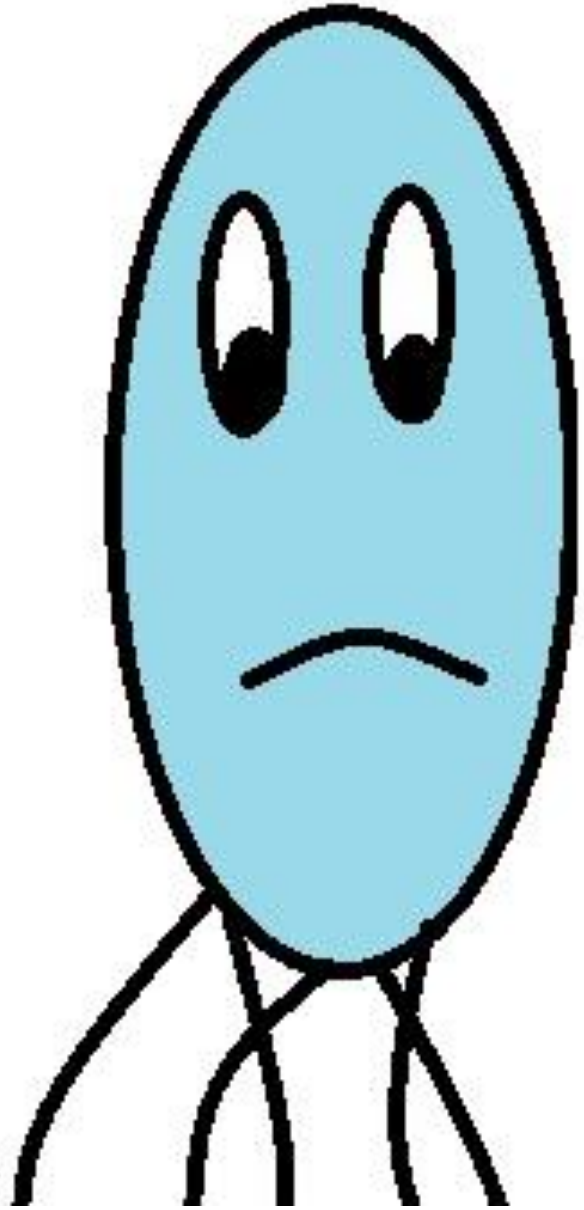


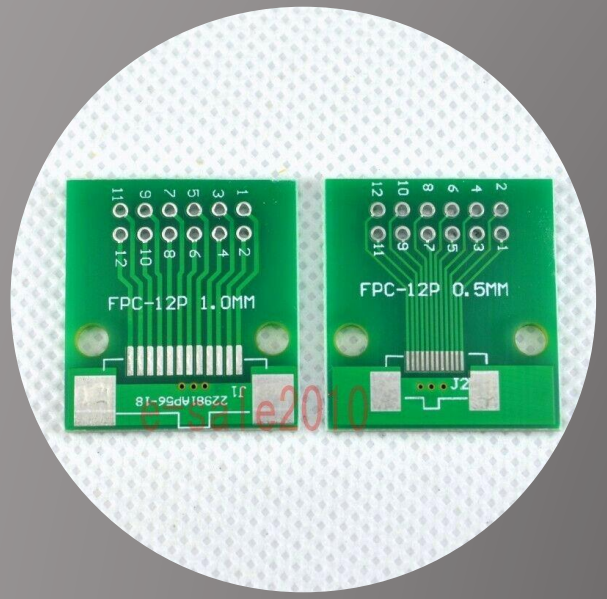
InmunoGame

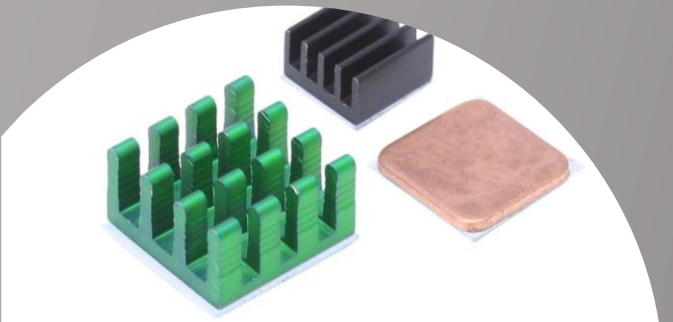
Diario de a bordo

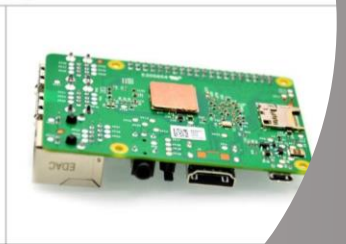
Paso 1. Configuración de la Raspberry Pi

Nuestro punto de partida: Raspberry Pi 3 Model B+

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)





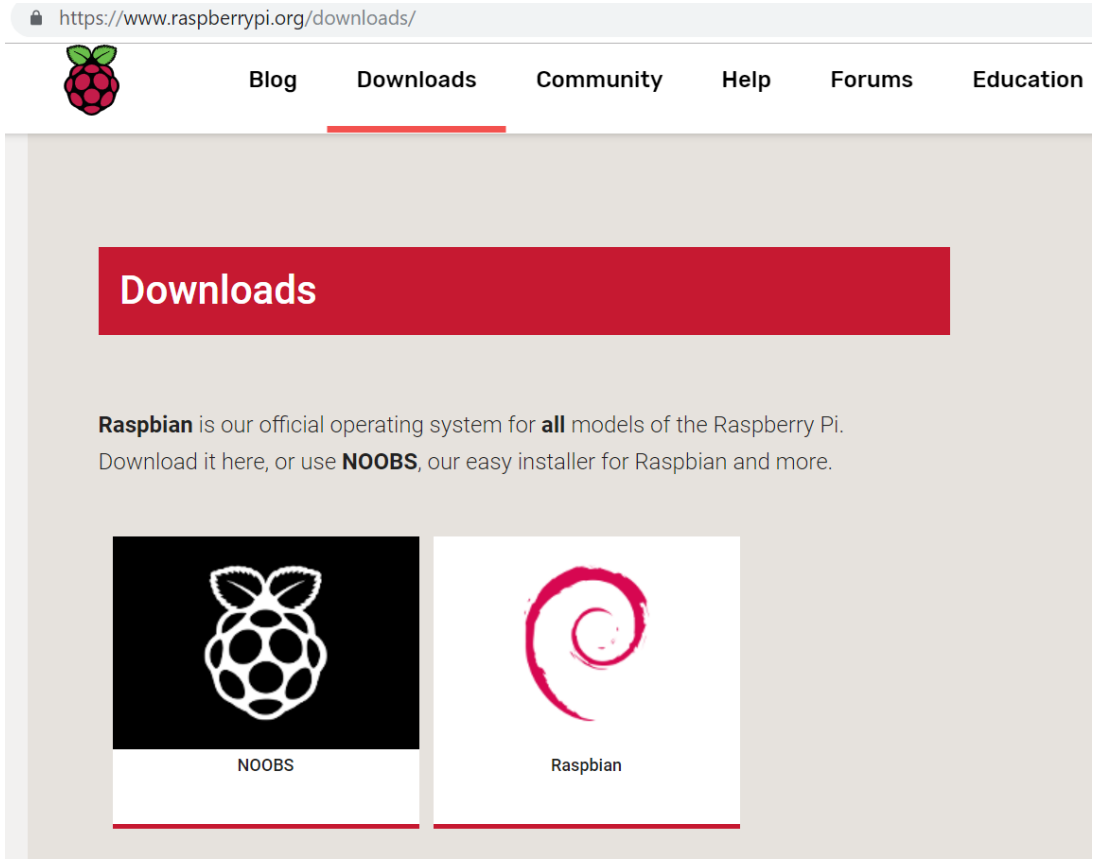


- Al mirar nuestro lector de QR, nos damos cuenta de que va a ser difícil soldar los extremos de los pins del cable flex en nuestra Raspberri PI.

- Tras indagar en la red, vemos que la mejor forma es comprar un adaptador para facilitar este paso

- Nos damos cuenta que nos faltan también los disipadores para la Raspberri Pi

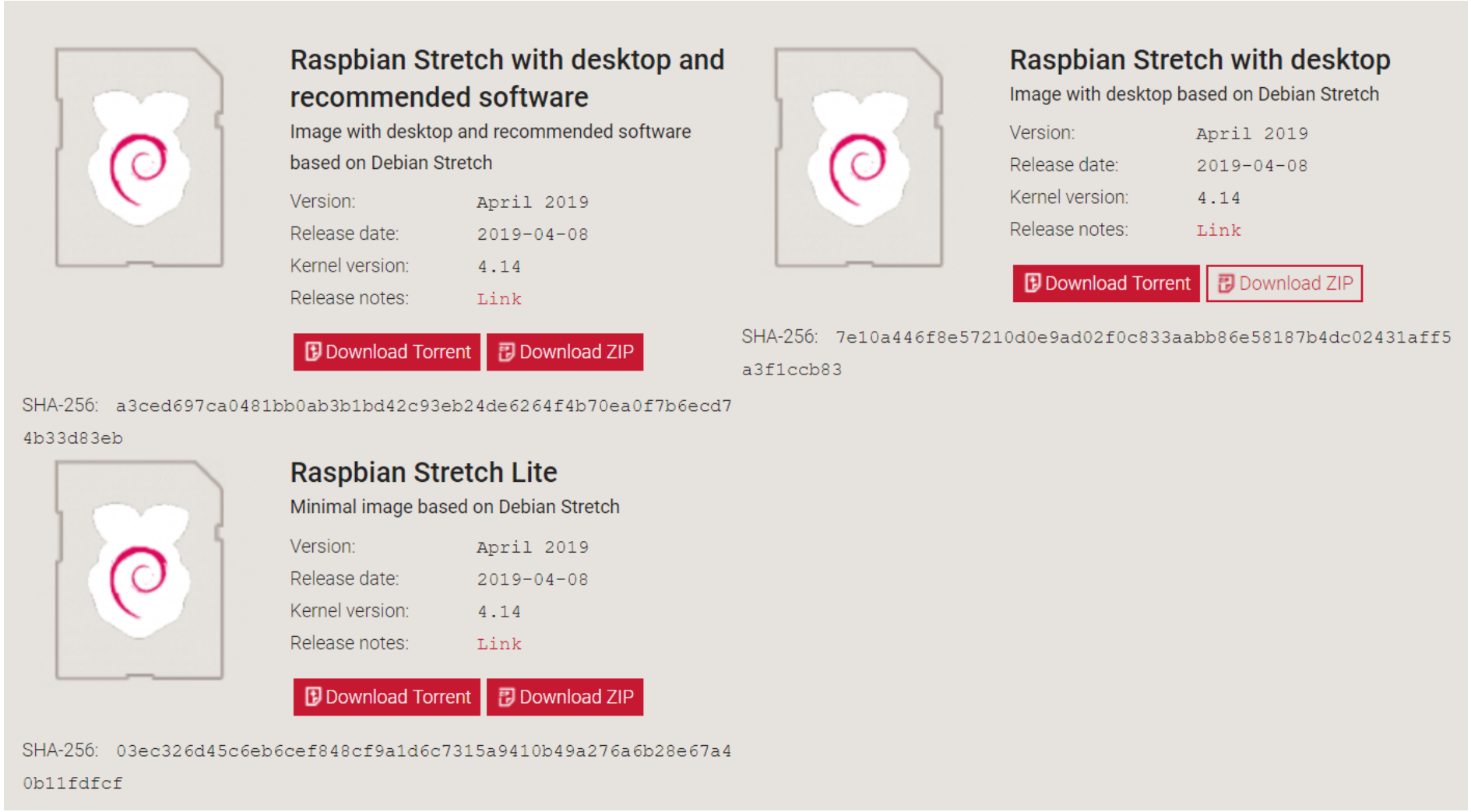
Descarga del sistema operativo para la Raspberry Pi.

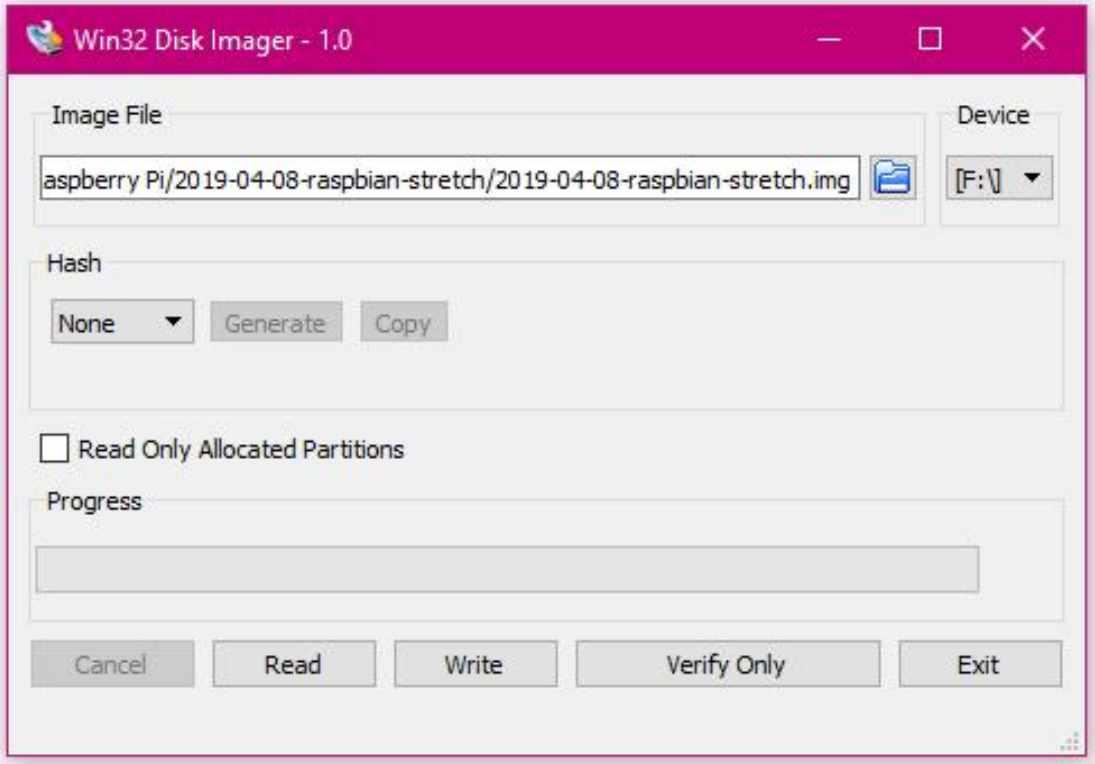


Raspbian es el Sistema operative official de la Fundación Raspberry Pi. Se puede seleccionar descargarlo directamente o utilizer NOOBS que es un instalador que permite elegir entre una serie de sistemas operativos alternativos.

Descargo el Raspbian con escritorio, se recomienda utilizar una tarjeta de al menos 8 GB. En nuestro caso utilizamos una de 16 GB.

Descargamos el zip

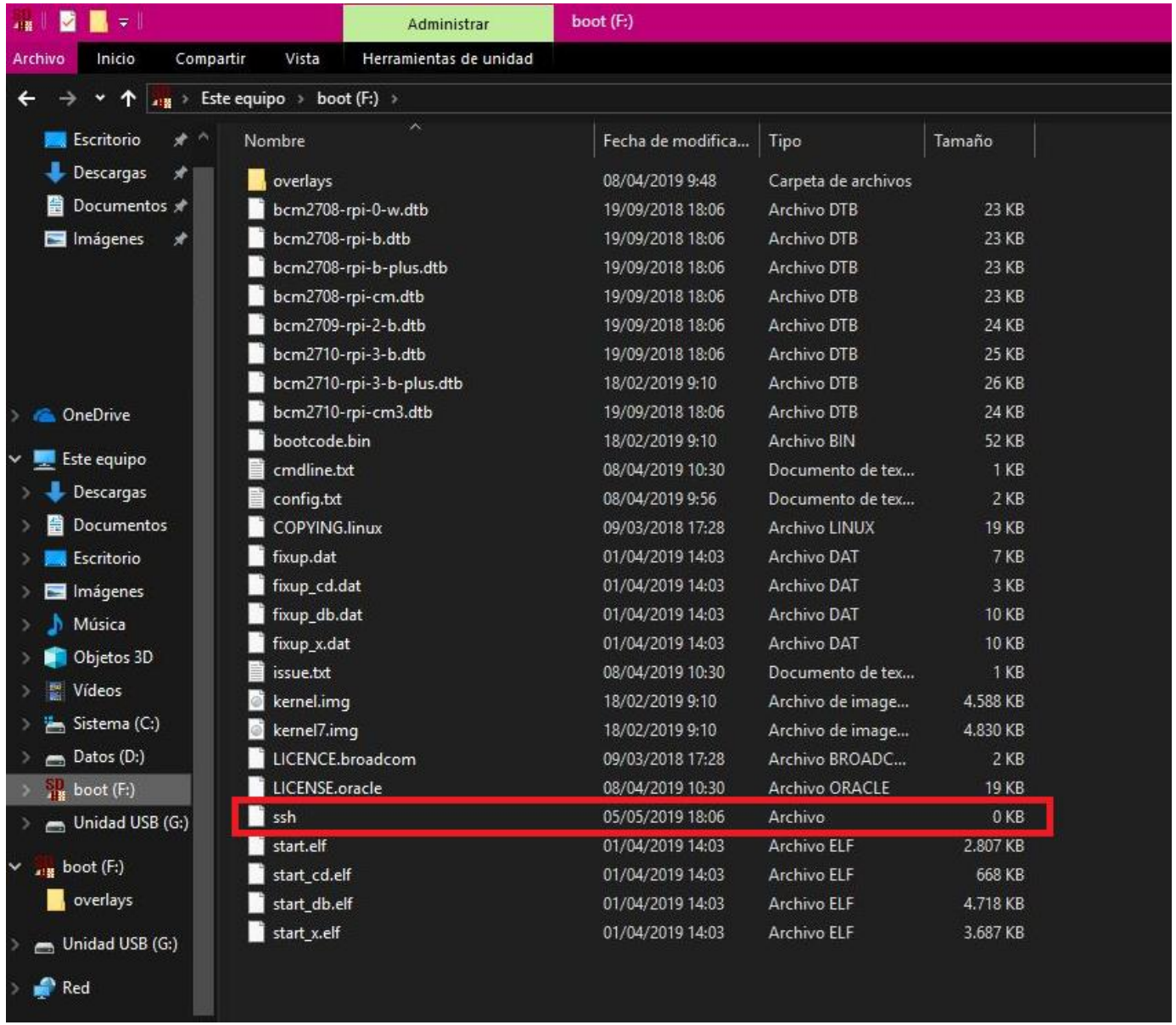




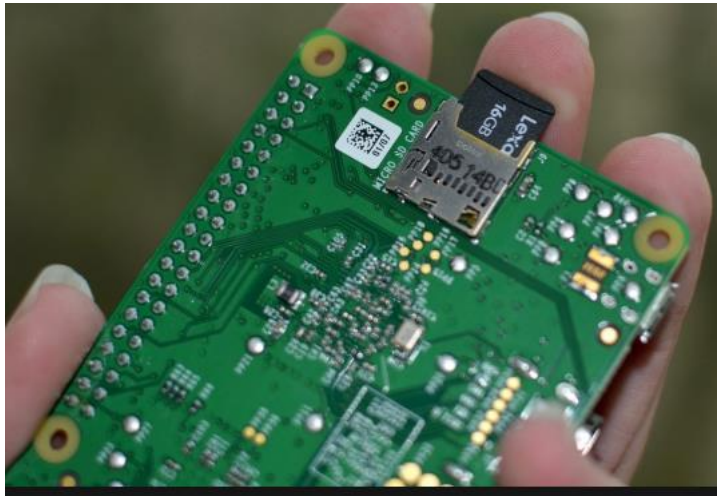
Una vez descargado el zip y se descomprime.

Con el programa Win32 Disk Imager se copia la imagen de Raspbian en nuestra tarjeta.

En la tarjeta de memoria donde hemos quemado la imagen del Sistema, aparecerán varias unidades. En la unidad boot de la tarjeta hay que crear un fichero ssh sin extension



Se introduce la tarjeta en la Raspberry Pi.



Después se prueba, para ello se conecta a un monitor gracias a la salida HDMI y se configura el idioma y se le añade la contraseña.

Después configuramos la wifi y nos instala actualizaciones.

Tras reiniciar, conectamos los altavoces por el Puerto USB y por el Jack y probamos que funcionen poniendo un video de youtube.

Añadimos los disipadores a la Raspberri Pi

Paso 2. Generación de hola mundo en Javascript of Things.

Desde la línea de comandos, actualizamos la Raspberry Pi.

Actualizamos los paquetes del Raspbian:

```
$ sudo apt full-upgrade –y
```

Actualizamos los paquetes que haya instalados en la Raspberri Pi:

```
$ sudo apt-get update
```

Instalamos VNC por si alguna vez necesitamos conectarlo a nuestro ordenador:

```
$ sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer
```

Eliminar versiones viejas de Node que pueda tener por defecto:

$ sudo apt-get remove nodered –y

$ sudo apt-get remove nodejs nodejs-legacy –y

Se bajan los paquetes del node y del npm

$ curl -sL http://deb.nodesource.com/setup_11.x | sudo bash -

Instalamos node, versión 8.11.1 y npm:

$ sudo apt-get install -y nodejs

Comprobamos versiones de node y de npm

$ node –v → 11.15

$ npm –v → 6.7.0

Entrar con el usuario root y dar permisos a cualquiera en el directorio de node_modules.

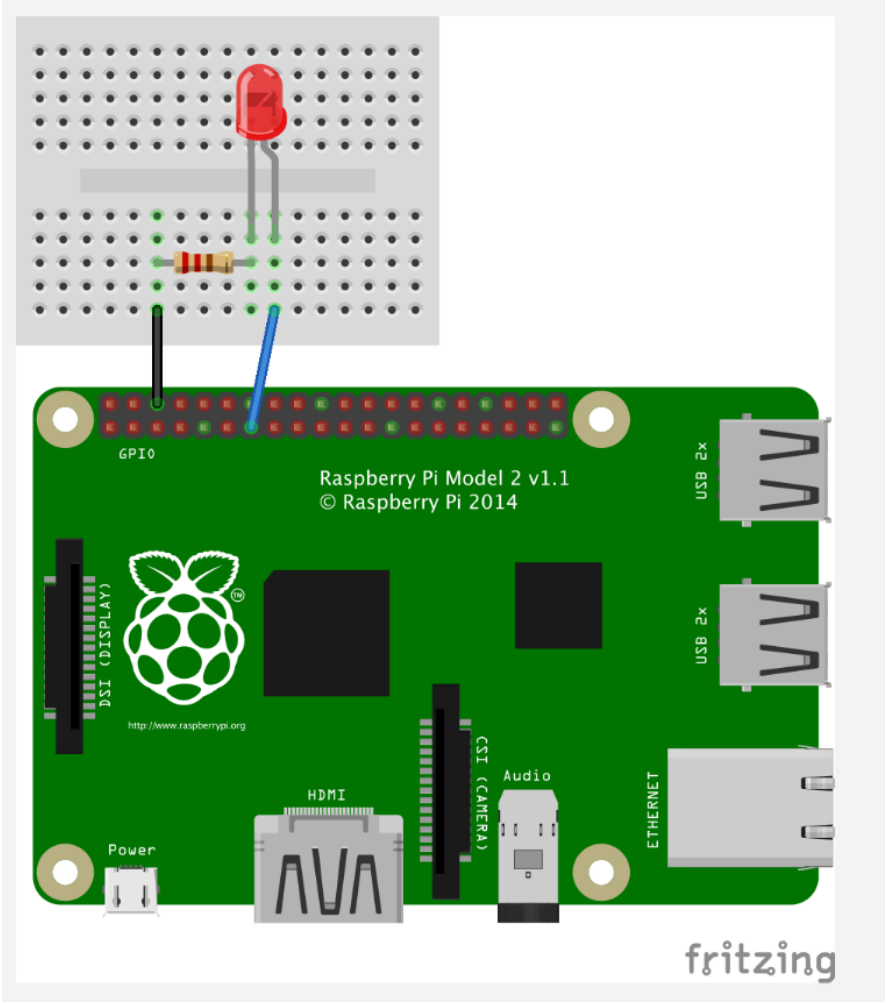
Instalamos Johnny Five:

```
$ npm install johnny-five –g
$ npm install johnny-five raspi-io -g
```

Creamos un hola mundo con un led (http://johnny-five.io/examples/raspi-io/)

Se codifica el siguiente archivo js:

```
var Raspi = require("raspi-io").RaspiIO;
var five = require("johnny-five");
var board = new five.Board({ io: new Raspi() });
board.on("ready", function() { var led = new
five.Led('P1-7'); led.blink(); });
```



Copiamos en un usb el Hola Mundo Led y se lo pasamos a la Raspberri Pi.

Se ejecuta por la consola con:

```
$ sudo node nombreArchivo
```

Paso 3. Speech to Text

Antes de instalar cualquier sistema tts:

- ❑sudo apt-get update
- ❑sudo apt-get upgrade
- ❑sudo apt-get install alsa-utils
- ❑si no hay sonido en la raspberry pi se lanza este comando:

  sudo nano /etc/modules
- ❑Esto abre el fichero y se le añade la línea:

  snd_bcm2835
- ❑sudo apt-get install mplayer
- ❑Para evitar errores de mplayer se lanza este comando:

  sudo nano /etc/mplayer/mplayer.conf
- ❑Esto abre el fichero y se le añade la línea

  nolirc=yes

Festival Speech Synthesis System

Framework para crear sistemas de síntesis de voz.

Ofrece text to speech para un número grande de APIs

Es multi-lenguaje, aunque el que tienen mas Avanzado es el inglés.

Es software libre.

Se distribuye bajo la licencia X11-type.

- Instalamos Festival:

```
sudo apt-get –y install festival
```

- Luego en la consola se ejecuta el idioma por defecto:

```
echo “My message” | festival - -tts
```

¡Y ya suena por los altavoces!

Modificaciones para ejecutar Festival desde archivo

Se instalan las voces de castellano:

sudo apt-get install festvox-ellpc11k

Modificación archivo language_castillian_spanish.scm si no funciona:

(define (language_castillian_spanish)
"(language_spanish)
Set up language parameters for Castillian Spanish."
(voice_el_diphone)
(set! male1 (lambda () (voice_el_diphone)))
(Parameter.set 'Language 'spanish)
)
(language.names.add 'castillian_spanish (list 'spanish 'castellano))

Ejecución de texto desde línea de comandos

- echo “Mi mensaje” | festival –tts - - language spanish

Ejecución de texto desde línea de comandos leyendo de archivo

- festival - - language Spanish - - tts /home/pi/Desktop/Spanish.txt

Ejecutar Festival en Castellano desde Javascript

Creamos un archivo Javascript con el siguiente contenido:

```
var sys = require('sys');
var exec = require('child_process').exec;
function puts(error, stdout, stderr) {
        sys.puts(stdout)
}
exec("echo 'Hola Mundo' | festival --tts --language
spanish", puts);
```

CUANDO LA RASPBERRY PI NO SUENA

Hemos estado peleando tres días hasta conseguir que funcionase Festival en castellano. Los primeros errores venían del archive: language_castillian_spanish.scm.

Mientras modificabamos y probabamos, nos dimos cuenta que si Festival daba un error, se quedaba colgado y era necesario reiniciar la Raspberry Pi para que volviese a funcionar.

Además, a veces, no sonaban los altavoces. Esto ocurre porque se desconfigura la tarjeta de sonido cuando la tenemos conectada a una pantalla por conexión HDMI, tiende a enviarle el sonido a la pantalla. Para que esto no ocurra se ejecuta el siguiente comando:

```
amixer cset numid=3 1
```

El valor detrás del numid si es 0 significa automático, si es 1 fuerza que la salida sea por el Jack y, siendo 2 la salida es por HDMI

Decidimos probar la integración de Johnny Five con nuestra librería Text To Speech, para ello implementamos un programilla básico que enciende y apaga un led en función de un botón. Una vez funciona añadimos a la funcionalidad del botón leer un texto, pero nos bloquea la Raspberry Pi y no suena.

```
// johnny five
var Raspi = require('raspi-io').RaspiIO;
var five = require('johnny-five');
var tToSpeech = require('./scripts/textToSpeech');


 var board = new five.board({
    io: new raspi()
 });
```

```
board.on('ready', function () {
      var led = new five.led('p1-12');
      button = new five.button('p1-16');
      var ledonoff = new five.led('p1-18');
      led.on();

      board.repl.inject({
        button: button
       });

      // button event api
        button.on("down", function() {
                        console.log("down");
        turnonled(ledonoff);
       });


      // "up" the button is released
      button.on("up", function() {
                                console.log("up");
          turnoffled(ledonoff);
       });

 });

function turnOnLed(led) {
    // tToSpeech.talkMessage("Hola Mundo", "spanish");
    ESpeak.speak("Hola mundo!");
    led.writeSync(1);
}

function turnOffLed(led) {
  // tToSpeech.talkMessage("Adios Mundo", "spanish");
    ESpeak.speak("Adios mundo!");
  led.writeSync(0);
}
```

Y lo mismo con Gpio, pero tampoco conseguimos que funcione la librería Text to Speech

```
// control puertos raspberry pi
var Gpio = require('onoff').Gpio;

var Raspi = require('raspi-io').RaspiIO;
var tToSpeech = require('./scripts/textToSpeech');

// control serial port
const SerialPort = require('serialport');
const ReadLine = require('@serialport/parser-readline');
const port = new SerialPort('/dev/serial0', { //
/dev/ttyAMA0
  baudRate: 9600,
  dataBits: 8,
  parity: 'none',
  stopBits: 1,
  flowControl: false,
  autoOpen: false
})

// pruebas GPIO
var Led = new Gpio(18, 'out');
Led.writeSync(1);

var ledOnOff = new Gpio(24, 'out');

var pushButton = new Gpio(23, 'in', 'both');
(0);
}
```

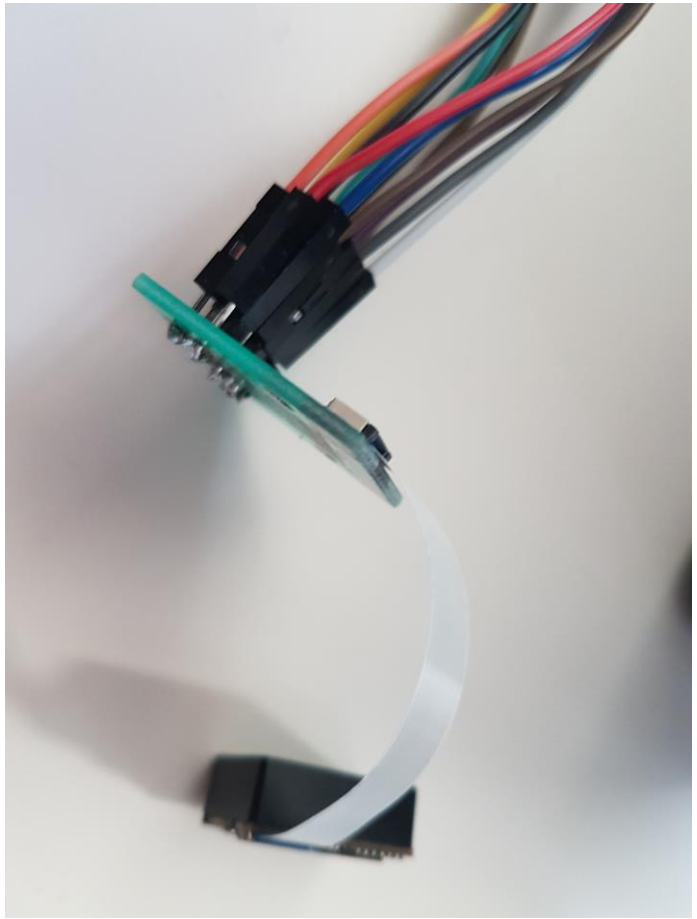
```
pushButton.watch(function (err, value) {
  if (err) { //if an error
    console.error('There was an error', err); //output error message
to console
    return;
  }

  if (value === 1) {
     turnOnLed(ledOnOff);
  } else {
     turnOffLed(ledOnOff)
  }
});

function unexportOnClose() { //function to run when exiting
program
  LED.writeSync(0); // Turn LED off
  LED.unexport(); // Unexport LED GPIO to free resources
  pushButton.unexport(); // Unexport Button GPIO to free resources
};



function turnOnLed(led) {
    // tToSpeech.talkMessage("Hola Mundo", "spanish");
    ESpeak.speak("Hola mundo!");
    led.writeSync(1);
}

function turnOffLed(led) {
  // tToSpeech.talkMessage("Adios Mundo", "spanish");
    ESpeak.speak("Adios mundo!");
  led.writeSync
}
```

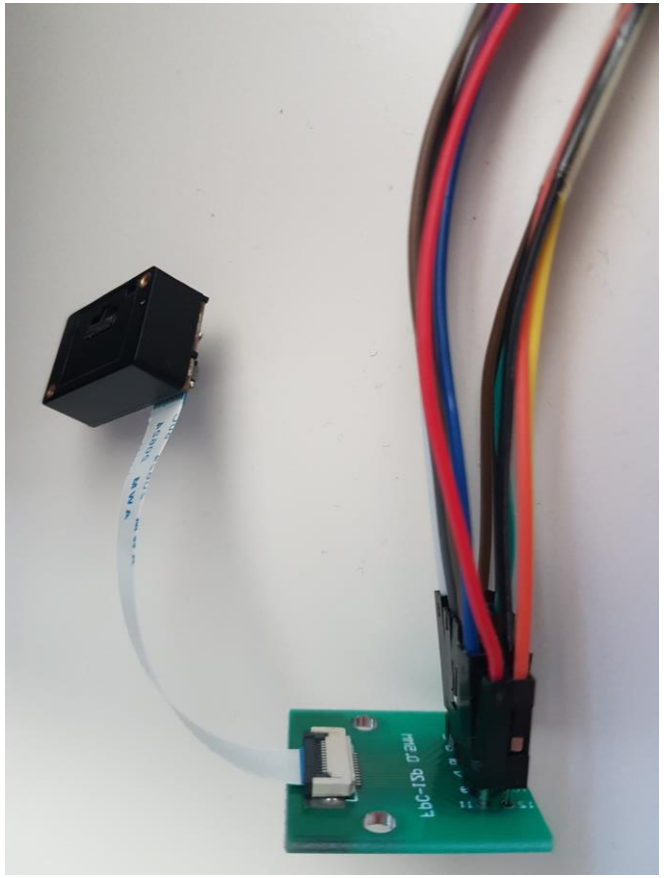
Finalmente lo conseguimos con la librería TTS Pikospeak y con GPIO.

```
// control puertos raspberry pi
var Gpio = require('onoff').Gpio;


// johnny five
var Raspi = require('raspi-io').RaspiIO;
var five = require('johnny-five');
// var tToSpeech = require('./scripts/textToSpeech');
// Require the module
var picoSpeaker = require('pico-speaker');



// control serial port
const SerialPort = require('serialport');
const ReadLine = require('@serialport/parser-readline');
const port = new SerialPort('/dev/serial0', { // /dev/ttyAMA0
baudRate: 9600,
dataBits: 8,
parity: 'none',
stopBits: 1,
flowControl: false,
autoOpen: false
})
```

```
// pruebas GPIO
var Led = new Gpio(18, 'out');
Led.writeSync(1);


var ledOnOff = new Gpio(24, 'out');


var pushButton = new Gpio(23, 'in', 'both');


// Espeak definition
// Define configuration
var picoConfig = {
LANGUAGE: 'es-ES'
};
// Initialize with config
picoSpeaker.init(picoConfig);
led.writeSync(1);
// Say hello
picoSpeaker.speak('La palabra inmune proviene del latin inmunis,
¿que os parece?').then(function() {
console.log("Hola done");
}.bind(this));
}


function turnOffLed(led) {
// tToSpeech.talkMessage("Adios Mundo", "spanish");
led.writeSync(0);
//picoSpeaker.speak('Adios mundo!').then(function() {
// console.log("adios done");
// }.bind(this));
}
```

Paso 4. Lector QR → prueba fallida





Soldamos los cables dupont al circuito impreso para conectar el lector QR a la Raspberri Pi.

Comenzamos el finde con varios pasos importantes:

- Con el programa Win32 Disk Imager se copia la imagen de Raspbian con lo instalado actualmente a modo de copia de seguridad.

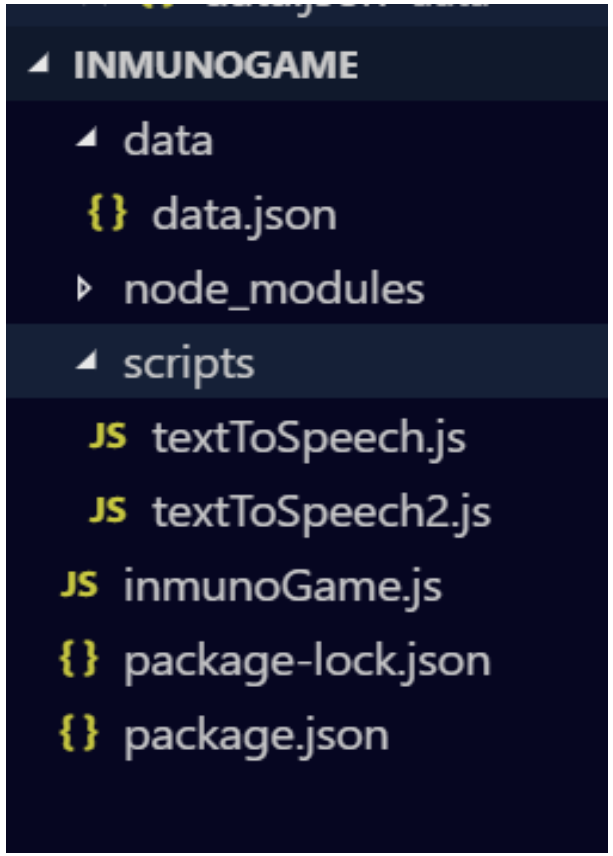
- Por otro lado buscamos los datos que va a necesitar el inmunogame: las preguntas fáciles, las difíciles y la información de saber mas.

- Buscamos un generador y lector de QRs para generar un primer QR que nos sirva para probar nuestro lector:

  http://www.codigos-qr.com/generador-de-codigos-qr/

  http://www.codigos-qr.com/lector-qr-online/

Instalamos Johnny five a nivel de proyecto y creamos la arquitectura del mismo.



Y comprobamos que siga funcionando el text to speech dentro de nuestro archivo JavaScript

Unir lector QR a Raspberry Pi

Lo primero es buscar en la página de Raspberry Pi el esquema con los números de cada PIN de la Raspberry.
https://www.raspberrypi.org/documentation/usage/gpio/

Luego en la página de w3cschool buscamos según el número de pin para que sirve cada uno. En rojo nos marca los de tensión e indica sus voltios y en azul los de tierra.

https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp

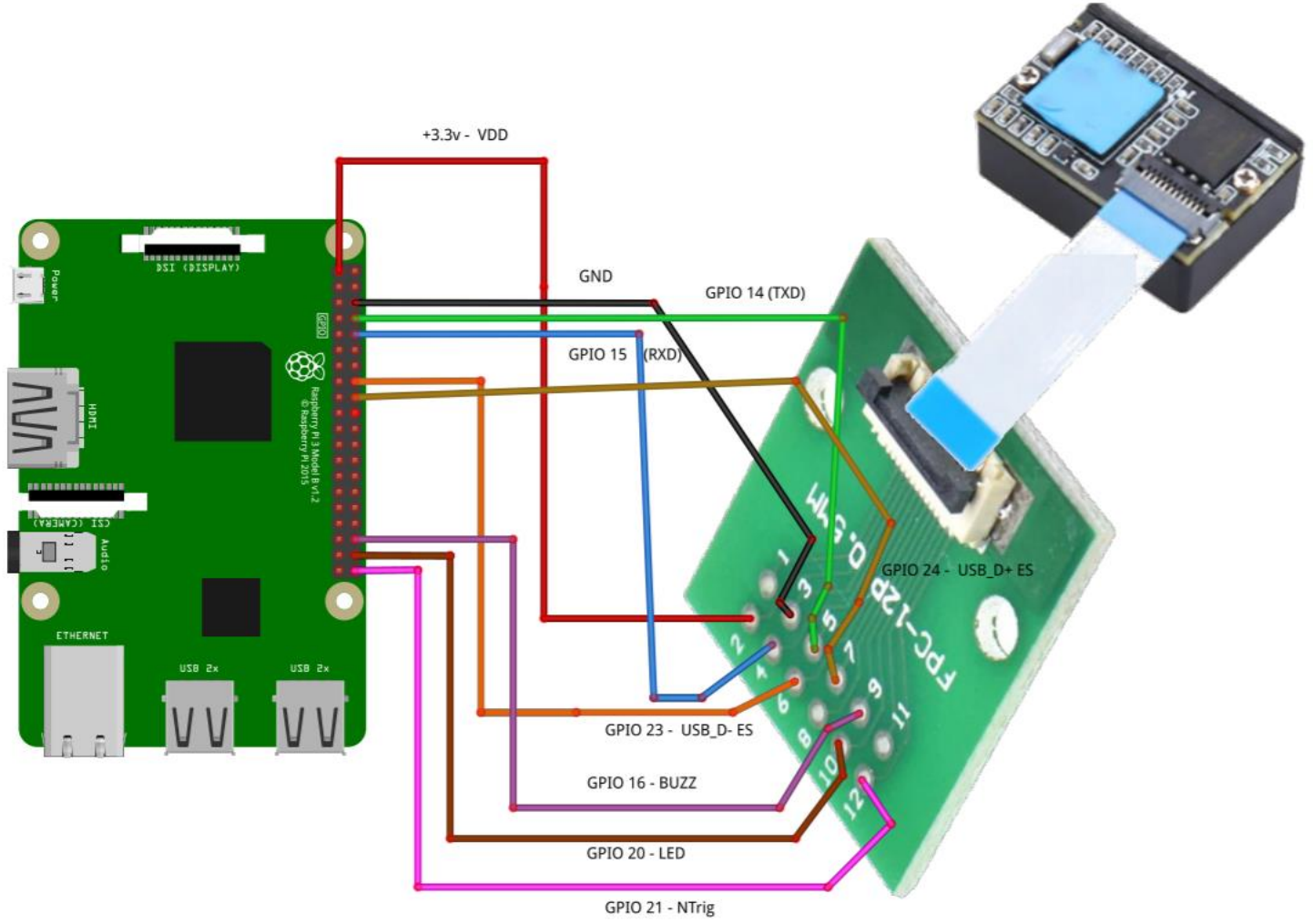
A continuación en la definición de pines del lector QR nos marca los 12 que tiene y cuales debemos conectar a cada lado:

- Tensión a tensión 3.3V
- RX a TX
- TX a RX
- Tierra a Tierra
- El resto como queramos

Definición de pines

PIN #	Señal de nombre	/O	Definición
1	NC	NC	-
2	VDD	-	3,3 V de entrada de alimentación
3	GND	-	GND
4	RX	De entrada	TTL-232 de entrada
5	TX	De salida	TTL-232 de salida
6	USB_D-	De entrada/salida	USB_D-signal
7	USB_D +	De entrada/salida	USB_D + señal
8	NG	-	Null
9	Buzz	De salida	
10	LED	De salida	
11	NG	-	
12	NTrig	De entrada	

Se sueldan cables dupont a la adaptador SMT de 12 pins para luego conectarlos a la Raspberri Pi



Una vez soldados los cables y conectado el lector QR a la Raspberry Pi se prueban dos librerías para ver con cual conseguimos codificar la lectura de los QRs:

La Raspi-serial que compila y no da error pero no vemos ninguna respuesta del lector de QR:
https://www.npmjs.com/package/raspi-serial

Luego probamos la SerialPort:
https://serialport.io/

La instalamos con npm en el proyecto y creamos un archivo.js para pruebas que contiene los siguientes puntos (ES SEUDOCÓDIGO):

- Require serialport
- Require parser-readline
- New SerialPort(puerto,{opciones}); → el puerto que usamos es: /dev/serial0 (raspbian define los puertos en /dev)
- New Readline({delimitador})
- Puerto.pipe(parser)
- Puerto.on('readable', function)

Con esto conseguimos que al apagar y encender se lea pero no de contínuo, a pesar de desactivar el log en sudo raspi.config.

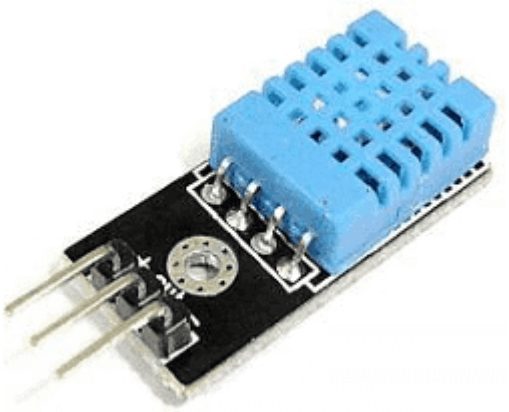
Tras darnos cuenta que el lector QR no era capaz de leer de continúo decidimos probar si es que había algún fallo de software o de hardware. Tras comparar nuestro código con el que encontramos en diversos recursos por internet y gracias a la ayuda de TxTheNoob en el Slack de Programar es una Mierda compramos una fuente de alimentación externa por usb.

Pero esto tampoco nos sirvió, aunque tras estar enchufando y desenchufando varias veces el lector QR nos dimos cuenta de que la conexión entre el cable flexible y la placa intermedia no funciona correctamente. No vimos forma de solucionar este punto y decidimos darnos por vencidos para poder prosperar en otras etapas.

En este punto nos replanteamos el proyecto y decidimos añadirle nueva funcionalidad.

Paso 5. Sensores

SENSOR TEMPERATURA Y HUMEDAD



El sensor de temperatura y humedad es un DHT-11. Tras buscar en npm encontramos una librería que recomiendan muchas personas y es la node-dht-sensor:
https://www.npmjs.com/package/node-dht-sensor

Tal y como indica en su página web es muy simple de usar. Se instala la librería

$ npm install node-dht-sensor

Después se conecta el sensor a nuestra Raspberry Pi por los pines que elijamos y la codificación para registrar los valores es muy sencilla, tal como indican en GitHub:

```
var sensor = require('node-dht-sensor');

sensor.read(22, 4, function(err, temperature, humidity) {
    if (!err) {
        console.log('temp: ' + temperature.toFixed(1) + '°C, ' +
            'humidity: ' + humidity.toFixed(1) + '%'
        );
    }
});
```

SENSOR DE MOVIMIENTO → PRUEBA FALLIDA

Con el sensor de movimiento no tenemos tanta suerte. Tras ver ejemplos de su uso en Arduino y Node nos decidimos a probar varias librerias, pero ninguna nos llega a funcionar.

El sensor de movimiento es el HCSR04

La librería que mejor resultado nos dio fue la mmm-usonic:
https://www.npmjs.com/package/mmm-usonic

No obstante siguiendo los pasos se produce un error que nos dice por consola que hay un hardware desconocido: CM2835. La solución la encontramos en un foro y es simplemente dentro de node_modules > mmm-usonic >lib >usonic.js en la línea 29 añadir ese hardware al if:

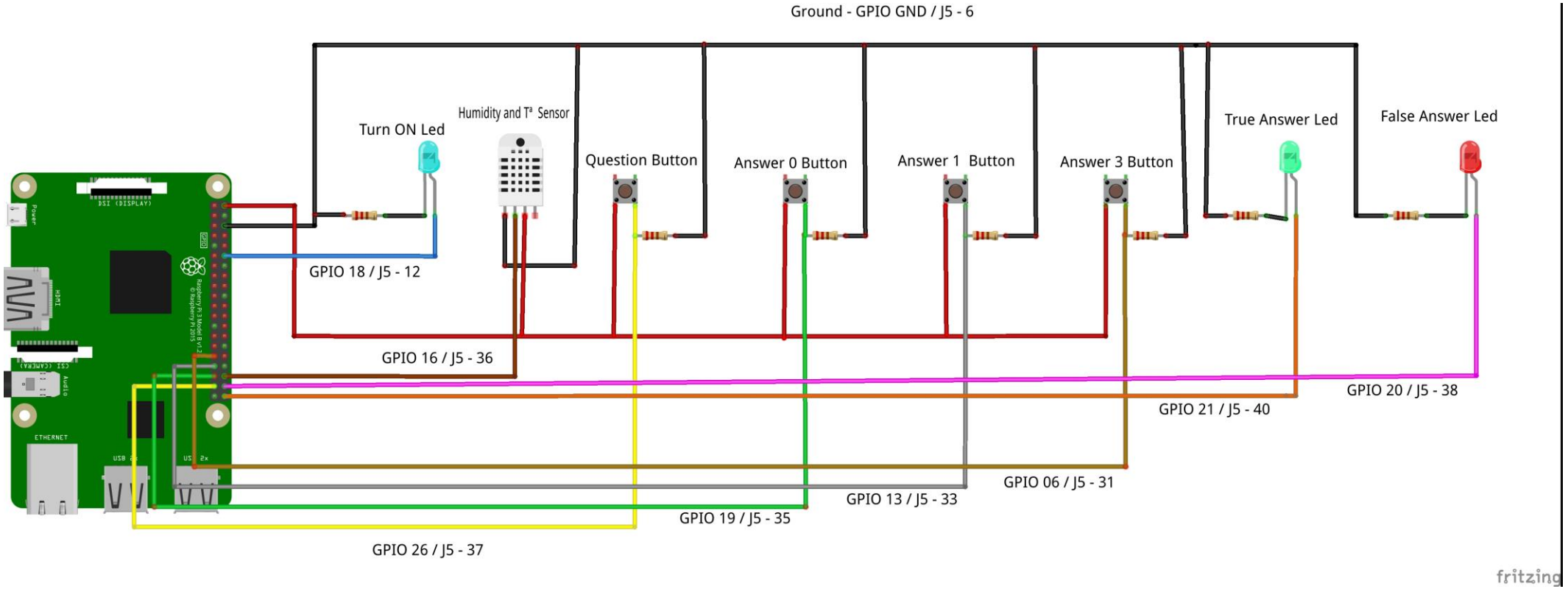
If(hardware === 'BCM2708' || hardware==='BCM2835')

Una vez solucionado no nos da error pero la medición que nos registra es siempre la misma: -1.

Probamos con Jhonny Five pero para la Raspberry Pi y este sensor en concreto no nos acaba de funcionar bien.

Así que finalmente así queda nuestro circuito, con un led que marca si está encendido, un led de respuesta acertado, uno de respuesta errónea. Un botón de siguiente pregunta y los botones de respuesta A, B o C.

Además el sensor de temperatura y humedad, que usaremos luego para la aplicación Android.



SENSOR DE PARTÍCULAS EN SUSPENSIÓN

KKmoon PM2.5 Sensor de Salida Digital de Polvo de Detección de Calidad del Aire de Alta Precisión SDS011

Encontramos una librería en npm específica para este sensor la sds011-client. Seguimos los pasos que definen en su Github pero no funciona:
https://www.npmjs.com/package/sds011-client

Tras analizar el error vemos que es porque no estamos indicando bien el puerto usb. Una vez indicamos el correcto, tras primero listar los disponibles con lsusb en la terminal y luego ya encontrar por internet que el que corresponde es “dev/ttyUSB0” vemos que el error viene porque si hay mas de dos usbs conectados de tipo datos a la Raspberry Pi no sabe diferenciarlos y nosotros teníamos una llave usb conectada.

El código queda como sigue:

Código para que lea una sola vez:

```
const SDS011Client = require("sds011-client");

const sensor = new SDS011Client("/dev/ttyUSB0");
Promise.all([sensor.setReportingMode('active'), sensor.setWorkingPeriod(10)])
.then((value) => {

console.log('' + JSON.stringify(value));
  console.log('then');
})
.catch(function(err) {
  console.log(err.message); // some coding error in handling happened
});

sensor.on('reading', r => {
console.log('' + JSON.stringify(r));
    if (r.pm2p5 > 10) {
        console.log('mayor 10');
    } else {
        console.log('menor 10');
    }
});
```

Código para que lea de contínuo:

```
const SDS011Client = require("sds011-client");
const sensor = new SDS011Client("/dev/ttyUSB0");

sensor
    .setReportingMode('query')
    .then(() => {
        console.log("Sensor is now working in query mode.");
        return sensor.setWorkingPeriod(0);
    })
    .then(() => {
        setInterval(() => {
            console.log("Querying...");
            sensor
                .query()
                .then((data) => {
                    console.log(`Received: ` + JSON.stringify(data));
                });

        }, 1000);

    });
```

Los valores adecuados para las PM 2.5 y PM10 son los siguientes:

ICA de la US-EPA para $PM_{2.5}$

Categoría ICA	Valor ICA	Concentración $PM_{2.5}$ Promedio en 24 Horas ($\mu g/m^3$)	
		Actual	Propuesta
Buena	0 - 50	0 - 15.4	0 - 15.4
Moderada	51 - 100	15.5 - 40.4	15.5 - 35.4
Dañina para Personas Sensibles	101 - 150	40.5 - 65.4	35.5 - 55.4
Dañina	151 - 200	65.5 - 150.4	55.5 - 150.4
Muy Dañina	201 - 300	150.5 - 250.4	150.5 - 250.4
Peligrosa	301 - 500	250.5 - 500.4	250.5 - 500.4

ICA de la US-EPA para PM_{10}

Categoría ICA	Valor ICA	Concentración PM_{10} Promedio en 24 Horas ($\mu g/m^3$)
Buena	0 - 50	0 – 54
Moderada	51 - 100	55 – 154
Dañina para Personas Sensibles	101 - 150	155 – 254
Dañina	151 - 200	255 – 354
Muy Dañina	201 - 300	355 – 424
Peligrosa	301 - 500	425 – 604

Los valores moderados ya deberían tenerse en cuenta para personas inmunodeprimidas

Paso 6. Lógica del juego

El siguiente paso ha sido básicamente codificar la lógica del juego del InmunoGame (cuando acabemos el proyecto subiremos el código a este repo):

- Encender el led que indica que está encendido mientras esté funcionando
- Tras apretar el botón de siguiente pregunta, seleccionar al azar una pregunta del data.json
- Leer la pregunta por los altavoces gracias al text to speech
- Una vez seleccionado uno de los botones de respuesta A, B o C comprobar si la respuesta es válida
- Encender el led de respuesta correcta (verde) o respuesta incorrecta(rojo)
- Leer la solución con algo mas de información con el text to speech

Paso 7. Servir una app Angular desde Raspberry Pi

Como el inmunoGame también se compone de una aplicación Angular que pueda ser consultada desde el móvil, comenzamos con una prueba de concepto.

La idea es tener un Hola Mundo en Angular que se pueda servir desde la Raspberry Pi gracias a Express (Node).

Los pasos que seguimos son:

1. - Install angular cli:
npm install -g @angular/cli

2.- Crear angular appV:
ng new my-app

3.- Instalar express en angular app:
npm install express --save

4.- Crear un fichero appServer.js en la carpeta raiz del proyecto con este contenido:

```
// Requires
const express = require('express');
const app = express();
const path = require('path');
const chalk = require('chalk');


// Static Routes
app.use(express.static(path.join(__dirname, 'dist')));



// Main App Route
app.get('/', (req, res, next) => res.sendFile(path.join(__dirname, 'dist', 'index.html')));
const port = 1337;


// Run Server
app.listen(process.env.PORT || port, () => console.log(chalk.blue(`Listening intently on port ${port}`)));
```

5.- editar fichero raspberry pi

```
sudo nano /etc/rc.local
```

6.- añadir la linea:

```
su pi -c 'node /home/pi/NAME_OF_DIRECTORY/NAMESERVER.js < /dev/null &'
```

reemplazando NAME_OF_DIRECTORY por nuestro directorio y NAMESERVER por el nombre del fichero server

7.- Configurar servir web externamente

```
sudo ip addr show
```

Fijarse punto 3, wlan0: (punto 2 para usar lan)

```
inet 192.168.1.136/24
```

probar con http://INTERNAL_IP_ADDRESS:PORT_SELECTED

ex: 192.168.1.136:1337

Paso 8. Probarlo todo junto, lanzando como hilos de forma automática

Para que el InmunoGame se ejecute solo, en el archivo de la Raspberry Pi /etc/rc.local hay que añadir los archivos o hilos que deben ejecutarse, tal como hicimos para el servidor Angular, ya que funcionan como demonios.
Se añaden las siguientes líneas

su pi –c ‘node /home/pi/Desktop/AngularServer/server-app/appServer.js < /devnull&’

forever start –c ‘node --**max**-**old**-**space**-**size**=2048’ /home/pi/Desktop/InmunoGame/inmunoGame.js /dev/null > /home/pi/Desktop/inmunoGame_log.txt &

La primera línea es la que inicializa el servidor Angular.
En la segunda línea hemos añadido el forever, para asegurarnos que esté ejecutando de continuo:
https://www.npmjs.com/package/forever
En la segunda línea también hemos añadido una orden para que nos escriba en un fichero de log los errores o warnings que se produzcan.

Nos encontramos con un problema que es que se ejecuta una sola vez, al llegar al final del archivo se produce un error y no da tiempo ni que a que picospeaker lea la explicación. Tras muchas pruebas, vemos que es porque en el método de comprobar si la respuesta está correcta estamos escribiendo en un archivo con una ruta local. Cuando lo ejecutábamos desde la terminal funcionaba porque estábamos ejecutando la terminal dentro de la carpeta con el archivo, pero al ejecutarlo desde este archivo no reconoce esa ruta. Le ponemos la ruta absoluta y ya funciona.

Paso 9. Crear un node server

INTENTO FALLIDO

Antes de lanzarnos a realizar el server de Node tratamos de hacer la lectura de los sensores desde una aplicación Angular y vemos que da error y no se puede por culpa de que las librerías de los sensores incluidas en los node_modules están en Javascript y tienen varias importaciones con require y TypeScript no las entiende.
Probamos a instalar @types/node con:

npm install - - save @types/node

Y luego en el archivo src/tsconfig.app.json añadimos lo siguiente:

“types”: [“node”],
“typeRoots”: [“../node_modules/@types]

Pero nos sigue fallando en la build, así que decidimos hacer una REST API en Node con Express que además nos permitirá tener una arquitectura mas limpia.

PASOS SEGUIDOS

Creamos una REST API sencilla con dos get(los sensores):

https://www.freecodecamp.org/news/building-a-simple-node-js-api-in-under-30-minutes-a07ea9e390d2/

El siguiente paso es que implementamos el login usando jwt y seguimos los pasos de este tutorial:

https://www.oscarblancarteblog.com/2018/01/16/implementar-json-web-tokens-nodejs/

Importante no olvidarse del body-parser, como nos pasó a nosotros. Y el jwt.verify a nosotros nos da falsos positivos. Así que nos leemos la documentación oficial de jwt para node y la comprobación del token la realizamos así:

```
try{
return jwt.verify(token, secretToken);
} catch(err){
return false;
}
```

Una vez lo tenemos funcionando entonces creamos nuestros tests con Mocha y supertest, como en el siguiente tutorial:

https://codeforgeek.com/unit-testing-nodejs-application-using-mocha/

Nos conectamos con Postman y vemos que funciona bien.

Paso 10. Crear una base de datos para guardar mediciones de sensores

Decidimos que además de mostrar en la aplicación Angular los sensores, vamos a mostrar un histórico de los riesgos para lo cual cada 4 horas vamos a guardar un registro en base de datos.
Decidimos usar MariaDB y lo hacemos todo a través de terminal.

1.- Instalar MySql
```
sudo apt-get install mysql-server -y
```

2.- Verificar instalacion
```
sudo /etc/init.d/mysql status
```

version instalada: MariaDB 10.1.38sudo

3.- Ingresar en la base de datos como root
```
sudo mysql -u root -p -h localhost
```

4.- Crear base de datos InmunoGameDB
```
CREATE DATABASE InmunoGameDB;
```

5.- Utilizar la base de datos
```
USE InmunoGameDB;
```

6.- Creamos un usuario con password para ser utilizado desde appServer

```
CREATE USER 'inmuno_user'@'localhost' IDENTIFIED BY ‘mypass';
```

7.- Asignamos permisos al usuario

```
GRANT ALL PRIVILEGES ON InmunoGameDB.* TO 'inmuno_user'@'localhost';

FLUSH PRIVILEGES;
```

8.- Reiniciar MySql

```
sudo service mysql restart (desde terminarl raspbian)
```

9.- Crear una tabla

```
CREATE TABLE esp32_dht11 (
dateandtime DATETIME,
sensor VARCHAR(32),
temperature DOUBLE,
humidity DOUBLE);

CREATE TABLE IF NOT EXISTS sensorsData (sensorsdata_id INT AUTO_INCREMENT,getData_date DATE,temperature
decimal,humidity decimal,particle25 decimal,particle100 decimal,PRIMARY KEY (sensorsdata_id));
```

10.- salir de MariaDB terminal

```
ctrl + c
```

Paso 11. Creamos una aplicación con Angular

Creamos una aplicación Angular 8.

Creamos modelos, services y componentes tanto para el Login, como para la home y para el histórico. La aplicación llamar al server realizado en Node para conseguir los datos.

Peculiaridades:
El Login se hace con JWT. Hay muchos manuales como por ejemplo:
https://code.tutsplus.com/es/tutorials/jwt-authentication-in-angular--cms-32006

Los tests los hacemos con Karma, nos gustan estos dos manuales:
https://medium.com/@jorgeucano/introducci%C3%B3n-al-testing-en-angular-da415ef8c47
https://angular.io/guide/testing

Los gráficos que muestran el histórico de las alertas los realizamos con Chart.js y nos sirve de punto de partida este artículo.
https://medium.com/innoventes/adding-a-graph-to-your-angular-app-eede3dd8bf1c
https://www.chartjs.org/

AGRADECIMIENTOS

La idea inicial nos viene gracias a Ruben **@agileando** que me enseñó la publicación de **@ChrisJPatty**, en la que mostraba una caja de música que había realizado con una Raspberry Pi y un lector de código de barras.
https://twitter.com/ChrisJPatty/status/1078065910487760896
En ello nos basamos para realizar el InmunoGame, la parte del juego.

La parte de los sensores viene gracias a los organizadores de PEUM, Alex **(@sinmsinm**) y Juanjo**(@jjmerono**) que me preguntaron si no podría hacer una aplicación que te avisase de que te ibas a poner enfermo. Tras muchas vueltas ahí están nuestros sensores de Temperatura, Humedad y partículas con nuestros algoritmos. Me gustaría agradecerles además el curro de crear esta comunidad con sus hackáthones y sus eventos que consiguen sacarme una sonrisa.

Durante el desarrollo del proyecto quisimos implementar un lector de QR pero no fuimos capaces con JavaScript of Things, no obstante, nos ayudaron muchísimo en el canal **PEUM IOT,** por ejemplo, **Tx**. También **@meri_minimeri** que ha sido una fuente de apoyo y ánimos contínuos.

Muchas gracias en general a los y las profesionales que escriben en **Medium,** hemos consultado a lo largo del proyecto varios artículos y todos han sabido desatascarnos o sernos fuente de inspiración.

Además estamos encantados de poder contar este otoño como ha sido este proceso tanto en **Codemotion Madrid, ng Spain, Barcelona Software Crafters y JsDayCan2019**. ¡Gracias por querer escucharnos!

Y SOBRE TODO A LA ALEGRÍA DE NUESTROS ÚLTIMOS 12 AÑOS, PETER, QUE AUNQUE NOS HA DEJADO ESTE AGOSTO SIEMPRE ESTARÁ EN NUESTROS CORAZONES