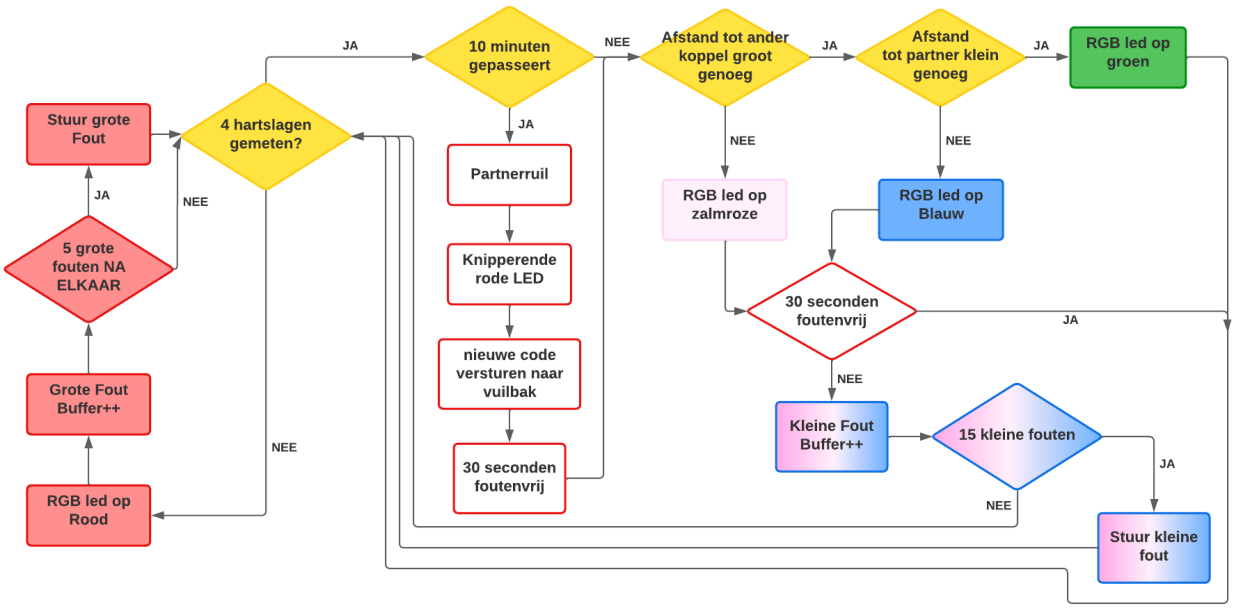
Software

> Visit our Github to find our code!

Overzicht:

Algemeen

We presenteren eerst een algemene flowchart van de volledige puzzel. Deze vindt u hieronder.



Flowchart in het groot

De code bestaat uit 3 grote delen. Als eerste hebben we onze variabelen, daarnaast hebben we al onze functie en dan ook nog de setup en de loop. We zullen in deze paragraaf proberen uit te leggen hoe onze code is opgebouwd.

Libaries en includes

In onze code gebruiken we 3 libaries. We gebruiken enerzijds eentje om io-pinnen van onze esp aan te sturen. Daarnaast gebruiken we ook een libary voor het manipuleren van de de hartslagsensor, en een laatste voor de MQTT connectie.

```
lib_deps =
    erropix/ESP32 AnalogWrite@^0.2
    sparkfun/SparkFun MAX3010x Pulse and Proximity Sensor Library@^1.1.1
    knolleary/PubSubClient@^2.8
```

```
#include "Arduino.h"
#include <string.h>
#include <Wire.h>
#include "MAX30105.h"
#include "analogWrite.h"
#include "heartRate.h"
#include <BLEDevice.h>
#include <BLECast.h>
#include "PubSubClient.h"
#include "WiFi.h"
```

Variabelen

Unieke variabelen

Het eerste deel van de code bevat 2 belangerijke variabelen. namelijk de naam van de esp en zijn ID. Die laatse wordt tijdens het verloop van het spel veranderd omdat deze wordt gebruikt voor de partnerruil.

Elke wristband heeft zijn eigen espNaam en id. Het vreemd is dat ze als volgt worden gegeven:

```
esp1 : ID = 3
esp2 : ID = 4
esp3 : ID = 1
esp4 : ID = 2
```

Dit wordt zo gedaan zodat de eerste code die wordt verstuurd naar de vuilbakken niet "1234" is, maar "3412". Dit wordt zo gedaan zodat het spel niet te makkelijk start.

```
    int id = 3;
    std::string espNaam = "esp1";
```

OPGELET: Wanneer aan de esp's andere id's worden gegeven moet de variabele "doorstuurCode" ook aangepast worden.

Aanpasbare variabelen

Vervolgens hebben we een deel variabelen die we kunnen aanpassen naargelang we bepaalde parameters in het spel willen veranderen.

```
int codetime =5000;
int codetimelong =60000;
const unsigned long switchPeriod = 600000;
int grenswaardeTeVer  = 60;
int grenswaardeTeDicht = 40;
int irThreshold = 60000;
int partnerTime = 30;
int hartslagFoutenMax = 5;
const unsigned long period = 1000;
int foutenMarge =10;
```

- **codetime**: Tijdens het spel wordt een code duidelijk gemaakt door knipperlichtjes. Als de code 3412 is, dan gaat esp3 eerst een ledje laten knipperen, vervolgens knipperen respectievelijk esp4, esp1, esp2 ook een ledje. De tijd die hiervoor tussen het knipperen van twee opeenvolgende esp's is hier voorgesteld in milliseconden.
- **codetimelong**: De code die wordt beschreven in vorig puntje wordt in dit geval elke minuut opnieuw getoont.
- **switchPeriod**: Tijdens de duur van de escaperoom wordt er ook gedaan aan partnerruil. Deze variabele heeft aan na hoeveel milliseconden dit gebeurd. In dit geval duurt dit 10 minuten.
- **patnerTime**: In het begin van een nieuwe ronde van 10 minuten krijgen de spelers een aantal seconden tijd om hun partner te vinden. Tijdens deze periode worden geen fouten gestuurd naar de buffer.
- **grenswaardeTeVer** en **grenswaardeTeDicht**: Dit is een grenswaarde om te beslissen wanneer twee esp te dicht of te ver bij elkaar zijn. Deze waarde stelt de sterkte van het ontvangen BLE-signaal voor.
- **hartslagFoutenMax**: Wanneer er geen hartslag word gedetecteerd wordt er niet direct een foutmelding gestuurd naar de buffer. De fout moet zich namelijk 5 keer na elkaar voordoen vooralleer deze wordt gestuurd. Dit doen we om te voorkomen dat er bij een foute meting van de hartslagsensor direct een grote fout zou worden gestuurd.
- **period**: elke esp stuurt continu een signaal. Na een bepaalde periode van, in dit geval, één seconde zal de esp zijn omgeving scannen.
- **foutenMarge**: Er wordt niet telkens wanneer iemand te ver of te dicht is een kleine fout gestuurd. Dit wordt pas gedaan wanneer er in het spel telkens 5 fouten zijn gemaakt.

Vaste variabelen

Verder hebben we heel wat variabelen die nodig zijn voor het spel te kunnen programmeren, maar ook voor het besturen van de componenten en de communicatie.

Random variabelen

Dit onderdeel bevat verschillende die we gebruiken in de code.

```
    //VARIABLES RANDOM
============================================================
    const char *cEspNaam = espNaam.c_str();
    BLECast bleCast(espNaam);
    int codePeriod = 10000000;
    std::string doorstuurCode ="Wristband-code 3412";

    int foutCounter = 0;
    int ronde = 0;

    int codetimelongcounter =0;
```

Zo zorgen de eerste twee variabelen voor de naam van de esp die we later gebruiken om een unieke naam te hebben bij de MQTT-communicatie.

De **codePeriod** wordt telkens aangepast in de loop. Dit wordt gelijk gesteld aan **codetime + id*codetime;**. Dit gebruiken we om de delay voor het knipperen van de leds te realiseren.

De **doorstuurCode** is de code die wordt doorgestuurd naar de vuilnisbakken. Deze wordt aangepast wanneer er aan partnerruil wordt gedaan en een nieuwe code wordt gecreerd.

Verder hebben cwe nog een variabele die het aantal fouten bijhoud, en een variabele die bijhoudt in welke ronde van het spel we zitten. Met rondes bedoelen we periodes waarin de partners dezelfde blijven.

Als laatste hebben we de variabele **codetimelongcounter**. Deze heeft misschien een iets wat verwarrende naam, maar deze is bedoelt om bij te houden hoeveel keer dat de code tijdens een ronde is weergegeven op de esp's.

variabelen en constanten voor de WIFI

Volgende variabelen zijn uiterst voor de WIFI en de MQTT communicatie. Zo definiëren we een paswoord en wifi-id, maar ook het ip-adres en de poort. Verder hebben we variabelen die ons helpen met het manipuleren van messages.

```
    //VARIABLES AND CONSTANTS WIFI
============================================================
#define SSID          "NETGEAR68"
```

```
#define PWD           "excitedtuba713"

#define MQTT_SERVER   "192.168.1.2"
#define MQTT_PORT     1883

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

variabelen en constanten voor de hartslagsensor

volgende variabelen worden gebruikt tijdens de connectie met de hartslagsensor. De hartslagsensorEnable zorgt er voor dat we de hartslagsensor vanop afstand kunnen uitschakelen.

```
//VARIABLES  AND CONSTANTS HARTSLAGSENSOR
===========================================
MAX30105 particleSensor;

const byte RATE_SIZE = 4; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; //Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

bool hartratesensorEnable;
int hartslagFoutenTeller;
```

Constanten voor de LED Via deze constanten kunnen we makkelijk de IO-pinnen van de ESP bepalen die onze LED aansturen

```
//Constants RGB LED
============================================================
const int PIN_RED   = 16;
const int PIN_BLUE = 4;
const int PIN_GREEN  = 2;
```

Variabelen en constanten voor BLE

Volgende variabelen en constanten zijn nodig voor het bepalen en bijhouden van de afstand met de andere ESP's

Eenderzijds gebruiken we de d-variabelen om de afstand tot een specifieke esp te weten. Deze waarden wordt dan toegekent aan de juiste distance-variabelen. Deze zijn belangrijk om makkelijk de partnerruil door te voeren via de functie "fixDistance".

```
//VARIABLES AND CONSTANTS BLE
============================================================
BLEScan *pBLEScan;

    int d1;  //afstand tot ESP1
    int d2;  //afstand tot ESP2
    int d3;  //afstand tot ESP3
    int d4;  //afstand tot ESP4

    int distance1;  //afstand tot id1
    int distance2;  //afstand tot id2
    int distance3;  //afstand tot id3
    int distance4;  //afstand tot id4



    const int scanTimeSeconds = 1;
    uint8_t cnt = 0;
    char data[5];
    uint8_t mode;
```

Variabelen voor timing

De laatste variabelen houden bepaalde tijd bij. Zo moeten we een tijd bijhouden die bepaald wanneer er een bluetooth signaal wordt gescand of verstuurd. Daarnaast houden we ook een tijd bij voor de partnerruil en voor het tonen van de code.

```
//VARIABLES TIMING
============================================================

    unsigned long startMillis;
    unsigned long currentMillis;

    unsigned long switchStartMillis;
    unsigned long switchCurrentMillis;

    unsigned long codeStartMillis;
    unsigned long codeCurrentMillis;
```

Functies

We kunnen de functies groeperen in verschillende groepjes.

LED

Om te beginnen hebben we voor verschillende functie geschreven om het led in een verschillende kleur te laten branden. Een voorbeeld ziet u hieronder.

```
void setCyan(){
 analogWrite(PIN_RED,   0,255);
 analogWrite(PIN_GREEN, 255,255);
 analogWrite(PIN_BLUE,  50,255);
}
```

Verder hebben we gewoon een **setupLed()** geschreven.

Bluetoothsignaal

Zo hebben we een **scanSetup()** en een **sendSetup()**, maar ook een **scanLoop()** en **sendLoop()**. Met de klasse **AdvertisedDeviceCallbacks()** kunnen we de afstanden met de andere ESP's bepalen.

Om die afstanden verder te gebruiken, maken we gebruik van functies zoals **checkTeVer()** en **checkTeDicht()**. Deze functies geven een boolean terug die zegt of een esp te dicht of te ver van een andere is. Onderstaande code is een voorbeeld om te kijken of 2 esp's te ver van elkaar zijn.

```
if(id==1 ||id ==2){
    if(distance1 < (-grenswaardeTeVer) || (distance2 < (-grenswaardeTeVer) ))
{

    return true;
    }
```

Wanneer dit het geval is worden respectievelijk de functies **teVer()** en **teDicht()**, die zowel het ledje van kleur laten veranderen als de juiste foutmelding sturen naar de buffer. Om deze foutmeldingen wat te minimaliseren, houden we de variabel foutCounter bij. In ons geval moeten er dus 15 fouten gebeuren vooralleer effectief een fout wordt verzonden. Dit is natuurlijk rekeninghoudende met het feit dat een esp altijd met minstens 2 in de fout zijn.

HartrateSensor

Aangezien we alleen maar de infraroodwaarde gebruiken van de hartslagsensor om te kijken of de speler de wristband goed aanheeft, hebben we enkel een functie nodig die de sensor initialiseerd. Dit

doen we met de **initializeSensor()**-functie. Deze functie is belangerijk om te weten of de hartslagsensor werkt. Wanner dit niet het geval is wordt er een error geprint in de serial monitor.

MQTT

Nog een belangrijk deel van de functies, zijn de functies van de wifi. Zo hebben we een **setup_wifi()** om te connecteren met de wifi. Daarnaast hebben we ook een reconnect en een callback. In de reconnect wordt er een MQTT connection aangegaan met de broker, maar er wordt ook beslist naar welke directories we gaan luisteren. Dit doen we aan de hand van een ingebouwde subscribe-functie van de PubSubClient libary.

```
client.subscribe("controlpanel/reset");
```

De callback gebruiken we eerder om met de ontvangen boodschappen verder te kunnen werken. Zo kunne we bijvoorbeeld de wristbands uitschakelen met een message die verzonden wordt via MQTT.

```
if (messageTemp== "Stop Wristbands"){
   wristbandEnable = false;
 }
```

Partner-rondes en code

Een laatste deel functies die we nog willen bespreken zijn diegene die betrekking hebben op de partnerruil en de code die moet verzonden worden naar de vuilnisbak. De code wordt duidelijk gemaakt aan de hand van een knipperend magenta-kleurig ledje. Elke wristband zal kort op elkaar dit ondergaan. Uit die volgorde kan de code worden gevonden dankzij de genummerde wristbands. De functie, **code()**, laat na een periode van "codetimelong" milliseconde de wristbands flikkeren volgens oplopende id. Dit doet hij gedurende heel de rond een aantal keer.

In het begin van elke ronde wordt de juiste code verzonden naar de vuilnisbak via de functie **sentCode()**. Dit veranderd telkens omdat de id's elke partnerronde worden aangepast. Dit doen we via de functie: **schuifDoor()**. Deze functie telt gewoon 1 op bij de huidige ID tenzij het id 4 was, dan wordt deze terug op 1 gezet.

Dankzij voorgaande techniek krijgen we de mogelijkheid om aan partnerruil te doen via de functie fixDistance. Deze functie zorgt ervoor dat de juiste d-variabelen worden gelinkt met de juiste distance-variabelen (zie variabelen voor BLE). Dit hebben we gedaan door een aantal if-lussen. Bovendien wordt hier ook de doorstuurCode aangepast. Een voorbeeld zie je hieronder.

```
if (espNaam.compare("esp1") == 0){
   if(id==3){
     distance3 = d1;
     distance4 = d2;
```

```
        distance1 = d3;
        distance2 = d4;
        doorstuurCode = "3412";
      }
```

Verder zijn er nog een aantal kleine functies die het coderen makkelijker maken zoals **resetESP()** of **noWristband()**.

Setup en Loop

Tijdens de setup worden eerst onze startMillis allemaal geinitialliseerd. Dit is zeer belangerijk aangezien de 4 wristbands niet gesynchroniseerd zijn met elkaar. Hierdoor moeten alle wristbands op hetzelfde moment gereset worden zodat ze met een gelijke klok kunnen lopen. Daarnaast wordt alles ook gesetupt via de bijhorende functies.

De loop zelf is relatief simpel geschreven. Dit komt omdat we alles in functies hebben geschreven.