MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Diseño de un juguete conectado para bebes

Autor: Alfonso Sanz Giner

Director: Ávaro Pérez Bello

Madrid

Julio de 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño de un juguete conectado para bebes

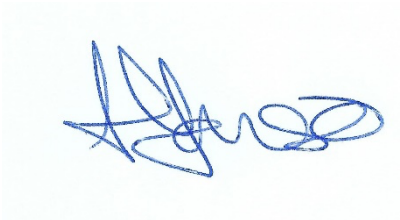
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/22 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Alfonso Sanz Giner Fecha: 19 / 08 / 2022

Autorizada la entrega del Proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Álvaro Pérez Bello Fecha: 19 / 08 / 2022

DISEÑO DE UN JUGUETE CONECTADO PARA BEBES

Autor: Sanz Giner, Alfonso.
Director: Pérez Bello, Álvaro.
Entidades Colaboradoras: ICAI – Universidad Pontificia Comillas y Altair Engineering Inc.

RESUMEN DEL PROYECTO

Palabras clave: Juguete, programación, circuito, código, Internet o Things (IoT), MQTT, HTTP, componentes, electrónica, Arduino, proyecto, construcción, prototipo…

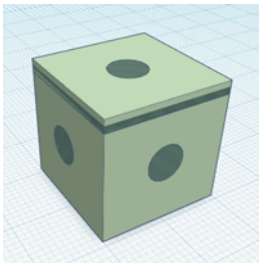
1. Introducción

Hoy en día existen innumerables juegos para bebés, pero todos ellos disponen de las mismas limitaciones de cara a su jugabilidad, variedad y duración de juego. La intención de estos juguetes es ser llamativos, con sonidos y luces, pero muy sencillos, para que sea fácil aburrirse de ellos y pasar al siguiente.

2. Definición del Proyecto

El objetivo de este Proyecto es el desarrollo de un nuevo juguete capaz de mostrar varios juegos y niveles para el entretenimiento y desarrollo cognitivo de bebes, pudiendo conectarse a través de internet con una plataforma IoT donde se puedan monitorizar los eventos y actualizarse con nuevos juegos de mayor dificultad.

3. Descripción del sistema



El juguete es una construcción cúbica de madera hueca con un agujero en cada cara para introducir los botones LED y un corte transversal para disponer de una apertura completa al interior para el resto de los componentes: microcontrolador (con capacidad Wifi), multiplexor, giroscopio/acelerómetro, altavoz…

Además de la circuitería y programación del juguete, también se hará uso de la plataforma de Altair One para la comunicación HTTP/MQTT entre el juguete y la nube, a la que enviará los eventos y recibirá instrucciones.

4. Resultados

El prototipo desarrollado es completamente funcional, tanto offline como online, pudiendo jugar a juegos de dificultad configurable, monitorizando los diferentes eventos (botones, LEDs, posición, estado, victoria/derrota…) y pudiendo jugar online ejecutando las instrucciones indicadas por la plataforma y, por tanto, continuar su desarrollo de forma ilimitada sin volverlo a conectar o reprogramar.

5. Conclusiones

El concepto desarrollado de este juguete conectado es muy versátil y – aun siendo un prototipo básico – ya dispone de un potencial muy superior a cualquier otro juego electrónico disponible en el mercado.

DESIGN OF A CONNECTED TOY FOR BABIES

Author: Sanz Giner, Alfonso.
Director: Pérez Bello, Álvaro.
Collaborating Entities: ICAI – Universidad Pontificia Comillas y Altair Engineering Inc.

PROYECT SUMMARY

Keywords: Game, programming, circuit, code, Internet o Things (IoT), MQTT, HTTP, components, electronics, Arduino, project, construction, prototype…

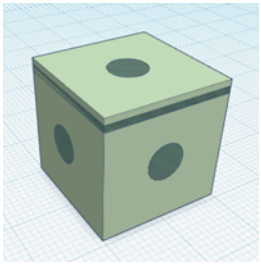
1. Introduction

Nowadays there are countless games for babies, but all of them count with the same limitations in terms of playability, variety, and duration of gameplay. The intention of these toys is to be flashy, with sounds and lights, but very simple so that it is easy to get bored of them and move on to the next one.

2. Definition of the Project

The objective of this project is the development of a new toy capable of showing several games and levels for the entertainment and cognitive development of babies, being able to connect through the internet with an IoT platform where events can be monitored and updated with new games of greater difficulty.

3. System description



The toy is a hollow wooden cubic construction with a hole on each side to insert the LED buttons and with a transversal cut to have a complete opening to the interior for the rest of the components: microcontroller (Wifi capable), multiplexer, gyroscope/accelerometer, speaker…

In addition to the circuitry and programming of the toy, the Altair One platform will also be used for HTTP/MQTT communication between the toy and the cloud, to which it will send events and receive instructions.

4. Results

The developed prototype is fully functional, both offline and online, being able to play games of configurable difficulty, monitoring the different events (buttons, LEDs, position, status, victory/defeat...) and being able to play online by executing the instructions indicated by the platform and, therefore, continue its development unlimitedly without having to reconnect it or reprogram it.

5. Conclusions

The concept developed of this connected toy is very versatile and – even though it is only a basic prototype – it already has a much higher potential than any other electronic game available on the market.

Índice de la memoria

Índice de figuras

Capítulo 1. INTRODUCCIÓN

Este documento expone el contexto en el que surge este Proyecto, los objetivos que pretende conseguir enmarcados dentro del Trabajo de Fin de Máster, la metodología utilizada para alcanzarlos, las herramientas y recursos a utilizar durante el mismo y el desarrollo realizado explicando en detalle todos los circuitos, componentes y programación involucrada.

El Proyecto pretende desarrollar un dispositivo electrónico que permita el acceso a bebés y niños a una plataforma de puzles y juegos que promueven su desarrollo cognitivo, el pensamiento lógico y la creatividad.

El dispositivo tendrá la forma de un cubo con una serie de componentes sencillos en su interior: sensores como un acelerómetro/giroscopio, un altavoz, luces y un microcontrolador o microprocesador que contendrá la programación necesaria para mostrar el problema y cuando se ha alcanzado la solución de cada nivel.

Los niveles irán avanzando en dificultad según el progreso del jugador. El dispositivo estará conectado a una red IoT de Altair para traquear dicho progreso, analizar los movimientos del bebé y poder introducir niveles nuevos o más avanzados según se requiera.

1.1 MOTIVACIÓN DEL PROYECTO

Como se ha mencionado anteriormente, la motivación detrás del Proyecto es el desarrollo de un nuevo juguete capaz de mostrar una gran cantidad y variedad de juegos y niveles para el entretenimiento y desarrollo de bebes, pero que pueda entretener a todas las edades.

Es decir, el cubo estará programado para mostrar una serie de niveles en creciente dificultad y traqueando el progreso a una red IoT.

El juguete, por tanto, solucionará las limitaciones de duración y dificultad por medio de su versatilidad a la hora de plantear problemas. Su única limitación será la cantidad de juegos que pueda almacenar la memoria disponible, pudiendo resolverse con la red IoT, permitiendo su actualización con otros niveles.

La cantidad de niveles posibles dependerá de las ideas que surjan durante su desarrollo y permitirá la escalabilidad necesaria para introducir nuevos.

A su vez, se plantea la posibilidad de que se conecten más juguetes a la plataforma IoT, disponiendo de una red de juguetes conectados, monitorizados y controlables. Estos darían lugar a una Base de Datos de eventos que registran el aprendizaje de los jugadores y permiten mejorar el sistema conjunto modificando parámetros de la configuración de los juegos.

Capítulo 2. ESTADO DE LA CUESTIÓN

Existe una gran variedad de juegos y puzles para niños para el desarrollo de su memoria o creatividad: encontrar parejas, hacer puzles con colores o música, aprender los números y letras, encajar figuras en huecos, etc.

Pero todos ellos cuentan con las mismas limitaciones: o son demasiado simples y los niños se cansan en seguida una vez que saben la solución, o son demasiado avanzados y se rinden antes de haberla alcanzado. En cualquier caso, la duración del juguete es reducida y pronto requiere su sustitución por uno nuevo para mantener la atención del niño.

A continuación, se enumeran una serie de soluciones disponibles en el mercado y que han inspirado algunas ideas del Proyecto:

- **JoyJam Electronic Music Cube** [1]: Este juguete parecido a un cubo de Rubik dispone de una matriz 2x2 de LEDs RGB y una serie de deslizaderas que giran con el objetivo de hacer coincidir los 4 colores con los de la matriz. La idea es original y – a pesar de tener solo un modo de juego – es posible que sea capaz de variar su dificultad, además de ser escalable, siendo posible construir una matriz mayor. Aun así, su carencia de sonidos o indicadores de victoria no diferencia entre los colores a asignar con el verde de haber completado el nivel.
- **Electronic Memory Game Toys** [2]: Otros juguetes cuentan con una matriz de colores, está vez con pulsadores para ejercitar la memoria, de dificultad reducida, pero sin variabilidad jugable. Aun siendo muy simple y poco original el concepto da mucho juego ya que se podría hacer una versión que vaya reduciendo progresivamente el tiempo permitido para pulsar los botones o para traquear los avances en número de botones en orden recordados.

- **Six Grid Electronic Drum Memory Game** [3]: Este dispositivo añade la idea de jugar con la música y puede ser más entretenido y sencillo que los anteriores, pero también carece de diferentes modos de juego. Al añadir una mecánica sencilla y divertida, como el sonido, se puede lograr captar mucho más la atención del bebe a la vez que mejora la compresión de los puzles.

Por otra parte, respecto a juguetes sin ninguna electrónica involucrada, ya hay infinidad de puzles clásicos y juegos de letras, números y formas [4] disponibles. Pero todos ellos cuentan con las limitaciones mencionadas: solo tienen una forma de resolverlos y eso restringe su duración a unos días o semanas a lo sumo.

Además, estos juguetes no llaman tanto la atención de los niños, las luces y sonidos son mucho más atractivas para los niños en temprana edad y no es hasta los primeros años que los retos de los puzles y el aprendizaje les pueden empezar a interesar. Es por ello que es tan común y necesario hoy en día el uso de electrónica en juguetes para bebes.

Capítulo 3. DEFINICIÓN DEL TRABAJO

En este capítulo se describen los objetivos del Proyecto, la metodología a emplear y la planificación a seguir para su desarrollo.

3.1 OBJETIVOS

Los objetivos principales del Proyecto son la construcción del prototipo e interconexión del dispositivo con todos sus componentes principales, la programación de una serie de juegos y puzles de ejemplo funcionales y la comunicación con la red local de las oficinas de Altair y sus servidores.

Concretamente los objetivos mínimos que ha de cumplir el Proyecto son:

- Construcción de un juguete novedoso, contenido en una estructura con forma de cubo, que ha de ser capaz de mostrar los puzles y detectar su solución.
- Validación del diseño y testeo por medio de uno o varios prototipos funcionales.
- Traqueo de todos los movimientos que realiza el jugador y el estado del problema, es decir, el nivel en el que está cuando se ha solucionado o si se ha equivocado.
- Conexión del dispositivo con la red de Altair y visualización de la información con una app.

Dado que a futuro se plantea la opción de control de múltiples juguetes simultáneamente, se ha de diseñar la conexión con la plataforma de Altair y organizar la programación de forma escalable y versátil para poderse incorporar nuevos juguetes idénticos o con variaciones.

3.2 METODOLOGÍA

Antes del inicio del Proyecto existirá una labor de investigación sobre diseños y juegos posibles dando lugar a una idea general que se tiene del dispositivo. En función de dicha idea se han diseñado varios juegos sencillos para tener en cuenta los componentes necesarios.

Posteriormente, se procederá a la construcción del prototipo, que se dividirá en una parte de circuitería y conexionado de los elementos y una de programación que se irá desarrollando según los componentes se vayan conectando para empezar a testearlos y familiarizarse con ellos. También será necesaria la construcción de una estructura con forma de cubo que mantenga fijos los componentes en su interior, como se muestra en la figura:

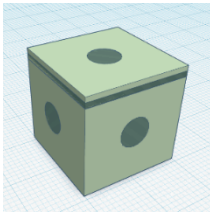


Ilustración 1: Boceto de la caja

Al ser un prototipo, se trataría de una estructura muy sencilla y meramente funcional, un cubo con un agujero para los botones en cada cara. Tiene que disponer de un corte transversal con algún mecanismo para poder abrirse y cerrarse por donde introducir los componentes, los cuales decidirán el tamaño del cubo.

Una vez diseñado un prototipo básico y funcional se podrá mejorar o realizar uno nuevo rediseñado en caso de que se desee implementar un nuevo elemento y se vayan introduciendo más juegos.

Una vez construido, testeado y programado cada uno de los componentes del juguete se requiere programar la máquina de estados en comunicación constante con la red IoT de Altair. Dicha comunicación se deberá tener en cuenta en la selección de componentes y durante la programación, de cara a su escalabilidad y el traqueo constante.

3.3 PLANIFICACIÓN

El cronograma orientativo muestra la duración planificada de cada etapa del Proyecto:

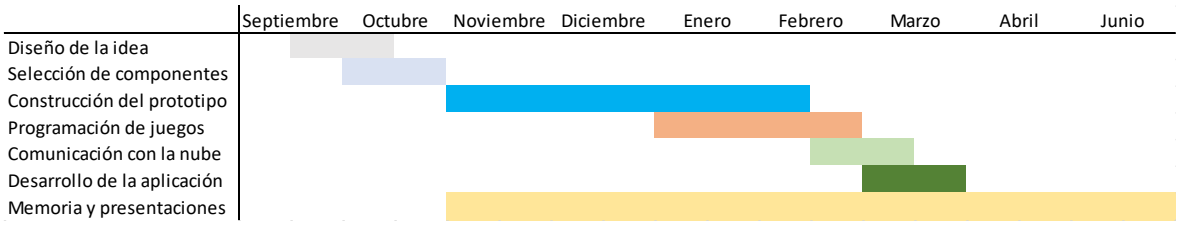


Tabla 1: Cronograma de planificación

La documentación y el desarrollo de la memoria se irá realizando en paralelo tomando las anotaciones oportunas durante la construcción y programación para, posteriormente, explicar en detalle las casuísticas que hayan surgido.

Una vez planificada la duración de cada etapa, se ha separado la construcción del prototipo en 5 subtareas principales:

- Planificación, investigación y preparación. Esta subtarea incluye la planificación general del Proyecto, las reuniones necesarias con el director o Altair, la selección de componentes y el diseño de circuitos (PCB) u organización en GitHub del código.
- Documentación: Desarrollo de los documentos intermedios de apoyo y del presente documento.
- Componentes: Testeo, circuito y programación individual de cada componente por separado, junto con sus funciones y clases para la programación conjunta final, haciendo uso de objetos.
- Construcción: Soldada, cableado, agujereado y testeo del prototipo.
- IoT: Programación y configuración de la plataforma IoT para la conexión con el juguete.

El gráfico de la derecha muestra el tiempo requerido del total del Proyecto para completar cada tarea de la planificación del Proyecto:

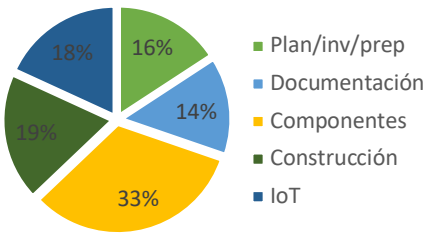


Tabla 2: Gráfico de tiempos

Capítulo 4. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Este apartado expone los diferentes medios físicos y lógicos empleados en el desarrollo del Proyecto, explicando su elección, funcionamiento y aplicación para la construcción y programación del juguete conectado.

4.1 HARDWARE: COMPONENTES ELECTRÓNICOS

La siguiente lista enumera el conjunto de componentes electrónicos que se han requerido para el desarrollo del Proyecto y que se explicarán en detalle durante este apartado:

- Microcontrolador (NodeMCU ESP8266 [5][6])
- Botones LED (Pulsadores Arcade Iluminado [7])
- Expansor de puertos I2C (MCP23017 [8])
- Acelerómetro/giroscopio (MPU6050 [9])
- Altavoz/zumbador (Mini Metal Speaker 8Ω [10])
- Estructura contenedora de madera (Artemio VIBB19 [11])
- Placa perforada PCB, pines y cableado
- Otros elementos

4.1.1 MICROCONTROLADOR

El microcontrolador escogido para el Proyecto es un NodeMCU v2, muy utilizado para proyectos de IoT debido a su reducido tamaño (comparable con un Arduino Nano), suficiente potencia para el procesamiento interrumpido para la comunicación MQTT y, sobre todo, su capacidad de conexión Wifi.

Sus especificaciones técnicas son las mencionadas a continuación:

El procesador utilizado es el ESP8266 de 32 bits a 80 MHz con 4 MB de memoria Flash y 8 KB de SRAM, cuenta con 11 pines digitales y 1 analógico, además de su conexión MicroUSB para su alimentación, cargar el código en su memoria y debuggear, aunque se puede alimentar a 4.5-10V con un voltaje operacional de 3.3V.

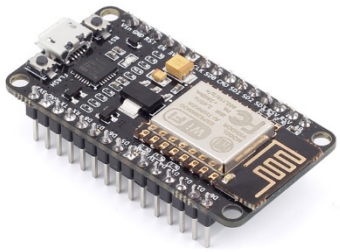


Ilustración 2: NodeMCU

A través de sus pines se puede realizar una conexión UART, SPI e I2C. Y su módulo de Wifi integrado es de categoría 802.11 b/g/n, capaz de operar en las bandas de 2.4 y 5 GHz a una velocidad de 2-4 Mbits/s, muy superior a la necesaria para una comunicación MQTT.

Todo ello con un tamaño reducido de 58mm x 32mm.

Su programación es versátil, pudiendo realizarse con varios frameworks: Non-OS SDK, RTOS SDK y – el más conocido y el que se utilizará en este Proyecto – Arduino.

En adelante se hará mención del microcontrolador como NodeMCU o ESP8266.

4.1.2 BOTONES LED

Aunque en un principio se pensó en utilizar pequeños pulsadores LED RGB (como los NeoPixel Mini Button PCB de Adafruit), al final se optó por botones más grandes y de un solo color para el primer prototipo, concretamente los clásicos botones de Arcade LED.

Ilustración 3: Botones LED

Se utilizará uno para cada cara del cubo, es decir, 6: 2 blancos para la cara superior e inferior y para los laterales uno azul, amarillo, verde y rojo en ese orden.

Estos botones son de un tamaño justo para la caja seleccionada y disponen de 4 conectores, 2 para el LED y 2 para el pulsador NO:

Ilustración 4: Botón LED

Los 2 conectores para el LED están a los lados (rojo y negro), aunque la posición del positivo negativo varía (se ha de descubrir por prueba y error, aunque no hay riesgo de fundir el LED en caso de confundirse) y puede alimentarse en un rango de 5 a 12 V. Pero, debido a la limitación del microcontrolador, se alimentarán a 3.3V generando una luz detectable y suficiente a pesar de no ser la nominal.

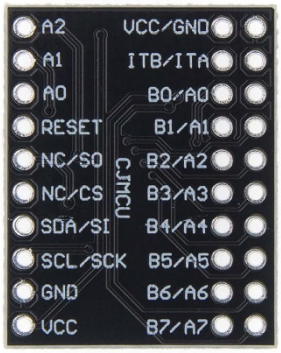
Para los 2 conectores del pulsador/conmutador NO (normalmente abierto) se conectarán a tierra por el conector inferior (COM) y al expansor de puertos – mencionado en el siguiente apartado – por el conector frontal (NO). Existen variantes de estos pulsadores que cuentan con un conector NC, pero en este caso no son necesarios.

Los pulsadores son cómodos de montar, pudiendo separar el LED del pulsador para su instalación. El diámetro exterior de la rosca es de 24 mm y el superior es de 33.5 mm permitiendo usar un tamaño intermedio de broca en ese rango para los agujeros.

Ilustración 5: Botón LED desmontado

4.1.3 EXPANSOR DE PUERTOS

Debido a la limitación de pines digitales del microcontrolador mencionado previamente y la cantidad de salidas necesarias (explicado en detalle en el *5.1.3 Cableado*) se añadió un expansor de puertos (en adelante, expansor o MCP23017) a la lista de componentes, en concreto el MCP23017.



La decisión se debe a su capacidad para 16 I/O pins y, sobre todo, debido a su control por I2C, compartida con el acelerómetro/giroscopio que se menciona en el siguiente apartado. Esto permite reutilizar los pines del microcontrolador, es decir, sin utilizar ningún otro pin del NodeMCU se tienen disponibles 16 entradas o salidas digitales, de las cuales se utilizarán 6 como entradas para los conmutadores de los botones.

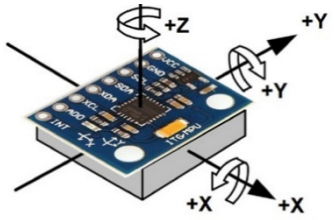
Además de contar con librerías públicas, que permiten su sencilla programación, también cuenta con resistencias PULLUP internas que permitirán conectar directamente los pulsadores sin hacer uso de resistencias adicionales.

4.1.4 ACELERÓMETRO/GIROSCOPIO

Para poder detectar la orientación del juguete, así como los movimientos utilizados, se hará uso de un módulo de acelerómetro/giroscopio, el MPU6050.

Se trata de un dispositivo muy común en proyectos debido a su versatilidad como inclinómetro, acelerómetro, giroscopio, termómetro… Sobre todo, se utilizará como inclinómetro, para saber cuál es la cara que está apuntando hacia arriba, aunque toda la funcionalidad estará cargada y toda la información de los sensores es accesible a través de una librería.

Ilustración 6: MPU6050



Se trabajará con una librería propia muy sencilla (en detalle en el *5.2.4.3 MPU6050.h*), que enviará las instrucciones como bytes a los registros necesarios del MPU6050 para solicitar la información de las aceleraciones (en las 3 direcciones X, Y y Z).

En función de estos 3 valores se puede deducir la orientación del juguete, simplemente la cara con una mayor aceleración será la inferior. Ej.: Si el acelerómetro mide -9.8 m/s^2 en la Z y 0 en la X e Y, claramente se encuentra apuntando hacia arriba. Y es posible pegar el sensor a la cara superior y así conocer la posición de dicha cara y la orientación del juguete.

Este método, a pesar de ser muy sencillo, tiene la limitación de que, en caso de dejar caer el cubo o bajarlo/moverlo muy rápido, el acelerómetro detectaría la dirección del suelo como la que está acelerando, pero no debería afecta al juego. Además, sin acelerar el juguete (solo rotándolo), es posible – por trigonometría – saber la inclinación exacta en cada ángulo.

Dado que el sensor toma también medidas de rotación y temperatura, otra forma de medir la inclinación es calibrándolo a 0 cuando se encuentra mirando hacia arriba y, en función de la aceleración angular del giroscopio (en rad/s) y el tiempo transcurrido desde la última medida, es posible calcular la nueva posición. Este es el método usado en combinación por el anterior por varias librerías para mejorar la precisión, pero en este Proyecto no hay necesidad de complicarlo más para obtener tanta precisión.

4.1.5 ALTAVOZ/ZUMBADOR

Uno de los componentes más sencillos y útiles es el altavoz. Es capaz de representar con un sonido el estado del juego: un zumbido en caso de error, una melodía en caso de victoria, y con un sonido de metrónomo el tiempo restante que tiene el jugador para introducir la secuencia correcta para ganar.

Para esto se utilizará un speaker sencillo con un rango de frecuencias suficiente para reproducir una melodía (~600-10KHz) a baja potencia (0.5W) y con una impedancia de 8Ω.

Ilustración 7: Altavoz

El altavoz escogido es de tamaño reducido (36mm de diámetro) y cuenta con solo 2 pines (tierra y Vcc) con los que se controlará por medio de un PWM y la función integrada de Arduino tone() además de hace ruso de una librería propia (*spkControl.h*) para reproducir melodías.

4.1.6 CAJA – ESTRUCTURA CONTENEDOR

Como contenedor de los componentes electrónicos se escogió una caja de madera cúbica con una apertura por medio de un corte transversal en una de las caras.

El tamaño exterior es de 12x12x12 cm con un peso de 130 g (modelo: Artemio VIBB19).

Se escogió este modelo por su tamaño suficiente para los botones Arcade que habrá en cada cara. Para ello, se requerirá taladrar 6 agujeros de 28 mm, además de uno pequeño para el cable de alimentación/debug. Se trata de madera blanda por lo que no es difícil de perforar con una broca básica, pero requiere tener cuidado para no astillarla.

Ilustración 8: Caja de madera

4.1.7 PLACA PERFORADA PCB, PINES Y CABLEADO

Para el conexionado fijo de los componentes se va a hacer uso de una placa perforada de cobre (PCB) en la que soldar los pines macho-hembra. Estos van a permitir conectar y desconectar los componentes para testearlos en la PCB o protoboard sin fijar ningún componente soldado de forma permanente, pero manteniéndolos sujetos firmemente.

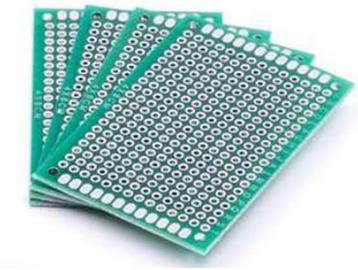


Ilustración 9: Placas BCBs

Posteriormente, en el *5.1.2 Soldado* se muestran en detalle los resultados de soldar utilizando esta combinación.

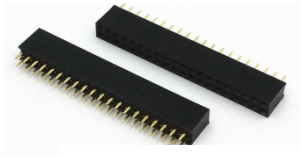


Ilustración 10: Pines macho-hembra

De esta forma, se conectará el microcontrolador y el expansor a través de estos headers macho-hembra que estarán soldados a la PCB. El resto de los componentes (botones LED y acelerómetro se conectarán también con estos headers, pero usando varios cables soldados a pines macho-macho para salvar la distancia entre la PCB y los componentes:

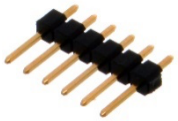


Ilustración 11: Pines macho-macho

Tanto los botones como el acelerómetro requieren de 4 cables únicamente para su conexión entre ellos y el NodeMCU. En el caso de los botones, ya se han explicado los 4 conectores en su *4.1.2*, y, en el caso del MPU6050, se trata de 2 cables para tierra y Vcc y otros 2 para la comunicación por I2C (SDA y SCL). Por tanto, se requerirán 7 segmentos de cable de 4 hilos (rojo, blanco, verde, amarillo).

Ilustración 12: Cable de 4

Para soldar el altavoz a la PCB se ha utilizado cable básico de cobre, del mismo tipo que se ha utilizado para testear los componentes previamente en la protoboard.

Y, finalmente, para conectar el NodeMCU al ordenador o alimentarlo se ha utilizado un cable microUSB que sale de la caja por un pequeño agujero en la apertura de la misma.

4.1.8 OTROS ELEMENTOS

A lo largo del desarrollo del Proyecto se requerirán otros componentes u herramientas para su construcción o el testeo de componentes, pero no forman parte del diseño final y, por lo tanto, solo se mencionarán brevemente a continuación:

- Soldador y estaño para el soldado entre cable y pines y de la PCB.
- Protoboard, para testear el funcionamiento y conexionado de componentes.
- Taladradora y brocas de varios tamaños (2, 24 y 28 mm) para el agujereado de la caja.
- Pistola y pegamento termofusible para fijar ciertos componentes (altavoz y acelerómetro).
- Funda para cables (con soplete de aire caliente para su compresión en torno al cable) y cinta aislante para nombrar los cables.

Ilustración 13: Ejemplos de otros elementos utilizados

4.2 SOFTWARE: PROGRAMAS

Este apartado pretende enumerar las diferentes herramientas de software utilizadas para el desarrollo del Proyecto y explicar su funcionamiento si procede.

4.2.1 VISUAL STUDIO CODE

Visual Studio Code [12] (o VSCode) es un editor de código multiplataforma gratuito y código abierto, con capacidad de depuración y control integrado Git. Se trata del editor de código más utilizado en el mundo seguido de su hermano mayor Visual Studio. Por ello, no se entrará en detalle en su funcionamiento, solo en los plugins específicos que se han utilizado.

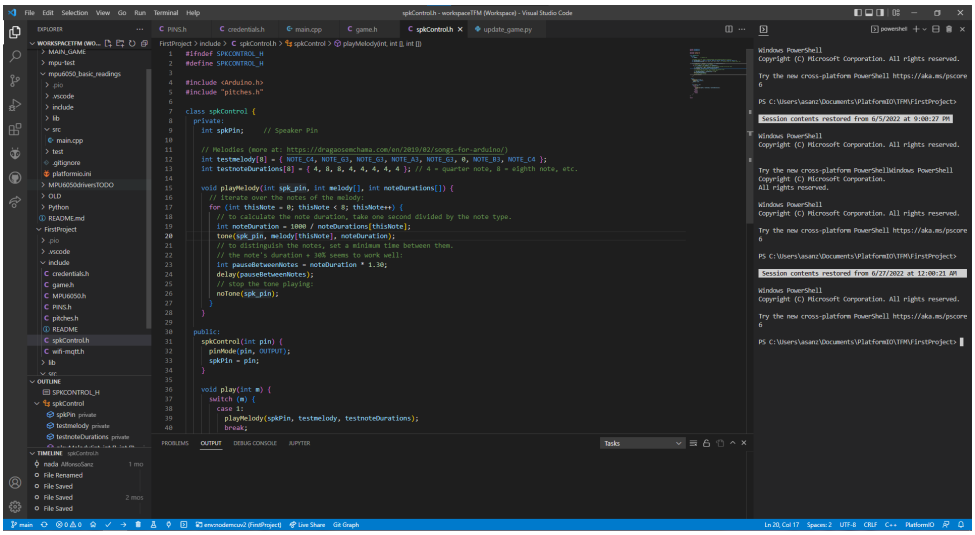


Ilustración 14: Visual Studio Code

VSCode cuenta con una gran cantidad de plugins para ampliar su funcionalidad, incluyendo los que mejoran la compatibilidad con Github y PlatformIO para la programación integrada de microcontroladores.

Ambos plugins se detallan a continuación:

4.2.1.1 Platform IO

Platform IO [13] es una herramienta profesional multiplataforma, multiframework y cross-architecture dedicada a la programación de sistemas integrados o aplicaciones para productos integrados. Es decir, va a permitir la programación del microcontrolador por medio y subida del código utilizando el entorno de VSCode, en contraste con usar Arduino IDE que, aun siendo más simple y sencillo, no cuenta con la gran funcionalidad y compatibilidad de VSCode.

Platform IO gestiona sus propios lenguajes de programación y permite desarrollar diferentes SDKs (NonOS, RTOS…), aunque se utilizará el de Arduino, debido a su mayor documentación y conocimiento previo.

También administra los drivers de los controladores y la instalación de librerías, que se descargarán en local y se añaden a los proyectos incluyendo el nombre de la librería en el archivo ".ini" correspondiente, donde se indicará también el nombre de la plataforma, controlador, framework y velocidad de monitorización Serial, entre otros.

Dicho archivo forma parte de una estructura específica de los proyectos creados con PlatformIO, donde lo más importante es el main.cpp en la carpeta "src" y las librerías privadas en la carpeta "include". La estructura y el archivo mencionados se muestran en la siguiente imagen:

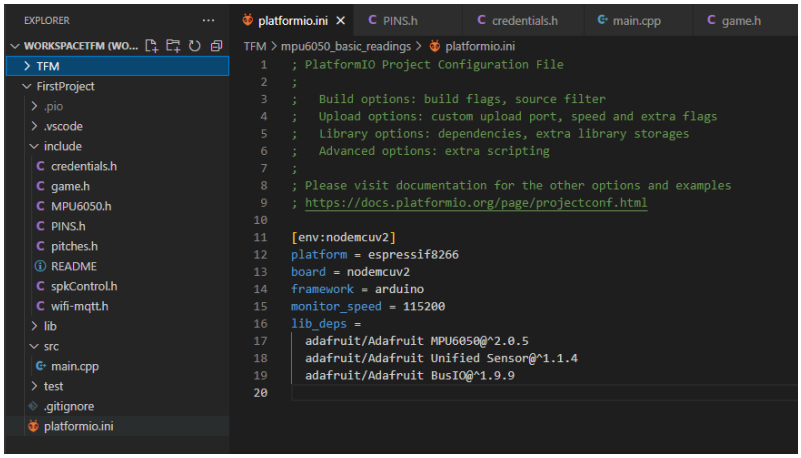


Ilustración 15: PlatformIO

4.2.2 GITHUB

Sin necesidad de mucha introducción, se ha de mencionar GitHub [14], plataforma gratuita de almacenamiento en la nube (principalmente de código) con control de versiones y enfocada al desarrollo colaborativo. Es accesible desde el navegador, pero su mayor utilidad es desde el terminal por medio de la instalación de Git y su uso en combinación con VSCode.

Una descripción breve de su funcionamiento requiere mencionar el uso de repositorios (contenedores) que es posible clonar y conectar con el repositorio en la nube. A partir de comandos se pueden bajar los cambios (pull), subirlos (push), hacer puntos de control (commit), crear ramas (branch), actualizar el árbol (fetch) y unirlas (merge), entre muchos otros. VSCode permite realizar dichos comandos desde el terminal o desde la propia interfaz gráfica, además de poder visualizar los cambios, el árbol de ramas, etc. por medio de plugins.

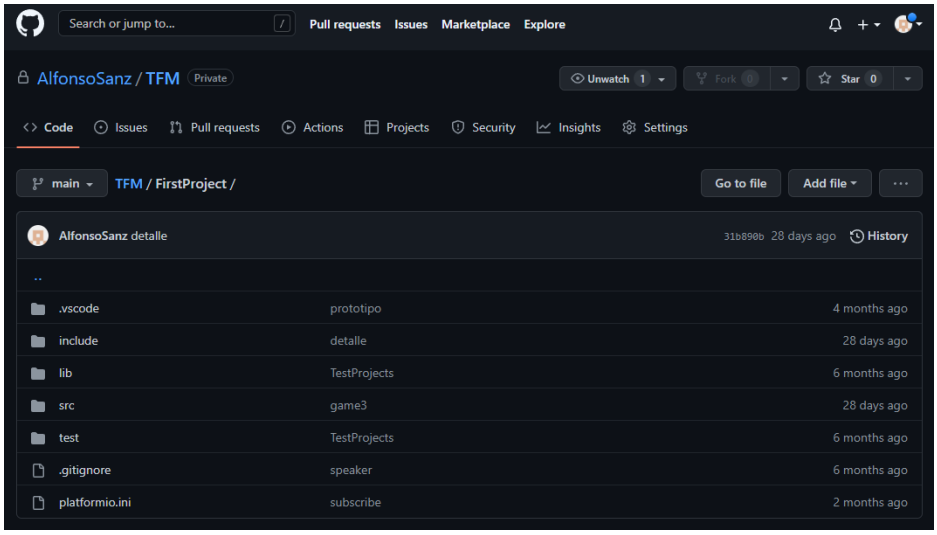


Ilustración 16: GitHub

4.2.3 ALTAIR SMARTWORKS IOT

Altair SmartWorks es un ecosistema corporativo en la nube de Inteligencia Artificial, Machine Leaning e Internet of Things.

Para el Proyecto se utilizará, concretamente, Altair SmartWorks IoT, una plataforma IoT escalable, segura de aplicaciones analíticas de datos. Se pude usar Cloud, híbrida u on-premises (para el Proyecto se usarán los servidores en la nube), aunque no se requiere en este caso una gran potencia computacional o almacenamiento y, por lo tanto, se utiliza una cuenta de estudiante con la que se podrá realizar de sobra la comunicación y programación de un dispositivo. En el *Parte I5.3 Desarrollo IoT* se entrará en detalle acerca de su funcionamiento y uso en el Proyecto.

4.2.4 EASYEDA

La herramienta web gratuita EasyEDA [16] ya se ha mencionado en el *5.1.2 Soldado* y no se entrará en más detalle debido a su uso sencillo, pero sí indicar que permite el diseño 2D y 3D de circuitos, en especial de placas PCB o protoboards. Incluso cuenta con opciones más complejas para su fabricación.

La siguiente ilustración muestra la interfaz gráfica a la que se accede desde el navegador:

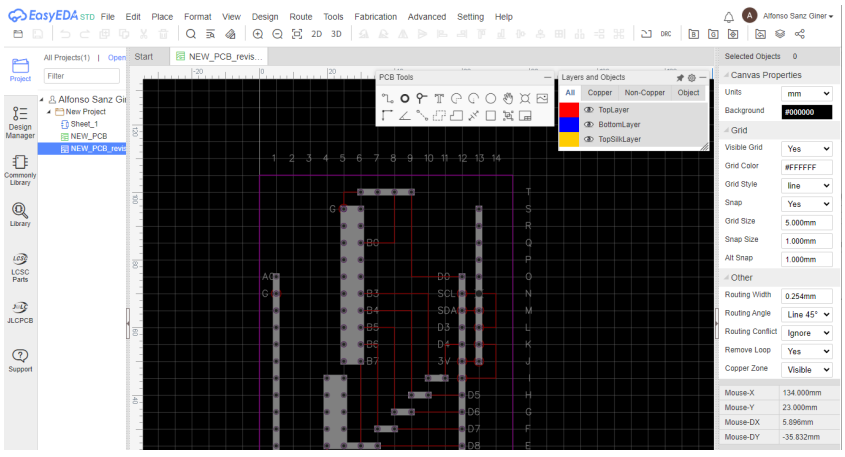


Ilustración 17: EasyEDA

Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo se detalla el proceso y la metodología de diseño del Proyecto, primero a nivel de construcción del prototipo físico y, posteriormente, acerca del diseño lógico y programación/conexión con la plataforma IoT.

5.1 CONSTRUCCIÓN DEL PROTOTIPO

En este apartado se describe el proceso de construcción del prototipo del juguete incluyendo el tratamiento de la madera, el soldado de los componentes a la PCB y el cableado del conjunto. La mayoría de la construcción se realizó en el taller de proyectos de electrónica de la Universidad y en el laboratorio de fabricación (FabLab) de ICAI.

5.1.1 AGUJEREADO

Debido a la elección de la caja se contaba con el tamaño justo para los botones LED de Arcade, que son el elemento que más ocupa debido a su posicionamiento en cruz en cada una de las 6 caras.

Teniendo en cuenta el largo del botón de 52,8 mm debería caber en el interior de la caja de 12 cm de arista.

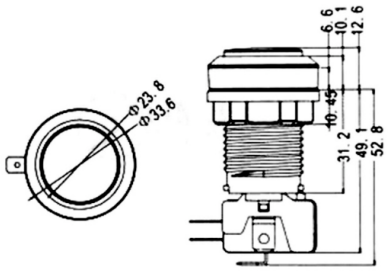


Ilustración 18: Plano de los botones LED

Se planteó inicialmente colocarlos en esquinas de cada lado en vez del centro de cada arista, pero estéticamente hubiese quedado peor, aunque hubiese permitido más espacio en el interior.

Se escogió una broca entre los 24 mm del diámetro de la rosca interior del botón y 33 mm del bloqueo superior, concretamente de 28 mm, y se taladró sin dificultad – teniendo cuidado de no astillar al ser madera blanda – el centro de cada cara de la caja.

La caja no se escogió de un tamaño superior pensando en su ergonomía. Por lo que, debido al largo de los botones y que el cableado está muy ajustado, los botones sobresalen con el tamaño de la tuerca de plástico. En futuros diseños es importante reducir significativamente el tamaño de la caja y, para ello, escoger botones LED más pequeños. Pero, por otro lado, el diseño es más rígido y fuerte dado que los botones Arcade están pensados para ser pulsados con fuerza y repetidamente, por lo que tendrán una mayor vida útil.

En la figura de la derecha se muestra el resultado:

Ilustración 19: Caja agujereada con botones

También se realizó un agujero de 2 mm de diámetro para introducir el cable y, aunque debido al tamaño del agujero la madera se astilló, fue posible arreglar el pequeño fragmento astillado pegándolo con una pistola de termofusible. La misma se utilizó para fijar el acelerómetro a la cara superior interior (tapa) de la caja en una de las esquinas, más en detalle en el *5.1.3 Cableado*.

5.1.2 SOLDADO

El soldado de los componentes fue la tarea de mayor duración en la construcción física del prototipo. Como se ha explicado en el *4.1.7 Placa perforada PCB, pines y cableado*, se ha escogido soldar una serie de headers macho-hembra a la PCB, en vez de soldar los elementos directamente, pudiendo así soltar todos los elementos y testearlos por separado en una protoboard en caso de fallo (lo cuál ha sido muy útil incontables veces).

Se hizo uso de la herramienta online EasyEDA para el diseño del circuito requerido. A continuación, se muestra el diseño realizado a dos caras:

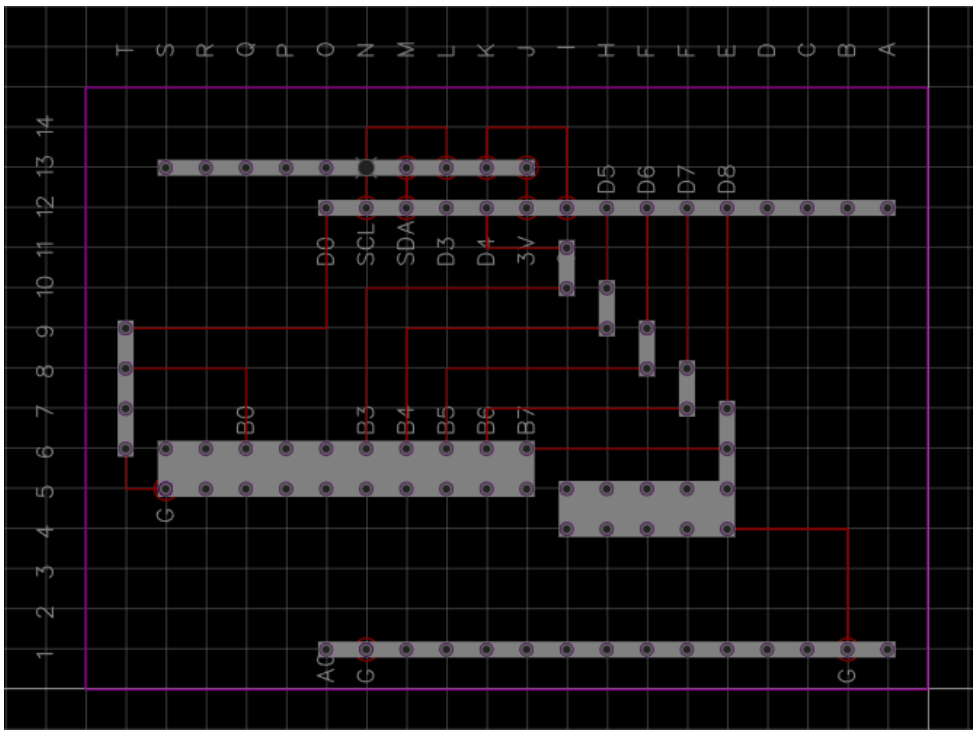


Ilustración 20: Diseño de la PCB

Lo más importante de cara a planificar la posición de cada objeto es conocer los tamaños, número de pines y distancias entre pines. Por ejemplo, la PCB dispone de un tamaño de 20x14 pines; el NodeMCU cuenta con 2 tiras de 15 pines (en la ilustración de la PCB corresponde a las tiras de coordenadas A1-O1 y A12-O12 en la PCB) separados por 10 pines (en total, 15x12 pines) y, por lo tanto, ocupa prácticamente por completo una de las caras.

Mientras, el MCP23017 dispone de una tira simple de 10 pines (J13-S13) y una doble (J5-S6) distanciadas 5 pines (10x8 pines), que se colocará en la otra cara.

En la misma cara se conectarán 6 headers de 4 pines más para los botones LEDs: como 2 de los 4 de cada botón van a tierra, se conectarán 10 (E4-I5) a GND del NodeMCU y los otros 10 pines irán de 2 en 2 en escalera (E5-I11) conectándose al NodeMCU y MCP23017, respectivamente. El botón blanco LED de la cara superior va por separado del resto en la misma cara (T6-T10).

El soldado es a dos caras, por lo que el estaño a veces se introduce por un nodo y se continúa la línea por el otro lado para llegar al pin del otro componente, como en caso de querer conectar MCP23017 y NodeMCU o los botones LED a ambos.

En las dos siguientes fotos se muestra el resultado del soldado de las 2 caras, en la izquierda se posiciona el NodeMCU y en la derecha el MCP23017 y los pines de todos los botones:

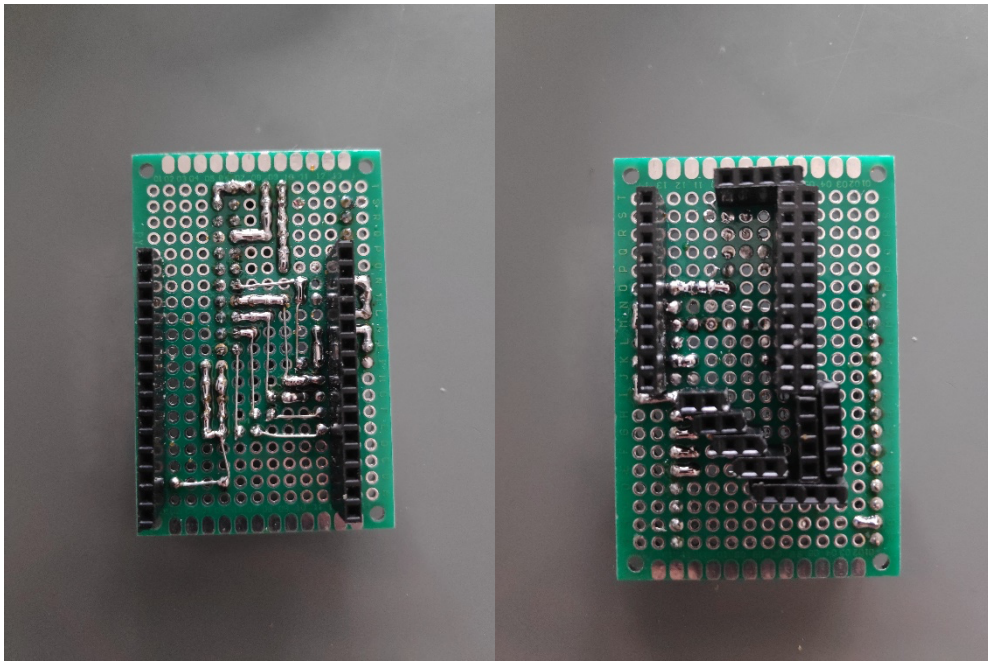


Ilustración 21: PCB soldado a ambas caras

Se utilizan cables de resistencias sobrantes para ahorrar estaño (en la foto de la izquierda). En las imágenes no se muestra la conexión posterior del altavoz y MPU6050.

El altavoz, al requerir 2 cables, se conectó directamente a tierra y al pin correspondiente.

Y el MPU6050, al ir pegado a la cara superior, requería la capacidad, no de soltarse de la PCB, sino de soltar el cable que viene de la PCB al MPU6050. Es decir, los pines/cables se soldaron directamente a la PCB y el header hembra al que va conectado se puede desconectar para abrir la tapa.

5.1.3 CABLEADO

Aunque en el apartado anterior parece complicado el circuito, en realidad es muy sencillo. A continuación, se lista una tabla con la lista de cables y pines:

Cableado	Conector	Blanco	Azul	Verde	Amarillo	Rojo	Blanco	MPU6050
rojo	COM (abajo)	GND	GND	GND	GND	GND	GND	VCC
blanco	LED (rojo)	GND	GND	GND	GND	GND	GND	GND
amarillo	NO (delante)	MCP8	MCP11	MCP12	MCP13	MCP14	MCP15	SDA (D1)
verde	LED (negro)	D0	D4	D5	D6	D7	D8	SCL (D2)
		normal	al revés	al revés	normal	normal	al revés	

Tabla 3: Pines de los botones

Para realizar dichas conexiones a los botones se han utilizado 6 cables cortos de 4 cables interiores (rojo, blanco, amarillo y verde), como se ha mencionado en el *4.1.7*. Estos cables con hilos de cobre en el interior se han enrollado a los agujeros que tienen los botones en los conectores y soldado.

También se ha utilizado una funda para cables del *4.1.8* dado que los conectores van juntos en un mismo punto apretados en el centro de la caja, así se evitan conexiones accidentales. La funda se corta y se mete el cable por dentro. Después de soldarlo, se calienta con un soplete de aire caliente o mechero para que se comprima alrededor del cable.

En la tabla se muestra – para cada cable – donde se conecta del botón y a donde se dirige: GND, NodeMCU en el caso de los LEDs (D0, D4-8) o al expansor en el caso de los pulsadores NO. La última línea indica si se han de cambiar los cables de los LEDs (verde y blanco) dado que a veces el ánodo y cátodo se encuentran intercambiados de fábrica.

Una vez conectados los botones el resultado es el siguiente:

En la imagen se observa todos los botones menos el blanco de la cara superior (tapa) conectados a la cara que dispone del MCP23017, mientras que por la otra cara se encuentra conectado el NodeMCU al que también están conectados.

Ilustración 22: Cableado de botones

Por último, se terminaron de conectar los últimos componentes: el MPU6050, como se indica en la tabla (GND, VCC y los pines de I2C), y el altavoz, a tierra y al pin D8 del NodeMCU.

El último botón se conecta separado del resto (más accesible) y el MPU6050 va pegado a la tapa y conectado por un cable corto con 4 cables interiores. Una vez todo colocado el resultado es el de las siguientes ilustraciones:

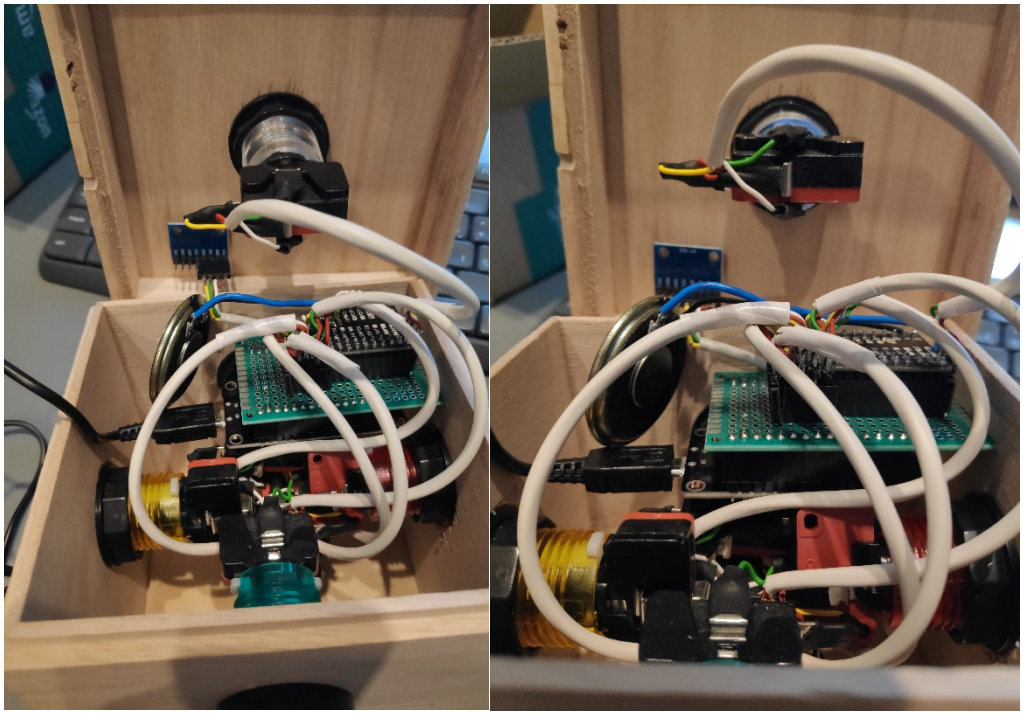


Ilustración 23: Cableado completo (tabla levantada)

5.2 PROGRAMACIÓN

Como ya se ha mencionado en apartados anteriores, la programación del juguete se realizará en Visual Studio Code en el lenguaje de Arduino (framework de C++) y utilizando PlatformIO como herramienta de soporte que organiza la estructura de archivos del Proyecto, conecta las librerías al Proyecto y permite compilar, debugear y subir el código al NodeMCU.

De cara a disponer de un código limpio, fácil de entender, y, sobre todo, versátil, se trabajará con librerías. Las librerías son un conjunto de funciones o clases definidas en archivos separados que facilitan la programación simplificando el control de los componentes o de los programas.

Gracias a las librerías el código principal "main.cpp" es muy sencillo. Únicamente se deberán importar 3 librerías (Arduino.h, game.h y mqtt.h), inicializar la instancia del objeto "Game" en el setup, comprobar en bucle la conexión MQTT por si se recibe alguna instrucción, y, finalmente, según el juego seleccionado llamará a la función del objeto `game` correspondiente.

En el siguiente subapartado se incluye el sencillo código que se ha descrito. Adicionalmente, se puede observar que el programa muestra el juego seleccionado por pantalla cada vez que se modifica y ejecuta 4 casos (que se verán en detalle en el 5.2.4.7 game.h):

- El primero es una función para testear el correcto funcionamiento del juguete.
- El segundo es un juego configurable que es posible jugar offline.
- El tercero es un programa que sigue las instrucciones de la plataforma de Altair.
- En cualquier otro caso, comprueba si han llegado mensajes nuevos que mostrar y los muestra por pantalla.

5.2.1 MAIN.CPP

```
#include <Arduino.h>
#include <wifi-mqtt.h>
#include <game.h>

Game game;
int prev_game_value = -1;

void setup(){
  // Initialize game
  game.init();
}

void loop(){
  // Check for new messages
  mqtt_check();

  // Print selected game
  if (game_value != prev_game_value) {
    prev_game_value = game_value;
    Serial.print("Game selected: ");
    Serial.println(game_value);
  }

  // Run selected game
  switch (game_value) {
    case 1:
      game.test();
      mqtt_check();
      delay(100);
      break;
    case 2:
      game.game2();
      break;
    case 3:
      game.slave();
      break;
    default:
      mqtt_check();
      print_new_message();
      delay(100);
  }
}
```

Pero, para entender el código por debajo de este sencillo programa, es necesario explicar en profundidad las librerías que requiere. A continuación, se listan las librerías utilizadas diferenciando entre las integradas en Arduino, las públicas (accesibles a través de PlatformIO) y las privadas desarrolladas para el Proyecto:

Librerías integradas

- Arduino.h
- Serial.h
- Wire.h

Librerías públicas:

- Adafruit_MCP23X17.h
- ESP8266WiFi.h
- PubSubClient.h (por knolleary)
- ArduinoJson.h (por bblanchon)

Librerías privadas

- pitches.h
- spkControl.h
- Adafruit MPU6050
- credentials.h
- wifi-mqtt.h
- PINS.h
- game.h

En los siguientes subapartados se detallarán su funcionalidad y las funciones más relevantes. El código de las librerías privadas se adjunta en el *ANEXO II: Código*.

5.2.2 LIBRERÍAS INTEGRADAS

5.2.2.1 Arduino.h

Librería necesaria al trabajar a través de PlatformIO. Aunque en el archivo platformio.ini se indica el uso del framework de Arduino, también es necesario incluir la librería para que se añadan las funciones específicas del lenguaje: `pinMode()`, `digitalRead()`, `digitalWrite()`…

5.2.2.2 Serial.h

Comenzando por la más sencilla y conocida, Serial es la librería de comunicación entre el microcontrolador y el ordenador. También define el protocolo utilizado para la transmisión de información por los canales RX y TX de la UART del NodeMCU (para comunicación entre microcontroladores) pero, en este caso, no se utilizará con esta finalidad. Sus funciones más importantes son:

- `Serial.begin()`: Inicializa la comunicación y establece la velocidad de transmisión. En este caso, se le mandará como parámetro el baud rate de 115200 bits/s, y también se ha de indicar este parámetro en el archivo platformio.ini en la variable `monitor_speed` que determinará la velocidad con la que ha de leer los mensajes que recibe por el puerto USB.
- `Serial.print()` o `Serial.println()`: Envía el mensaje ASCII por el puerto. La segunda variante incluye un salto de línea al final del mensaje.

Existen más funciones en esta librería `Serial.avalaible()`, `Serial.write()`, `Serial.read()`… pero no serán necesarias para el Proyecto al comunicarse por cable únicamente con el ordenador.

5.2.2.3 Wire.h

Librería de comunicación a través del protocolo I2C, el cual se basa en 2 cables (además de necesitar una tierra común): SCL para el reloj y SDA para los mensajes. Permite la conexión bidireccional de múltiples dispositivos en configuración master-slaves. Es decir, un dispositivo puede comunicarse con varios a través de un solo BUS. En este caso, se dispone del NodeMCU como maestro y el MCP23017 y MPU6050 como esclavos, pudiendo así mandarles instrucciones y recibir su respuesta.

Sus funciones son muy parecidas a las de la anterior librería:

- `Wire.begin()`: Inicializa la conexión, pero está vez recibe como parámetro la dirección de 7bits, por defecto es 0 para el caso del microcontrolador como maestro.
- `Wire.beginTransmission()`: Indica a qué dirección se enviará el mensaje, de fábrica el address del MPU6050 es 0x68 y el del MCP23017 es 0x20, aunque este último es irrelevante, ya que otra librería se encargará por completo de la comunicación.
- `Wire.write()`: Añade el mensaje a la cola, como parámetro requiere el mensaje, para el Proyecto en formato byte, no solo porque el protocolo transmita byte a byte sino por el tamaño de los registros del MPU6050.
- `Wire.endTransmission()`: Envía el mensaje anterior y cierra la comunicación (si recibe como parámetro `true` manda un stop) o manteniéndola abierta (`false` manda un restart).

Esta librería se utilizará en una explicada posteriormente (*MPU6050.h*) para comunicarse con dicho dispositivo.

5.2.3 LIBRERÍAS PÚBLICAS

5.2.3.1 Adafruit MCP23X17

Librería de Adafruit para el control del MCP2017. Es necesario incluir la librería, instanciar el objeto Adafruit_MCP23X17 que se llamará `mcp` (sin parámetro de construcción) y proceder con las siguientes funciones:

- `mcp.begin_I2C()`: Inicializa la conexión y devuelve `true` en caso de conexión correcta.
- `mcp.pinmode()`: Al igual que los pines del Arduino se ha de definir el modo INPUT u OUTPUT del pin. En caso de definirlo como INPUT el pin se conectará a una resistencia PULLUP que permite que se ponga a tensión (HIGH) en caso de soltar o desconectarse el pulsador.
- `mcp.digitalRead()`: Al igual que la función `digitalRead()` de Arduino devuelve el valor HIGH or LOW del pin indicado como parámetro, para facilitar los nombres y la iteración por los pines se utilizará la librería *PINS.h* que se menciona en apartados posteriores.

5.2.3.2 ESP8266WiFi.h

Conjunto de funciones y librerías que permiten la comunicación por Wifi del NodeMCU, específicamente el de la versión con el procesador ESP8266. Es muy parecida a la librería clásica Wifi.h integrada en Arduino y, de hecho, hace uso de la misma (entre otras) con la clase WifiClient. No obstante, en el Proyecto se utilizará directamente la siguiente librería PubSubClient.h.

5.2.3.3 PubSubClient.h

Librería encargada de crear el cliente wifi que enviará y recibirá los mensajes MQTT. Una vez inicializada a partir de un cliente WifiClient, sus funciones principales son:

- `client.setServer()`: Establece la IP/dominio del servidor y puerto de comunicación.
- `client.setCallback()`: Crea la función que se llamará en caso de recibir un mensaje.
- `client.connected()`: Devuelve `true`/`false` en función de si la conexión está abierta.
- `client.connect()`: Conecta el cliente con el id y autorización (nombre y contraseña) indicados como parámetros.
- `client.subscribe()`: Se subscribe al topic indicado como parámetro.
- `client.state()`: Devuelve el código de estado de la conexión.
- `client.loop()`: Chequea si se han recibido mensajes y los descarga del buffer.
- `client.publish()`: Envía un mensaje por el topic indicado.

5.2.3.4 ArduinoJson.h

Librería encargada del encoding/decoding de los mensajes JSON:

- `deserializeJson()`: Convierte una lista de bytes (recibidos por MQTT) en una instancia StaticJsonDocument, parecida a una lista, que se llamará `doc`.
- `doc.containsKey()`: Comprueba si la lista/JSON contiene una llave o identificador.
- `doc["key_name"]`: Permite acceder al elemento a través del nombre de la llave.

En resumen, a través de esta librería será posible (en la librería wifi-mqtt.h) de recibir un payload de bytes, como por ejemplo `"{'game':1}"` y acceder al valor con:

```
if (doc.containsKey("game")) { game_value = doc["game"]; }
```

5.2.4 LIBRERÍAS PRIVADAS

El código de las siguientes librerías se encuentra adjunto en el ANEXO II: Código.

5.2.4.1 pitches.h

Esta librería es únicamente una lista de las frecuencias (en Hz) de cada nota definidas como constantes: `#define NOTE_B0  31`, `#define NOTE_C1  33`… para su uso en la siguiente librería en las melodías.

5.2.4.2 spkControl.h

Define la clase encargada de controlar el altavoz (speaker o spk) por medio de un PWM. Cuenta con 3 métodos, 2 públicos y uno privado:

- `spkControl(int pin)`: El constructor define el pin que se va a utilizar como output.
- `void play(int m)`: Recibe el número de la melodía que se va a reproducir y llama a la siguiente función.
- `playMelody(int melody[], int noteDurations[])`: Función que ejecuta un bucle reproduciendo una lista de notas (melodía) con una duración determinada por una lista de duraciones. Hace uso de la función integrada de Arduino `tone()` que recibe como parámetros el pin, la frecuencia y duración en ms y regula el PWM de control del altavoz; en combinación con `delay()` entre notas y `noTone()` para asegurar que el PWM se ha detenido.

En resumen, por medio de esta clase, para reproducir una melodía únicamente es necesario llamar a `spk.play(numero_melodia)`. Solo se ha definido una melodía de victoria y dos que reproducen un sonido, en caso de introducir una secuencia incorrecta, y para contar el tiempo con un temporizador (por ejemplo: cada segundo).

5.2.4.3 MPU6050.h

Como ya se ha mencionado en el *4.1.4 Acelerómetro/giroscopio*, el MPU6050 cuenta con múltiples funcionalidades por medio de sus sensores, pero se utilizará principalmente el acelerómetro para su uso como inclinómetro.

También se ha mencionado previamente que se obtiene la información comunicándose por medio de I2C a través de la librería *Wire.h*. Para ello se ha creado una clase `MPU6050` que almacenará los valores transmitidos por el MPU6050, a petición de las siguientes funciones:

- `reset()`: Esta función únicamente "reseteará" el acelerómetro, es decir, escribirá en el registro 0x6B un 0 indicando que se ha de borrar el buffer de tiempos/posiciones calibrando el dispositivo. Para ello, el dispositivo debería encontrarse en una posición mirando hacia arriba, aunque como únicamente se utiliza la información de los acelerómetros no será necesario.
- `read()`: Función encargada de solicitar la información de las aceleraciones en las 3 coordenadas. Para ello, manda por I2C a la dirección del MPU6050 el byte 0x3B, que corresponde al registro del acelerómetro y hace un request a la misma dirección de 14 bytes. Posteriormente, hace una lectura del mensaje recibido, almacenando en las variables `AcX`, `AcY` y `AcZ` las aceleraciones y, finalmente, calcula la inclinación en cada eje en grados (`xAng`, `yAng` y `zAng`) y radianes (`x`, `y` y `z`).
  No se entrará en detalle en el cálculo pero es bastante directo gracias a las funciones `map()` y `atan2()` de las librerías integradas de matemáticas.
- `getOrientation()`: Por último, esta función es la que se llama para obtener la orientación en formato legible (String). Hará uso de la función anterior para actualizar los valores de aceleraciones y, en función de los mismos, hallará la cara con la mayor aceleración, devolviendo la cara opuesta (que será la superior) con el eje correspondiente (positivo o negativo) y el color LED. Por ejemplo: "-X: Blue".

Se ha de indicar que toda esta funcionalidad es accesible a través de librerías públicas como Adafruit MPU6050, Adafruit Unified Sensor (con Adafruit BusIO), pero se ha optado por acceder a los registros manualmente reduciendo el código a cargar significativamente.

5.2.4.4 credentials.h

Define las siguientes constantes:

- `const char* networkSSID`: Nombre de la red Wifi a la que se conectará el dispositivo.
- `const char* networkPASSWORD`: Contraseña de la red.
- `const char* mqttSERVER = "mqtt.swx.altairone.com"`: Dominio web broker MQTT.
- `const char* mqttUSERNAME = "va7c24@tfm-juguete"`: ID del "thing" y nombre de la colección del dispositivo en la red de Altair.
- `const char* mqttPASSWORD`: Contraseña de conexión MQTT.
- `String subscribeTopic = "spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/"`: Primera parte de los topics a los que se va a subscribir el dispositivo, definiendo el espacio, colección y thing.
- `String subscribeTopics[] = {"game","timeout","difficulty","print_message","outputs"}`: Lista de los subtopics del thing a las que subscribirse para obtener sus propiedades.
- `char publishTopic[] = "spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties"`: Topic a donde publicar para editar las propiedades en la plataforma IoT.
- `char publishTopic_data[] = "spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/data"`: Topic al que publicar para subir al histórico de mensajes.

5.2.4.5 wifi-mqtt.h

Librería encargada de organizar toda la comunicación Wifi y MQTT. Se ha partido de un código básico de ejemplo que recopila una serie de funciones y se han ido modificando y añadiendo más como se explica a continuación:

Como en el caso de todas las librerías privadas, el código se adjunta al documento en el *ANEXO II: Código* en el apartado correspondiente (*wifi-mqtt.h*), el cual se explica en detalle a continuación.

Para empezar, se incluyen las librerías explicadas en apartados anteriores (*ESP8266WiFi.h*, *PubSubClient.h*, *credentials.h*, *PINS.h* y *ArduinoJson.h*). Luego se recogen con punteros las direcciones a las constantes de datos de credenciales, se define el estado inicial del módulo Wifi y se instancia el cliente `PubSubClient` a partir de un `WiFiClient`.

Para almacenar los mensajes recibidos de las propiedades del juego por la plataforma IoT se utilizarán las siguientes variables: `game_value`, `timeout`, `difficulty`, `print_message`, `new_message` y `outputs[6]`. A partir de entonces se definirán las funciones:

- `callback()`: Un callback es una función que se llama automáticamente en caso de recibir un mensaje/petición. En este caso, el callback se encarga de procesar el mensaje JSON recibido y, en caso de contener información que se haya de almacenar para la configuración del juego, se guardará en la variable correspondiente.
- `wifi_mqtt_init()`: Como su nombre indica, se encarga de inicializar la conexión Wifi con `Wifi.begin()` en bucle hasta conectarse. Posteriormente imprime por pantalla la dirección IP, define (sin conectarse) el broker/servidor MQTT de Altair y termina uniendo la función anterior de callback a la instancia `client`.
- `mqtt_reconnect()`: Función encargada de revisar y establecer la conexión MQTT. Se conecta por MQTT y suscribe iterando por la lista de topics. En caso de fallo devuelve el "response code" y vuelve a intentarlo.

- `mqtt_check()`: Llamada a la función `client.loop()` que administra el buffer de entrada y salida de mensajes. Es decir, después de haber puesto el mensaje en cola es necesario llamar a esta función para enviarlo o para chequear si hay mensajes entrantes.
- `wifi_mqtt_send()`: Función encargada de revisar la conexión, llama a las funciones anteriores y publica el mensaje indicado como parámetro en el topic solicitado.
- `update_property()`: Esta función procesa el mensaje a JSON y llama a la función anterior para enviarlo. Tiene 2 llamadas por overloading de parámetros. En ambos casos recibe la String con el nombre de la propiedad a modificar y puede recibir un entero o un String en función del tipo de dato. La única diferencia entre ambas formas de llamar a la función es que, en el último caso, se requieren comillas en el JSON.
  En ambos casos se envía el mensaje a 2 topics, el de la "property", para actualizar el valor en la plataforma de Altair y a "data", donde se almacena el histórico de los mensajes.
- `print_new_message()`: Finalmente esta función se llama para comprobar que no se hayan enviado mensajes desde la plataforma IoT para mostrar por pantalla. En caso de haberlos los imprime en el terminal.

5.2.4.6 PINS.h

Define las constantes GPIO de los pines de los leds en el NodeMCU y de los botones en el MCP23017. También define la lista con los pines ordenados para poder iterar fácilmente por ella y modificar/leer todos los outputs/inputs y los nombres de las propiedades con las que se enviará el mensaje en caso de que se haya producido un cambio.

5.2.4.7 game.h

Librería principal del Proyecto, controla todos los componentes por medio de las clases/librerías anteriores y recopila los diferentes modos de juego posibles y su ejecución.

Este código no es especialmente extenso gracias la simplificación de las funciones en librerías, pero cuenta con más de 200 líneas de código (adjuntas con el resto en el apartado *ANEXO II: Código* en game.h), aunque el funcionamiento de los diferentes modos es parecido y se ha ido ampliando la funcionalidad.

A continuación, se explica en detalle el código desarrollado:

- Lo primero es importar las librerías (Arduino.h, wifi-mqtt.h, spkControl.h, MPU6050.h, PINS.h y Adafruit MCP23X17).
- Se instancian las clases con sus pintes o direcciones I2C correspondientes.
- También se definen 4 funciones requeridas para varios de los juegos y, aunque en los siguientes puntos se detalla su funcionamiento, en los siguientes apartados se explicará su funcionalidad en el juego:
    - `array_cmp()`: Recibe 2 punteros a arrays de booleanos y comprueba si todos sus valores son iguales. En caso de serlo, devuelve `true`, si alguno difiere, `false`.
    - `swap()`: Recibe 2 punteros a elementos de un array de enteros y los sustituye el uno por el otro (el valor de `a` a `b` y el valor de `b` a `a`).
    - `shuffle()`: Recibe un array de enteros y su longitud y lo "barajea". Es decir, itera por sus elementos sustituyendo por medio de la función anterior los elementos aleatoriamente (ejemplo: el 1 por el 4, el 2 por 1, el 3 por 5…).
    - `isInFirstn()`: Comprueba si un valor se encuentra entre los primeros n elementos de un array. Recibe el valor, el array y el número de elementos a comprobar.
- Por último, se define la **clase Game** con sus propiedades y funciones.

Clase Game

Para comenzar, se declaran las propiedades de la clase (variables de acceso privado):

- El listado del estado de los 6 inputs (1 presionado y 0 no presionado) que se va a leer y enviar a la plataforma de Altair. También se almacena el estado anterior de los inputs para comprobar en cada actualización de los mismos si se han modificado y así solo enviar el mensaje en caso de cambio.
- El listado de outputs viene inicializado y controlado por la librería *wifi-mqtt.h* por lo que solo se creará una variable para almacenar el estado anterior y, nuevamente, solo enviar los cambios en el vector.
- Variable de texto con la orientación (y orientación previa) en el formato indicado en el apartado del *MPU6050.h*.
- Variables de tiempo para el contador, indicando en ms cuando se inicia un juego y el tiempo actual.
- Booleano para guardar el estado de victoria y entero para registrar el número de victorias consecutivas.

Posteriormente, se definen las funciones públicas que contiene la clase:

- `Game()`: Constructor, inicializa las variables mencionadas previamente a 0 o vacías.
- `game.init()`: Esta función se llama en el `setup()` del main.cpp y recoge la inicialización de Serial, reset del MPU6050 y conexión con el MCP23017.
  Establece los pines del MCP como inputs con pullup (botones) y los pines del NodeMCU como outputs (LEDs) iterando a través de las listas definidas en *PINS.h*.
  Y por último se conecta al Wifi y MQTT por medio de las funciones definidas en la librería *wifi-mqtt.h*.

- `game.test()`: Modo de juego con valor 1 en la plataforma IoT. Permite testear el funcionamiento completo del Proyecto. Es decir, la conexión de los LEDs botones, acelerómetro y Wifi/MQTT.
  - Primero lee la orientación del juguete y, en caso de haberse modificado, lo notifica a la plataforma por MQTT.
  - Luego itera por todos los botones LED: Lee el estado del botón y, si se ha modificado, lo actualiza, tanto encendiendo/apagando el LED correspondiente como actualizando el valor en la plataforma para el botón y LED.

  Así, se dispone de un modo en el que al pulsar los botones se encienden. En un principio está diseñado para testear el funcionamiento, pero, incluyendo sonidos, se puede hacer que el juguete reaccione a los movimientos/pulsaciones, ya que está registrando la información y con un simple `if` se puede llamar al spkControl para reproducir sonidos en función de los botones.

- `game.game2()`: Segundo modo de los seleccionables en la plataforma IoT y define un modo de juego "offline" en el que se establece la configuración del juego y el resto se ejecuta en el microcontrolador.

  El funcionamiento del juego es el siguiente: Se van a encender un número de botones (equivalente a la dificultad que se haya establecido en la plataforma, entre 0 y 6) y se han de pulsar esos botones a la vez antes de finalizar un tiempo indicado (establecido también en la plataforma).
  - Para empezar, se inicializan las variables start_time y current_time al tiempo actual (en ms) y se pone el estado de victoria a False.
  - En este punto se hace uso de las funciones explicadas previamente.

    Se requiere definir un array aleatorio con unos en los botones que se van a encender (ej: {0,1,0,1,1,0}). El número de unos a escoger debe ser igual a la propiedad `dificulty` de la plataforma.

    Previamente, necesitamos un array con las posiciones de los n botones que se van a encender (para el caso anterior: {2,4,5}), pero esos números han de ser aleatorios y no se han de repetir.

La forma más eficiente (a nivel de tiempo de ejecución) de lograrlo es partiendo de un array con las 6 posiciones ({0,1,2,3,4,5}), reordenarlo aleatoriamente "barajándolo" con la función `shuffle()` (ej.: {2,4,5,1,0}) y luego, partiendo de un array de outputs vacío (con la función `memset()`: {0,0,0,0,0,0}), se va a iterar por cada uno de los botones y si la posición (índice `i` del bucle) se encuentra entre los n primeros elementos del array anterior (con la función `isInFirstn()` para dificultad 3 serían: {2,4,5}) entonces se encienden esos LEDs y se indica a la plataforma el array de outputs que son el objetivo a pulsar por el jugador.

Este proceso es rápido de ejecutar, pero requiere una programación algo compleja. En el *5.3.5 Programación IoT* se observará que el proceso es mucho más sencillo realizando la programación en Python desde la plataforma de Altair.

- Una vez definidos los botones a pulsar y encendidos, se procede a ejecutar un bucle durante un tiempo indicado por la propiedad `timeout` de la plataforma.

  Se comprueba que el tiempo transcurrido desde el inicio del juego sea menor a ese tiempo límite (`current_time - start_time < timeout`).

  En ese bucle se lee la orientación y la pulsación de los botones (registrando los eventos en la plataforma) y si los inputs concuerdan con los outputs (con `array_cmp()`) se cuenta una victoria y se sale del bucle con un `break`.
- Si al salir del bucle pasado el tiempo límite no se han pulsado los botones encendidos, se indica que se ha perdido y se resetea el contador de victorias consecutivas.

  En ambos casos – victoria o derrota – se reinicia el contador del juego, se escogen otros botones y se vuelve a empezar el juego.

Este juego es un sencillo ejemplo de programa configurable por medio de la plataforma. Es cierto que la variabilidad del juego complica su programación, pero permite añadir una curva de dificultad. A través de la plataforma es posible visualizar en tiempo real los progresos del jugador y ajustar la dificultad, en el próximo modo de juego y en el siguiente apartado (*5.3 Desarrollo IoT*) se podrá observar más en detalle el potencial de la plataforma de control de estas propiedades.

- `game.slave()`: Modo de juego número 3 y último modo debido a su funcionalidad. Permite recibir instrucciones de la plataforma y ejecutarlas a la vez que se temporiza un tiempo de juego y se notifica de todos los eventos a la plataforma como en el modo anterior.
- El objetivo de este modo de juego es una prueba de concepto. Se desea testear si es posible el funcionamiento completamente controlado por la plataforma de Altair. De esta forma, sería posible introducir más juegos sin alterar la programación de los juguetes. Esto permitiría la programación centralizada de los juegos en la misma plataforma de control, facilitando inconmensurablemente la mejora y versatilidad de la red de juguetes. Se entrará más en detalle en este concepto en el *0Ilustración 26:* Pestañas del Thing (detalles e interfaces)

5.2.5 PROPIEDADES DEL THING

Se han definido las siguientes propiedades para el Thing:

- 6 botones y 6 LEDs: Tipo number indicando su estado, 1 pulsado/encendido y 0 sin presionar/apagado (no booleano para poder indicar modos de iluminación, en el futuro, 2 podría ser parpadeo o definir el color en caso de RGB).
  El nombre de las propiedades es "led_" seguido del color y en el caso de los blancos se distingue la cara inferior de la superior con un "_up" o "_down".
- `game`: Número con el modo de juego del dispositivo (0 desactivado, 1 testeo, 2 juego offline, 3 slave de la plataforma).
- `orientation`: Valor en formato string legible del eje y de la cara superior.
- `outputs`: Array de valores con las instrucciones a activar los LEDs del juguete. Estos valores difieren de las variables con los estados al ser la instrucción indicada por la plataforma desde la que se pueden modificar las luces durante el juego. Una vez se haya encendido/apagado el botón indicado se actualizará el valor del "led_...".
- `print_message`: Mensaje enviado por la plataforma IoT al juguete para mostrar por pantalla en caso de victoria/derrota/error… También se puede utilizar como instrucción para el cambio de estado como se mencionó el en *Clase Game*.

- `state`: Define el estado del juguete como string; puede ser "Waiting" al comienzo del juego para esperar a la instrucción de outputs y comienzo del timer, "Running" mientras el juego esté corriendo – periodo durante el que se ha de comprobar si concuerdan los inputs y outputs objetivo en caso de victoria – y "Timeout" en caso de que se haya agotado el tiempo indicado en la configuración del juego (siguiente property) según el reloj del microcontrolador.
- `timeout`: Valor numérico en milisegundos que define el tiempo que dispone el jugador para introducir la secuencia correcta de inputs antes de entrar en estado de derrota.

5.2.6 HERRAMIENTAS (HISTÓRICO DE MENSAJES)

Pulsando encima de las pestañas de detalles/interfaces en el icono de "Raw History" se puede acceder al listado de mensajes recibidos en el topic correspondiente.

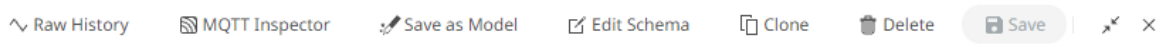


Ilustración 27: Barra de herramientas del Thing

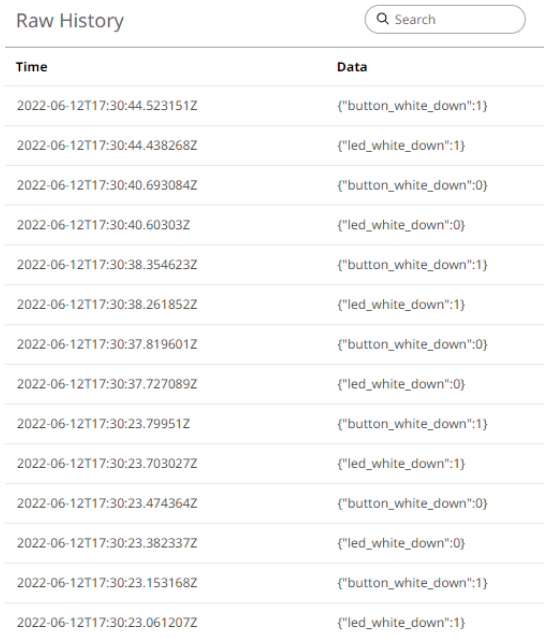


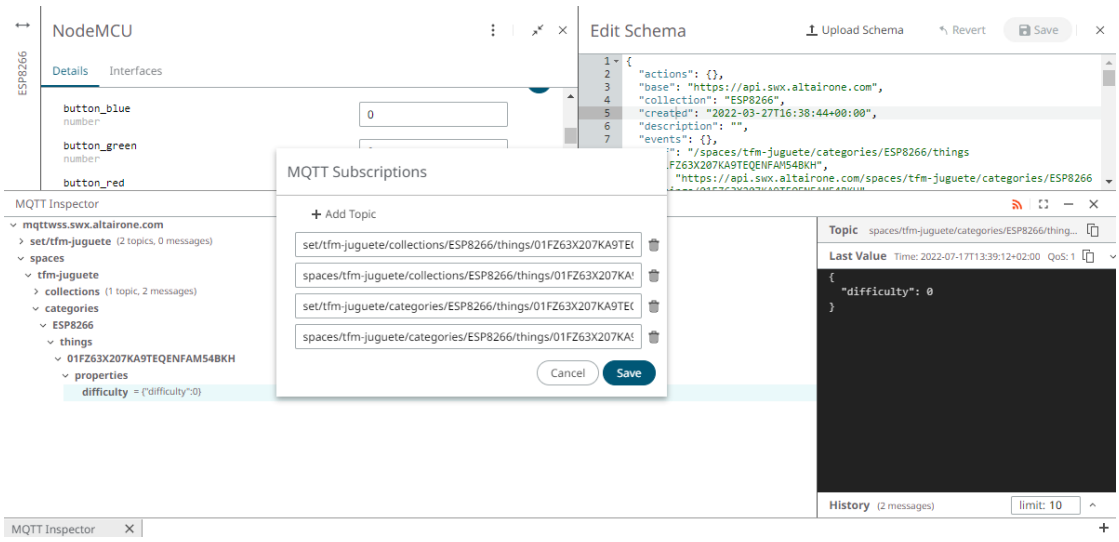
Ilustración 28: Histórico de mensajes

Aunque se actualiza en tiempo real y cuenta con un botón de refrescar, la plataforma también dispone de varias herramientas de inspección de los mensajes MQTT en la misma barra superior (junto con la posibilidad de guardar el Thing como "template", editarlo en formato Schema, clonarlo o eliminarlo).

El Schema es un formato JSON que define todas las propiedades y características del Thing de cara a exportarlo/importarlo con facilidad o editarlo en dicho formato, por ejemplo, para añadir las 12 propiedades de los botones/LEDs a la vez, más rápido que de uno en uno.

5.2.6.1 Inspector MQTT

En la siguiente imagen se muestra el inspector MQTT:



En la ventana de la izquierda se puede observar la estructura del topic y pulsando sobre él se muestran los mensajes publicados en la ventana de la derecha. Además, pulsando en el icono ( ) se listan los topics a los que se encuentra suscrito el inspector.

La estructura de topics es la siguiente:

1. Primero el space seguido de su nombre/identificador único, en este caso “tfm-juguete”.
2. Luego la collection y su nombre, como ya se ha mencionado corresponde a “ESP8266”.
3. Para acceder al Thing se indica su UID indicado en la ventana de interfaces.
4. Por último, se indica el recurso al que se quiera acceder:
    a. Si se quiere acceder a una propiedad se incluye “/properties/nombre_propiedad”.
    b. Si se quiere actualizar una propiedad se especifica “/properties” únicamente y en el mensaje se incluye el nombre de la propiedad.
    c. Y si se quiere publicar al histórico de mensajes se termina en “/data”.

Anteriormente se utilizaba un formato distinto con “set” para actualizar la property y “status” para leer su valor, pero se ha simplificado a un solo topic, y se va a volver a modificar con categorías, pero de momento se puede seguir utilizando de ambas formas.

Al cambiar el valor de una property manualmente en la plataforma también se envía un mensaje MQTT notificándolo. El topic resultante en este caso es: `spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/data`.

5.2.6.2 Schema del Thing (semi-compacto)


```
{ "actions": {},
  "base": "https://api.swx.altairone.com",
  "collection": "ESP8266",
  "created": "2022-03-27T16:38:44+00:00",
  "description": "",
  "events": {},
  "href": "/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "id": "https://api.swx.altairone.com/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "model": {"name": "","version": 0},
  "modified": "2022-05-29T21:09:28+00:00",
  "properties": {
    "button_blue": {"type": "number"},
    "button_green": {"type": "number"},
    "button_red": {"type": "number"},
    "button_white_down": {"type": "number"},
    "button_white_up": {"type": "number"},
    "button_yellow": {"type": "number"},
    "difficulty": {"type": "number"},
    "game": {"type": "number"},
    "led_blue": {"type": "number"},
    "led_green": {"type": "number"},
    "led_red": {"type": "number"},
    "led_white_down": {"type": "number"},
    "led_white_up": {"type": "number"},
    "led_yellow": {"type": "number"},
    "orientation": {"type": "string"},
    "outputs": {"items": [
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"}
      ], "type": "array"
    },
    "print_message": {"type": "string"},
    "state": {"type": "string"},
    "timeout": {"type": "number"}
  },
  "space": "tfm-juguete",
  "title": "NodeMCU",
  "uid": "01FZ63X207KA9TEQENFAM54BKH"}
```

5.2.7 FUNCIONES

En la barra de navegación de la izquierda se dispone, después del acceso a la AnythingDB, de las Funciones. Estas están divididas en 2 elementos:

Triggers

Los triggers de eventos son los activadores de las funciones y dictan cuando se las llama. Pueden ser de 2 tipos:

- Cron: Programados o temporizados, se ha definir la periodicidad con el mismo formato que la herramienta cron de Linux ("0 1 * * *" corresponde a una periodicidad diaria a la 01:00; siendo el primer 0 los minutos, el 1 las horas y los asteriscos indican que todos los días, meses y años).
- MQTT: La utilizada para el Proyecto ya que se activará al recibir un mensaje por un topic MQTT específico. En este caso, cada vez que el juguete indica que ha cambiado de estado (property "state"). Se ha de definir el tipo, descripción, bróker, puerto, nombre/contraseña MQTT, nivel de Quality of Service y topic como se muestra en la siguiente ilustración:

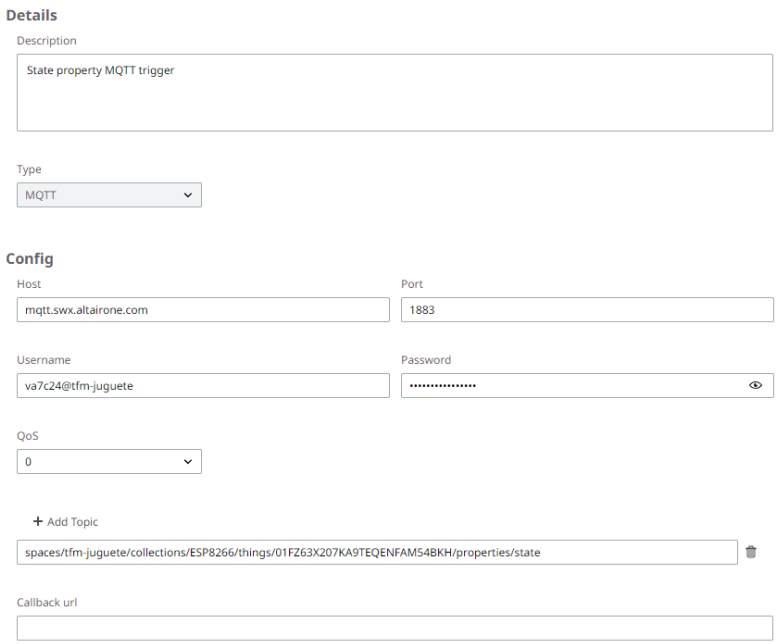


Ilustración 29: Trigger MQTT

5.2.7.1 Workers

Contienen el código de ejecución activado por los triggers que se hayan asignados.

Una vez definido su nombre, descripción y trigger en la primera pestaña, se ha de incluir el código en la segunda (partiendo de un template de Python/Go/FMU). La última pestaña de los workers permitirá testear el código, ejecutándolo e incluyendo el body del request manualmente, además de que es posible acceder a los logs de sus ejecuciones pasadas con su "response code".

En las siguientes imágenes se observa la interfaz de los workers en la plataforma:

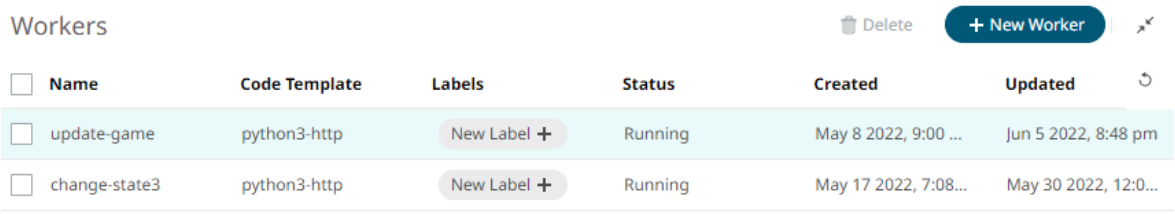


Ilustración 30: Listado de workers

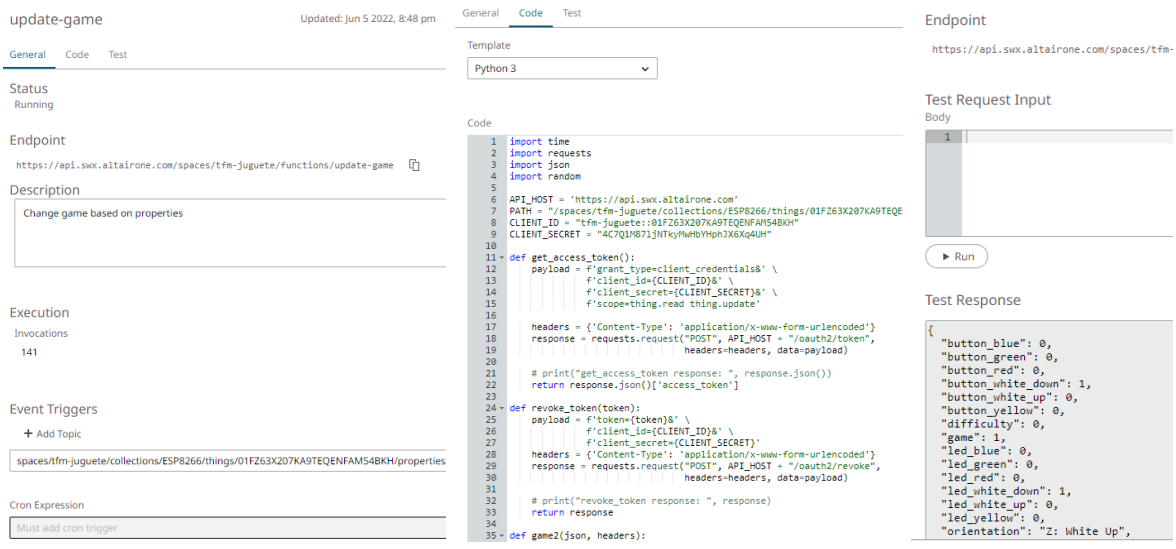


Ilustración 31: Workers (pestañas general, código y test)

Se explicará en detalle en el código desarrollado para estas funciones en el siguiente apartado (*5.3.5 Programación IoT*).

Además de los logs, es posible también acceder nuevamente al inspector MQTT, o a un inspector API para las request HTTP, desde el "Utility Belt" que es la barra inferior.

En la siguiente imagen se realiza una petición HTTP a la URL de las properties del Thing, devolviendo el JSON completo con sus valores. Para hacer dicha petición es necesario incluir en el header la autorización necesaria.

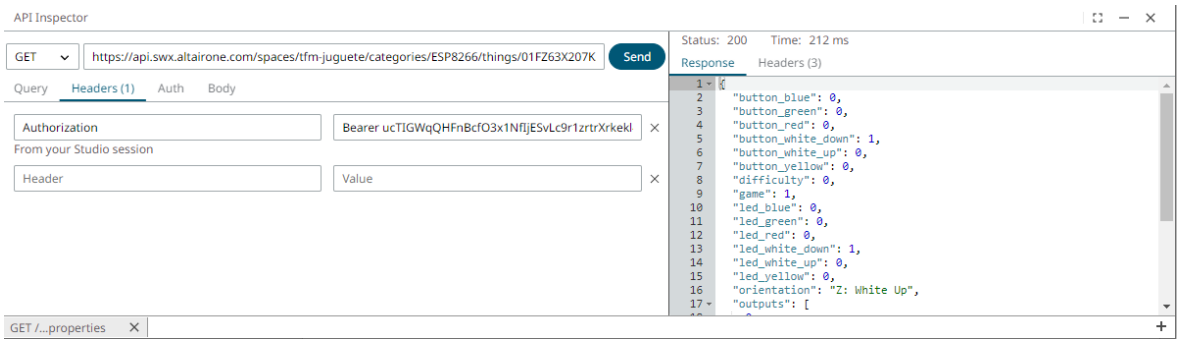


Ilustración 32: Inspector API (request HTTP)

5.2.7.2 Estados

Las funciones y los triggers pueden estar en varios estados al crearlas/modificarlas:

- Pending: Pendiente de creación/realización de la modificación en la plataforma.
- Building: Exclusivo de las funciones, compilando el código, proceso lento.
- Running: Preparada para ejecutarse manual/automáticamente.

Programar directamente en la plataforma de Altair es un proceso muy tedioso debido a la duración de dichos estados, por lo que se recomienda testear el código en Visual Studio y proceder a incluirlo una vez comprobado que funciona.

Para simular el trigger en VSCode es necesario subscribirse al mismo topic que del trigger "…/properties/state" y ejecutar el handle de la función, como callback, a la recepción de un mensaje. Como el worker programado no requiere de body en el mensaje, pasar el código de Visual Studio a la plataforma de Altair es copiar y pegar. En detalle en el siguiente apartado.

5.2.8 PROGRAMACIÓN IOT

La programación del worker se hará en Python. El código incluido en el worker y el ejecutable en Visual Studio se adjuntan en el *ANEXO II: Código*, en el *7.1 Código de la plataforma IoT*. Como ya se ha mencionado y se detalla en dicho apartado, el código incluye 2 partes:

1. Código ejecutable en la plataforma por el worker llamado "update_game". El cuál sigue el siguiente proceso:
   - Se importan 4 librerías de Python:
     - time: Para la conversión y el manejo de unidades de tiempo.
     - requests: Funciones necesarias para la comunicación por HTTP.
     - json: Requerida para el decoding de los payloads recibidos/enviados.
   - random: Función para la generación de números pseudoaleatorios.
   - Definición de 4 constantes (dominio HTTP de Altair, dirección URL a acceder y usuario y contraseña HTTP para el acceso.
   - Funciones `get_access_token()` y `revoke_token()`: Han sido aportadas por Altair y ejecutan una request HTTP tipo POST para obtener un token de autorización temporal de lectura y modificación de las propiedades de la Thing.
     La primera devuelve el token, la segunda lo recibe como parámetro y lo anula.
   - Función `game2()`: Simula el mismo proceso que la función game2() de la *Clase Game* de la librería *game.h*, pero enviándolo las instrucciones necesarias al juguete.

     En función del estado del juguete ejecuta las siguientes instrucciones:
     - Si el juego se está ejecutando (estado "Running") se ha de revisar que los estados de los botones sean iguales a los outputs objetivo, en caso de serlo se envía un mensaje de victoria. Si el estado de los inputs cambia, pero no es el correcto se envía un mensaje para mostrar por pantalla de seguir intentándolo.

- Si el juguete detecta que ha transcurrido el tiempo límite de juego y entra en modo "Timeout" se ha de enviar un mensaje de derrota.

  Si el juego acaba de ser seleccionado y el juguete está esperando instrucciones ("Waiting"), se han de indicar los LEDs que se han de encender, que serán los botones objetivo a pulsar por el jugador. Al igual que en la función `game2()`, es necesario crear un array de 6 elementos con ceros un número de unos igual a la dificultad seleccionada.

  En Python esto es muy sencillo, gracias a la función `random.sample()`, que devolverá n elementos de un array de 6 (`range(0,6)=[0,1,2,3,4,5]`, `sample(range, 2)=[2,4]`).

  Una vez seleccionados n LEDs a encender se utiliza "list comprehension" de Python, que permite crear una lista haciendo un bucle de 0 a 5 y si la posición está en el sample anterior se pone a 1 y sino a 0.

  Por último, se envía un mensaje indicando los outputs a encender y dando comienzo al programa.

- `handle()`: Función que se ejecuta en caso de llamar al worker con el trigger.

  - Primero obtiene el token para las peticiones HTTP y hace una request HTTP GET a las propiedades del Thing de la plataforma, está información se almacena en una variable json.

    Se ha de tener la cuenta que al estar suscrito al topic "state" ya se dispone del valor del estado del juguete, pero no del resto de las propiedades, es por ello que se realiza esta petición.

  - Si el juguete está en modo de juego 3 (slave) - en el que sigue las instrucciones de la plataforma – entonces se ejecuta la función `game2()`, en caso contrario se revoca el token y se devuelve el JSON con las propiedades.

2. La segunda parte simula la suscripción del trigger al topic "state" y la activación del handle del worker mencionado previamente.

   - Para ello, se importará una librería de Python llamada paho.mqtt.client.
   - Se definen las constantes MQTT (dominio, usuario, contraseña y topic).
   - `on_connect()`: Indica cuando se ha realizado la conexión en el terminal y se suscribe al topic indicado.
   - `on_message()`: Muestra el mensaje recibido y llama al handle del worker.
   - `handle1()`: Función principal. Crea el cliente, conecta las funciones anteriores a la conexión/recepción de mensajes, define el usuario y contraseña y se conecta al servidor de Altair.
   - Como el código se ejecuta como "`__main__`" por defecto al correr el archivo en el terminal, se ha de incluir un `if` para asegurar que si se está ejecutando el archivo (no importando) se llame al `handle1()` sin body (req=0).

El worker hace uso únicamente de la comunicación HTTP, aunque se activa por medio de un mensaje MQTT. Aun así, al enviar mensajes a la plataforma por HTTP para modificar las propiedades se genera un mensaje MQTT indicando su modificación. Este es el mensaje recibido por el juguete al estar suscrito a los topics de ciertas propiedades.

Este programa, como se puede observar, es muy fácil de escalar y ampliar con más juegos, debido a la simplicidad de control del juguete por medio de mensajes. Al modificar la programación de la plataforma es posible modificar el código de todos los juguetes conectados a la red IoT sin necesidad de conectarlos al ordenador y subir el código manualmente. Se entrará más en detalle acerca de su potencial en el siguiente capítulo.

5.2.9 VISUALIZACIÓN DE LOS DATOS

Aunque es posible visualizar los valores de las propiedades y su histórico en la propia Thing, es preferible crear un dashboard.

5.2.9.1 Apps

Para ello, primero es necesario crear una App (apartado de Stream Processing). Las aplicaciones permiten conectar las Data Sources con otras herramientas. Para el Proyecto se han creado 2 aplicaciones:

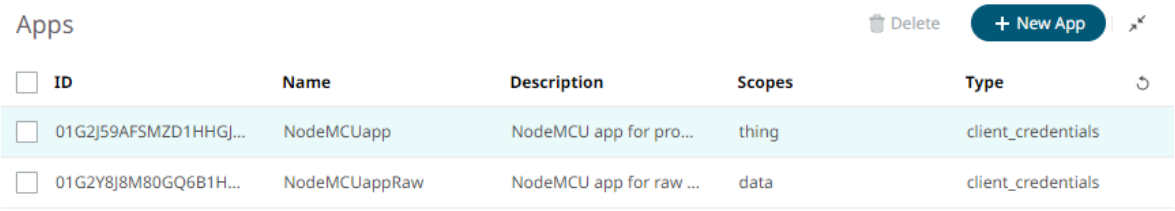


Ilustración 33: Apps IoT

- NodeMCUpropertiesDT: Es de scope "thing" y tiene acceso a las propiedades de la Thing asociada.
- NodeMCUMCUrawDT: Es de scope "data", permite acceder al historial completo de mensajes.

Es posible realizar una sola aplicación con acceso a ambas Data Tables, pero es preferible mantenerlas separadas. Las aplicaciones también cuentan con su propio Client ID y Client Secret de acceso como las Thing (en caso de ser tipo "client credentials").

En las siguientes capturas se muestran las pestañas de detalles de configuración y políticas de acceso, ya que es posible definir diferentes accesos (creación, lectura, modificación, borrado) de las Things. El resource indicado es "#" para simbolizar el acceso a todos:

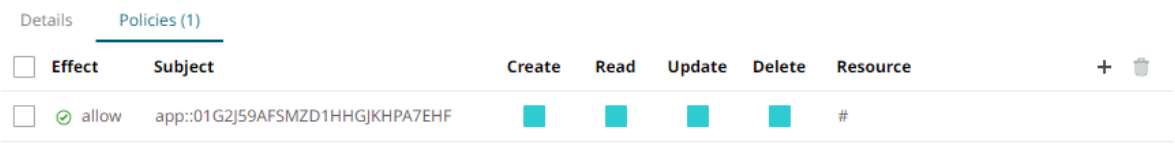


Ilustración 34: Apps IoT - Pestaña de policies

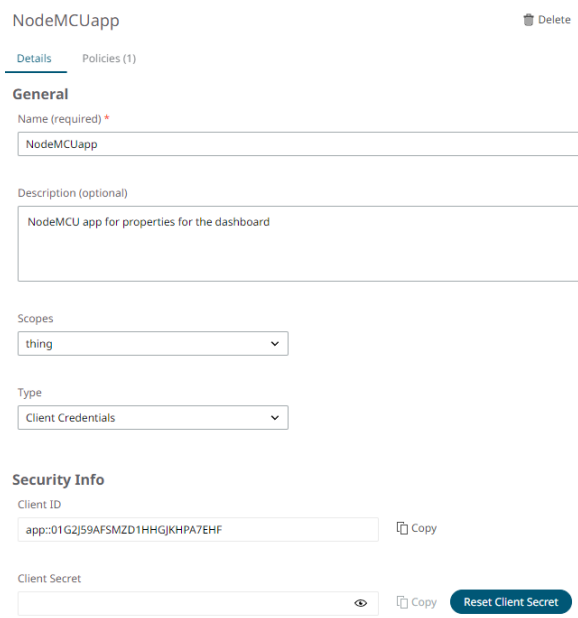


Ilustración 35: Apps IoT - Pestaña de detalles

5.2.9.2 Workbooks

Una vez creadas las Apps es posible conectarlas a un Workbook (en el apartado de Real Time Visualization) que contiene los dashboards. Es necesario crear la Data Table conectando el workbook con la App (definir credenciales, scope, URL…) y definir las columnas (o generarlas automáticamente). En la siguiente ilustración se muestra la configuración del primer workbook:

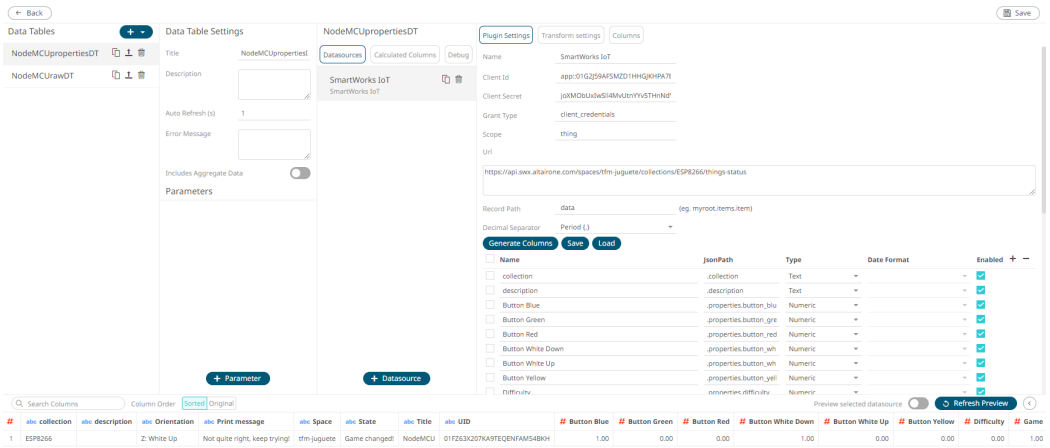


Ilustración 36: Workbook settings

En el caso del histórico, es necesario activar la opción de "Tansform to enable time series analysis" en "Transform Settings" para poder visualizar el tiempo de recepción en formato fecha y utilizar las gráficas correspondientes.

Con cada una de las aplicaciones se ha diseñado un dashboard. El primero muestra de forma visual las propiedades y estado del juguete (inputs, outputs y variables):

Ilustración 37: Dashboard

El segundo simplemente es una tabla con la lista de mensajes y una gráfica con los estados 0-1 en el tiempo, pero debido al volumen de registros (mayor a 100000) y otros fallos (que se han notificado a Panopticon que es el proveedor de la herramienta gráfica de dashboard de Altair) no muestra correctamente los resultados.

A futuro, este dashboard se puede ampliar, no solo con el histórico de datos, sino incluyendo más información de los sensores que ya se recoge en el dispositivo (como las aceleraciones en cada eje, o los tiempos que tarda el jugador en pasar el juego) y brindando la posibilidad de mostrar varios dispositivos simultáneamente.

En caso de poder conectar una red de dispositivos la información obtenida del aprendizaje de todos los niños jugando se puede utilizar para mejorar el sistema y añadir juegos o curvas de dificultad personalizadas. El análisis de estos puntos se realiza en el siguiente capítulo.

Análisis de Resultados. A continuación, se detalla su programación:

- Una vez activado el modo slave se ha de actualizar el estado en la plataforma de Altair a `"Waiting"`. A partir de entonces la plataforma indicará las instrucciones (outputs) que ejecutar y se recibirán por Wifi-MQTT. El último mensaje que envía es `{"print_message"="Started!"}`, el cual se imprime por pantalla dando lugar al comienzo del juego.
- Se actualiza el estado del juguete en la plataforma a `"Running"` y – como en el modo anterior – se establece el tiempo inicial y actual y se inicia un bucle durante el periodo indicado en la plataforma. Pero, esta vez, la condición de salida también será el cambio de modo de juego en la plataforma (`game_value != 3`) o la recepción de un mensaje de victoria (`print_message != "Victory!"`), ya que es la plataforma IoT la que comprobará que los inputs concuerdan con los objetivo.
- En el bucle la programación es parecida a casos anteriores. Se obtiene la orientación, el estado de los pines (iterando) y se encienden/apagan los LEDs según indique la plataforma, siempre actualizando los valores en la misma por MQTT.

  Al final de cada bucle se realiza un delay de 100ms, se comprueba el envío/recepción de paquetes y, en caso de mensajes a mostrar por pantalla, se imprimen.
- Finalmente, al salir del bucle, si el tiempo transcurrido es superior al límite, no se ha cambiado de juego ni se ha recibido mensaje de victoria se actualiza el estado del juguete a `"Timeout"`.

  Si se ha cambiado de juego se indica y se realiza un delay de 1 segundo.

  Y, en caso de victoria, el mensaje se muestra por pantalla automáticamente independientemente del modo de juego.

- Esta sería la programación principal del juguete. Como se puede ver hace uso de todas las librerías anteriores y se ha podido reducir significativamente el código a un nivel

comprensible. Por supuesto, siempre es mejorable a nivel de tiempos de computación, pero tampoco es relevante su optimización dado que el cuello de botella respecto a latencia está en la comunicación con la plataforma. No obstante, se puede observar en todos los modos de juegos (en especial en el último) que la latencia no es un factor notable ni perjudicial dado que es perfectamente jugable y – a falta de testearlo en situaciones más desfavorables de conexión – se puede decir que no es un factor significativo, como se describe más en detalle en el *0Ilustración 26:* Pestañas del Thing (detalles e interfaces)

5.2.10 PROPIEDADES DEL THING

Se han definido las siguientes propiedades para el Thing:

- 6 botones y 6 LEDs: Tipo number indicando su estado, 1 pulsado/encendido y 0 sin presionar/apagado (no booleano para poder indicar modos de iluminación, en el futuro, 2 podría ser parpadeo o definir el color en caso de RGB).
  El nombre de las propiedades es "led_" seguido del color y en el caso de los blancos se distingue la cara inferior de la superior con un "_up" o "_down".
- `game`: Número con el modo de juego del dispositivo (0 desactivado, 1 testeo, 2 juego offline, 3 slave de la plataforma).
- `orientation`: Valor en formato string legible del eje y de la cara superior.
- `outputs`: Array de valores con las instrucciones a activar los LEDs del juguete. Estos valores difieren de las variables con los estados al ser la instrucción indicada por la plataforma desde la que se pueden modificar las luces durante el juego. Una vez se haya encendido/apagado el botón indicado se actualizará el valor del "led_...".
- `print_message`: Mensaje enviado por la plataforma IoT al juguete para mostrar por pantalla en caso de victoria/derrota/error… También se puede utilizar como instrucción para el cambio de estado como se mencionó el en *Clase Game*.
- `state`: Define el estado del juguete como string; puede ser "Waiting" al comienzo del juego para esperar a la instrucción de outputs y comienzo del timer, "Running" mientras el juego esté corriendo – periodo durante el que se ha de comprobar si

concuerdan los inputs y outputs objetivo en caso de victoria – y "Timeout" en caso de que se haya agotado el tiempo indicado en la configuración del juego (siguiente property) según el reloj del microcontrolador.

- `timeout`: Valor numérico en milisegundos que define el tiempo que dispone el jugador para introducir la secuencia correcta de inputs antes de entrar en estado de derrota.

5.2.11 HERRAMIENTAS (HISTÓRICO DE MENSAJES)

Pulsando encima de las pestañas de detalles/interfaces en el icono de "Raw History" se puede acceder al listado de mensajes recibidos en el topic correspondiente.

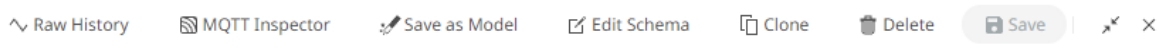


Ilustración 27: Barra de herramientas del Thing

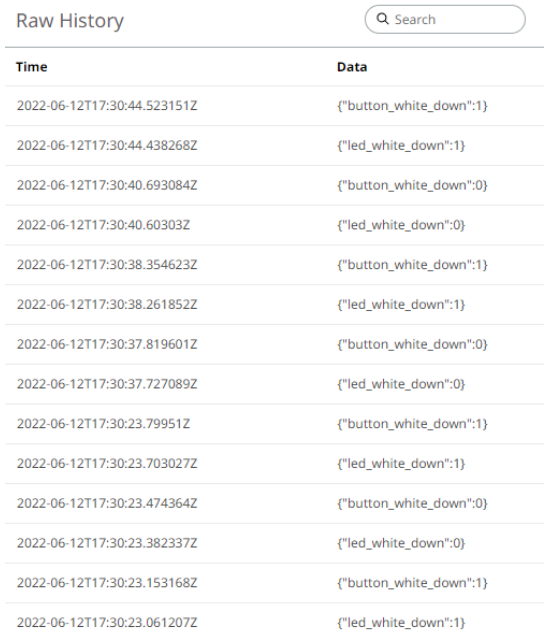


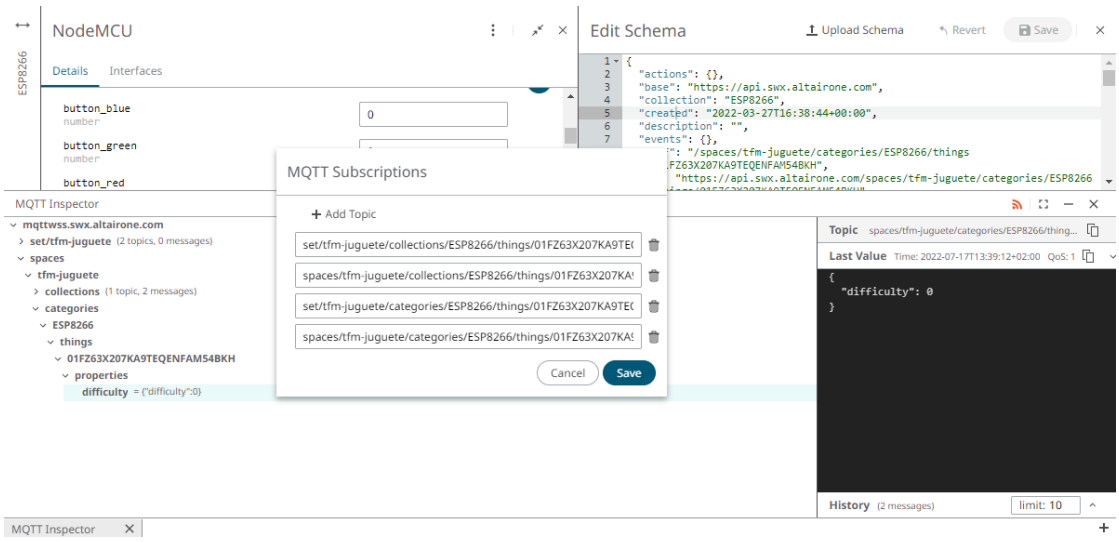
Ilustración 28: Histórico de mensajes

Aunque se actualiza en tiempo real y cuenta con un botón de refrescar, la plataforma también dispone de varias herramientas de inspección de los mensajes MQTT en la misma barra superior (junto con la posibilidad de guardar el Thing como "template", editarlo en formato Schema, clonarlo o eliminarlo).

El Schema es un formato JSON que define todas las propiedades y características del Thing de cara a exportarlo/importarlo con facilidad o editarlo en dicho formato, por ejemplo, para añadir las 12 propiedades de los botones/LEDs a la vez, más rápido que de uno en uno.

5.2.11.1 Inspector MQTT

En la siguiente imagen se muestra el inspector MQTT:



En la ventana de la izquierda se puede observar la estructura del topic y pulsando sobre él se muestran los mensajes publicados en la ventana de la derecha. Además, pulsando en el icono ( ) se listan los topics a los que se encuentra suscrito el inspector.

La estructura de topics es la siguiente:

5. Primero el space seguido de su nombre/identificador único, en este caso “tfm-juguete”.
6. Luego la collection y su nombre, como ya se ha mencionado corresponde a “ESP8266”.
7. Para acceder al Thing se indica su UID indicado en la ventana de interfaces.
8. Por último, se indica el recurso al que se quiera acceder:
    a. Si se quiere acceder a una propiedad se incluye “/properties/nombre_propiedad”.
    b. Si se quiere actualizar una propiedad se especifica “/properties” únicamente y en el mensaje se incluye el nombre de la propiedad.
    c. Y si se quiere publicar al histórico de mensajes se termina en “/data”.

Anteriormente se utilizaba un formato distinto con “set” para actualizar la property y “status” para leer su valor, pero se ha simplificado a un solo topic, y se va a volver a modificar con categorías, pero de momento se puede seguir utilizando de ambas formas.

Al cambiar el valor de una property manualmente en la plataforma también se envía un mensaje MQTT notificándolo. El topic resultante en este caso es: `spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/data`.

5.2.11.2 Schema del Thing (semi-compacto)


```
{ "actions": {},
  "base": "https://api.swx.altairone.com",
  "collection": "ESP8266",
  "created": "2022-03-27T16:38:44+00:00",
  "description": "",
  "events": {},
  "href": "/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "id": "https://api.swx.altairone.com/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "model": {"name": "","version": 0},
  "modified": "2022-05-29T21:09:28+00:00",
  "properties": {
    "button_blue": {"type": "number"},
    "button_green": {"type": "number"},
    "button_red": {"type": "number"},
    "button_white_down": {"type": "number"},
    "button_white_up": {"type": "number"},
    "button_yellow": {"type": "number"},
    "difficulty": {"type": "number"},
    "game": {"type": "number"},
    "led_blue": {"type": "number"},
    "led_green": {"type": "number"},
    "led_red": {"type": "number"},
    "led_white_down": {"type": "number"},
    "led_white_up": {"type": "number"},
    "led_yellow": {"type": "number"},
    "orientation": {"type": "string"},
    "outputs": {"items": [
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"}
      ], "type": "array"
    },
    "print_message": {"type": "string"},
    "state": {"type": "string"},
    "timeout": {"type": "number"}
  },
  "space": "tfm-juguete",
  "title": "NodeMCU",
  "uid": "01FZ63X207KA9TEQENFAM54BKH"}
```

5.2.12 FUNCIONES

En la barra de navegación de la izquierda se dispone, después del acceso a la AnythingDB, de las Funciones. Estas están divididas en 2 elementos:

Triggers

Los triggers de eventos son los activadores de las funciones y dictan cuando se las llama. Pueden ser de 2 tipos:

- Cron: Programados o temporizados, se ha definir la periodicidad con el mismo formato que la herramienta cron de Linux ("0 1 * * *" corresponde a una periodicidad diaria a la 01:00; siendo el primer 0 los minutos, el 1 las horas y los asteriscos indican que todos los días, meses y años).
- MQTT: La utilizada para el Proyecto ya que se activará al recibir un mensaje por un topic MQTT específico. En este caso, cada vez que el juguete indica que ha cambiado de estado (property "state"). Se ha de definir el tipo, descripción, bróker, puerto, nombre/contraseña MQTT, nivel de Quality of Service y topic como se muestra en la siguiente ilustración:

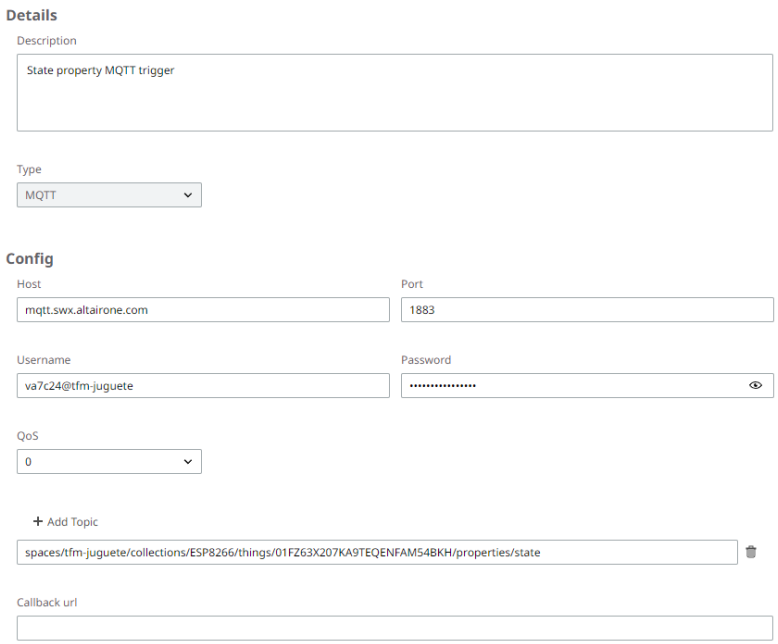


Ilustración 29: Trigger MQTT

5.2.12.1 Workers

Contienen el código de ejecución activado por los triggers que se hayan asignados.

Una vez definido su nombre, descripción y trigger en la primera pestaña, se ha de incluir el código en la segunda (partiendo de un template de Python/Go/FMU). La última pestaña de los workers permitirá testear el código, ejecutándolo e incluyendo el body del request manualmente, además de que es posible acceder a los logs de sus ejecuciones pasadas con su "response code".

En las siguientes imágenes se observa la interfaz de los workers en la plataforma:

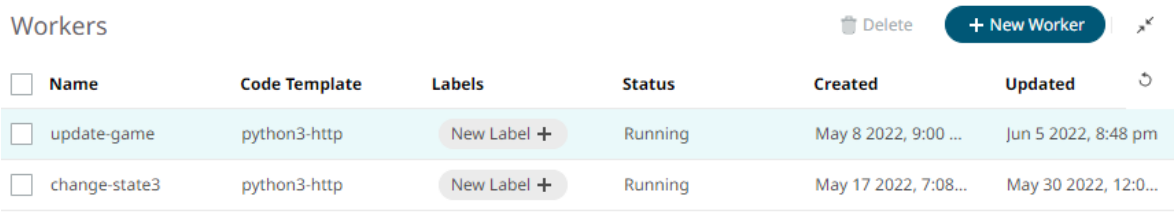


Ilustración 30: Listado de workers

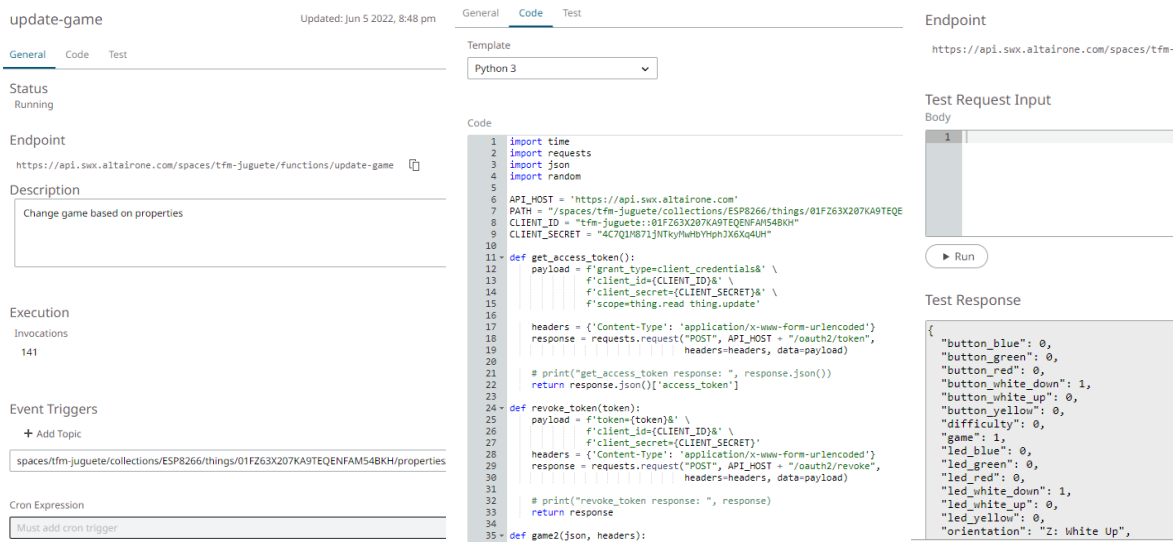


Ilustración 31: Workers (pestañas general, código y test)

Se explicará en detalle en el código desarrollado para estas funciones en el siguiente apartado (*5.3.5 Programación IoT*).

Además de los logs, es posible también acceder nuevamente al inspector MQTT, o a un inspector API para las request HTTP, desde el "Utility Belt" que es la barra inferior.

En la siguiente imagen se realiza una petición HTTP a la URL de las properties del Thing, devolviendo el JSON completo con sus valores. Para hacer dicha petición es necesario incluir en el header la autorización necesaria.

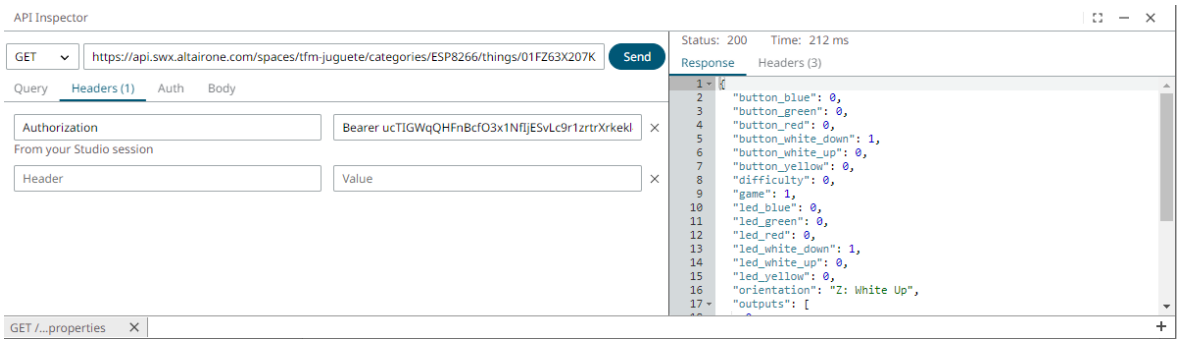


Ilustración 32: Inspector API (request HTTP)

5.2.12.2 Estados

Las funciones y los triggers pueden estar en varios estados al crearlas/modificarlas:

- Pending: Pendiente de creación/realización de la modificación en la plataforma.
- Building: Exclusivo de las funciones, compilando el código, proceso lento.
- Running: Preparada para ejecutarse manual/automáticamente.

Programar directamente en la plataforma de Altair es un proceso muy tedioso debido a la duración de dichos estados, por lo que se recomienda testear el código en Visual Studio y proceder a incluirlo una vez comprobado que funciona.

Para simular el trigger en VSCode es necesario subscribirse al mismo topic que del trigger "…/properties/state" y ejecutar el handle de la función, como callback, a la recepción de un mensaje. Como el worker programado no requiere de body en el mensaje, pasar el código de Visual Studio a la plataforma de Altair es copiar y pegar. En detalle en el siguiente apartado.

5.2.13 PROGRAMACIÓN IOT

La programación del worker se hará en Python. El código incluido en el worker y el ejecutable en Visual Studio se adjuntan en el *ANEXO II: Código*, en el *7.1 Código de la plataforma IoT*. Como ya se ha mencionado y se detalla en dicho apartado, el código incluye 2 partes:

3. Código ejecutable en la plataforma por el worker llamado "update_game". El cuál sigue el siguiente proceso:
   - Se importan 4 librerías de Python:
     - time: Para la conversión y el manejo de unidades de tiempo.
     - requests: Funciones necesarias para la comunicación por HTTP.
     - json: Requerida para el decoding de los payloads recibidos/enviados.
   - random: Función para la generación de números pseudoaleatorios.
   - Definición de 4 constantes (dominio HTTP de Altair, dirección URL a acceder y usuario y contraseña HTTP para el acceso.
   - Funciones `get_access_token()` y `revoke_token()`: Han sido aportadas por Altair y ejecutan una request HTTP tipo POST para obtener un token de autorización temporal de lectura y modificación de las propiedades de la Thing.
     La primera devuelve el token, la segunda lo recibe como parámetro y lo anula.
   - Función `game2()`: Simula el mismo proceso que la función game2() de la *Clase Game* de la librería *game.h*, pero enviándolo las instrucciones necesarias al juguete.

     En función del estado del juguete ejecuta las siguientes instrucciones:
     - Si el juego se está ejecutando (estado "Running") se ha de revisar que los estados de los botones sean iguales a los outputs objetivo, en caso de serlo se envía un mensaje de victoria. Si el estado de los inputs cambia, pero no es el correcto se envía un mensaje para mostrar por pantalla de seguir intentándolo.

- Si el juguete detecta que ha transcurrido el tiempo límite de juego y entra en modo "Timeout" se ha de enviar un mensaje de derrota.

  Si el juego acaba de ser seleccionado y el juguete está esperando instrucciones ("Waiting"), se han de indicar los LEDs que se han de encender, que serán los botones objetivo a pulsar por el jugador. Al igual que en la función `game2()`, es necesario crear un array de 6 elementos con ceros un número de unos igual a la dificultad seleccionada.

  En Python esto es muy sencillo, gracias a la función `random.sample()`, que devolverá n elementos de un array de 6 (`range(0,6)=[0,1,2,3,4,5]`, `sample(range, 2)=[2,4]`).

  Una vez seleccionados n LEDs a encender se utiliza "list comprehension" de Python, que permite crear una lista haciendo un bucle de 0 a 5 y si la posición está en el sample anterior se pone a 1 y sino a 0.

  Por último, se envía un mensaje indicando los outputs a encender y dando comienzo al programa.

- `handle()`: Función que se ejecuta en caso de llamar al worker con el trigger.

  - Primero obtiene el token para las peticiones HTTP y hace una request HTTP GET a las propiedades del Thing de la plataforma, está información se almacena en una variable json.

    Se ha de tener la cuenta que al estar suscrito al topic "state" ya se dispone del valor del estado del juguete, pero no del resto de las propiedades, es por ello que se realiza esta petición.

  - Si el juguete está en modo de juego 3 (slave) - en el que sigue las instrucciones de la plataforma – entonces se ejecuta la función `game2()`, en caso contrario se revoca el token y se devuelve el JSON con las propiedades.

4. La segunda parte simula la suscripción del trigger al topic "state" y la activación del handle del worker mencionado previamente.

    - Para ello, se importará una librería de Python llamada paho.mqtt.client.
    - Se definen las constantes MQTT (dominio, usuario, contraseña y topic).
    - `on_connect()`: Indica cuando se ha realizado la conexión en el terminal y se suscribe al topic indicado.
    - `on_message()`: Muestra el mensaje recibido y llama al handle del worker.
    - `handle1()`: Función principal. Crea el cliente, conecta las funciones anteriores a la conexión/recepción de mensajes, define el usuario y contraseña y se conecta al servidor de Altair.
    - Como el código se ejecuta como "`__main__`" por defecto al correr el archivo en el terminal, se ha de incluir un `if` para asegurar que si se está ejecutando el archivo (no importando) se llame al `handle1()` sin body (req=0).

El worker hace uso únicamente de la comunicación HTTP, aunque se activa por medio de un mensaje MQTT. Aun así, al enviar mensajes a la plataforma por HTTP para modificar las propiedades se genera un mensaje MQTT indicando su modificación. Este es el mensaje recibido por el juguete al estar suscrito a los topics de ciertas propiedades.

Este programa, como se puede observar, es muy fácil de escalar y ampliar con más juegos, debido a la simplicidad de control del juguete por medio de mensajes. Al modificar la programación de la plataforma es posible modificar el código de todos los juguetes conectados a la red IoT sin necesidad de conectarlos al ordenador y subir el código manualmente. Se entrará más en detalle acerca de su potencial en el siguiente capítulo.

5.2.14 VISUALIZACIÓN DE LOS DATOS

Aunque es posible visualizar los valores de las propiedades y su histórico en la propia Thing, es preferible crear un dashboard.

5.2.14.1 Apps

Para ello, primero es necesario crear una App (apartado de Stream Processing). Las aplicaciones permiten conectar las Data Sources con otras herramientas. Para el Proyecto se han creado 2 aplicaciones:

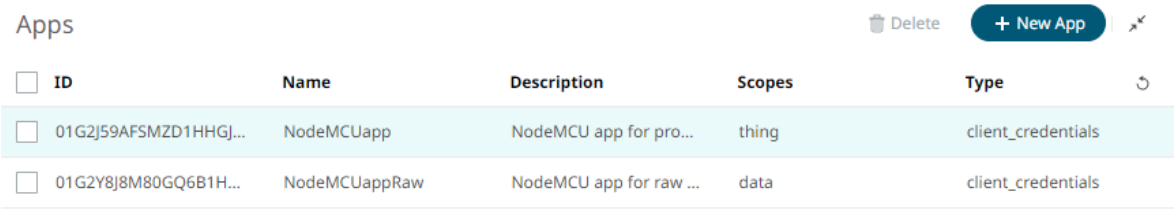


Ilustración 33: Apps IoT

- NodeMCUpropertiesDT: Es de scope "thing" y tiene acceso a las propiedades de la Thing asociada.
- NodeMCUMCUrawDT: Es de scope "data", permite acceder al historial completo de mensajes.

Es posible realizar una sola aplicación con acceso a ambas Data Tables, pero es preferible mantenerlas separadas. Las aplicaciones también cuentan con su propio Client ID y Client Secret de acceso como las Thing (en caso de ser tipo "client credentials").

En las siguientes capturas se muestran las pestañas de detalles de configuración y políticas de acceso, ya que es posible definir diferentes accesos (creación, lectura, modificación, borrado) de las Things. El resource indicado es "#" para simbolizar el acceso a todos:

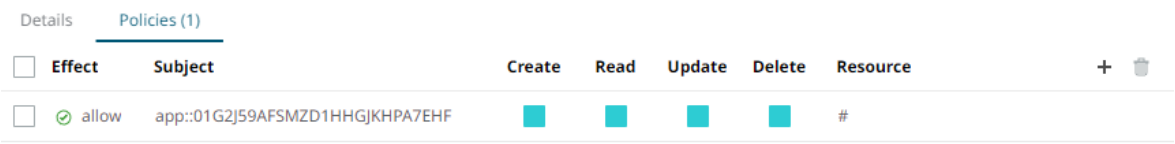


Ilustración 34: Apps IoT - Pestaña de policies

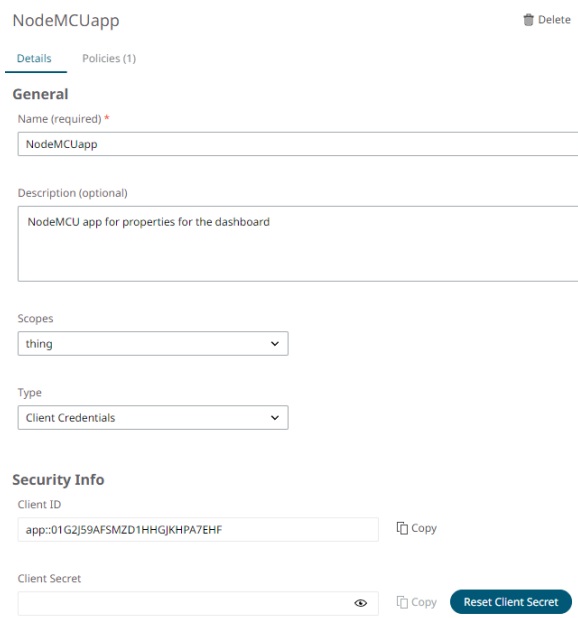


Ilustración 35: Apps IoT - Pestaña de detalles

5.2.14.2 Workbooks

Una vez creadas las Apps es posible conectarlas a un Workbook (en el apartado de Real Time Visualization) que contiene los dashboards. Es necesario crear la Data Table conectando el workbook con la App (definir credenciales, scope, URL…) y definir las columnas (o generarlas automáticamente). En la siguiente ilustración se muestra la configuración del primer workbook:

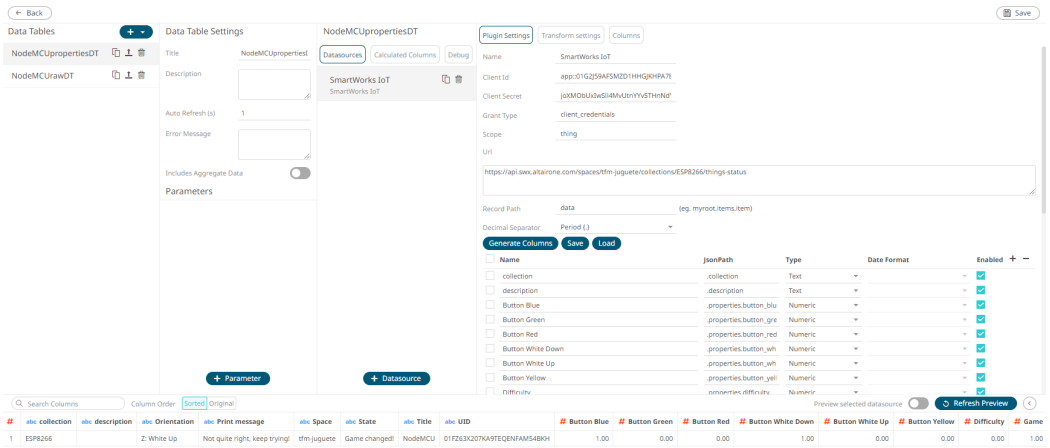


Ilustración 36: Workbook settings

En el caso del histórico, es necesario activar la opción de "Tansform to enable time series analysis" en "Transform Settings" para poder visualizar el tiempo de recepción en formato fecha y utilizar las gráficas correspondientes.

Con cada una de las aplicaciones se ha diseñado un dashboard. El primero muestra de forma visual las propiedades y estado del juguete (inputs, outputs y variables):

Ilustración 37: Dashboard

El segundo simplemente es una tabla con la lista de mensajes y una gráfica con los estados 0-1 en el tiempo, pero debido al volumen de registros (mayor a 100000) y otros fallos (que se han notificado a Panopticon que es el proveedor de la herramienta gráfica de dashboard de Altair) no muestra correctamente los resultados.

A futuro, este dashboard se puede ampliar, no solo con el histórico de datos, sino incluyendo más información de los sensores que ya se recoge en el dispositivo (como las aceleraciones en cada eje, o los tiempos que tarda el jugador en pasar el juego) y brindando la posibilidad de mostrar varios dispositivos simultáneamente.

En caso de poder conectar una red de dispositivos la información obtenida del aprendizaje de todos los niños jugando se puede utilizar para mejorar el sistema y añadir juegos o curvas de dificultad personalizadas. El análisis de estos puntos se realiza en el siguiente capítulo.

Análisis de Resultados.

5.3 DESARROLLO IOT

El último proceso, y de los más importantes en el desarrollo del Proyecto, es el desarrollo en la plataforma IoT de Altair SmartWorks. En este apartado se explicarán el funcionamiento de la herramienta y la configuración y programación de sus funciones.

5.3.1 NAVEGACIÓN POR LA HERRAMIENTA

Para empezar, nada más iniciar sesión se muestra la pantalla de inicio con los diferentes espacios creados o compartidos contigo. Es una herramienta de desarrollo IoT colaborativa, por lo que se pueden modificar los accesos de estos espacios. En la siguiente ilustración se muestra dicha pantalla inicial:

Ilustración 24: Pantalla de inicio de Altair SmartWorks IoT

El espacio en el que se ha desarrollado el Proyecto se ha llamado "tfm-juguete", es un nombre identificativo único dentro de la plataforma. Una vez accedido al espacio se muestra la ventana de AnythingDB, que es una base de datos separada por colecciones, y, dentro de las colecciones, se dispone de "Things" que en nuestro caso son los juguetes. La colección del Proyecto se ha llamado "ESP8266" como el procesador de los dispositivos y dentro de la colección el primer dispositivo conectado es el NodeMCU.

Seleccionando el dispositivo se pueden ver 2 pestañas (detalles e interfaces), como se muestra en la siguiente ilustración:

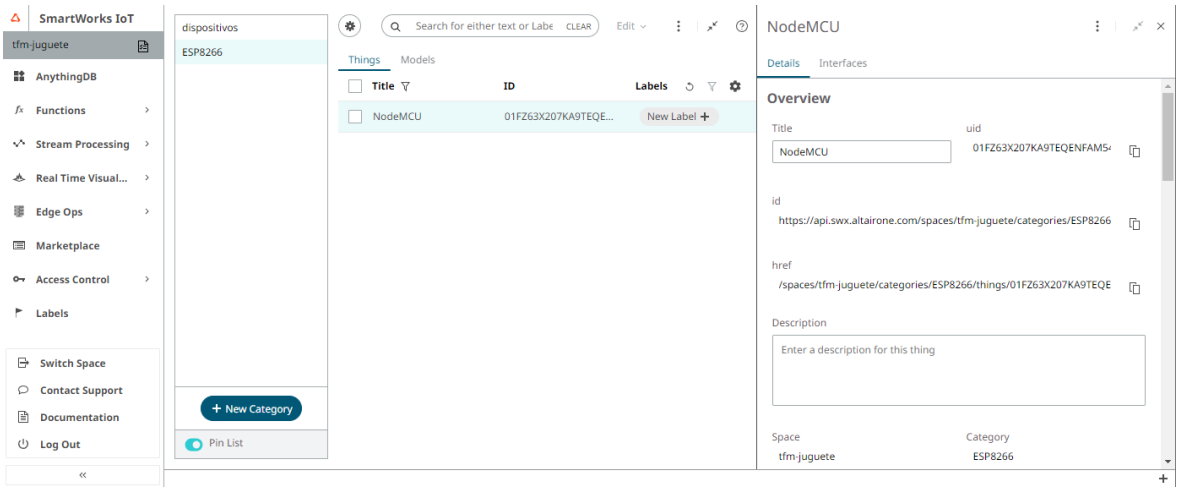


Ilustración 25: Captura de AnyThingDB

En la pestaña de detalles se puede observar el ID del Thing y sus propiedades, en la pestaña de interfaces se observan los parámetros de autenticación por HTTP y MQTT:

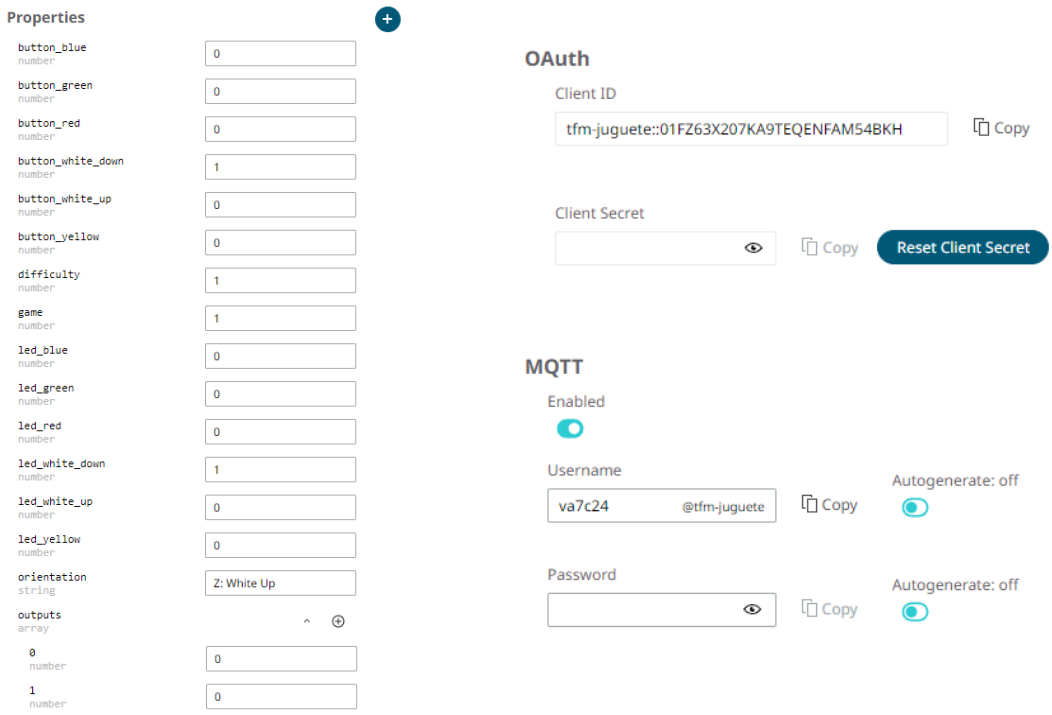


Ilustración 26: Pestañas del Thing (detalles e interfaces)

5.3.2 PROPIEDADES DEL THING

Se han definido las siguientes propiedades para el Thing:

- 6 botones y 6 LEDs: Tipo number indicando su estado, 1 pulsado/encendido y 0 sin presionar/apagado (no booleano para poder indicar modos de iluminación, en el futuro, 2 podría ser parpadeo o definir el color en caso de RGB).
  El nombre de las propiedades es "led_" seguido del color y en el caso de los blancos se distingue la cara inferior de la superior con un "_up" o "_down".
- `game`: Número con el modo de juego del dispositivo (0 desactivado, 1 testeo, 2 juego offline, 3 slave de la plataforma).
- `orientation`: Valor en formato string legible del eje y de la cara superior.
- `outputs`: Array de valores con las instrucciones a activar los LEDs del juguete. Estos valores difieren de las variables con los estados al ser la instrucción indicada por la plataforma desde la que se pueden modificar las luces durante el juego. Una vez se haya encendido/apagado el botón indicado se actualizará el valor del "led_...".
- `print_message`: Mensaje enviado por la plataforma IoT al juguete para mostrar por pantalla en caso de victoria/derrota/error… También se puede utilizar como instrucción para el cambio de estado como se mencionó el en *Clase Game*.
- `state`: Define el estado del juguete como string; puede ser "Waiting" al comienzo del juego para esperar a la instrucción de outputs y comienzo del timer, "Running" mientras el juego esté corriendo – periodo durante el que se ha de comprobar si concuerdan los inputs y outputs objetivo en caso de victoria – y "Timeout" en caso de que se haya agotado el tiempo indicado en la configuración del juego (siguiente property) según el reloj del microcontrolador.
- `timeout`: Valor numérico en milisegundos que define el tiempo que dispone el jugador para introducir la secuencia correcta de inputs antes de entrar en estado de derrota.

5.3.3 HERRAMIENTAS (HISTÓRICO DE MENSAJES)

Pulsando encima de las pestañas de detalles/interfaces en el icono de "Raw History" se puede acceder al listado de mensajes recibidos en el topic correspondiente.

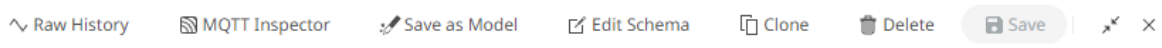


Ilustración 27: Barra de herramientas del Thing

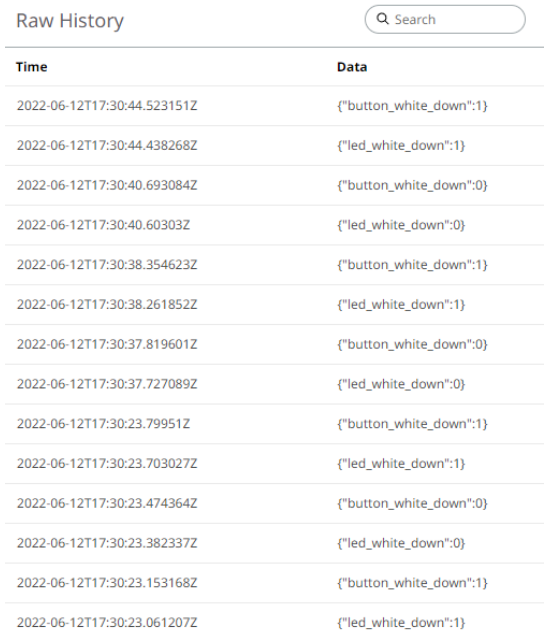


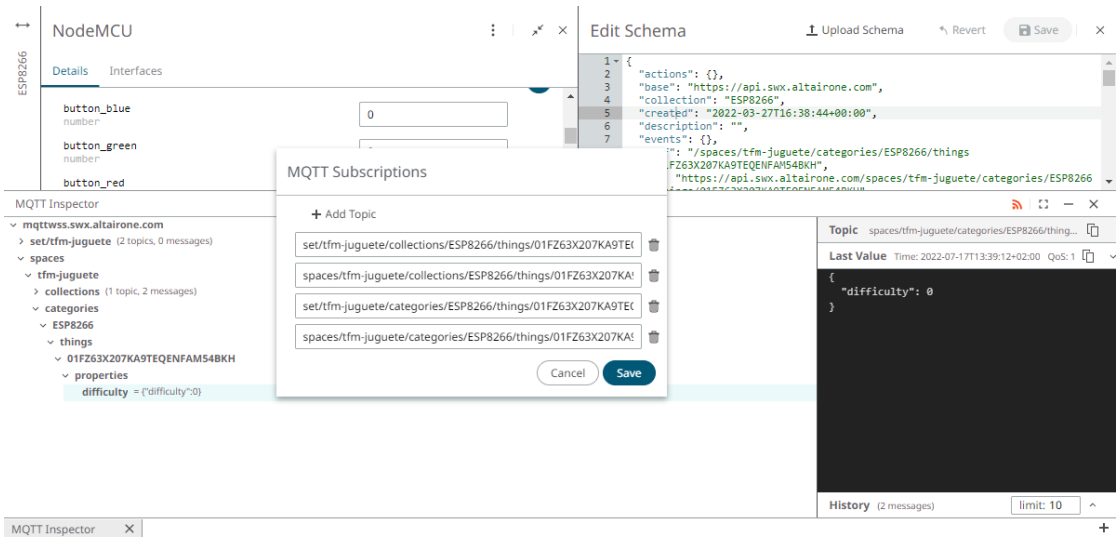
Ilustración 28: Histórico de mensajes

Aunque se actualiza en tiempo real y cuenta con un botón de refrescar, la plataforma también dispone de varias herramientas de inspección de los mensajes MQTT en la misma barra superior (junto con la posibilidad de guardar el Thing como "template", editarlo en formato Schema, clonarlo o eliminarlo).

El Schema es un formato JSON que define todas las propiedades y características del Thing de cara a exportarlo/importarlo con facilidad o editarlo en dicho formato, por ejemplo, para añadir las 12 propiedades de los botones/LEDs a la vez, más rápido que de uno en uno.

5.3.3.1 Inspector MQTT

En la siguiente imagen se muestra el inspector MQTT:



En la ventana de la izquierda se puede observar la estructura del topic y pulsando sobre él se muestran los mensajes publicados en la ventana de la derecha. Además, pulsando en el icono ( ) se listan los topics a los que se encuentra suscrito el inspector.

La estructura de topics es la siguiente:

9. Primero el space seguido de su nombre/identificador único, en este caso “tfm-juguete”.
10. Luego la collection y su nombre, como ya se ha mencionado corresponde a “ESP8266”.
11. Para acceder al Thing se indica su UID indicado en la ventana de interfaces.
12. Por último, se indica el recurso al que se quiera acceder:
    a. Si se quiere acceder a una propiedad se incluye “/properties/nombre_propiedad”.
    b. Si se quiere actualizar una propiedad se especifica “/properties” únicamente y en el mensaje se incluye el nombre de la propiedad.
    c. Y si se quiere publicar al histórico de mensajes se termina en “/data”.

Anteriormente se utilizaba un formato distinto con “set” para actualizar la property y “status” para leer su valor, pero se ha simplificado a un solo topic, y se va a volver a modificar con categorías, pero de momento se puede seguir utilizando de ambas formas.

Al cambiar el valor de una property manualmente en la plataforma también se envía un mensaje MQTT notificándolo. El topic resultante en este caso es: `spaces/tfm-juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/data`.

5.3.3.2 Schema del Thing (semi-compacto)


```
{ "actions": {},
  "base": "https://api.swx.altairone.com",
  "collection": "ESP8266",
  "created": "2022-03-27T16:38:44+00:00",
  "description": "",
  "events": {},
  "href": "/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "id": "https://api.swx.altairone.com/spaces/tfm-
juguete/categories/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH",
  "model": {"name": "","version": 0},
  "modified": "2022-05-29T21:09:28+00:00",
  "properties": {
    "button_blue": {"type": "number"},
    "button_green": {"type": "number"},
    "button_red": {"type": "number"},
    "button_white_down": {"type": "number"},
    "button_white_up": {"type": "number"},
    "button_yellow": {"type": "number"},
    "difficulty": {"type": "number"},
    "game": {"type": "number"},
    "led_blue": {"type": "number"},
    "led_green": {"type": "number"},
    "led_red": {"type": "number"},
    "led_white_down": {"type": "number"},
    "led_white_up": {"type": "number"},
    "led_yellow": {"type": "number"},
    "orientation": {"type": "string"},
    "outputs": {"items": [
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"},
        {"type": "number"}
      ], "type": "array"
    },
    "print_message": {"type": "string"},
    "state": {"type": "string"},
    "timeout": {"type": "number"}
  },
  "space": "tfm-juguete",
  "title": "NodeMCU",
  "uid": "01FZ63X207KA9TEQENFAM54BKH"}
```

5.3.4 FUNCIONES

En la barra de navegación de la izquierda se dispone, después del acceso a la AnythingDB, de las Funciones. Estas están divididas en 2 elementos:

Triggers

Los triggers de eventos son los activadores de las funciones y dictan cuando se las llama. Pueden ser de 2 tipos:

- Cron: Programados o temporizados, se ha definir la periodicidad con el mismo formato que la herramienta cron de Linux ("0 1 * * *" corresponde a una periodicidad diaria a la 01:00; siendo el primer 0 los minutos, el 1 las horas y los asteriscos indican que todos los días, meses y años).
- MQTT: La utilizada para el Proyecto ya que se activará al recibir un mensaje por un topic MQTT específico. En este caso, cada vez que el juguete indica que ha cambiado de estado (property "state"). Se ha de definir el tipo, descripción, bróker, puerto, nombre/contraseña MQTT, nivel de Quality of Service y topic como se muestra en la siguiente ilustración:

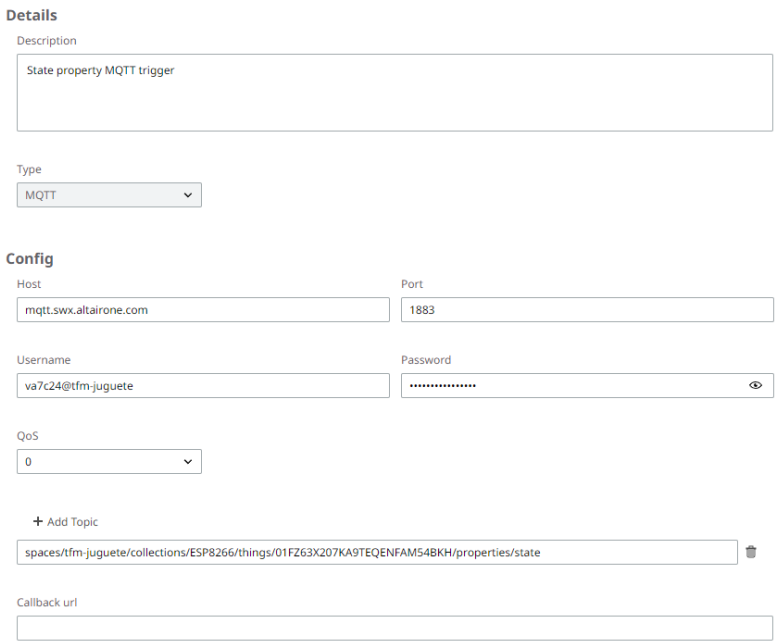


Ilustración 29: Trigger MQTT

5.3.4.1 Workers

Contienen el código de ejecución activado por los triggers que se hayan asignados.

Una vez definido su nombre, descripción y trigger en la primera pestaña, se ha de incluir el código en la segunda (partiendo de un template de Python/Go/FMU). La última pestaña de los workers permitirá testear el código, ejecutándolo e incluyendo el body del request manualmente, además de que es posible acceder a los logs de sus ejecuciones pasadas con su "response code".

En las siguientes imágenes se observa la interfaz de los workers en la plataforma:

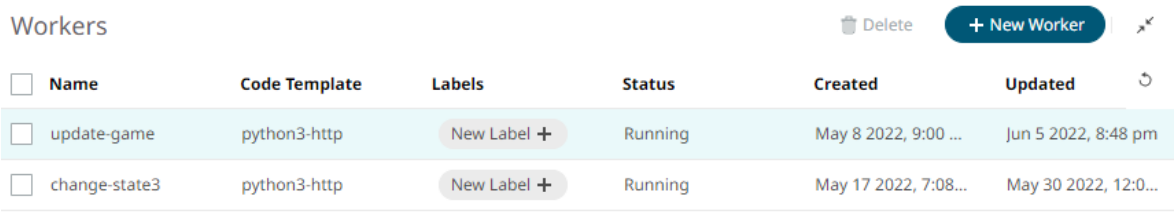


Ilustración 30: Listado de workers

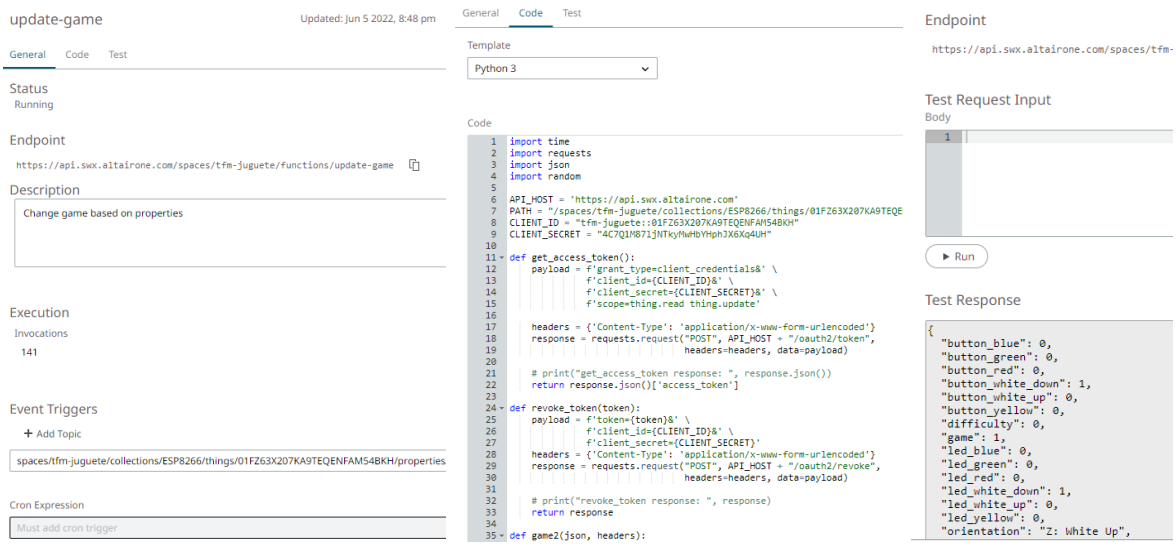


Ilustración 31: Workers (pestañas general, código y test)

Se explicará en detalle en el código desarrollado para estas funciones en el siguiente apartado (*5.3.5 Programación IoT*).

Además de los logs, es posible también acceder nuevamente al inspector MQTT, o a un inspector API para las request HTTP, desde el "Utility Belt" que es la barra inferior.

En la siguiente imagen se realiza una petición HTTP a la URL de las properties del Thing, devolviendo el JSON completo con sus valores. Para hacer dicha petición es necesario incluir en el header la autorización necesaria.

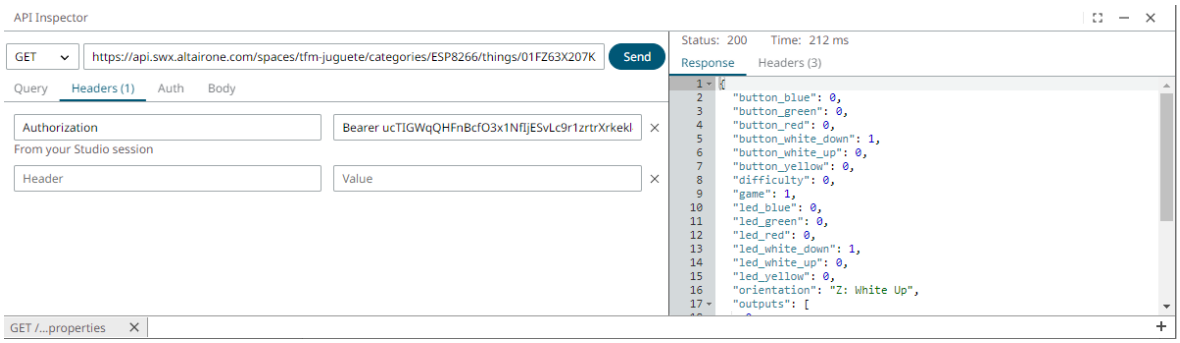


Ilustración 32: Inspector API (request HTTP)

5.3.4.2 Estados

Las funciones y los triggers pueden estar en varios estados al crearlas/modificarlas:

- Pending: Pendiente de creación/realización de la modificación en la plataforma.
- Building: Exclusivo de las funciones, compilando el código, proceso lento.
- Running: Preparada para ejecutarse manual/automáticamente.

Programar directamente en la plataforma de Altair es un proceso muy tedioso debido a la duración de dichos estados, por lo que se recomienda testear el código en Visual Studio y proceder a incluirlo una vez comprobado que funciona.

Para simular el trigger en VSCode es necesario subscribirse al mismo topic que del trigger "…/properties/state" y ejecutar el handle de la función, como callback, a la recepción de un mensaje. Como el worker programado no requiere de body en el mensaje, pasar el código de Visual Studio a la plataforma de Altair es copiar y pegar. En detalle en el siguiente apartado.

5.3.5 PROGRAMACIÓN IOT

La programación del worker se hará en Python. El código incluido en el worker y el ejecutable en Visual Studio se adjuntan en el *ANEXO II: Código*, en el *7.1 Código de la plataforma IoT*. Como ya se ha mencionado y se detalla en dicho apartado, el código incluye 2 partes:

5. Código ejecutable en la plataforma por el worker llamado "update_game". El cuál sigue el siguiente proceso:
   - Se importan 4 librerías de Python:
     - time: Para la conversión y el manejo de unidades de tiempo.
     - requests: Funciones necesarias para la comunicación por HTTP.
     - json: Requerida para el decoding de los payloads recibidos/enviados.
   - random: Función para la generación de números pseudoaleatorios.
   - Definición de 4 constantes (dominio HTTP de Altair, dirección URL a acceder y usuario y contraseña HTTP para el acceso.
   - Funciones `get_access_token()` y `revoke_token()`: Han sido aportadas por Altair y ejecutan una request HTTP tipo POST para obtener un token de autorización temporal de lectura y modificación de las propiedades de la Thing.
     La primera devuelve el token, la segunda lo recibe como parámetro y lo anula.
   - Función `game2()`: Simula el mismo proceso que la función game2() de la *Clase Game* de la librería *game.h*, pero enviándolo las instrucciones necesarias al juguete.

     En función del estado del juguete ejecuta las siguientes instrucciones:
     - Si el juego se está ejecutando (estado "Running") se ha de revisar que los estados de los botones sean iguales a los outputs objetivo, en caso de serlo se envía un mensaje de victoria. Si el estado de los inputs cambia, pero no es el correcto se envía un mensaje para mostrar por pantalla de seguir intentándolo.

- Si el juguete detecta que ha transcurrido el tiempo límite de juego y entra en modo "Timeout" se ha de enviar un mensaje de derrota.

  Si el juego acaba de ser seleccionado y el juguete está esperando instrucciones ("Waiting"), se han de indicar los LEDs que se han de encender, que serán los botones objetivo a pulsar por el jugador. Al igual que en la función `game2()`, es necesario crear un array de 6 elementos con ceros un número de unos igual a la dificultad seleccionada.

  En Python esto es muy sencillo, gracias a la función `random.sample()`, que devolverá n elementos de un array de 6 (`range(0,6)=[0,1,2,3,4,5]`, `sample(range, 2)=[2,4]`).

  Una vez seleccionados n LEDs a encender se utiliza "list comprehension" de Python, que permite crear una lista haciendo un bucle de 0 a 5 y si la posición está en el sample anterior se pone a 1 y sino a 0.

  Por último, se envía un mensaje indicando los outputs a encender y dando comienzo al programa.

- `handle()`: Función que se ejecuta en caso de llamar al worker con el trigger.

  - Primero obtiene el token para las peticiones HTTP y hace una request HTTP GET a las propiedades del Thing de la plataforma, está información se almacena en una variable json.

    Se ha de tener la cuenta que al estar suscrito al topic "state" ya se dispone del valor del estado del juguete, pero no del resto de las propiedades, es por ello que se realiza esta petición.

  - Si el juguete está en modo de juego 3 (slave) - en el que sigue las instrucciones de la plataforma – entonces se ejecuta la función `game2()`, en caso contrario se revoca el token y se devuelve el JSON con las propiedades.

6. La segunda parte simula la suscripción del trigger al topic "state" y la activación del handle del worker mencionado previamente.

   - Para ello, se importará una librería de Python llamada paho.mqtt.client.
   - Se definen las constantes MQTT (dominio, usuario, contraseña y topic).
   - `on_connect()`: Indica cuando se ha realizado la conexión en el terminal y se suscribe al topic indicado.
   - `on_message()`: Muestra el mensaje recibido y llama al handle del worker.
   - `handle1()`: Función principal. Crea el cliente, conecta las funciones anteriores a la conexión/recepción de mensajes, define el usuario y contraseña y se conecta al servidor de Altair.
   - Como el código se ejecuta como "`__main__`" por defecto al correr el archivo en el terminal, se ha de incluir un `if` para asegurar que si se está ejecutando el archivo (no importando) se llame al `handle1()` sin body (req=0).

El worker hace uso únicamente de la comunicación HTTP, aunque se activa por medio de un mensaje MQTT. Aun así, al enviar mensajes a la plataforma por HTTP para modificar las propiedades se genera un mensaje MQTT indicando su modificación. Este es el mensaje recibido por el juguete al estar suscrito a los topics de ciertas propiedades.

Este programa, como se puede observar, es muy fácil de escalar y ampliar con más juegos, debido a la simplicidad de control del juguete por medio de mensajes. Al modificar la programación de la plataforma es posible modificar el código de todos los juguetes conectados a la red IoT sin necesidad de conectarlos al ordenador y subir el código manualmente. Se entrará más en detalle acerca de su potencial en el siguiente capítulo.

5.3.6 VISUALIZACIÓN DE LOS DATOS

Aunque es posible visualizar los valores de las propiedades y su histórico en la propia Thing, es preferible crear un dashboard.

5.3.6.1 Apps

Para ello, primero es necesario crear una App (apartado de Stream Processing). Las aplicaciones permiten conectar las Data Sources con otras herramientas. Para el Proyecto se han creado 2 aplicaciones:

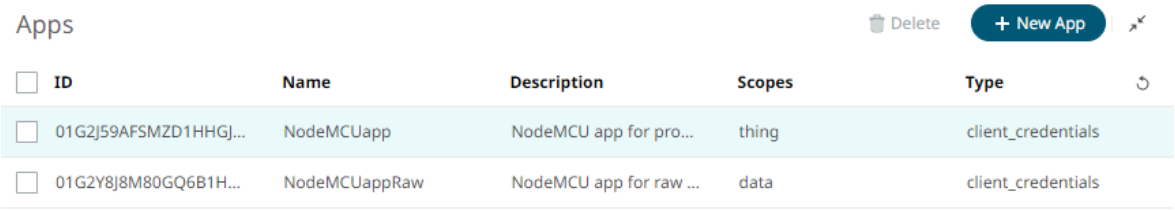


Ilustración 33: Apps IoT

- NodeMCUpropertiesDT: Es de scope "thing" y tiene acceso a las propiedades de la Thing asociada.
- NodeMCUMCUrawDT: Es de scope "data", permite acceder al historial completo de mensajes.

Es posible realizar una sola aplicación con acceso a ambas Data Tables, pero es preferible mantenerlas separadas. Las aplicaciones también cuentan con su propio Client ID y Client Secret de acceso como las Thing (en caso de ser tipo "client credentials").

En las siguientes capturas se muestran las pestañas de detalles de configuración y políticas de acceso, ya que es posible definir diferentes accesos (creación, lectura, modificación, borrado) de las Things. El resource indicado es "#" para simbolizar el acceso a todos:

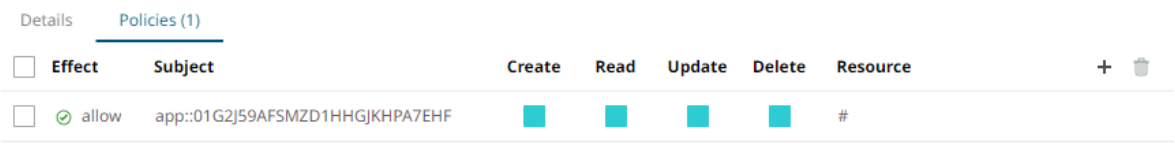


Ilustración 34: Apps IoT - Pestaña de policies

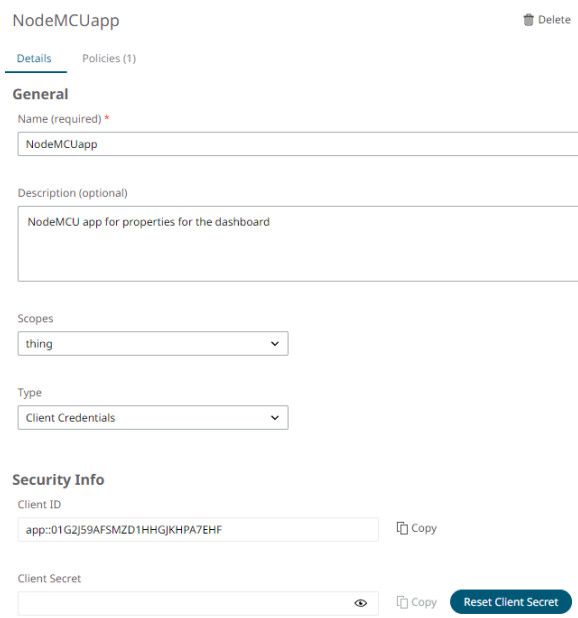


Ilustración 35: Apps IoT - Pestaña de detalles

5.3.6.2 Workbooks

Una vez creadas las Apps es posible conectarlas a un Workbook (en el apartado de Real Time Visualization) que contiene los dashboards. Es necesario crear la Data Table conectando el workbook con la App (definir credenciales, scope, URL…) y definir las columnas (o generarlas automáticamente). En la siguiente ilustración se muestra la configuración del primer workbook:

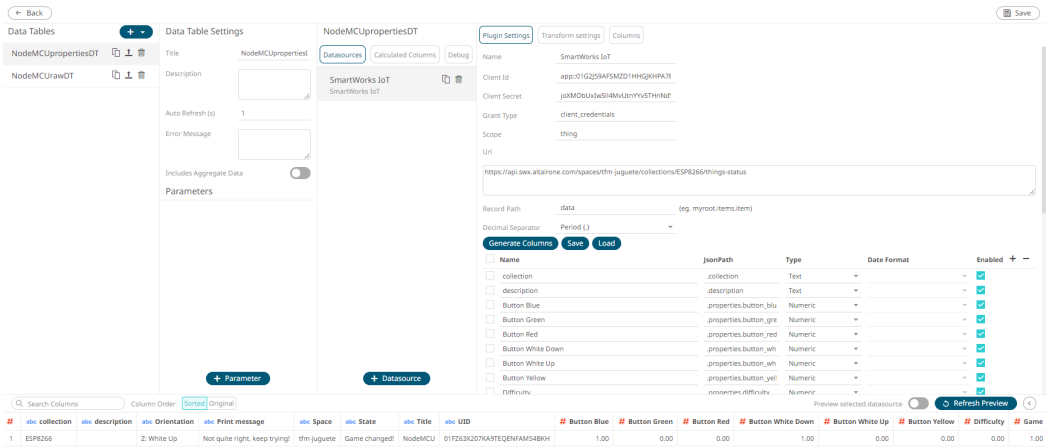


Ilustración 36: Workbook settings

En el caso del histórico, es necesario activar la opción de "Tansform to enable time series analysis" en "Transform Settings" para poder visualizar el tiempo de recepción en formato fecha y utilizar las gráficas correspondientes.

Con cada una de las aplicaciones se ha diseñado un dashboard. El primero muestra de forma visual las propiedades y estado del juguete (inputs, outputs y variables):

Ilustración 37: Dashboard

El segundo simplemente es una tabla con la lista de mensajes y una gráfica con los estados 0-1 en el tiempo, pero debido al volumen de registros (mayor a 100000) y otros fallos (que se han notificado a Panopticon que es el proveedor de la herramienta gráfica de dashboard de Altair) no muestra correctamente los resultados.

A futuro, este dashboard se puede ampliar, no solo con el histórico de datos, sino incluyendo más información de los sensores que ya se recoge en el dispositivo (como las aceleraciones en cada eje, o los tiempos que tarda el jugador en pasar el juego) y brindando la posibilidad de mostrar varios dispositivos simultáneamente.

En caso de poder conectar una red de dispositivos la información obtenida del aprendizaje de todos los niños jugando se puede utilizar para mejorar el sistema y añadir juegos o curvas de dificultad personalizadas. El análisis de estos puntos se realiza en el siguiente capítulo.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se analizará el Proyecto realizado, su diseño, implementación, descubrimientos, puntos favorables, áreas de mejora y pruebas de concepto logradas.

6.1 ANÁLISIS DEL SISTEMA

El resultado del Proyecto se puede estructurar en los siguientes puntos:

- Construcción de un prototipo completamente funcional, con LEDs, botones, acelerómetro y altavoz. El prototipo tiene margen de mejora en ergonomía (tamaño), pero es económico y con gran durabilidad, además de haber ayudado a sentar a futuro los métodos para construir otros juguetes iguales, parecidos o mejores.
- Programación estructurada y documentada de un código relativamente ligero, comprensible y sin necesidad de mejora debido a su funcionalidad conectada. La legibilidad del código principal ha sido lograda gracias a la organización en librerías y clases de control individualizado de cada uno de los componentes.
- Conexión IoT del Proyecto: Tanto la plataforma IoT de Altair como la programación desarrollada en la misma permiten una escalabilidad y versatilidad increíble. Muchas de las funcionalidades de la plataforma no se han llegado a rozar y, aun así, se ha demostrado la posibilidad del desarrollo completo de un juguete conectado y el potencial que dispone como fuente de datos.

En el siguiente apartado se entrará en detalle en los puntos favorables y apartados a mejorar que se han descubierto con la realización del Proyecto. Recapitulando, se pude observar que el prototipo desarrollado es completamente funcional, tiene capacidad de juego tanto offline como online, pudiendo jugar con una dificultad configurable, monitorizando los diferentes eventos (botones, LEDs, posición, estado, victoria/derrota…) y pudiendo jugar también online ejecutando las instrucciones indicadas por la plataforma y, por tanto, permitiendo a futuro continuar su desarrollo de forma ilimitada sin volverlo a conectar o reprogramar.

6.2 DISEÑO E IMPLEMENTACIÓN

Siguiendo la estructura del *Capítulo 5. Sistema/Modelo Desarrollado* y del anterior apartado el análisis final del Proyecto se divide en 3 subapartados, en función de la construcción física del prototipo, su programación y, finalmente, el desarrollo en la plataforma IoT de Altair. Asimismo, se incluye un subapartado adicional mencionando la documentación del presente documento.

6.2.1 DISEÑO Y CONSTRUCCIÓN DEL PROTOTIPO

Aunque ya se ha comentado durante el desarrollo de esta memoria, en este apartado se va a detallar el análisis del prototipo diseñado y construido.

El prototipo es una caja de un volumen elevado (12 cm de arista), además de contar con botones Arcade que sobresalen debido a la tuerca exterior de plástico que utilizan. Esto reduce significativamente la ergonomía del juguete, en especial para manos pequeñas como las de un bebé o niño pequeño.

Además, como se puede observar en las *Ilustración 23*, los conectores de los botones están en un espacio muy reducido. Por suerte, gracias al uso de protectores, es poco probable que hagan una conexión accidental, pero tener conectores grandes de metal sin recubrir conectados a diferentes tensiones (GND y VCC) no es lo ideal. A futuro es necesario sin duda hacer uso de botones de tamaño más reducido, dado que son el componente que más volumen ocupa y, por lo tanto, permitiría reducir significativamente el tamaño de la caja.

No obstante, es complicado encontrar botones de un ancho suficiente (la mayoría son en miniatura) y reducir el tamaño de los botones también supone mayor dificultad para presionarlos. Y, por ello, no deberían ser seleccionados unos de tamaño muy reducido en caso de tener la intención de ser utilizados por un bebé.

Con todo ello la construcción es muy sólida y son muy improbables las conexiones accidentales/cortocircuitos entre cables. Además, al ser un diseño cerrado, no suponen un riesgo para el jugador.

La madera es blanda, pero de un grosor suficiente para aportar resistencia. El trozo que se desprendió al agujerearla con la broca era pequeño y no se distingue posteriormente al usar el pegamento termofusible.

La conexión de la PCB con headers como los que se muestran en la Ilustración 21 permite la desconexión de los componentes y testeo en la protoboard. Los botones también pueden desconectar la bombilla del botón (*Ilustración 5: Botón LED desmontado*), manteniendo el botón en la caja y sacando la bombilla con los cables. El acelerómetro se puede desconectar y el microcontrolador está encajado por los cables, pero sin fijar. En resumen, es posible desmontar por completo el dispositivo y utilizar los componentes en otra caja o en la protoboard, permitiendo versatilidad a la hora de diseñarlo, construirlo y testearlo.

Como consecuencia, se ha requerido una cantidad de tiempo elevada en el soldado de los cables a los headers macho-macho y de los headers macho-hembra a la PCB. En el futuro se ha de evaluar la posibilidad de utilizar cable de conexión completa a los conectores de los botones que no requieran el soldado. En el caso de los botones Arcade existen soluciones con tierra común que simplifican el cableado, como los de la ilustración:

El uso de cable en vez de batería no es un punto a favor ni en contra del prototipo, al ser una decisión de diseño: es un prototipo, no un diseño final, y la intención es poderlo tener conectado al ordenador para debugear el código y programar sin tener que abrir la caja. Se ha probado a conectarlo a una pequeña batería portátil USB recogiendo el cable dentro de la caja y es completamente viable su uso. Por supuesto, el diseño final ha de contar con una batería recargable o hacer uso de pilas, ya que la programación se cargaría una única vez.

El prototipo físico es la parte del Proyecto con mayor margen de mejora, ya que se diseñó, como tal, como un prototipo para testear la posibilidad de diseño de un juguete conectado. De haber tenido más tiempo se habría realizado un segundo diseño, pero la construcción ya había requerido más horas y meses que la planificación original. El objetivo del diseño ya se había logrado: demostrando que es posible desarrollar un juguete con una versatilidad

superior a las opciones del mercado, gracias a sus sensores, botones y luces, y, sobre todo, control por microcontrolador programable conectado a una plataforma IoT.

Un último detalle en contra del conexionado es la imposibilidad de usar el altavoz en combinación con la opción de debugeo/cargando el código conectado al ordenador (debido a requerir tener el pin HIGH durante el reseteo). Es cierto que la capacidad de mostrar mensajes por pantalla es un añadido muy útil no solo para su programación y testeo sino como guía al jugador. Aun así, el juguete ha sido pensado para utilizarse desconectado en su versión final (alimentado por una batería) y es un problema solucionable a futuro con un condensador o haciendo un rediseño del conexionado de pines.

6.2.2 PROGRAMACIÓN INTEGRADA DEL MICROCONTROLADOR

Respecto a la elección del **NodeMCU y Arduino**:

En un principio no se sabía si se iba a usar un microcontrolador o microprocesador con sistema operativo integrado. Se optó por el microcontrolador con la capacidad de conexión por Wifi con la posible promesa de sustituir parte de la programación integrada por programación en la nube. Tampoco era necesaria una capacidad de computación elevada, pero sí que hubiese simplificado la programación el uso de un microprocesador. Se podría haber programado todo en Python con el mismo código de comunicación que se utiliza en la plataforma IoT, pero esto implica un consumo superior de batería innecesario.

Es cierto que el NodeMCU tardaba más de un minuto en compilar el código en Visual, cargarlo por USB, reiniciarse y, sobre todo, conectarse por Wifi y MQTT. Un minuto es un tiempo elevado para la programación, pero una vez terminada no es necesario volver a conectar el dispositivo y el resultado final es favorable en comparación con el consumo de una pequeña Raspberry o equivalente, incluso con una versión ligera de Linux (sin interfaz gráfica). Por ello, se concluye que la elección del microcontrolador ha sido la más adecuada a costa de un tiempo superior – pero tampoco especialmente significativo – dedicado a la programación. Aun así, sí sería recomendable usar la versión del procesador ESP32 actualizada en vez de la ESP8266 que además de mayor potencia, tiene un diseño un poco más compacto.

Por otro lado, a pesar de los distintos frameworks y SDKs que ya se han mencionado (compatibles con el NodeMCU), desde el principio estaba claro que se haría uso de Arduino, por su documentación y uso generalizado en estos proyectos, pero usando Visual Studio Code en vez de con Arduino IDE. Esta decisión ha requerido cierto periodo inicial de adaptación a Platform IO, pero, una vez acostumbrado, se reduce el tiempo significativamente de programación, organización… Visual Studio como editor de código es una opción muy superior a Arduino IDE y su compatibilidad con GitHub es un punto a favor, pero, además, PlatformIO ha facilitado notablemente la organización de los ficheros y librerías.

Respecto a la **programación del juguete**:

Ya se ha mencionado al final del *5.2 Programación*, pero es necesario reiterar que la mejora de la programación del juguete es innecesaria e irrelevante, dado que el código es lo suficientemente versátil como para adaptarse a una variedad de juegos en el modo "slave", en el cual sigue las instrucciones de la plataforma.

Lo que se ha de ampliar y desarrollar es la programación en la plataforma IoT, con más juegos, desarrollando herramientas de visualización de los datos y curvas de dificultad. Pero la recepción de instrucciones por el juguete ya está finalizada y solo se debería de modificar en el futuro para adaptarla a diferentes componentes (por ejemplo: botones RGB o añadir un motor de vibración); para incluir funcionalidad necesaria para un juego específico; o para incluir instrucciones directas permitan simplificar la programación en la plataforma. Por ejemplo: la opción de hace que un LED parpadee, esta opción ya es posible controlando los outputs desde la plataforma, pero requeriría cierta sincronización en la comunicación y es posible que no se lograse una frecuencia tan perfecta como desde el microcontrolador.

Es cierto que la existencia de un solo juego completo descargado en el dispositivo y disponible sin conexión es un tanto limitada, pero durante el desarrollo se programaron otros 2 juegos básicos y no se han incluido en la versión final por 3 motivos:

- Estaba en proceso su nivel de testeo y faltaba la conexión con la plataforma.
- Por lo tanto, no disponían de curva de dificultad u opciones de configuración y monitorización.
- Y, por último, una vez se planteó el concepto de control por la plataforma, la programación adicional de más juegos se pausó, con la promesa a futuro de poder desarrollarlos en la plataforma, donde es mucho más fácil y rápido.

Al fin y al cabo, el objetivo del Proyecto no era desarrollar una gran cantidad de juegos sino lograr una plataforma escalable y versátil, que no solo monitorice los eventos del jugador, sino que también disponga de la posibilidad de ampliar la dificultad y variedad de niveles. En el siguiente apartado se analizará más en detalle este punto.

6.2.3 DESARROLLO EN LA PLATAFORMA IOT

Así como el prototipo físico del Proyecto dispone de un gran margen de mejora, mientras que la programación integrada no requiere de ninguna modificación adicional, el desarrollo en la plataforma lo que requiere a futuro no es su mejora, sino su ampliación.

En el Proyecto se ha testeado y documentado el proceso de configuración y conexión de Altair SmartWorks IoT. Se ha demostrado el potencial de control de un juguete por medio de instrucciones programadas por workers, activados por MQTT y que acceden a la API por HTTP. Y se ha comprobado que, por medio de estos workers, es posible mostrar un juego y monitorizar los eventos en tiempo real.

El código desarrollado ya tiene configurada toda la comunicación y selección de juegos, solo requiere la implementación de más juegos. El juego `game2()` es capaz con unas pocas líneas de código en Python de mostrar un juego sencillo y configurable. De la misma forma, es posible desarrollar muchos más juegos haciendo uso del control por mensajes MQTT del juguete. Ejemplos de juegos que es posible desarrollar en la plataforma para el juguete son:

- Juegos de memoria: Encender los botones los botones en un orden durante un tiempo y luego esperar a que el jugador los pulse en el mismo orden. Una variante de este juego se ha probado, pero sin su funcionalidad online de monitorización y configuración. Es posible aumentar la dificultad de este juego simplemente aumentando el número de botones en la secuencia o reduciendo el tiempo para pulsarlos.
- Juegos de velocidad de reacción: Se enciende un botón y hay que pulsarlo para que se apague y, si parpadea, hay que posicionarlo mirando hacia arriba. Obviamente, sería posible configurar la velocidad a la que se encienden los botones y el tiempo límite para pulsarlos.
- Juegos de sonidos: Igual que los juegos de memoria, pero cada botón tiene un sonido: primero se identifican los sonidos con los botones y luego se reproducen, y hay que pulsar el botón correspondiente.

Estos son algunos ejemplos que se han considerado, pero existen muchísimos otros posibles. El potencial es inmenso, incluso se utilizando solo 2 tipos de sensores (botones y acelerómetro) y 2 salidas (luces y sonidos) y con LEDs RGB u otros elementos sencillos la variedad es aún mayor.

Sabiendo que juegos de este tipo son posibles por medio del prototipo y la plataforma – una vez desarrollados – el siguiente paso sería conectar más dispositivos, iguales o parecidos (clonando el Thing o modificándolo). Y probar así las posibilidades que brindaría disponer de una base de datos monitorizando una serie de juguetes y jugadores.

Con acceso en la plataforma de Altair a una mayor cantidad de muestras de eventos, sería posible el análisis de los patrones de aprendizaje. Y haciendo uso de otras herramientas más avanzadas de SmartWorks es posible con Machine Learning realizar curvas de dificultad óptimas generales o adaptadas a cada jugador. Una vez abierto al mundo de Big Data, el análisis masivo aporta un nuevo mundo de posibilidades, que se encuentra fuera del alcance del Proyecto, pero que ya se empiezan a vislumbrar con más de 100.000 registros de eventos recogidos durante el periodo del desarrollo del mismo.

6.2.4 DOCUMENTACIÓN DEL PROYECTO

Por último, se ha de mencionar, aunque sea un tanto redundante, la labor de documentación recogida, no solo en el presente documento del Proyecto, sino también del código de las librerías y componentes. Se ha recopilado un proceso de elaboración de un dispositivo conectado, se han documentado las funciones y el código necesario, además de la configuración de la plataforma. También se enumeran las posibilidades del Proyecto con tecnologías emergentes con una plataforma de dispositivos escalable.

En resumen, utilizando esta memoria como guía de desarrollo del Proyecto es posible eliminar el tiempo de investigación y preparación (mencionado en el *3.3*) para replicar el Proyecto. Reduciendo así significativamente el tiempo necesario para ampliar la red de dispositivos conectados a la plataforma IoT.

Capítulo 7. CONCLUSIONES

Durante el proyecto han surgido dificultades en diferentes aspectos:

- A nivel de construcción: Especialmente debido a la elección de cableado, que requiere un mayor tiempo de soldado, pero permite el desmontado completo de los componentes y cables. Algunos de los problemas se hubiesen evitado con mayor tiempo de investigación y planificación del cableado, pero se han podido resolver adecuadamente (como el uso del expansor de puertos debido al número de pines).
- A nivel de programación: Siempre surgen problemas en el código que son difíciles de entender en la programación integrada (debido en parte a la imposibilidad de debugear estos dispositivos), aunque se han podido resolver gracias a la monitorización en la plataforma. Además, se ha eliminado esta problemática a la hora de programar al realizar el resto de la programación en la plataforma.
- A nivel de desarrollo en la plataforma de Altair: Al carecer de experiencia con la misma han surgido dudas y problemas a pesar de su documentación. Existen casuísticas que no se han planteado previamente, pero han testeado que funcionan correctamente con el Proyecto. La plataforma está en constante cambio y se han tenido que hacer modificaciones durante el desarrollo para adaptarse (como el cambio 2 veces de los topics). Por otro lado, la programación se realizaba al principio directamente en la plataforma, lo que complicó y ralentizó el proceso hasta que se fue capaz de simular los workers/triggers en VSCode. Existen problemas por solucionar (en especial con el volumen de datos, como en el dashboard de histórico), pero gracias a la colaboración con Altair se han podido resolver rápidamente.

Al final del desarrollo del Proyecto se han ido resolviendo los problemas que surgían e impedían el progreso o se han encontrado soluciones alternativas. Estos problemas han planteado nuevas cuestiones y han re-enfocado el desarrollo. Y a su vez, con los cambios de enfoque han surgido nuevos problemas a resolver (como el control del dispositivo en).

Como **resumen** del apartado anterior de análisis del diseño realizado y su implementación, y como conclusiones del documento de desarrollo del Proyecto, se puede considerar que se han cumplido los objetivos iniciales del Proyecto (recabados en el Anexo B del TFM e incluidos en el *3.1 Objetivos*) de: construcción, validación, monitorización y conexión IoT.

El prototipo construido tiene el margen de mejora que ha de tener como diseño inicial y prueba de concepto. Y ha servido para validar, no solo la idea original de hacer un juguete monitorizando los eventos, sino también de controlar por completo el dispositivo y realizar la programación en la nube.

Y finalmente, la conexión a la plataforma IoT de Altair SmartWorks ha sido posible por medio de MQTT, junto con el uso de peticiones HTTP a la propia API por parte de los workers que permiten modificar parámetros de juego y mandar instrucciones.

Además de los 4 objetivos iniciales, se puede decir que se han logrado los que han motivado el desarrollo del Proyecto y entre lo que se encuentran los explicados en el *ANEXO I: SDG*.

El Proyecto brinda la posibilidad de desarrollo de una plataforma de entretenimiento, educación y estudio del aprendizaje. Es cierto que se existe una labor muy amplia a futuro de cara a lograrlo, pero el Proyecto ha verificado su posibilidad de implementación y demuestra que se trata de una tarea factible.

REFERENCIAS

[1] JoyJam Electronic Music Cube
https://www.youtube.com/watch?v=ZWeakDdXFnA

[2] Electronic Memory Game Toys
https://www.ubuy.vn/en/product/1CEBHVF0W-electronic-memory-game-toys-brain-teaser-puzzles-sequence-board-game-and-memory-game-test-educational-toy-for-kids-3-and-up-toddler-teens-aa

[3] Six Grid Electronic Drum Memory Game
https://www.amazon.es/Electronic-Childrens-Intelligent-Challenge-Education/dp/B082Y4JZKB

[4] Juegos de letras, números y formas
https://www.amazon.es/Electronic-Childrens-Intelligent-Challenge-Education/dp/B082Y4JZKB

[5] Node MCU website
https://www.nodemcu.com/index_en.html

[6] NodeMCU documentation
https://nodemcu.readthedocs.io/en/release

[7] Botones LED (Pulsadores Arcade Iluminado)
https://www.arcadexpress.com/es/botones-pulsadores-arcade/20-984-boton-de-servicio-iluminado.html

[8] Expansor de puertos I2C (MCP23017)
https://www.amazon.com/-/es/MCP23017-interfaz-MCP23S17-Bidireccional-Interface/dp/B07Y5LD4XK

[9] Acelerómetro/giroscopio (MPU6050)
https://www.conectrolinformatica.com/arduino-modulos/3600-modulo-mpu6050-gy-521-acelerometro-giroscopio-.html

[10] Altavoz/zumbador (Mini Metal Speaker 8Ω)
https://www.adafruit.com/product/1890

[11] Estructura contenedora de madera (Artemio VIBB19)
https://www.amazon.es/Artemio-VIBB19-Hubinont-Caja-cubos-madera/dp/B0018BMIGY/

[12] Visual Studio Code
https://code.visualstudio.com/

[13] Platform IO
https://platformio.org/

[14] GitHub
https://github.com/

[15] Altair SmartWorks
https://www.altair.com/smartworks

[16] Easy EDA
https://easyeda.com/

[17] Toy Marktel Size (Fortune Business Inights)
https://www.fortunebusinessinsights.com/toys-market-104699

[19] Librería "pitches.h"
https://gist.github.com/mikeputnam/2820675

ANEXO I: SDG

Los objetivos de desarrollo sostenible (ODS o SDG) que se encuentran en línea con el alcance del Proyecto, son 3, pero siendo 2 parte de los objetivos originales como motivación del mismo.

El primero y menos relevante debido, como se ha mencionado a su falta de intencionalidad inicialmente, es el de "Trabajo y crecimiento económico". Debido a haber desarrollado una idea y diseño muy baratos (con componentes que suman en total menos de 100 euros) y con un elevado potencial comercial debido al sector enfocado: los juguetes infantiles son uno de los productos con mayor rentabilidad y ventas debido a su necesidad y nicho de mercado estable, estimándose en 130 billones de euros a nivel global en 2020 [17]. Este ODS no era un objetivo original del Proyecto, pero sí que se ha mencionado dada su posible conexión.

El segundo ODS seleccionado es el de la **Educación**. Se ha descrito desde el principio el enfoque educativo de cara al desarrollo cognitivo, pensamiento lógico y creatividad de los bebes. Además, el objetivo está implícito en uno de los objetivos del Proyecto, que es la dificultad programada y escalable. El Proyecto realizado tiene un gran potencial educativo gracias a estos aspectos, permitiendo monitorizar el proceso de aprendizaje, adaptar la dificultad y tiempo permitido, y, en el futuro, realizar curvas de dificultad óptimas para el desarrollo del niño.

La plataforma tiene múltiples funcionalidades de Machine Learning y Big Data que no se han explorado; pero, teniendo una red de juguetes conectados, es posible un análisis exhaustivo de los patrones de aprendizaje, descubriendo deficiencias y puntos fuertes en el desarrollo cognitivo y sentando las bases para el futuro de una plataforma inteligente y en constante cambio (actualizable), conectada a infinidad de juguetes educativos y entretenidos en todo el mundo.

Y, finalmente, el último y posiblemente con mayor potencial, el de **Innovación**. El dispositivo hace uso de una plataforma IoT comunicándose por MQTT, y la plataforma a su vez hace llamadas HTTP para acceder al estado del dispositivo y suministrar órdenes.

De por sí ambos protocolos ya son muy comunes en proyectos de electrónica conectada o en las telecomunicaciones de todos los dispositivos que utilizamos habitualmente. Y la monitorización tampoco es un aspecto especialmente innovador en estos ámbitos, incluso a gran escala. Pero nunca se ha planteado para un dispositivo de entretenimiento y aprendizaje, principalmente por el enfoque que ya se ha explicado de estos juguetes de probar, solucionar y desechar una vez que el bebé ha crecido o se ha cansado. Poder monitorizar estos dispositivos brinda una cantidad de información muy elevada (>100.000 eventos por juguete en un periodo pequeño), y la labor de análisis posterior – aunque compleja – supone un estudio exhaustivo de la mente humana y su desarrollo en edades tempranas.

Y, además de las posibilidades de estudio de la monitorización, también existe un aspecto muy poco común o único de este Proyecto, que es el control centralizado de los juguetes en una plataforma en la nube. La programación y computación en la nube no tiene el objetivo de reducir la carga computacional del dispositivo (aunque también es un aspecto relevante), sino la centralización de la programación permitiendo configurar juegos y niveles actualizando una función en vez de cargar el código en cada dispositivo.

La programación en Python – simulando las funciones en Visual Studio y cargando el código en la plataforma – facilita enormemente la labor de elaboración y testeo de código. Y posibilita incluso la realización de pruebas A/B para el testeo de niveles o diferentes configuraciones de juego. Además, el código de comunicación ya se ha desarrollado y solo es necesario añadir funciones nuevas con nuevos niveles. El concepto de computación en la plataforma se ha testeado con un ejemplo sencillo (simulando el equivalente offline) y validando su posibilidad de uso como sistema de control centralizado.

En **resumen**, el enfoque educativo del Proyecto siempre ha sido primordial y, durante el desarrollo, han surgido posibilidades de innovación que se han puesto a prueba y validado.

ANEXO II: CÓDIGO

LIBRERÍAS

Las librerías externas en Arduino pueden contar con dos documentos como en C++, el archivo ".h" enumera las funciones y clases y es obligatorio, el archivo ".cpp" incluye el código fuente de las funciones y clases. Para reducir el número de ficheros es posible definir directamente el código en el archivo ".h". A continuación, se adjunta el código de las librerías privadas que se han utilizado para el Proyecto. El resto de librerías son accesibles a través de la documentación de Arudino o de PlatformIO, por lo que no se incluyen.

pitches.h

Listado de frecuencias de las notas, accesible desde GitHub [19].

```
#ifndef PITCHES_H
#define PITCHES_H

// Frecuncies for notes in Hz
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
...
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978


#endif
```

spkControl.h

Clase con las funciones de control del altavoz para la reproducción de melodías.

```
#ifndef SPKCONTROL_H
#define SPKCONTROL_H

#include "pitches.h"

class spkControl {
  private:
    int spkPin;      // Speaker Pin

    int win_melody[8] = { NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0,
NOTE_B3, NOTE_C4 };
    int win_noteDurations[8] = { 4, 8, 8, 4, 4, 4, 4, 4 }; // 4 = quarter note, 8
= eighth note, etc.
    int loose_melody[1] = { NOTE_C1 };
    int loose_noteDurations[1] = { 4 };
    int temp_melody[1] = { NOTE_A1 };
    int temp_noteDurations[1] = { 8 };

    void playMelody(int melody[], int noteDurations[]) {
      // iterate over the notes of the melody:
      for (int thisNote = 0; thisNote < 8; thisNote++) {
        // to calculate the note duration, take one second divided by the note
type.
        int noteDuration = 1000 / noteDurations[thisNote];
        tone(spkPin, melody[thisNote], noteDuration);
        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        // stop the tone playing:
        noTone(spkPin);
      }
    }

  public:
    spkControl(int pin) {
      pinMode(pin, OUTPUT);
      spkPin = pin;
    }

    void play(int m) {
      switch (m) {
        case 1:
          playMelody(win_melody, win_noteDurations);
          break;
        case 2:
          playMelody(loose_melody, loose_noteDurations);
          break;
```

```
        case 3:
          playMelody(temp_melody, temp_noteDurations);
          break;
        default:
          break;
      }
    }
};

#endif
```

MPU6050.h

Clase con las funciones de control del altavoz para la reproducción de melodías.

```
#ifndef MPU6050_H
#define MPU6050_H

#include <Wire.h>

class MPU6050 {
  private:
    int MPU_addr;
    int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
    double x, y, z;
    String orientation;

    int minVal = 265;
    int maxVal = 402;

  public:
    MPU6050(int mpu_addr) {
      MPU_addr = mpu_addr;
    }

    void reset() {
      Wire.begin();
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x6B); // Talk to the register 6B
      Wire.write(0x00); // Make reset - place a 0 into the 6B register
      Wire.endTransmission(true);
    }

    void read() {
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x3B);
      Wire.endTransmission(false);
      Wire.requestFrom(MPU_addr,14,1);

      AcX = Wire.read()<<8|Wire.read();
```

```
      AcY = Wire.read()<<8|Wire.read();
      AcZ = Wire.read()<<8|Wire.read();

      int xAng = map(AcX,minVal,maxVal,-90,90);
      int yAng = map(AcY,minVal,maxVal,-90,90);
      int zAng = map(AcZ,minVal,maxVal,-90,90);

      x = RAD_TO_DEG * (atan2(-yAng, -zAng)+PI);
      y = RAD_TO_DEG * (atan2(-xAng, -zAng)+PI);
      z = RAD_TO_DEG * (atan2(-yAng, -xAng)+PI);
    }

    String getOrientation() {
      read();

      if (abs(AcX) >= abs(AcY) && abs(AcX) >= abs(AcZ)) {
        if (AcX>0) orientation = "X: Green";
        else if (AcX<0) orientation = "-X: Blue";
      } else if (abs(AcY) >= abs(AcX) && abs(AcY) >= abs(AcZ)) {
        if (AcY>0) orientation = "Y: Red";
        else if (AcY<0) orientation = "-Y: Yellow";
      } else if (abs(AcZ) >= abs(AcX) && abs(AcZ) >= abs(AcY)) {
        if (AcZ>0) orientation = "Z: White Up";
        else if (AcZ<0) orientation = "-Z: White Down";
      }
      return orientation;
    }
};

#endif
```

credentials.h

Clase con las funciones de control del altavoz para la reproducción de melodías.

```
#ifndef CREDENTIALS_H
#define CREDENTIALS_H

const char* networkSSID = "MOVISTAR_933C";
const char* networkPASSWORD = "********";
const char* mqttSERVER = "mqtt.swx.altairone.com";
const char* mqttUSERNAME = "va7c24@tfm-juguete";
const char* mqttPASSWORD = "*********";

// Topics to subscribe to
// char subscribeTopic[] = "status/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/+";
String subscribeTopic = "spaces/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/";
```

```
String subscribeTopics[] =
{"game","timeout","difficulty","print_message","outputs"};

// Topics to publish the data to
char publishTopic[] = "spaces/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties";
char publishTopic_data[] = "spaces/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/data";

#endif
```

wifi-mqtt.h

Conjunto de funciones encargadas de la comunicación MQTT por Wifi.

```
#ifndef WIFI_MQTT_H
#define WIFI_MQTT_H

/* This file stores the functions needed to connect to both your WIFI and MQTT
broker.
The following libraries are needed:
-. SPI.h:  allows you to communicate with SPI devices, with the Arduino as the
master device.
-. WiFiNINA.h: allows you to use the following Arduinos' capabilities: UNO, NANO
33 IoT,MKR 1010 and MKR VIDOR 4000 WiFi.
-. PubSubClient.h: allows for MQTT messaging.
-. credentials.h: stores the credentials needed for the WIFI and MQTT connection.

Reference: arduino.cc
*/

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <credentials.h>
#include <PINS.h>
#include <ArduinoJson.h>

//Get the sensitive data stored at credentials.h

const char* ssid = networkSSID; // your network SSID (name)
const char* password = networkPASSWORD; // your network password
const char* mqttServer = mqttSERVER; // SmartWorks MQTT host

const char* mqttUsername = mqttUSERNAME; // MQTT Username of your thing in
SmartWorks IoT platform
const char* mqttPassword = mqttPASSWORD; // MQTT password of your thing in
SmartWorks IoT platform

int status = WL_IDLE_STATUS; // Initialice the Wifi radio's status
```

```
WiFiClient wifiClient; // Initialize the client library
PubSubClient client(wifiClient); // Creates a client that can connect to the
specified WIFI

// Store variables
int game_value = 0;
unsigned long timeout = 10000;
int difficulty = 2;
String print_message = "";
bool new_message = false;
bool outputs[6] = {0,0,0,0,0,0};

// Callback function handles the messages that are returned from the broker on
our subscribed channel
void callback(char* topic, byte* payload, unsigned int length){
  // Serial.print("Message arrived to ");
  // Serial.println(topic);
  // Serial.print("Payload: ");
  // for (unsigned int  i = 0; i < length; i++) {
  //   Serial.print((char)payload[i]);
  // }
  // Serial.println();

  StaticJsonDocument <256> doc;
  deserializeJson(doc, payload);

  if (doc.containsKey("game")) { game_value = doc["game"]; }
  if (doc.containsKey("timeout")) { timeout = doc["timeout"]; }
  if (doc.containsKey("difficulty")) { difficulty = doc["difficulty"]; }
  if (doc.containsKey("print_message")) {
    print_message = (const char*) doc["print_message"];
    new_message = true;
  }
  if (doc.containsKey("outputs")) { copyArray(doc["outputs"], outputs);  }
}

// wifi_mqtt_init function initializes wifi connection
void wifi_mqtt_init(){
  // Attempt to connect to Wifi network:
  while (status != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, password);
    // wait 10 seconds for connection:
    delay(10000);
  }

  // Connection successfull, print out the related data
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

```
  // Sets the mqtt server details: host and port
  client.setServer(mqttServer, 1883);
  // Sets the message callback function for subscribing to the specify topic
  client.setCallback(callback);
  delay(200);
}

// mqtt_reconnect function initializes or reestablished MQTT connection
void mqtt_reconnect(){
// Loop until we're reconnected
  while (!client.connected())
  {
     // Attempt to connect to MQTT broker
    Serial.print("Attempting MQTT thing connection...");
    // Try to connect to SmartWorks MQTT Broker, give it Arduino as the name
    if (client.connect("Arduino", mqttUsername, mqttPassword))
    {
      // Connection successfull, print out the related data
      Serial.println("connected");
      for (unsigned int  i = 0; i <
sizeof(subscribeTopics)/sizeof(subscribeTopics[0]); i++) {
        client.subscribe((subscribeTopic+subscribeTopics[i]).c_str());
      }
    } else{
      // Connection not successfull, print out error message
      Serial.print("failed, rc=");
      Serial.println(client.state());
      // Wait 5 seconds before retrying
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void mqtt_check() {
  client.loop();
}

// wifi_mqtt_send function sends data to the platform
void wifi_mqtt_send(char sentence[200], char pubTopic[300]){
// If the connection is lost, reconnect
  if (!client.connected()){
    mqtt_reconnect();
  }
  // Allow the client to process incoming messages and maintain its connection to
the server.
  client.loop();
  // Publishes a message to the specified topic.
  client.publish(pubTopic, sentence);
}

// update_property defines the logic to update the properties
void update_property(String property_name, String value){
```

```
  String json_payload = "{ \"" + property_name + "\":\"" + value + "\" }";

  //Define an auxiliar variable to convert the payload message to char
  char json_char[200];
  json_payload.toCharArray(json_char, 200);

  wifi_mqtt_send(json_char, publishTopic_data);     // Publish a message to the
raw history section
  wifi_mqtt_send(json_char, publishTopic);          // Publish a message to the
property topic
  // Serial.println(property_name+": "+value);        // Message to check the led
at PlatformIO Serial Monitor
}

void update_property(String property_name, int value){
  String json_payload = "{ \"" + property_name + "\":" + value + " }";

  //Define an auxiliar variable to convert the payload message to char
  char json_char[200];
  json_payload.toCharArray(json_char, 200);

  wifi_mqtt_send(json_char, publishTopic_data);     // Publish a message to the
raw history section
  wifi_mqtt_send(json_char, publishTopic);          // Publish a message to the
property topic
  // Serial.println(property_name+": "+value);        // Message to check the led
at PlatformIO Serial Monitor
}

// Print message if it has changed
void print_new_message() {
  if (new_message) {
    new_message = false;
    Serial.println(print_message);
  }
}

#endif
```

PINS.h

Definición de los pines con nombres de sus propiedades en la plataforma y número de pines.

```
#ifndef PINS_H
#define PINS_H

// LEDS
#define LED_WHITE_UP 16       // D0   - 0 WHITE UP
#define LED_BLUE 2            // D4   - 1 BLUE
#define LED_GREEN 14          // D5   - 2 GREEN
#define LED_YELLOW 12         // D6   - 3 YELLOW
#define LED_RED 13            // D7   - 4 RED
#define LED_WHITE_DOWN 15     // D8   - 5 WHITE DOWN
const int led_pins[6] = {LED_WHITE_UP, LED_BLUE, LED_GREEN, LED_YELLOW, LED_RED,
LED_WHITE_DOWN};
const String led_names[6] = {"led_white_up", "led_blue", "led_green",
"led_yellow", "led_red", "led_white_down"};

// BUTTONS
#define BUTTON_WHITE_UP 8     // B0   - 0 WHITE UP
#define BUTTON_BLUE 11        // B3   - 1 BLUE
#define BUTTON_GREEN 12       // B4   - 2 GREEN
#define BUTTON_YELLOW 13      // B5   - 3 YELLOW
#define BUTTON_RED 14         // B6   - 4 RED
#define BUTTON_WHITE_DOWN 15  // B7   - 5 WHITE DOWN
const int button_pins[6] = {BUTTON_WHITE_UP, BUTTON_BLUE, BUTTON_GREEN,
BUTTON_YELLOW, BUTTON_RED, BUTTON_WHITE_DOWN};
const String button_names[6] = {"button_white_up", "button_blue", "button_green",
"button_yellow", "button_red", "button_white_down"};

#endif
```

game.h

Clase con la funcionalidad principal del Proyecto, controla el funcionamiento de todos los componentes en función de los mensajes recibidos por MQTT.

```
#ifndef GAME_H
#define GAME_H

#include <Arduino.h>
#include <wifi-mqtt.h>

#define SPK 0  // D3
#include <spkControl.h>
spkControl spk(SPK);

const int MPU_addr = 0x68;
#include <MPU6050.h>
MPU6050 mpu(MPU_addr);

#include <PINS.h>
#include <Adafruit_MCP23X17.h>
Adafruit_MCP23X17 mcp;

// ------------------------ GENERAL FUNCTIONS ------------------------ //
// Function to compare arrays of booleans
boolean array_cmp(bool *a, bool *b, int len){
  for (int i=0; i<len; i++) if (a[i]!=b[i]) return false;
  return true;
}
// Swap 2 elements of array
void swap (int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
// Shuffle array
void shuffle(int arr[], int n) {
  for (int i = 0; i < 5; i++) {
    int j = random(0, n);
    swap(&arr[i], &arr[j]);
  }
}
// Check if value is in first n elements of array
boolean isInFirstn(int array[], int value, int n) {
 for (int i = 0; i < n; i++) {
    if (array[i] == value) {
      return true;
    }
  }
  return false;
}
```

```
// -------------------------- MAIN CLASS -------------------------- //
class Game {
  private:
    // Game variables
    bool inputs[6] = {0,0,0,0,0,0};
    bool prev_inputs[6] = {0,0,0,0,0,0};
    // bool outputs[6] = {0,0,0,0,0,0};
    bool prev_outputs[6] = {0,0,0,0,0,0};
    String orientation;
    String prev_orientation;
    unsigned long start_time;
    unsigned long current_time;
    bool win;
    int victory_count;

  public:
    Game() {
      start_time = 0;
      current_time = start_time;
      win = 0;
      victory_count = 0;
      orientation = "";
      prev_orientation = "";
    }

    // INIT - Initialize Serial, MPU6050, MCP23017, PINS and WifiMQTT
    void init() {
      Serial.begin(115200);
      // Reset MPU6050
      mpu.reset();

      // Connect to MCP23017 via I2C
      while (!mcp.begin_I2C()) {
        Serial.println("Error begining MPC23017");
        delay(1000);
      }
      // Set MCP button pins to input with pullup resistor
      for (auto button_pin: button_pins) mcp.pinMode(button_pin, INPUT_PULLUP);
      // Set nodeMCU LED pins to output
      for (auto led_pin: led_pins) pinMode(led_pin, OUTPUT);

      // Initialize Wifi and MQTT modules
      wifi_mqtt_init(); // Initialize wifi connection and mqtt server
      mqtt_reconnect(); // Initialize MQTT connection
      // init_delay(10000); //Set a default delay of 1000 miliseconds
    }

    // TEST - Turn on LEDs buttons when pressed and send data
    void test() {
      String orientation = mpu.getOrientation();
      if (orientation != prev_orientation) {
        prev_orientation = orientation;
```

```
        update_property("orientation", orientation);
      }
      for (unsigned int i = 0; i < sizeof(led_pins)/sizeof(led_pins[0]); i++) {
        inputs[i] = !mcp.digitalRead(button_pins[i]); // 1-HIGH Pressed   0-LOW
Not Pressed
        if (inputs[i] != prev_inputs[i]) {
          prev_inputs[i] = inputs[i];
          outputs[i] = inputs[i];
          digitalWrite(led_pins[i], outputs[i]);
          update_property(led_names[i], inputs[i]);
          update_property(button_names[i], outputs[i]);
        }
      }
    }

    // GAME2 - Press LEDs that turn on
    void game2() {
      // Game timer
      start_time = millis();
      current_time = start_time;
      win = 0;

      // Randomize list of indexes
      int num[] = {0,1,2,3,4,5};
      shuffle(num, 6);

      // The first n (difficulty) indexes are the on butttons
      memset(outputs, 0, sizeof(outputs));
      Serial.print("Outputs: ");
      for (unsigned int i = 0; i < 6; i++) {
        if (isInFirstn(num, i, difficulty)) outputs[i] = 1;
        Serial.print(outputs[i]);

        digitalWrite(led_pins[i], outputs[i]);
        if (outputs[i] != prev_outputs[i]) {
          prev_outputs[i] = outputs[i];
          update_property(led_names[i], outputs[i]);
        }
      }
      Serial.println();

      while (current_time - start_time < get_timeout()) {
        current_time = millis();

        // Get orientation
        String orientation = mpu.getOrientation();
        if (orientation != prev_orientation) {
          prev_orientation = orientation;
          update_property("orientation", orientation);
        }

        // Leer inputs
        Serial.print("Inputs: ");
```

```
        for (unsigned int i = 0; i < 6; i++) {
          inputs[i] = !mcp.digitalRead(button_pins[i]);
          Serial.print(inputs[i]);
          if (inputs[i] != prev_inputs[i]) {
            prev_inputs[i] = inputs[i];
            update_property(button_names[i], inputs[i]);
          }
        }
        Serial.println("");

        // Printear si correcto
        if (array_cmp(inputs, outputs, 6)) {
          victory_count++;
          Serial.print("You won: ");
          Serial.print(victory_count);
          Serial.println(" in a row");
          // spk.play(1); // Victory melody
          win = 1;
          break;
        }
        delay(200);
      }

      if (!win) {
        victory_count = 0;
        Serial.println("Timeout: You loose!");
      }
      delay(1000);
    }

    // SLAVE - Listen to MQTT and act
    void slave() {
      // Wait until message
      update_property("state", "Waiting");
      do {
        mqtt_check();
        print_new_message();
        delay(100);
      } while (print_message != "Started!");

      // Game timer
      update_property("state", "Running");
      start_time = millis();
      current_time = start_time;

      while (current_time - start_time < get_timeout() and game_value == 3 and
print_message != "Victory!") {
        current_time = millis();

        // Send inputs and write outputs
        String orientation = mpu.getOrientation();
        if (orientation != prev_orientation) {
          prev_orientation = orientation;
```

```
          update_property("orientation", orientation);
          update_property("state", "Running");
        }
        for (unsigned int i = 0; i < 6; i++) {
          inputs[i] = !mcp.digitalRead(button_pins[i]);
          if (inputs[i] != prev_inputs[i]) {
            prev_inputs[i] = inputs[i];
            update_property(button_names[i], inputs[i]);
            update_property("state", "Running");
          }
          digitalWrite(led_pins[i], outputs[i]);
          if (outputs[i] != prev_outputs[i]) {
            prev_outputs[i] = outputs[i];
            update_property(led_names[i], outputs[i]);
          }
        }

        delay(100);
        mqtt_check();
        print_new_message();
      }

      if (current_time - start_time >= get_timeout() and game_value == 3 and
print_message != "Victory!") {
        update_property("state", "Timeout");
      }
      else if (game_value != 3) update_property("state", "Game changed!");
      if (game_value == 3) delay(1000);
    }
};

#endif
```

7.1 CÓDIGO DE LA PLATAFORMA IOT

El siguiente código es el correspondiente a un archivo de Python llamado "update_game.py" y – hasta la parte de "`MQTT and MAIN`" – es el código que ejecuta el worker "update_game" en la plataforma de Altair:

```
import time
import requests
import json
import random

API_HOST = 'https://api.swx.altairone.com'
PATH = "/spaces/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties"
CLIENT_ID = "tfm-juguete::01FZ63X207KA9TEQENFAM54BKH"
CLIENT_SECRET = "4C7Q1M87ljNTkyMwHbYHphJX6Xq4UH"

def get_access_token():
    payload = f'grant_type=client_credentials&' \
              f'client_id={CLIENT_ID}&' \
              f'client_secret={CLIENT_SECRET}&' \
              f'scope=thing.read thing.update'

    headers = {'Content-Type': 'application/x-www-form-urlencoded'}
    response = requests.request("POST", API_HOST + "/oauth2/token",
                                headers=headers, data=payload)

    # print("get_access_token response: ", response.json())
    return response.json()['access_token']

def revoke_token(token):
    payload = f'token={token}&' \
              f'client_id={CLIENT_ID}&' \
              f'client_secret={CLIENT_SECRET}'
    headers = {'Content-Type': 'application/x-www-form-urlencoded'}
    response = requests.request("POST", API_HOST + "/oauth2/revoke",
                                headers=headers, data=payload)

    # print("revoke_token response: ", response)
    return response

def game2(json, headers):
    if json['state'] == "Running":
        # outputs = [ json['led_white_up'], json['led_blue'], json['led_green'],
json['led_yellow'], json['led_red'], json['led_white_down'] ]
        inputs = [ json['button_white_up'], json['button_blue'],
json['button_green'], json['button_yellow'], json['button_red'],
json['button_white_down'] ]

        if inputs == json['outputs']:
```

```
            requests.request("PUT", API_HOST + PATH, headers=headers,
json={"print_message": "Victory!"})
        else:
            requests.request("PUT", API_HOST + PATH, headers=headers,
json={"print_message": "Not quite right, keep trying!"})

    elif json['state'] == "Timeout":
        requests.request("PUT", API_HOST + PATH, headers=headers,
json={"print_message": "Timeout: You loose, try again!"})

    elif json['state'] == "Waiting":
        # Set n (difficulty) random buttons to 1
        sample = random.sample(range(0, 6), json['difficulty'])
        outputs = [1 if i in sample else 0 for i in range(6)]
        requests.request("PUT", API_HOST + PATH, headers=headers,
json={"print_message": "Started!", "outputs": outputs})


def handle(req):
    # body = req.body.decode("utf-8")
    # body = json.loads(body)
    token = get_access_token()
    # token = "hykuwHXjMrFI6HNTe11YH4OYshP-
8JzmP3fa3l1YFzw.QRw2ThME5xcfqXyzl6Gc4ctH1uixW4ntAVaiM0Gugqw"
    headers = {"Authorization": "Bearer " + token}

    # GET
    response = requests.request("GET", API_HOST + PATH, headers=headers)
    json = response.json()
    # print("response: ", json)

    if json['game'] == 3: game2(json, headers)

    revoke_token(token)
    return {
      "body": json,
      "status_code": response.status_code
    }


########## MQTT and MAIN ##########
import paho.mqtt.client as mqtt

host = "mqtt.swx.altairone.com"
user = 'va7c24@tfm-juguete'
password = '19VFH5AqIaU7AywG'
# property_topic = "status/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/#"
property_topic = "status/tfm-
juguete/collections/ESP8266/things/01FZ63X207KA9TEQENFAM54BKH/properties/state"

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
```

```
  print("Connected with result code "+str(rc))

  # Subscribing in on_connect() means that if we lose the connection and
  # reconnect then subscriptions will be renewed.
  client.subscribe(property_topic)

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    # print(msg.topic+": "+str(msg.payload))
    handle(msg)

def handle1(req):
  client = mqtt.Client()
  client.on_connect = on_connect
  client.on_message = on_message

  client.username_pw_set(username=user, password=password)
  client.connect(host, port=1883, keepalive=60, bind_address="")

  client.loop_forever()

if __name__ == "__main__":
    req = 0
    handle1(req)
```