Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS E INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado de Ingeniería de computadores

Curso Académico 2022/2023

INTERFAZ DE CONTROL DE UN CUBO DE RUBIK CON UNITY Y CREACIÓN DE UN ACCESORIO CON RESPUESTA HÁPTICA Y VISUAL

Autor: Carlos Chamizo Cano

Tutor: José San Martín López

Resumen

En este trabajo se desarrolla un interfaz háptico con un cubo de Rubik como base y su integración como plugin en UNITY.

Este proyecto consta de dos partes diferenciadas. La primera parte consiste en la conexión y captura de datos proveniente de un cubo de Rubik bluetooth ya existente y su implementación en UNITY. Y la segunda, la creación del prototipo de un accesorio que complementa a la interfaz del cubo mediante el uso de un microcontrolador compatible con Arduino para integrar respuesta háptica en forma de vibración, respuesta visual a través de una matriz de leds y comunicación a través de un interruptor de agitación.

El juego realizado para el TFG de videojuegos donde se han probado este proyecto es el que está en el siguiente ENLACE.

Agradecimientos

Me gustaría agradecer en primer lugar a mi tutor, José San Martín López por el tiempo que me ha dedicado en esta última etapa del TFG ya que si no fuera por su ayuda y disponibilidad no habría sido posible.

Gracias a mi familia, a mis padres por apoyarme y brindarme de todos los recursos necesarios además de a mi hermana y a Edgar por su apoyo e interés en mi proyecto.

Gracias a mi madrina que me cuida desde aquí con cariño y comprensión y a mi padrino que sé que está muy orgulloso de mí allí arriba.

Gracias a todo el resto de mi familia que han contribuido de alguna u otra forma.

Gracias a mis mascotas Nana y Lilly por el apoyo emocional e incondicional en los momentos más duros.

Gracias a todos mis amigos que me han apoyado y en específico a Javi por ayudarme a soldar el circuito.

Gracias a mi novia Andrea por apoyarme en los momentos más difíciles y ayudarme a seguir adelante con los proyectos.

Y finalmente gracias a todos los que estáis leyendo esto, espero que la lectura os sea leve.

Contenidos

Índice de Ilustraciones

1. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es aplicar y demostrar todos los conocimientos adquiridos a lo largo de la carrera en el desarrollo de un proyecto que cumpla con los siguientes requisitos:

- Desarrollar una interfaz para UNITY que sea capaz de detectar los movimientos de un cubo de Rubik. De forma que esta se pueda usar a gusto del desarrollador y tenga un funcionamiento robusto. Por ello debe poder cumplir los siguientes propósitos:
    - Conectividad con la mayor parte de Smart Cubes del mercado, pero especialmente con el cubo Giiker i3.
    - Ser capaz de registrar y notificar todos los posibles giros realizados en el cubo de Rubik.
    - Tener un funcionamiento robusto, es decir que a pesar de que se pierda la conexión sea capaz de reconectarse.
- Diseñar un accesorio complementario para el cubo de Rubik que se integre perfectamente con la interfaz previamente mencionada y esté especialmente creado para el juego desarrollado en el Trabajo de Fin de Grado de Diseño y Desarrollo de Videojuegos. Este accesorio debe cumplir con las siguientes capacidades y funcionalidades:
    - Debe poder caber en una pieza del cubo de Rubik.
    - El accesorio debe ser capaz de mostrar a través de la matriz de LEDs diferentes estados del juego.
    - Debe dar respuesta háptica de vibración a eventos concretos del juego.
    - Debe ser capaz de interactuar con el juego mediante un interruptor de agitación.

Para la realización de este proyecto se va a utilizar un Smart Cube marca Giiker, al cual se le hará ingeniería inversa para poder acceder a sus funcionalidades, así como un microcontrolador compatible con Arduino el cual irá conexionado a una matriz de LEDs de 8x8, un motor de vibración de 3 voltios y un interruptor de agitación.

Finalmente, espero que mi trabajo respecto con la interfaz de cubo de Rubik en UNITY, sea de utilidad para todos aquellos desarrolladores amantes de los cubos de Rubik que quieran realizar proyectos con ella.

2. Motivación

En la actualidad, hay una amplia variedad de cubos de Rubik inteligentes disponibles en el mercado. Sin embargo, su uso ha sido limitado en gran medida. Aunque los fabricantes ofrecen aplicaciones propias para estos cubos, más allá de eso, hay pocas opciones disponibles. La mayoría de las aplicaciones existentes se centran en proporcionar una mejora en la función de cronómetro, como se puede encontrar en la página Bluetooth Cubing [1].

Sin embargo, pensé que sería interesante explorar otras posibilidades para utilizar un cubo de Rubik como controlador en un juego, sin que necesariamente estuviera relacionado con el mundo de los cubos. Después de investigar, me di cuenta de que no se había desarrollado nada similar en UNITY, lo cual me pareció una gran oportunidad para crear la primera interfaz de cubos inteligentes para UNITY. Esto permitiría a cualquiera crear un juego en el que se utilice un cubo de Rubik como controlador, sin importar su relación con los propios cubos de Rubik.

Esta idea abre un mundo de posibilidades para la interacción con juegos, brindando una experiencia única y original a los jugadores. Al utilizar un cubo de Rubik como controlador, se puede agregar un elemento físico y táctil a la experiencia de juego, lo que podría resultar en una mayor inmersión y diversión para los usuarios. Espero que esta iniciativa pueda inspirar a otros desarrolladores a explorar nuevas formas de utilizar estos cubos inteligentes en el ámbito de los videojuegos.

Además, debido a que estoy estudiando un doble grado en videojuegos y computadores, tengo una perspectiva amplia sobre cómo debe ser un controlador para brindar comodidad y mejorar la inmersión del jugador. Por esta razón, me he propuesto no solo desarrollar una interfaz entre el cubo de Rubik y UNITY, sino también crear un sistema complementario que funcione en conjunto con el cubo. Este sistema permitirá al usuario recibir feedback háptico y visual para enriquecer su experiencia de juego de esta forma se proporcionará una representación visual clara y directa de los eventos en el juego, brindando al jugador una experiencia más inmersiva y envolvente.

3. Estado del Arte

3.1 Introducción

En este apartado se hablará sobre el marco teórico sobre el que se basa este trabajo de fin de grado para llegar hasta el resultado final. Primero, se explicará la historia de los videojuegos desde sus inicios hasta la actualidad. Después se hablará sobre los distintos controladores de los videojuegos a lo largo de la historia además de periféricos que han mejorado la experiencia de juego de títulos concretos.

Después, se hará una breve introducción al mundo de los cubos de Rubik además del mundo del speedcubing.

3.2 Videojuegos

Como explica Mark J P Wolf en su obra [1], los videojuegos abarcan una amplia variedad de perspectivas y funciones en nuestra sociedad actual. Estos pueden considerarse desde una forma lúdica de entretenimiento hasta una experiencia en la que se puede incluso vivir de ello. Los videojuegos se han convertido en una forma de narración, simulación e incluso una manifestación artística.

Los videojuegos tienen un impacto significativo en diversos aspectos de nuestras vidas, como la educación, la interacción social y el tiempo de ocio. Son una herramienta poderosa que puede fusionar distintas disciplinas en un solo concepto, incluyendo el cine, la informática, el arte digital y la música, entre otras.

Además, los videojuegos han demostrado ser una forma efectiva de aprendizaje, ya que permiten a los jugadores sumergirse en entornos interactivos que fomentan la exploración, la resolución de problemas y el pensamiento crítico.

3.2.1.2 La actualidad de los videojuegos

La actualidad está marcada por los avances que podemos ver año tras años, con procesadores y tarjetas gráficas más potentes las cuales permiten la creación de juegos consecuentemente más grandes y con mayor potencia gráfica. Sin embargo, también ha habido arriesgadas donde la potencia del dispositivo no es el fuerte. Un ejemplo de esto es la Nintendo Switch, la cual es una consola portátil por naturaleza pero que sin embargo da la posibilidad de usarse de forma estacionaria en una pantalla. Para poder conseguir esto, Nintendo tuvo que sacrificar la potencia gráfica, pero sin embargo lo supieron compensar con un estilo único y llamativo.

Ilustración 1: Nintendo switch

Otro ejemplo de innovación es la tecnología relacionada con la realidad virtual siendo ahora el auge de esta tecnología. La realidad virtual Permite al usuario experimentar en primera persona situaciones que de ninguna otra forma podría aportando una infinidad de nuevas posibilidades. Un ejemplo son las gafas de realidad virtual Oculus Quest 2, las cuales se pueden ver en Ilustración 2: Oculus Quest 2.

Ilustración 2: Oculus Quest 2

Por otro lado, en los últimos años ha surgido un fenómeno impulsado por la pandemia: los juegos en la nube. Estos juegos permiten a personas que no pueden permitirse comprar juegos acceder a una amplia biblioteca de títulos a través de una suscripción mensual. Mediante la tecnología de streaming, los jugadores pueden disfrutar de los juegos con un rendimiento óptimo, siempre y cuando cuenten con una buena conexión a internet.

3.3 Controles para videojuegos

3.3.1 Historia de los controladores

Es ampliamente conocido que para jugar en una consola o en un PC se requiere un mando o una forma de control. Sin embargo, los diseños ergonómicos y cómodos que encontramos en nuestros días no son producto del azar, sino del resultado de años de desarrollo y evolución. Las distintas empresas del sector han invertido tiempo y recursos en diseñar mandos que se adapten a las necesidades y preferencias de los jugadores. Este enfoque en el diseño de los controladores ha sido clave para mejorar la experiencia de juego y garantizar que los jugadores puedan disfrutar de sus videojuegos de manera más inmersiva y placentera.

A continuación, exploraremos una selección de los mandos más icónicos de la historia de los videojuegos, y destacaremos cómo Nintendo ha sido pionera en la creación de mandos únicos y vanguardistas.

Tabla 1: controladores a lo largo de la historia

Atari 2600: En 1977, Atari desarrolló los mandos CX10 para su consola. Estos mandos, como se muestra en la Ilustración 8: CX10, presentaban un diseño de joystick de dos ejes y un único botón de acción. [2].	 *Ilustración 3: CX10*

NES: El mando desarrollado para la NES es reconocido por su forma de "ladrillo". Cuenta con un “D-pad”, dos botones de juego, un botón de “start “y un botón de “select”. [3]	 *Ilustración 4: Mando NES*
Virtual Boy: Nintendo intentó introducir la realidad virtual con esta consola, pero no tuvo mucho éxito debido a su falta de comodidad y a que no ofrecía una experiencia del todo placentera. Una peculiaridad de este sistema es que el botón de encendido y apagado de la consola se encontraba en el propio mando. [4]	 *Ilustración 5: Virtual Boy*
N64: El primer mando de Nintendo 64 presentaba un diseño único con un agarre central y un joystick analógico en el centro, junto con una cruceta digital y varios botones adicionales. [5]	 *Ilustración 6: Mando N64*
Sega Dreamcast: Este presentaba un diseño guiado por las referencias de la época. Además, Introdujo características innovadoras, como una pantalla LCD en el centro del mando y un puerto de expansión para accesorios. [6]	 *Ilustración 7: Mando Dreamcast*
Gamecube: Tenía un diseño compacto. Destacaba por su disposición de botones y gatillos, así como por la inclusión del innovador botón "Z" en la parte superior del mando. [7]	 *Ilustración 8: Mando Gamecube*

Xbox 360: Presentaba un diseño ergonómico y moderno con dos joysticks analógicos, una cruceta digital y botones de acción. Destacaba por su conectividad inalámbrica y la inclusión de gatillos analógicos sensibles a la presión. [8]	 *Ilustración 9: Mando Xbox 360*
Steam controller: este es un mando creado por Valve para ser usado en Steam, este incluía dos característicos trackpads que permitían un control más preciso y con los cuales se buscaba simular el uso de un ratón. [9]	 *Ilustración 10: Steam controller*
Wii: los Wii remotes presentaban un diseño innovador con una forma similar a un mando de televisión, incluyendo un acelerómetro y un sensor infrarrojo para detectar movimientos y gestos. Revolucionó la forma de jugar al introducir la detección de movimiento. [10]	 *Ilustración 11: Wii remote*
Switch: los Joy-Cons son dos piezas desmontables que se pueden utilizar de forma independiente o acoplarse a una base para formar un controlador tradicional. Los Joy-Con ofrecen una amplia gama de funciones, como sensores de movimiento, vibración HD y la capacidad de usarse como controladores individuales. [11]	 *Ilustración 12: Joy-con*
Oculus Quest 2: los Oculus Touch presentan un diseño ergonómico con un agarre cómodo y controles intuitivos, incluyendo un joystick analógico, botones de acción y gatillos. El Oculus Touch permite una interacción precisa y natural en entornos virtuales. [12]	 *Ilustración 13: Oculus Touch*

3.3.2 Controladores dedicados

Por lo general los mandos se diseñan con un propósito general ya que, por ejemplo, con un mando de última generación como puede ser el de Xbox, se puede jugar al 90% del catálogo de juegos que hay para PC, pero sin embargo hay juegos que se ven muy beneficiados de controladores especializados en dichos juegos, un ejemplo de esto son los volantes para juegos de conducción como el que se puede ver en Ilustración 14: volante CSL DD F1 ESPORTS .

Ilustración 14: volante CSL DD F1 ESPORTS

Este tipo de controladores llevan la experiencia de juego a otro nivel, ya que mandos como el volante presentado en la imagen Ilustración 14: volante CSL DD F1 ESPORTS aporta [13] implementan sensaciones realistas propios de la conducción de un coche formula 1 como puede ser la resistencia del volante, o la funcionalidad de los muchos botones que este incluye.

Por otro lado, y siguiendo con los juegos de simulación, está el Thrustmaster HOTAS Warthog, el cual se puede ver en Ilustración 15: Thrustmaster HOTAS Warthog. Siendo un referente en cuanto a los controladores para simuladores de vuelo. Este está hecho con materiales de alta calidad que buscan recrear los mandos de un avión real. [14]

Ilustración 15: Thrustmaster HOTAS Warthog.

En el otro extremo en cuanto a juegos se refiere, podemos encontrar mandos como la famosa guitarra diseñada para jugar a juegos del estilo guitar Hero. Tal y como explica Edgar Fuentes en [15] las guitarras fueron una moda que explotó en los 2000 con el guitar Hero. Como se puede ver en Ilustración 16: guitarra guitar esta era un controlador con forma de guitarra la cual contaba con 5 botones los cuales eran representativos de las teclas que se deben presionar en el juego.

Ilustración 16: guitarra guitar Hero

Gracias a este controlador, muchas personas se animaron a probar instrumentos reales debido al alto grado de inmersión que esta guitarra creaba sobre los usuarios.

Finalmente, y enlazando con la música, hablaré sobre otro de los más famosos controladores dedicado. Las plataformas de baile como la que se pue de ver en Ilustración 17: plataforma de baile. Estas permiten a parte de controlad el juego, bailar realmente mientras se jugaba, por lo que es un ejemplo donde la simulación del juego y la realidad se juntan

Ilustración 17: plataforma de baile

3.3.2Periféricos en el mundo de los videojuegos

Además de los controladores, a lo largo de la historia de los videojuegos han surgido una amplia variedad de periféricos adicionales que, si bien no eran esenciales, mejoraban la experiencia de juego. Entre estos periféricos se incluyen accesorios como volantes de carreras, pistolas de luz, respuesta háptica, instrumentos musicales y dispositivos de detección de movimiento etc. A continuación, se hablará sobre algunos de los periféricos más curiosos y divertidos que han visto la luz.

- Nintendo Labo es un kit el cual se puede ver en Ilustración 18: Nintendo Labo y se vende junto al juego y unas plantillas de cartón. Cada plantilla permite usar los elementos básicos de la switch de una forma novedosa. Un ejemplo es el modo radio control donde usando la pantalla de la switch se puede mover como si fuera un coche radio control los mandos montados en una de las manualidades de cartón, esto funciona gracias a la vibración

inteligente de los mandos y a un sensor con cámara infrarroja y térmica que lleva incorporado el mando. [16]

Ilustración 18: Nintendo Labo

- **R. O. B**: este era un accesorio de la NES el cual funcionaba por medio de estímulos visuales. Este tenía el control del mando de la NES y en función de la información que captaba por medio de sus sensores visuales, realizaba distintas acciones sobre el mando, Además tenía funciones especiales para el juego "Gyromite" donde podía subir o bajar unas columnas que estaban representadas como botones en su forma física. [17]

Ilustración 19: R. O. B.

- **Pocket camera**: era una cámara que se acoplaba a la Gameboy. Esta podía realizar hasta 30 imágenes monocromáticas de con una resolución de 128x112 píxeles [18]. Además, estas fotos se podían imprimir en papel térmico con otro accesorio de la Gameboy, la Gameboy printer. Como dato curioso, esta cámara apareció en su momento en el libro Guinness de los récords por ser la cámara más pequeña del mundo.

Ilustración 20: Pocket camera

- **Poke Walker**: es un accesorio que se lanzó junto a los juegos Pokémon Oro HeartGold y Plata SoulSilver. Este dispositivo permitía transferir un Pokémon desde la Nintendo DS y cuidarlo a través del Poke Walker. Funcionaba como un podómetro, otorgando objetos que el Pokémon iba encontrando y brindando la oportunidad de combatir contra Pokémon salvajes que aparecían en el dispositivo. [19]

Ilustración 21: Poke Walker

- **Rumble Pak**: Los Rumble Pak eran unos módulos para algunas consolas de Nintendo que permitían añadir vibración a los mandos. Este funcionaba con algunos juegos en situaciones como de disparar un arma o al recibir daño. Este concepto también salió para consolas portátiles de Nintendo como la Nintendo DS [20] o la Gameboy Advance.

Ilustración 22: Rumble Pak

3.4 Cubo de Rubik y el Speedcubing

3.4.1 introducción al cubo de rubik

El cubo de Rubik es un puzle tridimensional creado por Ernő Rubik en 1974 [21]. Consiste en un cubo con 6 caras, cada una de un color distinto: rojo, naranja, verde, azul, blanco y amarillo. Cada uno de estos colores corresponde a una cara del cubo. El cubo de Rubik está compuesto por 6

centros, 8 esquinas y 12 aristas, y cada cara gira alrededor de un centro, siendo estos centros inmutables.

Este diseño permite una amplia variedad de combinaciones y posiciones del cubo, lo que aumenta la dificultad y el desafío al resolverlo. El objetivo del cubo de Rubik es volver a organizar los colores mezclados, de manera que cada cara tenga un único color. Para lograrlo, se deberá realizar una serie de movimientos estratégicos y rotaciones de las diferentes capas del cubo hasta alcanzar la solución deseada.

El cubo de Rubik es un puzle realmente complejo teniendo un total de 43 252 003 274 489 856 000 posibles combinaciones por lo que es muy poco probable que una persona resuelva el mismo cubo 2 veces por azar en su vida.

A lo largo de los años, se han mejorado diferentes métodos de resolución del cubo de Rubik, ya que depender únicamente de la lógica o el azar puede ser demasiado desafiante. Entre los métodos más populares, destaca el método CFOP[1] (también conocido como método Friedrich), nombrado en honor a su creadora, Jessica Friedrich. Este método consta de 4 pasos distintos:

- **Cruz**: En esta etapa, utilizando la lógica, se construye una cruz en una de las caras del cubo. La cruz debe coincidir con los colores de los centros adyacentes.
- **Primeras dos capas (F2L**): En esta fase, también mediante lógica, se resuelven las esquinas y aristas de las caras adyacentes a la cruz previamente formada. Se deben colocar las piezas en su posición correspondiente.
- **Orientación de la última capa (OLL**): En esta etapa, se resuelven las piezas de la última capa del cubo. Se utilizan algoritmos específicos para orientar las piezas correctamente y lograr un patrón determinado en la última cara.
- **Permutación de la última capa (PLL**): Esta es la última fase, donde solo quedan por resolver las piezas de la última capa del cubo. Se utilizan algoritmos específicos para permutar las piezas en su posición correcta, logrando así finalizar con el cubo completamente resuelto.

El método CFOP ha ganado popularidad debido a su eficiencia y rapidez en la resolución del cubo de Rubik. Los entusiastas del cubo de Rubik han practicado y perfeccionado estos algoritmos para lograr tiempos de resolución cada vez más rápidos. Sin embargo, hay otros métodos disponibles, y algunos cuberos incluso desarrollan sus propias técnicas personalizadas

3.4.2 Notación del cubo de Rubik

El cubo de Rubik, debido a su gran popularidad y amplio uso, ha establecido un consenso en cuanto a la notación utilizada para representar los giros y las caras del cubo. Esta notación se basa en dos centros de referencia: uno que representa la cara de enfrente y otro que representa la cara de arriba. La notación estándar utiliza las siguientes letras para representar los giros de cada cara:

3.4.2.1 caras

El estándar es tener como centro en frente la cara verde y como centro arriba la cara blanca.

- **F(Front)**: cara frontal.
- **B(Back)**: cara trasera
- **U(Up)**: cara superior
- **D(Down)**: cara inferior
- **R(Right)**: cara derecha.

[1] Cross, first two layers, orientation of last layer and permutation of last layer

- **L(Left)**: cara izquierda.

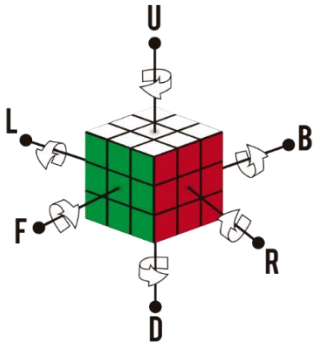


Ilustración 23: caras cubo de Rubik

3.4.2.1 giros

La notación de los giros funciona en función de si el giro es Horario o antihorario, en caso de ser un giro horario, sería tan solo la letra de la cara, pero en caso de ser un giro antihorario llevaría tras la letra una comilla simple.

- R: Giros de la cara derecha

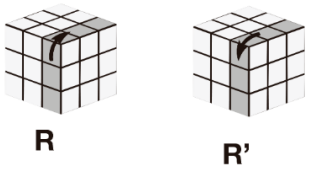


Ilustración 24; giros cara derecha

- L: Giros de la cara izquierda

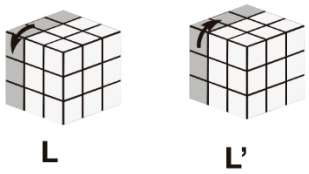


Ilustración 25: giros cara izquierda

- U: Giros de la cara de superior

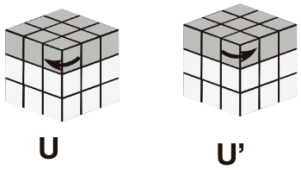


Ilustración 26: Giros de la cara de arriba

- D: Giros de la cara de inferior

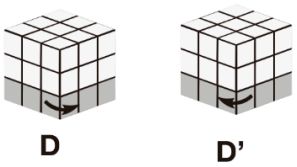


Ilustración 27: giros de la cara inferior

- F: Giros de la cara de frontal

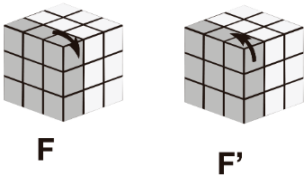


Ilustración 28: giros de la capa de frontal

- B: Giros de la cara de trasera

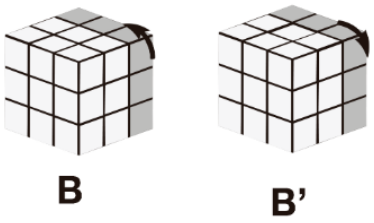


Ilustración 29: Cara trasera

- M: giros de la capa central

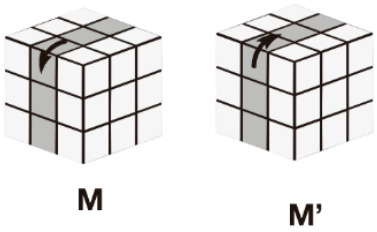


Ilustración 30: giros de la capa central

3.4.2 Speedcubing

El speedcubing es una disciplina en la que el objetivo es resolver un cubo de Rubik lo más rápido posible. Surgió como resultado del impacto que los cubos de Rubik tuvieron en la sociedad desde su lanzamiento. Tanto aficionados como académicos como puede ser Jessica Friedrich [22] se dedicaron a buscar métodos más eficientes que permitieran resolver el cubo en menor tiempo. El speedcubing ha evolucionado en una comunidad global con competiciones, récords mundiales y técnicas avanzadas, donde los participantes buscan constantemente mejorar su velocidad y habilidad en la resolución del cubo de Rubik.

La primera competición oficial de cubo de Rubik fue el campeonato del mundo de 1982 en Budapest, Hungría, en el hogar natal de Erno Rubik [23]. El ganador de esta competición fue Minh Thai estableciendo un récord mundial con un tiempo de 22.95 segundos. Hoy en día, Max Park ostenta el récord del mundo con un tiempo de 3.13 segundos y una media de 5 de 4.48 segundos.

En la actualidad, los campeonatos del cubo de Rubik están regulados por la WCA[2]. Gracias a la existencia de esta asociación, se han podido realizar competiciones oficiales de cubos de Rubik en por todo el mundo. La WCA ha desempeñado un papel fundamental en la expansión del interés y la participación en este singular rompecabezas. A través de sus eventos y estándares establecidos, ha fomentado una comunidad global de speedcubers, promoviendo la competitividad y el intercambio de conocimientos y habilidades en torno al cubo de Rubik. Como resultado, el cubo de Rubik ha ganado popularidad y seguidores apasionados en todo el mundo.

Además del icónico cubo de Rubik, las competiciones de speedcubing abarcan un amplio espectro de retos estableciendo un total de 17 categorías oficiales [24], entre las que se incluye resolver el cubo a una mano, resolverlo a ciegas, resolver múltiples cubos a ciegas, enfrentarse a cubos de mayor tamaño como el 4x4,5x5,6x6 o 7x7, así como otros rompecabezas como el clock y el skewb. Estas diversas categorías permiten a los competidores demostrar su habilidad y destreza en una variedad de desafíos, enriqueciendo aún más el apasionante mundo del speedcubing.

Gracias al fenómeno del speedcubing, ahora contamos con cubos de Rubik más rápidos y precisos. Todo esto ha sido posible gracias a la activa comunidad que ha perdurado a lo largo de los años, desde la invención del cubo de Rubik hasta el presente. Esta comunidad ha impulsado el desarrollo de cubos inteligentes.

Debido a mi pasión por los videojuegos y por el speedcubing, pensé que sería una maravillosa idea aprovechar mis conocimientos sobre ambos mundos para unirlos creando el primer videojuego independiente controlado con un cubo de Rubik.

[2] World Cube Association

4. Metodología

En este apartado se presentará la planificación del proyecto, dividiendo las tareas en función de su prioridad y la disponibilidad del trabajo de fin de grado de diseño y desarrollo de videojuegos. Estos dos proyectos se complementan mutuamente, por lo que fue necesario llevarlos a cabo de manera simultánea y coordinada.

4.1 Distribución de tareas

Las tareas en las que se ha separado este proyecto son las siguientes:

- **Interfaz Cubo de Rubik con UNITY:**
  - **Ingeniería invertida**: en esta tarea se hicieron pruebas en crudo con el cubo de Rubik inteligente para de esta forma comprobar cuál era la información saliente del cubo realmente importante y si era posible este proyecto.
  - **Diseño de la interfaz de conexión**: en esta tarea se pensó un esquema de las formas de comunicar el cubo con UNITY.
  - **Integración en UNITY**: en esta tarea se desarrolló la parte correspondiente con UNITY para poder usar el cubo como controlador.
  - **Prueba del correcto funcionamiento en el juego**: Esta tarea se realizó cuando el proyecto del juego estuvo más avanzado y esta consistía en probar el control del personaje con el cubo de Rubik, así como descubrir si era cómodo de usar.
- **Periférico que complementa la interfaz de cubo de Rubik.**
  - **Hardware**:
    - **Lluvia de ideas y comparativa de componentes**: en esta tarea se pensaron las diferentes posibilidades respecto a la integración de un circuito que permitiera interactuar con el sistema además de una comparativa de los diferentes componentes que podrían ser útiles.
    - **Diseño del circuito**: en esta tarea se seleccionaron los componentes finales y se hizo un esquema con las conexiones entre ellos.
    - **Prueba de cada componente**: en esta tarea se probaron los diferentes componentes para comprobar su funcionamiento.
    - **Soldadura y montaje del circuito**: en esta tarea se cerró el circuito soldándolo y fijando todas las piezas en su disposición final.
  - **Software**:
    - **Diseño de la interfaz entre UNITY y Arduino**: en esta tarea se diseñó la forma en la que se comunicarían todos los elementos del sistema.
    - **Desarrollo de la interfaz entre UNITY y Arduino**: en esta tarea se desarrolló el diseño anteriormente mencionado.
    - **Pruebas del circuito aislado**: en esta tarea se probó el sistema final sin entrar en la parte de implementación de UNITY, para comprobar el correcto funcionamiento del código de Arduino.
    - **Pruebas del circuito en el juego final**: en esta tarea se probó la integración en UNITY, así como su correcto funcionamiento en el juego.
- **Redacción de la memoria**: en esta tarea se redactó la memoria de todas las tareas previamente realizadas.

5. Descripción informática

5.1 Esquema general

En los siguientes apartados se explicará el funcionamiento y todas las formas en las que este proyecto se comunica. Como se puede ver en la Ilustración 31: esquema general este proyecto consta de 2 partes que comparten similitudes ya que se sustentan sobre una base común la cual es la comunicación BluetoothLE. Por un lado, está el Smart cube, el cual solo emite señales las cuales son recibidas por un proceso intermedio el cual se encarga de establecer la conexión con el cubo y captar las señales además de traducirlas de forma que UNITY las pueda tratar de forma sencilla. Por otro lado, está el circuito de Arduino, el cual establece comunicación de la misma forma que el cubo, pero con la diferencia de que no solo se recibirán señales provenientes del Arduino, sino que también se mandarán señales para que el circuito actúe en consecuencia.

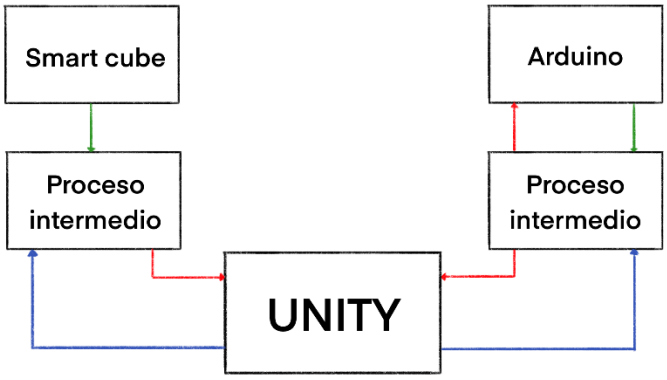


Ilustración 31: esquema general

5.2 Especificaciones Cubo de Rubik Bluetooth

5.2.1 Introducción a Cubos Inteligentes

Actualmente existen muchas opciones de cubos inteligentes de múltiples marcas. Los cuales a lo largo de los años han ido mejorando su sistema de tracking. A continuación, enumeraré algunas de las opciones más destacadas hoy en día.

- **Giiker i3 y i3s**: la versión i3 fue el primer cubo inteligente que se lanzó al mercado a un precio de alrededor de unos 30 euros. Tiene ligeros fallos en cuanto al sistema de tracking ya que algunos movimientos no se detectaban, pero este problema se intentó arreglar en la versión i3 el cual se puede ver en Ilustración 32: cubo 3x3 giiker.

Ilustración 32: cubo 3x3 giiker

- **MosnterGo AI 3x3**: este es un cubo inteligente el cual se puede ver en la diseñado para niños y principiantes por lo que cuanta con un sistema robusto además de batería extraíble. [25]Su precio es de 25,99 euros, este se puede ver en Ilustración 33: MonsterGo Ai.

Ilustración 33: MonsterGo Ai

- **Gan 356 i Carry:** Este es el cubo inteligente económico de la marca GAN el cual se puede ver en Ilustración 34: Gan 356 i carry. Está pensado para la práctica del speedcubing [26]. Su precio es de 34,99 euros.

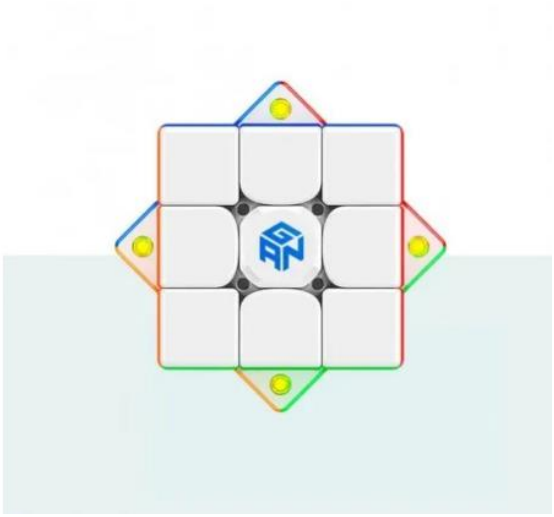


Ilustración 34: Gan 356 i carry

- **Gan i3 3x3**: Este es el buque insignia de la marca GAN este además de sus funcionalidades como cubo inteligente permite ajustes en el propio cubo para que se ajuste al usuario. [27] Su precio es de 59,90 euros, tal y como se puede ver en Ilustración 35: GAN I3 3x3.

Ilustración 35: GAN I3 3x3

- **Rubik's Connected 3x3**: Esta es la opción de cubos inteligentes que ofrece la marca Rubik's como se puede ver en Ilustración 36: cubo inteligente Rubiks es decir la marca original de cubos de Rubik. [28] Su precio es de 79, 90 euros.

Ilustración 36: cubo inteligente Rubiks

- **GoCube Edge 3x3**: Esta es posiblemente una de las opciones más sofisticadas en el mercado, la cual se puede ver en **Error! Reference source not found.**, ya que además del sistema de tracking y bluetooth incorpora indicadores luminosos y se carga directamente mediante un cable USB [29]. Su precio es de 96,80 euros.

Ilustración 37: Go cube

- **Moyu Weilong Ai Smart Cube**: Esta es la versión implementada por la marca MOYU, la cual se puede ver en la Ilustración 38: cubo inteligente de , estás pensada para ser usada para el speedcubing [30]. Su precio es de 65,20 euros.

Ilustración 38: cubo inteligente de Moyu

5.2.2 Justificación para la elección del cubo.

El cubo de Rubik elegido entre todos los disponibles en el mercado es el Xiaomi Giiker Super Cubei3. Este cubo se lanzó al mercado en 2018 y fue catalogado como pone en la página oficial de Giiker [31] como el primer cubo inteligente del mundo. La razón por la que se eligió este y no otro más moderno fue porque ya disponía de este cubo y las otras opciones resultaban económicamente costosas.

Como se puede ver en la Ilustración 39: especificaciones Giiker i3s el cubo cuenta con unas dimensiones de 56.5 milímetros, una batería de litio de 3.7v y un peso de 102g. Además, un dato importante es que usa la tecnología Bluetooth 4.0 ya que esto supondrá una limitación para ordenadores antiguos. [32]

Model	SUPERCUBE i3S	Cube Dimension	56.5mm
Battery	3.7V lithium battery	Cube Weight	102g
Charging Time	90 min	Working Time	90 days*
Ages	6 years+	Material	ABS
Package contents	1 SUPERCUBE i3S, 1 charger, 1 charging cable, 1 user manual		

Ilustración 39: especificaciones Giiker i3s

5.3 Conexión con el cubo

Para la realizar la conexión entre el programa y el cubo se ha optado por Bluetooth LE e implementando un cliente GATT [33] debido a que es un protocolo que encaja perfectamente con las necesidades de este proyecto y porque está muy bien documentado por Microsoft [34].

5.3.1 Introducción y conceptos básicos sobre Bluetooth LE

Bluetooth LE [34] (Low Energy) es una especificación que define protocolos para el descubrimiento y comunicación para dispositivos de baja energía.

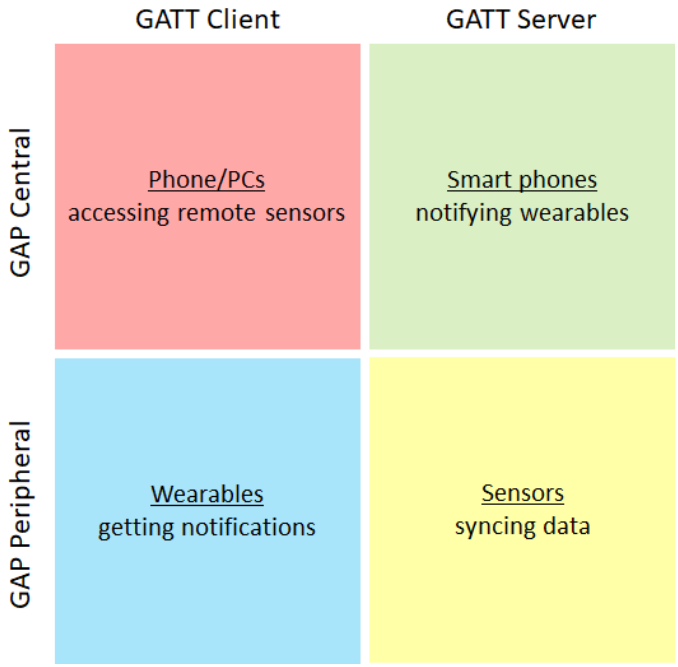


Ilustración 40: Diferencias entre los dispositivos GAP y GATT

En primer lugar, se realiza el descubrimiento un protocolo GAP (Generic Access Profile) y finalmente la comunicación se establece mediante un protocolo GATT (Generic Attribute).

Centrales y periféricos

Tal y como podemos ver en Ilustración 40: Diferencias entre los dispositivos GAP y GATT, existen dos roles a la hora de descubrir dispositivos. Por un lado, están las centrales, las cuales se encargan de buscar periféricos y por otro lado están los periféricos los cuales están a la espera de ser descubiertos por una central. Como posteriormente veremos en función de las necesidades la central se encargará de solo recibir o de también notificar al periférico.

Cliente y servidor

Una vez establecido la conexión se pueden diferenciar los dispositivos en cliente y servidor. El servidor es aquel que tiene la información y el cliente es aquel que recibe la información por lo tanto en el contexto de este trabajo, el ejecutable que está siendo ejecutado en el sistema operativo Microsoft Windows sería el cliente ya que es el que recibe y trabaja con la información y el cubo sería el servidor ya que es el que tiene y envía los datos.

Atributos

Los protocolos GATT definen bloques de información que especifican el modo en el que se comunican la central y el periférico. En definitiva, los atributos [35] son la base de los protocolos

GATT. Los atributos, conceptualmente, se encuentran en el servidor, y son accedidos por el cliente. Los atributos contienen tanto información estática de naturaleza invariable como puede ser el nombre del dispositivo, número de serie… e información cambiante proveniente de los sensores.

Toda esta información se encuentra en los diferentes apartados que componen un atributo.

- **Identificador(handle):** es un identificador único de 16 bits que diferencian los atributos dentro de un cliente GATT. Este se garantiza que sea inmutable ya que se usa para identificar un atributo en concreto. El Rango de posibles identificadores va desde el 0x0001 hasta el 0xFFFF siendo el 0x0000 el valor para identificador inválido.
- **Tipo:** este apartado corresponde con un UUID[3] que identifica el tipo de información que almacena el valor del atributo. Por lo general el tipo suele corresponder o bien con un servicio, una característica o un descriptor.
- **Permisos:** son metadatos[4] que especifican los permisos de acceso (ninguno, leer, escribir y leer y escribir), el nivel de encriptado y si requiere autorización.
- **Valor:** el campo de valor contiene la información como tal del atributo sin ningún tipo de restricción del tipo de dato que contiene por lo que este se debe interpretar por si especificación. Aunque no se especifica el tipo de dato, si que se conoce el máximo de información que contiene el cual es 512 bytes.
- **Numero de bytes:** informa sobre cuántos son los bytes del valor que contienen información.

Jerarquía de datos de un atributo

Los atributos se agrupan en servicios, los cuales están compuestos de cero o más características y estas características pueden contener 0 o más descriptores como se explica en [35].

- **Servicios**: corresponde con un atributo dentro de un cliente GATT o por verlo de otra forma los atributos son una lista de servicios.
- **Características**: son contenedores de información. Cada característica viene siempre como un par donde el primer campo es la declaración o identificador de la característica el cual además puede contener descriptores y el segundo es el propio valor del atributo.
- **Descriptores**: Estos aportan metadatos para ampliar la información sobre una característica. Estos se encuentran en la definición de una característica

5.3.2 Justificación de elección del lenguaje y el entorno de programación

Para la realización de este proyecto se ha utilizado C# con un proyecto .NET ya que cuenta con una amplia documentación respecto a la implementación y el uso de bluetooth LE.

Existen otras alternativas donde se podría haber realizado esta tarea como pueden ser Python o C++ sin embargo debido a que bluetooth LE está muy bien documentado por Microsoft y también debido a que UNITY usa C# como lenguaje principal se ha decidido usar este lenguaje para realizar el ejecutable que conecta el cubo con el proyecto de UNITY.

[3] Universal unique identifier: es un número de 16 bytes que garantiza que es único.

[4] Los metadatos (del griego μετα, meta, 'después de, más allá de'1 y latín datum, 'lo que se da', «dato»2), literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos que describen el contenido informativo de un objeto al que se denomina recurso. [9]

En cuanto al entorno de programación, he elegido RIDER por encima de Visual Studio ya que como usuario me siento más cómodo usando este entorno.

5.3.3 Estructura del código

La mayor parte del código se encuentra en el programa principal, sin embargo, ha sido necesario el uso de clases adicionales para modularizar el código y que sea más fácil de entender.

- **Programa principal**: es el núcleo del proyecto, su función principal es la de realizar la conexión bluetooth con el cubo y recibir información una vez conectado a este.
- **Clase Cube Tracker**: su función es la de interpretar los datos que recibe del programa y convertirlos a una cadena de caracteres más sencilla de entender.
- **Clase Devices List**: Es una estructura de datos en forma de lista que se encarga de mantener la información sobre los dispositivos actuales. También integra funciones que facilitan la adición y obtención de información para evitar posibles errores.

5.3.4 Fases y funcionamiento del proyecto

El objetivo de esta sección era conseguir un ejecutable en Windows con el que poder recibir señales del cubo y posteriormente interpretar esas señales.

Búsqueda de dispositivos cercanos

En esta fase se usa la clase DeviceWatcher la cual permite crear acciones para los diferentes eventos que se pueden dar a la hora de descubrir dispositivos. Tal y como se ve en el código presente en Ilustración 41: Código búsqueda de dispositivo, hay una serie de eventos que nos ofrece la clase DeviceWatcher. Es especialmente interesante el evento Added, el cual se ejecuta cuando se añade un dispositivo, ya que gracias a este podremos mantener una estructura con los dispositivos disponibles y que como se explicará más tarde permite la conexión con el dispositivo.

Una vez iniciada la búsqueda, los métodos DeviceWatcher_Added y DeviceWatcher_Removed, se encargan de añadir y quitar los dispositivos según se descubren. Esta lista contiene la información del dispositivo cuyos atributos más importantes son el ID y el nombre ya que con estos se podrá identificar los dispositivos en la selección.

```
// Query for extra properties you want returned
string[] requestedProperties = { "System.Devices.Aep.DeviceAddress", "System.Devices.Aep.IsConnected" };

DeviceWatcher deviceWatcher =
    DeviceInformation.CreateWatcher(
        aqsFilter: BluetoothLEDevice.GetDeviceSelectorFromPairingState(false),
        requestedProperties,
        DeviceInformationKind.AssociationEndpoint);

// Register event handlers before starting the watcher.
// Added, Updated and Removed are required to get all nearby devices
deviceWatcher.Added += DeviceWatcher_Added;
deviceWatcher.Updated += DeviceWatcher_Updated;
deviceWatcher.Removed += DeviceWatcher_Removed;

// EnumerationCompleted and Stopped are optional to implement.
deviceWatcher.EnumerationCompleted += DeviceWatcher_EnumerationCompleted;
deviceWatcher.Stopped += DeviceWatcher_Stopped;

// Start the watcher.
deviceWatcher.Start();
```

Ilustración 41: Código búsqueda de dispositivo

Selección del dispositivo

En esta fase el programa se quedará en bucle hasta que se haya encontrado y seleccionado un dispositivo válido con el cual realizar la conexión.

Para seleccionar un dispositivo, tal y como se puede ver en el fragmento de código de Ilustración 42: método ChooseDevice se muestra por la salida estándar el número de dispositivos disponibles y la lista anteriormente mencionada con todos los nombres de los dispositivos disponibles de esta forma desde se facilita al ejecutable de UNITY la información necesaria para mostrar al usuario una lista de dispositivos que facilite su selección. Posteriormente, se quedará a la espera de recibir el nombre de un dispositivo válido. Si recibiera un nombre incorrecto, este al no encontrarse en la lista de dispositivos recibiría un valor nulo, mostrará un mensaje que dice “Device not found” es decir que el dispositivo no se encontró y por lo tanto no se iniciaría la conexión.

```
private static void ChooseDevice()
{
    string[] devices = _devicesList.GetDevices();
    //Send the number of devices
    Console.WriteLine(devices.Length);
    for (int i = 0; i < devices.Length; i++)
        Console.WriteLine(devices[i]);

    device = _devicesList.GetDevice(name: Console.ReadLine());
    if (device==null)Console.WriteLine("Device not found");
}
```

Ilustración 42: método ChooseDevice

Conexión con el dispositivo

Una vez seleccionado el dispositivo se realiza la conexión y se espera a recibir el resultado de esta conexión. Como se puede ver en Ilustración 43: código conexión con el dispositivo, En caso de ser exitosa la conexión se procederá a descubrir el servicio deseado y en caso de fracaso, terminará la ejecución del programa. Será UNITY el encargado de cuando esto último pase, volver a ejecutar el programa.

```
BluetoothLEDevice bluetoothLeDevice = await BluetoothLEDevice.FromIdAsync(device.Id);
Console.WriteLine("Attempting to pair with the cube");
GattDeviceServicesResult result = await bluetoothLeDevice.GetGattServicesAsync();

if (result.Status == GattCommunicationStatus.Success){...}

if (connectionSuccess)
{
    Console.WriteLine("connection successful");
    Console.WriteLine("Press ENTER to exit.");
    Console.ReadLine();
}
else
    Console.WriteLine("connection failed");

break;
```

Ilustración 43: código conexión con el dispositivo

Descubrimiento del servicio

Como se ha explicado en la introducción de este capítulo, los servidores GATT poseen atributos o servicios, en esta fase se ha de encontrar aquel servicio que se encarga de notificar los giros de las caras del cubo.

En la documentación de Bluetooth [36] vienen recogidos los UUIDs de los servicios más comunes, sin embargo, no viene recogido el servicio correspondiente a los movimientos del cubo. El siguiente fragmento de código muestra todos aquellos servicios que ofrece el cubo, gracias a esta información se pudo descubrir algunos de los servicios que si aparecen en la lista de UUIDs [36].

```
if (result.Status == GattCommunicationStatus.Success)
{
    connectionSuccess = true;
    Console.WriteLine("pairing succeded");
    var services = result.Services;
    foreach (var service in services)
    {
        Console.WriteLine(service.Uuid);
```

Ilustración 44: código de descubrimiento de servicio

El resultado de ejecutar el fragmento presente en Ilustración 45: servicios cubo de Rubik es el siguiente:

```
00001800-0000-1000-8000-00805f9b34fb
00001801-0000-1000-8000-00805f9b34fb
0000180a-0000-1000-8000-00805f9b34fb
0000180f-0000-1000-8000-00805f9b34fb
0000aadb-0000-1000-8000-00805f9b34fb
```

Ilustración 45: servicios cubo de Rubik

Por lo tanto, se pudo deducir que aquellos servicios que no estaban en la documentación de Bluetooth eran potenciales candidatos a ser el servicio encargado de los movimientos del cubo.

Mediante prueba y error, haciendo la suscripción a mano de cada servicio desconocido, se descubrió que la información recibida se actualizaba cada vez que giraba una cara cuando estaba suscrito al servicio **0000aadb**.

Por lo tanto, la tabla de servicios que ofrece el dispositivo es la siguiente (Tabla 1).

UUID	Servicio
00001800-0000-1000-8000-00805f9b34fb	Generic Access
00001801-0000-1000-8000-00805f9b34fb	Generic Attribute
0000180ª-0000-1000-8000-00805f9b34fb	Información del dispositivo
0000180f-0000-1000-8000-00805f9b34fb	Batería
0000aadb-0000-1000-8000-00805f9b34fb	Rotaciones del cubo
0000aaaa-0000-1000-8000-00805f9b34fb	Desconocido
00001530-1212-efde-1523-785feabcd123	Desconocido

Tabla 2: servicios cubo de Rubik

Suscripción al servicio

Una vez encontrado el servicio que corresponde a los giros del cubo, se realiza una suscripción a este de forma que cada vez que cambie el valor nos lo comunique. Para ello se comprueba en las características si el servicio puede notificar cambios. Como se puede ver en el código, una vez encontrada la característica que permite notificar cambios y la comunicación se realizó con éxito, se le asignara a la acción de cambio de valor, el método Characteristics_ValueChanged, el cual se ejecutará cada vez que el servidor quiera notificar un cambio al cliente. Será en este método donde se obtengan los datos.

```
var characteristics :IReadOnlyList<GattCharacteristic> = characteristicsResult.Characteristics;
foreach (var characteristic in characteristics)
{
    Console.WriteLine("-----------------");
    Console.WriteLine(characteristic);
    GattCharacteristicProperties properties =
        characteristic.CharacteristicProperties;

    if (properties.HasFlag(GattCharacteristicProperties.Notify))
    {
        Console.WriteLine("notify property found");
        GattCommunicationStatus status =
            await characteristic // GattCharacteristic
                .WriteClientCharacteristicConfigurationDescriptorAsync(
                    GattClientCharacteristicConfigurationDescriptorValue
                        .Notify); // IAsyncOperation<GattCommunicationStatus>
        if (status == GattCommunicationStatus.Success)
            characteristic.ValueChanged += Characteristic_ValueChanged;
```

Ilustración 46: código suscripción a un servicio

Obtención de datos

Finalmente, solo queda recibir e interpretar los datos provenientes del cubo. Esta fue la fase más compleja de este apartado debido a la poca documentación de este servicio, sin embargo, gracias a un proyecto de Jeremy Fleischman [37] y diversos foros [38] se pudo deducir la información.

Los datos en bruto vienen en un paquete de 20 bytes como se puede ver en Ilustración 47: Datos en bruto obtenidos de las rotaciones del cubo..

```
10011-1010010-10000111-1000110-110001-10011-10011-100011-110101-11000010-10100110-10001001-1000111-10110001-101010-10000-10001-10011-100001-100011-
F
00010001
```

Ilustración 47: Datos en bruto obtenidos de las rotaciones del cubo.

Sin embargo, la información interesante se encuentra en el byte 16, cuyos primeros 4 bits indican el sentido y los últimos 4 bits, la cara que se rotó. Como se ve en el siguiente fragmento de código, la información del byte 16 se traduce a base binaria para facilitar su interpretación. Adicionalmente, se tiene que hacer un ajuste al valor resultante en binario ya que suprime los ceros previos al uno de más valor, por lo que se debe rellenar a mano hasta contar con 8 dígitos.

```
private static void Characteristic_ValueChanged(GattCharacteristic sender, GattValueChangedEventArgs args)
{
    var reader = DataReader.FromBuffer(args.CharacteristicValue);
    byte[] values = new Byte[20];
    reader.ReadBytes(values);
    string bin = Convert.ToString(values[16], toBase: 2);

    string aux = "";
    for (int j = 0; j < 8 - bin.Length; j++)
        aux += "0";

    aux += bin;
    bin = aux;

    Console.Out.WriteLine(_cubeTracker.ReadFaces(bin));
}
```

Ilustración 48: método Characteristic_ValueChanged

Finalmente, se cuenta con la clase CubeTracker, la cual se encarga de traducir la información en binario a notación del cubo de la WCA[5].

Haciendo pruebas se pudo deducir que la siguiente información recibida correspondía con los siguientes giros del cubo (Tablas 2 y 3):

Valor de entrada	Cara
0101	R
0011	L
0001	B
0110	F
0010	D
0100	U

Tabla 3: conversión entre valor y cara

Valor de entrada	Dirección
0001	Horario
0011	Antihorario

[5] Notación oficial que corresponde con los giros de cada cara del cubo en sentido horario o antihorario.

Tabla 4: conversión del sentido del giro

5.4 UNITY

5.4.1 Introducción a UNITY

5.4.1.1 ¿Qué es UNITY?

UNITY es un motor de videojuegos gratuito multiplataforma comúnmente usado en el desarrollo de videojuegos. Este ofrece un entorno 3D y 2D donde el usuario puede crear proyectos interactivos para una gran variedad de plataformas tales como Nintendo switch, PlayStation 4, Windows, Mac OS… [39]

En los últimos años UNITY ha crecido en popularidad bastante sobre todo entre los estudios independientes tal y como se muestra en Ilustración 49: gráfica de motores usados por empresas indies españolas , el 74% de los videojuegos creados por empresas españolas fue en UNITY [40]. Es por ello por lo que se ha preferido en este proyecto usar como herramienta UNITY por encima de otras alternativas como GODOT o UNREAL ENGINE, debido a que además de contar con una extensa documentación, cuenta con una gran comunidad con la que se podría compartir este proyecto y darle la oportunidad a otro desarrollador de crear su propio proyecto apoyado en este.

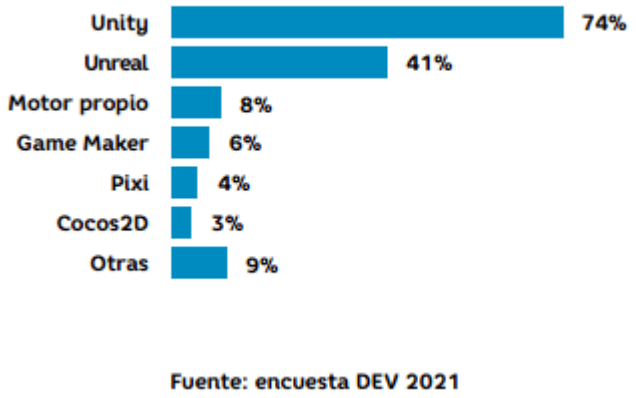


Ilustración 49: gráfica de motores usados por empresas indies españolas (libro blando de los videojuegos)

Algunas de las características principales de UNITY son las siguientes:

- Su motor gráfico es OpenGL.
- Usa como lenguaje de scripting principal C#.
- Cuenta con una interfaz gráfica muy accesible tanto para artistas como para programadores.

5.4.1.2 Historia de UNITY

Tal y como se cuenta en el artículo escrito por Dani Candil en Vida extra [41], UNITY surgió en 2004 cuando David Helgason, Nicholas Francis y Joachim Ante, tras el fracaso de su juego GooBall vieron el potencial del motor que crearon ya que desarrollaron herramientas muy potentes las cuales quisieron ponerlas a disposición del “pueblo”.

Su idea era crear un motor de videojuegos que pudieran usar tanto grandes como pequeñas empresas y que fuera amigable para los diferentes partes del desarrollo, es decir, artistas, programadores, game designers…

En 2005 se presentó la primera versión de UNITY en la conferencia mundial de Desarrolladores de APPLE, esta era exclusiva para MAC OS [41].

En 2008 se lanzó la versión 2.0 con la cual dio soporte a iOS siendo de los primeros motores en ofrecer este tipo de herramientas para esta plataforma. Y en 2010 con la versión 3.0 se da soporte al sistema operativo Android [41].

En 2014 con la versión 4.6 se incluyó un sistema de generación de interfaces de usuario haciendo que los desarrolladores no dependieran de herramientas externas para crear estas.

En 2017 UNITY cambió la forma de nombrar las versiones pasando a ser una nomenclatura basada en el año de la versión y la versión 2017.x. Además, por primera vez lanzarían una versión llamada LTS o Long Time Support la cual garantiza que durante 2 años tendría corrección de errores y que garantizaba que era un software estable [41].

5.4.2 Clases principales del proyecto

A continuación, se describen las clases principales que sustentan el proyecto.

- **RunProcess**: Clase encargada de iniciar el proceso que comunica la computadora con el cubo mediante Bluetooth LE. Además, se encarga de capturar los mensajes provenientes de dicho proceso y facilitar un mecanismo al resto de clases para comunicarse con este.
- **Move**: Clase que forma un objeto Move, Este objeto representa un mensaje proveniente del proceso, si tras tratar ese mensaje se valida como un movimiento realizado por el cubo, entonces se rellenan el resto de los campos necesarios para ser considerado un movimiento, es decir, la cara que se ha movido, su dirección, el color de la cara y el momento en el que se realizó.
- **MovesQueue**: Clase que implementa una cola de Movimientos y proporciona mecanismos de seguridad para su uso. Esta clase sirve tanto para almacenar los mensajes en bruto recibidos por el proceso como para almacenar los movimientos ya validados.
- **CubeConectionManager**: Clase que se encarga de ofrecer mecanismo para mantener la conexión con el proceso, así como facilitar una lista de dispositivos disponibles, selección del dispositivo deseado y restablecimiento de la comunicación.
- **CubeInputs**: Clase que se encarga de interpretar los mensajes provenientes del proceso, es decir una vez seleccionado un dispositivo, esta clase clasifica los mensajes provenientes y tras validarlos los convierte en movimientos.

5.4.3 Conexión con el ejecutable

Lo primero de todo es realizar una conexión entre el ejecutable que se encarga de la conexión bluetooth y UNITY. Esto supuso un reto debido a las múltiples formas en las que se podía realizar. La primera opción en la que se pensó fue en el uso de pipes para comunicar ambos procesos, pero esta opción requería una comunicación demasiado compleja para las necesidades además de frágil. La siguiente opción que se pensó fue que fuera el propio UNITY el que lanzara el proceso que se comunica con el cubo y redirigir la entrada y la salida estándar. Esta última opción fue la sin ninguna duda la mejor ya que es simple y no requiere de un mecanismo complejo.

Como se ve en Ilustración 50: fragmento de código, redirección de entrada y salida estándar, se redirige la entrada y la salida estándar a un método encargado de tratar los mensajes entrantes del proceso.

```
public void StartProcess()
{
    if (process != null && !process.HasExited)
        process.Kill();

    try
    {
        process = new Process();
        process.EnableRaisingEvents = false;
        process.StartInfo.FileName =
            Application.dataPath + "/Executable/BluetoothCubo.exe";
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        process.StartInfo.RedirectStandardOutput = true;
        process.StartInfo.RedirectStandardInput = true;
        process.StartInfo.RedirectStandardError = true;
        process.OutputDataReceived += new DataReceivedEventHandler(DataReceived);
        process.ErrorDataReceived += new DataReceivedEventHandler(ErrorReceived);
        process.Start();
        process.BeginOutputReadLine();
        messageStream = process.StandardInput;

        UnityEngine.Debug.Log( message: "Successfully launched app");
    }
    catch (Exception e)
    {
        UnityEngine.Debug.LogError( message: "Unable to launch app: " + e.Message);
    }
}
```

Ilustración 50: fragmento de código, redirección de entrada y salida estándar

El código de Ilustración 50: fragmento de código, redirección de entrada y salida estándar corresponde con el método StartProcess de la clase Runprocess. Esta clase es la encargada de manejar de primera mano todo lo relacionado con el proceso, es decir, terminarlo, iniciarlo, configurarlo, y recibir y mandar información.

El método SendMessageProcess sirve como medio para el resto de las clases para facilitar la comunicación y así poder mandar cadenas de caracteres como entrada al proceso. Por otro lado, está la clase DataReceived la cual se encarga de gestionar la salida del proceso.

Una vez iniciado el proceso, la salida estándar, a través de la clase anteriormente mencionada, DataRecevied, muestra por la consola de UNITY los mensajes recibidos y además se encolan las cadenas de caracteres en una instancia de la clase MovesQueue. De forma que se puede tener un control total sobre la información de proveniente del proceso. Además, esta clase se encarga de terminar el proceso cuando el ejecutable de UNITY se cierra.

Por otro lado, tenemos la clase CubeConectionManager la cual se encarga de leer los primeros mensajes del proceso que como se ve en el siguiente fragmento de código del método Get devices. Estos primeros mensajes recibidas corresponden al número de dispositivos disponibles y el nombre de dichos dispositivos los cuales se añaden a las opciones de un “Dropdown”. Esto permite crear un mecanismo de selección de dispositivo para el usuario. Y quedará a la espera de que el usuario seleccione un dispositivo de la lista de opciones. Adicionalmente en este punto de la ejecución del código es posible que se haya intentado conectar a un dispositivo el cual el proceso ya no tiene en su lista de dispositivos, en ese caso este método servirá como forma de actualizar la lista de dispositivos.

La conexión consiste en que mediante el uso del método ConnectButton que se ve en Ilustración 51:método GetDevices se encarga de escribir por la entrada estándar del proceso el nombre del dispositivo seleccionado.

Después será el proceso el que internamente se intentará conectar con el cubo como se explicó en la sección 5.3.4.

```
IEnumerator GetDevices()
{
    _dropdown.options.Clear();
    _cubeInputs.isActive = false;
    yield return new WaitForSeconds(1);

    Move msg = _movesQueue.Dequeue();
    if (msg.msg == "Device not found")
        msg = _movesQueue.Dequeue();

    int count = int.Parse(msg.msg); //the first message will always be the number of available devices to connect
    for (int i = 0; i < count; i++)
        _dropdown.options.Add( item: new TMP_Dropdown.OptionData(_movesQueue.Dequeue().msg));


    _cubeInputs.gameObject.SetActive(true);
    _cubeInputs.isActive = true;
    _refreshButton.interactable = true;
    _connectButton.interactable = true;
}
```

Ilustración 51:método GetDevices

```
public void ConnectButton()
{
    currentDevice = _dropdown.options[_dropdown.value].text;
    _process.SendMessageProcess(currentDevice);
    _connectButton.interactable = false;
    _refreshButton.interactable = false;
}
```

Ilustración 52: método ConnectButton

5.4.3.1 Interfaz de conexión

De forma gráfica, se otorga al usuario ciertas herramientas que le permiten establecer la conexión.

- **Dropdown**: es un desplegable que almacena los dispositivos disponibles para realizar la conexión.
- **Botón de Refresh**: al interactuar con este botón, se actualizará la lista de dispositivos disponibles. A nivel de implementación, lo que hace es mandar al proceso un nombre de dispositivo que es muy improbable que esté en la lista de dispositivos, entonces, el proceso al no encontrar este dispositivo volverá a mandar una nueva lista de dispositivos.
- **Botón Connect**: al pulsar este botón, se realizará la conexión con el dispositivo seleccionado, o en caso de no haber seleccionado ninguno el primer dispositivo de la lista.
- **Botón Cancel**: al pulsar este botón, se cancelará la conexión con el dispositivo seleccionado. A nivel de implementación, se encarga de poner a nulo el valor del string que almacena el dispositivo al que se está intentando conectar en ese instante por lo tanto se corta el bucle que se encarga de intentar realizar la conexión hasta que esta realice. Y se queda a la espera de que el usuario vuelva a pulsar el botón Connect.
- **Botón Continue**: Este permanecerá deshabilitado hasta que la conexión con el dispositivo se haya establecido. Una vez esto pase, el botón se habilitará y al ser pulsado dirigirá al usuario a la siguiente pantalla.

En función de en qué fase de la conexión este la ejecución, los componentes de la interfaz que se acaban de mencionar estarán activados o desactivado.

Estado inicial es el que se ve en Ilustración 53: estado inicial de la interfaz de conexión -, todos los componentes menos el botón cancel están interactuarles.

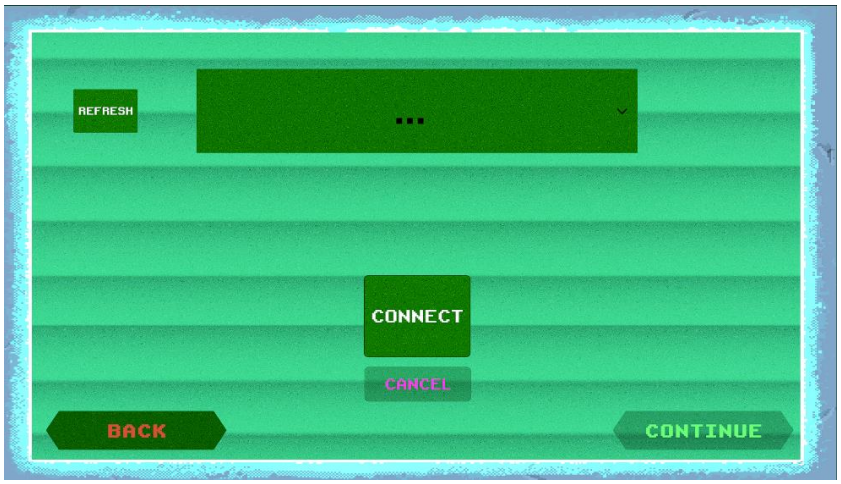


Ilustración 53: estado inicial de la interfaz de conexión

Tras pulsar el botón refresh, se deshabilita este botón para evitar sobrecargar el sistema como se ve en Ilustración 54: Botón refresh.

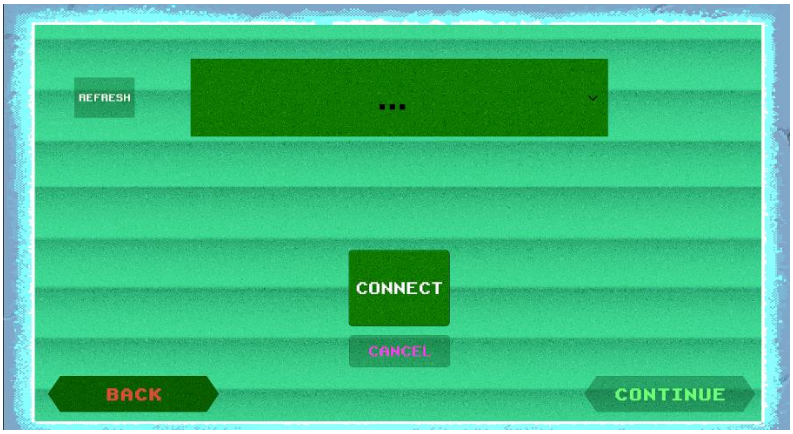


Ilustración 54: Botón refresh

Al desplegar el Dropdown, se muestran los dispositivos disponibles como se ve en Ilustración 55: Dropdown.

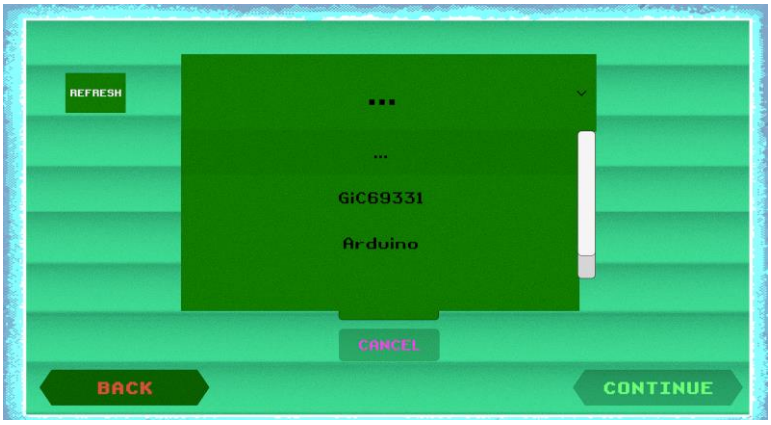


Ilustración 55: Dropdown

Tras pulsar el botón Connect, intenta realizar la conexión y habilita el botón cancel como se ve en Ilustración 56: Botón connect.

Ilustración 56: Botón connect

Conexión realizada, una vez conectado con el dispositivo se deshabilita la interacción con todos los componentes de la interfaz como se puede ver en Ilustración 57: Conexión realizada. Y se muestran por pantalla los movimientos que el usuario realice con el cubo en ese momento.

Ilustración 57: Conexión realizada

5.4.4 Interpretación de los datos recibidos por el ejecutable

Una vez Seleccionado el dispositivo, se pasa el control a la clase CubeInputs. Esta clase se encarga de leer los mensajes que hay en la cola de mensajes y actuar en consecuencia esto se puede ver en la clase ProcessMessages en Ilustración 58: método ProcessMessages. En función del mensaje que llegue se actuará de forma diferente. Si llega un mensaje con el texto “connection failed”, le pedirá a CubeConectionManager que vuelva a intentar conectarse a ese dispositivo. Si llega un mensaje con el texto “Device not found”, refrescará la lista de dispositivos. Si llega un mensaje con el texto “connection successful” se deshabilitará la interfaz de usuario que le permite realizar la conexión. Finalmente, en el resto de los casos se trata de giros del cubo, los cuales se validarán para verificar que son giros y no otros mensajes del proceso para posteriormente ser añadidos a la cola de movimientos.

Ilustración 58: método ProcessMessages

Como se ve en Ilustración 58: método ProcessMessages, el movimiento que se introduce en la cola de movimientos pasa por el método DoubleMove, Esto se debe a que el movimiento realizado junto con el movimiento anterior puede formar un movimiento de una de las capas del medio del cubo o un giro doble.

Se nombrará este tipo de giros como giros dobles ya que se pueden interpretar como el giro de una capa y de su puesta en sentidos diferentes con una mínima diferencia de tiempo, es decir un giro de una de las capas del medio. Esto puede causar varios problemas, el primero es que, si realmente el usuario gira una capa de en medio, el sistema de referencia debe cambiar ya que el centro que se disponía en frente pasa a estar arriba, este problema se tratará en el siguiente apartado. El segundo problema es que se giren las capas opuestas en direcciones opuestas sin que el usuario haya girado una capa intermedia. Este problema no se puede llegar a solucionar ya que el movimiento de las capas es relativo, pero sin embargo se puede ajustar la diferencia de tiempo entre los 2 giros para que sea mínima. Como se ve en Ilustración 59: método DoubleMove, con 500 milisegundos de diferencia debería ser suficiente para solucionar parcialmente este problema. Finalmente, si no cumple las dos condiciones anteriormente mencionadas, es decir, que sean caras opuestas en direcciones opuestas y con una diferencia de tiempo menor que 500 milisegundos, se almacenará en la cola el movimiento que se sacó en un inicio.

```
1 usage  Carlos Chamizo Cano
private Move DoubleMove(Move move)
{
    Move move1 = GetFace(move);
    if (_moves.lastMove == null) _moves.lastMove = move;
    Move move2 = _moves.lastMove;

    if (Math.Abs(move1.time.TotalMilliseconds - move2.time.TotalMilliseconds) < 500)
    {
        //could be a double move
        if (move1.IsMiddleLayer(move2))
        {
            switch (move1.face)
            {
                case FACES.L or FACES.R:
                    if (move1.face is FACES.L)
                        move1.direction *= -1;
                    move1.face = FACES.M;
                    offsetCentersX(move1.direction);
                    break;
                case FACES.U or FACES.D:
                    if (move1.face is FACES.U)
                        move1.direction *= -1;
                    move1.face = FACES.E;
                    offsetCentersY(move1.direction);
                    break;
                case FACES.F or FACES.B:
                    if (move1.face is FACES.B)
                        move1.direction *= -1;
                    move1.face = FACES.S;
                    offsetCentersZ(move1.direction);
                    break;
            }
        }
    }

    return move1;
}
```

Ilustración 59: método DoubleMove

5.4.5 Mejoras para su usabilidad

En el anterior apartado se mencionó que uno de los problemas era que, al realizar un giro doble, el usuario perdía la referencia de centros inicial, por lo que tendría que girar el cubo para volver al punto de referencia inicial. Por lo que para evitar esto y que la experiencia fuera incómoda, se ha implementado un cambio de referencia al realizar giros de capas de en medio y de esta forma el usuario no tendría que reacomodar el cubo. Esto es posible gracias a que se mantiene un array de 2 dimensiones que almacena cada centro del cubo en una posición, por lo que cada vez que se realiza un giro de la capa de en medio, se rotan los centros afectados cambiando así el sistema de referencia.

Cabe destacar que el cubo siempre manda los giros de las caras desde la misma referencia, es decir, centro verde de frente y blanco arriba. Por lo que, al validar los giros, se debe traducir este giro al sistema de referencia actual, es decir, si por ejemplo se recibe que el cubo detectó el movimiento “R”, pero el sistema de referencia actual es rojo en frente y amarillo arriba, este pasará a ser “F”.

Adicionalmente, para evitar confusiones, se muestra por pantalla los centros de referencia actuales ya que es cierto que, en ocasiones, el propio cubo falla y no registra algunos movimientos o al menos, no con la suficiente precisión.

Adicionalmente para que la conexión del cubo perdure, se han usado diferentes mecanismos. El primero es la función de UNITY DontDestroyOnLoad, la cual permite que un objeto, en este caso los objetos que lleva asociados algunas de las clases encargadas de la conexión y gestión de movimientos, permanezca intacta entre escenas, de forma que al cargar una nueva escena no se pierde la conexión. La segunda se encuentra dentro de la clase CubeComunicationManager y es la co-rutina RestablishComunicationRoutine la cual permite en cualquier momento de la ejecución reconectarse al dispositivo. La forma en la que lo hace tal y como se ve en Ilustración 60:RestablishComunicationRoutine es que debido a que ya conoce el nombre del dispositivo, realiza todo el proceso de conexión sin necesidad de indicarlo al igual que se hacía en 5.4.3.1 Interfaz de conexión .

```
Frequently called  1 usage  Carlos Chamizo Cano
public IEnumerator RestablishComunicationRoutine()
{
    _process.EndProcess();
    _process.StartProcess();
    //show cancel button
    if (_cancelButton)
    {
        _cancelButton.interactable = true;
    }

    yield return new WaitForSecondsRealtime( time: 1);
    if (currentDevice != null)
        _process.SendMessageProcess(currentDevice);
    else
    {
        if (continueButton)
            _connectButton.interactable = true;
        if (_cancelButton)
            _cancelButton.interactable = false;
    }
}
```

Ilustración 60:RestablishComunicationRoutine

5.4.6 Interfaz para su uso en un proyecto

De cara al usuario, se le darán ciertos mecanismos para que no tenga que lidiar con la implementación de este proyecto. Por un lado, como se ve en Ilustración 61: interfaz para uso 1, este Prefab[6] contiene todo lo necesario para lanzar el proceso, solo requiere que el usuario introduzca los elementos de la interfaz en el script de CubeConectionManager.

[6] Un prefab es una referencia a un objeto, por lo que se pueden usar múltiples prefabs y todos ellos serán idénticos incluso al realizar cambios en el propio prefab. También es una forma de guardar objetos que puedan resultar interesantes.

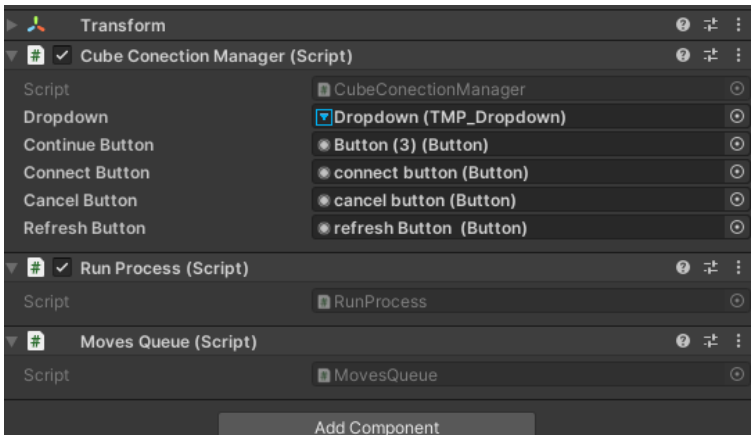


Ilustración 61: interfaz para uso 1

Por otro lado, está la clase MovesQueue la cual es una estructura que tiene como núcleo una cola. Esta clase se usa para múltiples cosas a lo largo del proyecto, pero sin embargo es el puente entre el usuario y la clase CubeInputs. La clase MovesQueue contiene 3 métodos esenciales los cuales se pueden ver en Ilustración 62: Clase MovesQueue. El método EnqueueMsg por el cual una instancia de la clase CubeInputs introduciría los movimientos recibidos y procesados, y las clases HasMessages y Dequeue los cuales permiten al usuario acceder a los movimientos de la cola.

```
Frequently called  5 usages  Carlos Chamizo Cano
public bool HasMessages()
{
    return messages.Count > 0;
}

/// add message to the queue ...
2 usages  Carlos Chamizo Cano
public void EnqueueMsg(string msg) => messages.Enqueue( item: new Move(msg));

1 usage  Carlos Chamizo Cano
public void Enqueue(Move move)
{
    messages.Enqueue(move);
}

/// take out the next message on the queue ...
Frequently called  11 usages  Carlos Chamizo Cano
public Move Dequeue()
{
    return messages.Dequeue();
}
```

Ilustración 62: Clase MovesQueue

5.4.7 Ejemplo de uso en un proyecto real

A continuación, se mostrará un ejemplo del uso de este proyecto en un escenario real, el cual es mi trabajo realizado para el trabajo de fin de grado de diseño y desarrollo de videojuegos.

La forma en la que se decidió usar fue creando un sistema de Inputs propio a medida de este proyecto. Para ello se crearon 3 tipos de clases distintas las cuales explicaré a continuación.

- **MyInputManager**: Se encarga de recibir los movimientos de la clase MovesQueue y comunicárselo al Input concreto activo en ese momento.
- **Inputs**: Estos son los diferentes escenarios que se pueden encontrar a lo largo del juego. Por ejemplo. Los inputs de caminar y mover la cámara, los inputs de un minijuego o lo

inputs del modo disparo. Estas clases se encargan de implementar por un lado los inputs por teclado de dicho escenario y por otro lado el método PerformAction el cual corresponde con el input deseado para cierta cara del cubo.

- **General Input**: Este es un input que permanece escuchando constantemente a pesar de que haya otro input activo en ese instante. Este se ha usado para implementar la pausa de forma que en cualquier momento se pueda acceder a ella.

Además, cabe destacar que esta implementación requiere de planificar cuantos escenarios distintos, es decir que requieran acciones diferentes para los mismos movimientos habrá. Para ello está el ENUM Current input el cual se puede ver en Ilustración 63: CurrentInput. Cada uno de los elementos corresponde a una clase input o escenario distinto.

```
69 usages  Carlos Chamizo Cano  19 exposing APIs
public enum CurrentInput
{
    Movement,
    RotatingWall,
    ColorsMinigame,
    MemoryMinigame,
    RollTheNutMinigame,
    AdjustValuesMinigame,
    AsteroidMinigame,
    ClickFastMinigame,
    LockerMinigame,
    DontTouchTheWallsMinigame,
    PuzzleMinigame,
    MiniBoss,
    StealthMovement,
    Conversation,
    ShootMovement,
    RaceMovement,
    None
}
```

Ilustración 63: CurrentInput

Yendo paso a paso en la implementación todo comienza en la clase MyInputManager donde en la función Update se espera a que en la cola haya algún movimiento. Es aquí donde se descartan aquellos mensajes que el proceso ha recibido y que debido a que en el momento de recepción los inputs estuvieran desactivados, debido a que podría dar una sobrecarga de movimientos cuando se habilitaran otra vez los inputs. La forma en la que se descartan mensajes es comprobando la hora en la que se recibió y en caso de ser más antiguo que 500 milisegundos se descartaría.

```
if (_inputsMoves && _inputsMoves.HasMessages())
{
    Move move = _inputsMoves.Dequeue();
    if (move.time.TotalMilliseconds + 500 < DateTime.Now.TimeOfDay.TotalMilliseconds) return;
```

Ilustración 64: descarte de movimientos

Una vez se tiene un movimiento válido, primero se muestra por GUI [7]y se procede a tratarlo con la clase General Input. Todos los movimientos válidos deben pasar por esta clase la cual más adelante se explicará el funcionamiento de esta clase.

```
if (_playerValues.GetCurrentInput() is not CurrentInput.None)
{
    gui.SetLastMoveText(move);
    generalnputs.PerformAction(move);
}
```

Ilustración 65: MyInputManager general inputs

Finalmente, en función de cuál sea el input actual, se redirige mediante un Switch el movimiento al Input correspondiente tal y como se ve en Ilustración 66: MyInputManager inputs.

```
if (!_playerValues.GetPaused())
{
    switch (_playerValues.GetCurrentInput())
    {
        case CurrentInput.Movement:
            _movementInputs.PerformAction(move);
            break;
        case CurrentInput.RotatingWall:
            _rotatingWallInputs.PerformAction(move);
            break;
        case CurrentInput.ColorsMinigame:
            _colorsInputs.PerformAction(move);
            break;
        case CurrentInput.MemoryMinigame:
            _memoryMinigameInputs.PerformAction(move);
            break;
        case CurrentInput.RollTheNutMinigame:
            _rollTheNutInputs.PerformAction(move);
            break;
        case CurrentInput.AsteroidMinigame:
            _asteroidsInputs.PerformAction(move);
            break;
```

Ilustración 66: MyInputManager inputs

La clase GeneralInputs está muy vinculada a otra clase la cual es PauseManager. Y esto es porque la implementación de la pausa se realiza mediante una secuencia de movimientos y no un solo movimiento es por ello por lo que tal y como se ve en la Ilustración 67: PerformAction GeneralInputs la clase GeneralInputs pregunta a la clase PauseManager si la secuencia introducida es correcta.

```
public void PerformAction(Move move)
{
    if (pauseManager.CheckCubePause(move))
        pauseManager.Pause();
}
```

Ilustración 67: PerformAction GeneralInputs

[7] Interfaz gráfica de usuario

La forma en la que funciona esta secuencia es la siguiente. Primero se establece una secuencia de movimientos, en el caso de este ejemplo se usó una secuencia muy famosa entre el mundo de los cubos de Rubik la cual se conoce como una T. Para crear la secuencia como se ve en Ilustración 68: secuencia de pausa. Para ello se ha usado la clase Move para de esta forma comparar los movimientos entrantes directamente con movimientos del mismo tipo.

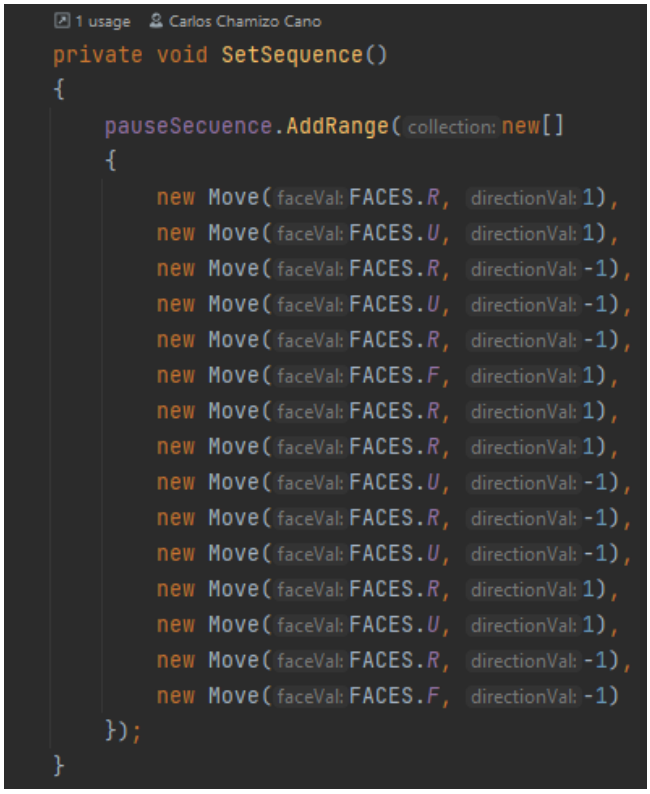


Ilustración 68: secuencia de pausa

Los movimientos recibidos son comparados con la secuencia que se muestra en Ilustración 68: secuencia de pausa, tal y como está implementado en Ilustración 69: Comprobación de la secuencia., En caso de coincidir se sumaría una unidad a un contador que almacena los movimientos coincidentes y en caso de fallar la secuencia se reiniciaría el contador a 0, de tal forma que si el contador llega al número de movimientos de la secuencia significa que se ha completado y por lo tanto se activaría la pausa.

```
Frequently called  1 usage  Carlos Chamizo Cano
public bool CheckCubePause(Move move)
{
    if (!playerValues.GetPaused())
    {
        guiManager.ShowPauseIndicator();
        if (move.Equals(pauseSecuence[sequenceIndex]))
        {
            if (sequenceIndex == pauseSecuence.Count - 1)
            {
                guiManager.HidePauseIndicator();
                sequenceIndex = 0;
                return true;
            }

            sequenceIndex++;
            guiManager.SetNextMoveText(pauseSecuence[sequenceIndex].ToString());
            guiManager.FillPauseSlider((float)sequenceIndex / (pauseSecuence.Count - 1));
            return false;
        }

        sequenceIndex = 0;
        guiManager.SetNextMoveText(pauseSecuence[sequenceIndex].ToString());
        guiManager.FillPauseSlider(0);
        return false;
    }

    return false;
}
```

Ilustración 69: Comprobación de la secuencia.

Finalmente queda por explicar los inputs concretos. Estos como ya se ha mencionado anteriormente, corresponden con un escenario concreto. Como se ve en Ilustración 70: PerformAction, aquellos movimientos que MyInputManager deriva a cada input acaban en este punto, donde para cada cara o posible movimiento del cubo se le asigna una acción dentro de ese escenario concreto.

```
Frequently called  1 usage  Carlos Chamizo Cano
public void PerformAction(Move move)
{
    //accelerate deccelerate
    if (_playerValues.GetInputsEnabled())
    {
        guiManager.SetTutorial(
            cad:"R - Increase/Decrease gear   U - Camera horizonatal axis    L - Camera Vertical Axis    B - Lights");

        if (move.face == FACES.R)
        {
            _playerValues.CheckIfStuck();
            if (move.direction == 1)
            {
                if (_playerValues.GetGear() < 3)
                    _playerValues.RiseGear();
            }
            else
                _playerValues.DecreaseGear();
        }
    }

    //turn camera in y axis
    if (move.face == FACES.L)
    {
        if (move.direction == 1)
            _cameraController.RotateVerticalDown();
        else
            _cameraController.RotateVerticalUp();
    }
    else if (move.face == FACES.U)
    {
        if (move.direction == 1) _cameraController.RotateClockwise();
        else _cameraController.RotateCounterClockwise();
    }
    else if (move.face == FACES.B)
    {
        if (move.direction == 1)
        {
            if (_playerValues.GetLights())
                _playerValues.NotifyAction(PlayerActions.TurnOffLights);
        }
        else
        {
            if (!_playerValues.GetLights())
            _playerValues.NotifyAction(PlayerActions.TurnOnLights);
        }
    }
```

Ilustración 70: PerformAction

5.5 Sistema de control

En este apartado se detallará el proceso de creación del dispositivo que actúa como complemento de la interfaz del cubo de Rubik. Este proyecto presentó un desafío significativo desde el principio, ya que se buscaba desarrollar un circuito que cumpliera con los requisitos específicos del proyecto. Sin embargo, se encontró un obstáculo importante relacionado con el tamaño objetivo, ya que ninguna de las opciones disponibles cumplía con las necesidades y restricciones del proyecto.

Ante esta situación, se decidió abordar el problema mediante la creación de un prototipo que permitiera explorar diferentes enfoques y soluciones, manteniendo como objetivo principal reducir al mínimo el tamaño de los componentes utilizados. Aunque no se impuso una restricción de tamaño específica para el prototipo, se buscó optimizar el diseño para lograr un dispositivo compacto y funcional.

5.5.1 Justificación del uso de Arduino

Tal y como se explica en la web oficial de Arduino [42], Arduino es una plataforma de código abierto que combina hardware y software de manera sencilla y accesible. Esta plataforma brinda a los usuarios la capacidad de crear una amplia variedad de proyectos físicos, que van desde robots hasta consolas de videojuegos. La versatilidad y la libertad que ofrece Arduino permiten que cualquier idea se convierta en realidad.

Por otro lado, las placas de marca Raspberry, son la competencia que más se asemeja a la hora de pensar en el desarrollo de un proyecto de estas características. Las placas Raspberry son por así llamarlo ordenadores de placa o en otras palabras un SBC[8]. Esto supone que dispone de una gran potencia además de venir incorporado con un amplio abanico de funcionalidades como bluetooth o wifi. Su software se basa en Linux sin embargo se le puede instalar una gran variedad de sistemas operativos según el propósito de su uso.

El uso de una placa compatible con Arduino frente a otros dispositivos como una Raspberry Pi se debe a varios motivos. El motivo principal es la disponibilidad limitada de microcontroladores que se ajustaran a las necesidades específicas del proyecto, y debido a que se encontró una la cual se ajustaba bastante bien al proyecto y la cual era compatible con Arduino se decidió optar por esta opción.

5.5.2 Introducción a Arduino

Arduino se compone de una placa de desarrollo y un entorno de programación fácil de usar. La placa de desarrollo está equipada con un microcontrolador y una serie de pines de entrada y salida, lo que permite conectar y controlar diversos componentes electrónicos.

El entorno de programación de Arduino se basa en un lenguaje de programación simplificado. Además de que cuenta con una amplia comunidad y documentación por lo que facilita el aprendizaje de este.

5.5.2.1 Placa de desarrollo

Para explicar los elementos básicos de Arduino, todos los ejemplos irán referidos a la clásica placa Arduino Uno la cual se puede ver en Ilustración 71: placa Arduino Uno. La placa es la base es, por

[8] Single board computer

así decirlo es el cerebro de un proyecto de Arduino ya que este se encarga de realizar operaciones y mandar y recibir señales.

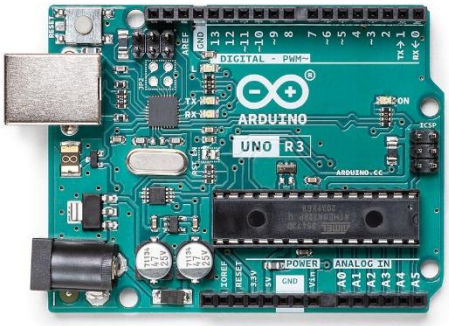


Ilustración 71: placa Arduino Uno

Las señales las recibe a través de los pines a los cuales se conectan los diferentes sensores y componentes del proyecto.

5.5.2.2 Señales

Las señales son el mecanismo que permite a la placa del Arduino comunicarse y recibir información de los sensores y componentes. Hay dos tipos de señales, las señales analógicas y las señales digitales.

- **Señales analógicas**: son aquellas señales que se reciben del exterior y tienen un valor continuo, estas son las que se reciben de fenómenos naturales como puede ser la temperatura o el sonido. Arduino posee 6 pines analógicos los cuales son lo pines del A0 al A5 y se encarga el Arduino de discretizar estos valores de forma que se puedan interpretar.
- **Señales Digitales**: las señales digitales son aquellas señales que pueden tener un valor de 0 o de 1 que para un componente como puede ser un leed significa encendido o apagado. Arduino Uno tiene 14 pines digitales.

5.5.2.3 Código

Para que un proyecto de Arduino funcione correctamente, es necesario instalar el código correspondiente en la placa de desarrollo. En este apartado, se mencionarán de manera general sin entrar mucho en detalle además de las funciones básicas del código de Arduino.

En primer lugar, para instalar el código en la placa de Arduino, es necesario utilizar el entorno de desarrollo de Arduino, el cual se puede descargar de forma gratuita desde su página oficial [43]. Este entorno viene preparado para detectar y comunicarse con las placas Arduino más modernas. Una vez instalado, se puede escribir el código en el editor de Arduino y luego cargarlo en la placa a través de una conexión USB.

En cuanto a las funciones mínimas de Arduino, es esencial comprender dos de ellas:

- La función "setup()": Esta función se encarga de realizar la configuración inicial del proyecto. En ella, se deben indicar los pines que se utilizarán y si serán configurados como pines de entrada o de salida mediante la función "pinMode()". Es común utilizar esta función para inicializar variables y configurar parámetros iniciales.

- La función "loop()": Dentro de esta función se define el comportamiento principal del proyecto. El código escrito dentro de esta función se repetirá en un bucle continuo mientras la placa de Arduino esté encendida. Aquí es donde se pueden incluir instrucciones y funciones específicas del proyecto. Algunas funciones comunes son:
    - "delay()": Esta función pausa la ejecución del programa durante un período de tiempo determinado, especificado en milisegundos.
    - "digitalWrite()": Esta función se utiliza para establecer el estado de un pin digital, es decir, si estará en un estado alto ("HIGH") o bajo ("LOW"). Se utiliza para controlar componentes como LEDs.
    - "analogRead()": Esta función permite leer valores analógicos de los pines de entrada analógicos. Es útil para capturar señales de sensores analógicos y utilizar esos valores en el programa.

5.5.3 Diferentes Microcontroladores

En el proceso de selección de microcontroladores para el proyecto, se tuvieron en cuenta varias opciones. A continuación, se presentarán algunos de los microcontroladores considerados, junto con sus especificaciones técnicas y algunas observaciones relevantes:

- **µduino:** tal y como se puede ver en Ilustración 72: µduino, el µduino es un microcontrolador compacto con dimensiones de 12 mm x 12 mm. Utiliza el microchip ATMEGA32U4 y ofrece 20 puertos de entrada/salida para la conexión de diferentes componentes. Puede recibir energía tanto del USB como de baterías y es compatible con dos modos de energía: 3.3V y 5V. Además, cuenta con 28KB de memoria programable [44]. Una ventaja del µduino es su tamaño compacto, que permite una integración fácil en proyectos con limitaciones de espacio. Sin embargo, es importante tener en cuenta que este microcontrolador ya no se fabrica y su precio puede ser más alto en comparación con otras opciones más comunes en el mercado, aproximadamente 18 euros.

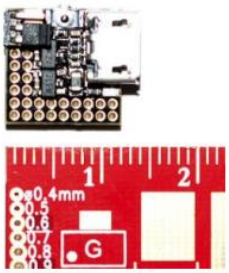


Ilustración 72: µduino

- **Seeeduino XIAO SAMD21g18**: El Seeeduino XIAO SAMD21g18, el cual se puede ver en Ilustración 73: Seeeduino XIAO SAMD21g18 es otro microcontrolador para considerar. Tiene dimensiones de 20 x 17.5 x 3.5 mm. Utiliza el microchip ARM Cortex-M0+ CPU (SAMD21G18) y ofrece 14 puertos de entrada/salida para la conexión de dispositivos externos. Puede recibir energía tanto a través de USB-C como de baterías y es compatible con dos modos de energía: 3.3V y 5V. Cuenta con 32KB de memoria SRAM y 256KB de memoria flash. Además, es compatible con el IDE de Arduino. [45]
  Una ventaja del Seeeduino XIAO SAMD21g18 es su precio más económico en comparación con el µduino. Sin embargo, es importante tener en cuenta las restricciones de espacio y evaluar si el tamaño y la forma del microcontrolador se adaptan al proyecto.

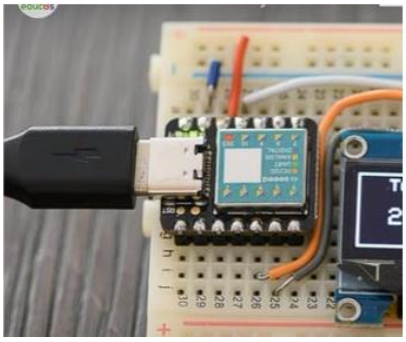


Ilustración 73: Seeeduino XIAO SAMD21g18

- **Seeed Studio XIAO nRF52840 (sense):** El Seeed Studio XIAO nRF52840 (sense) es otro microcontrolador a tener en cuenta. Tiene las mismas dimensiones que el Seeeduino XIAO SAMD21g18, con 20 x 17.5 x 3.5 mm. Este microcontrolador utiliza el Nordic nRF52840, un procesador ARM Cortex-M4 de 32 bits con FPU y una velocidad de 64 MHz. Ofrece un consumo de energía ultra bajo y cuenta con 2MB de memoria flash.
  Una característica destacada del Seeed Studio XIAO nRF52840 (sense) es que incluye componentes adicionales como un micrófono y un giroscopio integrados, lo que puede ofrecer mayores posibilidades de interacción. También cuenta con 14 puertos de entrada/salida externos [46].
  Al igual que los microcontroladores anteriores, puede recibir energía tanto a través de USB-C como de baterías y es compatible con los modos de energía de 3.3V y 5V. También es compatible con el IDE de Arduino, lo que facilita su programación.
  Una ventaja adicional de este microcontrolador es que viene con conectividad Bluetooth 5.0 integrada, lo cual quita de en medio el problema que suponía buscar una forma de agregar funcionalidad bluetooth a este proyecto.

Ilustración 74: Seeed Studio XIAO nRF52840 (sense)

Tras esta comparativa de microcontroladores se decidió optar por la última opción, el Seeed Studio XIAO nRF52840 (sense) ya que este a pesar de que es ligeramente más grande que el µduino este suplía su tamaño incorporando elementos esenciales para este proyecto como puede ser la conectividad bluetooth.

5.5.4 Comparativa de Componentes

Además del microcontrolador es importante elegir unos componentes que además de ser pequeños y compactos aporten un valor añadido al resultado final. Estos son algunos de los componentes que se consideraron para ser añadidos al proyecto.

Un **motor de vibración** el cual se puede ver en Ilustración 75: motor de vibración en un juego haría que el cubo vibre en momentos o estados específicos para transmitir información relevante al jugador. Este feedback háptico agrega una dimensión adicional a la experiencia de juego, permitiendo al jugador percibir eventos o acciones de una manera más inmersiva.	 *Ilustración 75: motor de vibración*
Un **interruptor de agitación** o **vibración** el cual se puede ver en Ilustración 76: sensor de agitación en un juego permite que el propio juego reaccione ante los movimientos físicos del cubo. Cuando el sensor detecta una agitación o vibración, se activa una respuesta específica en el juego, como, por ejemplo, la pausa. Sin embargo, también hay algunas consideraciones para tener en cuenta. La precisión y sensibilidad del sensor pueden variar, lo que podría afectar la respuesta del juego.	 *Ilustración 76: sensor de agitación*
El **buzzer,** el cual se puede ver en Ilustración 77: buzzer en un videojuego puede ser utilizado de diversas formas para indicar eventos o estados específicos al jugador. Su implementación es relativamente sencilla, ya que solo requiere enviar una señal eléctrica al buzzer para que emita el sonido deseado. Sin embargo, si el jugador utiliza auriculares o cascos para jugar, parte de la experiencia sonora puede perderse.	 *Ilustración 77: buzzer*

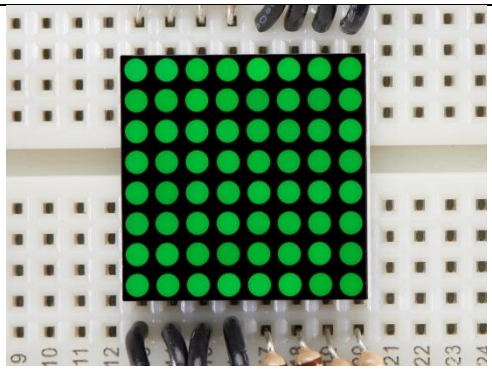
La utilización de una **matriz de LED de 8x8**, la cual se puede ver en Ilustración 78: matriz de LEDs 8x8 permite ofrecer información más compleja desde el juego. Con esta matriz, es posible mostrar formas, y patrones personalizados en una interfaz visual. Esto abre la posibilidad de mostrar mensajes, indicadores de estado, puntuaciones y otros elementos visuales relevantes para mejorar la experiencia de juego.	 *Ilustración 78: matriz de LEDs 8x8*

Tabla 5: componentes arduino

Finalmente, para el proyecto final se seleccionaron tres componentes clave: el motor de vibración, el sensor de agitación y la matriz de LEDs 8x8. Estos componentes fueron elegidos debido a las diferentes funciones que ofrecen y su capacidad para enriquecer la experiencia de juego.

El motor de vibración fue seleccionado por su capacidad de proporcionar feedback háptico al jugador. La matriz de LEDs 8x8 se eligió por su capacidad de ofrecer feedback visual. Los LEDs pueden iluminarse de diferentes maneras para indicar información importante. Por último, el sensor de agitación se incorporó para permitir una forma de interacción adicional con el juego. Cuando el jugador agita el cubo, el sensor de agitación detecta este movimiento y desencadena una respuesta en el juego.

5.5.5 Esquema de conexiones

Como se puede ver en Ilustración 79: Esquema de conexiones del proyecto esta es la disposición final de los diferentes dispositivos que se han elegido para el proyecto. A continuación, se especificará como se han conectado cada uno de ellos al microcontrolador.

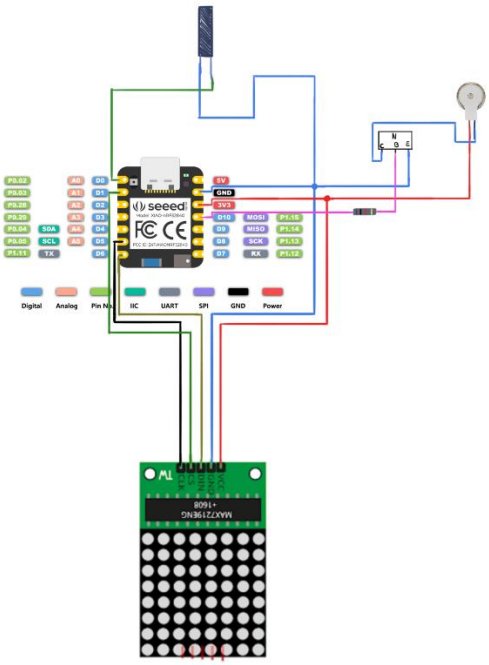


Ilustración 79: Esquema de conexiones del proyecto

Conexiones:

- **Matriz de LEDs 8x8**: la matriz de leds se conecta al microcontrolador de a través de una interfaz de 5 pines. El primer pin es CLK, este se conecta directamente a al pin 5. El siguiente es el pin CS, este va directamente conectado al pin 1 del microcontrolador, el siguiente es el pin Din de datos, este va conectado a directamente al pin de 6 del microcontrolador, finalmente quedan los pines GND el cual se conecta al pin de tierra del controlador y el pin VCC el cual se conecta al pin de 3 voltios.
- **Interruptor de agitación**: el interruptor de agitación es el componente el cual se ve en la parte superior de Ilustración 79: Esquema de conexiones del proyecto. Este tiene una sencilla conexión: posee dos pines de los cuales uno de ellos se distingue por ser más largo, el pin de menor tamaño va conectada al pin de tierra del microcontrolador y terminal largo va conectada a el pin 0 del microcontrolador.
- **Motor de vibración**: el motor de vibración tiene una conexión un poco más enrevesada. El motor tiene dos patillas, una de color rojo y otra de color azul. La patilla de color rojo se conecta directamente al pin de 3 voltios mientras que la azul va conectada al pin colector de un NPN. El NPN es el componente que permitirá al microcontrolador activar y desactivar el motor de vibración ya que este por sí mismo solo tiene un cable de tierra y otro de corriente. El pin Base del NPN, se conectará con el pin 10 del Arduino y finalmente el pin Emisor del NPN se conectará al pin de tierra.

5.5.6 Comportamiento del sistema

Cabe destacar que este proyecto y toda la información de esta parte del trabajo, está cuidadosamente diseñada para ser usada en el juego que se ha desarrollado para el trabajo de fin de grado de diseño y desarrollo de videojuegos. Por lo tanto, su uso está limitado a mejorar la experiencia de juego.

5.5.6.1¿Qué papel tiene cada componente?

Ya se ha visto cuales son los componentes que se han utilizado y como se conectan, pero falta saber qué hace realmente cada uno de los componentes. A continuación, se explicará el funcionamiento de cada uno de los componentes.

- **Matriz de leds 8x8**: la matriz de leds se ha usado para mostrar información relevante del juego. Su función principal es la de mostrar la cara del personaje de forma sincronizada con el juego, esto quiere decir que tal y como se ve en Tabla 6: estados de la matriz de LEDs se mostrará su cara en estado normal, en estado de pestañeo, en estado asustado y en estado de muerte, además de esto la matriz de leds muestra si el personaje del juego tiene o no las luces encendidas. Por otro lado, no solo se mostrará información del personaje sino también de eventos del juego como por ejemplo la pausa o que se está jugando a un minijuego concreto.

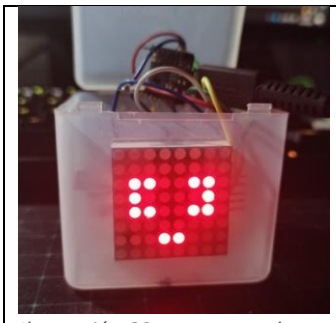


Ilustración 80: cara normal

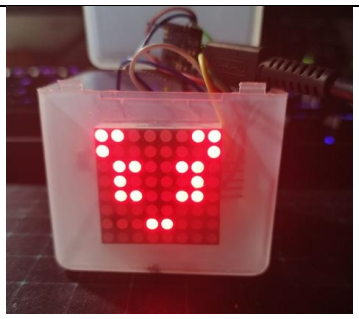


Ilustración 81: cara normal con luces

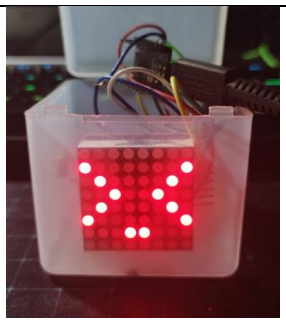


Ilustración 82: cara asustada

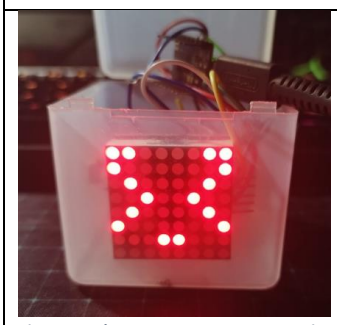


Ilustración 83: cara asustada con luces

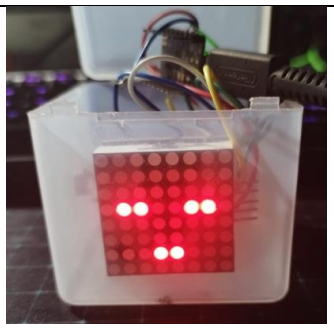


Ilustración 84: cara pestañeo

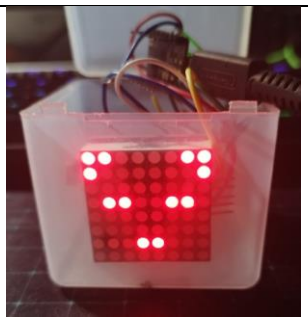


Ilustración 85: cara pestañeo con luces

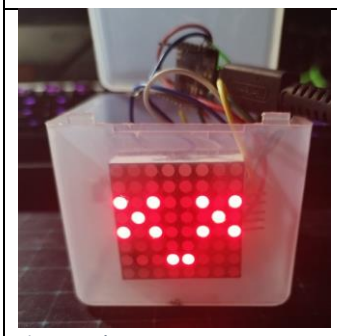


Ilustración 86: cara muerte

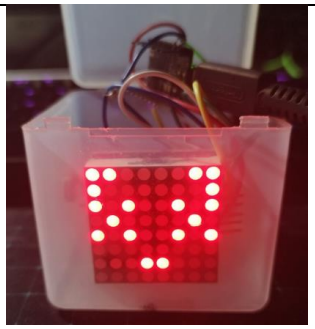


Ilustración 87: cara muerte con luces

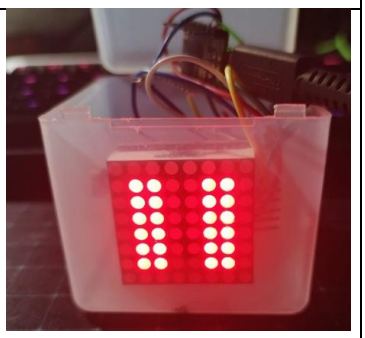


Ilustración 88: pantalla de pausa

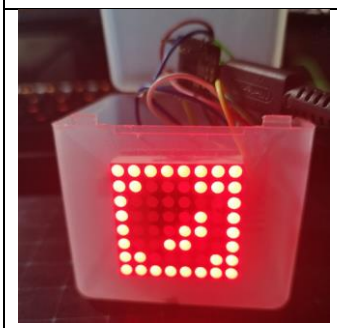


Ilustración 89: minijuego de asteroides

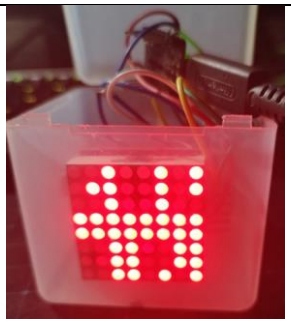


Ilustración 90: minijuego de colores

[9]

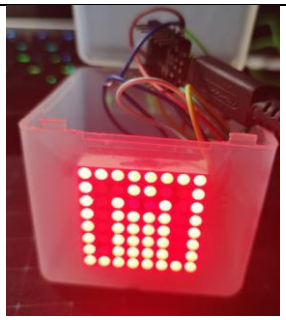


Ilustración 91: minijuego de memoria

[9] Está escrito “color” con los LEDs apagados

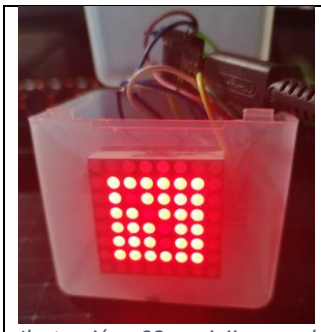
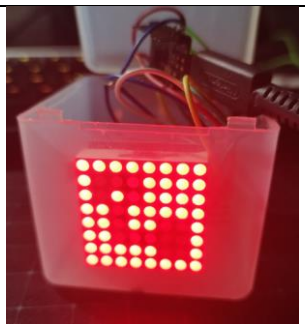
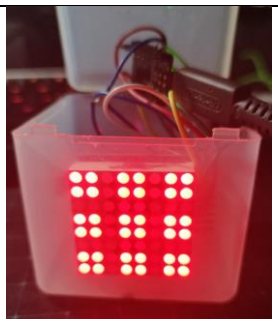
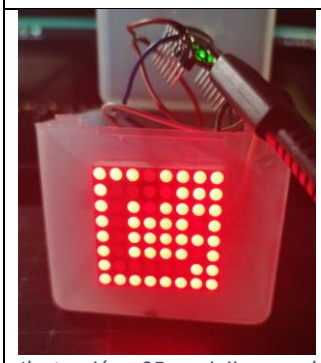
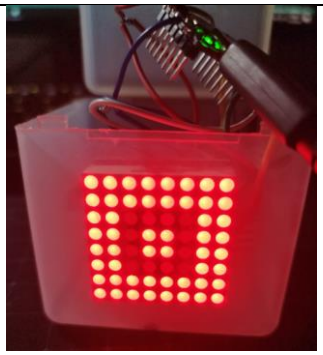
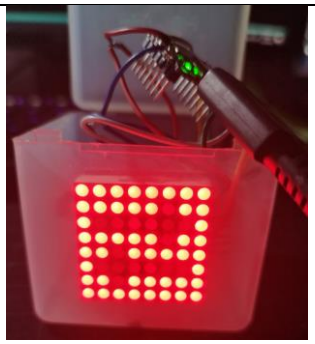
 *Ilustración 92: minijuego de esperar*	 *Ilustración 93:minijuego de no tocar paredes*	 *Ilustración 94: minijuego de puzle*
 *Ilustración 95: minijuego de girar la tuerca*	 *Ilustración 96: minijuego de pulsar rápido*	 *Ilustración 97: minijuego de ajustar valores*

Tabla 6: estados de la matriz de LEDs

- **Motor de vibración**: la función del motor de vibración es la de vibrar a petición, es decir cuando desde el juego ocurra un suceso que requiera vibración, el propio juego le pedirá que empiece y que para de vibrar, la gestión del tiempo de vibración se realiza desde el propio juego.

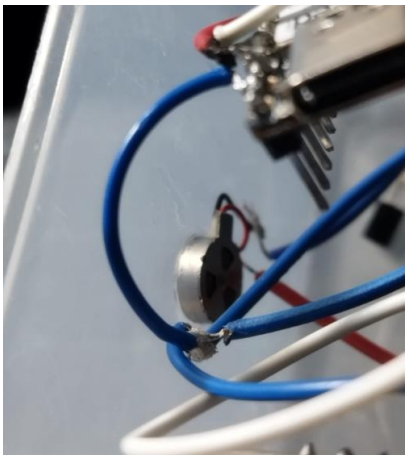


Ilustración 98: motor de vibración final

- **interruptor de agitación**: el cual se puede ver en Ilustración 99: sensor de agitación final Ilustración 99: sensor de agitación final, tiene la función de activar y desactivar el menú de pausa. Sin embargo, debido a que los datos recibidos por el sensor en ocasiones generan datos indeseados, se ha controlado por medio de código la interpretación de los datos recopilados.

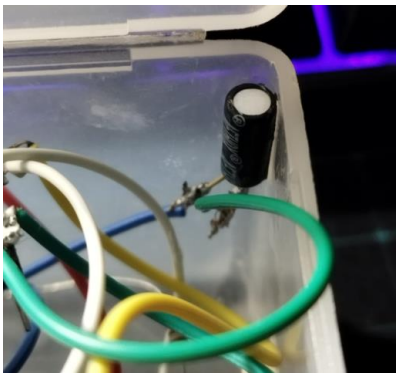


Ilustración 99: sensor de agitación final

5.5.6.2 Funcionamiento general

Como se puede observar en Ilustración 100: esquema general el sistema está formado por 3 partes, la primera es el microcontrolador el cual se encarga de mantener en funcionamiento los componentes adheridos al mismo, este para comunicarse con el juego usa el protocolo Bluetooth LE al igual que se hacía con la interfaz del cubo de Rubik 5.3.4 Fases y funcionamiento del proyecto Debido a que se usa Bluetooth LE esto facilita mucho la implementación de la comunicación entre Arduino, el proceso intermedio y UNITY. La siguiente parte es el proceso intermedio, el cual actúa de interfaz entre UNITY y Arduino. Y final mente está la parte de UNITY la cual se encarga de hacer peticiones al proceso intermedio y de recibir señales del sensor de agitación. En este apartado se explicará en detalle la implementación de estas tres partes.

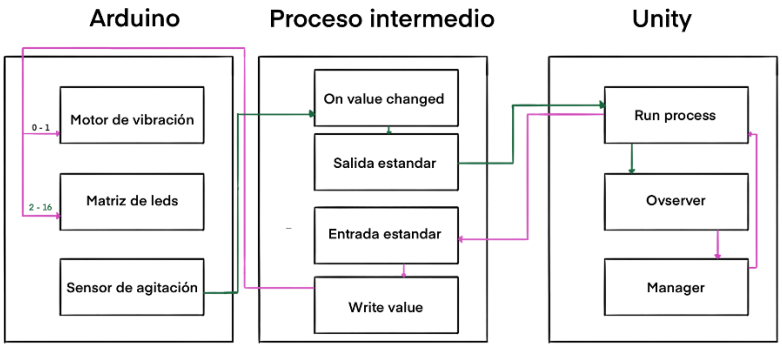


Ilustración 100: esquema general Arduino

5.5.6.2.1 Arduino

En Arduino a diferencia del cubo de Rubik, Hay que crear su propio servicio bluetooth además de sus propias características. Esto es una ventaja ya que a diferencia del cubo donde en Descubrimiento del servicio había que descubrir cada proceso a base de prueba error en este caso se puede introducir manualmente el servicio y tantas características como se necesiten. En este caso como se ve en Ilustración 101: servicios y características de se han usado 3 características, una para el motor de vibración la cual es “switchCharacteristic” con las propiedades de lectura y escritura (aunque solo se usará la propiedad de escritura), otra característica para el sensor de vibración la cual es “motionCharacteristic” la cual solo tiene la propiedad de notificar ya que solo se necesita recibir datos del sensor. Y final mente una característica para el panel LED la cual es "ledPanelCharacteristic", esta tiene las propiedades tanto de lectura como de escritura.

```
BLEService ledService("19B10000-E8F2-537E-4F6C-D104768A1214"); // Bluetooth® Low Energy LED Service
// Bluetooth® Low Energy LED Switch Characteristic - custom 128-bit UUID, read and writable by central
BLEByteCharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
BLEByteCharacteristic motionCharacteristic("19B10000-E8F2-537E-4F6C-D104768A1214", BLENotify);
BLEByteCharacteristic ledPanelCharacteristic("19B10002-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
```

Ilustración 101: servicios y características de Arduino

Posteriormente en la función “setUp”, como se ve en Ilustración 102: arranque de bluetooth , haciendo uso de la librería ArduinoBLE, se inicia el módulo de bluetooth del microcontrolador. Además, se añade el servicio y las características que se han creado previamente para posteriormente publicitar al resto de dispositivos que este dispositivo está disponible. Por lo tanto, el microcontrolador se queda a la espera de que un dispositivo realice la conexión.

```
// begin bluetooth initialization
if (!BLE.begin()) {
    Serial.println("starting Bluetooth® Low Energy module failed!");

    while (1);
}

// set advertised local name and service UUID:
BLE.setLocalName("LED");
BLE.setAdvertisedService(ledService);

// add the characteristic to the service
ledService.addCharacteristic(switchCharacteristic);
ledService.addCharacteristic(motionCharacteristic);
ledService.addCharacteristic(ledPanelCharacteristic);
// add service
BLE.addService(ledService);
// set the initial value for the characeristic:
switchCharacteristic.writeValue(0);
motionCharacteristic.writeValue(0);
ledPanelCharacteristic.writeValue(0);
// start advertising
BLE.advertise();
```

Ilustración 102: arranque de bluetooth Arduino

Dentro de la función “loop”, el microcontrolador se queda a la espera de que algún dispositivo se conecte. Mientras esto no ocurra, se muestra por la matriz de LEDs varios patrones. Se puede saber que la conexión se realizó correctamente ya que en la matriz de LEDs se podrá ver un “OK”.

El otro dispositivo que se deberá conectar al Arduino será el proceso intermedio. Una vez conectado el Arduino realizará el bucle de activades mientras este siga conectado, es decir que se pondrá a la espera de recibir valores a través de las características a la vez que envía el estado del motor de vibración.

El bucle de actividades es el que se puede ver en Ilustración 103: loop de actividades por cada iteración del bucle comprueba si se ha escrito algún valor en las características del motor de vibración o de la matriz de LEDs.

```
while (central.connected()) {
    // if the remote device wrote to the characteristic,
    // use the value to control the LED:
    if (switchCharacteristic.written()) {
        int value = switchCharacteristic.value();
        if (value == 1) {...} else if (value == 0) {...}
    }
    if (ledPanelCharacteristic.written()) {
        int value = ledPanelCharacteristic.value();
        if (value == LIGHTS) {   // any value other than 0
            if (lightsOn == 1)lightsOn = 0;
            else lightsOn = 1;
            chooseFace(currentFace);
        } else {
            currentFace = chooseFace(value);
        }
    }
    //motion sensor
    tiltSensorCurrentValue = digitalRead(motionSensorPin);
    if (tiltSensorPreviousValue != tiltSensorCurrentValue) {...}
    if (millis() - lastTimeMoved < shakeTime) {...} else {
        if (previousVal != 0) {...}

    }
}
```

Ilustración 103: loop de actividades

Para regular que acción se quería realizar, es decir que componente y de qué forma se quería usar, se ha asignado a cada número una acción de forma que desde UNITY solo sea necesario saber qué número corresponde a que acción. Los números 0 y 1 corresponden con apagar y encender el motor de vibración respectivamente y los números del 2 al 16 corresponde con una forma en la matriz de LEDs. Sin embargo, el número 6, tiene una función especial, y es que como se puede ver en Ilustración 103: loop de actividades, y conociendo que la constante LIGHTS equivale al valor 6, acciona y apaga las luces de aquellas formas de la matriz de LEDs que correspondan a una cara del personaje, de esta forma se puede representar que dentro del juego se encendieron o apagaron las luces.

5.5.6.2.2 Proceso intermedio

Respecto al proceso intermedio se van a explicar las cosas que difieren con el proceso intermedio de 5.3.3 Estructura del código explicado para la conexión del cubo de Rubik ya que la base es exactamente la misma.

La primera diferencia y ventaja es que desde el principio se conoce el servicio que se buscará, así como las características concretas lo cual se puede ver en Ilustración 104: características del servicio del Arduino. Esto permite asegurar que se encontró el servicio y las características exactas que se necesitan.

```
public static string VIBRATION_ID = "19B10001E8F2537E4F6CD104768A1214";
public static string MOTION_SENSOR_ID = "19B10000E8F2537E4F6CD104768A1214";
public static string LED_PANEL_ID = "19B10002E8F2537E4F6CD104768A1214";
```

Ilustración 104: características del servicio del Arduino

La siguiente diferencia viene en la función "Characteristic_ValueChanged" la cual se puede ver en Ilustración 105: "Characteristic_ValueChanged" ya que esta ahora deberá recibir los valores de captados por el sensor de agitación. En este caso ya que el sensor de agitación envía un byte siendo su valor 0, cuando no se está agitando y 1 cuando si se está agitando. En la función

"Characteristic_ValueChanged" solo se deberá leer un byte y mostrarlo por la salida estándar para que UNITY.

```
private static void Characteristic_ValueChanged(GattCharacteristic sender, GattValueChangedEventArgs args)
{
    var reader = DataReader.FromBuffer(args.CharacteristicValue);
    byte[] input = new byte[reader.UnconsumedBufferLength];
    reader.ReadBytes(input);

    Console.Out.WriteLine("Motion sensor: " + input[0]);
}
```

Ilustración 105: "Characteristic_ValueChanged"

Finalmente, la última diferencia está en la forma de tratar la información que entra por la entrada estándar como se puede ver en Ilustración 106: tratamiento de entrada estándar, ya que esta está preparada para enviar a la característica correspondiente el valor dentro del rango de valores especificado en la sección 5.5.6.2.1 Arduino.

```
string inputMsg = Console.ReadLine();
while (inputMsg != "f")
{
    inputMsg = Console.ReadLine();
    int value = 0;
    bool isNumeric = Int32.TryParse(inputMsg, out value);
    if (isNumeric)
    {
        if (value <= 1)
        {
            Console.WriteLine("Writting value in characteristic " + ledCharacteristicWrite.Uuid);
            writeDevice(ledCharacteristicWrite, ConvertInt32ToByteArray(value)[0]);
        }
        else
        {
            Console.WriteLine("Writting value in characteristic " + ledCharacteristicWrite.Uuid);
            writeDevice(ledPanelCharacteristicWrite, ConvertInt32ToByteArray(value)[0]);
        }
    }
}
```

Ilustración 106: tratamiento de entrada estándar

5.5.6.2.3 UNITY

En cuanto a la implementación de todo lo previamente visto dentro de UNITY, se realiza mediante el uso del patrón “observer” como explicaré más adelante en este apartado. Respecto a la interfaz de conexión y comunicación con el proceso intermedio no ha cambiado nada respecto a la implementación de la interfaz del cubo de Rubik de la sección 5.4.3 Conexión con el ejecutable.

Respecto a cómo se detecta cuando UNITY debe mandarle una señal al Arduino, se realiza mediante el patrón “observer”, este patrón ya se utiliza con otros propósitos dentro del juego, pero permite implementar el uso del Arduino sin afectar ni intervenir en ningún otro aspecto con el juego.

En primer lugar, se tienen los sujetos, estos corresponden con aquellas clases que heredan de la clase abstracta “Subject” la cual se puede ver en Ilustración 107: clase abstracta Subject. Esta clase abstracta implementa todos los mecanismos necesarios para comunicarse con los observadores, por lo que en el momento que se comunique una acción los observadores actuarán en consecuencia.

Ilustración 107: clase abstracta Subject

Las acciones que puede realizar el personaje son aquellas que están declaradas en el enum "PlayerActions". Algunas de las acciones que el personaje puede realizar son, disparar, sentarse, encender o apagar luces etc.

Por otro lado, están las clases que aplican la interfaz "IObserver" en esta interfaz tan solo hay un método a implementar el cual es "OnNotify" será aquí donde se implementen las acciones que se deben comunicar al Arduino.

La clase que se ha implementado con el propósito de mantener una comunicación de estas acciones con el Arduino es la clase ArduinoObsrver donde cada vez que se notifique una acción se pedirá a la clase ArduinoManager, la cual es la encargada de pedirle al proceso intermedio que realice las acciones en bruto. Un ejemplo de uso es el daño recibido como se puede ver en Ilustración 108: ejemplo de uso del patrón observer, cuando se notifica que se ha recibido daño, ya sea bajo, medio o alto, se lanza una co-rutina la cual se pude ver en Ilustración 109: ejemplo de uso de vibración donde pide al ArduinoManager que active o desactive la vibración. La función dentro del ArduinoManager que activa o la que desactiva la vibración se limita a escribir por la entrada estándar del proceso intermedio el valor correspondiente a la vibración, en este caso un 0 o un 1.

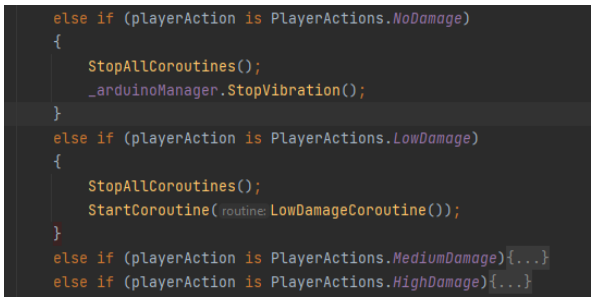


Ilustración 108: ejemplo de uso del patrón observer

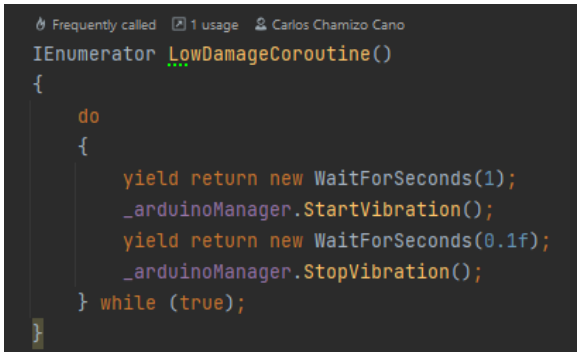


Ilustración 109: ejemplo de uso de vibración

6 Pruebas realizadas

Para llegar al resultado final se ha pasado por ciertas pruebas donde se ha ido incrementalmente añadiendo funcionalidad al circuito. Las pruebas se han realizado con el siguiente equipo: procesador AMD Ryzen 7 1700 Eight core, 16 GB de RAM, sistema operativo Windows 10 Education 22H2 64 bits y MicroControlador Seeed Studio XIAO nRF52840 (Sense).

6.1 Prueba LED

El objetivo de esta prueba era realizar una conexión sencilla entre el dispositivo seeed XIAO nRF52840 y la computadora a través de un ejecutable utilizando como medio de comunicación Bluetooth BLE donde el usuario a través del ejecutable manda una señal al dispositivo escribiendo un “2” por consola lo cual haría que se encienda o apague el LED incorporado en el propio microcontrolador.

Para ello, se ha usado en el microcontrolador que se ve en Ilustración 110: Prueba LED en la que se ve el microcontrolador con el LED integrado encendido el código de prueba proporcionado por el IDE de Arduino el cual se puede ver en Código de ejemplo de Arduino para el control de un LED. Este se queda a la espera de que algún dispositivo se conecte con el microcontrolador y este, una vez conectado queda a la espera de recibir una señal para encender o apagar el LED. Por otro lado, el código que se ejecuta en la computadora es una simplificación del código utilizado para realizar la conexión con el cubo y el cual una vez conectado, escribiendo un “2” por consola manda una señal al dispositivo.

El funcionamiento de esta prueba es el siguiente, primero se realiza la conexión y la búsqueda del servicio encargado del LED, así como su característica la cual debe tener propiedades de escritura y lectura. Una vez encontrados, este entra en un bucle infinito donde si se presiona “F”, se termina la ejecución, si se presiona “1” se lee el valor de la característica y si se presiona “2”, se escribe en la característica el valor 1 el cual el Arduino interpreta como que se debe encender o apagar el LED.

Ilustración 110: Prueba LED en la que se ve el microcontrolador con el LED integrado encendido

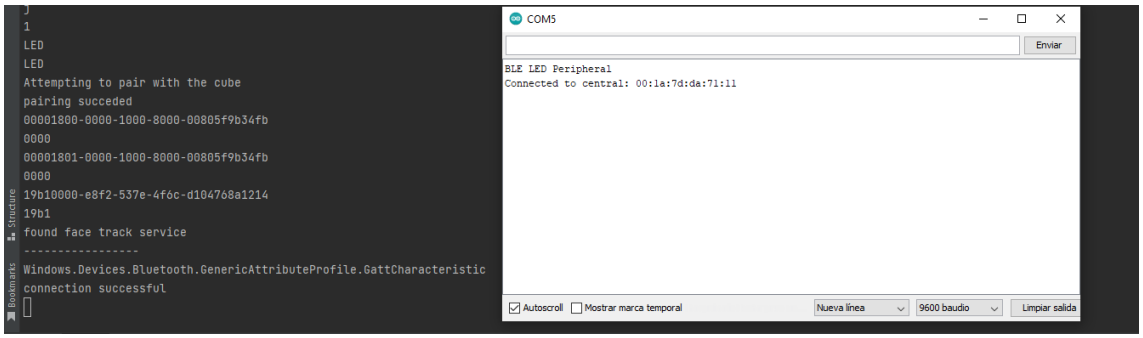


Ilustración 111: conexión entre Arduino y proceso

En Ilustración 111: conexión entre Arduino y proceso se puede ver a la derecha, en el puerto serial aparece la dirección de la computadora, lo cual significa que la conexión bluetooth se ha realizado con éxito y, por otro lado, a la izquierda se puede ver que los servicios y características bluetooth han sido encontradas por el proceso.

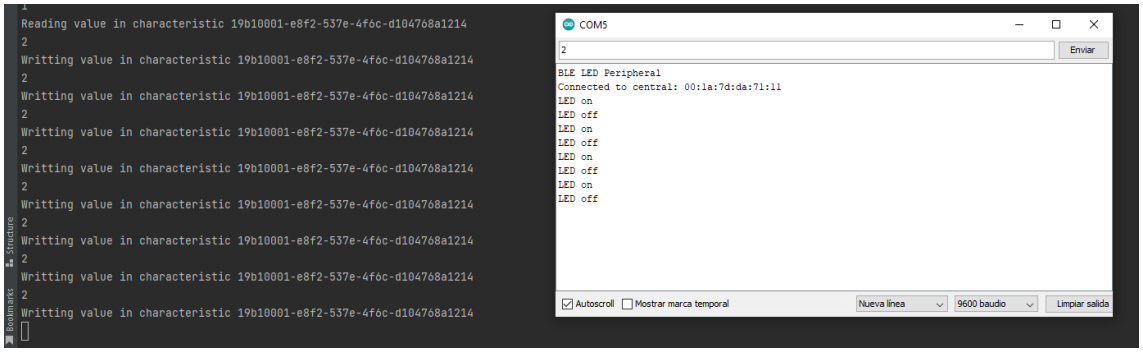


Ilustración 112: respuesta con información del puerto serial

En Ilustración 112: respuesta con información del puerto serial se pue de ver a la izquierda la entrada estándar del proceso donde se ha introducido repetidas veces el valor “2” y a la izquierda se puede ver la respuesta del microcontrolador a través del puerto serial.

El resultado de la prueba fue el esperado, este se puede ver en [47]. Las complicaciones fueron principalmente a la hora de configurar el microcontrolador con el IDE de Arduino y a la hora de encontrar ejemplos acerca a del uso de Bluetooth con este microcontrolador en específico.

6.2 Prueba Circuito con vibración

El objetivo de esta prueba era probar una aproximación del circuito final añadiendo un motor de vibración de 3.3V, como se puede ver en la parte inferior izquierda de Ilustración 113: Conexiones del circuito en la segunda prueba para comprobar que este funciona correctamente.

Esta prueba parte de la prueba anterior ya que el código del ejecutable es idéntico y al código del microcontrolador se ha modificado para adecuarse a las necesidades de esta prueba.

Por un lado, está el LED interno verde que nos indicará que se realizó la conexión de forma correcta y por otro lado está el motor de vibración el cual se activará al recibir una señal. La señal se manda desde el ejecutable (escribiendo un “2” por la consola) este activará o desactivará la vibración.

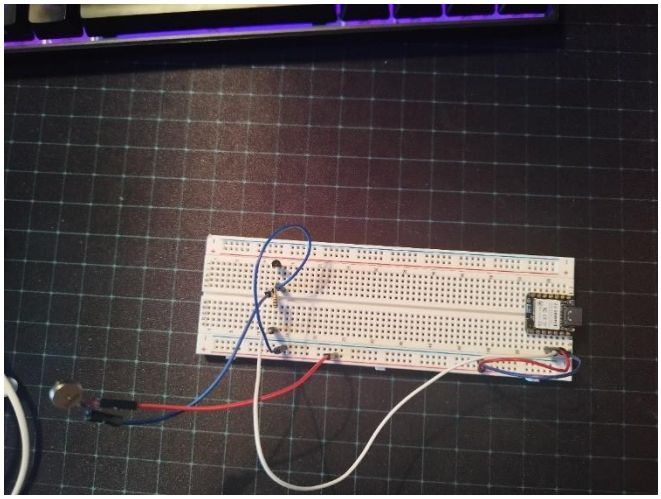


Ilustración 113: Conexiones del circuito en la segunda prueba

El resultado fue exitoso como se puede ver en [48] ya que el objetivo era poder activar a voluntad el motor de vibración, así como comprobar el buen funcionamiento de este, sin embargo, a la hora de realizar esta prueba tuve varios problemas relacionados con el propio dispositivo y con las conexiones del circuito físico en sí mismo. Respecto al dispositivo, debido a que la documentación resultaba un poco confusa fue un poco lioso el hecho de usar los puertos y las constantes que se refieren a los diferentes LEDs internos. Y respecto a el circuito físico, tuve problemas debido a que los contactos no son fijos y por lo tanto en ciertos momentos no hacían contacto por lo que no sabía si lo que causaba que el circuito no reaccionara era problema del código, la fuente de energía o las conexiones.

6.3 Prueba Circuito con sensor de agitación y matriz de LEDs con circuito soldado

El objetivo de esta prueba (Ilustración 46) era el de comprobar el buen funcionamiento tanto de la matriz de LEDs como de el sensor de agitación como del código desarrollado. El código del microcontrolador es el código que se explica en el apartado 5.5.2.3 Código

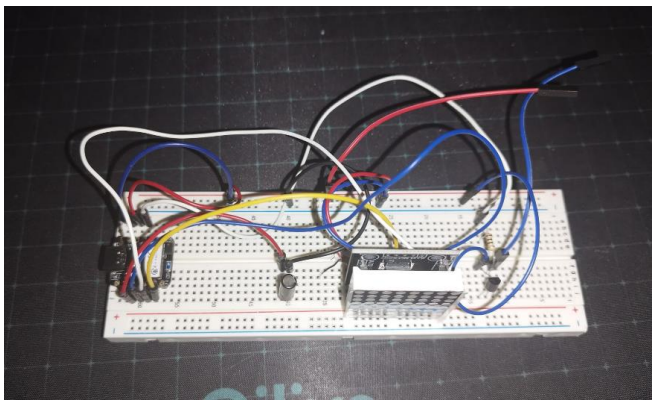


Ilustración 114: Circuito de prueba 3 en bread board en el que se aprecia el circuito con todos los componentes conectados.

En Ilustración 114: Circuito de prueba 3 en bread board se puede observar que se realizó en primera instancia el circuito en la BreadBoard del Arduino, sin embargo, Debido a la fragilidad de las conexiones resultó imposible hacer las pruebas en este estado. Por lo que se tuvieron que posponer hasta tener el circuito soldado.

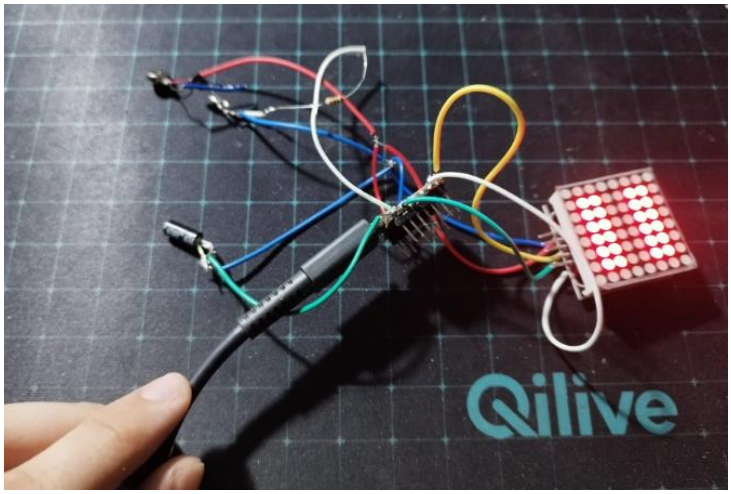


Ilustración 115:Circuito de prueba 3 soldado

Gracias a la soldadura se pudo comprobar que tanto el código como todos los componentes funcionaban de la forma esperada como se puede ver en [49], pero sin embargo debido a que era mi primera vez soldando, la soldadura resultante no es de muy buena calidad y como consecuencia cada cierto tiempo se soltaba algún cable teniéndolo que soldar de nuevo.

6.4 Prueba final dentro de la pieza

El objetivo de esta prueba final era la de comprobar que el resultado era funcional dentro de la caja en la que se introdujo y se fijaron con Loctite los componentes a la superficie.

Ilustración 116: prueba 4 dentro de pieza

Como se puede comprobar en el resultado en [50] tanto la matriz de LEDs como el motor de vibración funcionan con buenos resultado, sin embargo, el sensor de agitación no funciona como debería ya que tan solo se activa cuando se toca directamente y no con la agitación. Esto se puede deber a diversos motivos, el primero es una conexión pobre del circuito y la segunda, una opción que se me ocurre es que sea un fallo en el planteamiento de la conexión de este componente. Pero debido a que, en pruebas anteriores, este componente daba resultados aceptables, con lo que a este componente respecta la prueba es inconcluyente.

7. Estudio económico

En este apartado se estimará el coste de la realización del proyecto teniendo en cuenta tanto el coste de los componentes utilizados, como el coste de tiempo empleado en el proyecto.

7.1 Estimación del coste del Hardware

7.1.1 Estimación del coste de los componentes

En este apartado se enumeran todos los dispositivos presentes en el proyecto final, así como el lugar donde se pueden obtener y su coste el cual se puede ver en Tabla 7: estimación de coste de los componentes.

Componente	Cantidad	Enlace de compra	Precio
Cubo Giiker I3	1	Enlace	30.45 €
Seeed Studio XIAO nRF52840 Sense	1	Enlace	15.99 €
Matriz de LEDs 8x8	1	Enlace	6 €
Sensor de agitación	1	Enlace	0.8 €
Motor de vibración	1	Enlace	0.32 €
Kit de cableado	1	Enlace	6.99 €
BreadBoard	1	Enlace	8.99 €
Transistor NPN	2	Enlace	1.3€
Kit de soldadura Bakku	1	Enlace	63.56 €
Total			134.4 €

Tabla 7: estimación de coste de los componentes

7.1.2 Estimación coste del trabajo humano en hardware

En este apartado se aproximará el coste teniendo en cuanta las diferentes tareas realizadas en el proyecto.

Tarea	Responsable	Precio por hora	Horas dedicadas	Total
Diseño del circuito	Ingeniero de computadores	13.59 €/h	100h	1359€
Montaje del circuito	Técnico electrónica	11.98 €/h	50h	599€
Documentación del circuito	Product manager	19.23 €/h	40h	769.2€
Total				2727.2 €

Tabla 8: tabla de estimación del trabajo humano en hardware

7.1.3 Estimación Total de coste de hardware

En este apartado se muestra la suma del coste de componentes junto con la suma de las horas del trabajo realizado.

Estimaciones	Coste
Estimación componentes	134.4 €
Estimación de trabajo realizado	2727.2€
Total	2861.6€

Tabla 9: estimación del coste de la parte de hardware

7.2 Estimación del coste del Software

Para realizar esta estimación se tendrá en cuanta todas las tareas que han requerido desarrollo de código para este proyecto.

7.2.1 Estimación del coste del trabajo humano en software

En este apartado se aproximará el coste teniendo en cuanta las diferentes tareas realizadas en el proyecto.

Tarea	Responsable	Precio por hora	Horas dedicadas	Total
Tiempo dedicado a documentarse.	Ingeniero de computadores	13.59 €/h	50h	13.59 €/h
Diseño de conexiones de la interfaz de cubo de Rubik	Ingeniero de computadores	13.59 €/h	40h	542.8€
Diseño de conexiones del microcontrolador con UNITY	Desarrollador de videojuegos	17 €/h	50h	850 €
Ingeniería inversa del cubo de Rubik	Ingeniero de computadores	13.59 €/h	30h	407.1€
Implementación de la conectividad entre proceso intermedio y cubo	Ingeniero de computadores	13.59 €/h	90h	1221.3€
Implementación de la interfaz en UNITYpara el cubo de Rubik	Desarrollador de videojuegos	17 €/h	150h	2550
Diseño de conexiones del circuito	Ingeniero de computadores	13.59 €/h	40h	542.8€
Implementación del código en Arduino	Ingeniero de computadores	13.59 €/h	70h	949.9€
Implementación del código en UNITY	Desarrollador de videojuegos	13.59 €/h	70h	949.9€
Documentación del trabajo realizado en la interfaz del cubo	Ingeniero de computadores	13.59 €/h	50h	13.59 €/h
Documentación del trabajo realizado en el circuito	Ingeniero de computadores	13.59 €/h	50h	678.5€
Pruebas realzadas con la interfaz de cubo de Rubik	Desarrollador de videojuegos	17 €/h	20h	340 €
Pruebas realizadas con el circuito	Ingeniero de computadores	13.59 €/h	30h	407.7€
Total				10,118.5€

Tabla 10: estimación de coste de software

7.2.2 Estimación del coste de licencias

En este apartado se aproximará el coste de las diferentes licencias utilizadas para la realización del proyecto.

Licencia	Precio
UNITY	0€
IDE Arduino	0€
C-Lion	299€
Rider	149€
Total	448€

Tabla 11: estimación de coste de las licencias

7.2.3 Estimación total del coste de software

En este apartado se muestra la suma del coste de las licencias junto con la suma de las horas del trabajo realizado.

Licencia	Precio
Estimación trabajo realizado	10,118.5€
Estimación licencias	448€
Total	10,566.5€

Tabla 12: estimación de coste de las licencias

7.3 Estimación del coste total del proyecto

En este apartado se muestra la suma del coste total del proyecto.

Categoría	Precio
Hardware	2,861.6€
Software	10,566.5
Total	13,428.5€

Tabla 13: estimación de coste de las licencias

8. Conclusiones

Durante el último año, se ha dedicado un gran esfuerzo al desarrollo de este proyecto en paralelo con el juego creado para el Trabajo de Fin de Grado de Videojuegos. Durante este proceso, se ha realizado un exhaustivo estudio y aprendizaje sobre la tecnología Bluetooth, así como sobre los distintos microcontroladores y componentes utilizados en el proyecto. Además, se ha profundizado en las capacidades avanzadas que ofrece UNITY, explorando más allá de sus aplicaciones básicas. Este proceso de investigación y aprendizaje ha sido enriquecedor en cuanto a las cosas nuevas que antes de empezar veía muy complicadas o casi imposibles.

El proyecto empezó con la idea de cómo meter un circuito entero dentro de una de las piezas del cubo de Rubik, por ello se hizo una exploración en el mercado y encontramos una posibilidad que cumplía los requisitos, el µduino. Y hice los esquemas de conexión incluyendo el módulo bluetooth más pequeño que pude encontrar, junto con el motor de vibración. Sin embargo, más tarde descubrí que este microcontrolador formaba parte de un proyecto el cual dejó de estar en activo muchos años atrás y por lo tanto era inaccesible.

Tiempo después, encontré de casualidad el microcontrolador XIAO nRF52840 (sense) el cual parecía casi caído del cielo ya que este tenía todo lo que necesitaba y más, es decir, incorporaba bluetooth, micrófono, NFC, giroscopio… por lo que este microcontrolador ahorraba el tener que complicar el circuito, tan solo era necesario estudiar el funcionamiento de este.

Sin embargo, nos encontramos con un contratiempo logístico, ya que el dispositivo que se había pedido no llegó, a pesar de haberse adquirido en la tienda oficial. Como resultado, esta parte del proyecto se vio detenida durante varios meses. No obstante, se tomó la decisión de volverlo a pedir en otra tienda, y esta vez sí se recibió, lo que permitió retomar el avance en el desarrollo del proyecto.

Durante este tiempo de parón, se estuvo trabajando en la interfaz del cubo de Rubik con UNITY, en esta fase, Investigué acerca del protocolo bluetoothLE ya que nunca se había tocado en ninguna asignatura de la carrera. Sin embargo, me pareció muy interesante, así como sencillo su funcionamiento. Gracias a los conocimientos adquiridos sobre BluetoothLE pude empezar a hacer ingeniería inversa con el cubo, ya que este no está pensado para ser usado más allá que en la aplicación oficial del fabricante.

Durante el proceso de ingeniería inversa me surgieron varios problemas como explico en el apartado Conexión con el dispositivo del punto 5.3.4 Fases y funcionamiento del proyecto, ya que tras encontrar el servicio y la característica encargada de trasmitir la información sobre los giros, esta venía en un “churro” de bytes sin sentido, sin embargo, gracias a que otra gente se había enfrentado a este problema en otros entornos se descubrió la solución y por fin pude recibir información sobre los giros e interpretarla.

Más tarde cuando ya tenía suficientemente avanzado el juego del otro TFG, me puse a idear una forma de conectar el proceso que había creado con UNITY. Para ello probé muchas opciones, sin embargo, no fue hasta que revisé la teoría de sistemas operativos que se me ocurrió que podía lanzar el proceso comunicante desde UNITY y redirigir la entrada y la salida estándar de forma que tuviera control total de todo desde UNITY.

Posteriormente me puse a implementar la interfaz en UNITY ya que esta no solo debía poder aceptar los giros básicos del cubo, sino que también los giros dobles, esto supuso un aumento en la complejidad del proyecto, pero sin embargo al final se completó quedando totalmente funcional.

Finalmente, ya en la fase final del desarrollo del juego y cuando todos los componentes hubieron llegado, me puse al montaje del circuito final, pero, sin embargo, el nuevo controlador, así como los componentes no cabían físicamente en una pieza del cubo de Rubik (por lo menos con los medios disponibles) por lo que se abandonó la idea de introducirlo en una pieza y se quedó en un prototipo de lo que debería tener una pieza. Por otro lado, gracias a que el espacio ya no era una limitación, permitió que se pudieran incorporar componentes más sofisticados que enriquecían la experiencia de uso como es la matriz de LEDs la cual desde mi punto de vista es un pilar fundamental del circuito.

Tras tener todo el circuito llegó la hora de soldarlo, lo cual fue un verdadero gran reto ya que nunca había soldado, por lo que la soldadura final no era de gran calidad y en parte afectando al resultado final ya que algunas conexiones se soltaban con relativa facilidad teniendo que volver a soldarlo y ensuciando las conexiones. Sin embargo, finalmente se obtuvo un resultado aceptable con el inconveniente de que el sensor de vibraciones dejó de funcionar correctamente.

Tras realizar todas las pruebas pertinentes me puse a documentar y recopilar todo el trabajo realizado, así como a hacer un estudio económico del proyecto, acabando el cual se ve reflejado en esta memoria.

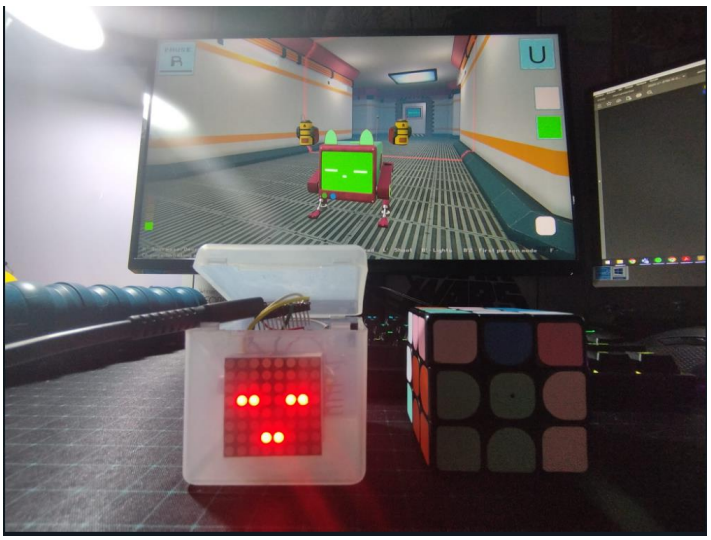


Ilustración 117: foto del resultado final del proyecto.

8.1 Retrospectiva de objetivos

Al principio de la memoria en el primer apartado se marcaron unos objetivos, en este punto se van a analizar punto por punto:

- **Desarrollar una interfaz para UNITY que sea capaz de detectar los movimientos de un cubo de Rubik. De forma que esta se pueda usar a gusto del desarrollador y tenga un funcionamiento robusto. Por ello debe poder cumplir los siguientes propósitos:**
    - **Conectividad con la mayor parte de Smart Cubes del mercado, pero especialmente con el cubo Giiker i3**: Como se puede ver en el apartado 5.3 Conexión con el cubo la conexión con el cubo Giiker i3 fue exitosa, sin embargo, al no disponer de otros cubos con los que realizar pruebas no se puede saber si funcionaría con otros modelos.

- **Ser capaz de registrar y notificar todos los posibles giros realizados en el cubo de Rubik:** como se puede ver en el apartado 5.4.4 Interpretación de los datos recibidos por el ejecutable , los giros del cubo son recibidos e interpretados por lo que este objetivo se cumpliría en su totalidad.
- **Tener un funcionamiento robusto, es decir que a pesar de que se pierda la conexión sea capaz de reconectarse:** como se puede ver en el apartado 5.4.5 Mejoras para su usabilidad se han implementado mecanismos para la reconexión en caso de pérdida de conexión.

- Diseñar un accesorio complementario para el cubo de Rubik que se integre perfectamente con la interfaz previamente mencionada y esté especialmente creado para el juego desarrollado en el Trabajo de Fin de Grado de Diseño y Desarrollo de Videojuegos. Este accesorio debe cumplir con las siguientes capacidades y funcionalidades:
  - **Debe poder caber en una pieza del cubo de Rubik**: este objetivo no se ha podido cumplir como se explica en 5.5 Sistema de control debido a que no se tenían los medios ni los componentes para hacerlo.
  - **El accesorio debe ser capaz de mostrar a través de la matriz de LEDs diferentes estados del juego:** como se puede ver en 5.5.6.1¿Qué papel tiene cada componente?, la matriz de LEDs es capaz de representar múltiples estados del juego**.**
  - **Debe dar respuesta háptica de vibración a eventos concretos del juego:** como se demostró en las pruebas, en el apartado 6.2 Prueba Circuito con vibración en adelante, el circuito es capaz de reaccionar a eventos del juego otorgándole retroalimentación háptica.
  - **Debe ser capaz de interactuar con el juego mediante un interruptor de agitación**: Como se puede ver en la prueba 6.3 Prueba Circuito con sensor de agitación y matriz de LEDs con circuito soldado el juego registra la agitación sin embargo por motivos inconcluyentes, este empezó a funcionar mal como se puede ver en 6.4 Prueba final dentro de la pieza.

En definitiva, se han cumplido los objetivos principales del proyecto, aunque se han encontrado algunos inconvenientes por el camino.

8.2 Trabajos Futuros

De cara al futuro hay aspectos del proyecto que se pueden mejorar y ampliar. Por lo que se trabajarán en ellos para poder conseguir una experiencia de uso más plena.

En primer lugar, en cuanto a la interfaz del cubo de Rubik, me gustaría abrirlo al público creando un Asset gratuito en Github para que de esta forma se pudieran crear proyectos con el ampliando el catálogo de juegos controlables con un cubo de Rubik, Por otro lado, tengo pensado documentarlo en una página web para añadirlo a mi portafolio. En cuanto a funcionalidades, me gustaría añadir más cubos a la lista de Smart Cubes compatibles con la interfaz para de esta forma hacerlo más accesible al público ya que el cubo Giiker i3 no es una versión que posea mucha gente debido a que fue la primera versión que salió al mercado.

En cuanto al circuito, en primer lugar, me gustaría, ya teniendo conocimientos sobre soldaduras, volver a realizar el montaje, pero de forma más limpia y organizada, ya que esto ha sido un imprevisto que tenía controlado. Por otro lado, me gustaría explotar las funcionalidades del microcontrolador XIAO nRF52840 (sense) ya que de las que dispone tan solo aprovecho su

funcionalidad de bluetooth, sin embargo, en un futuro me gustaría darle uso al giroscopio, así como al micrófono.

8.3 Conclusiones personales

La realización de este trabajo ha sido muy enriquecedora ya que me ha permitido descubrir y aprender sobre campos nuevos de los que en un pasado me hubiera dado miedo enfrentarme, pero sin embargo el hecho de que este trabajo haya tratado sobre algo que me motiva me ha mantenido interesado en todos los aspectos del proyecto, desde la programación que es la parte que personalmente más me gusta hasta la circuitería de la cual tenía bastante miedo al comenzar el proyecto.

Sin embargo, a medida que avanzaba el proyecto, y se iban consiguiendo resultados mi motivación para seguir tanto con este trabajo como con el juego, a pesar de las tardes y las noches dedicadas para conseguir sacarlo adelante. Pero como ya he mencionado, solo con ver los resultados y la opinión de la gente ya valía la pena el esfuerzo.

Finalmente documentar este proyecto también ha sido una experiencia en si misma ya que nunca había tenido que hacer una memoria de estas dimensiones, por lo cual también me ha servido como experiencia.

Bibliografía

[1] G. Lucas, F. Jeremy y T. Koki, «Bluetooth Cubing,» [En línea]. Available: https://bluetooth.cubing.net/.

[2] m. baraink, «Mandos de videojuegos: La historia del joystick,» [En línea]. Available: https://www.neoteo.com/la-historia-del-joystick-parte-1/.

[3] Nintendo fandom , «Mando de NES,» [En línea]. Available: https://nintendo.fandom.com/es/wiki/Mando_de_NES. [Último acceso: 12 julio 2023].

[4] Nintenderos, «Descubriendo Virtual Boy: los motivos por los que se considera el mayor fracaso de Nintendo,» [En línea]. Available: https://www.nintenderos.com/2022/11/descubriendo-virtual-boy-los-motivos-por-los-que-se-considera-el-mayor-fracaso-de-nintendo/. [Último acceso: 2023 julio 12].

[5] Next generation, «The ultra 64 Joypad,» p. 38, 14 febrero 1996.

[6] M. Oliveras, «Sega Dreamcast – La historia de un triunfo y de un fracaso,» 12 julio 2023. [En línea]. Available: https://www.yeabitinformatica.com/historia/sega-dreamcast/.

[7] S. Satterfield, «What's inside the GameCube?,» [En línea]. Available: https://www.zdnet.com/article/whats-inside-the-gamecube/. [Último acceso: 12 julio 2023].

[8] Steamworks, «Mando Microsoft Xbox 360,» [En línea]. Available: https://partner.steamgames.com/doc/features/steam_controller/device/x360_controller?l=spanish. [Último acceso: 12 julio 2023].

[9] Valve, «Steam Controller,» [En línea]. Available: https://store.steampowered.com/app/353370/Steam_Controller/. [Último acceso: 12 Julio 2023].

[10] nintendo fandom, «Wii Remote,» [En línea]. Available: https://nintendo.fandom.com/es/wiki/Wii_Remote. [Último acceso: 12 julio 2023].

[11] Nintendo, «Especificaciones técnicas,» [En línea]. Available: https://www.nintendo.com/es-mx/switch/tech-specs/. [Último acceso: 12 julio 2023].

[12] «Meta, especificaciones técnicas,» [En línea]. Available: https://www.meta.com/es/quest/products/quest-2/. [Último acceso: 12 julio 2023].

[13] Fanatec, «CSL DD F1 ESPORTS STARTER KIT FOR PC,» Fanatec , [En línea]. Available: https://fanatec.com/eu-en/ready-to-race/csl-dd-f1-esports-starter-kit-for-pc?gclid=Cj0KCQjwnrmlBhDHARIsADJ5b_kf61eLGHVTMVuc1duiVs5qqvghbkGYVJ7jlvnKwetJGqScXneMVNUaAg0jEALw_wcB. [Último acceso: 12 julio 2023].

[14] Hi-Fi pro, «Los mejores joysticks para PC: Consejos y guía de compra,» [En línea]. Available: https://www.tvhifipro.com/blog/los-mejores-joysticks-para-pc-consejos-y-guia-de-compra/. [Último acceso: 12 julio 2023].

[15] E. S. Fuentes, «Vandal, Guitar Hero. La fiebre de las guitarras de plástico,» 14 10 2020. [En línea]. Available: https://vandal.elespanol.com/vandalgamemusic/guitar-hero-la-fiebre-de-las-guitarras-de-plastico. [Último acceso: 12 julio 2023].

[16] Nintendo, «Nintendo Labo,» [En línea]. Available: https://www.nintendo.es/Nintendo-Labo/Nintendo-Labo-1328637.html. [Último acceso: 12 julio 2023].

[17] Wikipedia, «R.O.B.,» [En línea]. Available: https://es.wikipedia.org/wiki/R.O.B.. [Último acceso: 12 julio 2023].

[18] Wikipedia, «Game boy camera,» [En línea]. Available: https://es.wikipedia.org/wiki/Game_Boy_Camera.

[19] Pokemon. fandom, «Pokemon walker,» [En línea]. Available: https://pokemon.fandom.com/es/wiki/Pok%C3%A9walker. [Último acceso: 12 julio 2023].

[20] IGN, «First Look: Rumble Pak DS,» 18 octubre 2005. [En línea]. Available: https://www.ign.com/articles/2005/10/18/first-look-rumble-pak-ds. [Último acceso: 2023 julio 12].

[21] Wikipedia, «Cubo de Rubik,» [En línea]. Available: https://es.wikipedia.org/wiki/Cubo_de_Rubik.

[22] J. Friedrich. [En línea]. Available: http://www.ws.binghamton.edu/fridrich/. [Último acceso: 12 julio 2023].

[23] WCA, [En línea]. Available: https://www.worldcubeassociation.org/competitions/WC1982. [Último acceso: 2023 julio 12].

[24] WCA, «WCA,» [En línea]. Available: https://www.worldcubeassociation.org/. [Último acceso: 2023 julio 12].

[25] Kubekings, «Monster Go AI 3x3,» [En línea]. Available: https://kubekings.com/cubos-de-rubik-3x3/monster-go-ai-3x3.html.

[26] KuebeKings, «GAN 356 i Carry,» [En línea]. Available: https://kubekings.com/cubos-de-rubik-3x3/gan-356-i-carry.html#/color-stickerless.

[27] KubeKings, «GAN I3 3x3,» [En línea]. Available: https://kubekings.com/cubos-de-rubik-3x3/gan-i3-3x3.html#/color-stickerless.

[28] Kubekings, «Rubik's Connected 3x3,» [En línea]. Available: https://kubekings.com/cubos-de-rubik-3x3/rubik-s-connected-3x3.html.

[29] Kubekings, «GoCube 3x3,» [En línea]. Available: https://kubekings.com/cubos-de-rubik-3x3/gocube-3x3.html.

[30] Speedcubeshop, «MoYu WeiLong AI 3x3 Bluetooth Smart Cube,» [En línea]. Available: https://speedcubeshop.com/collections/smart-cubes/products/moyu-weilong-ai-3x3-bluetooth-smart-cube?variant=39460856823921.

[31] Giiker, «About Us,» [En línea]. Available: https://giiker.com/pages/about-us.

[32] Giiker, «SUPERCUBE user manual,» [En línea]. Available: https://cdn.shopify.com/s/files/1/0609/5611/5196/files/KM.pdf?v=1661327711.

[33] B. Karl y W. Matt, «Bluetooth GATT client,» Microsoft, 2022. [En línea]. Available: https://learn.microsoft.com/en-us/windows/uwp/devices-sensors/gatt-client.

[34] B. Karl y W. Matt, «Bluetooth Low Energy,» Microsoft, 21 10 2022. [En línea]. Available: https://learn.microsoft.com/en-us/windows/uwp/devices-sensors/bluetooth-low-energy-overview.

[35] T. Kevin, C. Cufí, Akiba y D. Robert, «Orelly, Getting Started with Bluetooth Low Energy,» [En línea]. Available: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html.

[36] B. SIG, «Bluetooth documentation,» 17 10 2022. [En línea]. Available: https://btprodspecificationrefs.blob.core.windows.net/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf.

[37] J. Fleischman, «Github,» 6 julio 2018. [En línea]. Available: https://github.com/jfly/giiker/blob/master/giiker.py.

[38] «Giiker cube api. Configuration from State,» Stack overflow, 10 june 2020. [En línea]. Available: https://stackoverflow.com/questions/62310356/giiker-cube-api-configuration-from-state/62319004#62319004.

[39] Unity, «Soporte multiplataforma de Unity,» [En línea]. Available: https://unity.com/es/solutions/multiplatform. [Último acceso: 12 julio 2023].

[40] Desarrollo español de videojuegos, «Libro Blanco del Desarrollo Español de Videojuegos 2021,» 4 mayo 2022. [En línea]. Available: https://dev.org.es/images/stories/docs/libro%20blanco%20del%20desarrollo%20espanol%20de%20videojuegos%202021.pdf.

[41] D. CANDIL, «VIDA EXTRA,» 21 Febrero 2014. [En línea]. Available: https://www.vidaextra.com/industria/unity-el-motor-de-desarrollo-capaz-de-partir-la-historia-de-los-videojuegos-en-dos.

[42] Arduino, «What is Arduino?,» [En línea]. Available: https://www.arduino.cc/en/Guide/Introduction.

[43] Arduino, « Descarga de software,» [En línea]. Available: https://www.arduino.cc/en/software.

[44] D. Chadwick, «crowd supply,µduino,» [En línea]. Available: https://www.crowdsupply.com/uduino/uduino. [Último acceso: 12 julio 2023].

[4 Seeedstudido, «Seeed Studio XIAO SAMD21(Seeeduino XIAO),» [En línea]. Available:
5] https://www.seeedstudio.com/Seeeduino-XIAO-Arduino-Microcontroller-SAMD21-Cortex-M0+-p-4426.html. [Último acceso: 12 julio 2023].

[4 SeeedStudio, «Getting Started with Seeed Studio XIAO nRF52840 (Sense),» [En línea].
6] Available: https://wiki.seeedstudio.com/XIAO_BLE/. [Último acceso: 12 julio 2023].

[4 C. C. Cano, «Youtube, prueba 1,» [En línea]. Available:
7] https://www.youtube.com/shorts/UFIWqQV4cfM.

[4 C. C. Cano, «youtube, prueba 2 con motor de vibración,» [En línea]. Available:
8] https://youtube.com/shorts/tj-htoN5aj8?feature=share.

[4 C. C. Cano, «Youtube prueba 4,» [En línea]. Available: https://youtu.be/jqcn3WYsNaA.
9]

[5 C. C. Cano, «youtube, prueba 5,» [En línea]. Available: https://youtu.be/WviE-Br9mak.
0]

[5 c. d. Wikipedia, «Metadatos,» Wikipedia, La enciclopedia libre., 30 septiembre 2022. [En
1] línea]. Available: https://es.wikipedia.org/w/index.php?title=Metadatos&oldid=146284194.

[5 Wikipedia,
2] «https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)#Historia:~:text=Unity%20(motor%20de%20videojuego),-Ir%20a%20la,» 26 dic 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)#Historia:~:text=Unity%20(motor%20de%20videojuego),-Ir%20a%20la. [Último acceso: 16 ene 2023].

[5 D. E. García, «Open Webinars,» 13 Junio 2019. [En línea]. Available:
3] https://openwebinars.net/blog/origen-evolucion-versiones-unity/.

Anexo

Código de ejemplo de Arduino para el control de un LED

```
/*
  LED Control

  This example scans for Bluetooth® Low Energy peripherals until one with the advertised service
  "19b10000-e8f2-537e-4f6c-d104768a1214" UUID is found. Once discovered and connected,
  it will remotely control the Bluetooth® Low Energy peripheral's LED, when the button is pressed or released.

  The circuit:
  - Arduino MKR WiFi 1010, Arduino Uno WiFi Rev2 board, Arduino Nano 33 IoT,
    Arduino Nano 33 BLE, or Arduino Nano 33 BLE Sense board.
  - Button with pull-up resistor connected to pin 2.

  You can use it with another board that is compatible with this library and the
  Peripherals -> LED example.

  This example code is in the public domain.
*/
```

```
18.
19. #include <ArduinoBLE.h>
20.
21. // variables for button
22. const int buttonPin = 2;
23. int oldButtonState = LOW;
24.
25. void setup() {
26.   Serial.begin(9600);
27.   while (!Serial);
28.
29.   // configure the button pin as input
30.   pinMode(buttonPin, INPUT);
31.
32.   // initialize the Bluetooth® Low Energy hardware
33.   BLE.begin();
34.
35.   Serial.println("Bluetooth® Low Energy Central - LED control");
36.
37.   // start scanning for peripherals
38.   BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
39. }
40.
41. void loop() {
42.   // check if a peripheral has been discovered
43.   BLEDevice peripheral = BLE.available();
44.
45.   if (peripheral) {
46.     // discovered a peripheral, print out address, local name, and advertised service
47.     Serial.print("Found ");
48.     Serial.print(peripheral.address());
49.     Serial.print(" '");
50.     Serial.print(peripheral.localName());
51.     Serial.print("' ");
52.     Serial.print(peripheral.advertisedServiceUuid());
53.     Serial.println();
54.
55.     if (peripheral.localName() != "LED") {
56.       return;
57.     }
58.
59.     // stop scanning
60.     BLE.stopScan();
61.
62.     controlLed(peripheral);
63.
64.     // peripheral disconnected, start scanning again
65.     BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
66.   }
67. }
68.
69. void controlLed(BLEDevice peripheral) {
70.   // connect to the peripheral
71.   Serial.println("Connecting ...");
72.
73.   if (peripheral.connect()) {
74.     Serial.println("Connected");
75.   } else {
76.     Serial.println("Failed to connect!");
77.     return;
78.   }
79.
80.   // discover peripheral attributes
81.   Serial.println("Discovering attributes ...");
82.   if (peripheral.discoverAttributes()) {
83.     Serial.println("Attributes discovered");
84.   } else {
85.     Serial.println("Attribute discovery failed!");
86.     peripheral.disconnect();
87.     return;
88.   }
```

```
 89.
 90.   // retrieve the LED characteristic
 91.   BLECharacteristic ledCharacteristic = peripheral.characteristic("19b10001-e8f2-537e-
4f6c-d104768a1214");
 92.
 93.   if (!ledCharacteristic) {
 94.     Serial.println("Peripheral does not have LED characteristic!");
 95.     peripheral.disconnect();
 96.     return;
 97.   } else if (!ledCharacteristic.canWrite()) {
 98.     Serial.println("Peripheral does not have a writable LED characteristic!");
 99.     peripheral.disconnect();
100.     return;
101.   }
102.
103.   while (peripheral.connected()) {
104.     // while the peripheral is connected
105.
106.     // read the button pin
107.     int buttonState = digitalRead(buttonPin);
108.
109.     if (oldButtonState != buttonState) {
110.       // button changed
111.       oldButtonState = buttonState;
112.
113.       if (buttonState) {
114.         Serial.println("button pressed");
115.
116.         // button is pressed, write 0x01 to turn the LED on
117.         ledCharacteristic.writeValue((byte)0x01);
118.       } else {
119.         Serial.println("button released");
120.
121.         // button is released, write 0x00 to turn the LED off
122.         ledCharacteristic.writeValue((byte)0x00);
123.       }
124.     }
125.   }
126.
127.   Serial.println("Peripheral disconnected");
128. }
129.
```