Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

Máster Universitario en
Ingeniería Industrial

Trabajo Fin de Máster

Reloj y tablero de ajedrez para jugadores con discapacidad visual

Autor/a: Daniel Lorenzo López

Tutor/a: David Abreu Rodríguez

Cotutor/a: Vanesa Muñoz Cruz

3 de septiembre de 2021

La publicación de este Trabajo Fin de Máster solo implica que el estudiante ha obtenido al menos la nota mínima exigida para superar la asignatura correspondiente no presupone que su contenido sea correcto, aunque si aplicable. En este sentido, la ULL no posee ningún tipo de responsabilidad hacia terceros por la aplicación total o parcial de los resultados obtenidos en este trabajo. También pone en conocimiento del lector que, según la ley de protección intelectual, los resultados son propiedad intelectual del alumno, siempre y cuando se haya procedido a los registros de propiedad intelectual o solicitud de patentes correspondientes con fecha anterior a su publicación.

Resumen

El presente Trabajo Fin de Máster consiste en el diseño y construcción de un prototipo de bajo coste de un reloj y tablero de ajedrez adaptado para jugadores con discapacidad visual. El ajedrez adaptado es uno de los deportes más practicados, pero presenta barreras de entrada como el coste, elementos especializados o la complejidad de uso.

Los objetivos de este proyecto son proponer una alternativa de bajo coste, con nuevas funcionalidades y mejorar la experiencia de usuario para facilitar el uso. Para lograrlo, se ha usado la tecnología de bajo coste como el diseño e impresión 3D y Arduino.

La solución propuesta para el reloj ha sido el rediseño de los botones, colocando un teclado de navegación para el menú, botón independiente para el pausado del tiempo y botones laterales para la escucha del tiempo restante en audio mediante salida de auriculares. Se han programado las funciones básicas de los relojes adaptados; la salida de información por audio, la edición de los tiempos y las configuraciones de volumen y de idioma (español, inglés y alemán). Las innovaciones del diseño han sido la inclusión de pantalla individual para cada jugador y la programación rápida de la configuración de tiempos de partida mediante etiquetas RFID.

El tablero se ha diseñado en 3D realizando las modificaciones normativas para la adaptación del juego de personas con reducción visual mediante marcas táctiles. Se ha incluido bajo el tablero una base con pulsadores en cada posición para la detección de los movimientos de las piezas. Esta información se envía al reloj y se reproduce mediante audio la notación de la jugada.

El coste final de la construcción del prototipo ha sido de 167,68 € y pretende ser un proyecto de software libre.

Abstract

This Master's Thesis consists in the design and construction of a low-cost prototype of a clock and chessboard adapted for visually impaired players. Adapted chess is one of the most practiced sports, but it presents barriers to entry such as cost, specialized elements, or complexity of use.

The objectives of this project are to propose a low-cost alternative, with new functionalities and improve the user experience to facilitate usage. To achieve this, low-cost technology such as 3D design and printing and Arduino have been used.

The proposed solution for the chess clock has been the redesign of the buttons, placing a navigation keyboard for the menu, independent button for the paused time and side buttons for listening to the remaining time in audio through headphone output. The basic functions of the adapted chess clocks have been programmed; audio information output, editing of times, and volume and language settings (Spanish, English and German). The design innovations have been the inclusion of an individual screen for each player and the quick programming of the configuration of starting times using RFID tags.

The board has been designed in 3D making the regulatory modifications for the adaptation of the game of people with visual reduction through tactile marks. A base with pushbuttons in each position has been included under the board for the detection of the movements of the pieces. This information is sent to the clock and the notation of the play is reproduced via audio.

The final cost of the construction of the prototype has been € 167.68 and aims to be a free software project.

Índice General

Índice de Figuras

Índice de Tablas

Memoria

1 Introducción

El ajedrez es uno de los deportes más practicados por las personas con discapacidad visual (1). Los recursos para jugar son similares al ajedrez tradicional: un tablero con fichas y un reloj. Tanto el tablero como el reloj deben estar adaptados, ya sea mediante marcas táctiles o avisos auditivos. El ajedrez es uno de los deportes que permiten la integración de las personas con discapacidad visual, ya que independientemente del grado de discapacidad dos jugadores pueden participar.

Sin embargo, este tipo de recursos especializados son escasos y pueden a tener un costo elevado comparado con el ajedrez tradicional. A veces, los tableros adaptados son hechos a mano en madera por personal especializado y existen pocas empresas que fabriquen relojes de competición adaptados. Incluso los relojes oficiales pueden ser complicados de utilizar o no están suficientemente adaptados.

Este Trabajo Fin de Máster trata de proponer una alternativa de bajo costo, añadir funcionalidades y adaptar mejor los recursos necesarios para jugar al ajedrez sin condicionantes.

2 Introduction

Chess is one of the most practiced sports by visually impaired people. The resources to play are like traditional chess: a chessboard, and a chess clock. Both the chessboard and the chess clock must be adapted, either by touch marks or audible warnings. Chess is one of the sports that allow the integration of people with visual disabilities, since regardless of the degree of disability two players can participate.

However, such specialized resources are scarce and can have a high cost compared to traditional chess. Sometimes adapted chess boards are handmade in wood by specialized personnel and there are few companies that manufacture adapted competition chess clocks. Even official chess clocks can be complicated to use or are not sufficiently adapted.

This Master's Final Project tries to propose a low-cost alternative, add functionalities, and better adapt the necessary resources to play chess without conditions.

3 Objetivo y desglose de tareas

El objetivo de este TFM es el diseño y realización de un prototipo funcional de reloj y tablero de ajedrez, a bajo coste, simplificando el uso y añadiendo nuevas funcionalidades según los requerimientos de usuario. Para conseguir el objetivo, se realizarán las tareas:

- Estudio de las alternativas de mercado; tipología, funcionalidades y procedimientos de uso.
- Estudio de normativa reguladora específica de ajedrez adaptado: materiales y reglas.
- Estudio de requerimientos de usuario final
- Diseño 3D de piezas de tablero y reloj
- Realización de planos de piezas
- Impresión 3D de los modelos diseñados
- Ensamblaje de las piezas 3D
- Selección de componentes electrónicos adecuados.
- Programación de reloj y tablero.
- Estudio de mejoras funcionales futuras.
- Presupuesto de realización de prototipo.

4 Objective and task breakdown

The objective of this TFM is the design and realization of a functional prototype of clock and chessboard, at low cost, simplifying the use and adding new functionalities according to user requirements. To achieve the objective, the following tasks will be carried out:

- Study of market alternatives; typology, functionalities, and procedures of use.
- Study of specific regulatory regulations of adapted chess: materials and rules.
- Study of end-user requirements
- 3D design of board and clock parts
- Making part drawings
- 3D printing of the designed models
- Assembling 3D parts
- Selection of suitable electronic components.
- Clock and board programming.
- Study of future functional improvements.
- Prototype realization budget.

5 Especificaciones y requerimientos

Las especificaciones y requerimientos de este proyecto son parte dados por los tutores según la descripción del TFM y ampliados gracias a la colaboración de Valeriano Septién, trabajador en la ONCE como experto en tiflotecnologías, jugador y profesor de ajedrez adaptado.

5.1 Generales

a) El prototipo debe ser de bajo costo.
b) Los diseños deberán cumplir las funciones tradicionales de reloj y tablero adaptados.
c) El reloj y el tablero podrán conectarse para compartir información.

5.2 Reloj

a) Debe disponer de un interruptor de jugadas; de forma que cuando un jugador termine su turno, al pulsar se active el tiempo del oponente.
b) Debe disponer de pantallas para la muestra de información del mayor tamaño posible. Preferentemente los dos jugadores podrán visualizar la información.
c) Dispondrá de un botón para pausar el tiempo transcurrido.
d) Dispondrá de alimentación mediante pilas o baterías.
e) Dispondrá de menú para la navegación de las opciones del reloj.

5.2.1 Audio

f) Los tiempos de cada jugador se podrán modificar durante una partida. Esto es en caso de penalización de algún jugador.
g) Cada jugador dispondrá de audio mediante conexión de salida tipo Jack 3.5mm.
h) Los jugadores podrán conocer el tiempo restante propio o del oponente mediante el audio pulsando en un botón.
i) Dispondrá de varios idiomas de audio; español, inglés y alemán.
j) El volumen del audio se podrá modificar

5.3 Tablero

a) Las piezas podrán ser identificadas con facilidad mediante el tacto; tipo y color.
b) Las piezas podrán ser encajadas en el tablero mediante un vástago en el tablero.
c) Las casillas del tablero tendrán dos alturas para diferenciar el color.
d) El tablero tendrá la notación de la posición (letras y números) en el borde de este con texto en relieve y en braille para facilitar la identificación.
e) Dispondrá de algún sistema de detección de la posición de la pieza.

6 Antecedentes

El modelo Kaissa (Figura 1) es el reloj oficial de competición aprobado por la FEDC (Federación Española de Deportes para Ciegos). Cuenta con un diseño sencillo de balancín, pantalla y botones en un lateral. La navegación por los menús de programación se realiza por combinaciones de los botones laterales y el balancín. También dispone de salida de audio para auriculares de cada jugador. El reloj utiliza pilas de 1,5 V tipo AA. (2)

El modelo Kaissa tiene dos modos de funcionamiento principales: el modo de juego y el modo de programación. Durante el modo de juego se muestra el tiempo restante de ambos jugadores. Mediante la pulsación del balancín se descuenta el tiempo del jugador correspondiente. Cada jugador puede oír el tiempo propio o el tiempo del oponente usando los botones laterales más cercanos.

Para entrar en el modo de programación se debe parar el tiempo de la partida. Los parámetros modificables en este modo son la programación de tiempo, de modo, de idioma, de volumen, pitido y conocer el estado de la batería. Dentro de la programación de tiempo y modo el reloj cuenta con 8 controles preestablecidos, aunque es posible programar controles personalizados.

Figura 1: Reloj de ajedrez adaptado Kaissa

Las ventajas de este reloj son su sencillez en el diseño y la diversidad de funciones. Sin embargo, la experiencia de usuario en la programación del reloj es compleja. Además, las competiciones siguiendo las medidas actuales por el COVID-19 requieren la utilización de 2 tableros a los lados del reloj, lo que impide la visualización de la pantalla de uno de los jugadores y el acceso a los botones de tiempo. Por último, el coste de este reloj en tiendas independientes puede llegar a los 381€.[1]

[1] Fuente: https://www.visionfarma.es/116950-17428-reloj-parlante-de-ajedrez-kaissa.html

El uso de los tableros es totalmente tradicional. Los tableros adaptados normalmente son realizados por artesanos expertos. Estos tableros son realizados en madera y normalmente de pequeño tamaño. Teniendo en cuenta la realización de modificaciones en el tablero y las piezas de juego el coste de estos es más elevado que un tablero tradicional. Se pueden encontrar en tienda en torno a los 96€. [2]

Figura 2: Tablero ajedrez adaptado de madera

Estos tableros pueden tener diferencias en las tolerancias que pueden dificultar el encaje de las piezas o no tener el tamaño adecuado para el manejo según los reglamentos oficiales. Además, hay que tener en cuenta que los tableros no incluyen funciones automatizadas como la lectura de la posición de las piezas.

Conociendo estas alternativas de mercado, se pueden extraer varias conclusiones. Los diseños actuales tienen necesidad de una mayor facilidad de programación, doble pantalla, acceso a los productos y a bajo coste.

[2]Fuente: https://tiendaajedrezescacimat.es/es/conjuntos-ajedrez-tematico/684-conjunto-ajedrez-para-invidentes.html

7 Diseño del reloj

El diseño del reloj (Figura 3) se basa en el modelo kaissa, realizando modificaciones para corregir las desventajas presentadas por el mismo. Las dimensiones del reloj son 220 x 170 x 130 mm. Se modifica el teclado de navegación del menú y se introducen botones para la salida de audio, separando las funciones de estos. Se añade una segunda pantalla para la visualización de la información por un segundo jugador, pudiendo realizar una partida con dos tableros. Por último, se añade un lector de etiquetas RFID para facilitar la programación de los tiempos de la partida.

Figura 3: Reloj de ajedrez adaptado para personas con discapacidad visual

7.1 Carcasa

El diseño de la carcasa permite la colocación de los componentes y la distribución de las funciones del reloj (1: Plano General y 2: Secciones y detalle Carcasa del reloj). Se ha realizado una carcasa donde se aprovechan todos los laterales, pensando en un uso con dos tableros y el reloj en posición central. De esta forma los dos oponentes tienen acceso a las mismas funcionalidades.

En el frontal del reloj (Figura 4) está colocada una pantalla y el teclado de navegación del menú. También se encuentra el cableado para la conexión con el tablero. El frontal será usado por el jugador de piezas blancas.

Figura 4: Frontal del reloj

En la parte trasera (Figura 5) se encontrará la segunda pantalla y será usada por el jugador de piezas negras. Además, se encontrarán las conexiones para el arduino y el botón de encendido y apagado del reloj.

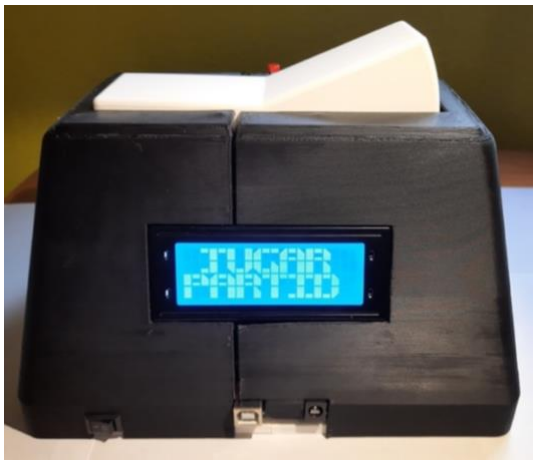


Figura 5: Trasera del reloj

Los laterales del reloj (Figura 6 y Figura 7) servirán para alojar los botones de tiempo propio y del oponente y para la conexión de salida de audio Jack 3.5 mm. El botón izquierdo más cercano al jugador será el tiempo propio y el botón derecho será el tiempo restante del oponente. En el lateral derecho del reloj en su parte superior se encontrará la zona del lector de etiquetas RFID, señalizada con un relieve, para la programación de los tiempos de la partida.

Figura 6: Lateral izquierdo del reloj

Figura 7: Lateral derecho del reloj

En la parte superior del reloj (Figura 8) se encontrará el interruptor de jugada o balancín y un pulsador para el pausado del tiempo.

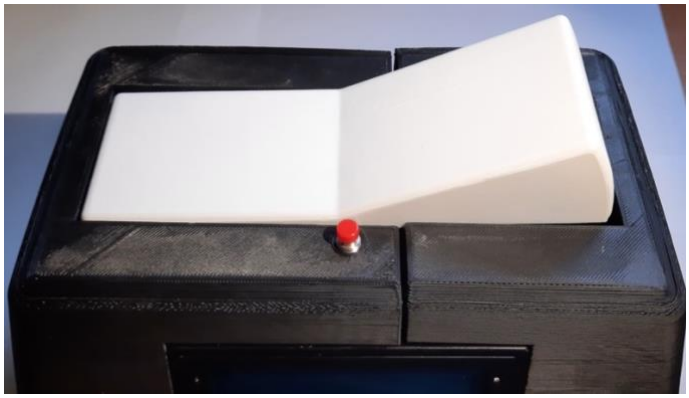


Figura 8: Parte superior del reloj

En la parte inferior la carcasa se unirá a la base del reloj mediante cuatro tornillos M4 en las esquinas (Base del reloj).

Figura 9: Parte inferior del reloj

En el interior de la carcasa se han diseñado salientes con agujeros para la colocación de algunos componentes con tornillos (pantallas, teclado de navegación y módulo RFID) o agujeros con el diámetro adecuado para la introducción de los componentes (conectores audio jack 3.5mm, pulsadores de tiempo y pausa).

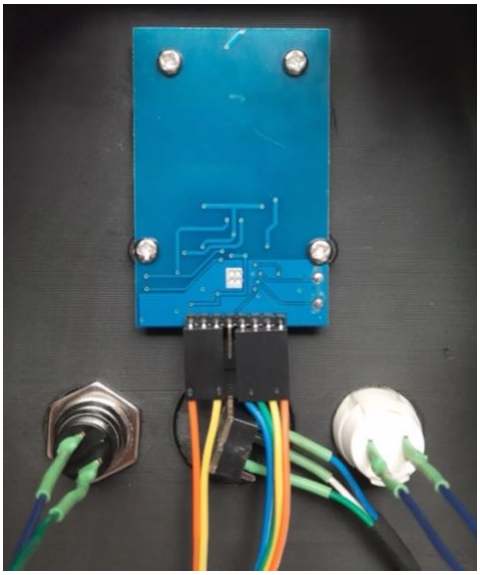


Figura 10: Montaje módulo RFID, pulsadores y conector jack 3.5mm

7.2 Interruptor de jugadas

El interruptor de jugada es el elemento principal del reloj de ajedrez, ya que marca el ritmo de la partida. Suele ser tipo palanca o balancín, ya que indica si el tiempo del jugador está descontando cuando el extremo del balancín del jugador está levantado. Para ello, se hará uso de un interruptor de palanca de 3 posiciones que estará sujeto a la carcasa (Figura 11) y al cual se le unirá una pieza qué hará de balancín horizontal (Figura 12 y Figura 13).

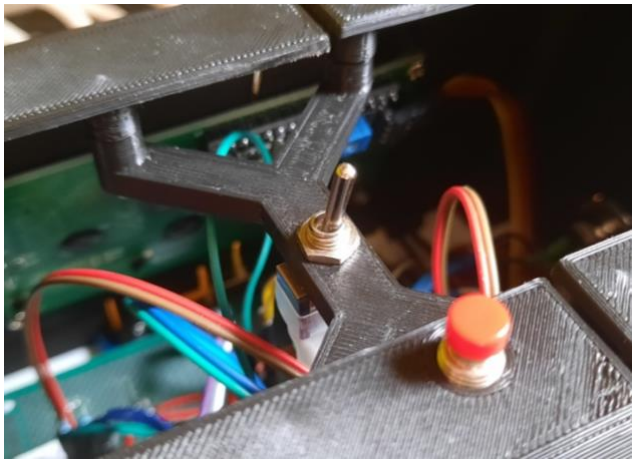


Figura 11: Interruptor de jugadas con sujeción a la carcasa

Para el diseño de la pieza balancín, se ha seguido el mismo ángulo de inclinación (24º) que tiene el interruptor. Siguiendo esto cada extremo presenta un ángulo de 12º y se ha buscado que, presionando un extremo, en el otro no sobresalga la parte inferior de la pieza.

Figura 12: Pieza Interruptor de jugadas

Figura 13: Parte inferior pieza interruptor de jugadas

7.3 Base del reloj

La base del reloj servirá para cubrir la parte inferior, alojar los componentes de alimentación y la electrónica del reloj (Figura 14). La base tendrá una abertura en la parte inferior para la colocación de las baterías. El portapilas estará sujeto a la base y se conectará a un interruptor de encendido y al convertidor de tensión a 5 V. El arduino mega se colocará en el centro de la base mediante tornillos (Figura 15).

Figura 14: Base de reloj (renderización)

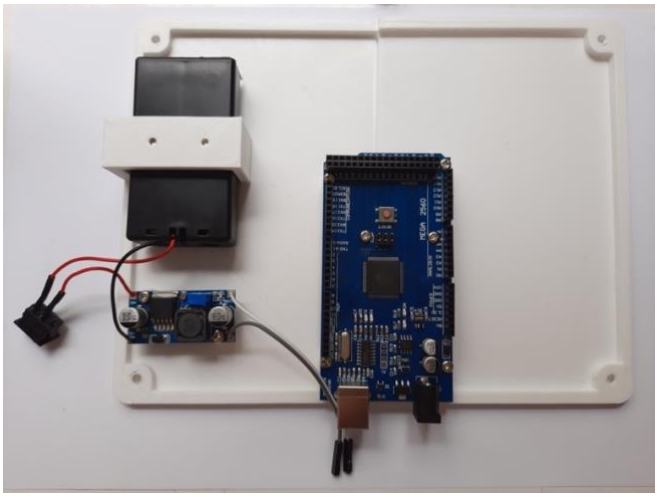


Figura 15: Base del reloj con circuito de alimentación

7.4 Conexiones

Las conexiones de los componentes con el arduino pueden resumirse gráficamente en la Figura 16. Están compuestas por botones y pulsadores, alimentación, pantallas, módulos MP3 y lector RFID.

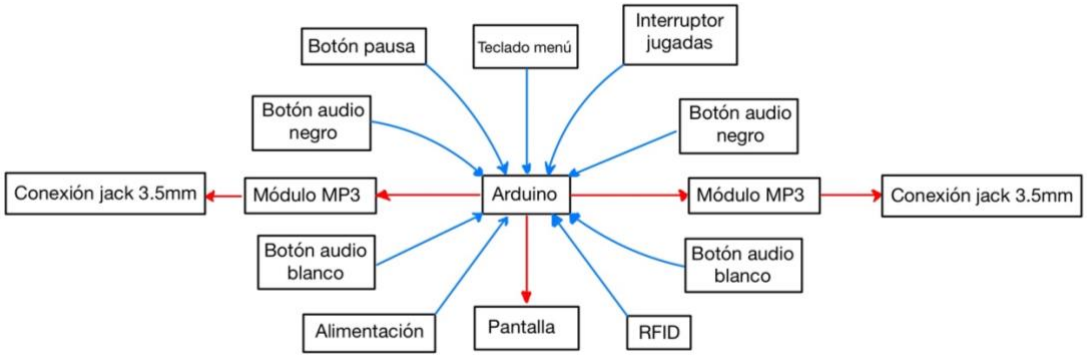


Figura 16: Gráfica de conexiones de módulos del reloj

Para realizar el conexionado físicamente, se ha hecho uso de una placa de prototipo, dada la flexibilidad en el diseño que aporta. Se han dispuesto pines para la colocación de la placa como una ampliación a la tarjeta controladora Arduino (Figura 17).

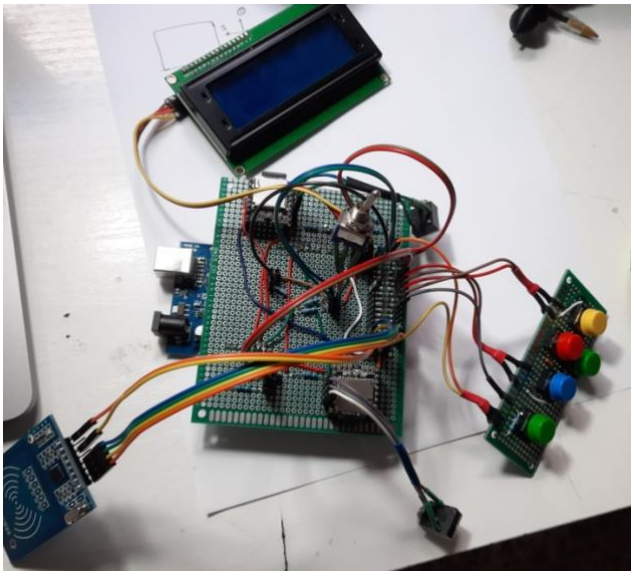


Figura 17: Placa de prototipo con algunos componentes

Los botones y pulsadores se alimentan a 5V por uno de sus pines, mientras que el otro pin que hará de señal deberá tener conectada una resistencia (330 Ω) pull-down a tierra. Esto se hace para que cuando no esté pulsado, la detección de la señal por el arduino no esté en un "estado flotante" donde puede ocasionar errores y por defecto la señal sea baja (0V). Además, estas resistencias evitan que se puedan producir cortocircuitos al conectar la alimentación a la entrada digital correspondiente.

El lector RFID se comunicará con el arduino mediante bus SPI (Serial Peripheral Interface), que tiene estructura maestro-esclavo full-duplex y requiere sincronización de señal de reloj (Figura 18). Se pasa un cable de datos a cada componente a conectar. Permite el envío y la

recepción de datos, por lo que este módulo sirve también para escribir información en las etiquetas. (3)

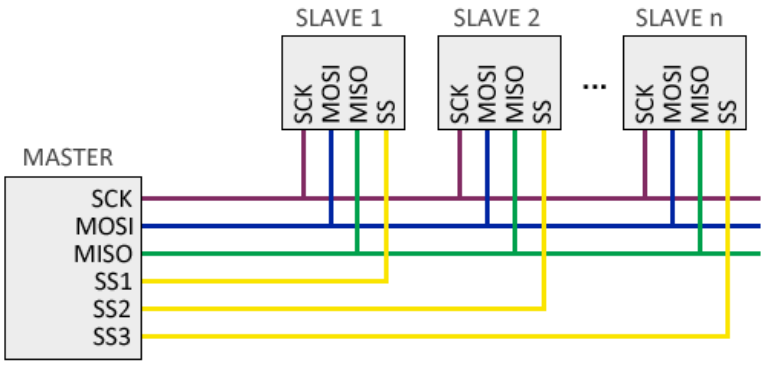


Figura 18: Comunicación SPI[3]

Las pantallas LCD disponen de un adaptador de comunicación I2C (Inter-Integrated Circuit). Al igual que el bus SPI, también tiene estructura maestro-esclavo. La conexión se realiza a través de 2 cables (SDA y SCK) de datos y señal de reloj (Figura 19). Cada componente debe tener una dirección distinta, para acceder a la información individualmente. Esto permite conectar dos pantallas a las mismas señales, con ahorro de cableado. (4)

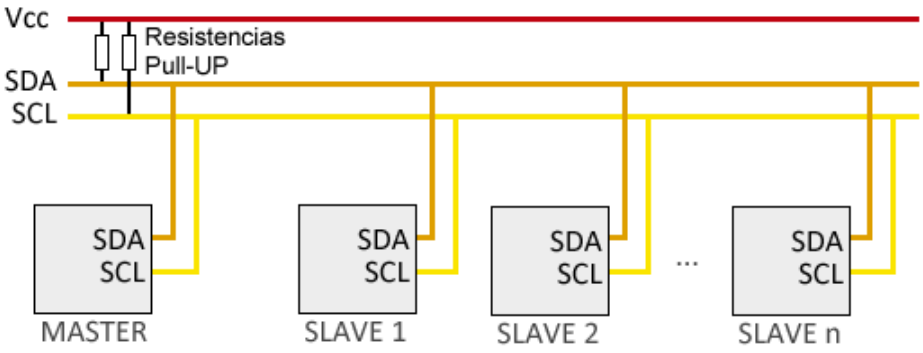


Figura 19: Comunicación I2C[4]

Los módulos MP3 para la reproducción de audio y la conexión con el tablero se comunican a través de puerto serie. Se conectan mediante 2 cables receptor y emisor (RX/TX). Este tipo de comunicación se utiliza también para la programación del propio controlador Arduino por cable USB. Para los módulos se utilizarán señales de salidas PWM como señales programándolas mediante software.

[3] Fuente: https://www.luisllamas.es/arduino-spi/
[4] Fuente: https://www.luisllamas.es/arduino-i2c/

8 Diseño del Tablero

Dado que el objetivo del diseño del tablero es realizarlo a bajo costo, se ha optado por la construcción de un prototipo con piezas impresas en 3d. La utilización de la tecnología de impresión 3d ofrece flexibilidad en el diseño, abaratamiento del prototipo y permite que la construcción pueda ser realizada por el público general.

Figura 20: Tablero de ajedrez adaptado para jugadores con discapacidad visual

El diseño del tablero se ha basado en un diseño para impresión 3D publicado en la plataforma de software libre Thingiverse (5). Se ha buscado un diseño de tablero tradicional con un tamaño de casillas y piezas típico según las normativas de las federaciones de ajedrez; el diseño Staunton (6).

Se trata de un diseño modular con una base tipo puzle donde se encajan las casillas para formar el tablero. Esta rejilla para las casillas permite que el diseño posterior para la colocación de los sensores pueda ser efectivo. Cada módulo cuenta con un agujero para atornillar a una base y hay espacio para que el vástago de cada pieza pueda pasar la base (Figura 22).

Figura 21: Diseño base tablero[5]

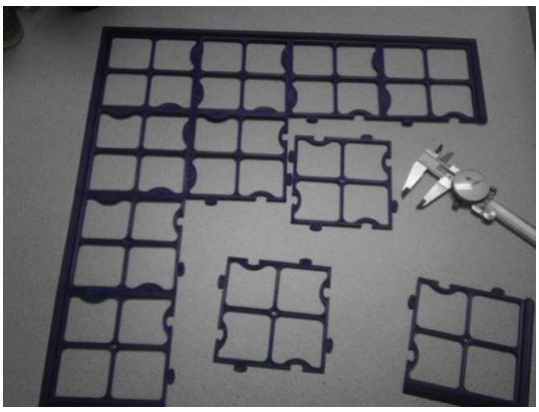


Figura 22: Rejilla puzle diseño base

8.1 Modificaciones de las piezas

Las modificaciones al diseño de estas piezas para que sean adaptadas al juego con personas con discapacidad visual han sido:

- A las piezas de juego se les ha añadido un vástago que permite que puedan ser encajadas en el tablero y no se caigan en la exploración de estas (Figura 23 y Figura 24). Para ello se ha optado por realizar un agujero en la base de estas y diseñar e imprimir el vástago de manera independiente, para facilitar la impresión y posteriormente unirlas con pegamento universal (cianoacrilato).

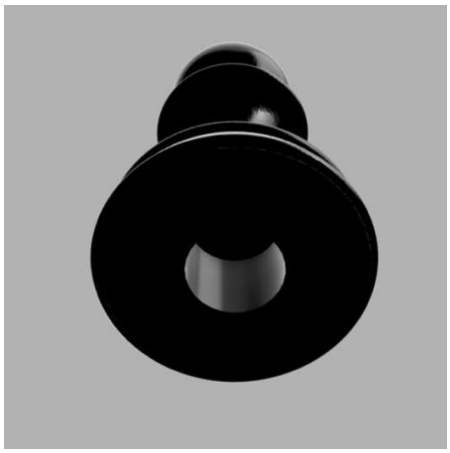


Figura 23: Base de peón con agujero para vástago (renderización)

Figura 24: Base de peón con vástago (impresión 3D)

[5] Fuente: https://www.thingiverse.com/thing:40605

- Para la diferenciación de las piezas negras y blancas, se añade a las piezas negras un pequeño identificador en la parte superior (Figura 25 y Figura 26). De esta forma, mediante el tacto se identifica el color de la pieza.

Figura 25: Identificador de pieza negra (renderización)

Figura 26: Identificador de pieza negra (impresión 3D)

- En cuanto al tablero, para diferenciar las casillas blancas y negras se ha aumentado la altura de las casillas negras con una diferencia de 5 mm (Plano 13: Casillas Negras y Blancas).
- Para garantizar el encaje de las piezas en el tablero, se ha realizado un agujero en las casillas con un pequeño chaflán para facilitar la introducción del vástago.

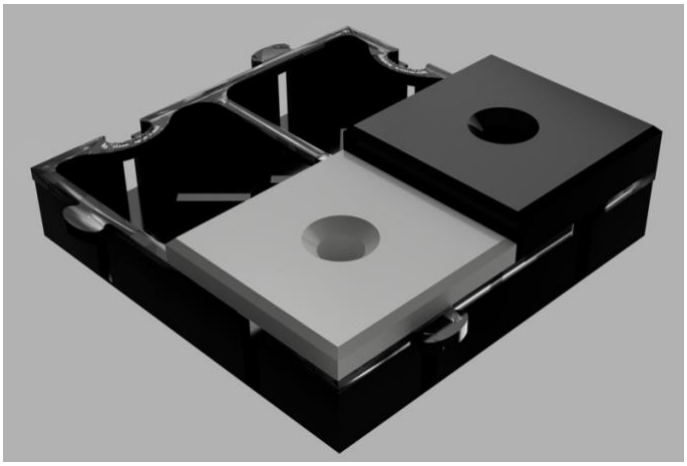


Figura 27: Ensamblaje casillas (renderización)

- Se ha añadido en los laterales del tablero la notación de las casillas (letras en los horizontales y números en los verticales) con relieve negativo y en braille.

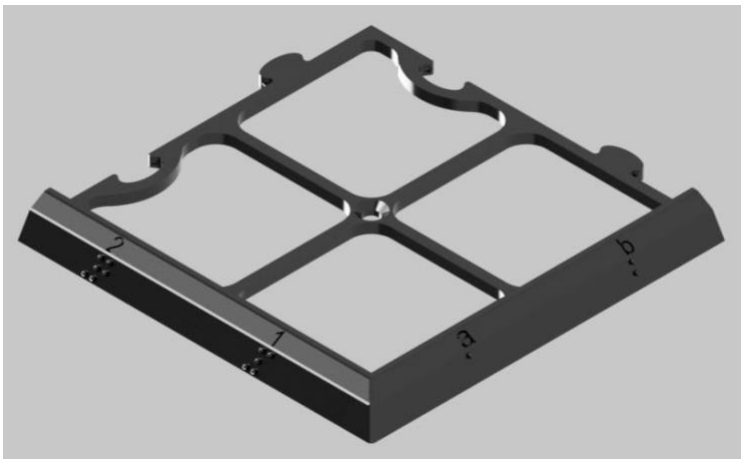


Figura 28: Esquina puzle con relieve en braille (renderización)

Figura 29: Numeración en braille (impresión 3D)

Por tanto, el número de piezas que se imprimen en 3D son:

- 32 piezas de juego (de cada color 16 piezas):
  - 2x8 peones
  - 2x2 torres
  - 2x2 caballos
  - 2x2 alfiles
  - 2x1 damas
  - 2x1 reyes
- 16 módulos de rejilla puzle (por la notación son individuales)
  - 4 centrales
  - 4 esquinas: 21ab, 78hg, ba87, gh12
  - 8 laterales: 34, 43, 56, 65, cd, dc, ef, fe
- 32 casillas de cada color (64 en total)

8.2 Base del tablero

Realizadas las modificaciones a las piezas obtenidas como iniciales, se procede a diseñar una base sobre la que se colocará el tablero, los componentes, cableado y electrónica del tablero. En cada módulo de la base se disponen 4 casillas y se fija al módulo puzle correspondiente mediante un tornillo en el centro. Esta pieza dispone de dos alturas distintas para la colocación de pulsadores, dado que las casillas presentan alturas distintas dependiendo del color y los vástagos de las piezas tienen la misma longitud. Además, se añaden muescas en las paredes exteriores para la interconexión del cableado entre módulos.

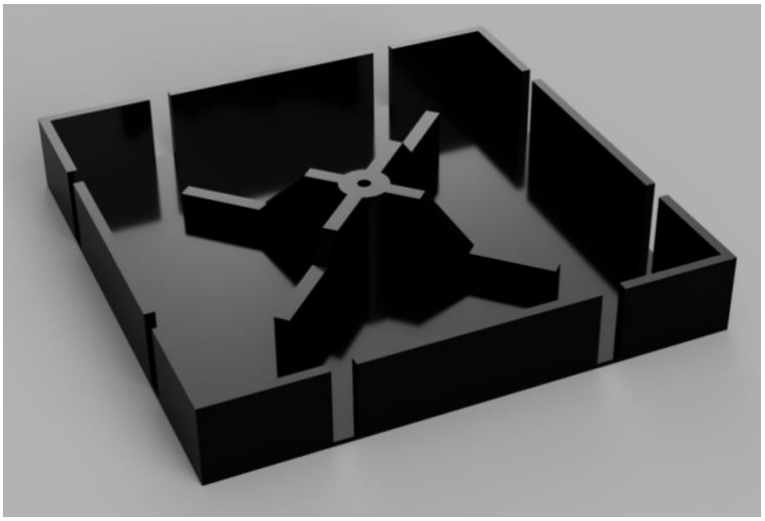


Figura 30: Base central del tablero (renderización)

Para los módulos base laterales (Plano 9: Módulos base laterales) y las esquinas (Plano 10: Base esquinas tablero) se realizará la misma configuración, exceptuando que las paredes que dan al borde exterior del tablero no incluyen muescas y se extienden unos milímetros para que coincida con el lateral de la rejilla y las casillas.

También hay que tener en cuenta la secuencia de casillas blancas y negras para que no coincidan 2 casillas contiguas con el mismo color. Por tanto, se han diseñado 2 tipos de esquinas y laterales distintas. Uno de los laterales será sustituido por un módulo central para anexionar otro módulo para la electrónica, incluyendo este su cubierta. De esta forma, la lista de módulos base necesarios son:

- 5 módulos centrales
- 2 módulos de esquina con casilla blanca
- 2 módulos de esquina con casilla negra
- 4 módulos laterales de letras

- 3 módulos laterales de números
- 1 módulo para electrónica
- 1 cubierta para electrónica

8.3 Módulo y cubierta para electrónica

Los módulos base y cubierta para electrónica se colocarán en un lateral del tablero. El arduino se fija a la base con los tornillos cercanos a los conectores USB y Jack de alimentación del arduino para garantizar la sujeción en caso de programar el controlador (Figura 31). La cubierta permite proteger la electrónica, fijarla y extraer el cable de interconexión con el tablero (Figura 32). La conexión con el tablero permite alimentarlo y enviar los datos para la reproducción de la notación.

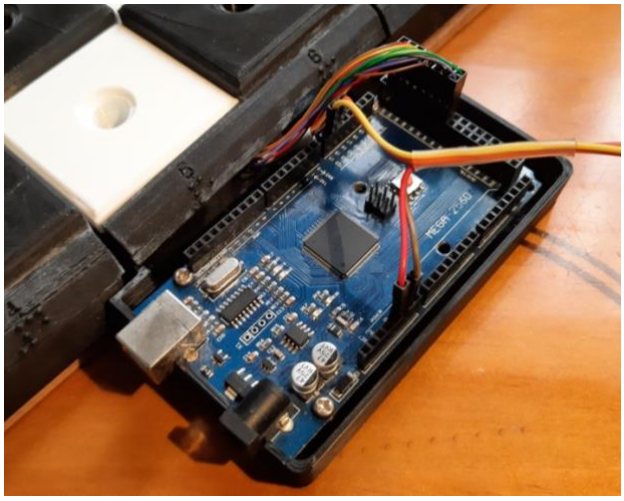


Figura 31: Módulo de electrónica sin cubierta

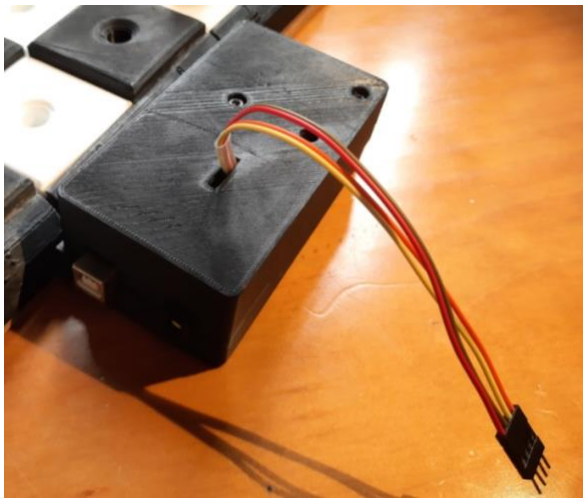


Figura 32: Módulo electrónica con cubierta

8.4 Circuito

El circuito del tablero consiste en una matriz de pulsadores de tamaño 8 filas y 8 columnas (Figura 34). Esto permite la reducción del número de señales de 64 a 16, al conectar los pulsadores en matriz en lugar de individualmente. También se elimina el uso de resistencias pull down. Este tipo de conexionado trata de simular un teclado matricial. Cuando uno de los pulsadores se active, la señal de la fila y de la columna se conectarán y se conocerá la posición.

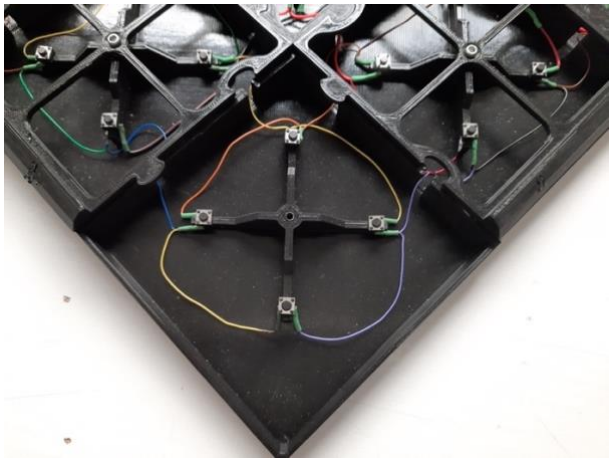


Figura 33: Detalle conexión de la matriz de pulsadores

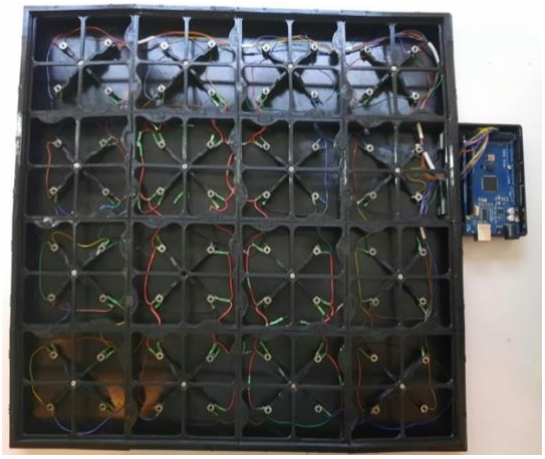


Figura 34: Matriz de pulsadores en la base del tablero

9 Entradas y sensores

9.1 Interruptor de jugadas

Se ha elegido un interruptor de palanca MTS-103 que hará de entrada para conocer qué jugador tiene el tiempo activo, accionado por la pieza que indicará también el turno. El interruptor se conecta a alimentación 5V y a las señales de entrada para el turno de cada jugador. Para el correcto funcionamiento de la lectura por parte del controlador, se añaden resistencias entre la salida de señal y tierra.

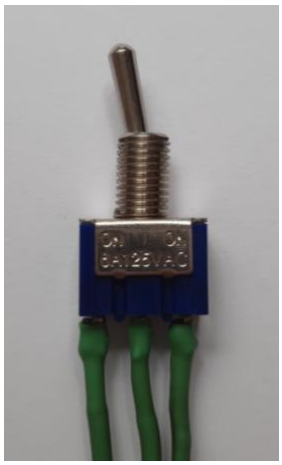


Figura 35: Interruptor de palanca para jugadas

El interruptor presenta una inclinación de 24º, por lo que el diseño de la pieza interruptor de jugadas tendrá el mismo ángulo.

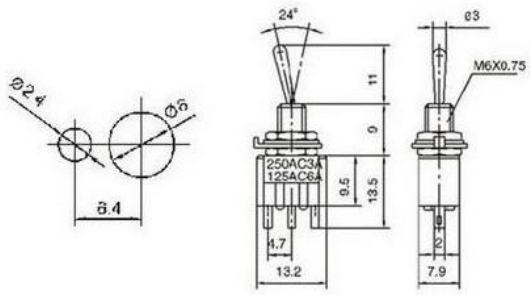


Figura 36: Interruptor palanca MTS-103[6]

[6] Fuente: http://k-electronica.es/complementos/342-mts-103-interruptor-de-palanca-de-tres-en-tenerife-canarias-la-laguna-8436545520451.html?search_query=interruptor&results=21

9.2 Teclado de navegación del menú

El teclado de navegación del menú se compone de 4 botones para el desplazamiento y 1 botón para aceptar. Se han seleccionado diferentes colores para los botones y con una disposición intuitiva para facilitar la navegación por el menú (Figura 37). Las funciones de los botones son:

- Botón amarillo: Atrás / Retroceder
- Botón Azul: Siguiente
- Botón verde superior: Subir cifra / Cambiar selección
- Botón rojo: Bajar cifra / Cambiar selección
- Botón verde derecho: Botón aceptar configuración

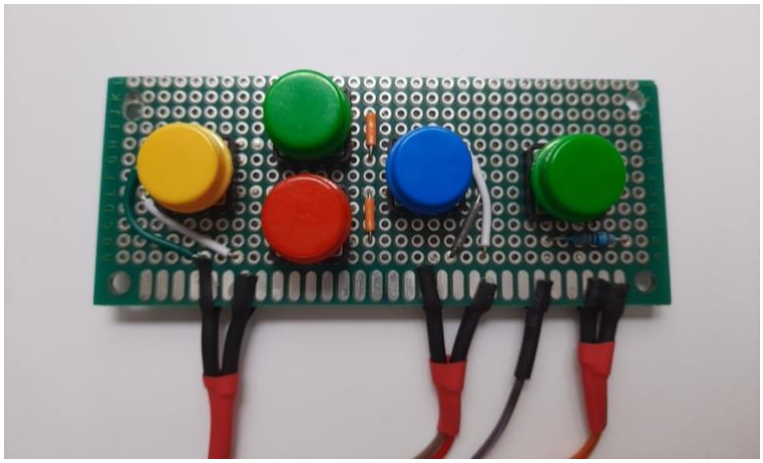


Figura 37: Teclado de navegación menú

9.2.1 Pulsador

Se ha elegido para el teclado el pulsador B3F 12x12mm Omron[7].

Figura 38: Pulsador para teclado de navegación

[7] Fuente: https://www.iberobotics.com/producto/pulsador-b3f-12x12mm-omron-tecla-color/

Características	
Corriente Max	50mA
Tensión DC	24V
Fuerza de funcionamiento	1,27N
Terminales de Interruptor	Soldables
Contacto	SPST-NO
Resistencia de contacto	100Mohm
Medidas	12 x 12 x 7,3 mm
Colores	verde, amarillo, azul, rojo

Tabla 1: Características pulsador B3F Omron

9.2.2 Circuito

Se ha realizado en una placa de prototipo, donde los 4 botones de selección forman una matriz de botones (4 señales de filas y columnas) y el botón de aceptar se alimenta con 5V y la señal se conecta a una resistencia pull down de 330 Ω .

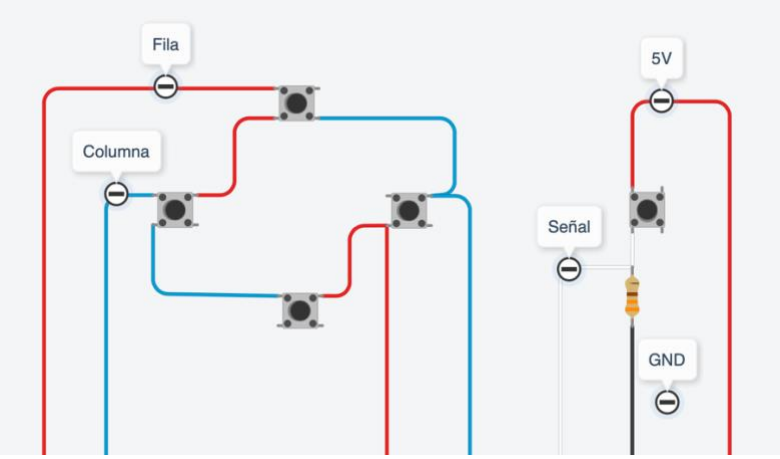


Figura 39: Circuito teclado de navegación

9.3 Botón de pausado

El botón de pausado permite detener el tiempo actual de la partida para realizar comprobaciones o modificaciones del juego por el árbitro cuando son solicitadas por alguno de los jugadores. El pulsador se alimenta a 5V y se conecta a una resistencia pulldown para el correcto funcionamiento del controlador.

Figura 40: Pulsador de pausado

9.4 Botones de escucha del tiempo restante

Son necesarios 4 pulsadores para la escucha del tiempo restante durante el desarrollo de la partida. Cada jugador contará con 2 pulsadores que le permitirá conocer el tiempo propio y el del oponente. Los pulsadores se podrán identificar mediante la posición en el reloj o por su color. Se alimentan a 5 V y conectan su señal de salida a una resistencia pull down.

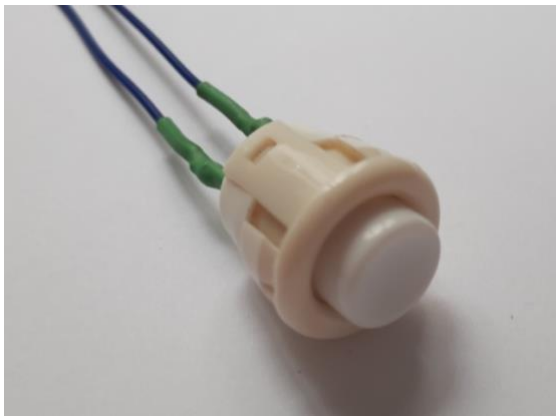


Figura 41: Pulsador blanco para audio de tiempo restante

Figura 42:Pulsador negro para audio de tiempo restante

9.5 Lector de etiquetas RFID

Para facilitar y añadir rapidez a la programación del reloj, se ha optado por incluir un lector de etiquetas RFID MFRC522 (Figura 43). La tecnología de identificación por radiofrecuencia permite que mediante etiquetas se puedan transmitir datos acercándolas al lector. En nuestro caso, los datos son los tiempos de juego de la partida (H:M:S +incremento de jugada), que se guardarán como variables para el sistema.

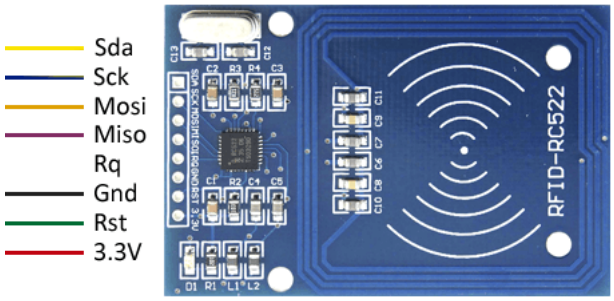


Figura 43: Módulo lector de etiquetas RFID[8]

El módulo lector RFID opera a frecuencia de 13.56 MHz, incorpora comunicación por bus SPI, bus I2C y UART, por lo que es sencillo de conectar con Arduino. El MFRC522 tiene un consumo de 13-26 mA durante la escritura, 10-13mA en stanby e inferior a 80uA en modo sleep. La tensión de alimentación es de 3.3V. (7)

Las conexiones del módulo MFRC522 con el arduino MEGA mediante SPI son las siguientes:

Señal	Pin
SPI SDA(SS)	53
SPI MOSI	51
SPI MISO	50
SPI SCK	52
GND	GND
RST	9
3.3V	3.3V

Tabla 2: Conexiones RFID-Arduino[9]

[8] Fuente: https://www.luisllamas.es/arduino-rfid-mifare-rc522/
[9] Fuente: Librería Arduino y https://www.luisllamas.es/esquema-de-patillaje-de-arduino-pinout/

Las etiquetas RFID tienen la ventaja de tener un bajo costo y una gran diversidad en su formato. Por ejemplo, pegatinas, llaveros (Figura 44) o tarjetas (Figura 45). Las etiquetas suelen tener 1024 bytes de memoria divididos en 16 sectores de 64 bytes.

Figura 44: Etiqueta RFID en formato llavero[10]

Figura 45: Etiqueta RFID en formato tarjeta[11]

Estos llaveros o tarjetas tienen la ventaja de que son personalizables, así que podría indicarse el tiempo de programación que se ha usado en la tarjeta con imprimación o pegatinas en braille.

[10] Fuente: http://k-electronica.es/complementos/208-llavero-de-electronica-inteligente-1356-mhz-rfid-en-tenerife-canarias-la-laguana-8436545519424.html?search_query=RFID&results=4

[11] Fuente:http://k-electronica.es/complementos/143-tarjetas-de-identificacion-rfid-1356-mhz-inteligentes-compatibles-arduino-en-tenerife-canarias-la-laguna-8436545518984.html?search_query=RFID&results=4

9.6 Pulsadores de posición en tablero

Se han seleccionado pulsadores simples utilizados en Arduino para la detección de la pieza y el conocimiento de la posición en el tablero (Figura 46). La ventaja de estos pulsadores es su pequeño tamaño y su bajo costo.

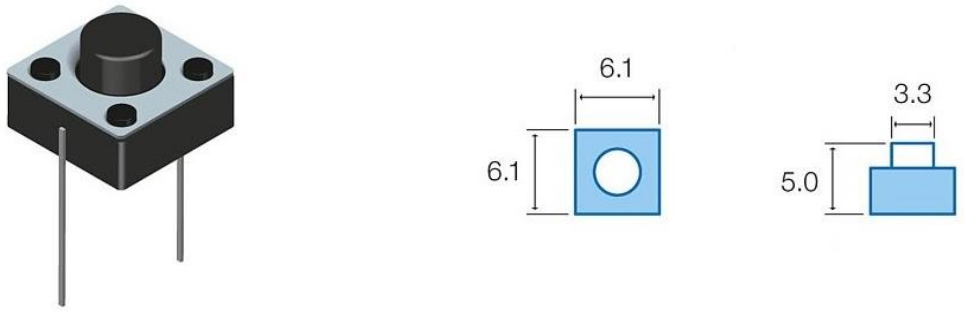


Figura 46: Pulsador para detección de pieza[12]

Se realiza la detección de la posición de la pieza formando una matriz de pulsadores por 8 filas y 8 columnas, de manera que, al accionar con el vástago, se activa la fila y columna correspondiente. Los pulsadores cuentan con 2 o 4 pines (2x2 pines puenteados), donde se conectan uno de ellos a la fila y el otro a la columna correspondiente. De esta forma, se crea un teclado matricial que será usado para identificar la posición. (8)

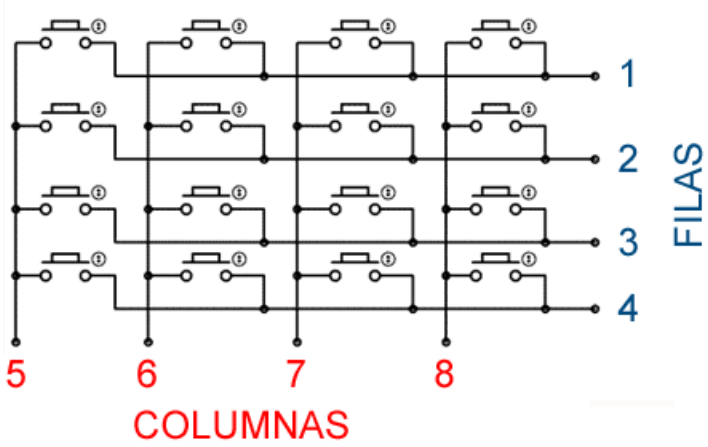


Figura 47: Esquema teclado matricial del tablero[13]

[12] Fuente: http://k-electronica.es/complementos/187-micro-interruptor-tactil-2-pin-arduino-canarias-tenerife-la-laguna-8436545519219.html?search_query=boton&results=15

[13] Fuente: https://www.luisllamas.es/arduino-teclado-matricial/

10 Salidas y actuadores

10.1 Pantallas LCD

Se ha seleccionado para la muestra de información 2 pantallas LCD 4x20 i2C (Figura 48). Estos displays cuentan con 4 filas y 20 columnas de caracteres. El tamaño de cada carácter es de 1 cm, pequeño para personas con discapacidad visual limitada, por lo que con uso de librerías se puede conseguir que el tamaño de caracter sea de 2 cm. Por eso se ha elegido el modelo con 4 filas evitando la pérdida de espacio en pantalla.

Figura 48: Pantalla LCD 4x20 i2C [14]

Figura 49: Adaptador I2C y pines de dirección[15]

Dispone de adaptador de interfaz de comunicación i2c, simplificando la conexión con el controlador arduino mediante 4 cables: alimentación; 5V, GND, señal de datos (SDA) y señal de reloj (SCL). Además, cuenta con un potenciómetro para regular la intensidad de brillo de la pantalla. El consumo típico de la pantalla es de 240 mA. (9)

Hay que tener en cuenta que de fábrica estas pantallas tienen la misma dirección i2C, así que para modificarla se puentean alguno de los pines A0, A1 y A2 que se encuentran detrás del adaptador (Figura 49).

[14] Fuente: http://k-electronica.es/complementos/36-modulo-lcd-compatible-arduino-robotica-educativa-en-canarias-tenerife-la-laguna-8436545518328.html?search_query=LCD&results=12

[15] Fuente: https://www.handsontec.com/dataspecs/I2C_2004_LCD.pdf

10.2 Módulo de audio MP3 y salida Jack 3.5mm

Para obtener una interfaz auditiva con la que utilizar el reloj, se ha seleccionado el módulo reproductor de audio MP3 DFplayer mini (Figura 50). Este módulo es capaz de reproducir el audio de una tarjeta SD con un circuito de forma autónoma o controlado mediante Arduino. Es utilizado en multitud de proyectos de audio por sus funcionalidades. (10)

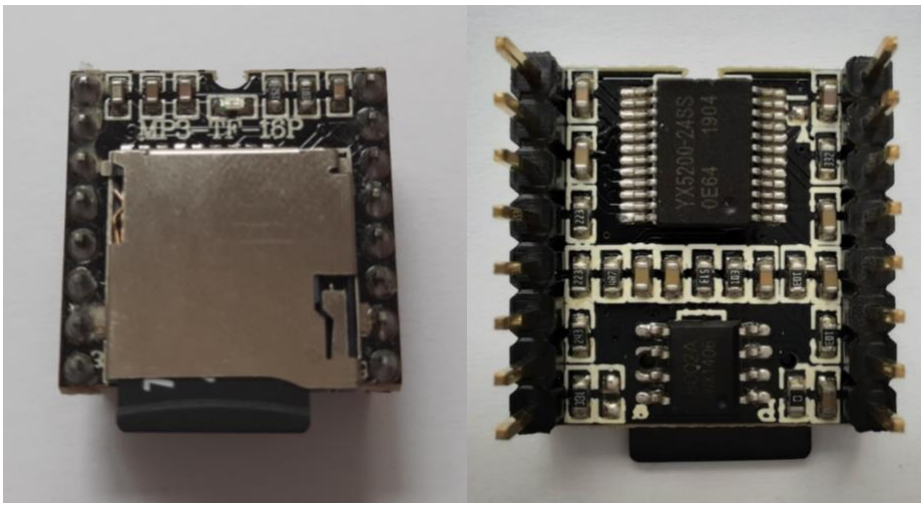


Figura 50: Módulo reproductor de audio MP3

Las características del módulo MP3 son:

- Distintas frecuencias de muestreo (kHz) 8/11.025/12/16/22.05/24/32/44.1/48
- Soportes de tarjetas SD hasta 32 Gb
- Variedad de opciones de control: serial, botones de función…
- Salida de señal ocupada para conocer si está reproduciendo
- Soporta más de 100 carpetas con 255 canciones en cada una
- 30 niveles de volumen con 6 opciones de ecualizador

El esquema de pines (Figura 51) permite conectarlo de diferentes formas. Sin embargo, se hará uso de la conexión serial (RX / TX) para el control mediante Arduino, la señal de ocupado (BUSY) para el control de la reproducción de los audios y la salida de audio mediante auriculares (DAC_R / DAC_I). La alimentación del módulo se realiza a 5V y conectado a GND. Para evitar el sonido de inicio del audio, se añade en la línea de la señal TX, una resistencia de 1kOhm.

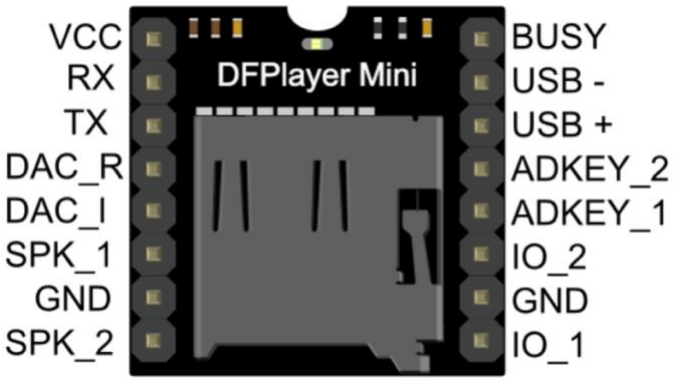


Figura 51: Pines de módulo DFPlayer mini MP3 [16]

La conexión de los pines de los módulos MP3 con el Arduino son:

Señal	Audio Negras	Audio Blancas
VCC	5V	
GND	GND	
RX	4	3
TX	5	2
BUSY	A9	A8

Tabla 3: Conexiones pines módulo MP3 - Arduino

Para la conexión de audio Jack 3.5mm, se ha hecho uso de un conector PG-324 (Figura 52). Se han añadido cables a los canales de audio derecho e izquierdo y la señal de tierra para la conexión con el módulo MP3. El esquema de pines (Figura 53) está compuesto por GND (1), Right (2, 3) y Left (4, 5).

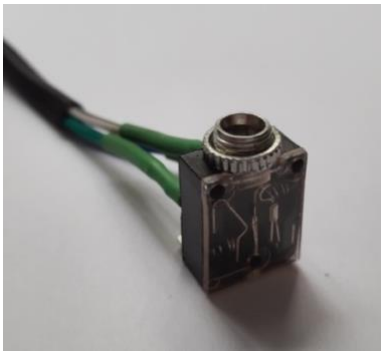


Figura 52: Conector Jack PG-324

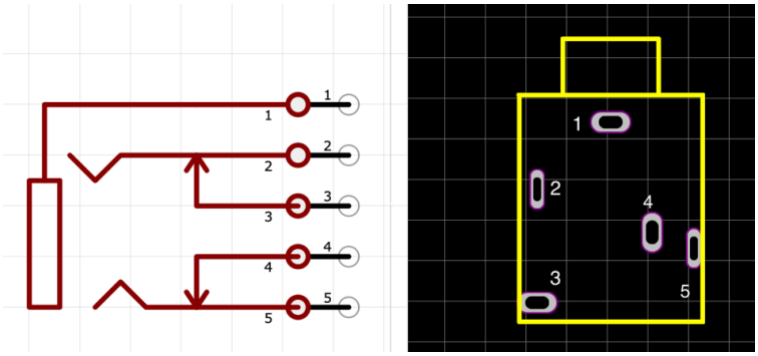


Figura 53: Esquema conector Jack PG-324 [17]

[16] Fuente: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299
[17] Fuente: https://easyeda.com/component/3b95c7a370174e5ca5f69662f818e993

11 Controlador

Teniendo en cuenta el número de conexiones de entrada y salida, se ha seleccionado como controlador del proyecto la placa Arduino MEGA (Figura 54), tanto para la programación de reloj como del tablero. La programación se realiza mediante Arduino IDE (11).

Figura 54: Placa controladora Arduino Mega[18]

Las características de la placa (12) son las siguientes:

- Microcontrolador basado en chip ATmega1280
- 54 pines entrada/salida (14 pines salida PWM)
- 16 entradas analógicas
- Oscilador de 16MHz
- Conexión USB y alimentación tipo Jack
- Voltaje de operación: 5V
- Memoria: flash 128kB, SRAM 8kB y EEPROM 4kB
- Comunicaciones: Serial (4 UARTs), SPI, I2C.

[18] Fuente: https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardMega

12 Alimentación

12.1 Pilas Panasonic NCR18650

Para la alimentación del proyecto, se requería suficiente autonomía para realizar una partida y voltaje estable para garantizar el correcto funcionamiento de todos los circuitos. Se ha optado por pilas de litio recargables PANASONIC NCR18650B (Figura 55)

Figura 55: Pila PANASONIC NCR18650B

Las características de la batería son:

- Capacidad: 3400mAh
- Voltaje nominal: 3.7 V
- Voltaje Máxima carga: 4.2 V
- Voltaje de corte: 2.75 V
- Dimensiones: 18mmx65mm
- Peso: 45g

Según el consumo estimado (2x240mA pantallas + 2x93mA Arduinos = 666 mA + consumo de resistencias, botones y resto de módulos) en torno a 1 A dependiendo de las condiciones de uso (refresco de la pantalla, uso de audio). Teniendo la curva de descarga (Figura 56) y considerando 3,3V como voltaje adecuado al finalizar, se obtiene una duración estimada de 165 minutos (2 horas y 45 minutos), suficiente para al menos una partida clásica (90 minutos con 40 movimientos y 30 minutos para el resto +30s de incremento).

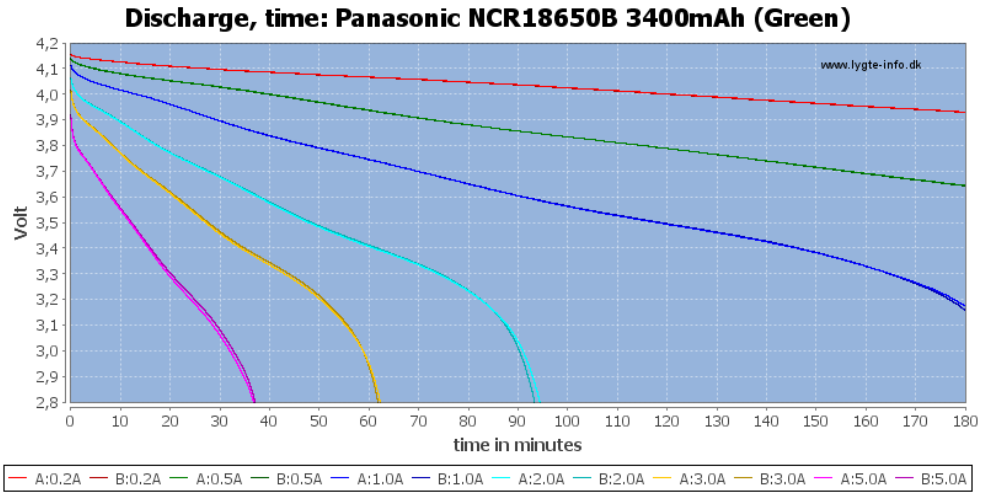


Figura 56: Gráfica descarga baterías 18650

El montaje se realiza con 2 baterías en serie, aumentando el voltaje a los 7,4 V nominales. Para ello, se hace uso de un porta pilas. Se podrá acceder a las baterías con una abertura en la base del reloj (Figura 57).

Figura 57: Montaje de pilas en la base del reloj

12.2 Módulo convertidor de tensión

Dado que el voltaje de trabajo del controlador y la mayoría de los componentes es de 5V se debe regular la tensión. Para ello, se ha decidido usar el módulo LM2596 (Figura 58). Se trata de un módulo convertidor de tensión Steep down, dado que desea obtener 5V regulados a partir del voltaje proporcionado por las baterías, 7,4V nominal. Cuenta con un potenciómetro para ajustar la tensión a la salida.

Las características del módulo son:

- Convertidor DC-DC Regulable
- LM2596 3A Steep Down
- IN: 4,5V - 35V
- OUT: 1,25V - 30V

Figura 58: Módulo convertidor DC-DC LM2596

12.3 Circuito de Alimentación

El circuito de alimentación se complementa con un interruptor para el apagado y encendido del reloj. Se conecta entre el positivo de las baterías y el módulo convertidor de tensión. La salida de 5V se conectará a la placa donde se realiza la interfaz con el controlador Arduino. Se puede ver el circuito en la Figura 15: Base del reloj con circuito de alimentación.

13 Funcionamiento del reloj

El reloj se compone por diferentes funciones que pueden ser combinadas para que la experiencia de usuario sea la adecuada y se cumplan los requerimientos del sistema. Las funciones son:

- Información en pantalla con texto grande
- Información mediante audio
- Menú de navegación entre las distintas funciones
    - Programación RFID
    - Edición de tiempos
    - Idioma
    - Volumen
- Temporizadores para el desarrollo de la partida
- Recepción de notación del tablero y reproducción en audio

13.1 Pantallas

Para la muestra de información por pantalla se crean caracteres personalizados grandes a partir de la composición de caracteres simples colocados en la posición de la pantalla adecuada. También se podrán usar caracteres vacíos o encendidos completamente.

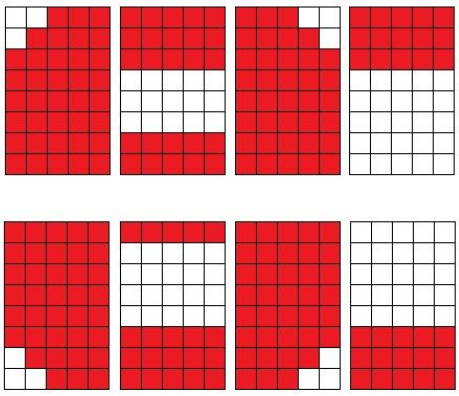


Figura 59: Caracteres especiales para composición de texto grande[19]

Cada letra o número grande ocupará 2 filas y varias columnas de caracteres de la pantalla LCD. Por cada caracter se realizará una función para su construcción y que devolverá el número de columnas usadas. Para componer una línea de texto con caracteres grandes, se irán ejecutando las funciones individuales, recorriendo el texto como vector de caracteres, y concatenando los

[19] Fuente: https://www.instructables.com/Custom-Large-Font-For-16x2-LCDs/

caracteres, dado que se conoce el número de columnas usadas (Figura 60). La programación de estas funciones está contemplada en el Anexo I: Programación → Caracteres grandes en pantalla.

Figura 60: Pantalla Menú Jugar Partida

13.2 Audio

Para reproducir información en formato de audio, se ha hecho uso del módulo MP3 DFPlayer Mini. Este módulo tiene soporte para tarjeta SD, donde se deben contener en carpetas los audios que se desean reproducir. Para ello se ha grabado en audio en los 3 idiomas elegidos (español, inglés y alemán) los números del 1 al 59, los indicadores de tiempo (hora/s, minuto/s y segundo/s), las funciones y acciones del reloj, los idiomas y las letras (codificadas por nombres) para la notación del tablero.

La edición de los audios se ha realizado con el programa Audacity (13), quitando los silencios y extrayendo únicamente el sonido para evitar tiempos muertos en la reproducción. Los audios que se contengan en la tarjeta SD deben estar numerados con 3 cifras y pueden estar en agrupados por carpetas numeradas por 2 cifras (para correcto funcionamiento las carpetas únicamente deben ser nombradas con la numeración):

Figura 61: Carpetas audios MP3 en SD

Para la comunicación del arduino con el módulo MP3, se ha hecho uso de la librería DFPlayer Mini Fast (14). Permite la selección del audio específico en carpetas y la configuración del volumen.

La salida de audio se utilizará para indicar los menús de navegación reproduciendo el audio correspondiente. Sin embargo, durante la partida o en la edición de los tiempos, se requiere que se reproduzca el tiempo restante completo; ya sea por activación de los pulsadores de tiempo laterales o por la navegación en el menú de forma automática. Para ello, se ha realizado una función para la decisión de qué secuencia de audio reproducir (Figura 62).

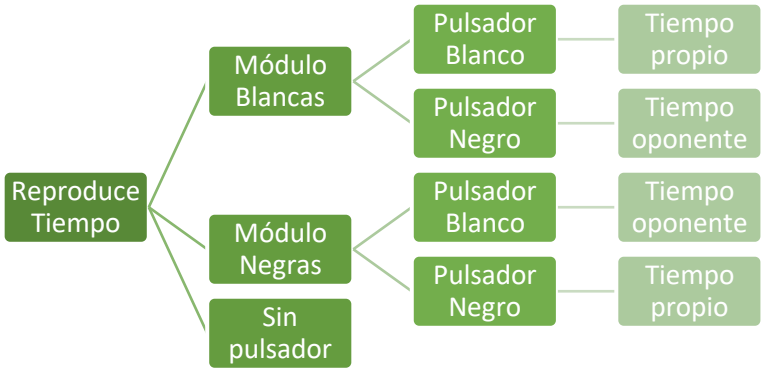


Figura 62: Gráfica decisión de reproducción de tiempo en audio

Se podrá activar la reproducción automática (sin pulsador) desde otras funciones, como al iniciar la partida o editar los tiempos manualmente. Esta opción reproduce el tiempo propio de cada jugador. Una vez decidido que tiempo reproducir, se activa la función correspondiente.

Teniendo un tiempo de jugador (H:M:S), se debe componer una secuencia de audio adecuada ("H" – "hora/s" – "M" – "minuto/s" – "S" – "segundo/s"), de forma que se reproduzca cuando el valor de cada una de las 3 variables sea distinto de 0. Esto quiere decir que cuando se tenga 0 se reproduce la siguiente variable. Por ejemplo, para un tiempo de 0:15:00 se reproducirá:

- *"quince minutos"*

Además, se tendrá en cuenta las características del idioma, principalmente español, con la reproducción del singular o el femenino. Por ejemplo, para un tiempo de 1:30:01 se reproducirá:

- *"una hora, treinta minutos, un segundo"*

La solución encontrada ha sido identificar cada audio o "palabra" de la secuencia con un número de identificador, de forma que cuando se da la orden de reproducir se añade 1 al identificador. La orden de reproducir el siguiente audio se realiza cuando se detecta que finaliza la duración del audio actual. Para ello, se utiliza el flanco de subida de la señal del pin BUSY que devuelve el módulo (Figura 63). Esta señal, que se lee como entrada analógica, permite conocer cuando se está reproduciendo un audio. Detectando la finalización del audio permite reproducir el siguiente sin realizar cortes.

Figura 63: Señal BUSY de módulo MP3

13.3 Menú de navegación

Para la creación de un menú de navegación se ha utilizado una red de Petri implementada en Arduino gracias a la librería PetriNet (15). Se ha utilizado este sistema para poder incluir las pantallas con texto grande, por la capacidad de personalización y la estabilidad ante errores una vez se ha configurado.

13.3.1 Red de Petri

La red presenta 12 estados con 27 transiciones (Figura 64). El estado inicial de la red es el estado 1.

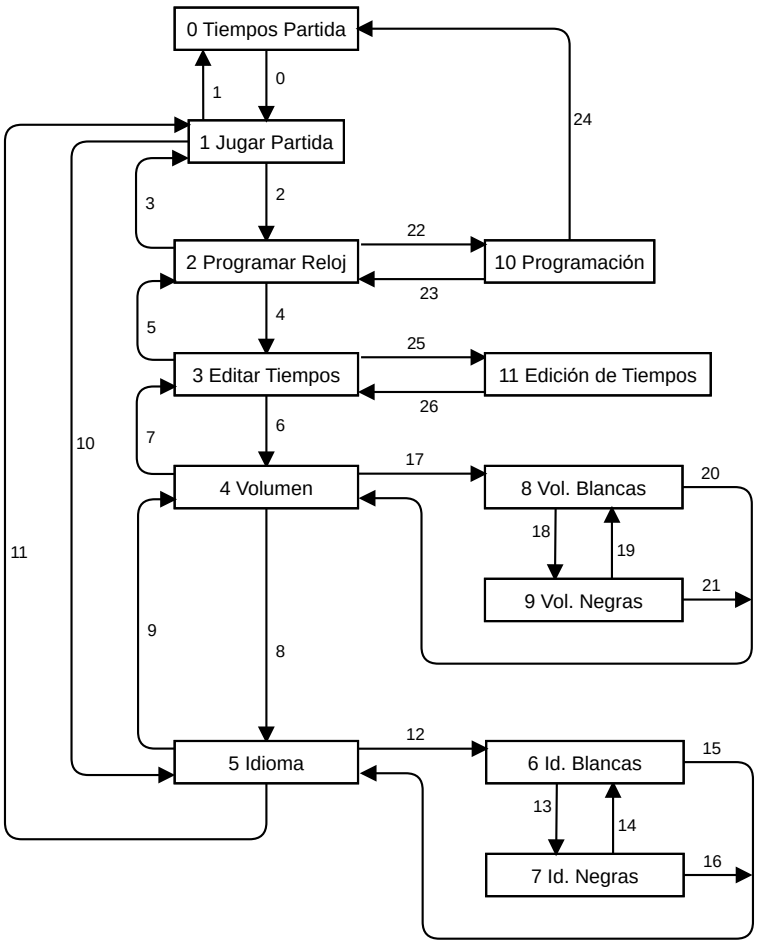


Figura 64: Red de Petri del menú de navegación

En la Tabla 4 se pueden ver las acciones de activación de la red, normalmente los botones del teclado de navegación: siguiente, atrás y enter.

Transición	Acción de activación	Transición	Acción de activación
0	Enter && Pausado	14	Siguiente || Atrás
1	Enter && Pausado	15	Enter
2	Siguiente	16	Enter
3	Atrás	17	Enter
4	Siguiente	18	Siguiente || Atrás
5	Atrás	19	Siguiente || Atrás
6	Siguiente	20	Enter
7	Atrás	21	Enter
8	Siguiente	22	Enter
9	Atrás	23	Atrás
10	Atrás	24	Programado (lectura correcta RFID)
11	Siguiente	25	Enter
12	Enter	26	Enter
13	Siguiente || Atrás		

Tabla 4: Transiciones Red de Petri de menú de navegación

El menú se ha configurado de forma que primero se seleccione la función a la que se quiera acceder y dentro se podrán modificar los parámetros que se deseen. Se podrá navegar por el menú cuando esté pausado el temporizador de la partida.

13.4 Tiempos de Partida

El estado de tiempos de partida es el conjunto de acciones fundamentales del reloj. Implica a varias funciones que combinadas permiten el desarrollo de la partida:

- Temporizadores de jugadores (Turnos y Pausa)
- Muestra de tiempos en pantalla
- Comprobación de ganador por finalización de tiempo
- Audio de tiempos solicitados por pulsador
- Audio de notación de movimiento del tablero

En este apartado se explicarán las funciones de temporizador, qué información se muestra en pantalla y la comprobación de ganador.

13.4.1 Temporizador

Las funciones de temporizador se componen de la lectura de la pausa y de la lectura de turnos. Para realizar el temporizador se usará la librería Count Up/Down Timer (16).

La pausa se activa o desactiva mediante la activación mantenida durante 0,5 segundos del pulsador superior del reloj. Así se evitan cambios en el estado de la pausa por pulsación no deseada. Para conseguir esta acción se activa una cuenta de tiempo en el flanco de subida de la señal del pulsador y tras el tiempo establecido; al detectar el flanco de bajada, se modifica el estado actual de la pausa. Cuando se modifica el estado, se inicia el temporizador del jugador activo (balancín extremo levantado) o se pausan los temporizadores.

Para modificar el turno, estando un temporizador activo; si se pulsa el balancín de un jugador indicando que su turno ha acabado, se detectará el flanco de subida de la señal del interruptor. Entonces, se pausará el temporizador activo, se le añadirá el incremento de tiempo si corresponde y se activará el temporizador del oponente.

13.4.2 Imprimir tiempo en pantalla

Se actualizará la información en pantalla cuando haya un cambio en las variables de tiempo o se cambie de turno. En la pantalla se mostrarán los tiempos de los 2 jugadores, uno en la línea superior y otro en la línea inferior, con indicadores en el lateral derecho de la pantalla (“B”-Blanco / “N”-Negro). Dado que cada jugador visualizará una pantalla, el tiempo propio estará en la parte superior y el tiempo del oponente en la parte inferior (Figura 65 y Figura 66).

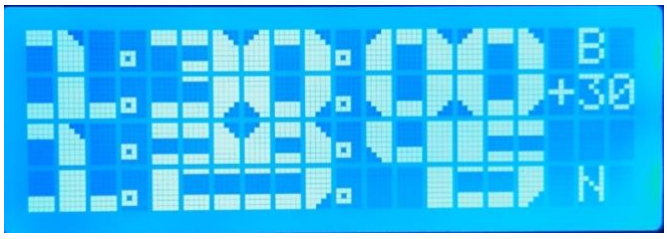


Figura 65: Pantalla Tiempos Jugador Blancas

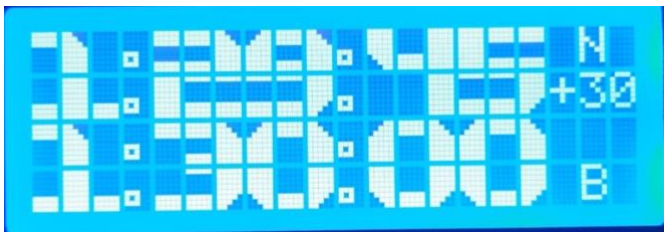


Figura 66: Pantalla Tiempos Jugador Negras

El tiempo restante se mostrará con las cifras suficientes; si es menor a 1 hora o 1 minuto solo se mostrarán los minutos y/o segundos restantes para evitar ceros a la izquierda (Figura 67).

Figura 67: Pantalla Tiempos pequeños

Por último, al llegar a cero el temporizador de uno de los jugadores, se habrá finalizado la partida y habrá ganado el oponente. El ganador se mostrará en pantalla (Figura 68 y Figura 69).

Figura 68: Pantalla Ganador Jugador Blancas

Figura 69: Pantalla Ganador Jugador Negras

13.5 Programación RFID de la partida

La programación mediante RFID se hace con el módulo MFRC522 y la librería del mismo nombre (17). Haciendo uso del ejemplo de lectura y escritura[20], se extraerá el código necesario para componer las dos funciones por separado; la lectura para el funcionamiento dentro del reloj y la escritura para la programación de los tiempos en las etiquetas como función externa.

La función se compone de dos pantallas (Figura 70 y Figura 71) se corresponden con los estados 2 y 10 de la red de Petri. Una vez se selecciona la función, se ejecuta la función de lectura de etiquetas.

[20] https://github.com/miguelbalboa/rfid/blob/master/examples/ReadAndWrite/ReadAndWrite.ino

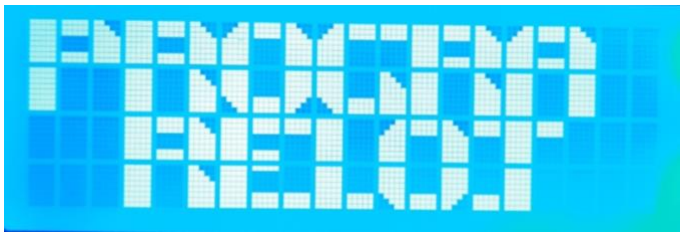


Figura 70: Pantalla Menú Programación Reloj

Al acercar la etiqueta a la zona de RFID en el lateral derecho del reloj (Figura 7), una vez leída la etiqueta se guardarán los valores y se programarán los tiempos. Se podrá iniciar la partida directamente. Internamente, el código que se ejecuta es el contenido en el Anexo I: 1.8 Lectura de etiquetas RFID.

Figura 71: Pantalla Programación RFID

Para entender el funcionamiento de la lectura y escritura, se debe entender la distribución de memoria de las etiquetas (18). La memoria de las etiquetas suele ser de 1Kb distribuidos en 16 sectores con 4 bloques cada uno. Cada bloque contiene 16 bytes de memoria. Al leer el contenido de la etiqueta tenemos:

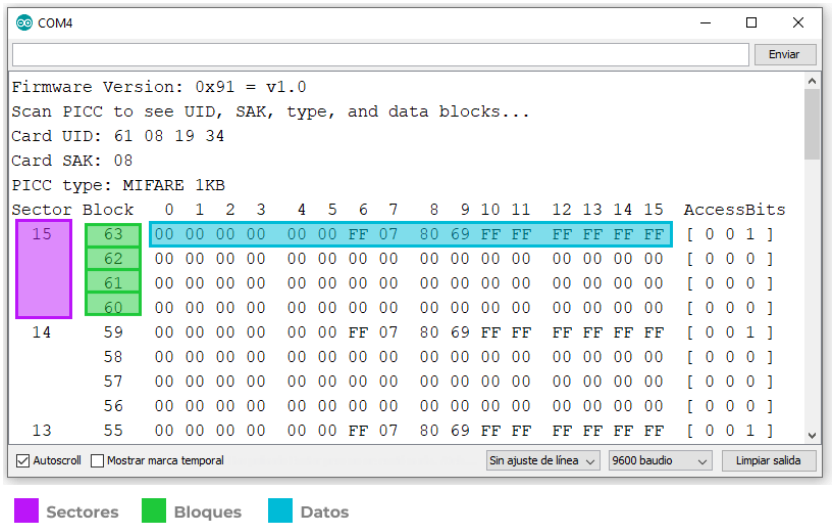


Figura 72: Distribución de memoria etiquetas RFID[21]

[21] Fuente: https://programarfacil.com/blog/arduino-blog/lector-rfid-rc522-con-arduino/

De cada sector solo se pueden escribir los 3 primeros. El último bloque de cada sector incluye las claves de cifrado o “tráiler” (Figura 73). Estas claves pueden ser modificadas para aumentar la seguridad. Por defecto son 00 y FF.

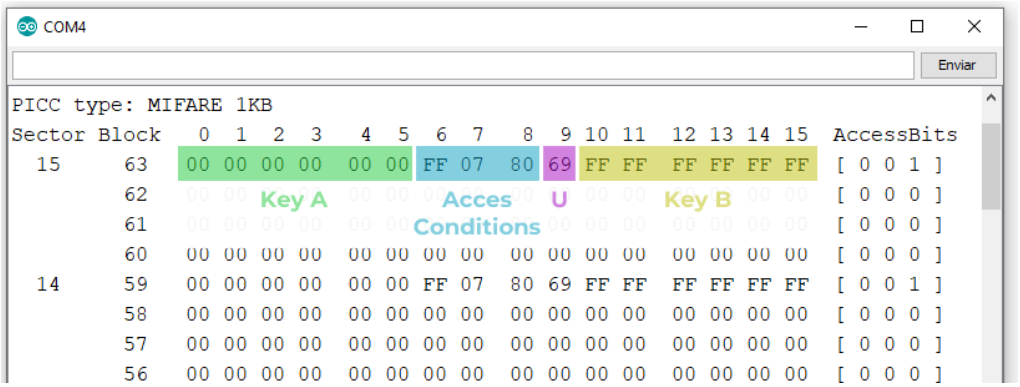


Figura 73: Claves de seguridad de etiquetas RFID

El primer bloque de memoria tampoco es usado ya que contiene el UID o identificador de la tarjeta y el identificador del fabricante.

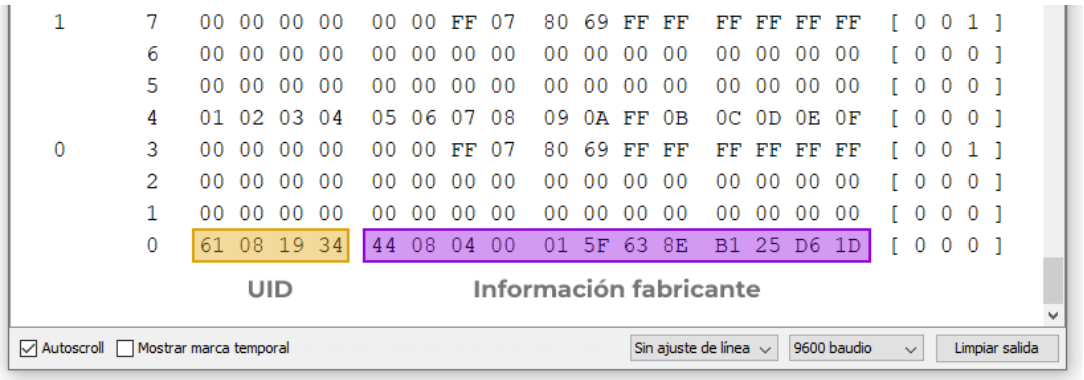


Figura 74: Información de tarjeta y fabricante etiqueta RFID

13.5.1 Programación de etiquetas RFID

Para la escritura de etiquetas se crea un código independiente que se ha incluido en el Anexo I: 1.9 Escritura de etiquetas RFID. La salida que imprime comprobando la correcta escritura:

Salida por monitor serie:

```
10:46:59.930 -> Escribir datos en sector 4 ...
10:46:59.967 ->  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10:47:00.005 ->
10:47:00.005 -> Leer datos del sector 4 ...
10:47:00.042 -> Data in block 4:
10:47:00.080 ->  00 00 015 00 00 00 00 00 00 00 00 00 00 00 00 00
10:47:00.117 -> 0 15 0 0
```

13.6 Edición de Tiempos

Para la edición de los tiempos de forma manual, en los estados de la red de Petri 3 y 11, corresponden a las pantallas (Figura 75 y Figura 76). La programación de las funciones de edición está contemplada en el Anexo I → Edición de Tiempos.

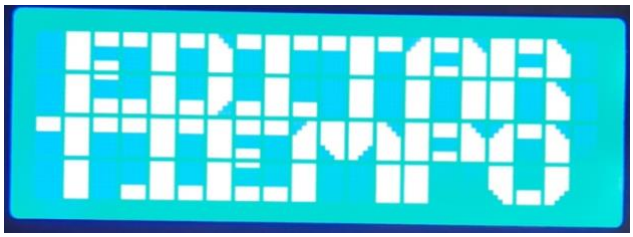


Figura 75: Pantalla Menú Editar Tiempos

La función de edición de tiempos identifica cada dígito editable con un identificador que permite asociar el cursor a su posición concreta. Se pueden modificar las decenas y unidades de los tiempos de cada jugador independientemente. La modificación del incremento es común para los dos jugadores.

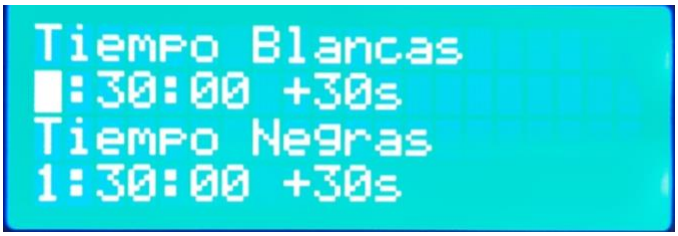


Figura 76: Pantalla Edición de Tiempos

La selección de la cifra a modificar se realiza con los botones; siguiente y atrás, y la modificación del valor se realiza con los botones más y menos. Configurados los tiempos se acepta con el botón Enter. Se reproduce en audio los indicadores de tiempo al seleccionarlos. Al modificar el tiempo, se reproduce la secuencia de audio automáticamente.

13.7 Volumen

Para la configuración del idioma, se realiza desde los estados 4, 8 y 9 de la red de Petri, correspondientes a las pantallas (Figura 77, Figura 78 y Figura 79).

Figura 77: Pantalla Menú Volumen

Una vez seleccionado la pantalla del volumen del jugador, se podrá modificar con un valor entre 0 y 30 con los botones de teclado de navegación de menú Más y Menos (verde superior o rojo inferior).

Figura 78: Pantalla Volumen Blancas

Figura 79: Pantalla Volumen Negras

13.8 Idioma

Para la configuración del idioma, se realiza desde los estados 5, 6 y 7 de la red de Petri, correspondientes a las pantallas (Figura 80, Figura 81 y Figura 82).

Figura 80: Pantalla Menú Idioma

Una vez seleccionado la pantalla del idioma del jugador, se podrá modificar entre español, inglés y alemán con los botones de teclado de navegación de menú Más y Menos (verde superior o rojo inferior).

Figura 81: Pantalla Idioma Blancas

Figura 82: Pantalla Idioma Negras

14 Funcionamiento del tablero

El funcionamiento del tablero se compone de dos partes, la lectura del movimiento de una pieza (Anexo I → Recepción de datos del tablero y reproducción de notación) y la reproducción en audio de la notación (Anexo I → Tablero). En realidad, el reloj realiza la segunda función, pero se ha decidido incluir en esta sección para entender mejor el proceso.

Se requiere que, al realizar un movimiento de una pieza del tablero, se reproduzca el audio que corresponde con la posición inicial y final del movimiento. Esto es útil para que los jugadores no tengan que realizar la tarea de "cantar la jugada" para que el oponente conozca el movimiento realizado. Los jugadores suelen anotar los movimientos que realizan, por lo que esta función puede ayudarles a comprobar los movimientos.

Sin embargo, esta función presenta limitaciones:

- No se comprueba que los movimientos realizados sean permitidos por las reglas de juego.
- No se reconoce el tipo de pieza que realiza el movimiento.

Será responsabilidad de los jugadores y árbitros la comprobación de jugadas y saber el tipo de pieza que realiza el movimiento.

El tablero hace un barrido de las filas y las columnas para detectar qué pieza está pulsada. Se ha añadido un tiempo de 300 milisegundos entre la pulsación y la detección para evitar detecciones no deseadas. Esto se realiza porque los jugadores tienen permitido tocar las piezas y el tablero para reconocer las posiciones.

Figura 83: Tablero vista superior

El proceso es el siguiente:

- En la primera detección se guarda la posición y se envía al tablero la indicación de que se ha iniciado el movimiento.
- Tras la segunda pulsación, el tablero compone la notación de la jugada con la información de la posición inicial y la posición actual. Por ejemplo, un peón de b2 a b4 el tablero compone la notación *b2b4.*
- El tablero envía los mensajes de pulsación inicial y notación del tablero mediante comunicación serial al reloj.
- El reloj podrá recibir información si los temporizadores están activos.
- El reloj recibe la información y reproduce un pitido si se detecta posición inicial. Si se recibe la notación, reproduce una secuencia de audio de las letras y números de esta.

En cuanto a la notación hay que tener en cuenta que por requerimientos de usuario los números estarán en idioma alemán y las letras tienen que estar codificadas con nombre que comienzan estas. Por tanto, las letras y números están codificadas de tal manera que los audios son:

- a = anna
- b = bella
- c = césar
- d = david
- e = eva
- f = félix
- g = gustav
- h = héctor

- 1 = ein
- 2 = zwei
- 3 = drei
- 4 = vier
- 5 = fünf
- 6 = sechs
- 7 = sieben
- 8 = acht

Por ejemplo, la notación c3e6 sería:

- *c3e6 = "césar, drei, eva, sechs"*

15 Presupuesto

El presupuesto se ha subdividido en apartado de impresión 3D y electrónica de reloj y tablero.

15.1 Impresión 3D

Los valores de peso y tiempo empleados pueden variar dependiendo de la impresora empleada, la calidad y relleno que se usen en las piezas. Los resultados son orientativos y personales dada la configuración usada.

Conociendo esto el presupuesto de impresión 3D está recogido en las tablas: Tabla 5: Mediciones de impresión 3D del tablero, Tabla 6: Presupuesto de Impresión 3D del tablero y Tabla 7: Presupuesto de Impresión 3D del reloj.

Descripción	Cantidad	Peso	(g)	Tiempo	(h)
		unid.	Total	unid.	Total
Base Central	5	27,97	139,85	1:47	8,92
Base Lateral	7	32,46	227,22	2:00	14,00
Base Esquina	4	32,46	129,84	2:00	8,00
Base electrónica	1	20,76	20,76	1:27	1,45
Cubierta electrónica	1	36,82	36,82	2:20	2,33
Puzzle Centro	4	8,17	32,68	0:33	2,20
Puzzle Lateral	8	12,59	100,72	0:47	6,27
Puzzle Esquina	4	17,33	69,32	1:03	4,20
Casilla Blanca	32	9,82	314,24	0:36	19,20
Casilla Negra	32	13,61	435,52	0:48	25,60
Rey	2	16,91	33,82	1:41	3,37
Dama	2	16,23	32,46	1:37	3,23
Peón	16	6,67	106,72	0:41	10,93
Torre	4	13,05	52,2	1:16	5,07
Caballo	4	16,56	66,24	1:38	6,53
Alfil	4	11,98	47,92	1:12	4,80
Vástago	32	2,15	68,8	0:15	8,00
Total	162		1915,13		134,10

Tabla 5: Mediciones de impresión 3D del tablero

Descripción	Coste Material	(€)	Coste elec.	€	Coste	Pieza (€)
	unid.	Total	unid.	Total	unid.	**Total**
Base Central	0,56 €	2,80 €	0,04 €	0,21 €	0,60 €	**3,01 €**
Base Lateral	0,61 €	4,27 €	0,05 €	0,34 €	0,66 €	**4,61 €**
Base Esquina	0,61 €	2,44 €	0,05 €	0,19 €	0,66 €	**2,63 €**
Base electrónica	0,42 €	0,42 €	0,03 €	0,03 €	0,45 €	**0,45 €**
Cubierta electrónica	0,74 €	0,74 €	0,06 €	0,06 €	0,80 €	**0,80 €**
Puzzle Centro	0,16 €	0,64 €	0,01 €	0,05 €	0,17 €	**0,69 €**
Puzzle Lateral	0,25 €	2,00 €	0,02 €	0,15 €	0,27 €	**2,15 €**
Puzzle Esquina	0,34 €	1,36 €	0,03 €	0,10 €	0,37 €	**1,46 €**
Casilla Blanca	0,19 €	6,08 €	0,01 €	0,46 €	0,20 €	**6,54 €**
Casilla Negra	0,25 €	8,00 €	0,02 €	0,61 €	0,27 €	**8,61 €**
Rey	0,35 €	0,70 €	0,04 €	0,08 €	0,39 €	**0,78 €**
Dama	0,32 €	0,64 €	0,04 €	0,08 €	0,36 €	**0,72 €**
Peón	0,13 €	2,08 €	0,02 €	0,26 €	0,15 €	**2,34 €**
Torre	0,26 €	1,04 €	0,03 €	0,12 €	0,29 €	**1,16 €**
Caballo	0,34 €	1,36 €	0,04 €	0,16 €	0,38 €	**1,52 €**
Alfil	0,24 €	0,96 €	0,03 €	0,12 €	0,27 €	**1,08 €**
Vástago	0,04 €	1,28 €	0,01 €	0,19 €	0,05 €	**1,47 €**
Total		36,81 €		3,22 €		**40,03 €**

Tabla 6: Presupuesto de Impresión 3D del tablero

Descripción	Cantidad	Peso (g)	Tiempo	Coste Material	Coste Electricidad	Coste total
Carcasa Reloj	1	203,07	9:37:00	4,06 €	0,23 €	**4,29 €**
Base Reloj	1	55,48	3:41:00	1,11 €	0,09 €	**1,20 €**
Balancín	1	55,33	3:22:00	1,11 €	0,08 €	**1,19 €**
Sujeción interruptor	1	5,65	0:28:00	0,11 €	0,01 €	**0,12 €**
Total	4	319,53	17:08:00	6,39 €	0,41 €	**6,80 €**

Tabla 7: Presupuesto de Impresión 3D del reloj

En conclusión, se han impreso un total de 162 piezas con un uso de material de 1,95kg de PLA y un coste de 36,81 €. Se han dedicado 134,1 h de impresión, es decir, 5 días 14 horas y 6 minutos; teniendo un coste de electricidad de 3,22 €. El coste de la impresión del tablero ha sido de 40,03 €.

El reloj ha requerido 319,53 g de PLA, con 17:08 h de impresión, un coste material de 6,39 € y un consumo de 0,41 €. El coste de impresión del reloj es de 6,8 €.

El presupuesto de impresión 3D del proyecto es de 46,83 €. El tiempo de impresión en total es de 6 días, 7 horas y 13 minutos.

15.2 Electrónica

Se procede a detallar el coste de los componentes electrónicos. Se recogen en las tablas: Tabla 8: Presupuesto de electrónica del reloj y Tabla 9: Presupuesto de electrónica del tablero

Descripción	Cantidad	Precio	Total
Arduino Mega	1	15,00 €	**15,00 €**
Módulo LCD 20x4 I2C	2	12,00 €	**24,00 €**
Módulo sonido DFPlayer mini	2	3,43 €	**6,86 €**
Conectores audio Jack 3.5mm	2	1,70 €	**3,40 €**
Tarjeta SD	2	2,50 €	**5,00 €**
Módulo RFID y 2 etiquetas	1	8,56 €	**8,56 €**
Pulsador Negro	2	1,95 €	**3,90 €**
Pulsador Blanco	2	0,85 €	**1,70 €**
Pulsador B3F Omron + tecla	5	0,60 €	**3,00 €**
Pila NCR18650B	2	7,50 €	**15,00 €**
Portapilas 2x NCR18650B	1	0,99 €	**0,99 €**
Convertidor DC-DC LM2596	1	2,65 €	**2,65 €**
Interruptor Encendido	1	0,32 €	**0,32 €**
Interruptor Palanca MTS-103	1	1,07 €	**1,07 €**
Placa de prototipo 90x150 mm	1	4,10 €	**4,10 €**
Resistencia 1k Ohm 0,25W	6	0,07 €	**0,42 €**
Resistencia 330 Ohm 0,25W	3	0,07 €	**0,21 €**
Tira 40 pines Macho	3	0,27 €	**0,81 €**
Tiras 40 pines Hembra	2	0,27 €	**0,54 €**
40x Cable DuPont Hembra-Hembra	1	2,56 €	**2,56 €**
Total			**100,09 €**

Tabla 8: Presupuesto de electrónica del reloj

Descripción	Cantidad	Precio	Total
Arduino MEGA	1	15,00 €	15,00 €
Pulsador 2 pines	64	0,07 €	4,48 €
Cableado	19,2	0,05 €	0,96 €
Terminales DuPont	16	0,02 €	0,32 €
Total			**20,76 €**

Tabla 9: Presupuesto de electrónica del tablero

Se concluye que el coste de componentes del reloj es de 100,09 € y el coste de electrónica del tablero es de 20,76 €. El presupuesto de componentes electrónicos es de 120,85 €

El coste final del proyecto es de 167,68 €.

16 Conclusiones y líneas futuras

Observando los objetivos planteados al inicio de este proyecto, se puede afirmar que se han cumplido. Los relojes y tableros adaptados tienen la problemática de un diseño y experiencia de usuario compleja. Gracias a la impresión 3D y a las tecnologías de bajo coste como Arduino, pueden adecuarse el diseño y las funciones a los requerimientos del usuario final.

Con la diversidad de funciones que estas herramientas aportan, se han añadido algunas innovaciones respecto a las alternativas existentes; el diseño de un tablero funcional que añade función de notación de los movimientos, las mejoras de experiencia de uso, la inclusión de 2 pantallas o la programación del reloj mediante RFID.

Agregando estas funciones, el objetivo principal era construir un prototipo de bajo coste, que no suponga una barrera de entrada a este deporte. El coste del proyecto es de 167,68 €. Viendo las alternativas existentes en el mercado, se ha conseguido una solución a bajo coste.

Sin embargo, se han encontrado varias limitaciones que pueden ser resultas en futuras versiones o mejoras del proyecto. La primera es el diseño y uso de placas PCB para la construcción de las conexiones al Arduino del reloj y el teclado de navegación. Otras son funciones que podrían añadirse como la programación de etiquetas RFID como función interna del reloj o la inclusión de memoria en el tablero; para la identificación por tipo pieza y modificación de posiciones en la partida.

17 Conclusions and future lines

Looking at the objectives set at the beginning of this project, it can be said that they have been met. Adapted clocks and boards have the problem of a complex design and user experience. Thanks to 3D printing and low-cost technologies such as Arduino, the design and functions can be adapted to the requirements of the end user.

With the diversity of functions that these tools provide, some innovations have been added with respect to the existing alternatives; the design of a functional chessboard that adds movement notation function, the improvements of user experience, the inclusion of 2 screens or the programming of the chess clock using RFID.

Adding these functions, the main objective was to build a low-cost prototype, which does not pose a entry barrier to this sport. The cost of the project is € 167.68. Seeing the existing alternatives in the market, a low-cost solution has been achieved.

However, several limitations have been found that may result in future versions or improvements of the project. The first is the design and use of PCB boards for the construction of connections to the Arduino clock and navigation keyboard. Others are functions that could be added such as the programming of RFID tags as an internal function of the clock or the inclusion of memory in the chessboard, for the identification by type piece and modification of positions in the consignment.

Bibliografía

1. **ONCE.** *FEDC: Federación Española de Deportes para Ciegos.* [En línea] https://www.fedc.es/deportes/ajedrez.

2. —. Kaissa. Manual rápido de usuario. [En línea] https://www.once.es/cti/biblioteca/Relojes,%20Despertadores%20y%20Medidores%20de%20tiempo/Reloj%20parlante%20de%20ajedrez%20KAISSA/RELOJ_AJEDREZ_KAISSA.pdf.

3. **Luis LLamas.** *EL BUS SPI.* [En línea] https://www.luisllamas.es/arduino-spi/.

4. **Luis Llamas.** *EL BUS I2C.* [En línea] https://www.luisllamas.es/arduino-i2c/.

5. **PerryT.** *Thingiverse.* [En línea] 7 de Enero de 2013. https://www.thingiverse.com/thing:40605.

6. **Wikipedia.** Piezas Staunton. [En línea] https://es.wikipedia.org/wiki/Piezas_Staunton.

7. **Luis Llamas.** *Lector MIFARE RC522.* [En línea] https://www.luisllamas.es/arduino-rfid-mifare-rc522/.

8. **Luis LLamas.** *Teclado matricial.* [En línea] https://www.luisllamas.es/arduino-teclado-matricial/.

9. **Handson Technology.** I2C Serial Interface 20x4 LCD Module. [En línea] https://www.handsontec.com/dataspecs/I2C_2004_LCD.pdf.

10. **DFRobot.** DFPlayer - A Mini MP3 Player SKU:DFR0299 . [En línea] https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299.

11. **Arduino.** Arduino IDE. [En línea] https://www.arduino.cc/en/software.

12. —. Arduino MEGA. [En línea] https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardMega.

13. **Audacity.** Audacity. [En línea] https://www.audacityteam.org.

14. **PowerBroker2.** GitHub. *DFPlayer Mini Fast.* [En línea] https://github.com/PowerBroker2/DFPlayerMini_Fast.

15. **luisllamasbinaburo.** GitHub. *Librería PetriNet.* [En línea] https://github.com/luisllamasbinaburo/Arduino-PetriNet.

16. **AndrewMascolo.** Arduino / GitHub. *Simple HMS Count Up/Down Timer*. [En línea] https://playground.arduino.cc/Main/CountUpDownTimer/.

17. **miguelbalboa.** GitHub. *MFRC522.* [En línea] https://github.com/miguelbalboa/rfid.

18. **Hernández, Luis del Valle.** programarfacil.com. *Lector RFID RC522 control de acceso RFID con Arduino.* [En línea] https://programarfacil.com/blog/arduino-blog/lector-rfid-rc522-con-arduino/ .

Anexo I: Programación

1 Reloj

1.1 Principal y configuración

```
/*
 * Reloj de ajedrez adaptado para jugadores con discapacidad visual
 * Autor: Daniel Lorenzo López
 */

 //Inclusión de librerías
#include <Keypad.h>             //teclado matricial
#include "CountUpDownTimer.h"   //Temporizador
#include <DebounceFilterLib.h>  //Filtros antirrebote
#include <PetriNetLib.h>        //Red de Petri
#include <Wire.h>               //Conexiones I2C
#include <LiquidCrystal_I2C.h>  //Pantalla LCD I2C

//Constructor pantalla LCD I2C
LiquidCrystal_I2C LCDBlancas(0x27,20,4);
LiquidCrystal_I2C LCDNegras(0x23,20,4);

//Constructor Filtros antirebote
DebounceFilter filtroBlancas, filtroNegras, filtroPausa, filtroEnter;
DebounceFilter pulsadorBlancasAudioBlancas, pulsadorNegrasAudioBlancas;
DebounceFilter pulsadorBlancasAudioNegras, pulsadorNegrasAudioNegras;
DebounceFilter filtroAudioBlancas, filtroAudioNegras;

//Variables Pausa
const int pinPausa = 34;
unsigned long tPausaInicio;
bool pausado = true;

//Variables Temporizador
const int TurnoBlancas = 22;
const int TurnoNegras = 24;
bool estadoB, estadoN;
bool reproducirSinPulsador = false;

//Variables Selección tiempo
// Valores por defecto
int selecHor = 1;
int selecMin = 30;
int selecSeg = 0;
int incremento = 30;

//Tiempos
int HoraB, MinutoB, SegundoB, HoraN, MinutoN, SegundoN;
//Variables temporizador blancas
char tiempoBlancas[6];
char tiempoEditadoBlancas[12];
//Variables temporizador negras
char tiempoNegras[6];
char tiempoEditadoNegras[12];

//Temporizadores para los jugadores
CountUpDownTimer TempBlancas(DOWN), TempNegras(DOWN);
```

```
//Audio
#include <SoftwareSerial.h>     //Librería Comunicación serie
#include <DFPlayerMini_Fast.h>  //Librería Módulo MP3

//Constructores Módulos MP3
SoftwareSerial serialAudioBlancas(3,2); //RX, TX
SoftwareSerial serialAudioNegras(4,5); //RX, TX
DFPlayerMini_Fast vozBlancas;
DFPlayerMini_Fast vozNegras;

//pines pulsadores audio y señales módulo reproduciendo
const int pinpulsadorNegrasAudioBlancas = 26;
const int pinpulsadorBlancasAudioBlancas = 28;
const int pinpulsadorNegrasAudioNegras = 30;
const int pinpulsadorBlancasAudioNegras = 32;
const uint8_t pinOcupadoBlancas = A8;
const uint8_t pinOcupadoNegras = A9;

//Carpetas Audios
const int Numeros[4] = {1, 2, 3};
const int Tiempos[4] = {11, 12, 13};
const int carpetaMenu[3] = {21, 22, 23};
const int carpetaIdioma = 31;
const int carpetaLetras = 41;

// Variables modificables en Menú
int volumenBlancas = 20, volumenNegras = 20;
int idiomaBlancas = 0, idiomaNegras = 0;

//Idiomas
String Idioma[3] = { "ES", "EN", "DE" };

/*
 *  Estados de Reproducción de Audio
 *  0 => No se está reproduciendo
 *  1 => Reproduciendo Tiempo de Blancas
 *  2 => Reproduciendo Tiempo de Negras
 */
const int noReproducir = 0, reproducirBlancas = 1, reproducirNegras = 2;
int estadoAudioBlancas, estadoAudioNegras;
int indicadorTiempoBlancas = 0, indicadorTiempoNegras = 0;
bool reproduciendoBlancas = false, reproduciendoNegras = false;

int indicadorNotacion = 0;
bool reproduciendoNotacion = false;

//Programación RFID
/**
 * Pines para RFID MFRC522
 * ------------------------
 *             Arduino
 *             Mega
 * Signal      Pin
 * ------------------------
 * RST/Reset   9
 * SPI SDA(SS) 53
 * SPI MOSI    51
 * SPI MISO    50
 * SPI SCK     52
 */
```

```
bool programado = false;     //variable de reloj programado
#include <SPI.h>             //Librería comunicación SPI
#include <MFRC522.h>         //Librería RFID

//pines RFID
#define RST_PIN         9
#define SS_PIN          53

MFRC522 mfrc522(SS_PIN, RST_PIN);  // Constructor lector de etiquetas
MFRC522::MIFARE_Key key;


//Menú
PetriNet petriMenu(12, 27); //Número de Estados y Transiciones

char teclasMenu[2][2] = {{'<', '+'},    //teclado menú
                         {'-', '>'}};

// pines teclas de menu
byte pinesColumnas[2] = {27, 23};
byte pinesFilas[2] = {29, 25};
int pinEnter = 31;

//Constructor teclado menú
Keypad keypad = Keypad(makeKeymap(teclasMenu), pinesFilas , pinesColumnas,
2, 2);

//Opciones de botones de navegación
enum tipoEntrada
{
  Siguiente = 0,
  Atras = 1,
  Mas = 2,
  Menos = 3,
  Enter = 4,
  Unknown = -1
};

tipoEntrada entrada;

//Audios de acciones de menú
enum Menu
{
  jugarPartida = 1,
  partidaDetenida = 2,
  partidaIniciada = 3,
  programarReloj = 4,
  relojProgramado = 5,
  editarTiempos = 6,
  tiempoEditado = 7,
  volumen = 8,
  volBlancas = 9,
  volNegras = 10,
  idioma = 11,
  idiBlancas = 12,
  idiNegras = 13
};
```

```
int indiceEditor = 0;  //indicador parpadeo en edición de tiempos

//Funciones
#include "BigChar.h"
#include "RFID.h"
#include "Audio.h"        //Funciones de reproducción en audio de tiempos
#include "Temporizador.h" //Funciones de temporizadores
#include "editarTiempos.h"
#include "tablero.h"    //Comunicación con tablero
#include "Menu.h"

void setup() {
  Serial.begin(115200);
  Serial3.begin(9600);

  //Definición de pines de entrada - salida
  pinMode(TurnoBlancas, INPUT);
  pinMode(TurnoNegras, INPUT);
  pinMode(INPUT, pinPausa);
  pinMode(INPUT, pinEnter);

  //Intervalo de tiempo para filtro antirebote
  filtroBlancas.SetInterval(50);
  filtroNegras.SetInterval(50);
  filtroPausa.SetInterval(50);
  filtroEnter.SetInterval(50);
  pulsadorBlancasAudioBlancas.SetInterval(50);
  pulsadorNegrasAudioBlancas.SetInterval(50);
  pulsadorBlancasAudioNegras.SetInterval(50);
  pulsadorNegrasAudioNegras.SetInterval(50);
  filtroAudioBlancas.SetInterval(50);
  filtroAudioNegras.SetInterval(50);

  //Configuración de Temporizadores
  TempBlancas.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste tiempo
Blancas
  TempNegras.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste tiempo Negras

  TempBlancas.StartTimer();
  TempNegras.StartTimer();

  TempBlancas.PauseTimer();
  TempNegras.PauseTimer();

  HoraB = HoraN = selecHor;
  MinutoB = MinutoN = selecMin;
  SegundoB = SegundoN = selecSeg;

  //Configuración de Audio
  //pines pulsadores de tiempo
  pinMode(pinpulsadorNegrasAudioBlancas, INPUT);
  pinMode(pinpulsadorBlancasAudioBlancas, INPUT);
  pinMode(pinpulsadorNegrasAudioNegras, INPUT);
  pinMode(pinpulsadorBlancasAudioNegras, INPUT);
```

```
//Módulos MP3
  serialAudioBlancas.begin(9600);
  serialAudioNegras.begin(9600);

  vozNegras.begin(serialAudioNegras);
  vozNegras.volume(volumenNegras);
  delay(20);
  vozBlancas.begin(serialAudioBlancas);
  vozBlancas.volume(volumenBlancas);
  delay(20);

  estadoAudioBlancas = noReproducir;
  estadoAudioNegras = noReproducir;

  //configuración lector de etiquetas
  SPI.begin();        // Init SPI bus
  mfrc522.PCD_Init(); // Init MFRC522 card

  // Preparar the clave por defecto del lector
  for (byte i = 0; i < 6; i++) {
      key.keyByte[i] = 0xFF;
  }

  //Configuración Red de Petri para Menú
  #include "ConfigPetri.h"

  //Configuración LCDs
  LCDBlancas.init();
  LCDBlancas.backlight();
  LCDNegras.init();
  LCDNegras.backlight();

  //Creación de caracteres especiales para texto grande
  LCDBlancas.createChar(1, char1);
  LCDBlancas.createChar(2, char2);
  LCDBlancas.createChar(3, char3);
  LCDBlancas.createChar(4, char4);
  LCDBlancas.createChar(5, char5);
  LCDBlancas.createChar(6, char6);
  LCDBlancas.createChar(7, char7);
  LCDBlancas.createChar(8, char8);

  LCDNegras.createChar(1, char1);
  LCDNegras.createChar(2, char2);
  LCDNegras.createChar(3, char3);
  LCDNegras.createChar(4, char4);
  LCDNegras.createChar(5, char5);
  LCDNegras.createChar(6, char6);
  LCDNegras.createChar(7, char7);
  LCDNegras.createChar(8, char8);

  //Pantalla de encendido
  LCDBlancas.clear();
  LCDBlancas.setCursor(2,0);
  LCDBlancas.print("Reloj de ajedrez");
  LCDBlancas.setCursor(3,1);
  LCDBlancas.print("adaptado para");
  LCDBlancas.setCursor(0,2);
  LCDBlancas.print("discapacidad visual");
```

```
  LCDNegras.clear();
  LCDNegras.setCursor(2,0);
  LCDNegras.print("Reloj de ajedrez");
  LCDNegras.setCursor(3,1);
  LCDNegras.print("adaptado para");
  LCDNegras.setCursor(0,2);
  LCDNegras.print("discapacidad visual");

  delay(2000);
  LCDBlancas.clear();
  LCDNegras.clear();

}

void loop() {

  entrada = static_cast<tipoEntrada>(leerEntradas()); //lectura de entradas
  petriMenu.Update();   //Actualización de red de Petri
  AccionesEstados();

}
```

1.2 Configuración de la Red de Petri

```
// Definición de la red de Petri

  // Entradas y salidas de las transiciones
  //Se indica el estado de entrada a la transición y el estado de destino

  //Funciones del Menú
  static uint8_t entradas0[] = { 0 };  static uint8_t salidas0[] = { 1 };
  static uint8_t entradas1[] = { 1 };  static uint8_t salidas1[] = { 0 };
  static uint8_t entradas2[] = { 1 };  static uint8_t salidas2[] = { 2 };
  static uint8_t entradas3[] = { 2 };  static uint8_t salidas3[] = { 1 };
  static uint8_t entradas4[] = { 2 };  static uint8_t salidas4[] = { 3 };
  static uint8_t entradas5[] = { 3 };  static uint8_t salidas5[] = { 2 };
  static uint8_t entradas6[] = { 3 };  static uint8_t salidas6[] = { 4 };
  static uint8_t entradas7[] = { 4 };  static uint8_t salidas7[] = { 3 };
  static uint8_t entradas8[] = { 4 };  static uint8_t salidas8[] = { 5 };
  static uint8_t entradas9[] = { 5 };  static uint8_t salidas9[] = { 4 };
  static uint8_t entradas10[] = { 1 };  static uint8_t salidas10[] = { 5 };
  static uint8_t entradas11[] = { 5 };  static uint8_t salidas11[] = { 1 };

  // E/S Configuración Idioma
  static uint8_t entradas12[] = { 5 };  static uint8_t salidas12[] = { 6 };
  static uint8_t entradas13[] = { 6 };  static uint8_t salidas13[] = { 7 };
  static uint8_t entradas14[] = { 7 };  static uint8_t salidas14[] = { 6 };
  static uint8_t entradas15[] = { 6 };  static uint8_t salidas15[] = { 5 };
  static uint8_t entradas16[] = { 7 };  static uint8_t salidas16[] = { 5 };

  // E/S Configuración Volumen
  static uint8_t entradas17[] = { 4 };  static uint8_t salidas17[] = { 8 };
  static uint8_t entradas18[] = { 8 };  static uint8_t salidas18[] = { 9 };
  static uint8_t entradas19[] = { 9 };  static uint8_t salidas19[] = { 8 };
  static uint8_t entradas20[] = { 8 };  static uint8_t salidas20[] = { 4 };
  static uint8_t entradas21[] = { 9 };  static uint8_t salidas21[] = { 4 };

  //E/S programación de Reloj
  static uint8_t entradas22[] = { 2 };  static uint8_t salidas22[] = { 10
};
  static uint8_t entradas23[] = { 10 };  static uint8_t salidas23[] = { 2
};
  static uint8_t entradas24[] = { 10 };  static uint8_t salidas24[] = { 0
};

  //E/S edición de Tiempos
  static uint8_t entradas25[] = { 3 };  static uint8_t salidas25[] = { 11
};
  static uint8_t entradas26[] = { 11 };  static uint8_t salidas26[] = { 3
};

  // Configuración de las Transiciones
  /*Se programa indicando el número de la transición actual,
  se le pasa las entradas y salidas y el número de ellas,
  se indica la acción de activación de la transición entre estados
  y se programan acciones que se ejecutan una vez al realizarse la
transición
  */
```

```
// Menús Principales

  petriMenu.SetTransition(0, entradas0, 1, salidas0, 1,
    []() -> bool {return entrada == tipoEntrada::Enter && pausado; },
    [](){//Serial.println("Transición 0-1");
          LCDBlancas.clear(); LCDNegras.clear();
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::jugarPartida);
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::jugarPartida);
          });

  petriMenu.SetTransition(1, entradas1, 1, salidas1, 1,
    []() -> bool {return entrada == tipoEntrada::Enter && pausado; },
    [](){//Serial.println("Transición 1-0");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::jugarPartida);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::jugarPartida);
          imprimirTiempo();
          delay(1500);
          reproducirSinPulsador = true;
          });

  petriMenu.SetTransition(2, entradas2, 1, salidas2, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente; },
    [](){//Serial.println("Transición 1-2");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::programarReloj);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::programarReloj);});

  petriMenu.SetTransition(3, entradas3, 1, salidas3, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    [](){//Serial.println("Transición 2-1");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::jugarPartida);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::jugarPartida);});

  petriMenu.SetTransition(4, entradas4, 1, salidas4, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente; },
    [](){//Serial.println("Transición 2-3");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::editarTiempos);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::editarTiempos);});

  petriMenu.SetTransition(5, entradas5, 1, salidas5, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    [](){//Serial.println("Transición 3-2");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::programarReloj);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::programarReloj);});
```

```
  petriMenu.SetTransition(6, entradas6, 1, salidas6, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente; },
    [](){//Serial.println("Transición 3-4");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::volumen);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volumen);});

  petriMenu.SetTransition(7, entradas7, 1, salidas7, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    [](){///Serial.println("Transición 4-3");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::editarTiempos);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::editarTiempos);});

  petriMenu.SetTransition(8, entradas8, 1, salidas8, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente; },
    [](){//Serial.println("Transición 4-5");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::idioma);
          vozNegras.playFolder(carpetaMenu[idiomaNegras], Menu::idioma);});

  petriMenu.SetTransition(9, entradas9, 1, salidas9, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    [](){//Serial.println("Transición 5-4");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::volumen);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volumen);});

  petriMenu.SetTransition(10, entradas10, 1, salidas10, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    [](){//Serial.println("Transición 1-5");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::idioma);
          vozNegras.playFolder(carpetaMenu[idiomaNegras], Menu::idioma);});

  petriMenu.SetTransition(11, entradas11, 1, salidas11, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente; },
    [](){//Serial.println("Transición 5-1");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::jugarPartida);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::jugarPartida);});

// Menú Idioma

  petriMenu.SetTransition(12, entradas12, 1, salidas12, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    [](){//Serial.println("Transición 5-6: Idioma ");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::idiBlancas);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::idiBlancas);});
```

```
  petriMenu.SetTransition(13, entradas13, 1, salidas13, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente ||
        entrada == tipoEntrada::Atras; },
    []() {//Serial.println("Transición 6-7");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::idiNegras);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::idiNegras);});

  petriMenu.SetTransition(14, entradas14, 1, salidas14, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente ||
        entrada == tipoEntrada::Atras; },
    []() {//Serial.println("Transición 7-6");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::idiBlancas);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::idiBlancas);});

  petriMenu.SetTransition(15, entradas15, 1, salidas15, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición 6 a 5: Idioma Blancas config");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::idioma);
          vozNegras.playFolder(carpetaMenu[idiomaNegras], Menu::idioma);});

  petriMenu.SetTransition(16, entradas16, 1, salidas16, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición de 7 a 5: Idioma Negras config");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::idioma);
          vozNegras.playFolder(carpetaMenu[idiomaNegras], Menu::idioma);});

// Menú Volumen

  petriMenu.SetTransition(17, entradas17, 1, salidas17, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición 4-8: Idioma ");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::volBlancas);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volBlancas);});

  petriMenu.SetTransition(18, entradas18, 1, salidas18, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente ||
              entrada == tipoEntrada::Atras; },
    []() {//Serial.println("Transición 8-9");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::volNegras);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volNegras);});
```

```
  petriMenu.SetTransition(19, entradas19, 1, salidas19, 1,
    []() -> bool {return entrada == tipoEntrada::Siguiente ||
                         entrada == tipoEntrada::Atras; },
    []() {//Serial.println("Transición 9-8");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::volBlancas);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volBlancas);});

  petriMenu.SetTransition(20, entradas20, 1, salidas20, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición 8 a 4: Volumen Blancas config");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::volumen);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volumen);});

  petriMenu.SetTransition(21, entradas21, 1, salidas21, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición de 9 a 4: Volumen Negras config");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas], Menu::volumen);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::volumen);});

//Programación Reloj

    petriMenu.SetTransition(22, entradas22, 1, salidas22, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición de 2 a 10: Programar Reloj");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::programarReloj);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::programarReloj);});

    petriMenu.SetTransition(23, entradas23, 1, salidas23, 1,
    []() -> bool {return entrada == tipoEntrada::Atras; },
    []() {//Serial.println("Transición de 10 a 2");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::programarReloj);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::programarReloj);});

    petriMenu.SetTransition(24, entradas24, 1, salidas24, 1,
    []() -> bool {return programado; },
    []() {//Serial.println("Transición de 10 a 0");
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::relojProgramado);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::relojProgramado);
```

```
          TempBlancas.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste
tiempo Blancas
          TempNegras.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste
tiempo Negras
          imprimirTiempo();
          delay(1600);
          reproducirSinPulsador = true;
          });

//Editar Tiempos

    petriMenu.SetTransition(25, entradas25, 1, salidas25, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición de 3 a 11");
          LCDBlancas.clear(); LCDNegras.clear();
          imprimirTiemposEditados();
          reproducirSinPulsador = true;
          });

    petriMenu.SetTransition(26, entradas26, 1, salidas26, 1,
    []() -> bool {return entrada == tipoEntrada::Enter; },
    []() {//Serial.println("Transición de 11 a 3");
          LCDBlancas.noBlink(); LCDNegras.noBlink();
          LCDBlancas.clear(); LCDNegras.clear();
          vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::tiempoEditado);
          vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::tiempoEditado);});

  // Marcado inicial
  petriMenu.SetMarkup(1, 1); //Estado 1 Inicial: Menú Jugar Partida
```

1.3 Menú

```
//Funciones de menú de navegación o Estados de la Red de Petri
/*Lecturas de entradas para la navegación del menú Teclado de navegación*/

int leerEntradas(){

  tipoEntrada entradaActual = tipoEntrada::Unknown; //variable teclado

    char key = keypad.getKey(); //lectura de cruz del teclado matricial

    switch (key) {
        case '>':
            entradaActual = tipoEntrada::Siguiente;   //Siguiente pantalla
/ mover cursor
            break;
        case '<':
            entradaActual = tipoEntrada::Atras;       //Pantalla anterior /
mover cursor
            break;
        case '+':
            entradaActual = tipoEntrada::Mas;         //Aumentar valor
            break;
        case '-':
            entradaActual = tipoEntrada::Menos;       //Decrementar valor
            break;
        default:
            break;
    }

//Pulsador Enter
  filtroEnter.AddValue(digitalRead(pinEnter));     //lectura pulsador enter
  if(filtroEnter.IsRising()){
    entradaActual = tipoEntrada::Enter;
  }

  return entradaActual;   //devolver pulsador activo
}

/*Función de acciones de estados de red de petri
  -Imprime las pantallas del menú
  -Acciones en los estados función del reloj:
      * Jugada
      * Programación
      * Edición
      * Volumen
      * Idioma
*/
```

```
void AccionesEstados(){

  if(petriMenu.GetMarkup(0)){   //Partida

    leerTurnos(); //Actualización de turno y cambio de tiempo temporizador
    leerPausa();            //Paralización de la partida
    if((pausado == false) && (estadoAudioBlancas == noReproducir)
          && (estadoAudioNegras == noReproducir))
          recibirDatosTablero();              //Notación de jugada del
tablero en audio
    reproduceTiempos();             //Tiempos restantes en audio
    comprobarGanador();             //Fin de la partida por fin de tiempo
    actualizarTiempo();             //Imprimir en pantalla el tiempo si se
modifica

    programado = false;
  }else if(petriMenu.GetMarkup(1)){   //Jugar Partida

    textoGrande("JUGAR", 2, 0, LCDBlancas);
    textoGrande("PARTID", 0, 2, LCDBlancas);
    textoGrande("JUGAR", 2, 0, LCDNegras);
    textoGrande("PARTID", 0, 2, LCDNegras);

  }else if(petriMenu.GetMarkup(2)){   //Programar Reloj

    textoGrande("PROGRA", 0, 0, LCDBlancas);
    textoGrande("RELOJ", 3, 2, LCDBlancas);
    textoGrande("PROGRA", 0, 0, LCDNegras);
    textoGrande("RELOJ", 3, 2, LCDNegras);

  }else if(petriMenu.GetMarkup(3)){   //Editar Tiempos

    textoGrande("EDITAR", 1, 0, LCDBlancas);
    textoGrande("TIEMPO", 0, 2, LCDBlancas);
    textoGrande("EDITAR", 1, 0, LCDNegras);
    textoGrande("TIEMPO", 0, 2, LCDNegras);

  }else if(petriMenu.GetMarkup(4)){   //Volumen

    textoGrande("VOLUM", 1, 1, LCDBlancas);
    textoGrande("VOLUM", 1, 1, LCDNegras);

  }else if(petriMenu.GetMarkup(5)){   //Idioma

    textoGrande("IDIOMA", 0, 1, LCDBlancas);
    textoGrande("IDIOMA", 0, 1, LCDNegras);

  }else if(petriMenu.GetMarkup(6)){ // Idioma Blancas

    //Imprimir pantalla
    textoGrande("IDIOMA", 0, 0, LCDBlancas);
    textoGrande("BLN:", 0, 2, LCDBlancas);
    textoGrande("IDIOMA", 0, 0, LCDNegras);
    textoGrande("BLN:", 0, 2, LCDNegras);
```

```
//Imprimir el idioma actual
    char idioma[3];
    Idioma[idiomaBlancas].toCharArray(idioma, 3); //Conversión String a
Vector de Char
    textoGrande(idioma, 12, 2, LCDBlancas);
    textoGrande(idioma, 12, 2, LCDNegras);


    if(entrada == tipoEntrada::Mas){
      LCDBlancas.clear(); LCDNegras.clear();
      if(idiomaBlancas < 2){ idiomaBlancas++;       // Siguiente idioma
      }else if(idiomaBlancas == 2){ idiomaBlancas = 0;
      }
      vozBlancas.playFolder(carpetaIdioma, idiomaBlancas+1);  //Reproduce
idioma seleccionado

    }else if(entrada == tipoEntrada::Menos){
      LCDBlancas.clear(); LCDNegras.clear();
      if(idiomaBlancas > 0){idiomaBlancas--;       // Anterior idioma
      }else if(idiomaBlancas == 0){
        idiomaBlancas = 2;
      }
      vozBlancas.playFolder(carpetaIdioma, idiomaBlancas+1);  //Reproduce
idioma seleccionado
    }
  }else if(petriMenu.GetMarkup(7)){ //Idioma Negras

    textoGrande("IDIOMA", 0, 0, LCDBlancas);
    textoGrande("NEG: ", 0, 2, LCDBlancas);
    textoGrande("IDIOMA", 0, 0, LCDNegras);
    textoGrande("NEG: ", 0, 2, LCDNegras);

    //Imprimir el idioma actual
    char idioma[3];
    Idioma[idiomaNegras].toCharArray(idioma, 3);
    textoGrande(idioma, 12, 2, LCDBlancas);
    textoGrande(idioma, 12, 2, LCDNegras);

    if(entrada == tipoEntrada::Mas){
      LCDBlancas.clear(); LCDNegras.clear();
      if(idiomaNegras < 2){idiomaNegras++;       // Siguiente idioma
      }else if(idiomaNegras == 2){ idiomaNegras = 0;
      }
      vozNegras.playFolder(carpetaIdioma, idiomaNegras+1);

    }else if(entrada == tipoEntrada::Menos){
      LCDBlancas.clear(); LCDNegras.clear();
      if(idiomaNegras > 0){ idiomaNegras--;       // Anterior idioma
      }else if(idiomaNegras == 0){ idiomaNegras = 2;
      }
      vozNegras.playFolder(carpetaIdioma, idiomaNegras+1);
    }
  }else if(petriMenu.GetMarkup(8)){ // Volumen Blancas

   textoGrande("VOLUM", 0, 0, LCDBlancas);
   textoGrande("BLN:", 0, 2, LCDBlancas);
   textoGrande("VOLUM", 0, 0, LCDNegras);
   textoGrande("BLN:", 0, 2, LCDNegras);
```

```
 //Imprimir el volumen actual
   char volBlancas[2];
   sprintf(volBlancas, "%01d", volumenBlancas);
   textoGrande(volBlancas, 12, 2, LCDBlancas);
   textoGrande(volBlancas, 12, 2, LCDNegras);

    if(entrada == tipoEntrada::Mas && volumenBlancas < 30){ //Máximo
volumen: 30
      LCDBlancas.clear(); LCDNegras.clear();
      volumenBlancas++;                       //Añadir volumen
      vozBlancas.volume(volumenBlancas);      //Configurar volumen
      delay(20);
      //Reproduce para escuchar el audio al volumen seleccionado
      vozBlancas.playFolder(Numeros[idiomaBlancas], volumenBlancas);

    }else if(entrada == tipoEntrada::Menos && volumenBlancas > 0){ //Mínimo
      LCDBlancas.clear(); LCDNegras.clear();
      volumenBlancas--;                       //Reducir Volumen
      vozBlancas.volume(volumenBlancas);      //Configurar volumen
      delay(20);
      //Reproduce para escuchar el audio al volumen seleccionado
      vozBlancas.playFolder(Numeros[idiomaBlancas], volumenBlancas);
    }

  }else if(petriMenu.GetMarkup(9)){ // Volumen Negras

   textoGrande("VOLUM", 0, 0, LCDBlancas);
   textoGrande("NEG:", 0, 2, LCDBlancas);
   textoGrande("VOLUM", 0, 0, LCDNegras);
   textoGrande("NEG:", 0, 2, LCDNegras);

   //Imprimir el volumen actual
   char volNegras[2];
   sprintf(volNegras, "%01d", volumenNegras);
   textoGrande(volNegras, 12, 2, LCDBlancas);
   textoGrande(volNegras, 12, 2, LCDNegras);

    if(entrada == tipoEntrada::Mas && volumenNegras < 30){ //Máximo
volumen: 30
      LCDBlancas.clear(); LCDNegras.clear();
      volumenNegras++;                        //Añadir volumen
      vozNegras.volume(volumenNegras);      //Configurar volumen
      delay(20);
      //Reproduce para escuchar el audio al volumen seleccionado
      vozNegras.playFolder(Numeros[idiomaNegras], volumenNegras);

    }else if(entrada == tipoEntrada::Menos && volumenNegras > 0){
      LCDBlancas.clear(); LCDNegras.clear();
      volumenNegras--;                         //Reducir Volumen
      vozNegras.volume(volumenNegras);      //Configurar volumen
      delay(20);
      //Reproduce para escuchar el audio al volumen seleccionado
      vozNegras.playFolder(Numeros[idiomaNegras], volumenNegras);
    }
```

```
 }else if(petriMenu.GetMarkup(10)){ // Programar Reloj

    //pantalla lectura RFID
    LCDBlancas.setCursor(0,1);
    LCDBlancas.print("Acerque la etiqueta");
    LCDBlancas.setCursor(6,2);
    LCDBlancas.print("al lector");

    LCDNegras.setCursor(0,1);
    LCDNegras.print("Acerque la etiqueta");
    LCDNegras.setCursor(6,2);
    LCDNegras.print("al lector");

    leerEtiqueta(mfrc522, key);   //función lectura RFID

  }else if(petriMenu.GetMarkup(11)){ // Editar Tiempos

    edicionTiempos();     //función de edición tiempo
    reproduceTiempos();   //Tiempo en audio

  }

}
```

1.4 Temporizador

```
//Funciones de temporizador para partida
/*Leer Turnos
  Leer Pausa
  Imprimir tiempo
  Actualizar tiempo
  Comprobar ganador*/

  /*Se lee el interruptor de jugadas,
  dependiendo de la posición corresponde al turno activo de 1 jugador
  En el cambio de turno, se incrementa un cierto tiempo al jugador que ha
terminado*/
void leerTurnos(){

  //Actualización de temporizador
  TempBlancas.Timer();
  TempNegras.Timer();

  //Filtro de Interruptor Balancín
  filtroBlancas.AddValue(digitalRead(TurnoBlancas));
  estadoB = filtroBlancas.IsRising();
  filtroNegras.AddValue(digitalRead(TurnoNegras));
  estadoN = filtroNegras.IsRising();

  //Turno Blancas
  if (estadoB && (pausado == false)){
    TempNegras.PauseTimer();    //Pausar Temporizador Negras
    TempNegras.SetTimer(TempNegras.ShowTotalSeconds() + incremento);
//añadir incremento
    TempBlancas.ResumeTimer();  //Reanudar Temporizador Blancas
  }//fin turno blancas

  //Turno Negras
  if (estadoN  && (pausado == false)){
    TempBlancas.PauseTimer();   //Pausar Temporizador Blancas
    TempBlancas.SetTimer(TempBlancas.ShowTotalSeconds() + incremento);
//añadir incremento
    TempNegras.ResumeTimer();   //Reanudar Temporizador Negras
  }//fin turno negras

}

/*Lectura del pulsador de pausa

Para activar/desactivar la pausa se mantiene presionado medio segundo
Se reproduce audio en el cambio de estado*/

void leerPausa(){

  //Filtro para pulsador de pausa
  filtroPausa.AddValue(digitalRead(pinPausa));
  if(filtroPausa.IsRising()){
    tPausaInicio = millis();
  }
```

```
  //Cuando se suelta el pulsador tras medio segundo
  if(((millis() - tPausaInicio) > 500) && filtroPausa.IsFalling()){

     if(pausado == false){    //Si la partida no está pausada
        TempBlancas.PauseTimer();
        TempNegras.PauseTimer();
        //Audio partida detenida
        vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::partidaDetenida);
        vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::partidaDetenida);
        pausado = true;       //Variable de pausa

    }else{  //Reinicio de la pausa
      if(filtroBlancas.GetState()){       //Temporizador Blancas activo
        TempBlancas.StartTimer();
        TempNegras.PauseTimer();

      }else if(filtroNegras.GetState()){  //Temporizador Negras activo
        TempBlancas.PauseTimer();
        TempNegras.StartTimer();
      }

      //Audio partida iniciada
      vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::partidaIniciada);
      vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::partidaIniciada);
      pausado = false;
      tPausaInicio = millis();    //Actualización tiempo pulsación pausa
    }
  }

}

/*Imprimir en pantallas el tiempo restante de la partida

Se muestra el tiempo restante de forma que no hayan 0 a la izquierda
El tiempo superior es el tiempo propio
El tiempo inferior es el tiempo del oponente*/
void imprimirTiempo(){

      //Actualización del tiempo
      HoraB = TempBlancas.ShowHours();
      MinutoB = TempBlancas.ShowMinutes();
      SegundoB = TempBlancas.ShowSeconds();

      HoraN = TempNegras.ShowHours();
      MinutoN = TempNegras.ShowMinutes();
      SegundoN = TempNegras.ShowSeconds();

      //Borrado de las pantallas si hay modificación por debajo de 1 hora o
1 minuto
      //Para adecuar la visualización
      if(TempBlancas.ShowTotalSeconds() == 3599 ||
TempBlancas.ShowTotalSeconds() == 59 ||
         TempNegras.ShowTotalSeconds() == 3599 ||
TempNegras.ShowTotalSeconds() == 59){
        LCDBlancas.clear();
        LCDNegras.clear();
      }
```

```
      /*Mostrar el tiempo restante de Blancas*/
      if(HoraB > 0){    //Más de 1 hora
        sprintf(tiempoBlancas, "%01d:%02d:%02d", HoraB, MinutoB, SegundoB);
      }else if(HoraB == 0 && MinutoB > 0 && SegundoB <= 59){      //Más de
1 minuto
        sprintf(tiempoBlancas, "%02d:%02d", MinutoB, SegundoB);
      }else if(HoraB == 0 && MinutoB == 0 && SegundoB >
0){       //Segundos
        sprintf(tiempoBlancas, "%02d", SegundoB);
      }
      textoGrande(tiempoBlancas,0,0, LCDBlancas);   //Línea superior
      textoGrande(tiempoBlancas,0,2, LCDNegras);    //Línea inferior

      /*Mostrar el tiempo restante de Negras*/
      if(HoraN > 0){    //Más de 1 hora
        sprintf(tiempoNegras, "%01d:%02d:%02d", HoraN, MinutoN, SegundoN);
      }else if(HoraN == 0 && MinutoN > 0 && SegundoN <= 59){      //Más de
1 minuto
        sprintf(tiempoNegras, "%02d:%02d", MinutoN, SegundoN);
      }else if(HoraN == 0 && MinutoN == 0 && SegundoN >
0){       //Segundos
        sprintf(tiempoNegras, "%02d", SegundoN);
      }
      textoGrande(tiempoNegras, 0, 2, LCDBlancas);  //Línea inferior
      textoGrande(tiempoNegras, 0, 0, LCDNegras);   //Línea superior

      /*Mostrar en un lateral la indicación del tiempo e incremento de
jugada
      B -> Tiempo Blancas
      N -> Tiempo Negras*/
      LCDBlancas.setCursor(18,0);
      LCDBlancas.print("B");
      LCDBlancas.setCursor(18,3);
      LCDBlancas.print("N");
      LCDNegras.setCursor(18,0);
      LCDNegras.print("N");
      LCDNegras.setCursor(18,3);
      LCDNegras.print("B");
      LCDNegras.setCursor(17,1);
      LCDNegras.print("+");
      LCDNegras.print(incremento);
      LCDBlancas.setCursor(17,1);
      LCDBlancas.print("+");
      LCDBlancas.print(incremento);

} //fin imprimirTiempo
```

```
/*Función para actualizar el tiempo en pantalla únicamente si se ha
modificado
  -Tiempo de los temporizadores cambia
  -Cambio de turno
  -Temporizador llega a 0

  Si se llega a tiempo 0, se reinicia a los valores de programación
determinados y se paran
*/
void actualizarTiempo(){
    //Mostrar el tiempo
  if (TempBlancas.TimeHasChanged() || TempNegras.TimeHasChanged() ||
filtroBlancas.IsRising() || filtroNegras.IsRising()){
    if(!(TempBlancas.TimeCheck(0,0,0,0) || TempNegras.TimeCheck(0,0,0,0))){

      imprimirTiempo();

    }else{  //Tiempo llega a 0
      TempBlancas.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste tiempo
Blancas
      TempNegras.SetTimer(0,selecHor,selecMin,selecSeg); //Ajuste tiempo
Negras
      TempBlancas.StopTimer();
      TempNegras.StopTimer();
    }
  }

}//end MostrarTiempo

/*Función para la comprobación de ganador por tiempo restante*/
void comprobarGanador(){

  //Tiempo finalizado por algunos de los jugadores
    if ((TempBlancas.TimeCheck(0,0,0,0) || TempNegras.TimeCheck(0,0,0,0))
){

        if (TempBlancas.TimeCheck(0,0,0,0)){      //Blancas finalizan /
Ganan negras
          /*
          Serial.println("Partida finalizada");
          Serial.println("Ganador: Blancas");
          */
          LCDBlancas.clear();
          LCDNegras.clear();

          textoGrande("GANA",3,0, LCDBlancas);
          textoGrande("NEGRO",2,2, LCDBlancas);
          textoGrande("GANA",3,0, LCDNegras);
          textoGrande("NEGRO",2,2, LCDNegras);
        }
```

```
        if (TempNegras.TimeCheck(0,0,0,0)){       //Negras finalizan /
Ganan blancas
          /*
          Serial.println("Partida finalizada");
          Serial.println("Ganador: Negras");
          */
          LCDBlancas.clear();
          LCDNegras.clear();

          textoGrande("GANA",3,0, LCDBlancas);
          textoGrande("BLANCO",0,2, LCDBlancas);
          textoGrande("GANA",3,0, LCDNegras);
          textoGrande("BLANCO",0,2, LCDNegras);
        }

        pausado = true;     // juego pausado

        //Reproducción de audio partida detenida
        vozBlancas.playFolder(carpetaMenu[idiomaBlancas],
Menu::partidaDetenida);
        vozNegras.playFolder(carpetaMenu[idiomaNegras],
Menu::partidaDetenida);
  }
}
```

1.5 Caracteres grandes en pantalla

```
/*Funciones de texto grande en pantalla

Se definen los caracteres especiales que componen los caracteres grandes
 -Números de 0 a 9
 -Letras necesarias

 Se convierte variable de texto y se imprime en tamaño grande por pantalla
 colocada en fila y columna elegida*/

// Definicion de caracteres simples
byte char1[8] = { 0x1F, 0x1F, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x00 };//char 1
byte char2[8] = { 0x18, 0x1C, 0x1E, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F };//char 2
byte char3[8] = { 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x0F, 0x07, 0x03 };//char 3
byte char4[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x1F, 0x1F };//char 4
byte char5[8] = { 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1E, 0x1C, 0x18 };//char 5
byte char6[8] = { 0x1F, 0x1F, 0x1F, 0x00, 0x00, 0x00, 0x1F, 0x1F };//char 6
byte char7[8] = { 0x1F, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x1F, 0x1F };//char 7
byte char8[8] = { 0x03, 0x07, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F };//char 8

//Creación de dígitos y letras grandes

int big0(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(1);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(4);
  lcd.write(5);
  return 3;
}

int big1(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(1);
  lcd.write(2);
  lcd.write(32);
  lcd.setCursor(col, fil+1);
  lcd.write(4);
  lcd.write(0XFF);
  lcd.write(4);
  return 3;
}

int big2(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(6);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0XFF);
  lcd.write(7);
  lcd.write(7);
  return 3;
}
```

```
int big3(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(1);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(4);
  lcd.write(7);
  lcd.write(5);
  return 3;
}

int big4(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(3);
  lcd.write(4);
  lcd.write(0XFF);
  lcd.setCursor(col, fil+1);
  lcd.write(32);
  lcd.write(32);
  lcd.write(0XFF);
  return 3;
}

int big5(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(6);
  lcd.setCursor(col, fil+1);
  lcd.write(7);
  lcd.write(7);
  lcd.write(5);
  return 3;
}

int big6(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(6);
  lcd.write(6);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(7);
  lcd.write(5);
  return 3;
}

int big7(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(3);
  lcd.write(1);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(32);
  lcd.write(32);
  lcd.write(5);
  return 3;
}
```

```
int big8(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(7);
  lcd.write(5);
  return 3;
}

int big9(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(7);
  lcd.write(7);
  lcd.write(5);
  return 3;
}

int bigA(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(0xFF);
  return 3;
}

int bigB(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(5);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(7);
  lcd.write(2);
  return 3;
}

int bigC(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(1);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(4);
  lcd.write(4);
  return 3;
}
```

```
int bigD(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(1);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(4);
  lcd.write(5);
  return 3;
}

int bigE(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(7);
  lcd.write(4);
  return 3;
}

int bigF(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  return 3;
}

int bigG(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(1);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(4);
  lcd.write(2);
  return 3;
}

int bigH(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(4);
  lcd.write(0xFF);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(0xFF);
  return 3;
}
```

```
int bigI(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(1);
  lcd.write(0xFF);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(4);
  lcd.write(0xFF);
  lcd.write(4);
  return 3;
}

int bigJ(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(1);
  lcd.write(0xFF);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(4);
  lcd.write(0xFF);
  return 3;
}

int bigK(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(2);
  lcd.write(4);
  lcd.write(5);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(2);
  return 3;
}

int bigL(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(4);
  return 2;
}

int bigM(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(3);
  lcd.write(5);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(32);
  lcd.write(0xFF);
  return 4;
}
```

```
int bigN(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(2);
  lcd.write(32);
  lcd.write(0xFF);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(3);
  lcd.write(5);
  return 4;
}

int bigO(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(1);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(4);
  lcd.write(5);
  return 3;
}

int bigP(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(32);
  return 3;
}

int bigR(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(6);
  lcd.write(2);
  lcd.setCursor(col, fil+1);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(2);
  return 3;
}

int bigS(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(8);
  lcd.write(6);
  lcd.write(6);
  lcd.setCursor(col, fil+1);
  lcd.write(7);
  lcd.write(7);
  lcd.write(5);
  return 3;
}
```

```
int bigT(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(1);
  lcd.write(0xFF);
  lcd.write(1);
  lcd.setCursor(col, fil+1);
  lcd.write(32);
  lcd.write(0xFF);
  lcd.write(32);
  return 3;
}

int bigU(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0xFF);
  lcd.write(32);
  lcd.write(0xFF);
  lcd.setCursor(col, fil+1);
  lcd.write(3);
  lcd.write(4);
  lcd.write(5);
  return 3;
}

int bigV(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(3);
  lcd.write(32);
  lcd.write(32);
  lcd.write(5);
  lcd.setCursor(col, fil+1);
  lcd.write(32);
  lcd.write(3);
  lcd.write(5);
  lcd.write(32);
  return 4;
}

int big2Puntos(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(0b10100001);
  lcd.setCursor(col, fil+1);
  lcd.write(0b10100001);
  return 1;
}

int bigEspacio(int col, int fil, LiquidCrystal_I2C &lcd){
  lcd.setCursor(col, fil);
  lcd.write(32);
  lcd.setCursor(col, fil+1);
  lcd.write(32);
  return 1;
}
```

```
//Creación de texto grande
/*Conversión de cadena de texto a gran tamaño
contruyendo con dígitos y letras grandes*/

void textoGrande(char *str, int col, int fil, LiquidCrystal_I2C &lcd){
  char c;
  while (c = *str++){
    switch(c){
      case '0':
        col += big0(col, fil, lcd);
        break;
      case '1':
        col += big1(col, fil, lcd);
        break;
      case '2':
        col += big2(col, fil, lcd);
        break;
      case '3':
        col += big3(col, fil, lcd);
        break;
      case '4':
        col += big4(col, fil, lcd);
        break;
      case '5':
        col += big5(col, fil, lcd);
        break;
      case '6':
        col += big6(col, fil, lcd);
        break;
      case '7':
        col += big7(col, fil, lcd);
        break;
      case '8':
        col += big8(col, fil, lcd);
        break;
      case '9':
        col += big9(col, fil, lcd);
        break;
      case 'A':
        col += bigA(col, fil, lcd);
        break;
      case 'B':
        col += bigB(col, fil, lcd);
        break;
      case 'C':
        col += bigC(col, fil, lcd);
        break;
      case 'D':
        col += bigD(col, fil, lcd);
        break;
      case 'E':
        col += bigE(col, fil, lcd);
        break;
      case 'F':
        col += bigF(col, fil, lcd);
        break;
      case 'G':
        col += bigG(col, fil, lcd);
        break;
      case 'H':
        col += bigH(col, fil, lcd);
```

```
        break;
      case 'I':
        col += bigI(col, fil, lcd);
        break;
      case 'J':
        col += bigJ(col, fil, lcd);
        break;
      case 'K':
        col += bigK(col, fil, lcd);
        break;
      case 'L':
        col += bigL(col, fil, lcd);
        break;
      case 'M':
        col += bigM(col, fil, lcd);
        break;
      case 'N':
        col += bigN(col, fil, lcd);
        break;
      case 'O':
        col += bigO(col, fil, lcd);
        break;
      case 'P':
        col += bigP(col, fil, lcd);
        break;
      case 'R':
        col += bigR(col, fil, lcd);
        break;
      case 'S':
        col += bigS(col, fil, lcd);
        break;
      case 'T':
        col += bigT(col, fil, lcd);
        break;
      case 'U':
        col += bigU(col, fil, lcd);
        break;
      case 'V':
        col += bigV(col, fil, lcd);
        break;
      case ':':
        col += big2Puntos(col, fil, lcd);
        break;
      case ' ':
        col += bigEspacio(col, fil, lcd);
        break;
      default:
        break;
    }
  }
}
```

1.6 Reproducción de tiempo restante en audio

```
/*Funciones de reproducción de Tiempo en Audio

Para cada módulo de audio se tiene una función,
reproduceTiempoBlancas / reproduceTiempoNegras

Para controlar la reproducción del audio
a través de pulsador o de manera automática
se tiene la función reproduceTiempos*/

 // Tiempo Blancas
void reproduceTiempoBlancas(DFPlayerMini_Fast &voz,DebounceFilter
&filtroAudio, uint8_t pinOcupado,
      int hora, int minuto, int segundo,
      int carpetaNumeros, int carpetaTiempos){ // parámetros de entrada

  int secuencia[6] = {hora, 0, minuto, 0, segundo, 0}; // Secuencia de
audios a reproducir

//Detección de reproducción activa del módulo
/*Para detectar cuando el módulo está reproduciendo y
conocer cuando finaliza un audio para iniciar el siguiente de la secuencia
se lee la señal BUSY y se filtra para detectar el flanco de bajada*/

  bool ocupado;
  int lecturaOcupado = analogRead(pinOcupado); //lectura analógica
  /*Señal estado bajo -> Reproduciendo
    Señal estado alto -> No Reproduciendo*/
  if(lecturaOcupado > 600){ //Mitad de la lectura analógica
    ocupado = false;
  }else{
    ocupado = true;
  }

  filtroAudio.AddValue(ocupado);

/*Inicio de la reproducción de audio si no está reproduciendo (audio
inicial)
o si está reproduciendo y finaliza el audio actual */

    if((reproduciendoBlancas && filtroAudio.IsFalling()) ||
!reproduciendoBlancas){

     reproduciendoBlancas = true; //activación reproducción

     if(indicadorTiempoBlancas == 0){  // Reproduce horas

        if(secuencia[indicadorTiempoBlancas] >= 1){ // si es mayor o igual
a 1 hora
          if(secuencia[indicadorTiempoBlancas] == 1){
            voz.playFolder(carpetaNumeros, 100); // una hora
          }else{
            voz.playFolder(carpetaNumeros,
secuencia[indicadorTiempoBlancas]); // número
          }

          indicadorTiempoBlancas++;
```

```
        }else if(secuencia[indicadorTiempoBlancas] == 0){ // si es menor a
1 hora
          voz.playFolder(carpetaNumeros, 110); // silencio
          indicadorTiempoBlancas= 2; // Salto secuencia a minutos
        }

     }else if(indicadorTiempoBlancas == 1){ // Reproduce indicador de horas
        if(secuencia[0] == 1){
          voz.playFolder(carpetaTiempos, 1); // hora
        }else{
          voz.playFolder(carpetaTiempos, 2); // horas
        }
        indicadorTiempoBlancas++;

     }else if(indicadorTiempoBlancas == 2){ // Reproduce minutos
        if(secuencia[indicadorTiempoBlancas] >= 1){
          voz.playFolder(carpetaNumeros,
secuencia[indicadorTiempoBlancas]); // número
          indicadorTiempoBlancas++;
        }else if(secuencia[indicadorTiempoBlancas] == 0){ //Si es menor a 1
minuto
          indicadorTiempoBlancas = 4;   // Salto a segundos
        }

     }else if(indicadorTiempoBlancas == 3){ // Reproduce indicador de
minutos
        if(secuencia[2] == 1){
          voz.playFolder(carpetaTiempos, 3); // minuto
        }else{
          voz.playFolder(carpetaTiempos, 4); // minutos
        }
        indicadorTiempoBlancas++;

     }else if(indicadorTiempoBlancas == 4){ // Reproduce segundos
        if(secuencia[indicadorTiempoBlancas] >= 1){
          voz.playFolder(carpetaNumeros,
secuencia[indicadorTiempoBlancas]); // número
          indicadorTiempoBlancas++;
        }else if(secuencia[indicadorTiempoBlancas] == 0){
          indicadorTiempoBlancas = 6;   // No se reproducen los segundos
        }

     }else if(indicadorTiempoBlancas == 5){ // Reproduce indicador de
segundos
        if(secuencia[4] == 1){
          voz.playFolder(carpetaTiempos, 5); // segundo
        }else{
          voz.playFolder(carpetaTiempos, 6); // segundos
        }
        indicadorTiempoBlancas++;
     }

   }

    if(indicadorTiempoBlancas >= 6){      // fin de la secuencia
      indicadorTiempoBlancas = 0;         //reinicio secuencia
      reproduciendoBlancas = false;       //fin de la reproducción
      estadoAudioBlancas = noReproducir;  //Control de audio
    }

}
```

```
// Tiempo Negras
void reproduceTiempoNegras(DFPlayerMini_Fast &voz,DebounceFilter
&filtroAudio, uint8_t pinOcupado,
      int hora, int minuto, int segundo,
      int carpetaNumeros, int carpetaTiempos){

  int secuencia[6] = {hora, 0, minuto, 0, segundo, 0}; //secuencia

  bool ocupado;
  int lecturaOcupado = analogRead(pinOcupado);
  /*Señal estado bajo -> Reproduciendo
    Señal estado alto -> No Reproduciendo*/
  if(lecturaOcupado > 600){ //Mitad de la lectura analógica
    ocupado = false;
  }else{
    ocupado = true;
  }

  filtroAudio.AddValue(ocupado);

    if((reproduciendoNegras && filtroAudio.IsFalling()) ||
!reproduciendoNegras){

     reproduciendoNegras = true;

     if(indicadorTiempoNegras== 0){  // Reproduce horas
        if(secuencia[indicadorTiempoNegras] >= 1){
          if(secuencia[indicadorTiempoNegras] == 1){
            voz.playFolder(carpetaNumeros, 100); // una
          }else{
            voz.playFolder(carpetaNumeros,
secuencia[indicadorTiempoNegras]); // número
          }

          indicadorTiempoNegras++;

        }else if(secuencia[indicadorTiempoNegras] == 0){ //menor a 1 hora
          voz.playFolder(carpetaNumeros, 110); // silencio
          indicadorTiempoNegras= 2; // Salto a minutos
        }

     }else if(indicadorTiempoNegras == 1){ // Reproduce indicador de horas
        if(secuencia[0] == 1){
          voz.playFolder(carpetaTiempos, 1); // hora
        }else{
          voz.playFolder(carpetaTiempos, 2); // horas
        }
        indicadorTiempoNegras++;

     }else if(indicadorTiempoNegras == 2){ // Reproduce minutos
        if(secuencia[indicadorTiempoNegras] >= 1){
          voz.playFolder(carpetaNumeros, secuencia[indicadorTiempoNegras]);
// número
          indicadorTiempoNegras++;
        }else if(secuencia[indicadorTiempoNegras] == 0){
          indicadorTiempoNegras = 4;    // Salto a segundos
        }
```

```
     }else if(indicadorTiempoNegras == 3){ // Reproduce indicador de
minutos
        if(secuencia[2] == 1){
          voz.playFolder(carpetaTiempos, 3); // minuto
        }else{
          voz.playFolder(carpetaTiempos, 4); // minutos
        }
        indicadorTiempoNegras++;

     }else if(indicadorTiempoNegras == 4){ // Reproduce segundos
        if(secuencia[indicadorTiempoNegras] >= 1){
          voz.playFolder(carpetaNumeros, secuencia[indicadorTiempoNegras]);
// número
          indicadorTiempoNegras++;
        }else if(secuencia[indicadorTiempoNegras] == 0){
          indicadorTiempoNegras = 6;   // No se reproducen los segundos
        }

     }else if(indicadorTiempoNegras == 5){ // Reproduce indicador de
segundos
        if(secuencia[4] == 1){
          voz.playFolder(carpetaTiempos, 5); // segundo
        }else{
          voz.playFolder(carpetaTiempos, 6); // segundos
        }
        indicadorTiempoNegras++;
     }

   }

    if(indicadorTiempoNegras >= 6){       // fin de la secuencia
      indicadorTiempoNegras = 0;          //reinicio secuencia
      reproduciendoNegras = false;        //fin de reproducción
      estadoAudioNegras = noReproducir;   //control de estado audio
    }

}

//Función de decisión de reprodución de tiempos
/*Se contempla la reproducción del tiempo
-Automática: inicio de partida, programación o edición de tiempo*/
void reproduceTiempos(){

//Reproducción del tiempo automática sin pulsación de pulsadores
   if(reproducirSinPulsador){

      //Actualización de los tiempos
        HoraB = TempBlancas.ShowHours();
        MinutoB = TempBlancas.ShowMinutes();
        SegundoB = TempBlancas.ShowSeconds();

        HoraN = TempNegras.ShowHours();
        MinutoN = TempNegras.ShowMinutes();
        SegundoN = TempNegras.ShowSeconds();
```

```
      /*Inicialización de las secuencias
      y reproducción de los tiempos correspondientes al jugador*/

        indicadorTiempoBlancas = 0; indicadorTiempoNegras = 0;
        estadoAudioBlancas = reproducirBlancas;
        estadoAudioNegras = reproducirNegras;

        //Desactivación de  reproducción automática (entrada única a la
función)
        reproducirSinPulsador = false;
   }

  //Control de tiempo en audio jugador piezas Blancas
  /*lectura de pulsadores de tiempo*/
  pulsadorBlancasAudioBlancas.AddValue(digitalRead(pinpulsadorBlancasAudioB
lancas));
  pulsadorNegrasAudioBlancas.AddValue(digitalRead(pinpulsadorNegrasAudioBla
ncas));

  /*Si se activa un pulsador y no está reproduciendo*/
  if(pulsadorBlancasAudioBlancas.IsRising() && (estadoAudioBlancas ==
noReproducir)){
    indicadorTiempoBlancas = 0;               //inicio de secuencia
    estadoAudioBlancas = reproducirBlancas;   //Tiempo propio

    //Actualización de tiempo
    HoraB = TempBlancas.ShowHours();
    MinutoB = TempBlancas.ShowMinutes();
    SegundoB = TempBlancas.ShowSeconds();

  }else if(pulsadorNegrasAudioBlancas.IsRising() && (estadoAudioBlancas ==
noReproducir)){
    indicadorTiempoNegras = 0;                //inicio de secuencia
    estadoAudioBlancas = reproducirNegras;    //Tiempo oponente

    //Actualización de tiempo
    HoraN = TempNegras.ShowHours();
    MinutoN = TempNegras.ShowMinutes();
    SegundoN = TempNegras.ShowSeconds();
  }

  //Control de tiempo en audio jugador piezas Negras
  /*lectura de pulsadores de tiempo*/
  pulsadorBlancasAudioNegras.AddValue(digitalRead(pinpulsadorBlancasAudioNe
gras));
  pulsadorNegrasAudioNegras.AddValue(digitalRead(pinpulsadorNegrasAudioNegr
as));

  /*Si se activa un pulsador y no está reproduciendo*/
  if(pulsadorBlancasAudioNegras.IsRising() && (estadoAudioNegras ==
noReproducir)){
    indicadorTiempoBlancas = 0;             //inicio de secuencia
    estadoAudioNegras = reproducirBlancas;  //Tiempo oponente

    //Actualización de tiempo
    HoraB = TempBlancas.ShowHours();
    MinutoB = TempBlancas.ShowMinutes();
    SegundoB = TempBlancas.ShowSeconds();
```

```
  }else if(pulsadorNegrasAudioNegras.IsRising() && (estadoAudioNegras ==
noReproducir)){
    indicadorTiempoNegras = 0;              //inicio de secuencia
    estadoAudioNegras = reproducirNegras;   //Tiempo propio

    //Actualización de tiempo
    HoraN = TempNegras.ShowHours();
    MinutoN = TempNegras.ShowMinutes();
    SegundoN = TempNegras.ShowSeconds();
  }

  /*Reproducción de tiempo seleccionado
  Se introduce módulo, filtro antirrebote y pin ocupado
  Tiempos seleccionados
  Carpetas de números y tiempos en el idioma del jugador*/
  if(estadoAudioBlancas == reproducirBlancas){ // Tiempo Blancas Jugador
Blancas
    reproduceTiempoBlancas( vozBlancas, filtroAudioBlancas,
pinOcupadoBlancas,
      HoraB, MinutoB, SegundoB, Numeros[idiomaBlancas],
Tiempos[idiomaBlancas]);

  }else if(estadoAudioBlancas == reproducirNegras){ // Tiempo Negras
Jugador Blancas
    reproduceTiempoBlancas( vozBlancas, filtroAudioBlancas,
pinOcupadoBlancas,
      HoraN, MinutoN, SegundoN, Numeros[idiomaBlancas],
Tiempos[idiomaBlancas]);
  }
  /*Reproducción de tiempo seleccionado
  Se introduce módulo, filtro antirrebote y pin ocupado
  Tiempos seleccionados
  Carpetas de números y tiempos en el idioma del jugador*/
  if(estadoAudioNegras == reproducirBlancas){ // Tiempo Blancas Jugador
Negras
    reproduceTiempoNegras( vozNegras, filtroAudioNegras, pinOcupadoNegras,
      HoraB, MinutoB, SegundoB, Numeros[idiomaNegras],
Tiempos[idiomaNegras]);

  }else if(estadoAudioNegras == reproducirNegras){ // Tiempo Negras Jugador
Negras
    reproduceTiempoNegras( vozNegras, filtroAudioNegras, pinOcupadoNegras,
      HoraN, MinutoN, SegundoN, Numeros[idiomaNegras],
Tiempos[idiomaNegras]);
  }


}
```

1.7 Edición de Tiempos

```
//Funciones Edición Tiempos

/*Función para mostrar los tiempos e incremento editados en pantalla
Cuando se modifiquen valores de tiempo se actualizará la pantalla*/
void imprimirTiemposEditados(){

//Tiempo blancas
  LCDBlancas.setCursor(0,0);
  LCDBlancas.print("Tiempo Blancas");
  LCDNegras.setCursor(0,0);
  LCDNegras.print("Tiempo Blancas");

  sprintf(tiempoEditadoBlancas, "%01d:%02d:%02d +%02ds", HoraB, MinutoB,
SegundoB, incremento);
  LCDBlancas.setCursor(0,1);
  LCDBlancas.print(tiempoEditadoBlancas);
  LCDNegras.setCursor(0,1);
  LCDNegras.print(tiempoEditadoBlancas);

//Tiempo negras
  LCDBlancas.setCursor(0,2);
  LCDBlancas.print("Tiempo Negras");
  LCDNegras.setCursor(0,2);
  LCDNegras.print("Tiempo Negras");

  sprintf(tiempoEditadoNegras, "%01d:%02d:%02d +%02ds", HoraN, MinutoN,
SegundoN, incremento);
  LCDBlancas.setCursor(0,3);
  LCDBlancas.print(tiempoEditadoNegras);
  LCDNegras.setCursor(0,3);
  LCDNegras.print(tiempoEditadoNegras);


}

/*Función de audios en la edición de tiempos
En la selección se reproduce el indicador de tiempo
Si se modifica un valor, se reproduce todo el tiempo o el valor del
incremento*/
void reproducirTiemposEditados(){

  //Selección de valor modificable

  if((entrada == tipoEntrada::Atras) || (entrada ==
tipoEntrada::Siguiente)){
    if(indiceEditor == 0 || indiceEditor == 7){
      vozBlancas.playFolder(Tiempos[idiomaBlancas], 2); // horas
      vozNegras.playFolder(Tiempos[idiomaBlancas], 2); // horas

    }else if(indiceEditor == 1 || indiceEditor == 2 ||
             indiceEditor == 8 || indiceEditor == 9){
      vozBlancas.playFolder(Tiempos[idiomaBlancas], 4); // minutos
      vozNegras.playFolder(Tiempos[idiomaBlancas], 4); // minutos

    }else if(indiceEditor == 3 || indiceEditor == 4 ||
             indiceEditor == 10 || indiceEditor == 11){
      vozBlancas.playFolder(Tiempos[idiomaBlancas], 6); // segundos
      vozNegras.playFolder(Tiempos[idiomaBlancas], 6); // segundos
```

```
    }else if(indiceEditor == 5 || indiceEditor == 6 ||
              indiceEditor == 12 || indiceEditor == 13){
      vozBlancas.playFolder(Tiempos[idiomaBlancas], 7); // incremento
      vozNegras.playFolder(Tiempos[idiomaBlancas], 7); // incremento
    }

  //Modificación del valor

  }else if((entrada == tipoEntrada::Mas) || (entrada ==
tipoEntrada::Menos)){
    if(indiceEditor <= 4){
      indicadorTiempoBlancas = 0; indicadorTiempoNegras = 0;
      estadoAudioBlancas = reproducirBlancas;                 // Tiempo
Blancas
      estadoAudioNegras = reproducirBlancas;

    }else if(indiceEditor >= 7 && indiceEditor <= 11){
      indicadorTiempoBlancas = 0; indicadorTiempoNegras = 0;
      estadoAudioBlancas = reproducirNegras;                  //Tiempo
Negras
      estadoAudioNegras = reproducirNegras;

    }else if(indiceEditor == 5 || indiceEditor == 6 ||
              indiceEditor == 12 || indiceEditor == 13){
      vozBlancas.playFolder(Numeros[idiomaBlancas], incremento); // valor
del incremento
      vozNegras.playFolder(Numeros[idiomaBlancas], incremento); // valor
del incremento
    }
  }

}

/*Función edición de Tiempos de la partida
  Se podrán modificar decenas y unidades de cada parámetro de los 2
jugadores
      H:M:S + Inc
*/
void edicionTiempos(){

  //Variables para el parpadeo del dígito seleccionado
  int fila = 1;   //por defecto segunda fila
  int columna;    //Se modifica en cada identificador

  //Actualización de Tiempos
  HoraB = TempBlancas.ShowHours();
  MinutoB = TempBlancas.ShowMinutes();
  SegundoB = TempBlancas.ShowSeconds();
  HoraN = TempNegras.ShowHours();
  MinutoN = TempNegras.ShowMinutes();
  SegundoN = TempNegras.ShowSeconds();
```

```
/*Selección del parámetro a modificar

  Se asocia a un identificador para realizar la selección*/
 switch (entrada) {
 case tipoEntrada::Siguiente :
   if(indiceEditor < 13){  //menos de 13 parámetros
     indiceEditor++;       //incremento
   }
   break;
 case tipoEntrada::Atras :
   if(indiceEditor > 0){
     indiceEditor--;       //decremento
   }
   break;
 default:
   // statements
   break;
 }

 /*Modificación del valor en cada identificador en unidades y decenas
 Siempre que el valor esté dentro de los límites
 Horas -> 0-9
 Minutos y segundos -> 0-59
 Incremento 0-60
 */
 switch (indiceEditor) {

   case 0 : // horas blancas
     if((entrada == tipoEntrada::Mas) && (HoraB < 9)){
       HoraB++;
       imprimirTiemposEditados();
     }else if((entrada == tipoEntrada::Menos) && (HoraB > 0)){
       HoraB--;
       imprimirTiemposEditados();
     }
     columna = 0; //Posición cursor
     break;

   case 1 : //decenas minutos blancas
     if((entrada == tipoEntrada::Mas) && (MinutoB +10 < 60)){
       MinutoB += 10;              //incrementos de 10
       imprimirTiemposEditados();
     }else if((entrada == tipoEntrada::Menos) && (MinutoB -10 >= 0)){
       MinutoB -= 10;              //decrementos de 10
       imprimirTiemposEditados();
     }
     columna = 2; //Posición cursor
     break;

   case 2: //unidades minutos blancas
     if((entrada == tipoEntrada::Mas) && (MinutoB < 59)){
       MinutoB++;
       imprimirTiemposEditados();
     }else if((entrada == tipoEntrada::Menos) && (MinutoB > 0)){
       MinutoB--;
       imprimirTiemposEditados();
     }
     columna = 3; //Posición cursor
   break;
```

```
case 3 : //decenas segundos blancas
  if((entrada == tipoEntrada::Mas) && (SegundoB +10 < 60)){
    SegundoB += 10;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (SegundoB -10 >= 0)){
    SegundoB -= 10;
    imprimirTiemposEditados();
  }
  columna = 5; //Posición cursor
  break;

case 4: //unidades segundos blancas
  if((entrada == tipoEntrada::Mas) && (SegundoB < 59)){
    SegundoB++;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (SegundoB > 0)){
    SegundoB--;
    imprimirTiemposEditados();
  }
  columna = 6; //Posición cursor
  break;

case 5 : //decenas incremento blancas
  if((entrada == tipoEntrada::Mas) && (incremento +10 <= 60)){
    incremento += 10;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (incremento -10 >= 0)){
    incremento -= 10;
    imprimirTiemposEditados();
  }
  columna = 9; //Posición cursor
  break;

case 6: //unidades imcremento blancas
  if((entrada == tipoEntrada::Mas) && (incremento < 60)){
    incremento++;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (incremento > 0)){
    incremento--;
    imprimirTiemposEditados();
  }
  columna = 10; //Posición cursor
break;


case 7 : // horas negras
  if((entrada == tipoEntrada::Mas) && (HoraN < 9)){
    HoraN++;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (HoraN > 0)){
    HoraN--;
    imprimirTiemposEditados();
  }
  columna = 0; //Posición cursor
  fila = 3;
  break;
```

```
case 8 : //decenas minutos blancas
  if((entrada == tipoEntrada::Mas) && (MinutoN +10 < 60)){
    MinutoN += 10;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (MinutoN -10 >= 0)){
    MinutoN -= 10;
    imprimirTiemposEditados();
  }
  columna = 2; //Posición cursor
  fila = 3;
  break;

case 9: //unidades minutos blancas
  if((entrada == tipoEntrada::Mas) && (MinutoN < 59)){
    MinutoN++;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (MinutoN > 0)){
    MinutoN--;
    imprimirTiemposEditados();
  }
  columna = 3; //Posición cursor
  fila = 3;
break;

case 10 : //decenas segundos blancas
  if((entrada == tipoEntrada::Mas) && (SegundoN +10 < 60)){
    SegundoN += 10;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (SegundoN -10 >= 0)){
    SegundoN -= 10;
    imprimirTiemposEditados();
  }
  columna = 5; //Posición cursor
  fila = 3;
  break;

case 11: //unidades segundos blancas
  if((entrada == tipoEntrada::Mas) && (SegundoN < 59)){
    SegundoN++;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (SegundoN > 0)){
    SegundoN--;
    imprimirTiemposEditados();
  }
  columna = 6; //Posición cursor
  fila = 3;
  break;

case 12: //decenas incremento blancas
  if((entrada == tipoEntrada::Mas) && (incremento +10 <= 60)){
    incremento += 10;
    imprimirTiemposEditados();
  }else if((entrada == tipoEntrada::Menos) && (incremento -10 >= 0)){
    incremento -= 10;
    imprimirTiemposEditados();
  }
  columna = 9; //Posición cursor
  fila = 3;
  break;
```

```
    case 13: //unidades incremento blancas
      if((entrada == tipoEntrada::Mas) && (incremento < 60)){
        incremento++;
        imprimirTiemposEditados();
      }else if((entrada == tipoEntrada::Menos) && (incremento > 0)){
        incremento--;
        imprimirTiemposEditados();
      }
      columna = 10;
      fila = 3;
    break;

  default:
    // statements
    break;
  }

  //Colocación del cursor selección y parpadeo
  LCDBlancas.setCursor(columna, fila);
  LCDBlancas.blink();
  LCDNegras.setCursor(columna, fila);
  LCDNegras.blink();

//Ajuste de Temporizadores
  TempBlancas.SetTimer(0, HoraB, MinutoB, SegundoB);
  TempNegras.SetTimer(0, HoraN, MinutoN, SegundoN);

  TempBlancas.StartTimer();
  TempNegras.StartTimer();

  TempBlancas.PauseTimer();
  TempNegras.PauseTimer();

//Reproducir en audio los tiempos
  reproducirTiemposEditados();

}
```

1.8 Lectura de etiquetas RFID

```
//Lectura de etiqueta RFID

void leerEtiqueta(MFRC522 &mfrc522, MFRC522::MIFARE_Key &key){

   if (!mfrc522.PICC_IsNewCardPresent())
      return;
   if (!mfrc522.PICC_ReadCardSerial())
      return;

   MFRC522::StatusCode status;

   //Zona de memoria de etiqueta
   byte trailerBlock = 7;
   byte sector = 1;
   byte blockAddr = 4;

   //Comprobacion de la etiqueta
   status = (MFRC522::StatusCode) mfrc522.PCD_Authenticate(
        MFRC522::PICC_CMD_MF_AUTH_KEY_A, trailerBlock, &key,
&(mfrc522.uid));
   if (status != MFRC522::STATUS_OK) {
      //Serial.print(F("PCD_Authenticate() failed: "));
      //Serial.println(mfrc522.GetStatusCodeName(status));
      return;
   }

   byte buffer[18];
   byte size = sizeof(buffer);

   // Lectura de datos del bloque
   status = (MFRC522::StatusCode) mfrc522.MIFARE_Read(blockAddr, buffer,
&size);

   if (status != MFRC522::STATUS_OK) {
      //Serial.print(F("MIFARE_Read() failed: "));
      //Serial.println(mfrc522.GetStatusCodeName(status));
   }

   // Halt PICC
   mfrc522.PICC_HaltA();
   // Stop encryption on PCD
   mfrc522.PCD_StopCrypto1();

   // Guardado de variables
    selecHor = buffer[0];
    selecMin = buffer[2];
    selecSeg = buffer[4];
    incremento = buffer[6];
    programado = true;        //Activacion de transicion

}
```

1.9 Escritura de etiquetas RFID

Este código se ejecuta independiente de las funciones del reloj

```
/*
 * Programa para escritura de etiquetas RFID
 * con tiempos partida reloj adaptado
 * Autor: Daniel Lorenzo López
 */

/*Pines e inclusión de librerías*/
//RST           D9
//SDA(SS)      D10
//MOSI         D11
//MISO         D12
//SCK          D13
#include <SPI.h>
#include <MFRC522.h>
const int RST_PIN = 9;
const int SS_PIN = 53;

//Variables de tiempos a escribir
int horas = 0;
int minutos = 15;
int segundos = 0;
int incremento = 0;
unsigned int data[16] = { horas, minutos, segundos, incremento,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

//Constructores
MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;

//Función para imprimir memoria
void printArray(byte *buffer, byte bufferSize) {
   for (byte i = 0; i < bufferSize; i++) {
      Serial.print(buffer[i] < 0x10 ? " 0" : " ");
      Serial.print(buffer[i], DEC);
   }
}

void setup()
{
  /*Inicialización de librerías y comunicación serial*/
   Serial.begin(9600);
   SPI.begin();
   mfrc522.PCD_Init();
   for (byte i = 0; i < 6; i++) {
      key.keyByte[i] = 0xFF;
   }
}

void loop()
{
  //Detección de nueva etiqueta
   if (!mfrc522.PICC_IsNewCardPresent())
      return;
   if (!mfrc522.PICC_ReadCardSerial())
      return;

   //Variable de estado de la etiqueta
```

```
   MFRC522::StatusCode status;

   //Zona de memoria para escritura
   byte trailerBlock = 7;
   byte sector = 1;
   byte blockAddr = 4;

   //Autenticación
   status = (MFRC522::StatusCode)
mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
                                                trailerBlock, &key,
&(mfrc522.uid));

    //Comprobación de autenticación
   if (status != MFRC522::STATUS_OK) {
      Serial.print(F("PCD_Authenticate() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
      return;
   }


   // Escritura de datos
   Serial.print(F("Escribir datos en sector "));
   Serial.print(blockAddr);
   Serial.println(F(" ..."));
   printArray((byte*)data, 16); Serial.println();

   status = (MFRC522::StatusCode) mfrc522.MIFARE_Write(blockAddr,
(byte*)data, 16);

   //Comprobación de escritura
   if (status != MFRC522::STATUS_OK) {
      Serial.print(F("MIFARE_Write() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
   }
   Serial.println();

   //buffer de memoria para lectura
   byte buffer[18];
   byte size = sizeof(buffer);

   // Lectura de memoria para comprobación
   Serial.print(F("Leer datos del sector ")); Serial.print(blockAddr);
   Serial.println(F(" ..."));
   status = (MFRC522::StatusCode) mfrc522.MIFARE_Read(blockAddr, buffer,
&size);

   //Comprobación de lectura
   if (status != MFRC522::STATUS_OK) {
      Serial.print(F("MIFARE_Read() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
   }
```

```
   //Muestra por pantalla la información

   Serial.print(F("Data in block ")); Serial.print(blockAddr);
Serial.println(F(":"));

   printArray(buffer, 16); Serial.println();


   Serial.print(buffer[0]); Serial.print(" ");

   Serial.print(buffer[2]); Serial.print(" ");

   Serial.print(buffer[4]); Serial.print(" ");

   Serial.print(buffer[6]); Serial.println(" ");


   //Cierre de la comunicación

   // Halt PICC

   mfrc522.PICC_HaltA();

   // Stop encryption on PCD

   mfrc522.PCD_StopCrypto1();

}
```

1.10 Recepción de datos del tablero y reproducción de notación

```
/*Funciones para la recepción de notación del tablero
 * y reproducción en audio
*/
String textoEntrada = "";    // Cadena de texto de entrada
bool notacionActiva = false;  //Variable para la reproducción de la
notación

 // Reproduce la notación del movimiento en audio
 /*Con el mismo sistema de reproducción que los tiempos
  * se cambia la secuencia de audios y se reproduce en los dos módulos a la
vez*/

void reproduceNotacion(){

  /*Secuencia de 4 audios de la notación letra/número/letra/número  */
  int secuencia[4] = {textoEntrada[0], textoEntrada[1], textoEntrada[2],
textoEntrada[3]};
  int letra = 0;        //Variable para adaptación de letra al número de
carpeta correspondiente

/*Se leen las señales de módulos MP3 ocupados
 * Se filtran para detectar los flancos
 */

  bool ocupadoB;
  int lecturaOcupadoB = analogRead(pinOcupadoBlancas);
  /*Señal estado bajo -> Reproduciendo
  Señal estado alto -> No Reproduciendo*/
  if(lecturaOcupadoB > 600){
    ocupadoB = false;
  }else{
    ocupadoB = true;
  }

  bool ocupadoN;
  int lecturaOcupadoN = analogRead(pinOcupadoNegras);
  // Se invierte la señal del módulo
  if(lecturaOcupadoN > 600){
    ocupadoN = false;
  }else{
    ocupadoN = true;
  }

  //Filtro de señales módulos ocupados
  filtroAudioBlancas.AddValue(ocupadoB);
  filtroAudioNegras.AddValue(ocupadoN);

/*
 * Reproduce el siguiente audio de la secuencia
 * si se detecta el flanco de bajada del audio de negras
 * o no se está reproduciendo y los módulos están libres
 */
    if((reproduciendoNotacion && filtroAudioNegras.IsFalling())
        || !reproduciendoNotacion && (filtroAudioNegras.GetState() == 0) &&
        (filtroAudioBlancas.GetState() == 0)){

     reproduciendoNotacion = true;
     const int idiomaNotacion = 2;
```

```
     if(indicadorNotacion == 0 || indicadorNotacion == 2){ // letras de la
secuencia

//Conversión de las letras en números para los audios
      switch (secuencia[indicadorNotacion]) {
        case 'a':
          letra = 1;
          break;
        case 'b':
          letra = 2;
          break;
        case 'c':
          letra = 3;
        break;
        case 'd':
          letra = 4;
        break;
        case 'e':
          letra = 5;
        break;
        case 'f':
          letra = 6;
        break;
        case 'g':
          letra = 7;
        break;
        case 'h':
          letra = 8;
        break;
        default:
          letra = 0;
          break;
       }

     }

     if(indicadorNotacion == 0){  // Reproduce letra inicial

        //secuencia[indicadorNotacion]

        vozBlancas.playFolder(carpetaLetras, letra);
        vozNegras.playFolder(carpetaLetras, letra);

        indicadorNotacion++;

     }else if(indicadorNotacion == 1){ // Reproduce número inicial

        //Conversión de tipo String a tipo int
        char num1 = textoEntrada[1];
        int fila = num1 - '0';

        vozBlancas.playFolder(Numeros[idiomaNotacion], fila);
        vozNegras.playFolder(Numeros[idiomaNotacion], fila);

        indicadorNotacion++;

     }else if(indicadorNotacion == 2){ // Reproduce letra final

          vozBlancas.playFolder(carpetaLetras, letra);
          vozNegras.playFolder(carpetaLetras, letra);
```

```
          indicadorNotacion++;

     }else if(indicadorNotacion == 3){ // Reproduce reproduce número final

        //Conversión de tipo String a tipo int
        char num1 = textoEntrada[3];
        int fila = num1 - '0';

        vozBlancas.playFolder(Numeros[idiomaNotacion], fila);
        vozNegras.playFolder(Numeros[idiomaNotacion], fila);

        indicadorNotacion++;
     }

   }

    if(indicadorNotacion >= 4){ // fin de la secuencia
      indicadorNotacion = 0;          //Reinicio de la secuencia
      reproduciendoNotacion = false;  //Apagado de la reproducción
secuencia
      notacionActiva = false;         //Apagado de la función de
reproducción
    }

}

/*Función para la recepción de la notación a través de comunicación serial
Se usa para esta función tanto en reloj como tablero el Serial 3
*/
void recibirDatosTablero(){

   if (Serial3.available() ) {  //Disponible la recepción de datos
    /*Se lee el dato recibido hasta el caracter _
    no se incluye en la lectura
    Se guarda en la variable de texto de entrada*/
    textoEntrada = Serial3.readStringUntil('_');

    //Serial.println(textoEntrada);

    if(textoEntrada == "ini"){ // Recepción de primera posición de la pieza

        //reproduce pitido de detección de posición inicial
        vozBlancas.playFolder(carpetaLetras, 9);
        vozNegras.playFolder(carpetaLetras, 9);

    }else{ //Recepción de notación
      notacionActiva = true;
    }
  }

  if (notacionActiva){
    reproduceNotacion();    //Función de reproducción de notación
  }

}
```

2 Tablero

```
/*
 * Tablero de ajedrez adaptado para jugadores con discapacidad visual
 * Autor: Daniel Lorenzo López
 */

//Libreria filtro antirrebote
#include <DebounceFilterLib.h>

//Definición de pines
const byte pinesFilas[8] =       { 31, 33, 35, 37, 36, 34, 32, 30};
const byte pinesColumnas[8] =    { 23, 25, 27, 29, 28, 26, 24, 22};

// Posiciones del tablero
String casillas[8][8] = {
  //a     b     c     d     e     f     g     h
  {"a8", "b8", "c8", "d8", "e8", "f8", "g8", "h8"}, // 8
  {"a7", "b7", "c7", "d7", "e7", "f7", "g7", "h7"}, // 7
  {"a6", "b6", "c6", "d6", "e6", "f6", "g6", "h6"}, // 6
  {"a5", "b5", "c5", "d5", "e5", "f5", "g5", "h5"}, // 5
  {"a4", "b4", "c4", "d4", "e4", "f4", "g4", "h4"}, // 4
  {"a3", "b3", "c3", "d3", "e3", "f3", "g3", "h3"}, // 3
  {"a2", "b2", "c2", "d2", "e2", "f2", "g2", "h2"}, // 2
  {"a1", "b1", "c1", "d1", "e1", "f1", "g1", "h1"}  // 1
};


// Contructores filtro antirrebote
DebounceFilter filtroCasillas;

//Tiempos pulsador mantenido
unsigned long tiempoInicio, tiempoFinal, tiempoPulsado;

//Posición Actual
byte fila, columna;
//Posición inicial pieza
byte filaInicial, columnaInicial;
//Contador pulsaciones
int contMovimientos = 0;
//texto enviado
String envio;

// Inicializacion
void setup()
{
  //Inicio comunicación serial PC (0) y entre arduinos (3)
  Serial.begin(115200);
  Serial3.begin(9600);
```

```
 //Configuración de los pines de filas y columnas
  // Columnas en alta impedancia
  for (byte c = 0; c < 8; c++)
  {
    pinMode(pinesColumnas[c], INPUT);
    digitalWrite(pinesColumnas[c], HIGH);
  }

  // Filas en pullup
  for (byte r = 0; r < 8; r++)
  {
    pinMode(pinesFilas[r], INPUT_PULLUP);
  }

//Configuración filtro antirrebote
  filtroCasillas.SetInterval(50);
}

void loop()
{
  //leer pulsadores
  filtroCasillas.AddValue(leerCasillas());

  //Obtencion de tiempo entre flanco de subida y bajada
  if(filtroCasillas.IsRising()){
    tiempoInicio = millis();
  }else if(filtroCasillas.IsFalling()){
    tiempoFinal = millis();
  }
  tiempoPulsado = tiempoFinal - tiempoInicio; //diferencia de tiempo

  // Detección de tecla pulsada durante tiempo determinado
    if ((filtroCasillas.IsFalling()) && (tiempoPulsado >= 300)) {

      //Comienzo de la notacion si es la pulsacion inicial
      if ((contMovimientos == 0) ){

        Serial.println("inicial");  //puerto serie pantalla
        Serial3.print("ini_");      //envúo al reloj

        //Se guarda la fila y columna
        filaInicial = fila;
        columnaInicial = columna;

        contMovimientos ++;
      } //end posicion Inicial

      //Envúo de la notación de la jugada si no es la misma posicion
      if((contMovimientos == 1) && (casillas[fila][columna] !=
casillas[filaInicial][columnaInicial]) ){

        //Se crea el texto
        envio = casillas[filaInicial][columnaInicial] +
casillas[fila][columna] + "_";

        Serial.println(envio); //puerto serie pantalla
        Serial3.print(envio);  //envío al reloj

        contMovimientos = 0; //reinicio del contador

      } // fin posición Final
```

```
    }// fin detección posiciones

} //end loop


// Leer el estado del teclado
bool leerCasillas(){
  bool resultado = false;

  // Barrido de columnas
  for (byte c = 0; c < 8; c++)
  {
    // Poner columna a LOW
    pinMode(pinesColumnas[c],OUTPUT);
    digitalWrite(pinesColumnas[c], LOW);

    // Barrer todas las filas comprobando pulsaciones
    for (byte f = 0; f < 8; f++)
    {
      if (digitalRead(pinesFilas[f]) == LOW)
      {
        // Pulsacion detectada, guardar fila y columna
        fila = f;
        columna = c;
        resultado = true;
      }
    }
    // Devolver la columna a alta impedancia
    digitalWrite(pinesColumnas[c], HIGH);
    pinMode(pinesColumnas[c], INPUT);
  }
  return resultado;
}
```

Anexo II: Planos

1: Plano General

2: Secciones y detalle Carcasa del reloj

3: Interruptor de jugadas

4: Sujeción Interruptor de Jugadas

5: Montaje Interruptor de jugadas

6: Base de reloj

7: Ensamblaje base del tablero

8: Módulo base centro

9: Módulos base laterales

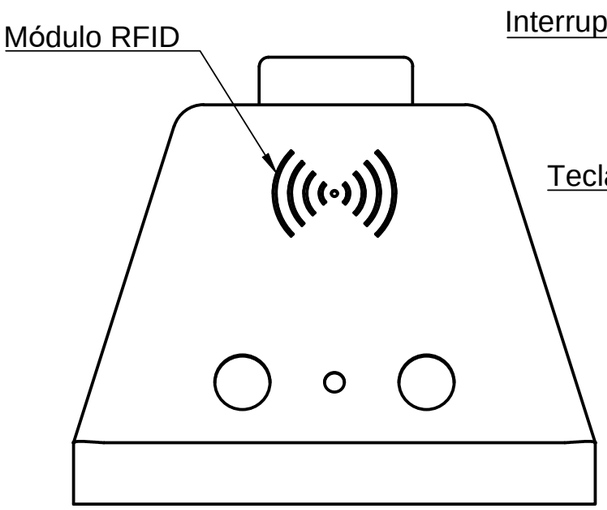
10: Base esquinas tablero

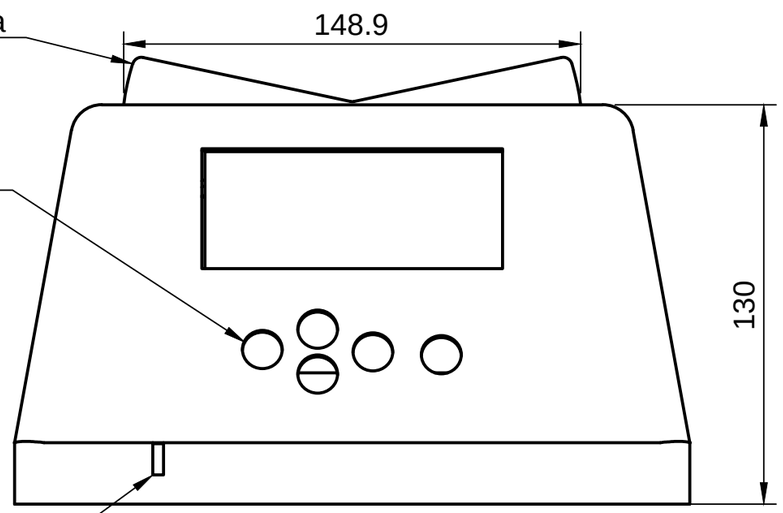
11: Base módulo electrónica

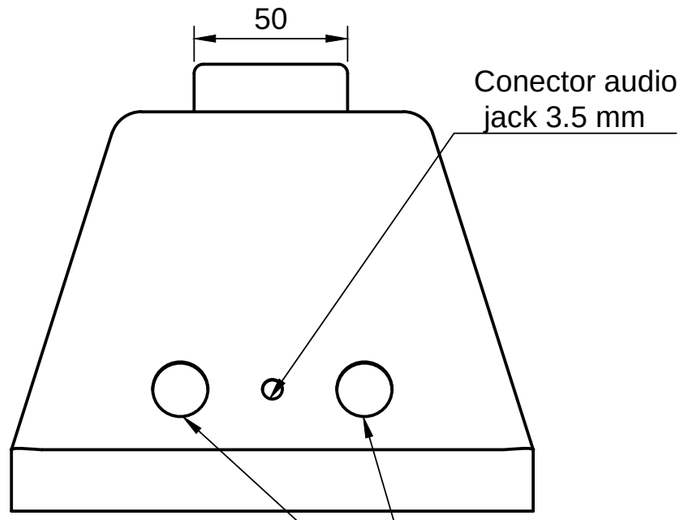
12: Cubierta módulo electrónica

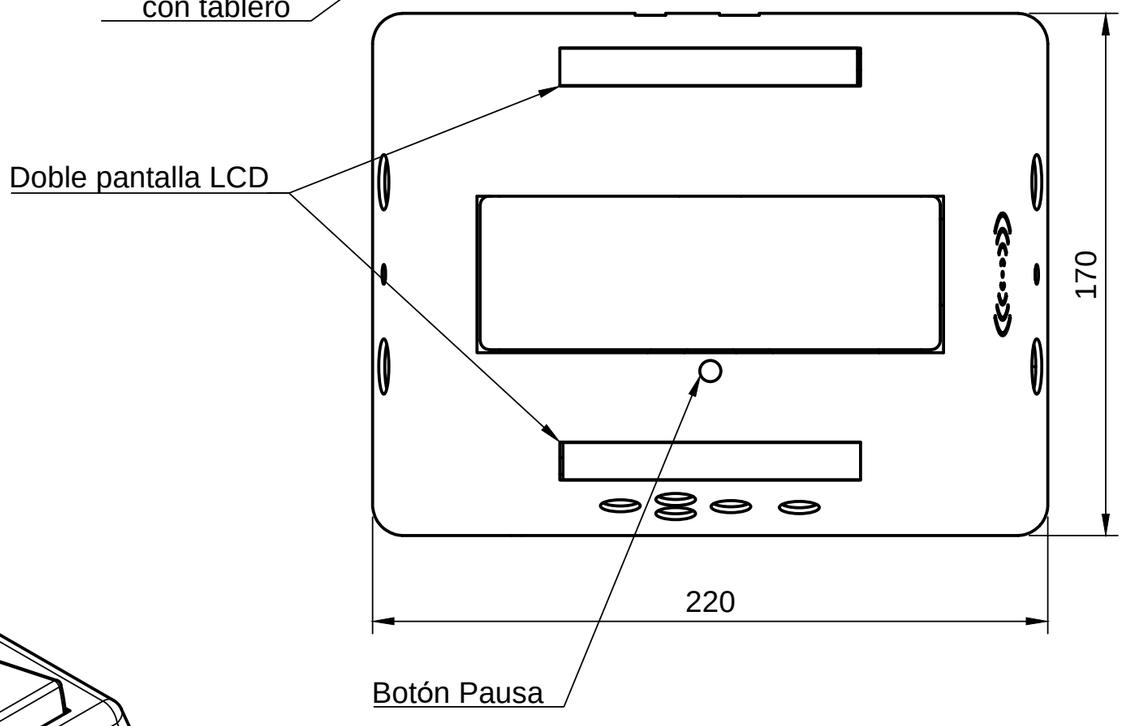
13: Casillas Negras y Blancas

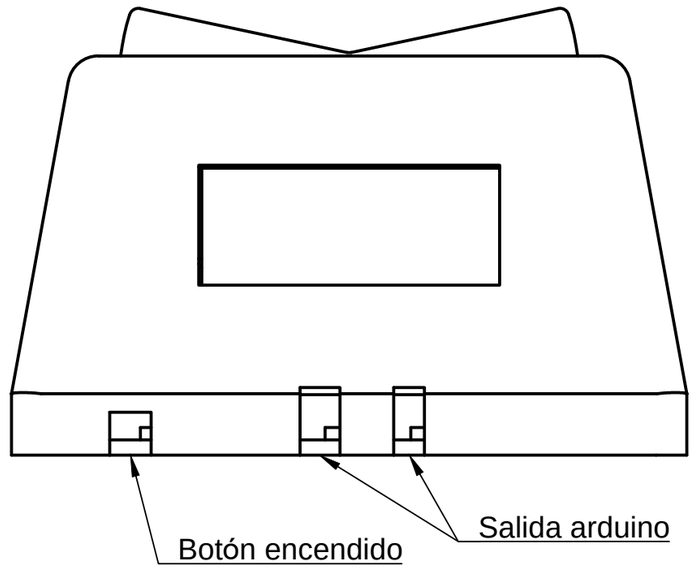
14: Ensamblaje módulo tablero

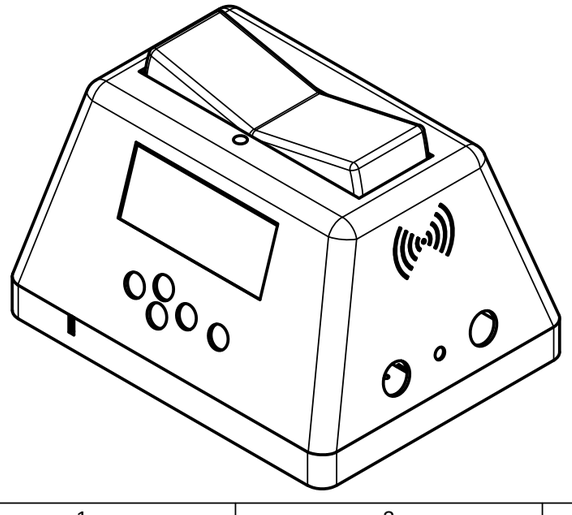






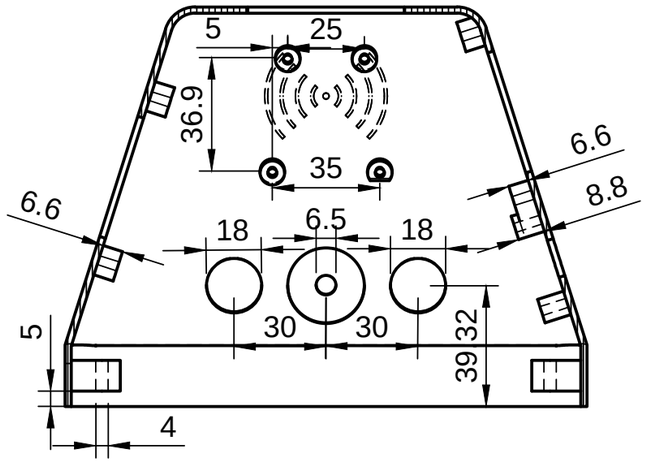




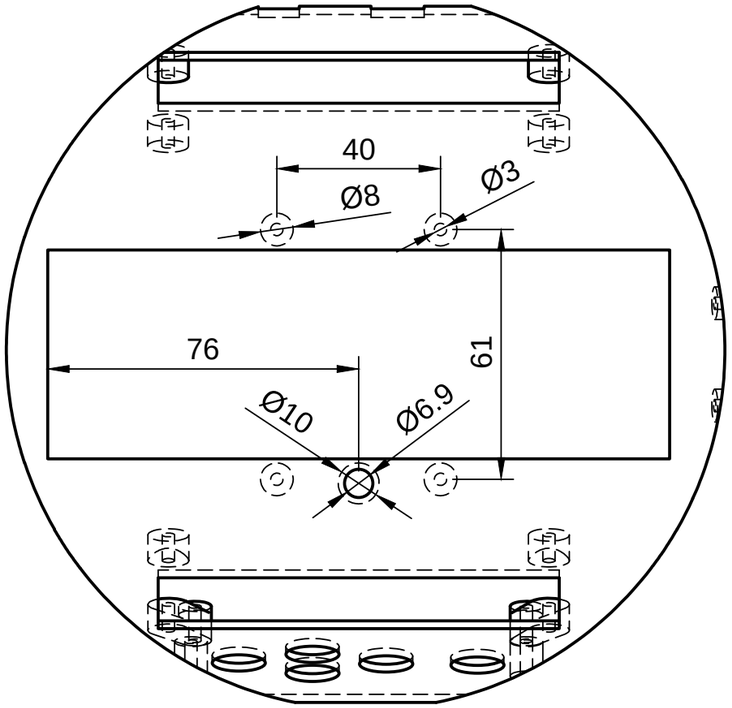


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Plano General — Reloj de Ajedrez Adaptado
	Escala: 1:2
	Página: 1/14

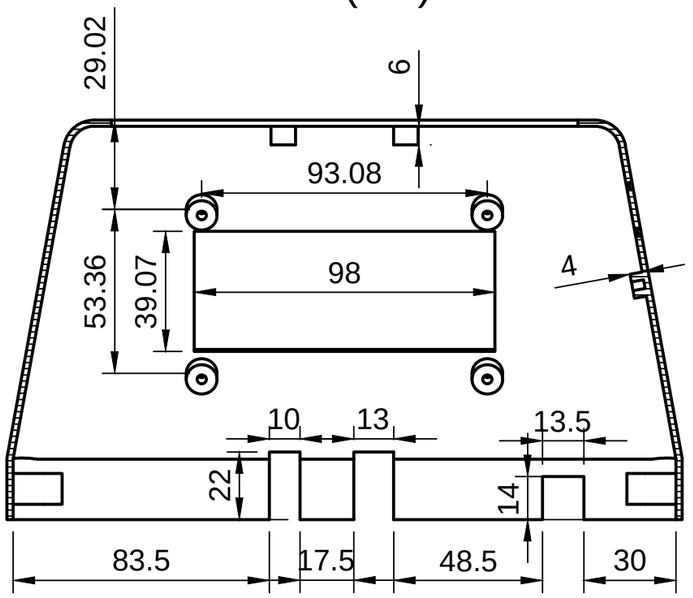
A-A (1:2)

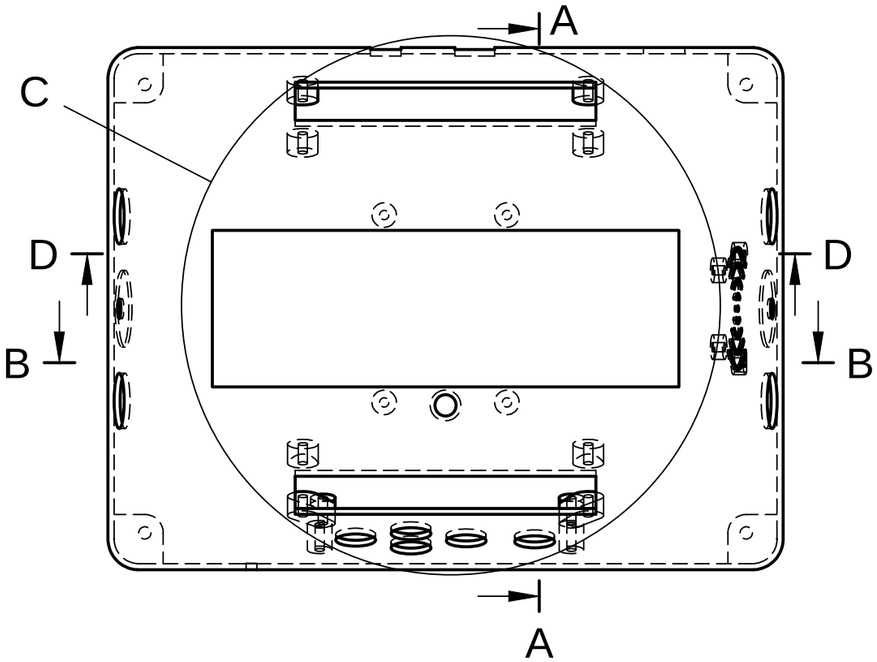


C (1:1.5)

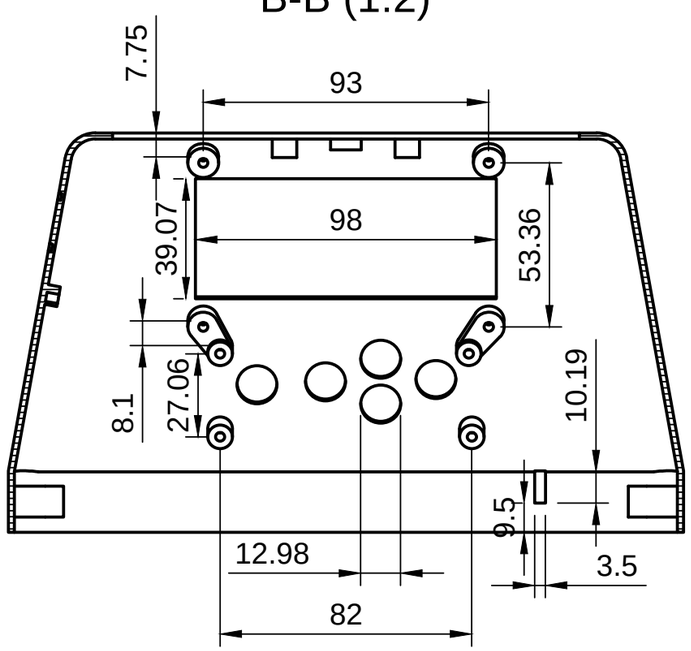


D-D (1:2)

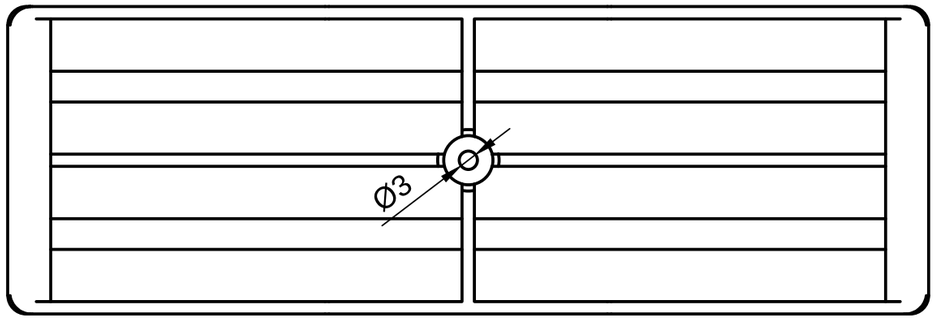


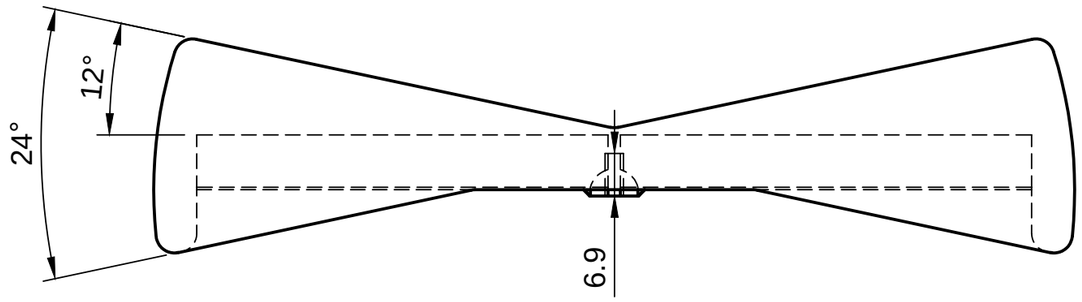


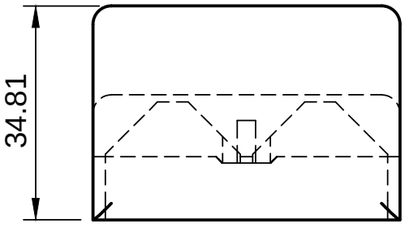
B-B (1:2)

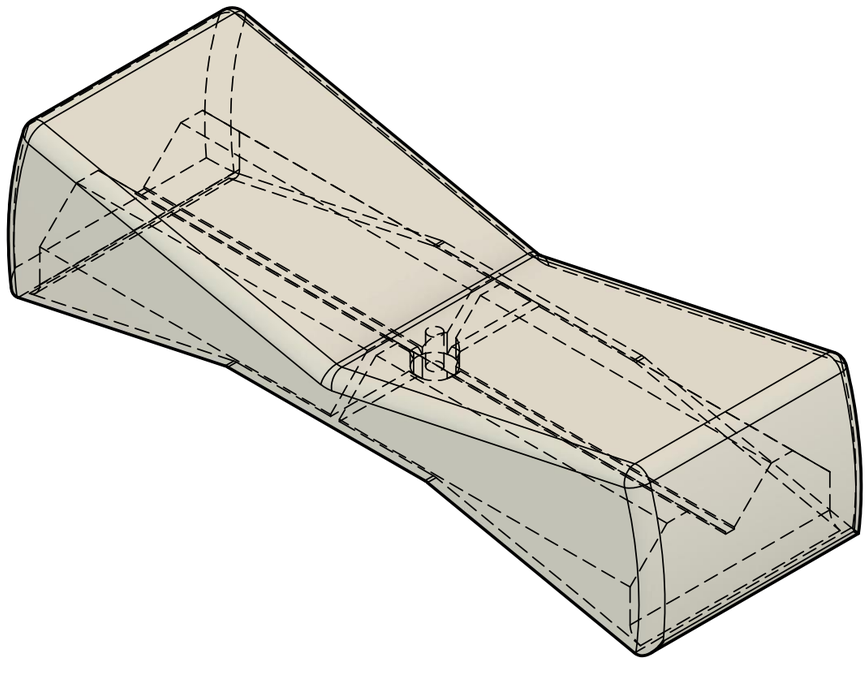


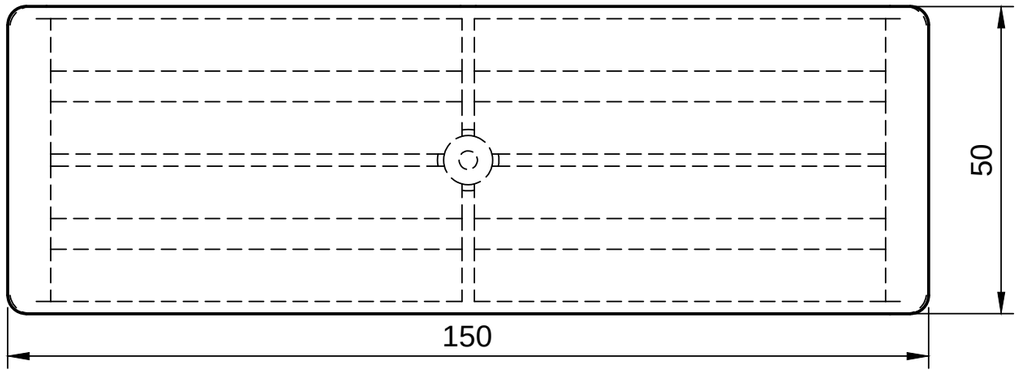
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
Título: Secciones y vistas detalle — Carcasa Reloj Ajedrez Adaptado	Escala: 1:2
	Página: 2/14



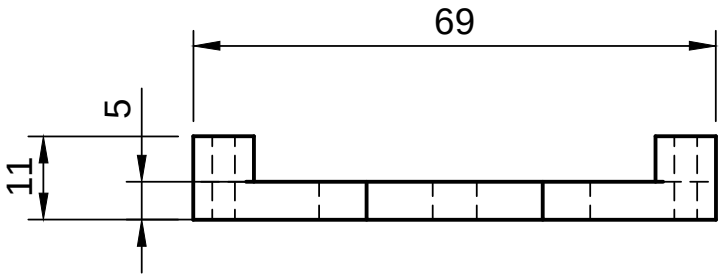


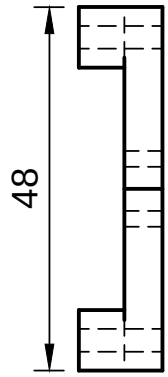


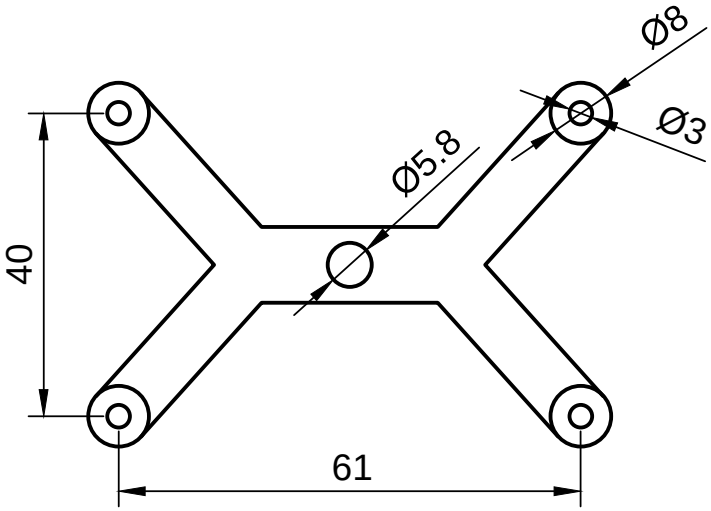


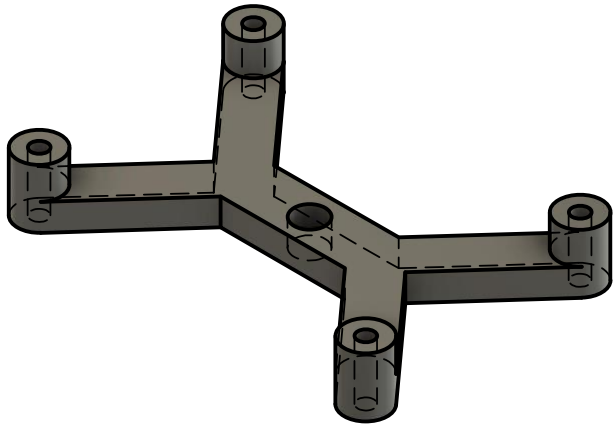


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Interruptor jugada
	Escala: 1:1
	Página: 3/14



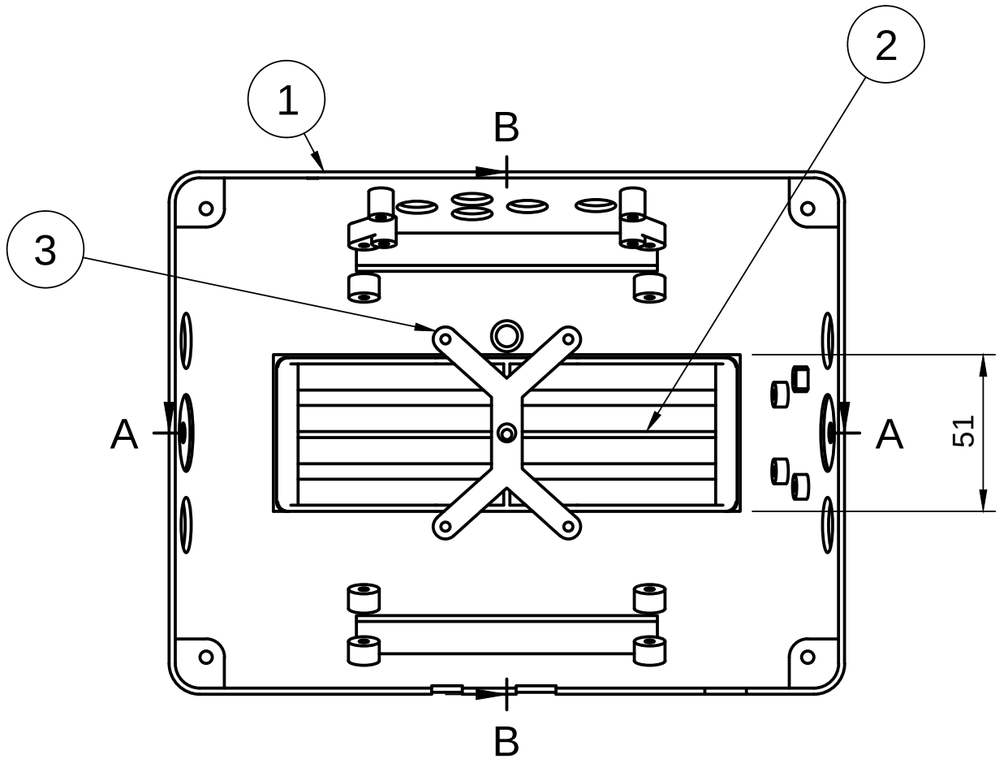




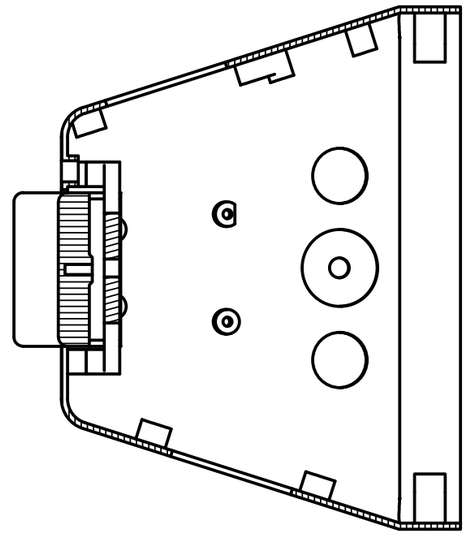


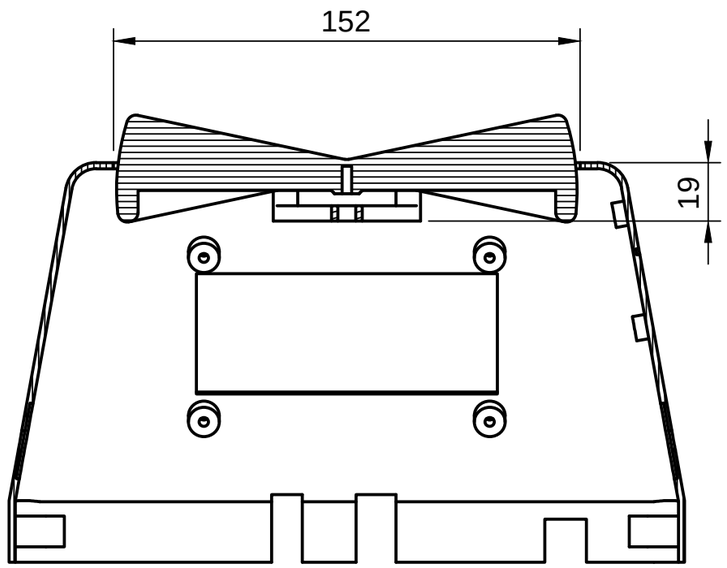
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Sujeción Interruptor Jugada Reloj Ajedrez Adaptado
	Escala: 1:1
	Página: 4/14

Lista de piezas		
elemento	ctd	nombre de pieza
1	1	Carcasa
2	1	Interruptor jugadas
3	1	Sujeción interruptor



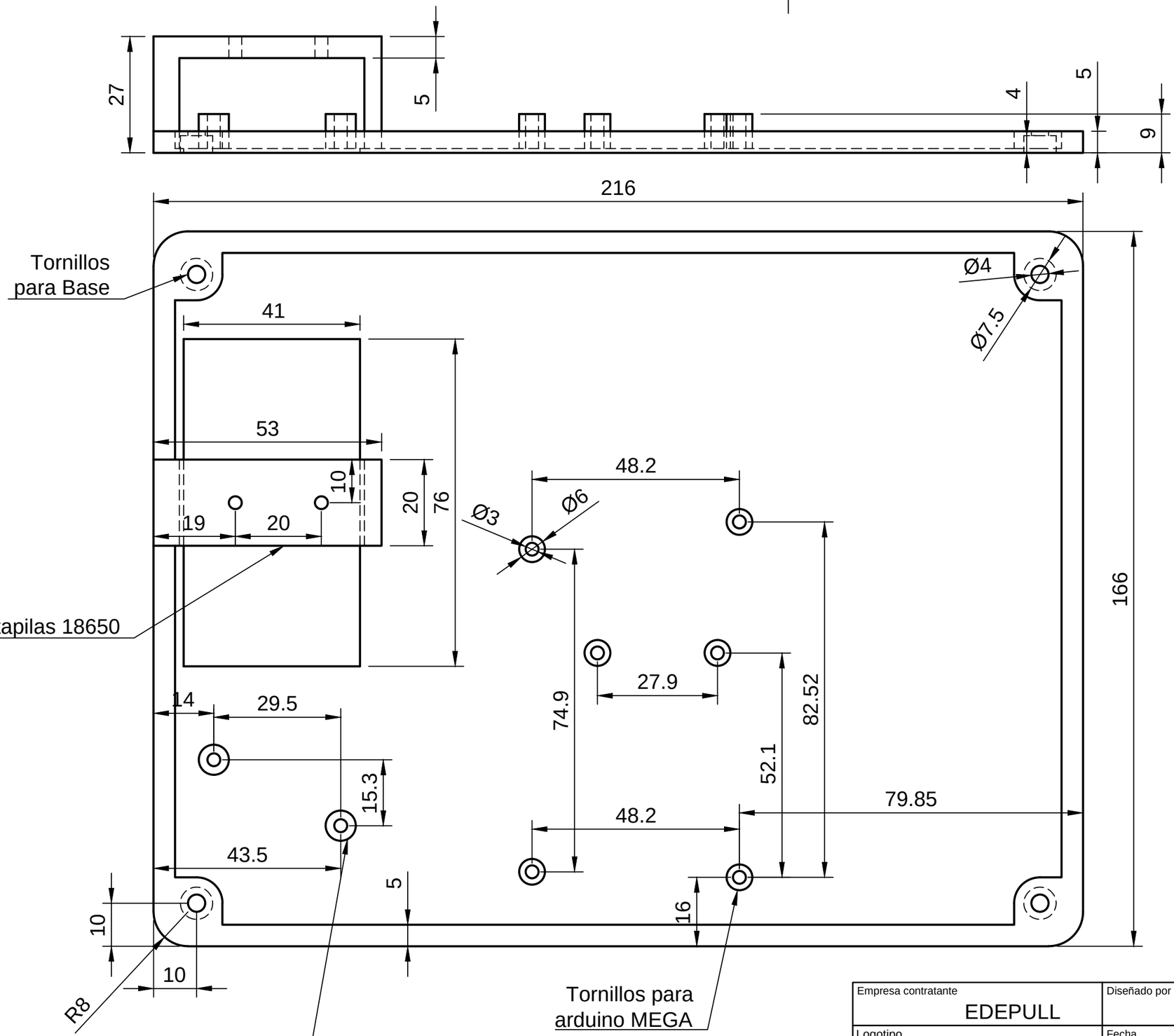
B-B (1:2)





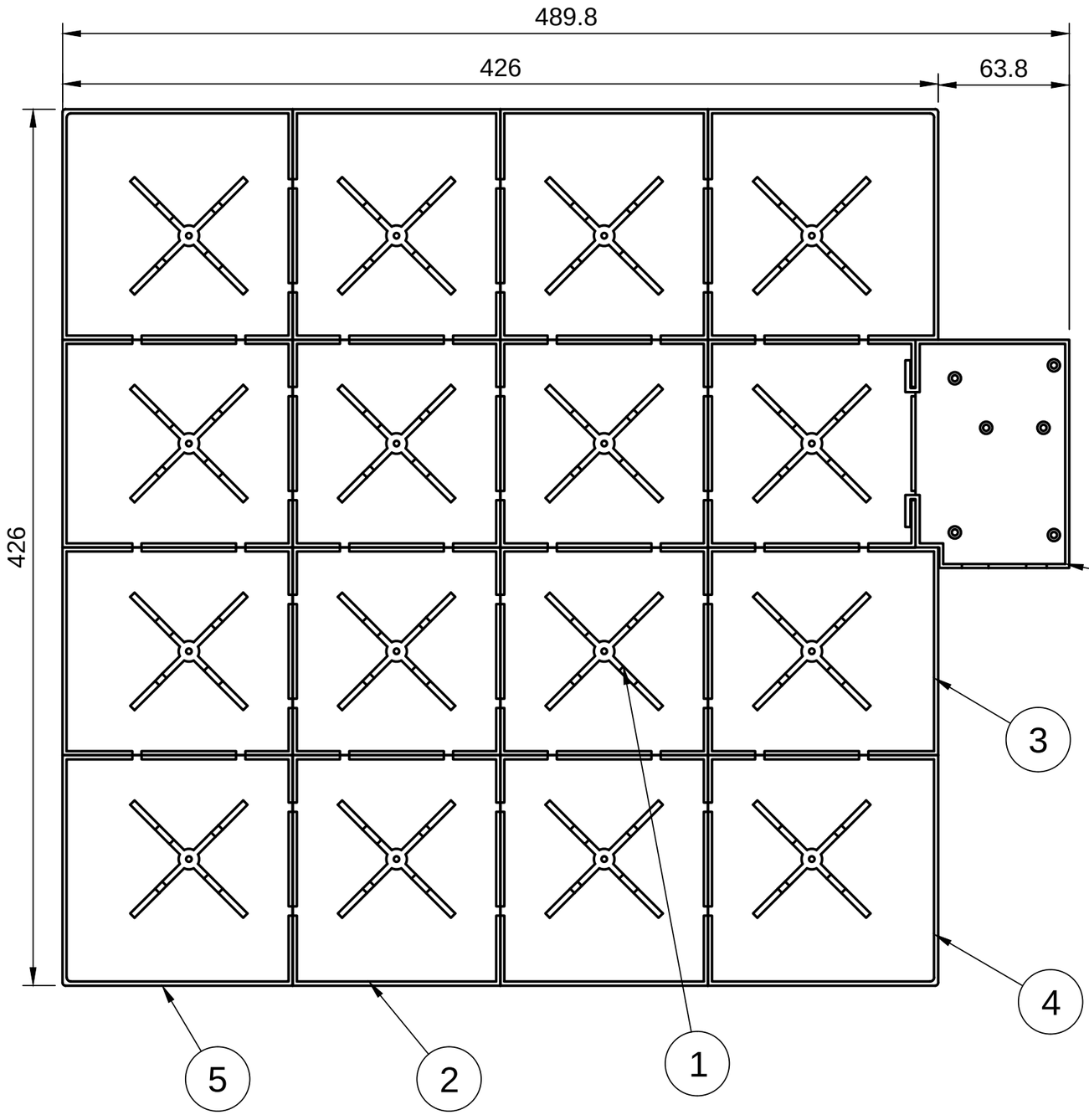
A-A (1:2)

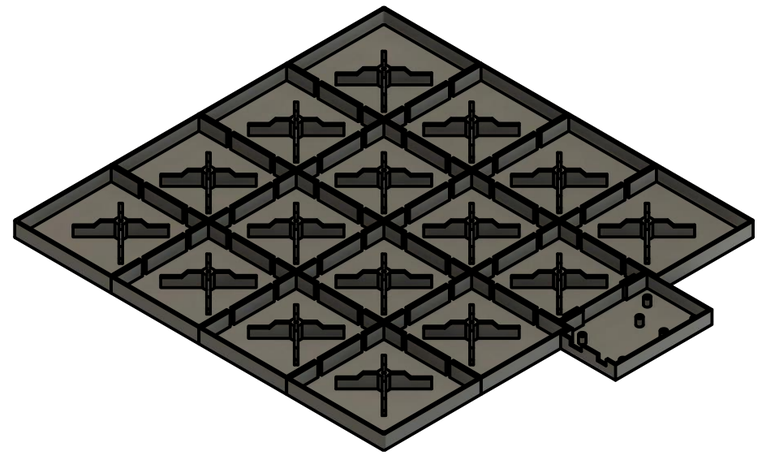
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Montaje Interruptor Jugadas
	Escala: 1:2
	Página: 5/14



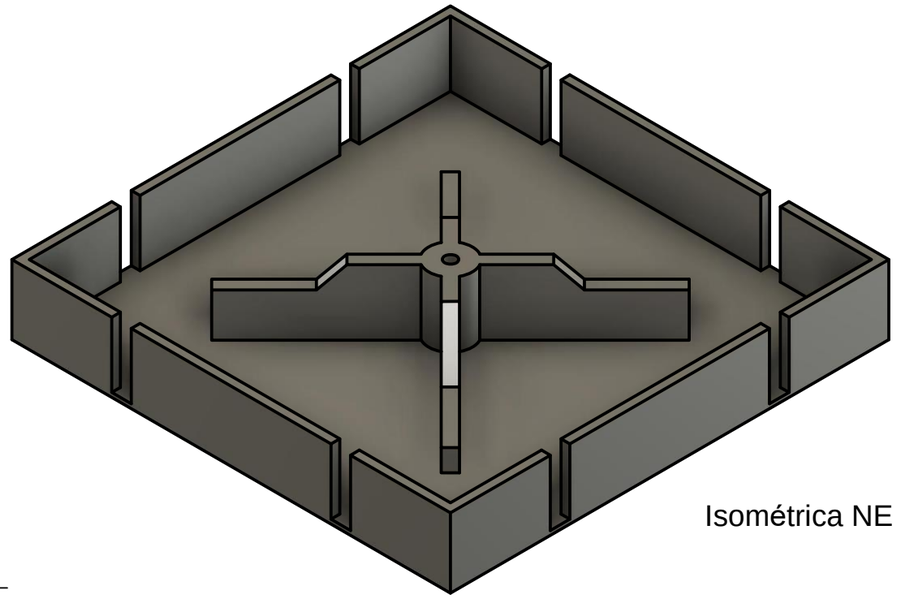
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
Título: Base Reloj Ajedrez Adaptado	Escala: 1:1
	Página: 6/14

Lista de piezas		
Elemento	ctd	Descripción
1	5	Base centro
2	4	Base lateral letras
3	3	Base lateral números
4	2	Base esquina casilla blanca
5	2	Base esquina casilla negra
6	1	Módulo electrónica

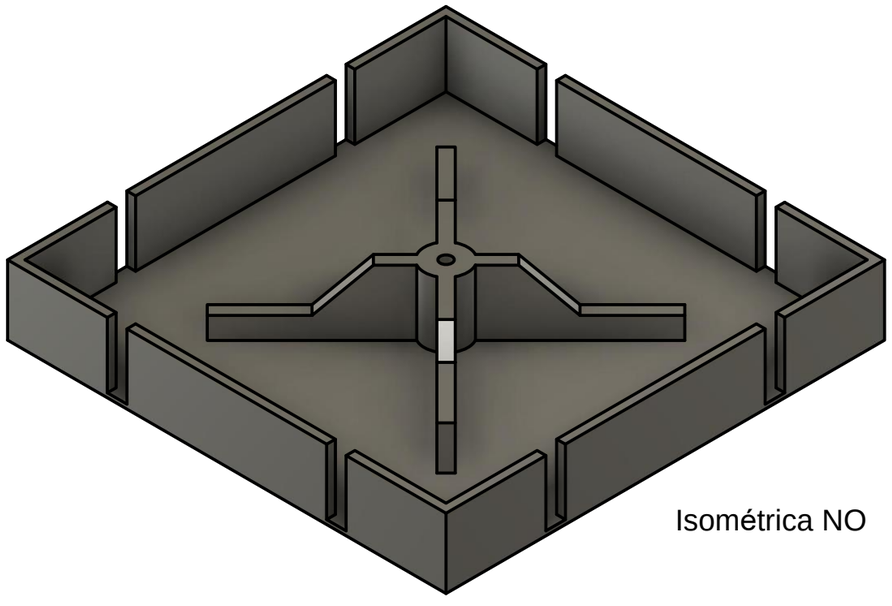




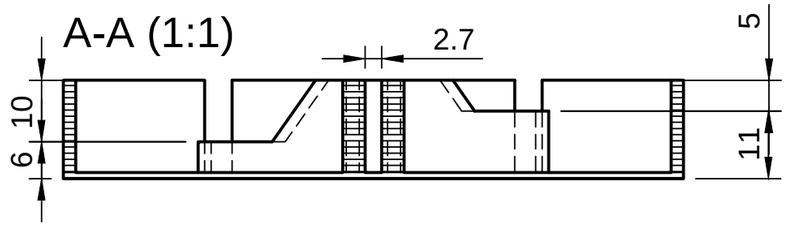
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
Título: Ensamblaje Base Tablero	Escala: 1:2
	Página: 7/14

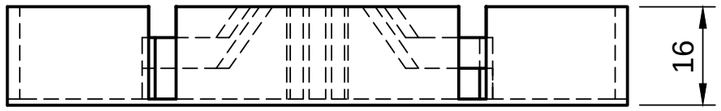


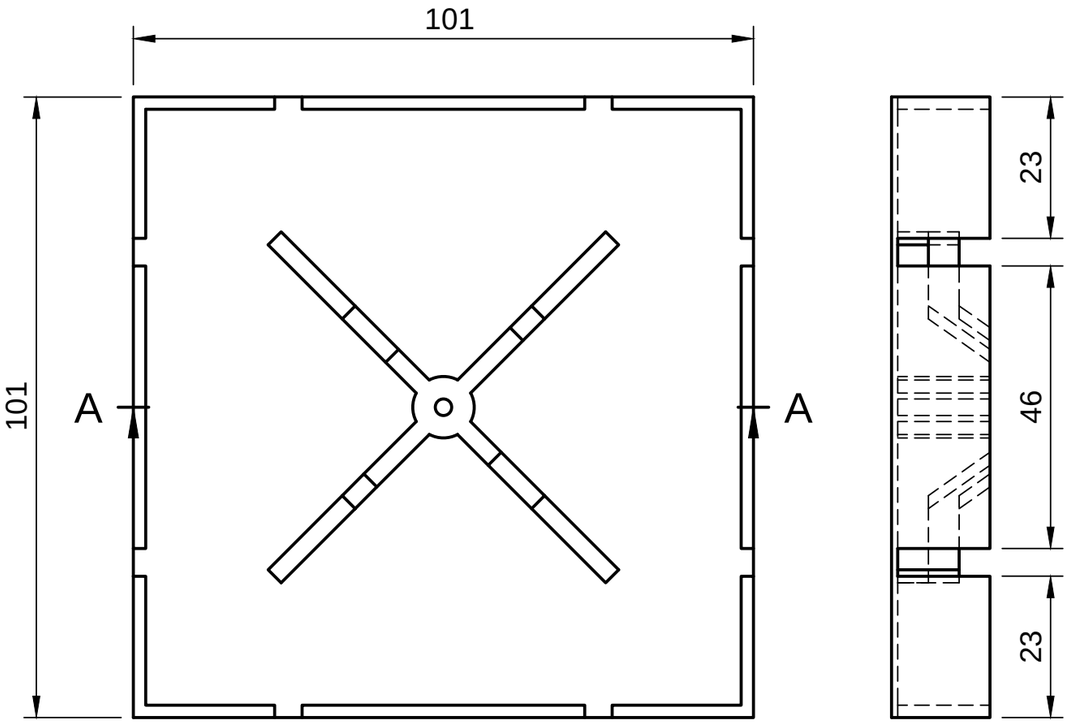
Isométrica NE



Isométrica NO

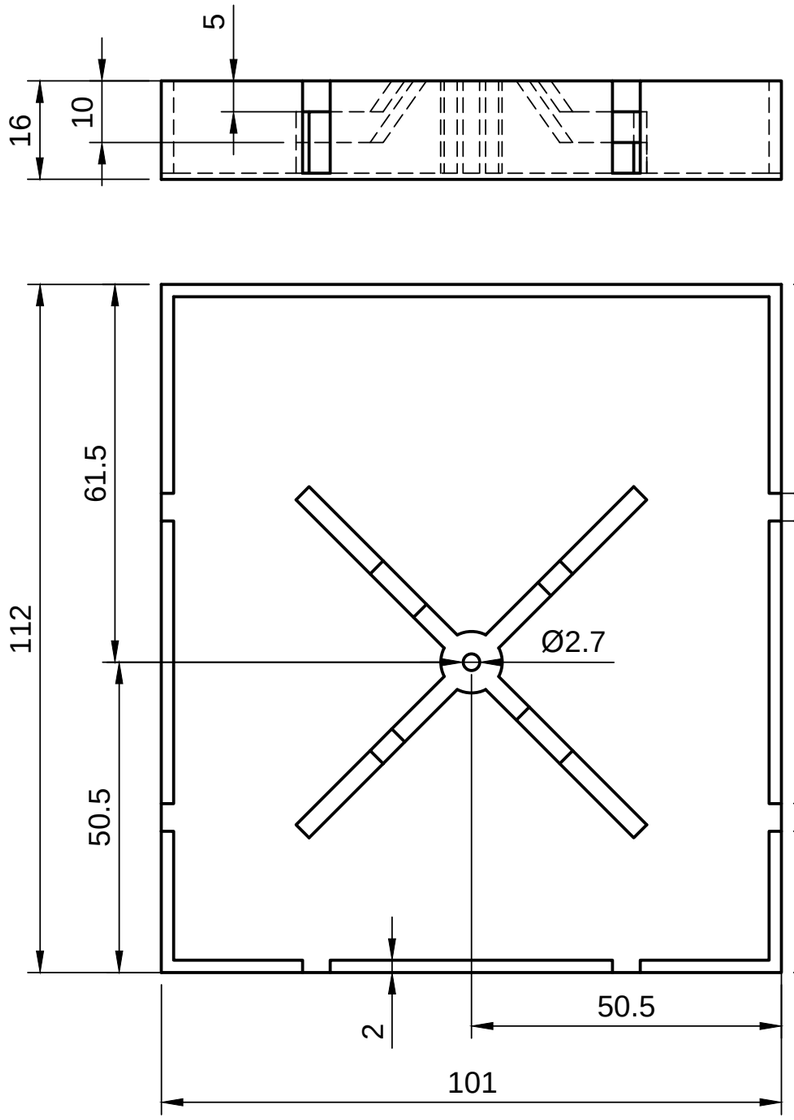


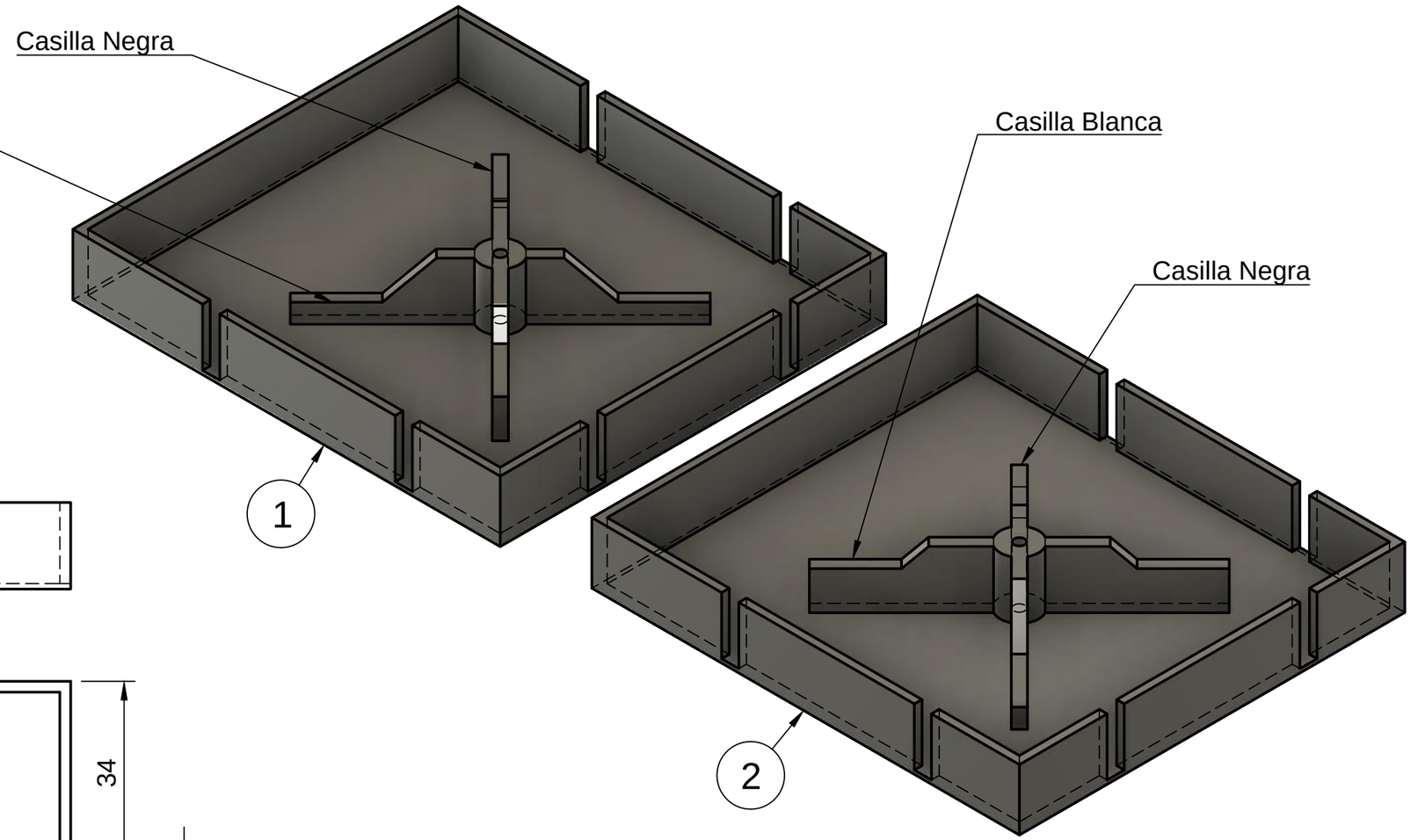




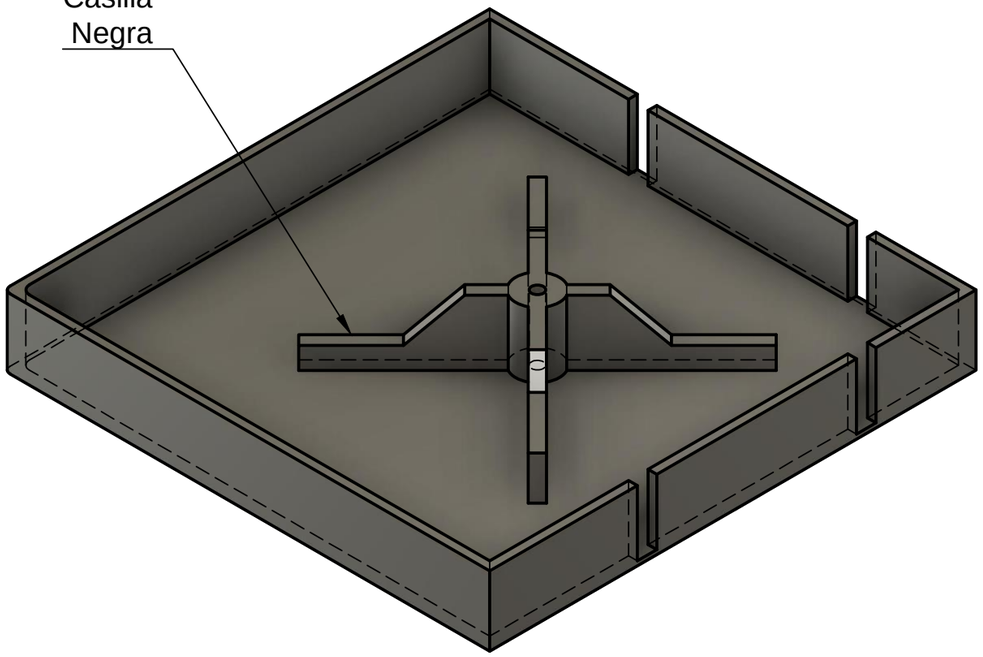
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Base centro
	Escala: 1:1
	Página: 8/14

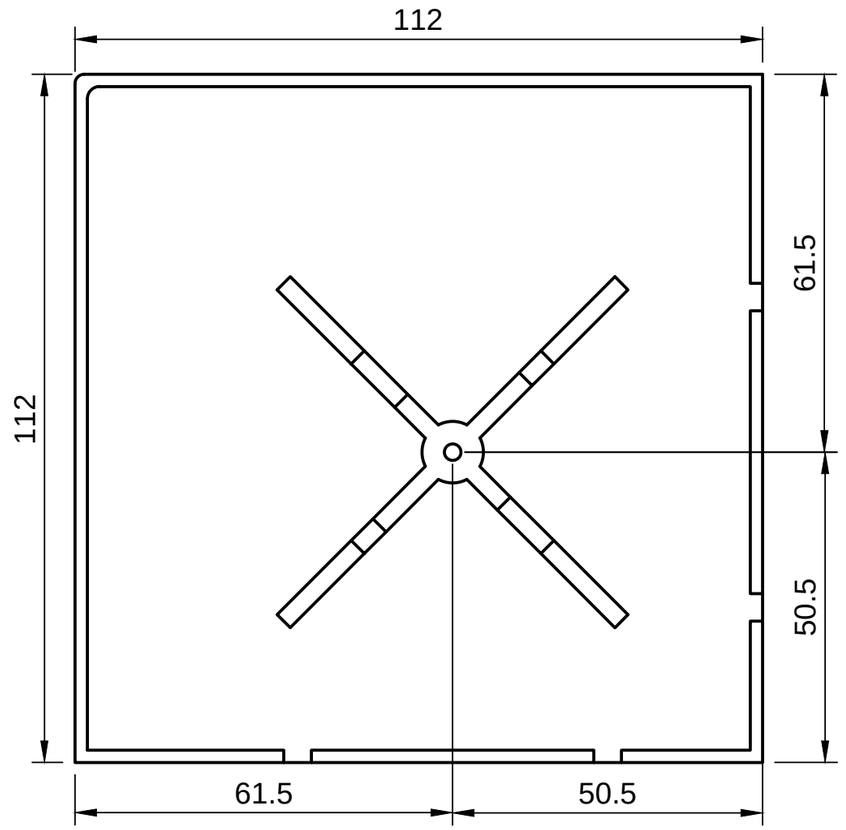
Lista de piezas		
elemento	ctd	nombre de pieza
1	1	Base lateral letras
2	1	Base lateral números

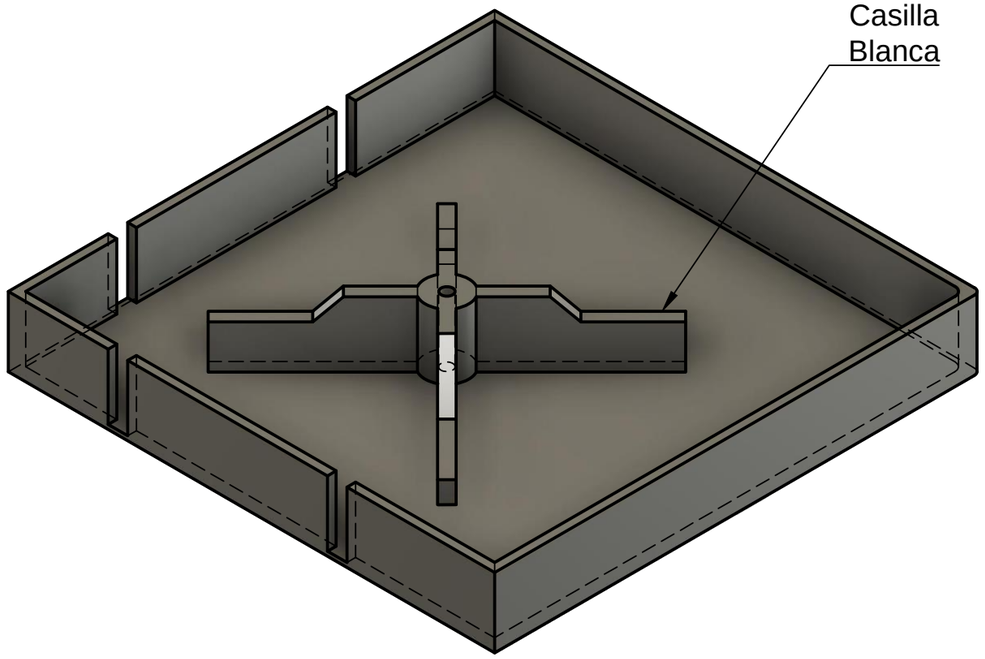




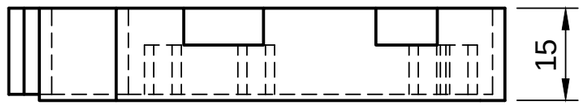
Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Laterales Base Tablero
	Escala: 1:1
	Página: 9/14

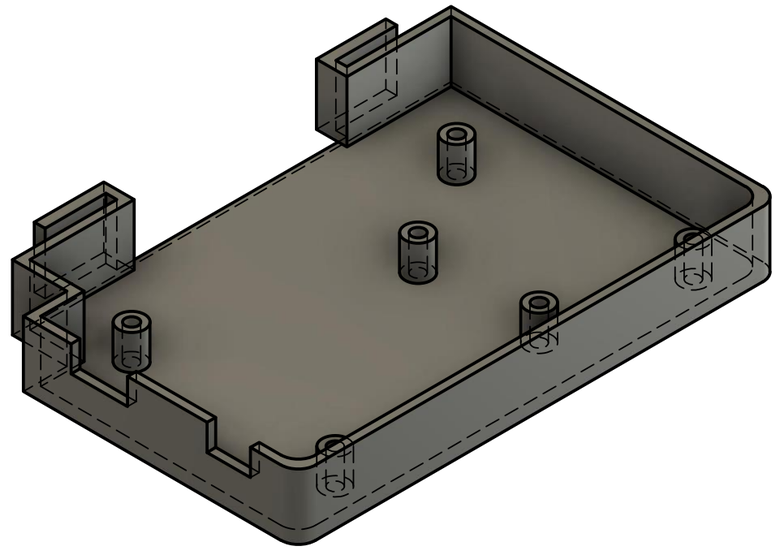


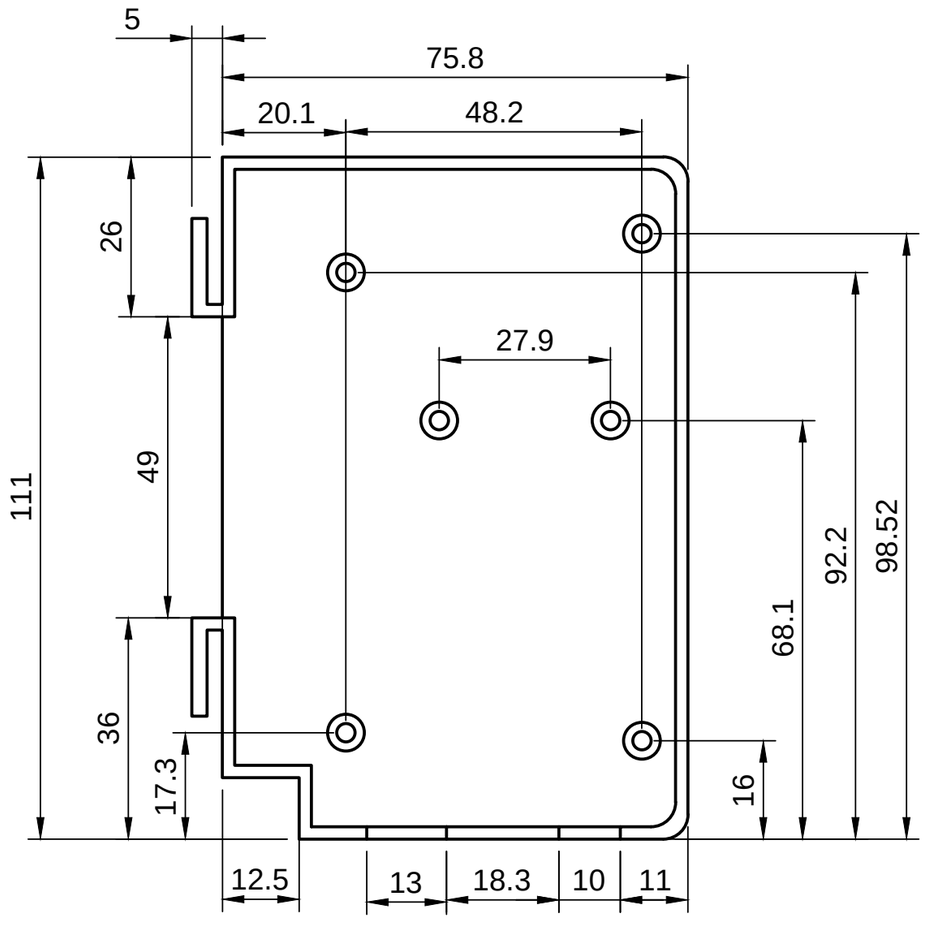


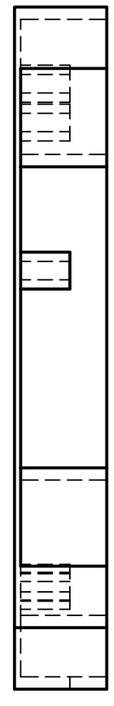


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
Título: Base Esquinas tablero casillas blancas y negras	Escala: 1:1
	Página: 10/14

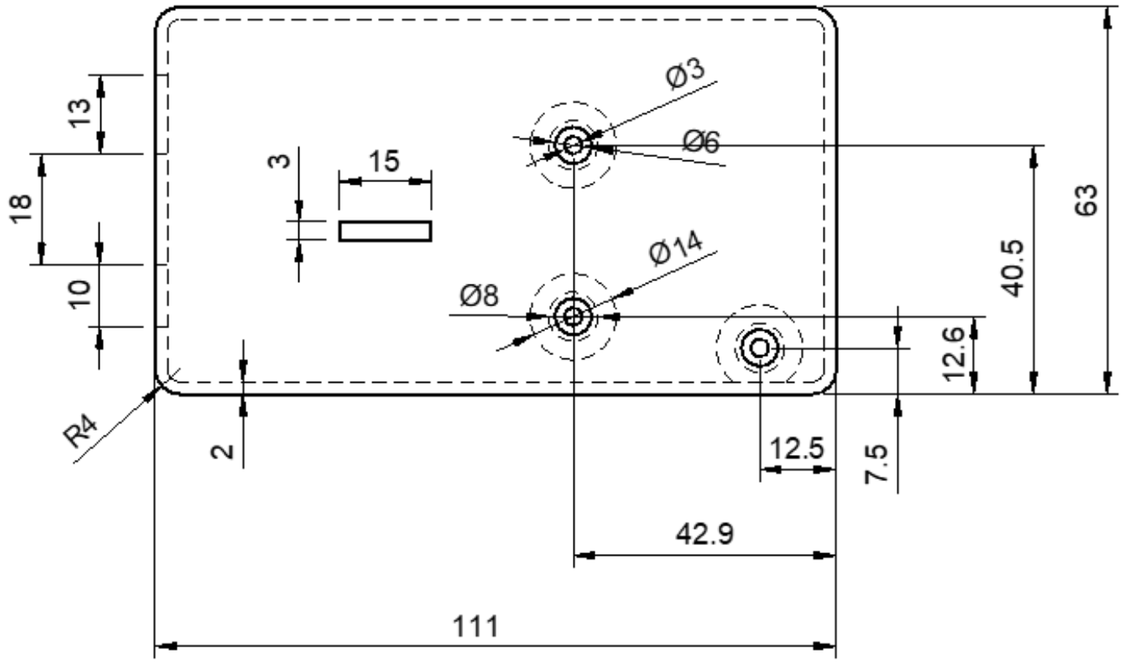


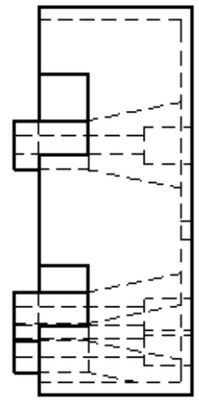


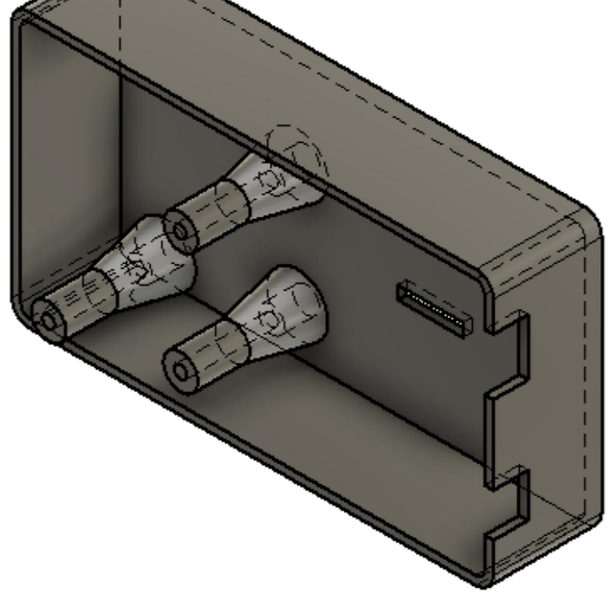


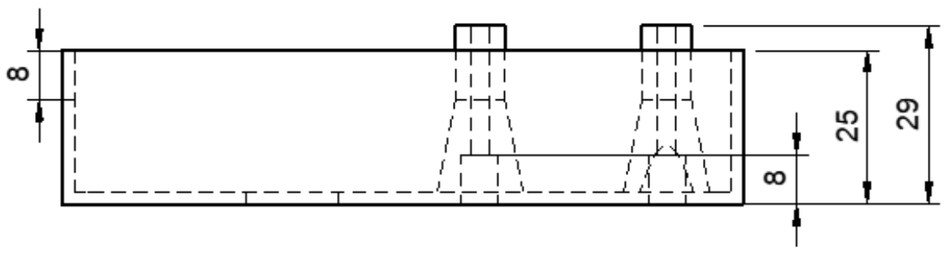


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Base módulo electrónica
	Escala: 1:1
	Página: 11/14



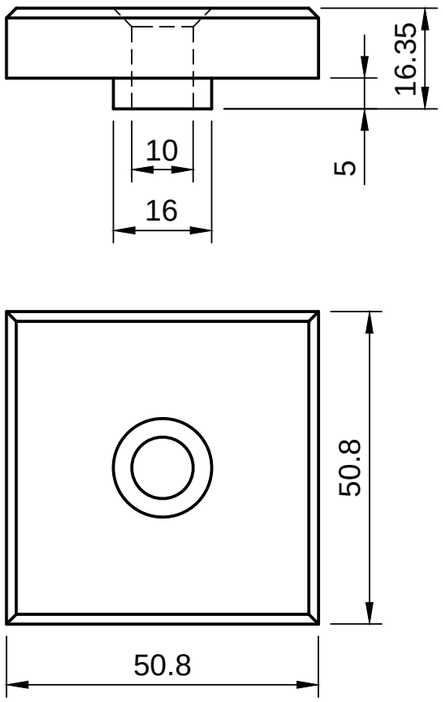


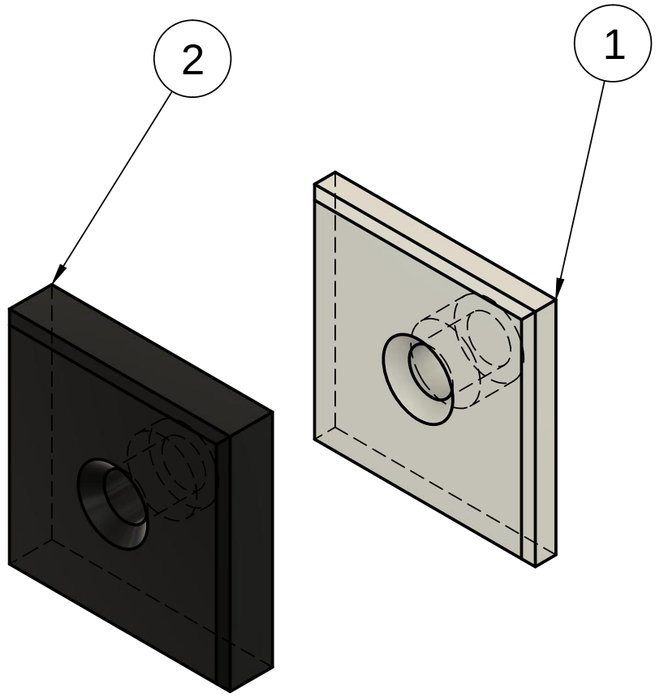


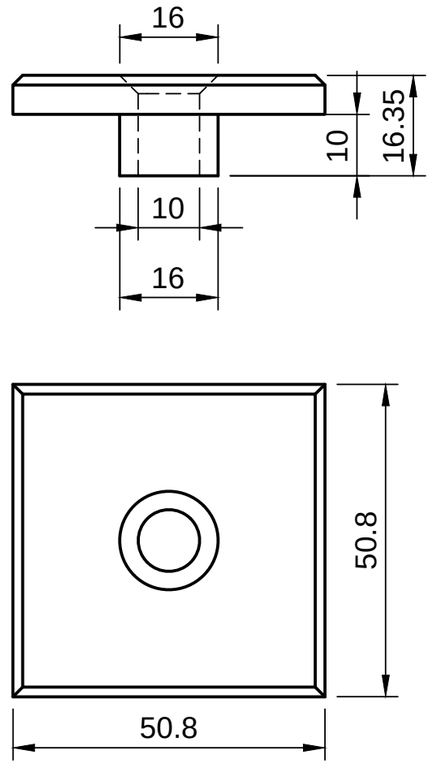


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Cubierta Módulo Electrónica
	Escala: 1:1
	Página: 12/14

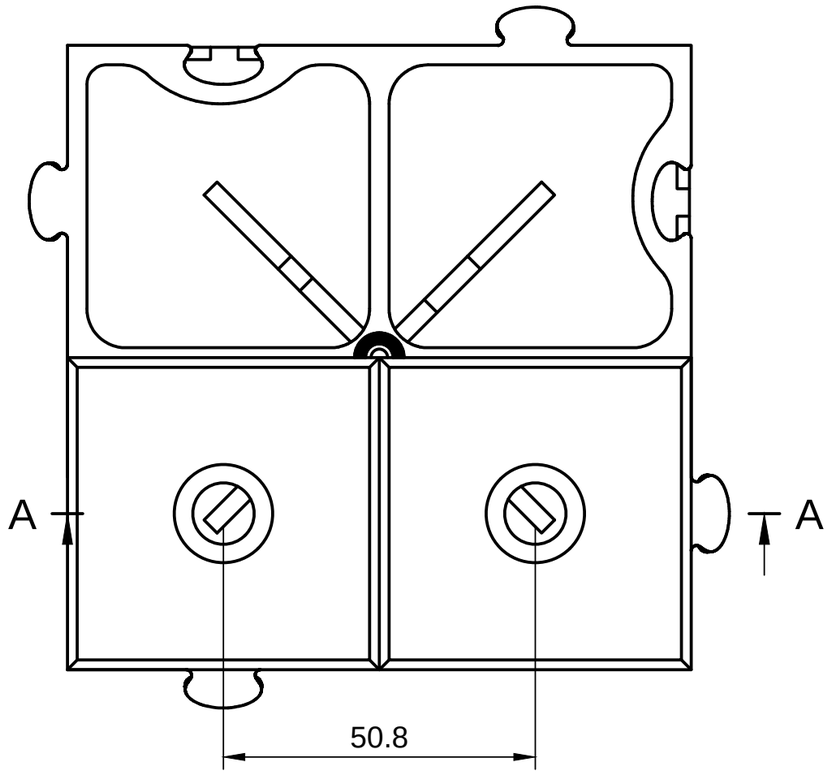
Lista de piezas		
elemento	ctd	nombre de pieza
1	1	Casilla blanca
2	1	Casilla negra

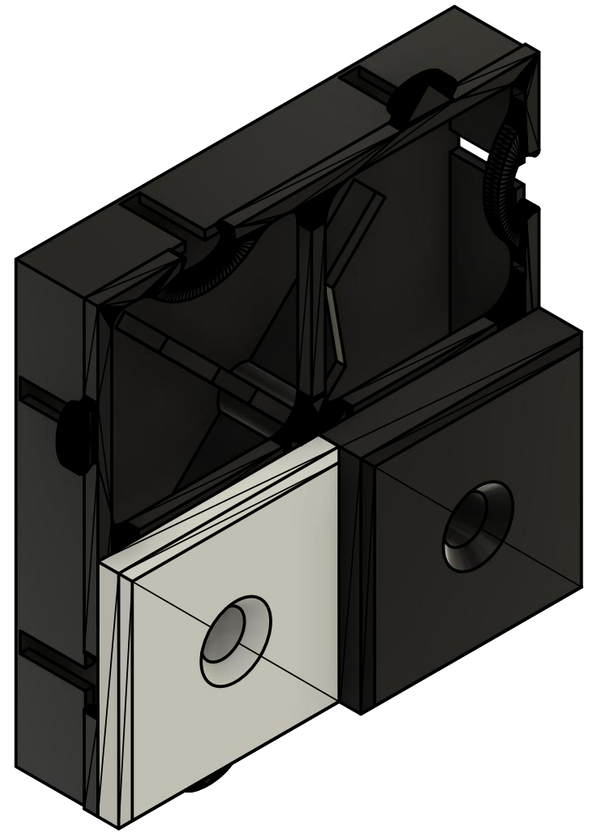




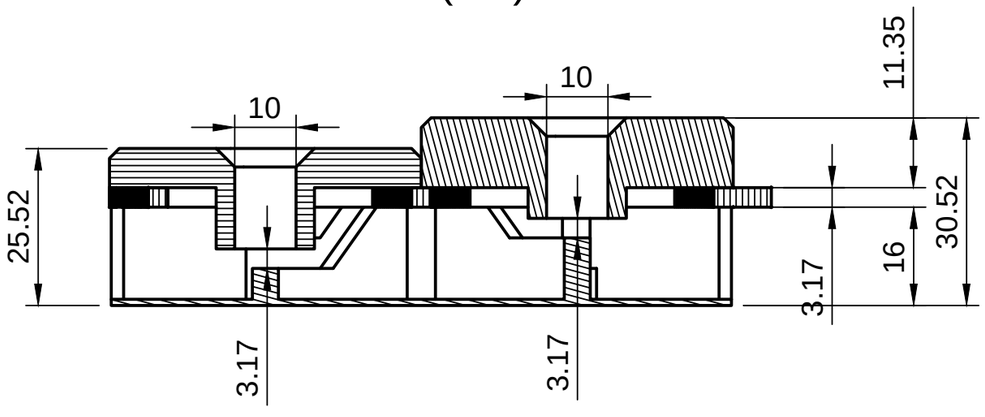


Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Casillas Negras y Blancas
	Escala: 1:1
	Página: 13/14





A-A (1:1)



Empresa contratante: EDEPULL	Diseñado por: Daniel Lorenzo López
Logotipo: Escuela de Doctorado y Estudios de Posgrado, Universidad de La Laguna	Fecha: 17/8/21
	Título: Ensamblaje tablero Ajedrez
	Escala: 1:1
	Página: 14/14