluye funciones para dibujar la cuadrícula y los círculos de colores sobre la imagen.

2.6.2. Funciones principales

`get_masks(frame)`

- **Descripción**: genera una máscara binaria para cada color definido en la configuración. Convierte la imagen de BGR a HSV y aplica `cv2.inRange` para cada rango de color. Si el rango es circular (es decir, el límite inferior de H es mayor que el superior), la función divide el rango en dos intervalos y combina las máscaras con una operación OR.
- **Entrada**: `frame` (imagen BGR con la perspectiva corregida).
- **Salida**: diccionario `masks` donde cada clave es el nombre del color y el valor es una máscara binaria (blanco y negro). En ella los píxeles blancos indican las regiones que están dentro del rango y los negros el resto.

- **Acciones realizadas**:
  - Convierte la imagen de BGR a HSV.
  - Para cada color en `config.COLORS`, extrae su nombre, rango inferior y superior.
  - Si el límite inferior de H es mayor que el superior (rango circular), genera dos máscaras:
    - Una desde el límite inferior hasta 179.
    - Otra desde 0 hasta el límite superior.

    Las combina con `cv2.bitwise_or`.
  - En caso contrario, aplica `cv2.inRange` directamente.
  - Almacena cada máscara en el diccionario con el nombre del color como clave.
- **Excepciones**: no captura excepciones explícitas; asume que los rangos de color en `config` son válidos.

`detect_pieces_grid(frame, masks)`

- **Descripción**: determina la posición de las piezas dividiendo la imagen en una cuadrícula y evaluando el porcentaje de ocupación de cada color en cada celda. Devuelve una lista de piezas detectadas con su color y coordenadas de celda.
- **Entrada**:
  - `frame`: imagen con la perspectiva corregida.
  - `masks`: diccionario con las máscaras de color obtenidas de `get_masks`.
- **Salida**: lista de diccionarios, cada uno con:
  - «`name`»: nombre del color.
  - «`cell`»: tupla `(col, row)` con las coordenadas de celda.
  - «`colorBGR`»: color BGR para dibujarlo en la interfaz.
- **Acciones realizadas**:
  - Calcula el ancho y alto de cada celda dividiendo las dimensiones del frame por `GRID_WIDTH` y `GRID_HEIGHT`.
  - Recorre todas las celdas por filas y columnas.
  - Para cada celda, extrae la región de interés (ROI) de cada máscara y cuenta los píxeles blancos.

- Selecciona el color con mayor número de píxeles que supere el umbral `PERCENT_FILLED_CELL`.
- Si se encuentra un color válido, añade la pieza a la lista.
- Llama a `draw_pieces_grid` para dibujar la cuadrícula y los círculos sobre el frame.

- **Excepciones**: La función no captura excepciones explícitas. Si no se detectan piezas (frame vacío o máscaras sin regiones), retorna una lista vacía. Este es el comportamiento esperado, ya que la lista vacía se envía por UDP para que el motor de videojuegos elimine los actores correspondientes y refleje fielmente el estado de la base.

```
draw_pieces_grid(frame, pieces, draw_grid=config.SHOW_PIECES_GRID)
```

- **Descripción**: dibuja sobre la imagen la cuadrícula (líneas moradas) y los círculos de colores en el centro de cada celda ocupada por una pieza.
- **Entrada**:
  - `frame`: imagen con la perspectiva corregida.
  - `pieces`: lista de piezas detectadas.
  - `draw_grid` (booleano, opcional): controla si se dibuja la cuadrícula. Por defecto usa `config.SHOW_PIECES_GRID`.
- **Salida**: ninguna (modifica la imagen directamente).
- **Acciones realizadas**:
  - Si `draw_grid` es `True`, dibuja líneas moradas verticales y horizontales para formar la cuadrícula.
  - Para cada pieza, calcula el centro de la celda en píxeles y dibuja un círculo relleno del color detectado con un borde negro para mejorar su visibilidad sobre la imagen.

2.6.3. Variables globales utilizadas

- `config.COLORS`: lista de colores detectables (nombre, rangos HSV, color BGR).
- `config.GRID_WIDTH`, `config.GRID_HEIGHT`: número de divisiones de la cuadrícula.
- `config.PERCENT_FILLED_CELL`: umbral mínimo de ocupación de una celda (0-1, por defecto 60 %) para evitar falsos positivos.

- `config.SHOW_PIECES_GRID`: booleano que controla la visualización de la cuadrícula.

2.6.4. Dependencias

- `cv2` (OpenCV): conversión de color, operaciones con máscaras, dibujo de círculos y líneas.
- `numpy`: gestión de arrays, necesaria para el manejo interno de imágenes en OpenCV.
- `config`: proporciona los parámetros de configuración (colores, cuadrícula, umbrales).

2.6.5. Notas de implementación

- El espacio de color HSV se utiliza por su robustez frente a cambios de iluminación, ya que el tono (H) se mantiene relativamente estable.
- Los rangos de color pueden ser circulares. Cuando el límite inferior de H es mayor que el superior, se generan dos máscaras y se combinan.
- La cuadrícula se ajusta mediante `GRID_WIDTH` y `GRID_HEIGHT`. Los valores por defecto (16×16) dividen la base LEGO de 32×32 studs en celdas de 2×2 studs, que se corresponden con las piezas LEGO más comunes y básicas.
- El umbral `PERCENT_FILLED_CELL` compensa pequeños errores de perspectiva y filtra ruido en las máscaras.
- Las piezas se dibujan con un círculo relleno del color detectado y un borde negro para mejorar la visibilidad.
- La cuadrícula se puede ocultar cambiando `SHOW_PIECES_GRID` a `False` en el archivo de configuración.

2.7. udp_sender.py

2.7.1. Descripción general

Este archivo implementa la comunicación mediante el protocolo UDP entre el módulo de detección y el motor de videojuegos. Proporciona una clase que reutiliza un mismo socket para enviar la lista de piezas detectadas en formato JSON a la dirección IP y puerto configurados. El envío se realiza incluso cuando la lista está

vacía, permitiendo que el motor gestione la eliminación de todos los actores cuando la base queda vacía.

2.7.2. Clase `UDP_socket`

`__init__(self, ip=config.UDP_IP, port=config.UDP_PORT)`

- **Descripción**: constructor de la clase. Crea un socket UDP y almacena la dirección IP y el puerto de destino (tomados de `config` por defecto).
- **Entrada**:
  - `ip` (str, opcional): dirección IP de destino. Por defecto usa `config.UDP_IP`.
  - `port` (int, opcional): puerto de destino. Por defecto usa `config.UDP_PORT`.
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Almacena la IP y el puerto en atributos de la instancia.
  - Crea un socket UDP con `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`.
  - Inicializa el flag `socket_error_printed` a `False` para controlar la impresión de errores de red.

`send_pieces(self, pieces)`

- **Descripción**: serializa la lista de piezas detectadas en formato JSON y la envía mediante UDP a la dirección y puerto configurados.
- **Entrada**: `pieces` (lista de diccionarios, cada uno con «`name`» y «`cell`»).
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Construye un diccionario con la clave «`pieces`» que contiene una lista de objetos con «`color`», «`x`» y «`y`».
  - Serializa el diccionario a JSON con `json.dumps`.
  - Codifica el mensaje a UTF-8 y lo envía mediante `sock.sendto`. la ejecución).
- **Excepciones**: captura `OSError` en el envío. Si ocurre un error de red, imprime un mensaje solo la primera vez para evitar saturar la consola (`socket_error_printed` lo controla y persiste durante toda la ejecución).

`close_socket(self)`

- **Descripción**: cierra el socket UDP para liberar el recurso.
- **Entrada**: ninguna.
- **Salida**: ninguna.
- **Acciones realizadas**: llama a `self.sock.close()`.

2.7.3. Variables globales utilizadas

- `config.UDP_IP`: dirección IP de destino (por defecto `127.0.0.1`).
- `config.UDP_PORT`: puerto de destino (por defecto `5005`).

2.7.4. Dependencias

- `socket`: creación y manejo del socket UDP.
- `json`: serialización de la lista de piezas a formato JSON.
- `config`: proporciona los parámetros de conexión (IP y puerto).

2.7.5. Notas de implementación

- El socket se crea en el constructor y se reutiliza en todos los envíos, evitando abrir y cerrar el socket en cada iteración del bucle principal, lo que mejora el rendimiento.
- El mensaje se envía incluso si la lista de piezas está vacía. Esto permite que el motor de videojuegos sepa cuándo la base no contiene piezas y elimine todos los actores de la escena.
- Los errores de red (como un puerto no disponible) se capturan y se imprimen una sola vez para no inundar la consola con mensajes repetidos en cada fallo.
- La clase está diseñada para ser utilizada desde `main.py`, que crea una instancia al inicio y la reutiliza durante todo el bucle. El socket se cierra al finalizar el programa mediante `close_socket`.

3. Módulo de visualización (Unreal Engine)

3.1. Estructura general del módulo

El módulo de visualización es el encargado de recibir los datos enviados por el módulo de detección (Python) y traducirlos en una representación 3D dentro del motor de videojuegos. Está implementado en C++ para UE y se integra en el proyecto como código fuente, permitiendo su reutilización en distintos proyectos.

El flujo de procesamiento del módulo se organiza en dos etapas principales:

1. **Recepción y procesamiento de datos**: mediante un receptor UDP asíncrono, el módulo escucha en un puerto configurable, recibe los datagramas y los convierte en una lista estructurada de piezas (color y coordenadas de celda). La recepción se realiza en un hilo secundario para no bloquear el hilo principal del juego.

2. **Instanciación y actualización de actores**: los datos recibidos se utilizan para crear, actualizar o eliminar actores 3D en la escena. La actualización es diferencial, solo se modifican las celdas que han cambiado, lo que optimiza el rendimiento. La correspondencia entre colores y clases de actor es configurable desde el editor, lo que permite adaptar el comportamiento sin modificar el código.

El módulo está diseñado para ser fácilmente configurable y reutilizable. Todos los parámetros relevantes (asignación color/actor, tamaño de celda, altura de instanciación, dirección IP y puerto, etc.) se exponen en el editor de UE y son modificables desde Blueprints (BP). Esto permite que diseñadores sin conocimientos de C++ puedan adaptar la herramienta a sus necesidades.

Además, gracias al uso de un protocolo estándar (UDP) y un formato de intercambio legible (JSON), el módulo de detección es independiente del motor de visualización, por lo que podría adaptarse a otros motores de videojuegos (Godot, Unity, etc.) implementando un receptor y un instanciador equivalentes.

3.2. UDPReceiver

3.2.1. Descripción general

La clase `UDPReceiver` es un actor de UE que implementa la recepción de datagramas UDP desde el módulo de detección (Python). Funciona de forma asíncrona, utilizando un hilo secundario para la escucha y recepción, lo que evita bloquear

el hilo principal del juego. Los mensajes recibidos (en formato JSON) se procesan en el hilo principal para extraer la lista de piezas, se notifican a través del evento `OnPiecesReceived` y se envían al componente `PieceSpawnerComponent` para actualizar la escena.

La clase expone parámetros configurables desde el editor y funciones para controlar manualmente la recepción, lo que la hace reutilizable en distintos proyectos.

3.2.2. Variables principales

Configurables en el editor

- `bAutoStart`: booleano que controla si el receptor se inicia automáticamente al comenzar la partida (`true` por defecto).
- `SocketName`: nombre del socket (cadena, por defecto «`UDPReceiver`»).
- `IP`: dirección IPv4 de destino (cadena, por defecto «`127.0.0.1`»).
- `Port`: puerto de escucha (entero, por defecto `5005`).
- `BufferSize`: tamaño del buffer de recepción en bytes (entero, por defecto `2097152`, es decir, 2 MB).

Internas

- `PieceSpawner`: puntero al componente `UPieceSpawnerComponent` asociado (se obtiene automáticamente en `BeginPlay`).
- `Socket`: puntero al socket UDP subyacente.
- `UDPReceiver`: puntero al receptor asíncrono de UE.

3.2.3. Estructura de datos

`FPieceData` es una estructura que representa una pieza detectada, con tres campos:

- `Color` (`FString`): nombre del color.
- `X` (`int32`): coordenada de columna en la cuadrícula.
- `Y` (`int32`): coordenada de fila en la cuadrícula.

Sus propiedades son accesibles desde BPs.

3.2.4. Funciones principales

`StartUDPReceiver()`

- **Descripción**: inicia el receptor UDP. Crea un socket UDP configurado con la IP, puerto y tamaño de buffer especificados, y configura un receptor asíncrono en un hilo secundario. Si el receptor ya está activo, no hace nada y retorna `true`.
- **Entrada**: ninguna.
- **Salida** (bool): `true` si se inició correctamente, `false` en caso de error (IP inválida, fallo al crear el socket, etc.).
- **Acciones realizadas**:
  - Valida la dirección IP con `FIPv4Address::Parse`.
  - Crea el socket con `FUdpSocketBuilder` (no bloqueante, reutilizable, con tamaño de buffer).
  - Crea el receptor asíncrono y enlaza el callback `OnDataReceived` al evento de recepción.
  - Inicia el receptor con `Start()`.
  - Registra un mensaje de log en caso de éxito o error.
- **Excepciones**: no lanza excepciones, los errores se notifican mediante logs y el valor de retorno.

`StopUDPReceiver()`

- **Descripción**: detiene el receptor UDP, cierra y destruye el socket, liberando los recursos asociados.
- **Entrada**: ninguna.
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Si el receptor existe, llama a `Stop()` y elimina el objeto.
  - Si el socket existe, lo cierra y destruye mediante `ISocketSubsystem`.
  - Registra un mensaje de log.

`IsReceiving()`

- **Descripción**: consulta si el receptor UDP está activo.
- **Entrada**: ninguna.
- **Salida** (bool): `true` si el receptor está activo y escuchando, `false` en caso contrario.

`OnPiecesReceived(const TArray<FPieceData>& Pieces)`

- **Descripción**: evento implementable en BPs que se dispara en el hilo principal cuando se recibe un nuevo mensaje con una lista de piezas (incluso vacía). Permite añadir lógica personalizada sin modificar C++.
- **Entrada**: `Pieces`, array de estructuras `FPieceData` con los datos de las piezas detectadas.
- **Salida**: ninguna.
- **Nota**: no tiene implementación en C++.

`OnDataReceived(const FArrayReaderPtr& Message, const FIPv4Endpoint& EndPt)`

- **Descripción**: callback ejecutado en el hilo secundario cuando llega un datagrama. Convierte los bytes a una cadena `FString`, limpia caracteres residuales (elimina todo lo que haya después de la última llave «`}`») y programa una tarea asíncrona en el hilo principal para procesar el mensaje.
- **Entrada**: `Message`, puntero a los datos recibidos.
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Convierte los datos a `FString` usando codificación UTF-8.
  - Busca la última llave «`}`» y descarta cualquier carácter posterior.
  - Ejecuta `ProcessMessage` en el hilo principal mediante `AsyncTask`.

`ProcessMessage(const FString& Message)`

- **Descripción**: procesa el mensaje JSON en el hilo principal. Deserializa el JSON, extrae el array `pieces`, convierte cada elemento a `FPieceData`, dispara el evento `OnPiecesReceived` y actualiza el `PieceSpawnerComponent` si existe.
- **Entrada**: `Message`, cadena con el JSON limpio.

- **Salida**: ninguna.
- **Acciones realizadas**:
  - Comprueba que el objeto y el mundo sean válidos (evita llamadas durante el cambio de nivel o destrucción).
  - Deserializa el JSON con `FJsonSerializer::Deserialize`.
  - Si falla, registra un warning y retorna.
  - Extrae el array `pieces` con `TryGetArrayField`.
  - Itera sobre el array, convierte cada elemento a `FPieceData` (con `TryGetStringField` y `TryGetNumberField`).
  - Dispara el evento `OnPiecesReceived`.
  - Si `PieceSpawner` es válido, llama a `UpdatePieces` para actualizar la escena.
- **Excepciones**: no captura excepciones, los errores de parseo se notifican mediante logs y no interrumpen el flujo.

`BeginPlay()`

- **Descripción**: función virtual de UE, ejecutada al inicio de la partida. Busca el componente `PieceSpawner` en el actor y, si `bAutoStart` es `true`, inicia el receptor UDP.
- **Entrada**: ninguna.
- **Salida**: ninguna.

`EndPlay(EEndPlayReason::Type EndPlayReason)`

- **Descripción**: función virtual de UE, ejecutada cuando el actor se destruye o se cambia de nivel. Detiene el receptor UDP y libera el socket para evitar fugas de recursos.
- **Entrada**: `EndPlayReason`, razón de la finalización (no se usa).
- **Salida**: ninguna.

3.2.5. Dependencias

- `CoreMinimal.h`, `GameFramework/Actor.h`: núcleo de UE.
- `Networking.h`: sockets y comunicaciones UDP.

- `Json.h`: parseo de JSON.
- `PieceSpawnerComponent.h`: componente que instancia los actores 3D.
- `Async/Async.h`: tareas asíncronas para cambio de hilo.
- `UDPReceiver.generated.h`: generado por UE.

3.2.6. Notas de implementación

- La recepción UDP se realiza en un hilo secundario (`FUdpSocketReceiver`) para no bloquear el hilo principal del juego.
- El mensaje JSON se limpia descartando cualquier carácter que aparezca después de la última llave «`}`», ya que durante la conexión UDP se puede añadir basura al final del datagrama.
- El procesamiento del JSON y la actualización de la escena se ejecutan en el hilo principal (`AsyncTask` con `ENamedThreads::GameThread`) porque las operaciones de creación de actores y comunicación con BPs deben realizarse allí.
- El evento `OnPiecesReceived` es un `BlueprintImplementableEvent`, lo que permite a diseñadores añadir lógica personalizada sin modificar C++.
- La referencia a `PieceSpawner` se obtiene automáticamente en `BeginPlay` mediante `FindComponentByClass`, lo que simplifica la configuración en el editor.
- El parámetro `BufferSize` permite ajustar el tamaño del buffer de recepción (por defecto 2 MB) para evitar pérdida de datagramas en caso de ráfagas de datos.
- El socket se configura como no bloqueante y reutilizable para permitir su uso continuo y su cierre limpio.

3.3. PieceSpawnerComponent

3.3.1. Descripción general

El componente `PieceSpawnerComponent` es el encargado de gestionar la instanciación y actualización de los actores 3D en la escena a partir de los datos recibidos por el receptor UDP. Se añade a un actor (normalmente al mismo que contiene el `UDPReceiver`) y mantiene un mapeo configurable entre nombres de color y clases

de actor. Cuando recibe un nuevo array de piezas, actualiza la escena de forma diferencial: elimina los actores en celdas que ya no están ocupadas, mantiene los que coinciden y spawnea nuevos actores cuando es necesario.

La clase expone todos sus parámetros configurables en el editor, lo que permite a diseñadores ajustar el comportamiento sin modificar el código fuente.

3.3.2. Variables principales

Configurables en el editor

- `ColorToActor`: mapa que asocia un nombre de color (`FString`) con una clase de actor (`TSubclassOf<AActor>`). Se configura directamente en el editor y puede modificarse en tiempo de ejecución.
- `CellSize`: distancia entre centros de celdas adyacentes (en unidades de UE). Por defecto `100.0`.
- `SpawnHeight`: altura en Z a la que se instancian los actores. Por defecto `0.0` (el suelo).

Internas

- `CurrentActorsByCell`: mapa que almacena, para cada celda (coordenadas `FIntPoint`), el puntero al actor instanciado. Se usa para gestionar la actualización diferencial.

3.3.3. Funciones principales

`UpdatePieces(const TArray<FPieceData>& Pieces)`

- **Descripción**: actualiza la escena a partir del array de piezas recibido. Compara el estado actual con el nuevo y realiza las operaciones necesarias (eliminación, mantenimiento o instanciación) de forma eficiente.
- **Entrada**: `Pieces`, array de estructuras `FPieceData` (color, X, Y) procedentes del receptor UDP.
- **Salida**: ninguna.
- **Acciones realizadas**:
  - Si el array está vacío, elimina todos los actores y retorna.
  - Construye un conjunto (`TSet`) con las coordenadas de las nuevas celdas.

- Itera sobre `CurrentActorsByCell`: si una celda ya no está en el conjunto, destruye el actor y elimina la entrada.
- Para cada pieza:
  - Busca la clase de actor asociada al color en `ColorToActor`. Si no existe, registra un `Warning` y continúa.
  - Comprueba si ya hay un actor en la celda.
  - Si la clase mapeada es `None`, destruye el actor existente (si lo hay) y elimina la entrada.
  - Si la clase es válida:
    - Si el actor existente es de la misma clase, lo mantiene.
    - Si el actor existente es de otra clase, lo destruye.
    - Spawnea un nuevo actor de la clase correspondiente en la posición calculada con `CellToWorld`.

- **Excepciones**: no captura excepciones. Si no se puede obtener el mundo o el actor no se spawnea correctamente, se registra un mensaje de error y se continúa con la siguiente pieza.

`ClearAllPieces()`

- **Descripción**: elimina todos los actores instanciados y vacía el mapa de seguimiento.
- **Entrada**: ninguna.
- **Salida**: ninguna.
- **Acciones realizadas**: itera sobre `CurrentActorsByCell`, destruye cada actor y vacía el mapa.

`CellToWorld(int32 X, int32 Y)` **(privado)**

- **Descripción**: convierte las coordenadas de celda (columna, fila) a una posición absoluta en el mundo, centrada en la celda y con altura `SpawnHeight`.
- **Entrada**: `X`, `Y`; coordenadas de celda.
- **Salida**: `FVector`, posición mundial del centro de la celda.
- **Acciones realizadas**: calcula `CenterX = (X + 0.5f) * CellSize` y `CenterY = (Y + 0.5f) * CellSize`, y devuelve `FVector(CenterX, CenterY, SpawnHeight)`.

3.3.4. Dependencias

- `CoreMinimal.h`, `Components/ActorComponent.h`: núcleo de UE.
- `UDPReceiver.h`: define la estructura `FPieceData` utilizada como entrada.
- `PieceSpawnerComponent.generated.h`: generado por UE.

3.3.5. Notas de implementación

- El componente utiliza una actualización diferencial para minimizar el coste computacional: solo destruye actores en celdas que han quedado vacías, reutiliza los que coinciden y spawnea los nuevos.
- El mapeo `ColorToActor` es un `TMap` configurable en el editor, lo que permite añadir, modificar o eliminar entradas sin modificar el código.
- Si se asigna `None` a una clase en el mapa, ese color no generará ningún actor, lo que permite desactivar temporalmente un tipo de pieza.
- La posición de los actores se calcula en el centro de la celda, con la altura configurable `SpawnHeight` (útil para ajustar actores que puedan aparecer flotando o dentro del suelo).
- El componente no tiene tick activado (`PrimaryComponentTick.bCanEverTick = false`), ya que solo actúa cuando recibe nuevos datos a través de `UpdatePieces`.
- Las funciones `UpdatePieces` y `ClearAllPieces` están expuestas a BPs, lo que permite su uso desde la lógica visual del proyecto.

3.4. Integración en proyectos de UE

Esta sección describe los pasos técnicos necesarios para añadir el módulo de visualización a un proyecto de UE desde la perspectiva del programador. Para una guía visual paso a paso, consúltese el Anexo E (Manual de usuario).

3.4.1. Creación de las clases en el editor

Antes de copiar ningún archivo, es necesario crear las dos clases C++ desde el editor de UE:

1. `UDPReceiver` de tipo **Actor**.
2. `PieceSpawnerComponent` de tipo **ActorComponent**.

Este paso es imprescindible porque UE genera los archivos base (`.h` y `.cpp`) y los registra en el sistema de compilación del proyecto. Sin estos archivos generados, el motor no reconocerá las clases aunque se copien manualmente en la carpeta `Source`.

3.4.2. Sustitución por los archivos del módulo

Una vez creadas las clases y cerrado el editor:

1. Copiar los archivos proporcionados (`UDPReceiver.h`, `UDPReceiver.cpp`, `PieceSpawnerComponent.h`, `PieceSpawnerComponent.cpp`) en la carpeta `Source/[NombreProyecto]/`, **sobrescribiendo** los archivos generados por UE. También se puede hacer manualmente, abriendo cada archivo y pegando el contenido.

2. En los archivos `.h` de ambas clases, reemplazar el prefijo `CHROMAVISIONUE_API` por el nombre del proyecto en mayúsculas seguido de `_API` (ej. `MI_PROYECTO_API`).

3. En el archivo `[NombreProyecto].Build.cs`, añadir las dependencias: `Sockets`, `Networking` y `Json` en la sección `PublicDependencyModuleNames`.

4. Compilar el proyecto desde Visual Studio.

3.4.3. Configuración en el editor

Una vez compilado, el módulo se configura desde el editor:

- Crear un BP a partir de `UDPReceiver` y añadirle un componente `PieceSpawnerComponent`.

- Configurar el mapa `ColorToActor` (asignación color/clase de actor), así como `CellSize`, `SpawnHeight`, IP y puerto.

- Colocar una instancia del BP en el nivel. La recepción UDP se iniciará automáticamente porque por defecto `bAutoStart` está activo.

3.4.4. Portabilidad a otros motores

El módulo de detección es independiente del motor de videojuegos. Gracias al uso de UDP y el formato JSON como estándares de comunicación, cualquier motor que pueda recibir datagramas UDP y procesar JSON (Godot, Unity, etc.) puede implementar su propio módulo de visualización. El módulo de detección no requiere ninguna modificación para adaptarse a otros motores, solo es necesario replicar la lógica de recepción e instanciación en el motor de destino.