



# VNiVERSiDAD D SALAMANCA

## ***Trabajo Final Periféricos Implementación de WhatsApp***

Periféricos 3º Grado Ingeniería Informática  
Luis Blázquez Miñambres, DNI 70910465Q  
Miguel Cabezas Puerto, DNI 70911497J  
Alberto Hernández Pintor, DNI 70958670J  
13 de Mayo de 2019

## Contenido

1.	Introducción .....	3
a.	Objetivos.....	3
b.	Consideraciones y mejoras añadidas .....	3
2.	Montaje y disposición del circuito .....	4
3.	Funcionamiento de la aplicación.....	7
4.	Análisis del código.....	10
a.	Arduino.....	10
5.	Dificultades encontradas .....	27
6.	Bibliografía .....	29

## 1. Introducción

### a. Objetivos

El trabajo propuesto consiste en la implementación de un chat de texto similar a aplicaciones bien conocidas como *WhatsApp* o *Telegram* mediante el uso de una placa *GSM Shield 2* conectada a una placa de *Arduino* mostrando los mensajes por pantalla.

La placa *GSM Shield 2* con la biblioteca *GSM* nos permite realizar las operaciones que se pueden realizar con un teléfono *GSM*: realizar y recibir llamadas de voz, enviar y recibir SMS y conectarse a Internet a través de una red *GPRS*. Para ello es necesario introducir una tarjeta SIM convencional, es decir, cualquier tarjeta SIM de las utilizadas en los actuales SmartPhones.

El Shield, al tener incorporado un módem que transfiere datos desde un puerto serie a la red *GSM* se convierte en el complemento ideal de *Arduino* para realizar las tareas comentadas anteriormente.

### b. Consideraciones y mejoras añadidas

Inicialmente nuestro objetivo fue utilizar *WhatsApp* para este cometido, pero para poder hacerlo se tiene que hacer uso de la plataforma *Twilio* prescindiendo para ello de la placa *GSM* y abandonando con ello el principal objetivo de la práctica, además *WhatsApp* es un servicio de mensajería privado que cifra los mensajes haciéndonos imposible el poder enviar o recibir mensajes por lo que optamos por utilizar otro servicio de mensajería, *SMS* (Sort Messages Service) el cuál es accesible utilizando la librería ya nombrada (*GSM.h*).

Una vez elegido el servicio a utilizar, el siguiente objetivo a alcanzar se trataba de conseguir enviar un mensaje a un número de teléfono y posteriormente no solo esta funcionalidad, si no también poder recibir mensajes de otros números, es decir, se trataba de una comunicación unidireccional puesto que los equipos que intervienen en la comunicación no pueden mandar mensajes a la vez, esto lo solucionamos posteriormente convirtiendo la comunicación en bidireccional de tal forma que se puede tener una conversación fluida normal al estilo *WhatsApp*.

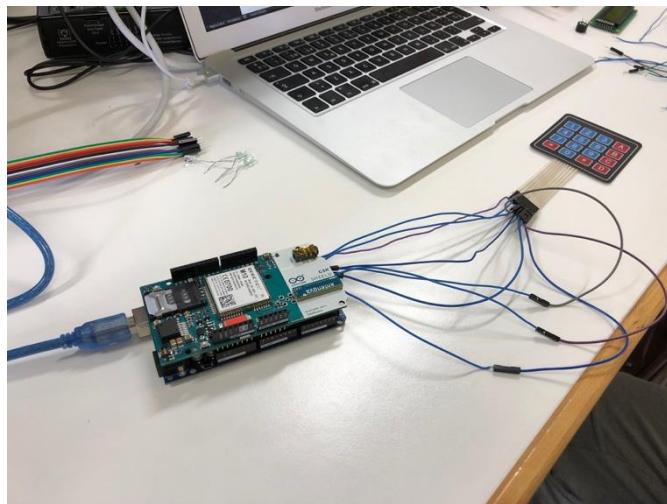
A partir de este momento optamos por empezar a introducir más periféricos tanto de entrada (Keypad) como de salida (LEDs y Buzzer) con el fin de conseguir un mayor número de funcionalidades, cuyo propósito se explica en puntos posteriores.

Por último, con el fin de poder tener una agenda de contactos y guardar las conversaciones (para que, aunque te salgas de una conversación cuando te vuelvas a meter en la misma, continúe por donde se quedó) decidimos añadir persistencia de datos, lo cual nos obligó a utilizar Python. Finalmente decidimos añadir una interfaz gráfica en la terminal del ordenador para una mejor visualización de la interacción, una pantalla LCD donde poder visualizar menús y notificaciones y un joystick para movernos por los menús en el LCD.

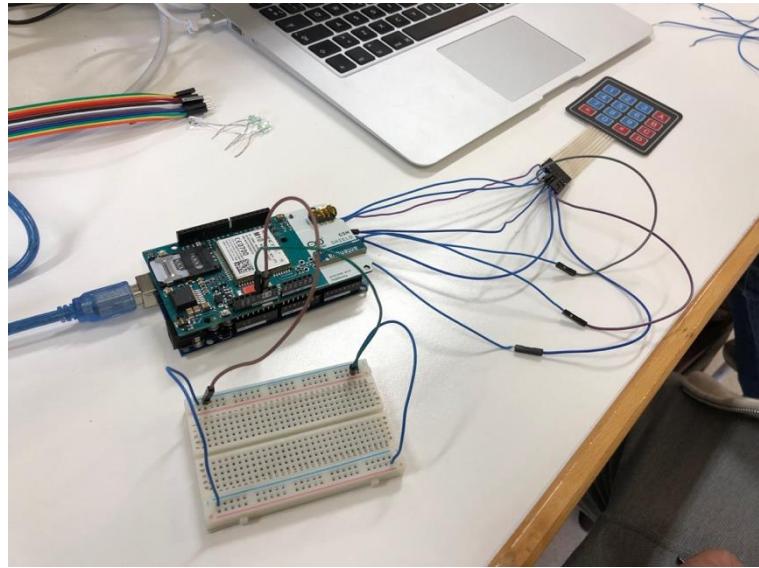
## 2. Montaje y disposición del circuito



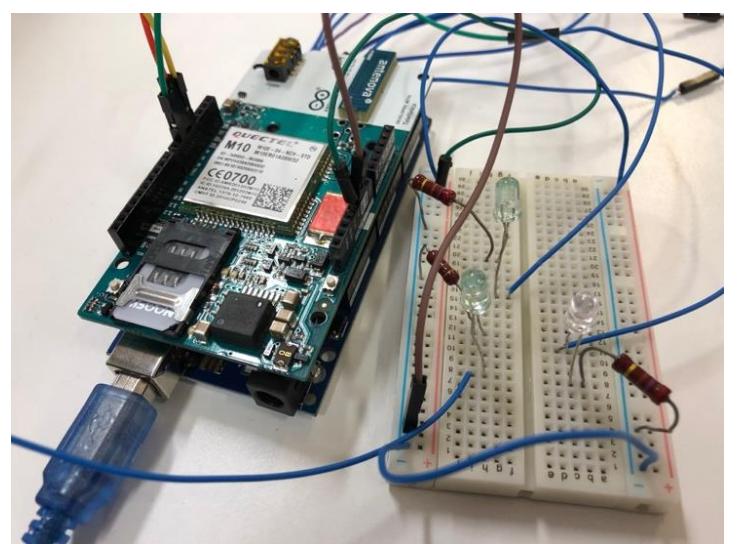
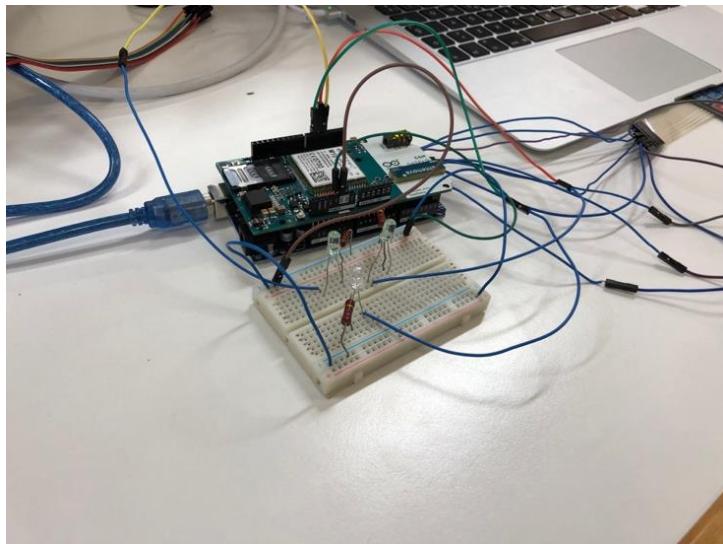
En primer lugar, conectamos el *Shield GSM* a la placa de *Arduino* como podemos observar en la imagen de la izquierda, dejando sin conectar la pata 2 de la placa **GSM** como se observa en la imagen de la derecha (para evitar problemas de conexión de la placa).



A continuación, como observamos en la imagen de encima, añadimos el **Keypad** que va conectado a 8 pines digitales, cuatro de ellos para las filas y los otros cuatro para las columnas.

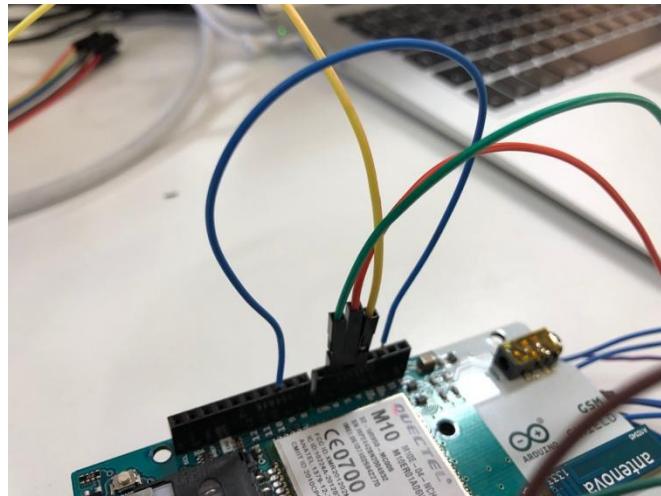


Conectamos la protoboard a la placa para suministrarle energía y un punto a tierra ambos necesarios para el funcionamiento de los componentes conectados a la misma.

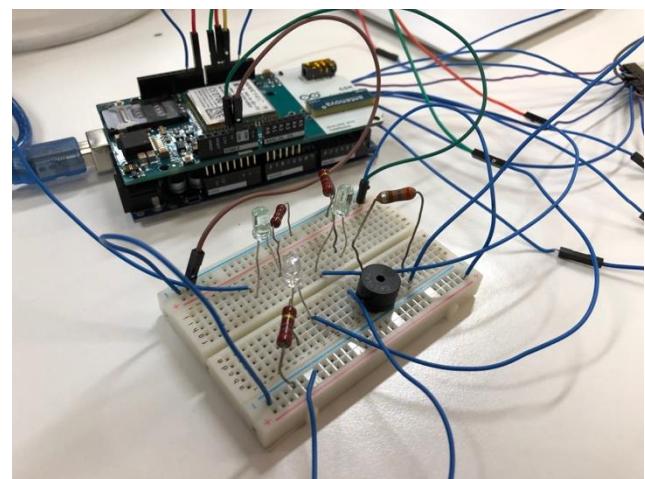
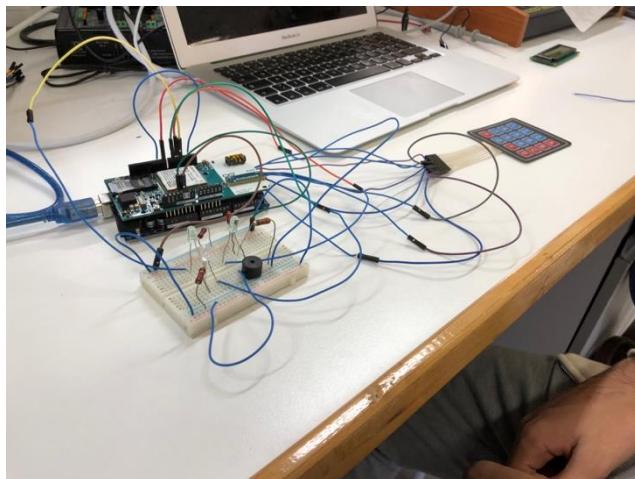


Conectamos tres **LEDs** a la protoboard de la siguiente forma:

- La pata positiva, la más larga, va conectada a un pin de la placa.
- La pata negativa, la más corta, va conectada mediante una resistencia a tierra.

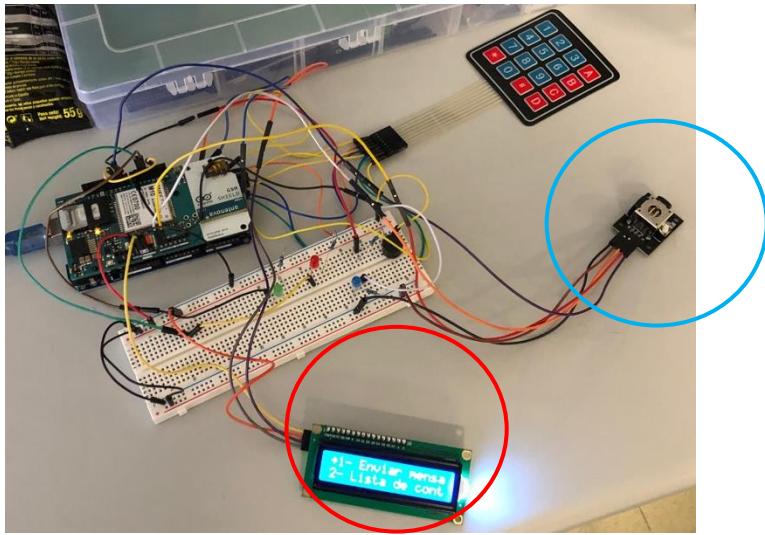


Hacemos un puente con los pines 2 y 10 (cable azul), esto se debe a que utilizamos un Arduino MEGA y con esta versión de Arduino es necesario realizar este puente para que el GSM encuentre señal de red.



Conectamos un **buzzer** de la siguiente forma:

- La pata positiva marcada por un símbolo '+' se conecta con una pata de una resistencia, la otra pata de la resistencia va conectada a un pin de la placa.
- La pata negativa va directamente conectada a tierra.



La conexión del **LCD** (Círculo rojo):

- Conexión GND a tierra
- Conexión VCC a corriente, 5V
- Conexión SDA al pin 20 (Datos)
- Conexión SCL al pin 21 (Reloj)

La conexión del **Joystick** (Círculo azul):

- Conexión GND a tierra
- Conexión VCC a corriente, 5V
- Conexión VRX a un pin analógico (Controla el movimiento en el eje X)
- Conexión VRY a un pin analógico (Controla el movimiento en el eje Y)
- Conexión SW a un pin digital (Controla la pulsación del joystick)

### 3. Funcionamiento de la aplicación

Hemos desarrollado una aplicación de mensajería bidireccional estilo WhatsApp mediante la utilización del servicio de mensajería SMS.

En primer lugar, antes de la aparición del menú principal, el usuario, mediante la pulsación de las teclas del teclado de membrana, deberá introducir un código que emula el código de desbloqueo de un dispositivo móvil.

En nuestro caso el código elegido para el desbloqueo es “4568”. En caso de tres intentos de desbloqueo incorrectos el programa acabará, simbolizando el bloqueo de un móvil en caso de realizar demasiados intentos. Una vez desbloqueado aparece una pantalla (a través de la consola del ordenador) con el mensaje “Bienvenido”, en este instante el usuario deberá esperar que el módulo *GSM Shield 2* se conecte a la red móvil GPRS/GSM. Una vez conectado, el usuario puede visualizar el menú principal de la aplicación consistente en 3 apartados (*Enviar mensaje, lista de contactos, salir*) detallados a continuación.

Para la elección de uno de ellos el usuario deberá pulsar el número asociado a cada uno de ellos de nuevo haciendo uso del teclado de membrana. En líneas generales, el manejo de cada uno de los menús y submenús de la aplicación se realizará mediante la pulsación de las teclas del teclado de membrana. Además, incorporamos la aparición de los mensajes que se muestran en la consola también en la pantalla LCD como se explicará en puntos posteriores, así como el uso de un joystick para poder desplazarnos en los diferentes menús y submenús que aparezcan

en la pantalla LCD.

- **Enviar mensaje**: destinado a escribir mensajes bien a un nº recogido en la agenda de contactos o bien a un número desconocido. Con la pulsación de esta opción aparecerá un submenú compuesto por:
  - **Enviar mensaje a contacto**: mediante la introducción bien del nombre del contacto o bien del número desconocido se podrá abrir la conversación con ese número de teléfono, en caso de tener mensajes previos con dicho número se podrán ver.

En caso de entablar conversación con un número desconocido y posteriormente guardarlo, al abrir en otro momento la conversación con ese número solo aparecerán los mensajes desde que se guardó a ese número de teléfono como contacto (el motivo puede verse en el apartado *Análisis del código*)
  - **Salir**: permite al usuario volver al menú principal.
- **Lista de contactos**: destinado a visualizar la lista de contactos guardados, además de la visualización de estos al pulsar esta opción aparecerá un menú compuesto por:
  - **Añadir**: permite al usuario añadir un nº de teléfono bajo un alias o nombre mediante la escritura con el teclado del ordenador. Para ello escribirá un número y el nombre o alias asociado.
  - **Eliminar**: permite al usuario borrar un contacto, para ello debe escribir el nombre con el que lo tenga guardado.
  - **Modificar**: permite al usuario cambiar bien el número de teléfono bien el alias o nombre asignado a un usuario. Para ello deberá escribir el nombre con que tenga guardado al usuario. A la hora de modificar deberá meter ambos campos de nuevo, aunque no modifique uno de ellos (nombre/alias y número de teléfono)
  - **Ver contactos**: mediante esta opción podrá visualizar los contactos, esta funcionalidad es para poder ver los contactos también en el monitor LCD ya que al quitar las cabeceras de los menús y submenús en este “Lista de contactos” solo aparecen las opciones comentadas sin la lista de contacto en la parte superior como se hace en la consola del ordenador.
  - **Volver**: permite al usuario volver al menú principal.
- **Salir**: cierra la aplicación produciéndose la reproducción de una melodía a través del buzzer.

En cuanto a la lógica de envío/recepción de mensajes, este se notifica mediante la iluminación de los diferentes LEDs usados siguiendo la siguiente cadena:

- *Encendido de un LED verde* cuando el mensaje es escrito. Lo que simboliza el primer y segundo tick de WhatsApp.
- *Encendido del segundo LED verde*, manteniendo el primero encendido, cuando el mensaje es enviado. Lo que simboliza los ticks “azules” de WhatsApp como el

mensaje ha sido leído por el destinatario. Pasados unos segundo ambos LEDs verdes se apagan.

- *Parpadeo durante unos segundos del LED azul.* Simboliza que se ha recibido un mensaje, bien en la conversación abierta, bien un mensaje de cualquier otro número. Además, en cada recepción de mensajes se produce el pitido de una melodía “Super Mario Victory” haciendo uso del buzzer.

En cuanto a la lógica de las conversaciones, se puede establecer, como ya se ha explicado, conversaciones tanto con números contenidos en los contactos como con números desconocidos; todo ello haciendo uso de la consola de Python mediante la pulsación de las teclas del ordenador.

La comunicación es bidireccional, mientras que se está escribiendo puedes recibir mensajes, tanto de la conversación abierta como de cualquier otro número. Si mientras se tiene abierta una conversación otro número manda un mensaje, en la consola aparecerá el mensaje “Mensaje de [nombre contacto o nº emisor]” dependiendo si el número emisor se encuentra en los contactos o no.

El contenido de dicho mensaje se almacenará en un fichero con el nombre del contacto o nº móvil del emisor en caso de no estar en los contactos (persistencia). Si posteriormente se abre dicha conversación se podrán observar todos los mensajes con dicho contacto/número.

En general, al abrir una conversación con un nº de teléfono con el que ya se habían intercambiado mensajes, aparecerán todos ellos gracias a la persistencia de datos de la que dispone nuestra aplicación y de la forma que se explica en el apartado *Análisis del código*.

En cuanto al uso del buzzer:

- En el encendido de la aplicación, una vez que el módulo *GSM Shield 2* ha conectado con la red móvil se produce la reproducción de una melodía.
- Cuando la aplicación se apaga, bien por superar el nº máximo de intentos de desbloqueo bien por salir normalmente de ella, se reproduce otra melodía diferente.
- En la recepción de mensajes, bien de la conversación abierta o bien de otro número de teléfono, se reproducirá una tercera melodía diferente. Tanto para la reproducción del sonido como para el encendido de los LEDs será necesario que el usuario esté en el apartado del menú *Enviar mensaje* sino no se producirá sonido ni luz alguna, simplemente se almacenarán los mensajes en el fichero de la conversación correspondiente. Una vez que entras en la conversación, aparecerán los mensajes y sonará y lucirá. En general, se produce el sonido y el parpadeo del LED cuando se recibe un mensaje y se está dentro de alguna conversación.

En cuanto al uso de la pantalla LCD, en ella aparecen los mensajes aparecidos en la consola con algunas salvedades debido al menor tamaño de esta, mediante el uso del joystick nos podremos desplazar arriba y abajo en ella como se explicará en el siguiente punto. Los mensajes imprimidos en la pantalla LCD serán:

- Los diferentes menús y submenús quitando la cabecera, es decir, solo las opciones en las que podremos desplazarnos haciendo uso del joystick.
- En las conversaciones aparecerá el último mensaje recibido emulando las notificaciones emergentes en el móvil con WhatsApp, así mismo al enviar un mensaje aparecerá el mensaje mandado emulando la función de responder sin entrar en la conversación de la que disponen algunos dispositivos móviles. El formato de los mensajes en la pantalla LCD será el siguiente:

- Si el mensaje es de envío aparecerá la palabra “Yo” en la primera fila del LCD y el mensaje en la segunda fila.
- Si el mensaje es de recepción y de la conversación que se tiene abierta (el origen es el mismo nº con el que se está hablando) aparecerá su número en la primera fila y el mensaje en la segunda
- Si el mensaje es de recepción, pero de un nº cuya conversación no es la que actualmente se tiene abierta aparecerá el mensaje “Mensaje de: nº de móvil emisor”.

En cualquier caso, en el monitor LCD aparecerá simplemente el último mensaje emulando las notificaciones emergentes de WhatsApp. Si se desea ver la conversación entera se deberá usar la pantalla del ordenador (emulando abrir la conversación de WhatsApp en el móvil).

## 4. Análisis del código

Antes de empezar a analizar y describir el código implementado para la realización de la práctica , cabe mencionar que hemos utilizado como recurso adicional al lenguaje de Arduino , el lenguaje Python para conseguir persistencia, es decir, que se pudieran almacenar y restaurar los contactos de una lista de contactos, así como las conversaciones de los chats, ya que el lenguaje Arduino no permite interaccionar con el escritorio del ordenador debido a que todo lo realiza a través del puerto serie , por lo que necesitábamos un lenguaje de programación que sirviera como intermediario para realizar dicha tarea. También probamos con otros lenguajes como Processing, pero debido a que este lenguaje tomaba el control de puerto serie para él y Arduino hace lo mismo, era imposible hacer que ambos estuvieran escribiendo y leyendo del puerto serie sin que uno de los dos se apropiara del mismo.

### a. Arduino

En cuanto al código en Arduino, hemos dividido la implementación en varios ficheros o módulos, uno para cada dispositivo o herramienta que hemos utilizado en el circuito:

- **Menú principal:** en la primera parte del método *setup()* del archivo *main* o principal declaramos todos los pines de todos los dispositivos, así como diversos flags y variables que utilizaremos en este mismo método.
- También ahí declaramos el número PIN de la tarjeta SIM incorporada al

```
main | sketch | joystick | LCD | SMS_Service | buzzer | keypad | main.h
#include "main.h"

void setup() {
    GSM gsmAccess;
    boolean notConnected = true; // GSM Connection state
    char pin[] = "5056"; // PIN number of the SIM Card of the GSM Shield we are using

    // Inicializar el puerto serie con los baudios requeridos
    Serial.begin(9600);

    // Definición de las dimensiones del LCD e inicialización de las mismas
    lcd.begin(16,2); // 16 columnas y 2 filas
    lcd.backlight(); // Encender la luz de fondo
    lcd.createChar(5, flechaD); // Crear el símbolo de la flecha -> a la derecha sobre el byte 5

    while (!Serial) {
        ; // Esperar a que se conecte el puerto serie
    }

    pinMode(pinBuzzer, OUTPUT);

    // Inicialización de pines del joystick
    pinMode(pinEjeX, INPUT); // Eje X
    pinMode(pinEjeY, INPUT); // Eje Y
    pinMode(pinPulsador, INPUT_PULLUP); // Botón central - Pulsador

    // Inicialización de los pines del KeyPad
    for(byte i=0;i<rows;i++) {
        pinMode(Output[i],OUTPUT);
    }

    for(byte s=0;s<columns;s++) {
        pinMode(Input[s],INPUT_PULLUP);
    }
}
```

GSM y que utilizaremos para acceder al mismo.

A continuación, llamamos al método *unlockPhone()* dentro del módulo *Keypad.ino* y que ya explicaremos posteriormente, pero cuya idea básica reside en simular que desbloqueamos un móvil con el Keypad. Para acceder a los servicios del GSM, tenemos que acceder a la tarjeta SIM como ya explicamos anteriormente y, para ello, utilizamos la función de la librería de GSM, *gsm.begin()* al que le pasamos el PIN y con ello iniciamos la comunicación con el GSM.

Este proceso puede tardar de unos segundos a minutos dependiendo de la cobertura, por lo que es necesario introducirlo en un bucle para que se mantenga a la espera hasta que encuentre finalmente la conexión. Posteriormente, limpiamos la pantalla del LCD, así como la pantalla de la terminal de nuestro ordenador, emitimos el sonido de comienzo por el buffer con la función *play\_tone()* que se explicará posteriormente.

```
// Pantalla que logra simular el desbloqueo de un móvil como si de ello se tratase
// nuestra aplicación
unlockPhone();

Serial.println("-----");
Serial.println(" Accessing WhatsApp application ");
Serial.println("-----");

// Comienzo y conexión del GSM shield
// Le pasamos el pin de la tarjeta SIM a través del parámetro begin para que logre acceder a los servicios GSM y GPRS

while(notConnected)
{
    if(gsmAccess.begin("5056", true)==GSM_READY) {
        lcd.print("Connected");
        notConnected = false;
    } else {
        Serial.println("Not connected");
        delay(1000);
    }
}

delay(2000);
lcd.clear();
lcd.setCursor(0,0); /* LCD */

// Limpiamos la pantalla para dejar lugar al menú principal
whatsapp_serial_limpiar_pantalla_menus();

// Tono de indicación acceso a la aplicación
play_tone("startup");

// Iniciación de los pines para los LEDs verde y azul de notificaciones
pinMode(A0, OUTPUT); // LED verde (cuando se envía el mensaje)
pinMode(A1, OUTPUT); // LED verde (cuando aparece nuestro mensaje en la pantalla). Simboliza que el receptor lo ha recibido
pinMode(A2, OUTPUT); // LED azul (parpadeante). Cuando llega el WhatsApp
```

Y finalmente accedemos al bucle de nuestro código principal que accionará, lo primero de todo, el menú principal de nuestra aplicación llamando al método *whatsapp\_serial\_menu()*;

```
// Código principal del programa
void loop() {

    // Aquí comienza el menú principal
    whatsapp_serial_menu();

}
```

- **GUI (Graphic User Interface):** en este módulo mostraremos las distintas funciones, donde cada menú tiene la suya propia en donde se manejan diferentes opciones que se mostrarán a continuación. Por ello, dado que la estructura en cuanto al código es la misma, solo mostraremos la estructura de la función de manejo del menú principal, en el resto de los menús indicaremos detalles concretos de ese menú que son distintos en el resto.

En primer lugar, tenemos la aplicación del menú principal donde tenemos un array de String conteniendo los diferentes submenús como cadenas que serán las que manejemos con el Joystick a través del LCD. Mostramos el menú por el puerto serie con un *Serial.println()* e imprimimos por el LCD los dos primeros menús, dejando la flecha sobre el primer menú para tener la referencia y poder movernos a continuación. Dentro del bucle while, indicamos el movimiento del joystick a través del menú (solo puede ir hacia arriba o hacia abajo o pulsar en el botón central, por eso solo capturamos estas opciones a través del método *getJoystickDirection()* ) hasta que el usuario pulse en el botón central indicando que ha pulsado una de las opciones.

Cabe decir que tenemos una variable llamada '*actual*' que es un entero que registra en cada momento el nº del menú en el que se encuentra el usuario. Por último, en la estructura de cualquiera de los menús, tenemos un switch para decidir a qué opción acceder por medio del valor de la variable '*actual*'. En concreto en este menú podemos acceder al servicio de envío de mensajes, a la lista de contactos o a la opción de salir del programa.

En cuanto a los otros dos menús, como ya mencionamos anteriormente la estructura es similar en los tres, pero debido a la complejidad con la que ya nos encontrábamos con el código decidimos no modular esa estructura para que fuera una función y decidimos ponerla de manera repetida en cada uno de ellos. A la izquierda, está el switch de elección de opción del menú de envío de mensajes llamando a la función `whatsapp_serial_enviar_mensaje()` que permite empezar la comunicación (que explicaremos en el modulo SMS) y la opción de volver al menú anterior. En cuanto al menú de contactos, una vez se accede a él hay que esperar unos segundos a que carguen los contactos.

Esto se realiza de la siguiente manera: enviamos desde el código de Arduino una cadena o palabra clave que Python recogerá y filtrará para decidir qué hacer. Como el código en Python está escuchando continuamente del puerto serie, recoge la cadena *contacts\_transfer* para realizar la tarea de transferir contactos. Python lee de un fichero de texto los contactos que tenemos registrados, los procesa y los manda a través del puerto serie. Para que el código de Arduino los recoja, lo ponemos a escuchar del puerto serie con la función *Serial.read()* que lee byte a byte poniendo un bucle con dos delays de 30 segundos para dejar al buffer del código Arduino llenarse (ya que no es una acción instantánea) para que posteriormente lea y procese la cadena de contactos entera y lo muestre por pantalla.

Posteriormente nos dimos cuenta de que podíamos simplemente mostrar los contactos a través de código de Python, pero eso sería romper con la idea principal del trabajo, ya que aunque Python sirva como apoyo al funcionamiento de la práctica, todo se debe realizar a través del Arduino.

Por último, dentro del menú de contactos, para añadir, eliminar, modificar o ver los contactos de nuestra lista, mandamos una palabra clave al código de Python a través del puerto serie para que realice la tarea que tenga que llevar a cabo y pueda mostrar los cambios en tiempo real a través del Arduino. Para que la acción se complete, introdujimos la opción de que una vez realizada una de las 4 acciones, se pulsase arriba con el joystick para confirmar los cambios realizados y que se recargase la lista actualizada con los contactos.

```

if (actual == 0) actual = 1;
switch(actual) {
    case 1:
        /* Manda la orden al código Python a través de puerto serie para añadir un contacto */
        Serial.println("contacts_add");
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Creando contacto ...");
        break;
    case 2:
        /* Manda la orden al código Python a través de puerto serie para eliminar un contacto */
        Serial.println("contacts_delete");
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Eliminando contacto ...");
        break;
    case 3:
        /* Manda la orden al código Python a través de puerto serie para modificar un contacto */
        Serial.println("contacts_modify");
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Modificando contacto ...");
        break;
    case 4:
        /* Manda la orden al código Python a través de puerto serie para mostrar los contactos */
        whatsapp_mostrar_contactos(list_contacts, i);
        break;
    case 5:
        whatsapp_volver_menu();
        break;
    default:
        break;
}
lcd.clear();
lcd.setCursor(0, 1);
lcd.print("Pulse arriba para volver");

// Si quiere salir de alguno de las opciones de la lista de contactos tiene que pulsar arriba
// con el joystick
mov = 'T';
while(mov != 'W'){
    delay(100);
    mov = getJoystickDirection();
}

lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Volviendo");
delay(400);

```

- **SMS communication:** en este submenú comienza la comunicación de mensajes entre un número de teléfono o un contacto de nuestra lista. Una vez pulsada la opción de enviar mensaje con el joystick, aparecerá en la terminal del ordenador para que introduzcamos el contacto o número con el que se desea conversar. A partir de aquí, si el contacto introducido no está en nuestra lista, se advertirá al usuario de que no se ha encontrado y de que lo introduzca de nuevo.

En caso de que sepamos el número de teléfono, pero no queramos añadirlo a nuestra lista, bastará con introducir el número de teléfono de la persona con la que queremos iniciar una conversación (importante que el número de teléfono sea añadido con el prefijo de telefonía correspondiente de cada país para que el GSM lo reconozca). En cualquier otro caso, introducimos el nombre de un contacto de nuestra lista y a continuación, en apenas unos segundos, se cargará la conversación que hayamos tenido hasta ese momento con él.

Esto sucede cuando mandamos una palabra clave al código Python indicándole que cargue la conversación de ese contacto de su fichero de conversaciones (cada contacto tendrá un fichero con la conversación mantenida hasta el momento) y comience el chat.

En ese momento en Arduino, nos encontramos en un bucle infinito con dos opciones. Utilizando la función *sms.available()* comprobamos si hay mensajes disponibles que se hayan recibido para leerlos, de esa manera pasariamos a procesarlos (ya sean del mismo con el que estamos hablando o de otro número o contacto). Si no hay mensajes pasamos a enviar un esperar a que el usuario introduzca un mensaje para enviar.

```
/* Comunicación bidireccional de conversación de whatsapp. Mientras se escribe a un número o contacto
 * se pueden recibir mensajes
 */
void start_communication()
{
    get_contact_number();

    while(1) {
        if (sms.available()) {
            // Get SIM number and print it
            recvMessage();
        } else {
            value = sendMessage();
            if (value == 0){
                digitalWrite(pinLED_2, LOW);
                break;
            }
        }
    }

    whatsapp_serial_enviar_mensaje();
}
```

A la hora de enviar un mensaje, si se ha introducido algo que queremos mandar, se envía a través del puerto serie desde el código Python escribiendo en la terminal hasta el código Arduino. Una vez el Arduino lo ha leído y procesado activamos el primer LED verde indicando el primer tick verde de Whatsapp de que hemos enviado el mensaje al contacto, y lo enviamos con la función *sms.print()* al contacto con *sms.beginSMS()* y cortamos la comunicación con *sms.endSMS()*. Pasados 5 segundos, que es lo que normalmente tarda un mensaje en llegar a un contacto en unas condiciones de cobertura normales, encendemos el segundo LED verde indicando el segundo tick verde de Whatsapp de que lo ha recibido. A partir de aquí los apagamos y volvemos al bucle principal de envío-recepción.

```

/* Tratamiento del envío de mensajes */
int sendMessage()
{
    String msg_send = "";           // Message to send through the serial port to the mobile number
    Serial.println("messages_1");    // Mensaje enviado al código en Python
    print_lcd("Introduce_message", ""); /* LCD */
    delay(3000);

    do { // If data is available to read
        while(!Serial.available()) {
            if (sms.available())
                break;
        }

        while (Serial.available())
        {
            delay(300); //delay to allow buffer to fill
            if (Serial.available() >0)
            {
                char c = Serial.read(); //gets one byte from serial buffer
                msg_send += c; //makes the string readString
            }
        }

        if (sms.available()){
            recvMessage();
        }
    } while (msg_send.equals("")||

    // IMPORTANTE: si en una conversación recibimos 'exit' nos salimos de ella
    if (msg_send.equals("exit")){
        digitalWrite(pinLED_2, HIGH);
        return 0;
    }

    // Turns on second green LED
    lcd.clear(); /* LCD */
    digitalWrite(pinLED_1, HIGH);

    delay(5000);
    print_lcd("msg_send", msg_send); /* LCD */

    // Turns on second green LED
    digitalWrite(pinLED_2, HIGH);

    int num = sms.beginSMS(receiver_phoneNumber);
    num = sms.print(msg_send);
    sms.endSMS();
    delay(2000);

    digitalWrite(pinLED_1, LOW);
    digitalWrite(pinLED_2, LOW);

    return 1;
}

```

En cuanto a la recepción cabe mencionar que podemos entrar en este método de dos maneras, la primera es si se ha recibido un mensaje cuando entramos en el menú de envío de mensajes y otra es si hemos entrado en la función anterior para escribir un mensaje, pero aún no lo hemos mandado y, en esos segundos que el mensaje aún no se ha enviado podemos recibir un mensaje, entonces lo procesamos.

Para recibir el mensaje lo primero que hacemos es descartar mensajes extraños o del buzón de voz de compañías de teléfono que interrumpan nuestra aplicación, para eso comprobamos que si el primer carácter del mensaje SMS enviado empieza por # este sea descartado. Después mandamos la palabra clave al Python para que recoja el mensaje, pero no lo lea. Esto lo hacemos para que registre el mensaje del puerto serie en el fichero que guarda la conversación con ese contacto, pero no lo lea, pues esto es tarea del código Arduino.

Posteriormente, leemos el mensaje y activamos el LED azul parpadeantes, simulando los ticks azules de visto de Whatsapp y comprobamos si el mensaje recibido es del contacto, para mostrarlo por pantalla y por el LCD, o por el contrario es de otro contacto o número distinto de con quien estamos manteniendo la conversación. En ese caso se muestra en el LCD que se ha recibido un nuevo mensaje de ese número o contacto, pero no se muestra su contenido, para ello tendremos que ir a la conversación con

ese contacto o número.

Y eliminamos el mensaje de la memoria del GSM para que no sea recibido en conversaciones posteriores con `sms.flush()`.

```
void recvMessage()
{
    char c, d;
    int i = 0;
    String msg_recv = ""; // Message to receive from the serial port from another phone number
    sms.remoteNumber(unk_receiver_phoneNumber, 20);

    // This is just an example of message disposal
    // Messages starting with # should be discarded
    if(sms.peek()=='#')
    {
        sms.flush();
    }

    // Read message bytes and print them
    Serial.println("recv_on"); // Mensaje enviado al codigo en Python

    if(strcmp(unk_receiver_phoneNumber, receiver_phoneNumber) == 0) {
        Serial.println(String(receiver_phoneNumber) + ":");

    } else {
        Serial.println(String(unk_receiver_phoneNumber) + ":");

        print_msg_LCD("Received", ""); /* LCD */
    }

    while(c=sms.read()) {
        msg_recv = msg_recv + c;
    }

    Serial.println(msg_recv);
    Serial.println("recv_off"); // Mensaje enviado al codigo en Python

    play_tone("recv");

    // Turns on blue LED
    while (i < 5) {
        digitalWrite(pinLED_3, HIGH);
        delay(200);
        digitalWrite(pinLED_3, LOW);
        delay(200);
        i++;
    }
    digitalWrite(pinLED_3, LOW);

    print_msg_LCD("msg_recv", msg_recv);/* LCD */

    // delete message from modem memory
    sms.flush();
}
```

Por último, tenemos la función que recoge el numero o el nombre del contacto con el que queremos conversar de la terminal de Python y lo lee el código de Arduino para procesarlo y enviarle el mensaje.

```
void get_contact_number()
{
    Serial.println("Introduzca el contacto:");
    Serial.println("contacts_msg"); // Mensaje enviado al codigo en Python
    int i = 0;
    while (1) {
        while (!Serial.available()) {}

        while (Serial.available())
        {
            delay(300); //delay to allow buffer to fill VERY IMPORTANT, it worked because of this
            if (Serial.available() >0)
            {
                char c = Serial.read(); //gets one byte from serial buffer
                receiver_phoneNumber[i] = c; //makes the string readString
                i++;
            }
            if (i == 12) break;
        }
        if (i == 12) break;
    }

    receiver_phoneNumber[i] = '\0';
    print_msg_LCD("Telephone numbers", ""); /* LCD */
}
```

- **LCD:** en este fichero solo hay una función y es la de imprimir los mensajes de inicio de LCD cuando se realizan determinadas acciones, en las que se pasan como parámetros la acción que se va a realizar para imprimir los mensajes y el mensaje que mostrar en pantalla. Para la utilización de esta herramienta hemos usado los métodos *lcd.setCursor()* para situar el cursor en una posición (fila, columna) determinada. Aparte cada vez que imprimíamos algo por el LCD, teníamos que limpiar la pantalla con el método *lcd.clear()*.

```

/* Imprime un mensaje en el LCD atendiendo al menú desde el que se imprima y el mensaje que se quiera
 */
void print_msg_LCD(String mode, String msg)
{
    if (mode.equals("msg_send")) { // Sender
        Serial.println("Sender writes LCD");
        lcd.setCursor(15,0);
        lcd.print("<");
        lcd.setCursor(13,0);
        lcd.print("Yo");

        lcd.setCursor(0,1);
        lcd.print(msg);

        delay(4000);
        lcd.clear();

    } else if (mode.equals("msg_recv")) { // Receiver
        Serial.println("Receiver writes LCD");
        lcd.setCursor(0,0);
        lcd.print(">");
        lcd.setCursor(1,0);
        lcd.print(receiver.phoneNumber);
        lcd.print(":");

        lcd.setCursor(0,1);
        lcd.print(msg);

        delay(1000);
        lcd.clear();

    } else if (mode.equals("keypad")) { // Keypad
        lcd.setCursor(0,0);
        lcd.print("Introducir clave");

        lcd.setCursor(1,9);

    } else if (mode.equals("Telephone_numbers")) { // Telephone Numbers
        Serial.println("Telephone Numbers write LCD");
        lcd.setCursor(15,0);
        lcd.print("<");
        lcd.setCursor(1,4);
        lcd.setCursor(4,1);
        lcd.print(receiver.phoneNumber);

    } else if (mode.equals("Introduce_message")) { // Introduce message
        lcd.clear();
        Serial.println("Message introducing writes LCD");
        lcd.setCursor(0,0);
        lcd.print("Introduzca el");

        lcd.setCursor(0,1);
        lcd.print("mensaje a enviar");

    } else if (mode.equals("Received")) {
        //Serial.println("Message received writes LCD");

        lcd.setCursor(0,0);
        lcd.clear();
        lcd.print("**Mensaje recibido**");
        delay(4000);

        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("From:");
        lcd.setCursor(0,1);
        lcd.print(unk_receiver.phoneNumber);

        delay(1000);
    }
}

```

- **Joystick:** en este fichero hay dos funciones principales. La primera es la que recoge el movimiento del joystick que se ha realizado. Para determinar los patrones, hay que calibrar los valores que devuelven los pines analógicos a los que están conectados los pines del joystick para el ejeX y el ejeY. Esto lo hemos realizado por prueba y error y es importante hacerlo pues el nivel de calibración puede variar de una ejecución a otra dependiendo del ángulo de giro del joystick hacia cualquiera de los lados.

Por otro lado, el pin digital 9 ha sido utilizado para leer el valor del pulsador o botón central, que en este caso por defecto es true y cuando se pulsa devuelve un valor booleano false.

```
main GUI.h Joystick LCD SMS_Service buzzer keyPad main.h
/*
 * Devuelve una letra W, A, S, D u O dependiendo de la dirección de movimiento leída del joystick
 * 0-0 --> Las lecturas han de ser calibradas ya que para cada joystick son diferentes los valores leídos
 * correspondientes a cada dirección (arriba, abajo, izquierda y derecha)
 */
char getJoystickDirection()
{
    bool pulsador = false;
    int sensorX = analogRead(pinEjeX); // Lee del pin A0, conectado al eje de las X del joystick
    int sensorY = analogRead(pinEjeY); // Lee del pin A0, conectado al eje de las Y del joystick
    pulsador = digitalRead(pinPulsador); // Lee del pin digital 9, conectado al pulsador del joystick

    if(sensorX == 2 && sensorY == 0) { // Arriba
        return 'W';
    } else if(sensorX >= 5 && sensorX <= 7 && sensorY >= 300) { // Abajo
        return 'S';
    } else if(sensorX == 0 && sensorY == 2) { // Izquierda
        return 'A';
    } else if (sensorX >= 300 && sensorY == 5){ // Derecha
        return 'D';
    }

    if(!pulsador) { // Centro del joystick presionado
        pulsador = false;
        return 'O';
    }

    return 'T'; // En caso de que no se pulse ninguna tecla, para mantener la lógica del código
}
```

Además, tenemos la función siguiente `getMenu()` que lo que hace es a partir de la posición actual en la que nos encontramos dentro del menú del LCD y del array de opciones del menú (array de string que en cada menú será diferente) permite calcular la posición relativa de la flecha a partir del movimiento del joystick que se haya realizado.

```
/* Función que imprime un menú distinto en el LCD atendiendo a los textos que se incluyan en el
 * array 'menus' y a la posición actual de la flecha en el LCD
 */
void getMenu(int pos_actual, String menus[])
{
    if (pos_actual <= 0)
        return;

    lcd.clear();

    // Menu de la primera fila
    if (pos_actual == 1) {
        // Con flecha
        lcd.setCursor(0, 0);
        lcd.write (byte (5));
        lcd.setCursor(1, 0);
        lcd.print(menus[0]);
    }

    // Sin flecha
    lcd.setCursor(0, 1);
    lcd.print(menus[1]);
} else if (pos_actual == 2) {
    // Sin flecha
    lcd.setCursor(0, 0);
    lcd.print(menus[0]);

    // Con flecha
    lcd.setCursor(0, 1);
    lcd.write (byte (5));
    lcd.setCursor(1, 1);
    lcd.print(menus[1]);

    limit = 3;
    limit2 = 3;
} else if (pos_actual == 3) {
    // Sin flecha
    lcd.setCursor(0, 0);
    lcd.print(menus[1]);

    // Con flecha
    lcd.setCursor(0, 1);
    lcd.write (byte (5));
    lcd.setCursor(1, 1);
    lcd.print(menus[2]);
    limit2 = 4;
} else if (pos_actual == 4) {
    // Sin flecha
    lcd.setCursor(0, 0);
    lcd.print(menus[2]);

    // Con flecha
    lcd.setCursor(0, 1);
    lcd.write (byte (5));
    lcd.setCursor(1, 1);
    lcd.print(menus[3]);
    limit2 = 5;
} else if (pos_actual == 5) {
    // Sin flecha
    lcd.setCursor(0, 0);
    lcd.print(menus[3]);

    // Con flecha
    lcd.setCursor(0, 1);
    lcd.write (byte (5));
    lcd.setCursor(1, 1);
    lcd.print(menus[4]);
}
}
```

- **Keypad:** en este fichero nos ha sido más difícil implementarlo a nivel de código. Esto se debe a que, como ya se mencionará más adelante en el apartado dificultades, las librerías *Keypad.h* y *GSM.h* eran incompatibles. Debido a esto tuvimos que implementar un método que mejoramos a partir de uno que venía en internet sobre como recoger el valor de un keypad manualmente. Para ello y a modo simplificado, cuando leímos de los pines Input de las columnas para rastrear la columna de la tecla que se había pulsado y una vez encontrada, empezar a rastrear la fila de la tecla a través de las funciones *digitalWrite()*.

Obviamente esta funcionalidad la realiza la librería *Keypad.h* internamente, y básicamente se trata de encontrar el pulso leyendo y escribiendo en todos los pines de filas y columnas hasta encontrar aquel pin al que se le ha mandado la señal y ha sido activado, entonces esa es la tecla que ha sido pulsada. A partir de esta función se devuelve un byte indicando el carácter de la tecla del keypad que se ha pulsado.

```
/* Function used to detect which button is used. No need using Keypad.h */
byte keypad()
{
    static bool no_press_flag=0; // Static flag used to ensure no button is pressed
    byte h=0,v=0; // Variables used in for loops

    for(byte x=0;x<columns;x++) {
        if (digitalRead(Input[x])==HIGH); // Read evry input if high continue else break;
        else
            break;

        if(x==(columns-1)) {
            no_press_flag=1;
            h=0;
            v=0;
        }
    }

    if(no_press_flag==1) {
        for(byte r=0;r<rows;r++) // For loop used to make all output as low
            digitalWrite(Output[r],LOW);

        for(h=0;h<columns;h++) {
            if(digitalRead(Input[h])==HIGH) { // If specific input is remain high (no press on it) continue
                continue;
            } else {
                for (v=0;v<rows;v++) {
                    digitalWrite(Output[v],HIGH); // Make specified output as HIGH

                    //If the input that selected from first sor loop is change to high
                    if(digitalRead(Input[h])==HIGH) {
                        no_press_flag=0; // Reset the no press flag;
                        for(byte w=0;w<rows;w++)
                            digitalWrite(Output[w],LOW);
                    }
                }
            }
        }
    }
}

return v*4+h; //return number of button
} // End of loop
} // End of loop
} // End of if
}

return 50;
}
```

Y a continuación en otra función `getKey()`, a través de un switch filtraremos el byte que hemos recibido de la función `keypad()` para determinar que tecla o carácter se ha pulsado. Tendremos adicionalmente un flag que se activará y será únicamente para indicar al resto de funciones donde se llame a esta, que se ha pulsado una tecla y se ha leído del keypad pero esta debe ser finalmente capturada por el código Arduino.

```
/* Captures the key pressed in the keypad from the pin detected by the code */
char getKeyTC()
{
    const unsigned long period=50; // Little period used to prevent error
    if(millis()-kdelay>period) {
        kdelay=millis(); //capture time from millis function
        switch (keypad()) {

            case 0:
                flag = true;
                return '1';

            case 1:
                flag = true;
                return '2';

            case 2:
                flag = true;
                return '3';

            case 3:
                flag = true;
                return 'A';

            case 4:
                flag = true;
                return '4';

            case 5:
                flag = true;
                return '5';

            case 6:
                flag = true;
                return '6';

            case 7:
                flag = true;
                return 'B';

            case 8:
                flag = true;
                return '7';

            case 9:
                flag = true;
                return '8';

            case 10:
                flag = true;
                return '9';

            case 11:
                flag = true;
                return 'C';

            case 12:
                flag = true;
                return '*';

            case 13:
                flag = true;
                return '0';

            case 14:
                flag = true;
                return '#';

            case 15:
                flag = true;
                return 'D';

            default:
                flag = false;
                return '0';
        }
    }
}
```

- **Buzzer:** en este fichero tenemos tres funciones principales. La primera de todas es la función `play_tone()` que es la que se llamará desde otras funciones de Arduino pasando com parámetro una cadena con el nombre de la acción que se va a realizar. En esta función tenemos declarados en arrays de tipo double las frecuencias de los sonidos que vamos a reproducir a través del buzzer. Debido a que muchas de estas son frecuencias muy altas, son sonidos muy agudos por lo que algunos de estos sonidos en un ambiente de ruido serían difícilmente perceptibles y se escucharían mejor en un entorno donde haya poco ruido. Nosotros, como ya se ha mencionado, hemos reproducido sonidos al entrar en la aplicación, al salir de ella y cuando se recibe un mensaje. Para que estos sonidos se escuchen correctamente y tengan sentido, hay que aplicar una duración concreta a cada frecuencia o nota musical.

Por lo que también hemos añadido un array de duración para cada uno de los arrays de frecuencias. Todas estas frecuencias se las pasamos en un bucle for para que se procesen una a una a la función *playNote()* que calcula a partir de la frecuencia y la duración de la misma, la pausa que se debe realizar entre nota y nota. El valor de referencia para las pausas entre notas es de 20 por lo que sería restar este valor de duración al de referencia y tendríamos el valor real de la duración de las frecuencias.

```

/* Toca el array de frecuencias que le pasamos como argumento conteniendo el conjunto de frecuencias
 * de la canción o sonido a reproducir
*/
void playNote(int noteInt, double noteFreqArr[], int noteDurationArr[], long breath = 20)
{
    noteDurationArr[noteInt] = noteDurationArr[noteInt] - breath;
    buzz(pinBuzzer, noteFreqArr[noteInt], noteDurationArr[noteInt]);
    if(breath > 0) { //take a short pause or 'breath' if specified
        delay(breath);
    }
}

/* Indicamos en el código de arduino el tipo de sonido que queremos que se reproduzca en el buffer
 * dependiendo del menú en el que estemos y la situación
*/
void play_tone(String mode)
{
    int i;
    int countTonesSend = 511;
    int countTonesRecv = 0;

    double noteFreqArrSendStartup[] = { /* StartUp */
        165.4064, 130.813, 261.626, 523.251, 1046.50, 2093.00, 4186.01
    };

    double noteFreqArrSendMario[] = { /* Mario victory */
        466, 588, 699, 933, 933, 933, 1047
    };

    double noteFreqArrSendShutdown[] = { /* ShutDown --> Delay es 100*/
        523.251, 391.995, 329.628, 261.626
    };

    int noteDurationArrSendStartup[] = { /* Startup */
        100, 100, 100, 100, 100, 100, 100
    };

    int noteDurationArrSendShutdown[] = { /* ShutDown */
        100, 100, 100, 200
    };

    int noteDurationArrSendMario[] = { /* Mario Victory */
        100, 100, 100, 300, 100, 100, 100, 400
    };

    int countTonesSendStartup = 7;
    int countTonesSendShutdown = 4;
    int countTonesSendMario = 8;
    const int pinBuzzer = 8;

    Serial.println("Playing tone in mode " + mode);

    if (mode.equals("startup")) { // Se reproduce cuando el usuario entra en la aplicación , el primer menú

        for (i = 0 ; i < countTonesSendStartup ; i++)
            playNote(i, noteFreqArrSendStartup, noteDurationArrSendStartup);

    } else if (mode.equals("recv")) { // Se reproduce cuando el usuario recibe un mensaje

        for (i = 0 ; i < countTonesSendMario ; i++)
            playNote(i, noteFreqArrSendMario, noteDurationArrSendMario);

    } else if (mode.equals("shutdown")) { // Se reproduce cuando el usuario sale de la aplicación

        for (i = 0 ; i < countTonesSendShutdown ; i++)
            playNote(i, noteFreqArrSendShutdown, noteDurationArrSendShutdown);
    }
}

```



Una vez se ha calculado la duración real de las notas, estas se pasan a otra función separada *buzz()* que modula y manda la señal a los pines correspondientes donde , a través de un bucle for, se van alternando la escritura en los pines entre alta y bajo (HIGH y LOW) según la forma de la onda de la frecuencia que se ha pasado como parámetro y de la longitud de la misma.

```
/* Una vez procesado el array de frecuencias e indicadas las duraciones y pausas, se enlaza a los pines del buzzer
 * y se reproduce el sonido
*/
void buzz(int targetPin, long frequency, long length)
{
    long delayValue = 1000000/frequency/2; // calculate the delay value between transitions

    //// 1 second's worth of microseconds, divided by the frequency, then split in half since
    //// there are two phases to each cycle
    long numCycles = frequency * length/ 1000; // Calculate the number of cycles for proper timing

    //// Multiply frequency, which is really cycles per second, by the number of seconds to
    //// get the total number of cycles to produce
    for (long i=0; i < numCycles; i++){ // For the calculated length of time...
        digitalWrite(targetPin,HIGH); // Write the buzzer pin high to push out the diaphragm
        delayMicroseconds(delayValue); // Wait for the calculated delay value
        digitalWrite(targetPin,LOW); // Write the buzzer pin low to pull back the diaphragm
        delayMicroseconds(delayValue); // Wait again for the calculated delay value
    }
}
```

- **Fichero .h de declaración de variables y pines**

```
main   GUI   Joystick   LCD   SMS_Service   buzzer   keyPad   main.h

#include <GSM.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Other variables
const byte rows=4;           // Numero de filas del keypad
const byte columns=4;         // Number of columns del keypad

/* ===== PINES DE CONEXION ===== */

// Pines usados para los LEDs. Arduino MEGA GSM shield tiene los pines 2,3,7 y 10 reservados.
const int pinLED_1 = 4;        // LED green cuando se envía el mensaje
const int pinLED_2 = 5;        // LED green cuando el mensaje es recibido por el receptor
const int pinLED_3 = 6;        // LED blue cuando el mensaje es leído por el receptor

// Pines para el joystick
const int pinEjeX = A0;
const int pinEjeY = A1;
const int pinPulsador = 9;

// Pin usados para el Buzzer
const int pinBuzzer = 8;

// Pines usados para el Keypad
const byte Output[rows]={25,27,29,31}; // Array de pines usados para indicar las filas del keypad
const byte Input[columns]={33,35,37,39}; // Array de pines usados para indicar las columnas del keypad

/* ===== TECLADO DE MEMBRANA ===== */

unsigned long kdelay=0;          // Variable used in non-blocking delay
bool flag = false;               // Flag to add a new introduced digit to the complete password

/* ===== SMS ===== */

// GSM library instances to send SMS and to access GSM service in the shield
GSM_SMS sms;

char sender_phoneNumber[20];      // Variable used to store the number of the SENDER
char receiver_phoneNumber[13];     // Variable used to store the number of the RECEIVER
char unk_receiver_phoneNumber[20]; // Variable used to store the number of a person who sends a message
                                   // and we are in the middle of a conversation with another person

int8_t answer;                  .
int value = -1;
int limit = 2;                   // Limits of the values of the joystick
int limit2 = 2;

/* ===== PANTALLA LCD ===== */

//I2C pins declaration
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

//Bytes para crear la figura 'Flecha izquierda' en el LCD
byte flechaD[8] = {
    0b000000,
    0b000100,
    0b000110,
    0b11111,
    0b00110,
    0b00100,
    0b00000,
    0b00000
};
```

## b. Python

El código se encuentra en el fichero ***whatsapp.py***. Dado que la explicación del código en Python se complementa con lo anteriormente explicado del código Arduino, hemos de añadir un detalle importante a este. Aparte del hecho de que Python, al contrario que Processing, no toma el control del puerto serie permitiendo el control al código Arduino. Si que permite escuchar de él y por tanto leer y escribir del mismo sin modificar ningún parámetro del código Arduino. Para ello tiene una función de una biblioteca en la que , pasando únicamente el puerto serie del que escuchar y la banda de frecuencia (en baudios) por la que se escucha , es suficiente.

Otro aspecto importante para el desarrollo de la aplicación de Whatsapp, aparte del hecho de poder mantener un fichero para cada uno de los contactos con las conversaciones que se han mantenido con ellos, y registrar los contactos de la aplicación en un fichero y poder manejar esa lista a nuestro antojo, es la comunicación bidireccional.

Después de documentarnos mucho acerca de la comunicación en Whatsapp y otros servicios de mensajería, vimos que otra ventaja que nos ofrecía utilizar Python era para desarrollar una comunicación bidireccional, de tal forma que pudiésemos recibir mensajes y escribir al mismo tiempo, manteniendo la fluidez en las conversaciones, del mismo estilo que se hace en Whatsapp.

Para ello hemos utilizado hilos de ejecución del sistema operativo, de tal manera que creábamos un hilo de ejecución que se encargaba únicamente de escuchar del puerto serie cuando se iniciaba una comunicación y estaba atento a cuando se recibía un mensaje remoto desde el móvil del contacto con el que estábamos hablando. Y por otro lado otro hilo de ejecución distinto, que se encargaba de atender los mensajes que nosotros escribíamos a la otra persona, de manera que había dos hilos distintos manejando dos tareas distintas al mismo tiempo, consiguiendo un chat fluido y eficiente a través del cual poder implementar la comunicación en tiempo real

```

1 #!/usr/bin/python
2
3 import serial
4 import syslog
5 import time
6 import threading
7 import re
8 import fileinput
9
10 #####
11 #
12 #           VARIABLES
13 #
14 #####
15
16
17 # The following line is for serial over GPIO
18 serial_port = "/dev/cu.usbmodem14201";
19 baud_rate = 9600; #In arduino, Serial.begin(baud_rate)
20
21 # Variables for I/O management
22 contacts_file_path = "contacts.txt"
23 chats_file = ""
24 user_chats_file_path = ""
25 unknown_chats_file_path = ""
26 chat_new_file = ""
27
28 # Connect to serial port
29 ser = serial.Serial(serial_port, baud_rate)
30 time.sleep(2) # wait for Arduino
31
32 # Other flags and message contents . As well as phone_number contacts stored variables.
33 serial_content = ""
34 new_line = ""
35 phone_number = ""
36 msg_rcv = ""
37 phone_number_contact = ""
38
39 setTempCar1 = ''
40 contact_var = ''
41 playingID = ''
42 finished = True
43 termin = False
44 flag_found = 0
45 flag_found_2 = 0
46

```

```

48 #####
49 #
50 #           FUNCTIONS
51 #
52 #####
53
54 # Function that creates another thread that is listening for new messages while principal one is
55 # waiting for input to send
56 def kbdlistener():
57     global setTempCar1, finished
58     setTempCar1 = raw_input("ttttttt::")
59     ser.flush()
60     serTemp1 = str(setTempCar1)
61     ser.write(serTemp1)
62     new_line = "ttttttt" + serTemp1 + "\n\n"
63     chats_file.write(new_line)
64     finished = True
65
66 # Handles the management of the reception of new messages from people different from the
67 # one we are talking to
68 def whatsapp_recv(phoneNumber):
69     global chat_new_file
70     if phoneNumber == phone_number_contact:
71         with open(contacts_file_path) as search:
72             for line in search:
73                 line = line.rstrip() # remove '\n' at end of line
74                 if phoneNumber in line:
75                     start = 0;
76                     end = line.find('+', start)
77                     contact = line[start:end]
78                     contact = contact.rstrip()
79                     print_msg = "*Nuevo whatsapp de " + contact + "*"
80                     flag_found = 1
81                     break
82
83                 if flag_found == 0:
84                     print_msg = "*Nuevo whatsapp del numero " + phoneNumber + "*"
85                     contact = phoneNumber
86                 else:
87                     flag_found = 0
88
89             search.close()
90
91             print(print_msg)
92             unknown_chats_file_path = "chats_" + contact + ".txt";
93             print(unknown_chats_file_path)
94             chat_new_file = open(unknown_chats_file_path, "a+");
95
96             else:
97                 phoneNumber = phoneNumber + "\n"
98                 chats_file.write(phoneNumber)

```

```

99 #####
100 #           MAIN
101 #
102 #           MAIN
103 #
104 #####
105
106 while (1):
107
108     # Serial read section
109     serial_content = ser.readline() # read all characters in buffer
110     serial_content = serial_content[:-2]
111
112     if serial_content == "phone_number": # 1- Introduce the phone number of the receiver
113
114         setTempCar1 = raw_input("Telefono: ")
115         ser.flush()
116         serTemp1 = str(setTempCar1)
117         ser.write(serTemp1)
118
119         setTempCar1 = ''
120         time.sleep(5)
121
122     elif serial_content == "messages_1": # 2- Message sender communication to another contact/numero
123
124         with open(user_chats_file_path) as search:
125             for line in search:
126                 line = line.rstrip() # remove '\n' at end of line
127                 print(line)
128
129             while(1):
130                 if playingID != setTempCar1:
131                     playingID = setTempCar1
132                 elif setTempCar1 == "exit":
133                     # Kill thread
134                     break
135                 else:
136                     if termin:
137                         serial_content = ser.readline() # Read all characters in buffer
138                         serial_content = serial_content[:-2]
139                         if serial_content == "recv_on":
140                             new_line = ""
141                             phone_number = ser.readline() # Reads the phone number first
142                             phone_number = phone_number[:-3]
143                             print( (phone_number_contact) # -- DEBUG --
144
145                             whatsapp_rcv(phone_number)
146
147                             while (1): # Reads the message of the receiver
148                                 chr_rcv = ser.readline()
149                                 chr_rcv = chr_rcv[:-2]
150                                 if chr_rcv == "recv_of":
151                                     break # Stops reading the message
152                                 if phone_number == phone_number_contact:
153                                     print(chr_rcv)
154                                     msg_rcv = msg_rcv + chr_rcv
155
156                                     msg_rcv = msg_rcv + "\n"
157
158                                     if phone_number == phone_number_contact:
159                                         chats_file.write(msg_rcv)
160                                     else:
161                                         print(chat_new_file)
162                                         chat_new_file.write(msg_rcv);
163                                         chat_new_file.close()
164
165             if finished: # Creates the new thread for listening to upcoming messages
166                 finished = False
167                 termin = True
168                 listener = threading.Thread(target=kbdl.listener)
169                 listener.start()
170
171             time.sleep(2)
172
173     elif serial_content == "recv_on": # 2- A new message has been received and has to be treated
174
175     new_line = ""
176     phone_number = ser.readline() # Reads the phone number first
177     print( (phone_number)
178     phone_number = phone_number[:-3]
179     msg_to_print = "El nro es " + phone_number
180     print( (phone_number_contact)
181
182     whatsapp_recv(phone_number)
183
184     while (1): # Reads the message of the receiver
185         chr_rcv = ser.readline()
186         chr_rcv = chr_rcv[:-2]
187         if chr_rcv == "recv_of":
188             break
189         if phone_number == phone_number_contact:
190             print(chr_rcv)
191
192         msg_rcv = msg_rcv + chr_rcv
193
194         msg_rcv = msg_rcv + "\n"
195         if phone_number == phone_number_contact:
196             chats_file.write(msg_rcv)
197         else:
198             chat_new_file.write(msg_rcv);
199             chat_new_file.close()
200
201     elif serial_content == "contacts_transfer": # 4- Load the contacts and sends it to arduino
202
203     contacts_file = open(contacts_file_path, "r");
204
205     for x in contacts_file:
206         ser.write(x)
207         time.sleep(5)
208
209     contacts_file.close()
210
211     elif serial_content == "contacts_msg": # 5- Send message to a contact
212
213     contact_var = raw_input("Contacto: ")
214     ser.flush()
215     serTemp1 = str(contact_var)
216
217     with open(contacts_file_path) as search:
218         for line in search:
219             line = line.rstrip() # remove '\n' at end of line
220             if serTemp1 in line:
221                 print("Contacto encontrado")
222                 print(line)
223                 start = line.find('+')
224                 end = line.find('*', start)
225                 phone_number_contact = line[start:end]
226                 ser.write((line[start:end]))
227                 flag_found_2 = 1
228                 break
229
230                 if flag_found_2 == 0:
231                     print("El contacto introducido no existe")
232                 else:
233                     flag_found_2 = 0
234
235     setTempCar1 = ''
236     search.close()
237
238     user_chats_file_path = "chats_" + contact_var + ".txt";
239     chats_file = open(user_chats_file_path, "a+");
240
241     elif serial_content == "contacts_add": # 6- Add a new contact to the list of contacts
242
243     contacts_file = open(contacts_file_path, "a+");
244
245     print("Introduzca el nombre y telefono movil del nuevo contacto")
246     name_contact = raw_input("Nombre: ")
247     phone_contact = raw_input("Telefono: ")
248     row = name_contact + " " + phone_contact + "\n"
249
250     contacts_file.write(row)
251
252     contacts_file.close()
253
254

```

```

252
253     elif serial_content == "contacts_delete": # 7- Delete an existing contact from the list of contacts
254
255         print("Introduzca el nombre del contacto a eliminar")
256         name_contact = raw_input("Nombre: ")
257
258         with open(contacts_file_path, "r") as f:
259             lines = f.readlines()
260
261         with open(contacts_file_path, "w") as f:
262             for line in lines:
263                 if name_contact not in line.strip("\n"):
264                     f.write(line)
265
266         f.close()
267
268     elif serial_content == "contacts_modify": # 8- Modify a contact from the list of contacts
269
270         print("Introduzca el nombre del contacto a modificar")
271         name_contact = raw_input("Nombre: ")
272         new_name_contact = raw_input("Nuevo nombre: ")
273         new_phone_contact = raw_input("Nuevo telefono: ")
274
275         with open(contacts_file_path) as search:
276             for line in search:
277                 line = line.rstrip() # remove '\n' at end of line
278                 if name_contact in line:
279                     #print(line)
280                     start = line.find('+')
281                     end = line.find('*', start)
282                     phone_contact = line[start:end]
283
284             # Read in the file
285             with open(contacts_file_path, 'r') as file :
286                 filedata = file.read()
287
288             # Replace the target string
289             if new_name_contact != '':
290                 filedata = filedata.replace(name_contact, new_name_contact)
291
292             if new_phone_contact != '':
293                 print(phone_contact)
294                 print(new_phone_contact)
295                 filedata = filedata.replace(phone_contact, new_phone_contact)
296
297             # Write the file out again
298             with open(contacts_file_path, 'w') as file1:
299                 print("Escribir")
300                 print(filedata)
301                 file1.write(filedata)
302
303             search.close()
304             file.close()
305             file1.close()
306
307     elif serial_content == "terminate": # 9- Terminate the application and kill all threads
308         print "Exiting"
309         exit()
310     else:
311         print (serial_content)

```

## 5. Dificultades encontradas

Durante la realización de la aplicación propuesta fuimos encontrando diversas dificultades que conllevaron la toma de ciertas decisiones en previos apartados explicadas. La resolución de los problemas, así como la elección de ciertas versiones ha conllevado tiempo y esfuerzo considerable debido al carácter cíclico del proceso (ciertas decisiones no resultaron fructíferas por lo que se desecharon y se tuvo que volver atrás, por ejemplo). Entre las dificultades, no todas solventadas, podemos encontrar:

- La incompatibilidad de las librerías *GSM.h* y *Keypad.h* en Arduino. Dichas librerías se utilizan para el manejo del módulo *GSM Shield 2* y el teclado de membrana respectivamente. El problema encontrado es que en la implementación de las mismas existen macros definidas bajo el mismo nombre. Para solucionar esto optamos por la implementación manual de las labores necesitadas para el manejo hecho del teclado de membrana siguiendo las indicaciones de diversas páginas que podemos observar en el apartado *Referencias*.
- La conectividad red GPRS/GSM telefónica del módulo *GSM Shield 2* ha resultado en ocasiones ineficiente con el correspondiente tiempo perdido para la búsqueda de esta debido principalmente al comportamiento errático o inesperado del mismo, ya que depende fundamentalmente de la cobertura que haya en el lugar en el que se conecte. Para lograr esta conectividad hemos tenido que realizar algunas conexiones en la placa, así como modificaciones físicas en la misma. En primer lugar, hemos debido puentejar los pines 2 y 10 porque así se dispone para que el GSM trabaje con la versión MEGA del

Arduino. En segundo lugar, hemos doblado la pata del *GSM Shield 2* correspondiente al pin 2, esto es debido a que el modelo usado, *Arduino Mega*, tiene reservados ciertos pines. Para más detalles consultar las referencias.

- Restricción y falta de pines en la placa *Arduino Uno* para la cantidad de periféricos de entrada/salida implementados en nuestro trabajo por lo que hemos optado por la utilización de una placa con mayor nº de pines como es *Arduino Mega*, teniéndola prestada de otros alumnos.
- Búsqueda complicada, debido a la falta de información, acerca de los comandos AT para el envío de mensajes multimedia (MMS) en nuestro trabajo. Finalmente decidimos no mandar este contenido, por una parte, por falta de información y por otra, por el no reconocimiento del nuestro modelo *M10* obteniendo siempre “error”. Como causas encontradas de este error hemos encontrado que la librería necesaria para estos comandos, *SoftwareSerial.h*, es incompatible con la librería *GSM.h* por el mismo motivo que la incompatibilidad recogida en el primer punto de este mismo apartado. Creemos además que los comandos AT son aceptados en mayor medida por otros módulos como pueden ser *SIM800* y *SIM900*.
- Aprendizaje de nuevos lenguajes de programación como *Python* y *Processing*, con el necesario periodo de adaptación y pruebas que conlleva aprender nuevas sintaxis. Aunque finalmente solo hemos utilizado Python para lograr la persistencia de datos.
- Hemos descartado el uso de *Processing* porque, aunque lo consideramos un lenguaje más adecuado y adaptable a este entorno de trabajo, ocupaba el puerto serie a través del cual conectamos la placa Arduino con el ordenador. Entonces, bien solo podíamos mantener abierto el monitor puerto serie o el *Processing* debido a que ambos hacen uso de dicho puerto.
- El uso de la librería *SoftwareSerial.h*, además de las incompatibilidades previamente mencionadas con otras librerías, nos ha supuesto más problemas. Esta librería añade un puerto serie lógico adicional al físico. Cuando utilizamos los comandos AT nuestro propósito era mandarlos por el puerto físico obteniendo las respuestas por el puerto lógico. Hemos observado que nuestro modelo *GSM M10* al tratarse de una versión más moderna de *GSM* trabaja a más alto nivel suponiendo esto tratar los comandos AT como cadenas de texto en vez de como comandos (órdenes) propiamente dichos como sí hacen otros modelos como los ya mencionados *SIM800* y *SIM900*.
- El joystick usado resultó una complicación debido a la complejidad del mismo. En primer lugar, se debe calibrar resultando cada modelo diferente en el calibrado. En segundo lugar, el joystick conlleva un retardo desde que se mueve hasta que la señal producida surge efecto sobre todo en el desplazamiento de textos en el LCD. En último lugar, el descubrimiento de cómo detectar los movimientos en los ejes X e Y así como la pulsación conllevó un proceso de investigación laborioso.

## 6. Bibliografía

1. Página oficial de Arduino, <https://www.arduino.cc/en/Guide/Environment#toc8>
2. Página Librería GSM Arduino, <https://www.arduino.cc/en/Reference/GSM>
3. Página Librería LiquidCrystal, <https://www.arduino.cc/en/Reference/LiquidCrystal>
4. Tutorial Manejo LCD 16x2, <https://www.youtube.com/watch?v=R-CRIthB7ZY>
5. Foro Arduino AT commands, <https://forum.arduino.cc/index.php?topic=362390.0>
6. Tutorial Funcionamiento LCD, <https://www.youtube.com/watch?v=VMIRy5zuLqQ>
7. Mapa de pines del Arduino MEGA,  
<https://www.arduino.cc/en/Hacking/PinMapping2560>
8. Pines reservados del Arduino MEGA para conexión con el GSM,  
<https://www.arduino.cc/en/Guide/ArduinoGSMShield>
9. Como enviar mensajes de Whatsapp desde Arduino,  
<https://www.instructables.com/id/Arduino-Whatsapp-Messages-Send-Whatsapp-Messages-U/>
10. Como implementar Keypad sin la biblioteca Keypad.h,  
<https://www.instructables.com/id/Keypad-With-Arduino-Without-USing-Keypad-library-F/>
11. ¿Cómo funciona Whatsapp?. Servicio de mensajería privado  
<https://www.xatakamovil.com/movil-y-sociedad/por-que-las-conversaciones-de-whatsapp-no-pueden-ser-admitidas-como-prueba-concluyente-en-un-juicio>
12. Montaje y explicación funcionamiento LCD,  
<https://www.allaboutcircuits.com/projects/interface-an-lcd-with-an-arduino/>
13. Montaje y explicación funcionamiento Buzzer,  
<https://www.luisllamas.es/reproducir-sonidos-arduino-buzzer-pasivo-altavoz/>
14. Frecuencias Buzzer Sonido1,  
<https://github.com/ShaneMcC/beeps/blob/master/mario-victory.sh>
15. Frecuencias Buzzer Repositorio, <https://github.com/evertarias/EasyBuzzer/tree/master/src>
16. Manejo de ficheros en Python, como extraer porciones de un texto,  
<https://www.computerhope.com/issues/ch001721.htm>
17. Filtrar palabras en un texto con Python,  
<https://stackoverflow.com/questions/4940032/how-to-search-for-a-string-in-text-files>
18. Página web de tonos musicales,  
<https://www.zedge.net/find/ringtones/whatsapp%20message>
19. Conexión de un teclado de ordenador a una placa Arduino,  
<https://www.instructables.com/id/Connect-Arduino-UNO-to-USB-Keyboard/>
20. Como eliminar una línea de un fichero con Python,  
<https://stackoverflow.com/questions/4710067/using-python-for-deleting-a-specific-line-in-a-file>
21. Pentagrama del tono del Iphone, <https://www.xatakamovil.com/apple/partitura-del-tono-del-iphone>
22. Como manejar entrada de información por el puerto serie al Arduino,  
<http://www.ardulink.org/how-to-manage-input-from-arduino/>
23. Como guardar información en el ordenador usando Arduino,  
[https://www.researchgate.net/post/How\\_can\\_I\\_save\\_data\\_directly\\_in\\_the\\_pc\\_using\\_an\\_Arduino\\_UNO](https://www.researchgate.net/post/How_can_I_save_data_directly_in_the_pc_using_an_Arduino_UNO)
24. Como guardar información en un fichero de texto con Arduino,  
<https://electronics.stackexchange.com/questions/54/saving-arduino-sensor-data-to-a-text-file>
25. Funcionamiento de la comunicación sobre el puerto serie entre Python y Arduino,  
<https://pyserial.readthedocs.io/en/latest/shortintro.html>
26. Como formar una subcadena de una cadena en Python,

<https://stackoverflow.com/questions/663171/how-to-substring-a-string-in-python>

27. Como enviar MMS o mails a través de Arduino,  
<https://forum.arduino.cc/index.php?topic=128965.0>
28. Control y funcionamiento de un Joystick, <https://www.luisllamas.es/arduino-joystick/>
29. Funcionamiento de un joystick en un LCD,  
<https://www.youtube.com/watch?v=ztsKNyuUmzM>
30. Montaje LCD con joystick, <https://www.instructables.com/id/Arduino-Joystick-Breadboard-with-LCD-Output/>