Projet Personnel 1 : Station Météo

Sommaire :

Introduction

Etant actuellement en deuxième année de cycle préparatoire au CESI (spécialité informatique), je me suis lancer dans un projet personnel combinant plusieurs notions :

- Electronique ;
- Informatique.

Je me suis lancer dans projet, afin d'apprendre et de comprendre comment fonctionnent les systèmes embarqués.

Contexte du projet

Afin de connaître la température et le taux d'humidité dans ma maison, j'ai construit une petite station météorologique qui me permet pour le moment de connaître la température et le taux d'humidité de la pièce.

Liste du matériel utilisé

Afin de réaliser ce projet, j'ai utilisé le matériel suivant :

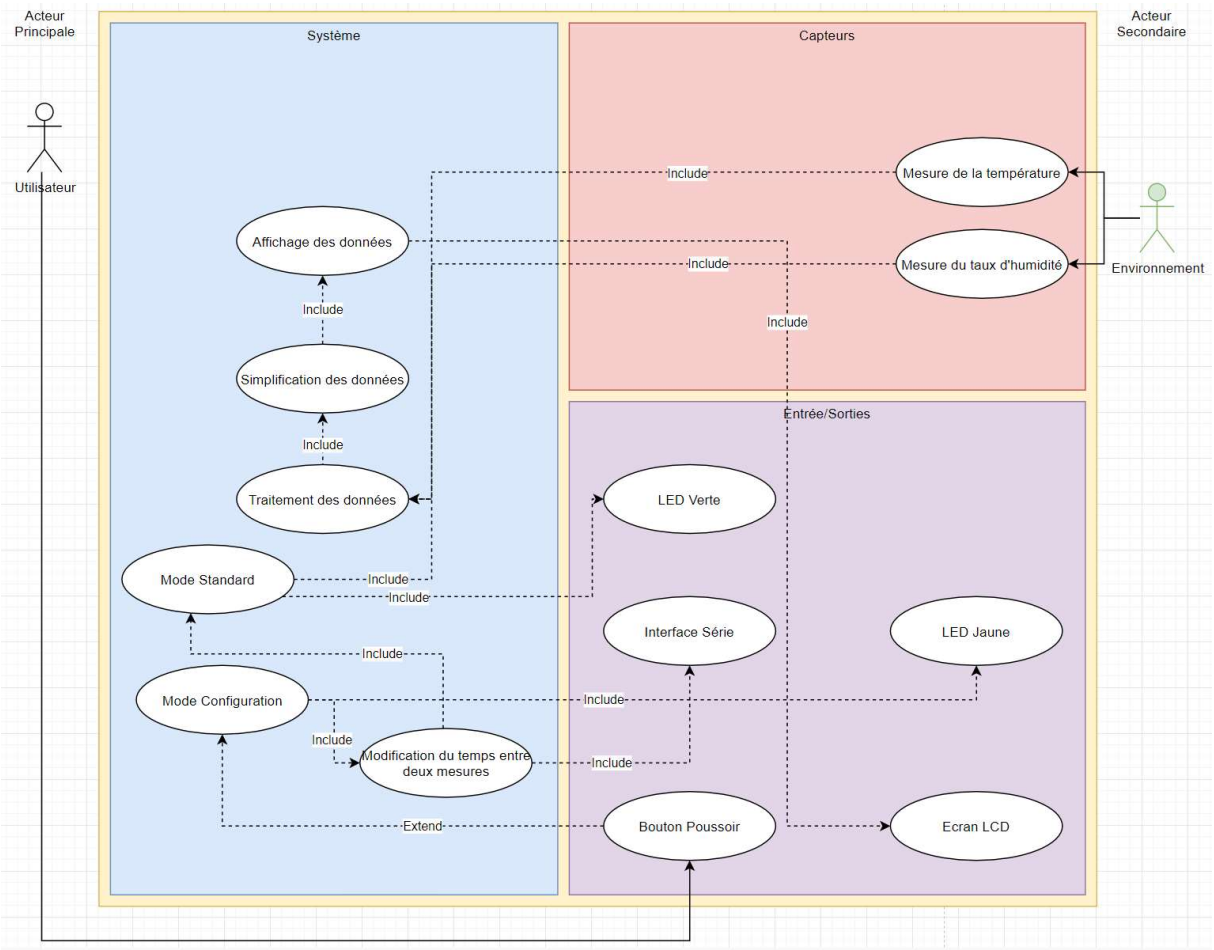
- Un Arduino Uno ;
- Deux LEDs (Jaune et Verte) ;
- Cinq résistances de 220 Ohms ;
- Un bouton poussoir ;
- Un potentiomètre de 10K Ohms ;
- Un capteur de température et d'humidité
- Un écran LCD 1602 Qapass ;
- Des fils.

Explication du projet

Nous allons commencer par voir, les bases du projet.

Diagramme UseCase

Dans un premier temps, j'ai essayer de chercher un mode de fonctionnement pour mon système. Dans un premier temps, j'ai donc identifier les besoins nécessaire à mon projet, à savoir :



Pour m'aider à comprendre le fonctionnement, j'ai fais un diagramme useCase qui me permet de mettre en avant les différents cas d'utilisation du dispositif.

Premièrement, l'acteur principale, sera la personne qui utilisera le dispositif. Les seules interactions que l'utilisateur aura avec le dispostif sont les suivantes :

- Le bouton poussoir, en effet ce bouton nous permet de passer du mode standard au mode configuration.
- L'interface série, cette interface est utilisée que dans le mode configuration, elle permet de faire de rentrer le nouveau temps d'intervalle.

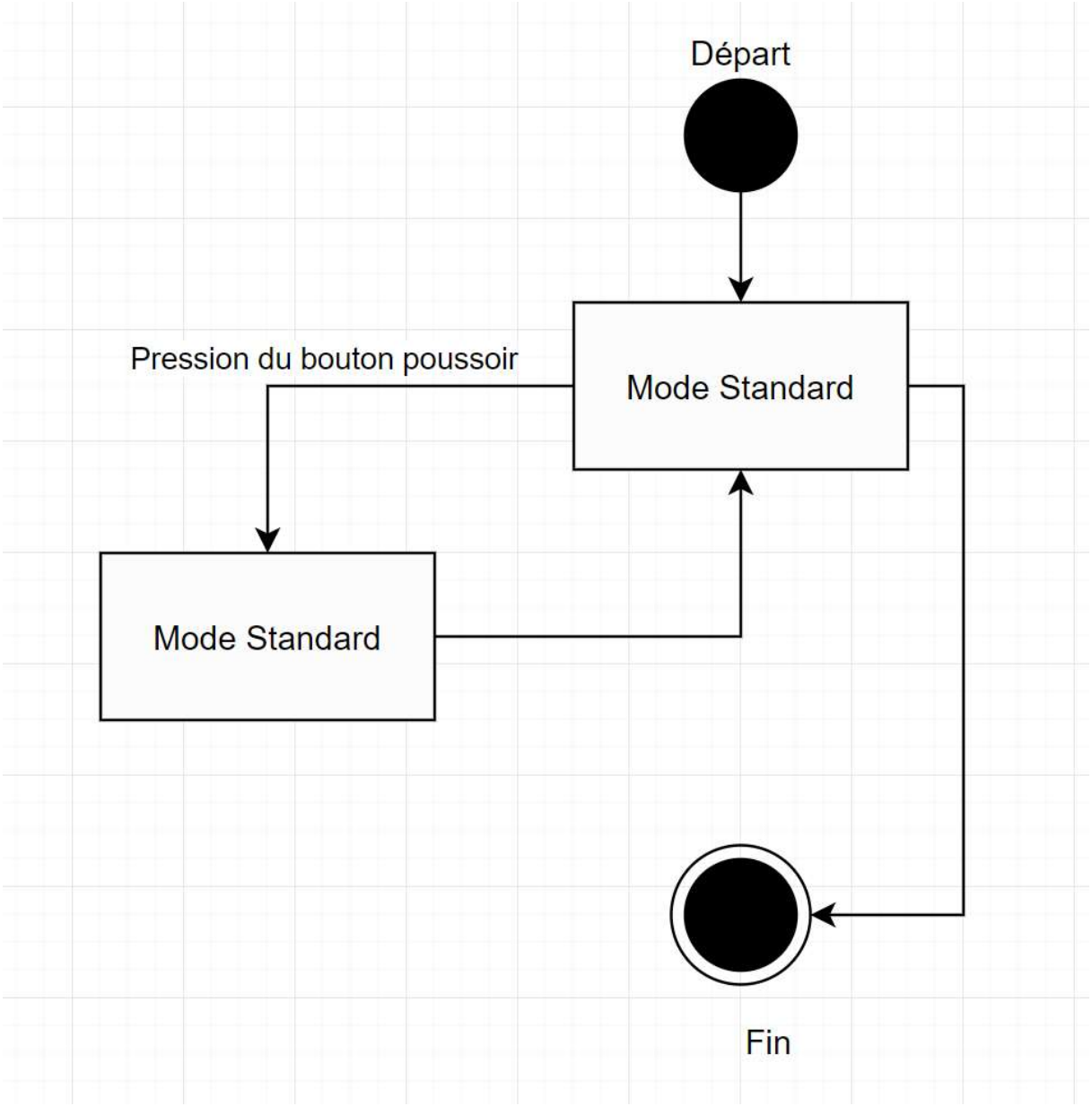
- L’écran LCD, sur cet écran se trouvent les données récoltés par le capteur et traitées par le système.
- Les LEDs qui nous permettent de connaitre le mdoe en cours d’utilisation. (La LED jaune pour le mode configuration et la LED verte pour le mode Standard)

L’acteur secondaire est l’environnement. En effet les mesures effectués par les capteurs se font sur l’environnement (dans notre cas dans une pièce de la maison).

Comme nous pouvons le constater, nous récupèrons les données brutes des capteurs, mais il y a bien une phase de traitement de ces données. Cela est dût au format des données renvoyées par le capteur et à l’utilisation des données dans le système.

Diagramme d’état

Nous allons maintenant parler rapidement du diagramme d’état. Ce diagramme nous permet de mettre en avant les état que va avoir notre système.



Comme nous pouvons le voir, le mode Standard s'active par défaut. Si le bouton poussoir est pressé par l'utilisateur, alors, le système passe en mode configuration. Une fois que les paramètres changeable dans le mode configuration ont été manipulés, le système repasse en mode Standard.

Code

Nous allons maintenant passer à la partie du code. Pour vous aider, j'ai mis des commentaire, mais je vais expliquer plus en détails le fonctionnement du code ici.

Dans un premier temps, nous allons voir la déclaration des variables globales et l'introduction des librairies.

```
#include <LiquidCrystal.h>
#include <DHT.h>

bool modeStandardCheck = true;

const static int portLEDVerte = 8;
const static int portLEDJaune = 10;
const static int portCapteurDHT = 7;
const static int portButton = 2;

int timeDelayLCD = 250;

int intervalleMesure = 1000;

String modeText;

LiquidCrystal lcd(12, 11, 6, 5, 4, 3);

#define DHTTYPE DHT11

DHT dht(portCapteurDHT, DHTTYPE);
```

Comme nous pouvons le voir, il y a deux librairies qui ont été importées.

- **LiquidCrystal.h** qui nous permet d'utiliser l'écran LCD.
- **DHT.h** qui nous permet d'utiliser le capteur de température et d'humidité.

Ensuite, nous allons parler des variables. Je les aies réparties par type. Parmi les variables booléennes :

- **modeStandardCheck** qui a true pour valeur par défaut. Cette variable permet de vérifier si nous sommes dans le mode standard ou non. Le choix d'une variable booléenne a été fait car nous n'avons que deux modes, dans le cas contraire, nous aurions utiliser une variable entière pour numéroter le mode.

Nous allons maintenant passer aux variables entières, parmi ces variables, il y en a qui sont constantes :

- **portLEDVerte** qui a pour valeur 8. C'est le port de la LED verte.
- **portLEDJaune** qui a pour valeur 10. C'est le port de la LED jaune.
- **portCapteurDHT** qui a pour valeur 7. C'est le port du capteur de température et d'humidité.
- **portButton** qui a pour valeur 2. C'est le port du bouton poussoir. Le port 2 a été choisi car nous pouvons y faire des interruptions.

Nous allons maintenant passer au reste des variables entières :

- **timeDelayLCD** qui a pour valeur 250. Cette valeur est un temps en millisecondes, elle correspond au delais de rafraichissement du LCD.
- **intervalleMesure** qui a pour valeur par défaut 1000. Cette valeur est un temps en millisecondes, elle correspond au delais entre deux mesures.

Nous avons aussi initier la variable modeText qui est une variable de type String. Elle nous permet d'afficher sur l'écran LCD le mode en cours.

Nous avons ensuite définits deux objets et une macroconstante. Les objets sont :

- **lcd** qui est du type LiquidCrystal. Il nous permet d'initialiser l'objet de notre LCD, dans lequel nous affectons tous les ports à utiliser.
- **dht** qui est du type DHT. Il nous permet d'initialiser l'objet de notre capteur de température et d'humidité. En paramètre nous y mettons le port que nous utilisons pour le capteur et la macroconstante.

La macroconstante :

- **DHTTYPE** qui a pour valeur DHT11 qui est le type de capteur de température et d'humidité que nous utilisons.

```
//*************************************************************
//* Nom :          ON                                         *
//* Paramètre :    Port                                       *
//* Utilité :      Allumer une LED                            *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void ON(int port)
{
    digitalWrite(port, 1);
}

//*************************************************************
//* Nom :          OFF                                        *
//* Paramètre :    Port                                       *
//* Utilité :      Eteindre une LED                           *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void OFF(int port)
{
    digitalWrite(port, 0);
}
```

Nous avons ensuite créer deux fonctions qui nous permettent d'allumer et d'éteindre une seule LED à la fois.

```
//*****************************************************************
//* Nom :          modeStandard                                   *
//* Paramètre :    Aucun                                          *
//* Utilité :      mode de fonctionnement normal                  *
//* Retour :       Aucun                                          *
//*                                                               *
//*****************************************************************

void modeStandard()
{
    OFF(portLEDJaune);
    ON(portLEDVerte);

    modeText = "SMODE";
    getData();
}
```

Nous pouvons voir ici la fonction modeStandard. Dans cette fonction, nous allumons la LED verte et éteignons la LED jaune. La variable modeText prend la valeur « SMODE » pour Standard Mode. Puis nous faisons appelle à la fonction getData() qui permet de récupérer les données.

```
//*****************************************************************
//* Nom :          modeConfiguration                              *
//* Paramètre :    Aucun                                          *
//* Utilité :      mode de configuration des paramètres           *
//* Retour :       Aucun                                          *
//*                                                               *
//*****************************************************************

void modeConfiguration()
{
    OFF(portLEDVerte);
    ON(portLEDJaune);

    modeText = "CONFIGURATION";

    affichageLCD(0,0,"USE SERIAL PORT");
    affichageLCD(0,1,modeText);

    Serial.println(F("Mode configuration !"));
    Serial.println(F("Changement du temps d'intervalle entre deux mesures : "));

    int newIntervalle;
    String reponse;

    while (Serial.available() == 0){}          //Tant qu'aucune réponse n'est rentrée on ne continue pas le programme
    reponse = Serial.readString();
    newIntervalle = reponse.toInt();

    Serial.print(F("Le temps que vous venez de mettre est "));
    Serial.print(newIntervalle);
    Serial.println(F(" secondes."));

    intervalleMesure = newIntervalle * 1000;

    Serial.print(F("Retour en mode Standard !"));

    delay(500);

    modeStandardCheck = true;
}
```

Voici la fonction du mode de configuration. Nous pouvons voir que dans un premier temps, nous allumons la LED jaune et éteignons la LED verte. La variable modeText prend la valeur « CONFIGURATION ». Nous affichons ensuite sur la première ligne de l'écran LCD un message disant à l'utilisateur, d'utiliser l'interface série. En dessous, nous mettons en avant le mode. Ensuite, nous

affichons du texte sur l'interface série, pour que l'utilisateur comprenne plus facilement ce qu'il doit faire lors de l'utilisation de ce mode. Dans un soucis de performance, nous utilisons la mémoire flash pour afficher nos messages. Nous utilisons ici, deux variables locales à la fonction :

- newIntervalle qui est du type entier, cette variable n'a pas de valeur par défaut.
- reponse qui est du type String, cette variable n'a pas non plus de valeur par défaut.

Nous passons ensuite dans une boucle while de telle façon à ce que le programme s'arrête tant que l'utilisateur n'a pas rentrer et envoyer à l'Arduino (par le biais de l'interface série) le nouveau paramètre à modifier. Ensuite, la donnée entrée est du format String grâce à la fonction Serial.readString(). Puis, nous retransformons la string en entier. Nous affichons un autre message pour informer l'utilisateur du temps qu'il vient de mettre afin qu'il soit certain du temps qu'il vient d'entrer. Nous convertissons ensuite le temps qui est en secondes en millisecondes. Puis nous affichons un message informant l'utilisateur que le dispositif va repasser en mode standard. Nous mettons un delay de 500 millisecondes. Enfin nous changeons la variable modeStandardCheck pour repasser en mode standard.

```
//*************************************************************************
//* Nom :          getData                                                *
//* Paramètre :    Aucun                                                  *
//* Utilité :      Obtenir des données                                    *
//* Retour :       Aucun                                                  *
//*                                                                       *
//*************************************************************************

void getData()
{
    String dataString1;
    String dataString2;

    dataString1 = "Temp: ";
    dataString1 += String((int)dht.readTemperature()) + " C ";
    dataString1 += modeText;

    dataString2 = "Humidity: ";
    dataString2 += String((int)dht.readHumidity()) + "%";

    affichageLCD(0,0,dataString1);
    affichageLCD(0,1,dataString2);

    delay(intervalleMesure);
}
```

Nous allons maintenant parler de la fonction getData. Nous avons créer deux variables de type String :

- dataString1 qui est la string que nous allons utiliser sur la première ligne de l'écranLCD.
- dataString2 qui est la string que nous allons utiliser sur la deuxième ligne de l'écranLCD.

Nous construisons ensuite la string de telle façon à ce qu'il y ait le type de mesure qui est faite, la mesure en elle-même et l'unité de mesure. Puis nous utilisons la fonction affichageLCD pour afficher nos strings sur l'écran LCD. Puis nous mettons un delais, ce delais est définit par la variable intervalleMesure.

```
//*************************************************************
//* Nom :          interruptButton                            *
//* Paramètre :    Aucun                                      *
//* Utilité :      Interrompre le programme                   *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void interruptionButton()
{
    if (digitalRead(portButton) == 1)
    {
        modeStandardCheck = !modeStandardCheck;
    }
}

//*************************************************************
//* Nom :          initInterrupt                              *
//* Paramètre :    Aucun                                      *
//* Utilité :      Déclrarer l'interruption                   *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void initInterrupt()
{
    attachInterrupt(digitalPinToInterrupt(portButton), interruptionButton, CHANGE);
}
```

Nous allons maintenant parler des fonctions d'interruption, dans un premier temps, nous allons voir la fonction qui va être réaliser lors de l'interruption. Dans la fonction interruptionButton, dans un premier temps, nous vérifions si le bouton est pressé. Si il est pressé, nous changeons la valeur de la variable modeStandardCheck. Ce qui permet au programme de changer de mode.

Puis nous passons à la fonction d'initialisation des interruptions. Dans cette fonction nous créons l'interruption sur le port du bouton (d'où le choix du port 2, car ce port permet de faire des interruptions), nous mettons la fonction a éxécuter lors de l'interruption et l'état provoquant l'interruption.

```
//*************************************************************
//* Nom :         initialisation                              *
//* Paramètre :   Aucun                                       *
//* Utilité :     Initialisation de l'Arduino                 *
//* Retour :      Aucun                                       *
//*                                                           *
//*************************************************************

void initialisation()
{
    Serial.begin(9600);                     //Initialisation de la fréquence de l'Arduino à 9600 bauds

    initInterrupt();

    ON(portLEDVerte);
    ON(portLEDJaune);
    delay(100);

    OFF(portLEDVerte);
    OFF(portLEDJaune);
    delay(100);

    ON(portLEDVerte);
    ON(portLEDJaune);
    delay(100);

    OFF(portLEDVerte);
    OFF(portLEDJaune);

    pinMode(portLEDVerte, OUTPUT);          //Déclaration de la LED Verte en tant que sortie
    pinMode(portLEDJaune, OUTPUT);          //Déclaration de la LED Jaune en tant que sortie
    pinMode(portButton, INPUT);             //Déclaration du bouton en tant qu'entrée

    lcd.begin(16,2);                        //Configuration du nombre de colonnes et de lignes sur l'écran LCD

    affichageLCD(4,0,"Starting");
    affichageLCD(5,1,"System");

    dht.begin();                            //Lancement du capteur

    delay(1500);

    resetLCD();
}
```

Nous allons donc maintenant passer dans la fonction d'initialisation. Dans cette fonction, nous mettons la fréquence de l'Arduino à 9600 bauds. Ensuite, nous faisons appelel à la fonction initInterrupt qui permet d'initialiser les interrptions (la fonction décrite au-dessus). Puis nous faisons clignoter nos LED, pous montrer à l'utilisateur que le dispositif est en cours de démarage. Nous déclarons les entrées et sorties que nous avons. Concernant les entrées, nous n'avons que le bouton poussoir. Pour les sorties, nous avons les deux LEDs. Nous configurons ensuite notre écran LCD, les paramètres sont le nombre de colonnes et le nombre de lignes. Nous affichons sur l'écran, le message « Starting System », toujour pour montrer à l'utilisateur que le sytème est en cours de démarrage. Nous allumons le capteur de température et d'humidité. Nous mettons un delay d'une seconde et demi puis nous effaçons l'écran grâce à la fonction resetLCD.

```
//*************************************************************
//* Nom :          affichageLCD                               *
//* Paramètre :    colone, ligne, texteToDisp                 *
//* Utilité :      Afficher du texte sur l'écran LCD          *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void affichageLCD(int colonne, int ligne, String texteToDisp)
{
    lcd.setCursor(colonne, ligne);          //Positionnement du texte à afficher
    lcd.print(texteToDisp);                 //Affichage du texte
}

//*************************************************************
//* Nom :          resetLCD                                   *
//* Paramètre :    Aucun                                      *
//* Utilité :      Nettoyer l'écran                           *
//* Retour :       Aucun                                      *
//*                                                           *
//*************************************************************

void resetLCD()
{
    for (int y = 0; y < 2; y++)
    {
        for (int x = 0; x < 16; x++)
        {
            affichageLCD(x,y," ");
        }
    }
}

void setup()
{
    initialisation();
}

void loop()
{
    if (modeStandardCheck)
        modeStandard();
    else
        modeConfiguration();
}
```

Pour finir, il nous reste quatre fonctions. La première est la fonction affichageLCD. Cetet fonction a pour paramètre le numéro de la colonne de départ du message à afficher, la ligne sur laquelle le message doit être afficher et le texte à afficher. Dans cette fonction, nous utilisons la fonction SetCursor pour afficher l'endroit de départ du message. Puis, nous utilisons la fonction print qui permet d'afficher le message sur l'écran LCD.

Nous allons maintenant parler la fonction resetLCD, cette fonction n'a pas de paramètres et permet d'effacer l'écran LCD. Pour cela nous utilisons deux boucle for pour cadriller l'écran LCD et pour chaque position sur l'écran, nous mettons un espace, ce qui permet d'effacer l'écran.

Enfin, dans le void setup qui la phase d'initialisation de notre Arduino, nous utilisons la fonction initialisation que nous avons créée auparavant. Puis dans le void loop, qui est le code que notre Arduino va jouer en boucle, nous ne faison que vérifier si nous sommes en mode standard par le biais de la variable modeStandardCheck. Si nous sommes en mode standard, nous faisons appelle à la focntion modeStandard, sinon nous faisons appelle à la fonction modeConfiguration.

Amélioration possibles

En effet pour le moment, la petite station météo que je viens de mettre au point, n'est pas très perfectionnée. J'ai donc penser à quelques idées qui pourrait me permettre de l'améliorer.

Dans un premier temps, je pourrais investir dans d'autres capteurs, qui me permettraient d'avoir la pression atmosphérique par exemple, ou même la localisation. Avec ces nouveaux capteurs, je pourrais par exemple mettre en avant le temps qu'il fait. Cependant, comme nous avons pu le constater, avec les photos du montage précédent, notre Arduino ne pourrait pas supporter beaucoup d'autres capteurs. Il faudrait donc que j'investisse aussi dans un nouvel Arduino Uno ou un Mega par exemple.

Il est aussi possible de faire un peu de domotique avec ce projet. Par exemple, en fonction de la température, nous pouvons allumer ou éteindre le chauffage de la maison.

Dans le cas où je me lancerais dans une amélioration de la station météo, je ferais sans doute aussi une interface graphique, qui me permettrait simplement de voir remplacer l'interface série et l'écran LCD. Ce pourrait être un moyen simple de libérer des port digitaux (notammement ceux de l'écran LCD qui sont nombeaux). Cette amélioration ferait appelle à d'autres connaissance, telle que la maitrise de la programmation orientée objet telle que le C# ou le C++ par exemple.

Annexe

- Diagramme UseCase : https://github.com/Pierre-Lctx/ProjetsPersonnel/blob/main/StationMeteo/Diagrams/UseCase/UseCase.png
- Diagramme d'état : https://github.com/Pierre-Lctx/ProjetsPersonnel/blob/main/StationMeteo/Diagrams/State/State.png
- Code : https://github.com/Pierre-Lctx/ProjetsPersonnel/blob/main/StationMeteo/stationMeteoMaison/stationMeteoMaison.ino
- Photo du montage : https://github.com/Pierre-Lctx/ProjetsPersonnel/tree/main/StationMeteo/Images%20du%20montage