MEMORIA IMEI

# CLOCK SPEAKER

HAZ CLICK Y RECUERDA


**DESARROLLADORES**

ANA SIERRA
JAIME GIMENO
MIGUEL ALIENDE
PAULA CHUECA



UNIZAR | 2020

**INDICE**

## 1. OBJETIVO

Diseñar una aplicación móvil que ayude a las personas mayores, con problemas de memoria propios de la edad, o por sufrir una enfermedad como el Alzheimer, a seguir una rutina diaria.

El objetivo es crear un sistema de sensores en una casa, con un sensor en cada habitación, conectados por Wi-Fi a dicha aplicación.
Mediante un sistema de alarmas, se programará las actividades de la persona, bien sean rutinas para las comidas, horas de sueño, etc… o recordatorios para la toma de su medicación. Cuando sea la hora a la que está programada la alarma, la app emitirá un mensaje de voz, diciendo lo que tiene que hacer y hasta que no pase por la habitación correspondiente, se lo seguirá repitiendo con un intervalo de 5 minutos.

Además de su funcionalidad principal, la aplicación cuenta con juegos multimedia para entrenar la memoria y estimular la actividad cerebral.

## 2. ¿A QUIÉN VA DESTINADA ESTA APLICACIÓN?

El sistema entero se ha diseñado para personas mayores, de avanzada edad o personas con alguna enfermedad que implique problemas de memoria o para seguir una rutina. Sin embargo, puede usarla cualquier persona, tenga la edad que tenga, además puede funcionar sin los sensores para aquellos que quieran instalársela como un sistema alternativo a las alarmas normales, que incluyen los móviles por defecto.

## 3. PROTOTIPO DE LA APLICACIÓN.

Se ha desarrollado un prototipo de la aplicación. La app actual cuenta con único sensor, y no funciona en segundo plano. Esto no supone ningún problema para su desarrollo completo posterior, puesto que se corregiría añadiendo el resto de sensores y especificando las direcciones de cada uno y la habitación correspondiente, y para que funcionase en segundo plano, habría que hacer un servicio nativo de Android Studio que se comunicase con la aplicación.

## 4. HERRAMIENTAS UTILIZADAS.

### 1. Entorno de desarrollo

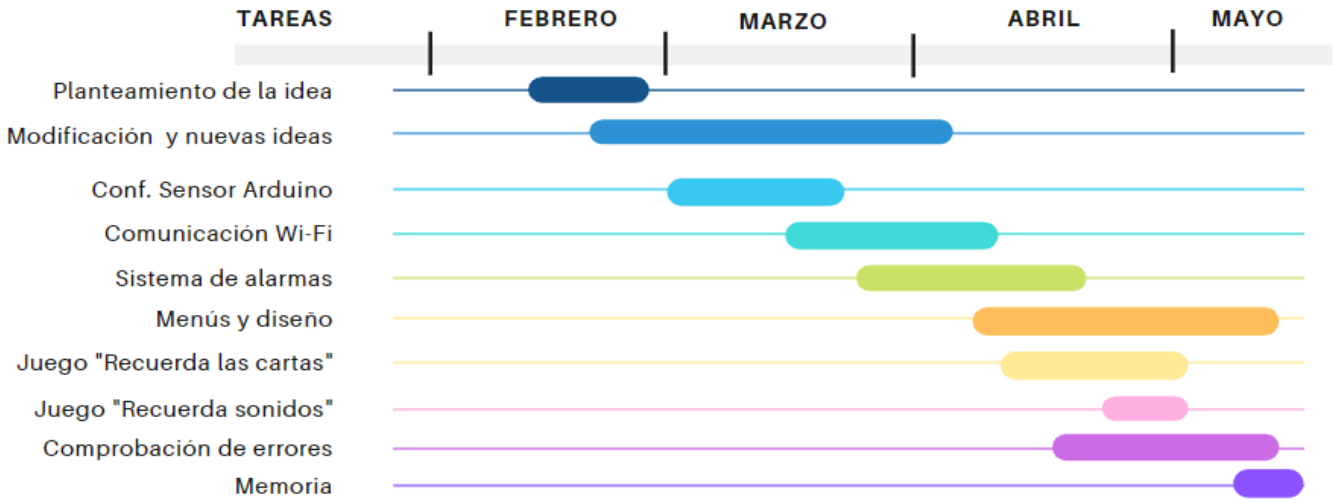
Se ha elegido Unity como plataforma para la creación de la aplicación.
Unity es un motor de videojuego multiplataforma disponible para Windows, Mac OS y Linux. Permite compilar la aplicación en diferentes plataformas, entre las que se encuentra el soporte de aplicaciones móvil Android, eso sumado a la sencillez a la hora de diseñar la interfaz de usuario ha sido el motivo de su elección. El lenguaje usado para la programación es C#.

### 2. Android Developer SDK & NDK Tools.

Para poder exportar la aplicación de Unity a Android es necesario instalar estas herramientas externas, Android SDK, Java Development Kit (JDK) y NDK tools.

## 5. PLANIFICACIÓN Y REPARTO DE TAREAS

### 1. Planificación.




### 2. Reparto de tareas.

**Ana Sierra:** Programación de las alarmas y gestión de peticiones HTTP, programación de los juegos de memorizar parejas y recordar sonidos.

**Jaime Gimeno:** Configuración del sensor, programación del ESP-32 para comunicarse con la App y Reproductor de texto a voz.

**Miguel Aliende:** Parte grafica de la aplicación, navegación por los menús de la aplicación y opciones de personalización (cambio de fondo de pantalla).

**Paula Chueca:** Presentación en PowerPoint y juego de memorizar parejas.

## 6. DESARROLLO DE LA APLICACIÓN.

### 1. Estructura de la aplicación.

En Unity se puede dividir todo en escenas, siendo cada escena una de las pantallas que vemos. En este caso tenemos 7 escenas, que son las siguientes:

- Panel Alarmas
  Esta escena contiene todos los scripts y los objetos relacionados con el sistema de alarmas, está formado por un sistema de Canvas dónde se encuentran los menús de la aplicación. Se trata de la escena principal de la aplicación desde la que podremos acceder a la configuración de las alarmas, los juegos y las opciones de personalización.
- Juego 1.1
  Es el primer nivel del juego “Recuerda las parejas”. Nivel fácil.
  Consta de 8 cartas, hay que encontrar 4 parejas.
- Juego 1.2

Nivel intermedio del juego “Recuerda las parejas”.
Consta de 12 cartas, hay que encontrar 6 parejas.

- Juego 1.3
  Nivel difícil del juego “Recuerda las parejas”.
  Consta de 16 cartas, hay que encontrar 8 parejas.
- Juego 2.1
  Primer nivel, siendo el nivel fácil del segundo juego “Recuerda los sonidos”.
  Se oyen 3 sonidos.
- Juego 2.2
  Nivel intermedio del juego “Encuentra los sonido”.
  Se escuchan 5 sonidos.

- Juego 2.3
  Nivel difícil del juego “Encuentra los sonidos”.
  Hay que recordar 7 sonidos.

Cada juego está constituido por tres niveles en los que se varía su dificultad

## 2. Menú Principal

### 1. Explicación

Se trata del Canvas principal, desde aquí podemos elegir si ir a la configuración de alarmas, a los juegos o los ajustes de personalización. Las transiciones entre un menú y otro por medio de los botones hemos hecho que sean mediante la activación y desactivación de diferentes Canvas, porque es más rápido y fluido que hacer cambios de escenas. Todos estos Canvas que forman el menú están colgados de un Canvas principal con los parámetros de *Canvas Scaler* configurados para que haga correctamente el escalado de pantalla y se ajuste a diferentes dispositivos.

*Fig. 1 Menú principal*

Los menús constan principalmente de botones, que al pulsar cada uno de ellos desactivas el Canvas en el que se encuentran y a la vez activan el Canvas correspondiente, alguno de ellos tiene otras acciones específicas como guardar o eliminar datos, de lo que ya hablaremos más adelante. Para hacer los que los botones sean visualmente más atractivos que los predeterminados de Unity hemos utilizado una aplicación externa de diseño.

## 3. Ajustes

### 1. Explicación

Para hacer la aplicación algo más completa hemos añadido ajustes de personalización, podremos cambiar el fondo de pantalla de la aplicación.

Para que esto se haga en todos los menos lo que hemos hecho es que el Canvas General que contiene el resto de Canvas tenga una imagen, cada una de las fotos que se ven en la selección de fondos de pantalla son botones que al pulsarlos se sustituirá la imagen del Canvas General por la imagen seleccionada.

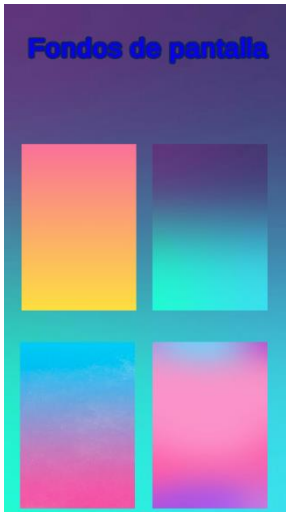



*Fig. 2 Ejemplo fondos 1*

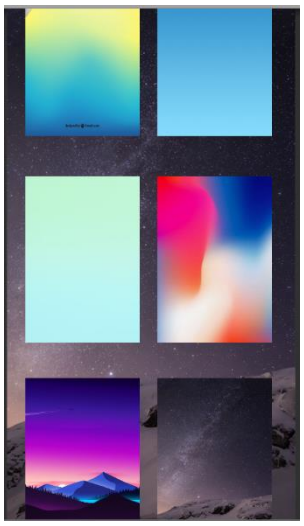
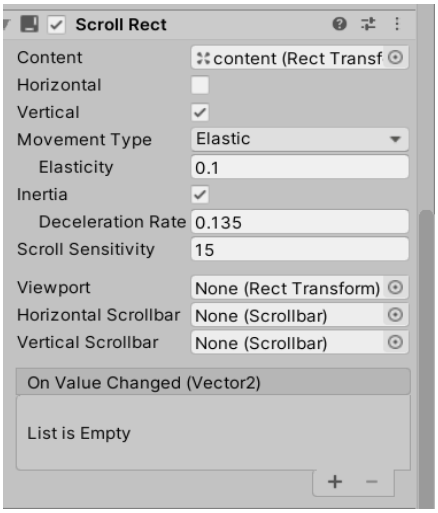


*Fig. 3 Ejemplo fondos 2*

Para hacer el desplazamiento vertical dentro del Canvas hemos creado un objeto *content,* de aquí hemos colgado todas las imágenes. En el Canvas de ajustes hemos creado un componente tipo *ScrollRect*y la hemos configurado de la siguiente manera:




En el campo Content asignamos el objeto mencionado antes que contiene todos las fotos y botones.
Hemos decidido que el tipo de movimiento sea “elástico” porque es a lo que estamos acostumbrados en la mayoría de las aplicaciones móviles.

## 2. Programación

### Script fondo

Para programar el cambio de fondo hemos creado un array de sprites, para ello hemos guardado todas las imágenes dentro de un carpeta en el directorio *Resoruces,* y las has hemos enumerado, de tal manera que asignamos cada imagen a un entero, de tal manera que al pulsar un botón u otro se modificara el valor del entero indexado en el array de sprites y por lo tanto se cambiara la imagen.

## 4. Panel Alarmas.

### 1. Explicación

Consta de 6 alarmas. La hora se introduce siguiendo la estructura 00:00, y se puede poner cualquier frase. Cuando llega la hora se reproducirá el mensaje y hasta que el sensor no responda cuando ha detectado un movimiento, la alarma se repetirá en un intervalo de 5 minutos.


*Fig. 5 MenúAlarmas*


*Fig. 6 Ejemplo Alarma*

## 2. Programación

Un script general llamado *Guarda_datos* se encarga de guardar las horas y el contenido de cada alarma, cuando se pulsa el botón guardar correspondiente.

*Eliminar_alarmas* es un script general, que borra el contenido y hora de la alarma cuando se pulsa el botón "Eliminar" de la alarma correspondiente.

Cada alarma tendrá un *script ctr_alarma*, que detecta si la hora del dispositivo coincide con la hora de esa alarma, o si hay una alarma ejecutándose; manda hacer las peticiones al sensor para comprobar si ha pasado alguien por ahí y reproduce el mensaje.

Todas las alarmas tienen su propio *script Peticion* que consulta al arduino mediante mensajes ***get***, si ha detectado movimiento. Si la respuesta es negativa, indica que se siga reproduciendo la alarma cuando toque, si la respuesta es afirmativa interrumpe la reproducción de la alarma.

***Script Guarda_datos:***

Clickar(): hay una función *Clickar*por cada alarma. Está asociada al botón guardar correspondiente a cada una. Cuando se ejecuta, se guarda en un *PlayerPrefab* el contenido de la alarma, y en otro*PlayerPrefab* la hora de la alarma. De esta manera están disponibles entre sesiones y es accesible desde cualquier escena de la aplicación.

***Script Eliminar_alarmas:***

Eliminar(): el script contiene un método de este tipo para cada una de las alarmas, y se actualiza el valor de los *PlayerPrefabs*a *Null*, cuando se pulsa en el botón Eliminar.

***Script Crt_alarma:***

Update(): es un método que se está ejecutando constantemente, cada frame, y comprueba si la hora del dispositivo coincide con la hora que tiene programa esa alarma, o si dicha alarma ha sido ejecutada antes, han pasado 5 minutos y aun se sigue esperando una respuesta afirmativa del sensor. En caso que una de las dos sea cierta, se manda seguir haciendo peticiones web al sensor, y se llama a la rutina *DownloadTheAudio()*.

DownloadTheAudio(): hace una petición a la url del traductor de google, que se encarga de convertir el texto en voz. El texto que le pasamos es el contenido de la alarma. Hemos encontrado esta función mucho más sencilla que algunas otras librerías para Unity.

***Script Peticion:***

Update(): comprueba si tiene que empezar a hacer peticiones porque hay una alarma, y si no hay ninguna petición en curso, en caso afirmativo llama a la rutina *UsingYield()*.

UsingYield(): actualiza que se está ejecutando una rutina para hacer peticiones web, llama a dicha rutina *GetText()* y espera unos segundos para que no se superpongan peticiones.

GetText(): hace una petición web a la url del arduino. Evalúa la respuesta recibida por su parte, si no ha detectado movimiento, sigue haciendo peticiones y la alarma volverá a sonar. Si ha detectado movimiento no hace mas peticiones y "apaga" la alarma.

## 5. Juego "Recuerda las parejas"

### 1. Explicación del juego.

Un juego para ejercitar la memoria basado en cartas, con imágenes de la vida cotidiana, como animales, objetos de deporte, flores etc. Estimulan la memoria visual y aparte de entrenar la memoria por el funcionamiento del propio juego, se han usado imágenes que puedan evocar recuerdos y asociaciones con su vida.

Inicialmente aparecen todas las cartas boca-abajo, y es el jugador quien tiene que ir pulsando de una en una intentando encontrar su pareja, que es una carta idéntica a ella. Si no se acierta se vuelven a poner boca-abajo esas dos cartas, de esta manera poco a poco irá viendo dónde están situadas y tendrá que recordar la posición de cada una para poder emparejarlas. Cada vez que

acierte se sumará 1 al contador de parejas. Si se completa el juego, pasa automáticamente al siguiente nivel.

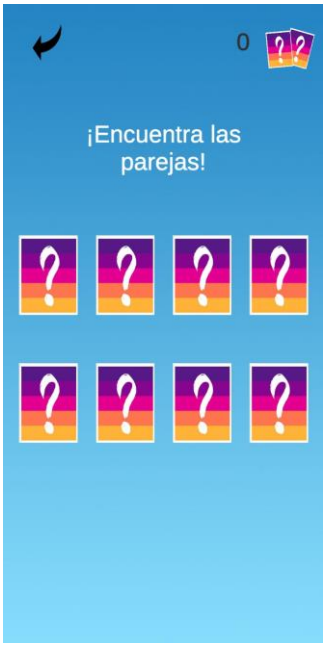



*Fig. 6 Inicialmente nivel 1*

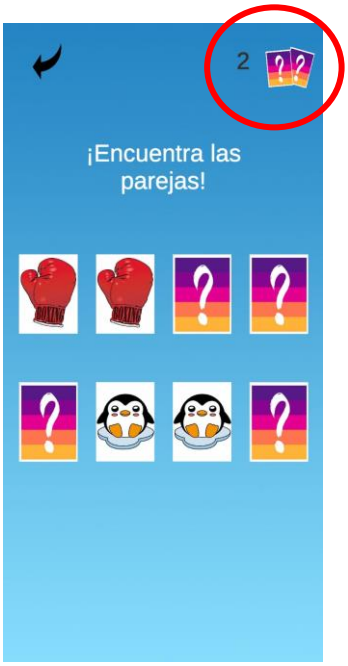



*Fig. 7 Al encontrar parejas*


*Fig. 8 Nivel 2*


*Fig.9 Nivel 3*

## 2. Programación

Consta de dos scripts, para controlar el funcionamiento del juego.

El objeto *carta*, está formado por dos sprites, la carta con la imagen y encima de ella, la parte trasera de una carta, para que no se vea el contenido.

Mediante los *Script Carta* y *SceneCtr*, se controla cuando se visualiza una u otra.

- ***Script Carta*:**
  Este script controla el funcionamiento de la carta, implementando los siguientes métodos:

  OnMouseDown() : detecta cuando se pulsa sobre la carta, para que se pueda ejecutar este método es necesario añadirle un collider a las cartas. De esta manera cuando se pulsa sobre ella, la parte superior se hace invisible y se revela el contenido de la carta.

  Id() : este método nos devuelve el id de la carta, que es necesario asignarle para saber cuál es, de esta manera cuando dos cartas tengan el mismo id es que son pareja.

  ChangeSprite() : se llama a esta función desde el *SceneCtr*, las cartas se crean dinámicamente, y en función del id que le toca a la carta, le corresponde un script.

  Unreveal() : también se llama a esta función desde el *SceneCtr*, cuando no se detecta parejas, se llama a este método para "darle la vuelta a la carta" y ver otra vez la parte trasera.

- ***Script SceneCtr:***
  En este script se controla todo el juego y la creación de la escena.

  Start(): es un método implementado por Unity , que se ejecuta en cuanto entramos en la escena y es lo primero que hace. Dentro de este método esta todo el código que crea el resto de cartas dinámicamente basándose en una **carta base** con el *script Carta* que tiene asociado.

  En un vector de longitud ***n*** siendo ***n = número de cartas que hay en el nivel***se declara los posibles id para las cartas, teniendo en cuenta que tienen que estar repetidos de dos en dos para poder ser pareja. Este vector se desordena pasándoselo a *ShuffleArray(),* así las cartas se crean aleatoriamente y cada vez estarán en una posición.

  Se recorre en un bucle el vector, se crean las cartas y se llama al método *ChangeSprite()* pasándole su id, para que le asigne el sprite correspondiente.

  Anteriormente se declara las variables indicando el número de filas y columnas, y la distancia con respecto a ***x*** e ***y*** que tendrá cada carta y se posicionan según corresponda.

ShuffleArray(): recibe como parámetro de entrada un vector y lo desordena aleatoriamente.

CardRevealed(): se llama a esta función desde el *script Card*, cuando se detecta que se ha pulsado una carta. Se comprueba si hay una carta revelada antes, si es así se guarda esta como la segunda y se llama al método*ChekedMatch()* para ver si son pareja. Si no, se pone esta carta como la primera revelada y se espera a una segunda.

CheckedMatch(): comprueba si las dos cartas seleccionadas tienen el mismo id, en ese caso se suma 1 a la puntuación y se dejan las dos cartas reveladas. Si no, se espera 0.5 s y se les da la vuelta llamando al método *Unreveal()*.

El código es igual para los 3 niveles, se varia el número de cartas que tiene que crear, la posición en la que deben estar, se añade más sprites..etc, pero es el mismo funcionamiento.

Es muy importante para que el juego tenga sentido, que las cartas cambien de posición en cada partida, puesto que si no, estarían siempre en el mismo sitio y no entrenaría la memoria.

## 6. Juego “Recuerda los sonidos”

### 1. Explicación del juego

Es un juego para entrenar la memoria auditiva. Consiste en una sucesión de sonidos, de forma que el jugador tiene que recordar el orden en el que han aparecido.

Al igual que en el juego de las cartas, los sonidos proceden de objetos cotidianos, que forman parte de la vida de la persona. De esta manera se pretende una mayor facilidad para asociarlos, y que le sirvan en su vida diaria.
Así, aparte del entrenamiento propio del juego, los sonidos pueden despertar recuerdos de manera inconsciente en la persona.

El funcionamiento del juego es muy sencillo, en cada escena cuando el usuario esté listo le da al botón “Empezar” y se reproducen los sonidos, relacionados con las imágenes que aparecen. Una vez que han acabado, el jugador pulsa sobre las imágenes en el orden en el que los ha escuchado, y le da a “Comprobar”. Si es correcto, aparecerá un tick verde, si no, una cruz roja y se repetirá el proceso, con los sonidos en el mismo orden.

En cada partida se reproducen aleatoriamente para que el orden sea distinto.


*Fig. 10 Nivel 1 Inicialmente*


*Fig. 11 Nivel 1 acertado*


*Fig. 12 Nivel 2 no acertado*

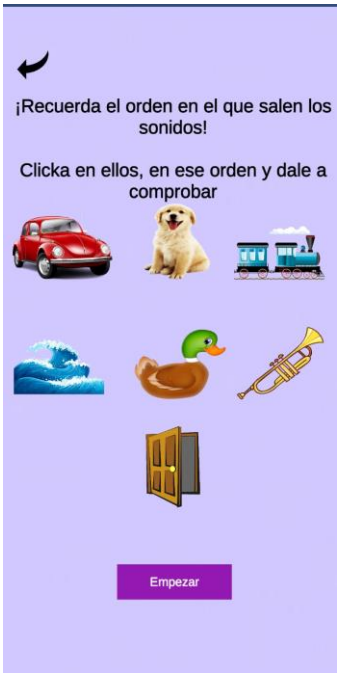



*Fig. 13 Nivel 3*

## 2. Programación

Cada uno de los sonidos, tiene una imagen asociada que se controla con un *script* propio, para detectar cuando se pulsa sobre ella. La escena completa se controla con el *Scrip Ctr_juego2.*

***Script "Imagen":***

El *GameObject* de cada imagen, está formado por dos sprites, que son idénticos con la diferencia que uno es más grande que otro. Ambos sprites están superpuestos al igual que en el juego de las cartas, de manera que, al seleccionar la imagen, se agranda durante un segundo y luego vuelve a la posición inicial. Esto se hace para que al pulsar sobre la imagen el jugador reciba una respuesta visual de que se ha detectado correctamente la pulsación.

Los métodos que lo controlan son los siguientes:

Start(): nada más empezar hace invisible el sprite de la imagen agrandada.

OnMouseDown(): detecta cuando se pulsa sobre la imagen, hace visible el sprite correspondiente a la imagen grande y llama a la corutinaUsingYield() para esperar un segundo y volver a poner el sprite pequeño. Si el número de objetos guardados es menor que el máximo de objetos disponibles, y ya se ha terminado de reproducir los sonidos, entonces se guarda en un vector global el id correspondiente del sonido de esta imagen.

UsingYield(): una co-rutina que espera los segundos que se le pase como parámetro antes de realizar una acción.

***Script Ctr_Juego2:***

Se controla la escena del juego, reproduciendo los sonidos de forma aleatoria y se encarga de comprobar si el orden marcado por el usuario es el correcto.

Start(): se ponen invisible los sprites de las imágenes grandes, y se hace visible el botón "Empezar".

Update(): este método se ejecuta cada frame. En su código se comprueba si se ha pulsado el botón "Empezar", en ese caso llama a la rutina *UsingYield()*que reproduce los sonidos.

Si en lugar de "Empezar" se ha pulsado "Comprobar", compara el vector en el que ha guardado el orden marcado por el usuario, con el vector que guarda el orden de los sonidos, si coincide aparece un tick verde y activa el botón replay. Si no coincide, aparece una cruz roja y vuelven a reproducirse los sonidos.

ShuffleArray(): desordena un vector, se usa para que los sonidos se reproduzcan cada partida aleatoriamente.

UsingYield(): recorre el vector de los sonidos, según el id en cada posición reproduce el sonido correspondiente. Cuando ha terminado hace visible el botón "Comprobar".

Change(): método asociado al botón “Next”, sirve para pasar al siguiente nivel del juego.

ClickEmpezar(): asociado al botón “Empezar”.

ClickComprobar(): asociado al botón “Comprobar”, una vez pulsado hace invisible dicho botón.

ClickBack(): cuando se pulsa sobre el botón “Back”, vuelve al menú principal.

Replay(): función que se ejecuta cuando se pulsa el botón “Jugar de Nuevo”, se crea una partida nueva.

Al igual que en el juego de las cartas, todos los niveles funcionan exactamente igual, se varía cual es el número de sonidos máximos y el vector, pero el funcionamiento es el mismo. Hay un *script Imagen* por cada sonido, y el *Crt_Juego*de cada escena.

## 7. HARDWARE

Para hacer funcionar el sistema se hace uso de los siguientes dispositivos:

- Smartphone con acceso a una red Wifi: en él se instalará la aplicación.
- Sensores PIR: Sensores de luz infrarroja que actúan como sensores de movimientos, estos detectarán si hay movimiento en alguna habitación de la casa.

*Fig. 14 Sensor PIR*

- Módulo ESP-32: El módulo wifi Arduino con CPU integrada permite realizar la conexión entre los sensores y la aplicación móvil. Gracias a su CPU podemos programar las acciones que debe realizar según la información recibida de los sensores PIR.

*Fig. 15 Módulo ESP-32*

- Módulo de fuente de alimentación y batería de 9V: Necesitaremos un par para cada sensor, se encargan de proporcionar la potencia necesaria para alimentar al módulo ESP-32 y a los sensores PIR. Dado que ambos son de bajo consumo al estar destinados para IOT, la batería tendrá una larga vida útil.


*Fig. 16 Módulo de fuente de alimentación y batería*

El montaje final podemos verlo en la siguiente imagen (Figura 17).

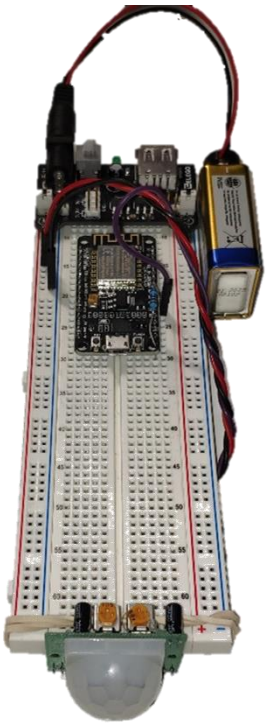


*Fig.17 Montaje final*

Como vemos es bastante compacto y si prescindimos del uso de una placa de pruebas o usamos una más pequeña el espacio que ocupa es mínimo. Solo necesitamos alimentar tanto al módulo como al sensor PIR y conectar la salida del sensor PIR a un pin del módulo. Tanto el módulo ESP-32 como el sensor se alimentan con 5 V, así que con la salida del módulo de fuente de alimentación de 5 V proporcionada por la pila de 9 V es suficiente.

## 8. PROGRAMACIÓN DEL MÓDULO ESP-32

Una opción para programar el módulo ESP-32, la que hemos escogido nosotros, es por medio de Arduino. Arduino utiliza un lenguaje basado en C++ y ya disponíamos de cierta experiencia programando en este lenguaje y usando otros módulos Arduino.

Para proceder a programar en esta placa necesitamos usar el programa de Arduino y seleccionar "NodeMCU" como placa. Además, las funciones que necesitaremos para toda la comunicación Wifi las obtenemos de la librería "ESP8266WiFi.h".

Cuando ya esta todo el entorno configurado correctamente lo primero será inicializar la placa, concretamente a una frecuencia de 115200 baudios/segundo que es la configuración más común para esta. Después configuraremos los pines de entrada y salida, en nuestro caso solo necesitamos un pin de entrada para el sensor PIR. Para las pruebas usamos un pin de salida para un LED que se encienda o apague en función del movimiento, pero no es necesario para el proyecto final.

Además, haremos que nada mas inicializarse se conecte a una red Wifi, deberemos indicar el nombre (SSID) y contraseña de la red. Seguidamente bloqueamos la CPU en un bucle hasta que se confirme la conexión y una vez se ha conectado imprimimos por pantalla la dirección IP que se ha asignado al módulo para poder comunicarnos con él.

Como ya se ha explicado anteriormente, el sistema es activo, la aplicación del smartphone hace peticiones en bucle en el momento que necesita la información sobre el movimiento y sigue pidiendo información hasta que la respuesta es que sí hay movimiento. Es por esto que hacemos que el módulo se comporte como un servidor (HTTP). Lo ponemos a escuchar peticiones (GET) mediante una función de la librería "ESP8266WiFi.h", cuando recibe una petición, inmediatamente accede a una función que hemos programado llamada *movimiento*, la cual devuelve un booleano indicando si hay movimiento o no. En función de este booleano se enviará una respuesta (200 OK) con un texto que indique si hay movimiento ("MOVIMIENTO: SI") o no ("MOVIMIENTO: NO"). Tras esto se desconectará y se pondrá de nuevo a la escucha de nuevas peticiones.

La función *movimiento* que hemos mencionado es sencilla, simplemente lee el voltaje del pin de entrada conectado al sensor PIN. Si es HIGH significa que hay movimiento, si es LOW no lo hay, conforme a esto devuelve un booleano indicando la información respectiva.

Una duda que puede surgir es que, si las peticiones que hace la aplicación al módulo caen en instantes en los que el sujeto está quieto, pero realmente ha habido movimiento en algún momento entre las peticiones, no lo detectaría. Hemos tenido esto en cuenta, el sensor PIR cuando detecta movimiento mantiene el estado HIGH durante unos segundos (6 más o menos) mientras actualiza la nueva imagen de luz infrarroja. Hemos programado las peticiones para que se hagan en intervalos menores a este tiempo de actualización del PIR.

## 9. VENTAJAS Y FUNCIONALIDAD DE LA APLICACIÓN

**Funcionalidad:**

Sistema de alarmas por voz para mantener una rutina y no olvidarse de las cosas.

Juegos para entrenar la memoria visual y auditiva, que incluyen imágenes típicas de la vida cotidiana para estimular los recuerdos.

**Ventajas:**

- Llama más la atención al sustituir una melodía por voz.
- Botones grandes, fácil de usar.
- Muy visual, todo con imágenes, poca letra.
- 2x1, la misma aplicación incluye las alarmas y los juegos para el entrenamiento de memoria. Así las personas mayores no tienen que usar distintas apps que puede resultarles más complejo.
- Puede ser usada por cualquier usuario e independiente de los sensores.