

# Rapport de projet de 5<sup>ème</sup> année

*Création d'une interface graphique pour le système embarqué SimpleEmbOS*

## Table des matières

1. Introduction.....	2
2. Définition des termes techniques .....	4
2.1 - Système embarqué .....	4
2.2 - Capteurs sans fils.....	4
2.3 - Réseaux de capteurs sans fils .....	4
3. Cahier des charges et prototype .....	6
3.1 - Résultats attendus de l'interface graphique .....	6
3.1 - Prototype de l'interface .....	6
4. Diagramme UML & Outils utilisés .....	12
4.1 - UML .....	12
4.2 - Outils utilisés .....	14
5. Présentation du code .....	15
5.1 - MainWindow .....	15
5.2 - ParametrageProjet .....	16
5.3 - ParametrageNoeud .....	17
5.4 - PopUpNbNoeudsMax et PopUpNbNoeudsMax.....	18
5.5 - Constructeur de Projet .....	19
5.6 - Constructeur et affichage de Nœud.....	19
6. Résultats finaux .....	21
7. Bilan .....	23
7.1 - Difficultés rencontrées .....	23
7.2 - Améliorations possibles .....	23
8. Conclusion .....	24

# 1. Introduction

**SimpleEmbOS** est un système embarqué destiné aux réseaux de capteurs sans fil. C'est un système événementiel conçu pour faciliter le développement d'applications multiprocesseur et tolérantes aux pannes. Il a été développé par l'équipe SMIR du Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS). Les principaux utilisateurs de ce système embarqué sont des ingénieurs agronomes, il est principalement utilisé pour prélever des informations dans des zones agricoles.

Les utilisateurs de SimpleEmbOS rencontrent des difficultés à paramétrer le système afin de prendre les mesures adéquates grâce aux capteurs. Ce système est en effet programmé en C, paramétrer les capteurs afin de prendre les bonnes mesures nécessite des notions avancées en programmation, ce qui n'est pas forcément le cas de tous les utilisateurs.

Le but de mon projet de 5ème année est de permettre à tous les utilisateurs de SimpleEmbOS de prendre des mesures de manière simple et intuitive via le système embarqué, c'est-à-dire sans passer du temps à développer des programmes. L'idée est donc de développer une interface graphique, afin de pouvoir réaliser des programmes pour prendre des mesures sans avoir à programmer.

Dans un premier temps je donnerai plus d'informations sur certains thèmes récurrents de ce projet, comme les systèmes embarqués, les capteurs sans fils et les réseaux de capteurs sans fils. Je vous présenterai ensuite le cahier des charges que j'ai dressé afin d'être sûre de ne pas m'éloigner de l'objectif initial du projet. Dans un même temps je vous proposerai un prototype de l'interface graphique. Par la suite je vous introduirais le diagramme UML correspondant à l'implémentation de l'interface, et je ferai un point sur les outils utilisés pour mener à bien ce projet.

Une fois avoir fini de présenter la partie théorique, je détaillerai la manière dont j'ai commencé l'implémentation de l'interface, et vous présentant les différentes classes utilisées et la manière dont j'ai créé les fenêtres graphiques. Je vous présenterai ensuite le résultat final graphique et comment se déplacer dans l'interface. Je ferai ensuite un point sur les difficultés rencontrées et le bilan que je ressors de ce projet.

## 2. Définition des termes techniques

Ce projet est centré autour du système embarqué SimpleEmbOS qui est associé à un réseau de capteur sans fils. Pour rendre la compréhension de ce projet plus clair, je vais définir les termes de système embarqué, capteur sans fils, et réseau de capteurs sans fils.

### 2.1 - Système embarqué

Un système embarqué est un système électronique et informatique autonome, temps réel et dédié à un nombre limité de tâches. Il a une taille limitée et une consommation énergétique restreinte. Un système temps réel est un système dont les services répondent en un temps borné, et dont on peut interrompre l'exécution. Ces systèmes prennent en compte des contraintes temporelles : ils délivrent leurs résultats dans les délais imposés.

Les systèmes embarqués exécutent des tâches prédéfinies tout en respectant certaines contraintes, qui peuvent par exemple être spatiales, énergétiques, temporelles, relatives à la puissance de calcul ... Il est généralement composé de microprocesseur à basse consommation d'énergie et de capteurs. Il existe des systèmes embarqués dans toutes sortes de domaines, comme l'électroménager, les transports ou bien encore l'astronomie.

### 2.2 - Capteurs sans fils

Par définition, un capteur sans fil est un outil capable de recueillir des informations et des changements dans son environnement. Les capteurs sont conçus pour mesurer des paramètres spécifiques de leur environnement physique et produire des sorties pour un traitement ultérieur. Par exemple, des capteurs peuvent mesurer la température de l'air, l'humidité ...

Il existe deux types de capteurs sans fils, les capteurs passifs et actifs. Les capteurs passifs sont des éléments auto-alimentés qui réagissent aux entrées provenant des environnements environnants, ils ne nécessitent pas d'alimentation extérieure. Par exemple, un thermomètre à mercure est un capteur passif. Les capteurs actifs, eux, ont besoin d'une alimentation externe pour surveiller en permanence les environnements locaux. Par exemple, un dispositif possédant un radar est un capteur actif.

Les capteurs sans fils nous permettent de recueillir des informations dans des zones difficiles d'accès, par exemple où la température ou la pression sont très élevées. De plus, ces capteurs peuvent être utilisés pour former un réseau, et ainsi surveiller des endroits différents à partir d'une station. Ils sont également très économes en énergie, car ils n'effectuent pas de traitement lourd des données au niveau local.

### 2.3 - Réseaux de capteurs sans fils

Un réseau de capteurs sans fils est un réseau ad hoc, c'est-à-dire un réseau sans fils décentralisé capable de s'organiser sans infrastructure définie préalablement. Chaque nœud du réseau sans fils participe au routage en transmettant les informations aux autres nœuds de manière autonome. Dans un réseau de capteur sans fils, les nœuds sont des unités de traitement embarquées capables de

recueillir et transmettre des informations. Ces capteurs peuvent être aléatoirement répartis dans un espace. Les réseaux de capteurs sans fils trouvent des applications dans le domaine militaire, médical, ou encore commercial.

Bien comprendre ces notions est primordial quand on s'intéresse au système embarqué SimpleEmbOS. Dans la prochaine partie, je vous présenterai le cahier des charges concernant l'implémentation de l'interface graphique pour ce système embarqué.

### 3. Cahier des charges et prototype

Le but de mon projet de 5<sup>ème</sup> année est d'implémenter une interface graphique afin de permettre aux ingénieurs de LIMOS de paramétrer le système embarqué sans avoir de bonnes notions de programmation en C.

Imaginons qu'un ingénieur veut effectuer des mesures dans un champ par le biais d'un réseau de capteurs sans fils comprenant le système embarqué SimpleEmbOS. Pour mener à bien ce projet, l'ingénieur doit spécifier le nombre de capteurs qu'il souhaite dans son réseau. Pour éviter toute confusion, j'ai appelé ces capteurs des **nœuds** dans l'implémentation de l'interface. Il va alors paramétrer ces nœuds afin qu'ils prennent des mesures. Ces nœuds, en fonction de leurs types, possèdent des capteurs. L'utilisateur peut alors paramétrer ces nœuds avec une fréquence de mesure du capteur et un nombre de **threads**. Les threads permettent d'exécuter un ensemble d'instructions en rapport avec les capteurs (par exemple capter 3 fois par jour l'humidité d'un champ). Un thread est défini par des **actions**.

#### 3.1 - Résultats attendus de l'interface graphique

L'utilisateur doit être en mesure de choisir le nombre de **nœuds** dans le réseau, en précisant **leurs types, leurs noms et les nombres de threads associés**. Il doit pouvoir associer un capteur à chaque thread, en se focalisant pour l'instant sur **3 capteurs : luminosité, température et humidité**. L'utilisateur doit pouvoir **régler la fréquence** de mesure associée aux capteurs. Il doit également pouvoir ajouter des actions aux threads. Pour simplifier le paramétrage du projet, les différents capteurs associés à un nœud seront générés automatiquement, suivant une base de données ou un fichier texte.

**L'interface générera alors un petit terminal** avec un squelette de code déjà établi, qui sera en accord avec les précédents paramètres qu'a rentré l'utilisateur. Ce petit terminal apparaîtra quand l'utilisateur double cliquera sur une action. Sur un côté du terminal, des petits rappels de syntaxe de programmation en C pourront être affichés pour aider l'utilisateur à implémenter quelques lignes s'il en a besoin. Il pourra à la fin générer la totalité du code dans un fichier texte.

#### 3.1 - Prototype de l'interface

Dans cette partie je vous présenterai le prototype de l'interface graphique que nous avons mis en place afin qu'elle soit la plus intuitive et la plus complète possible pour l'utilisateur.

##### 1. *Création d'un projet*

Tout d'abord, l'utilisateur peut créer un nouveau projet via le menu principal de l'application (fig 1).

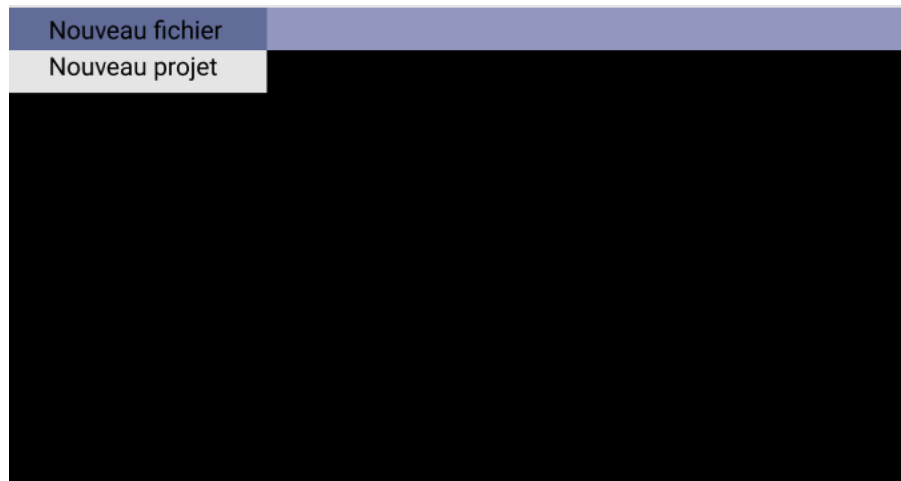


Fig 1: Menu principal

Pour créer un projet, l'utilisateur a besoin de saisir le nombre de nœuds qu'il souhaite dans une fenêtre pop-up (Fig 2).

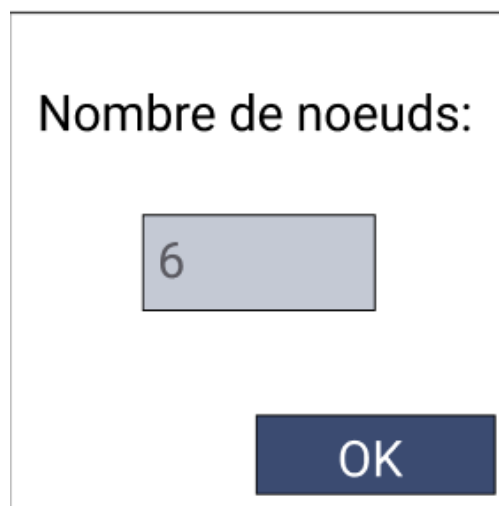


Fig 2: Configuration nombre de noeuds

## 2. Visualisation des Nœuds

Une fois le projet créé avec le nombre de nœuds, on peut visualiser les nœuds que nous avons dans notre réseau.

De prime abord, la liste des nœuds se présente de la manière suivante (Fig 3).

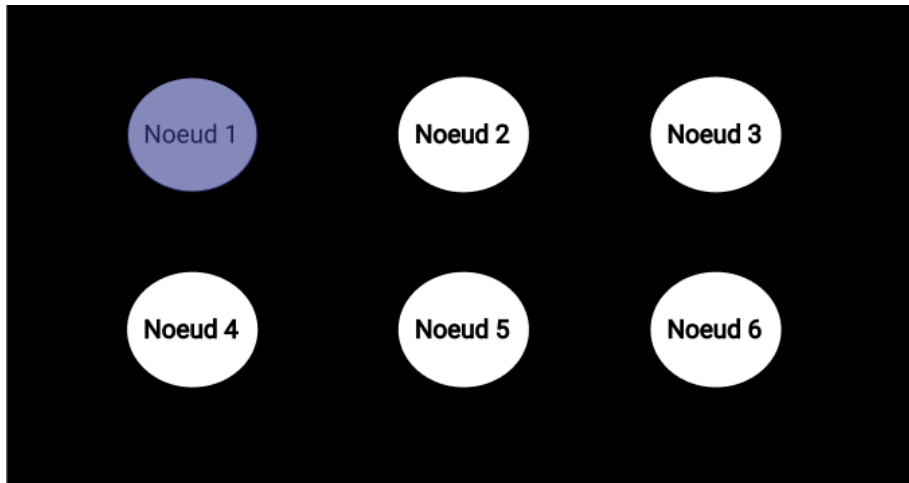
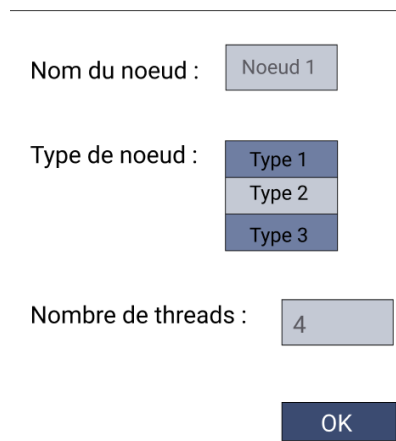


Fig 3: Liste des noeuds

Quand on double clique sur un nœud, une fenêtre pop-up s’ouvre pour paramétrer le nœud (Fig 4). L’utilisateur peut donner un nom au nœud, choisir son type et le nombre de threads associés.

A configuration window for a node. It contains three fields: 'Nom du noeud' with a text input field containing 'Noeud 1', 'Type de noeud' with a dropdown menu showing 'Type 1', 'Type 2', and 'Type 3', and 'Nombre de threads' with a text input field containing '4'. There is an 'OK' button at the bottom right.

Nom du noeud :

Type de noeud :

Nombre de threads :

Fig 4 : Configuration d'un noeud

Une fois le nœud configuré, une image du type de capteur a remplacé la mention “Noeud x”, et le nom du nœud est marqué au-dessus de la photo (Fig 5). De cette manière, on peut visualiser quels nœuds ont été configurés, et quels nœuds ne le sont pas.



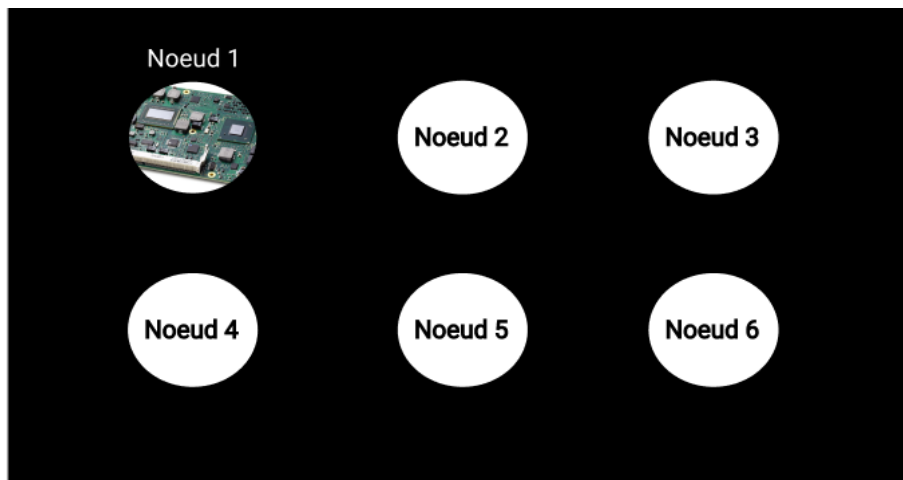


Fig 5 : Visualisation d'un noeud configuré

### 3. Visualisation des threads

L'utilisateur peut ensuite voir les threads associés à chaque nœud. En haut à droite de la fenêtre sont écrits les différents capteurs qui sont disponibles pour le nœud (ces données sont récupérées automatiquement d'une base de données/fichier texte). On peut ajouter ou supprimer des threads manuellement grâce à des boutons (Fig 6).

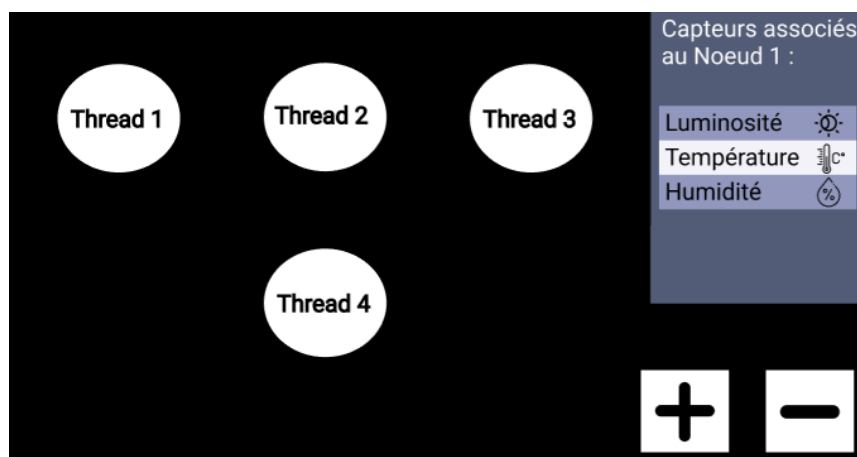
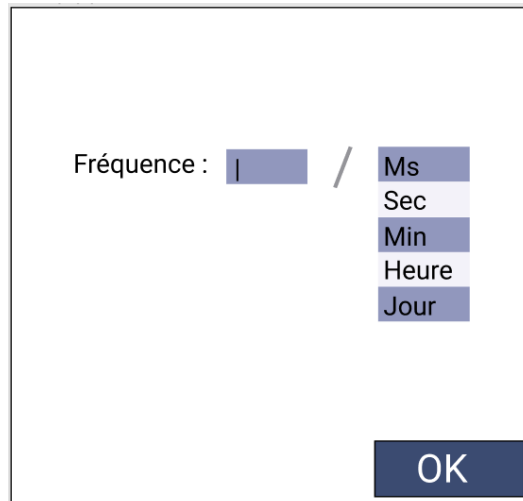


Fig 6 : Listes des threads d'un noeud

Pour associer un thread à un capteur, il suffit de faire glisser la souris du capteur en haut à droite jusqu'au thread voulu. Une fenêtre pop-up s'affiche alors, sur laquelle on peut choisir la fréquence de mesure (Fig 7).



Fréquence :  / 

Ms  
Sec  
Min  
Heure  
Jour

OK

Fig 7 : Configuration thread

Une fois le thread configuré, une petite icône correspondant au capteur qu'on lui a associé apparaît en dessous du thread, pour indiquer qu'il a déjà été paramétré (Fig 8).

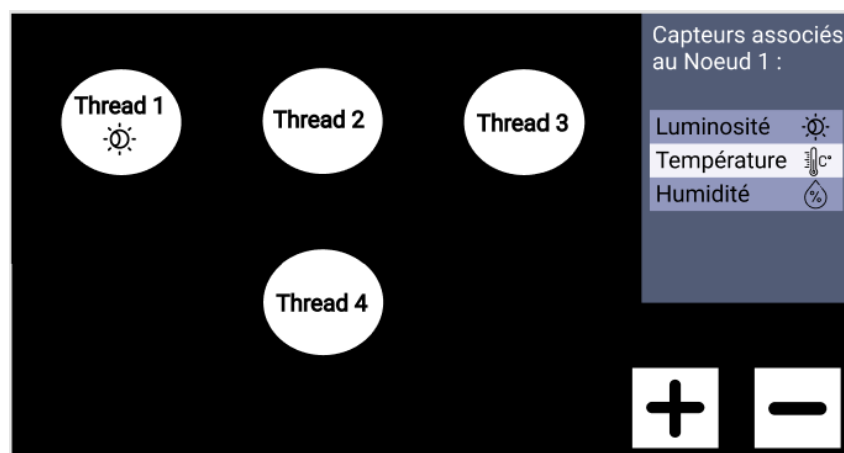


Fig 8 : Exemple thread configuré

#### 4. Liste des actions associées à un thread

En double cliquant sur un thread, on peut lui associer des actions. On peut en ajouter ou en retirer manuellement grâce à des boutons (Fig 9). Un récapitulatif du nom du nœud et du numéro du thread auxquels on se place est affiché en haut à droite de la fenêtre.

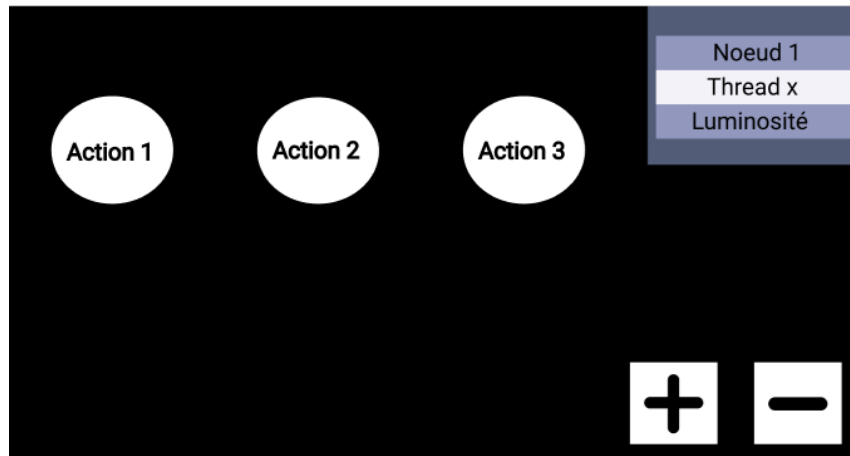


Fig 9 : Liste des actions associées à un thread

Quand l'utilisateur double clique sur une action, un terminal s'ouvre avec le code correspondant au paramétrage du projet. L'utilisateur peut rajouter quelques lignes s'il souhaite d'autres fonctionnalités. L'utilisateur peut à tout moment générer l'entièreté du code du projet.

Ce prototype m'a permis d'avoir une vision plus claire du projet, et m'a permis de réfléchir beaucoup plus simplement à l'implémentation de ce dernier. Suite à la finalisation du prototype, j'ai créé un diagramme de classes UML avec les classes qui allaient apparaître dans l'implémentation de mon projet, ainsi que leurs attributs et leurs méthodes.

## 4. Diagramme UML & Outils utilisés

### 4.1 - UML

Après avoir fini le prototype de l'interface graphique, j'ai pu me concentrer sur le diagramme UML du projet. J'ai choisi de faire un diagramme de classe, pour avoir une vision plus claire sur l'implémentation de l'interface. La figure ? correspond au diagramme UML que j'avais initialement pensé.

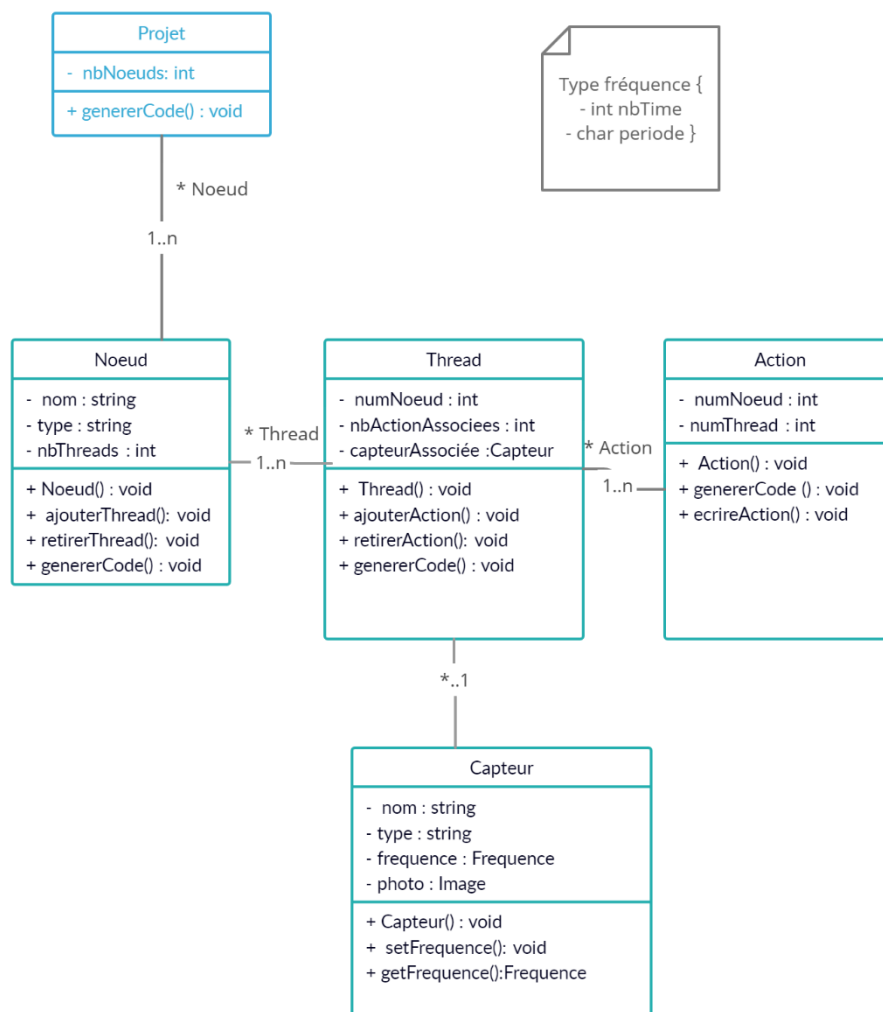


Fig 10 : Diagramme UML au début du projet

J'ai débuté la programmation de mon projet avec comme base ce diagramme UML. Cependant lors de mon avancement j'ai dû le modifier pour qu'il colle plus à la réalité de l'implémentation du projet. J'ai dû ainsi modifier les classes, leur rajouter ou enlever des attributs et des méthodes. Dans l'état actuel de l'avancement du projet, le diagramme UML de mon implémentation correspond à celui de la figure 11.

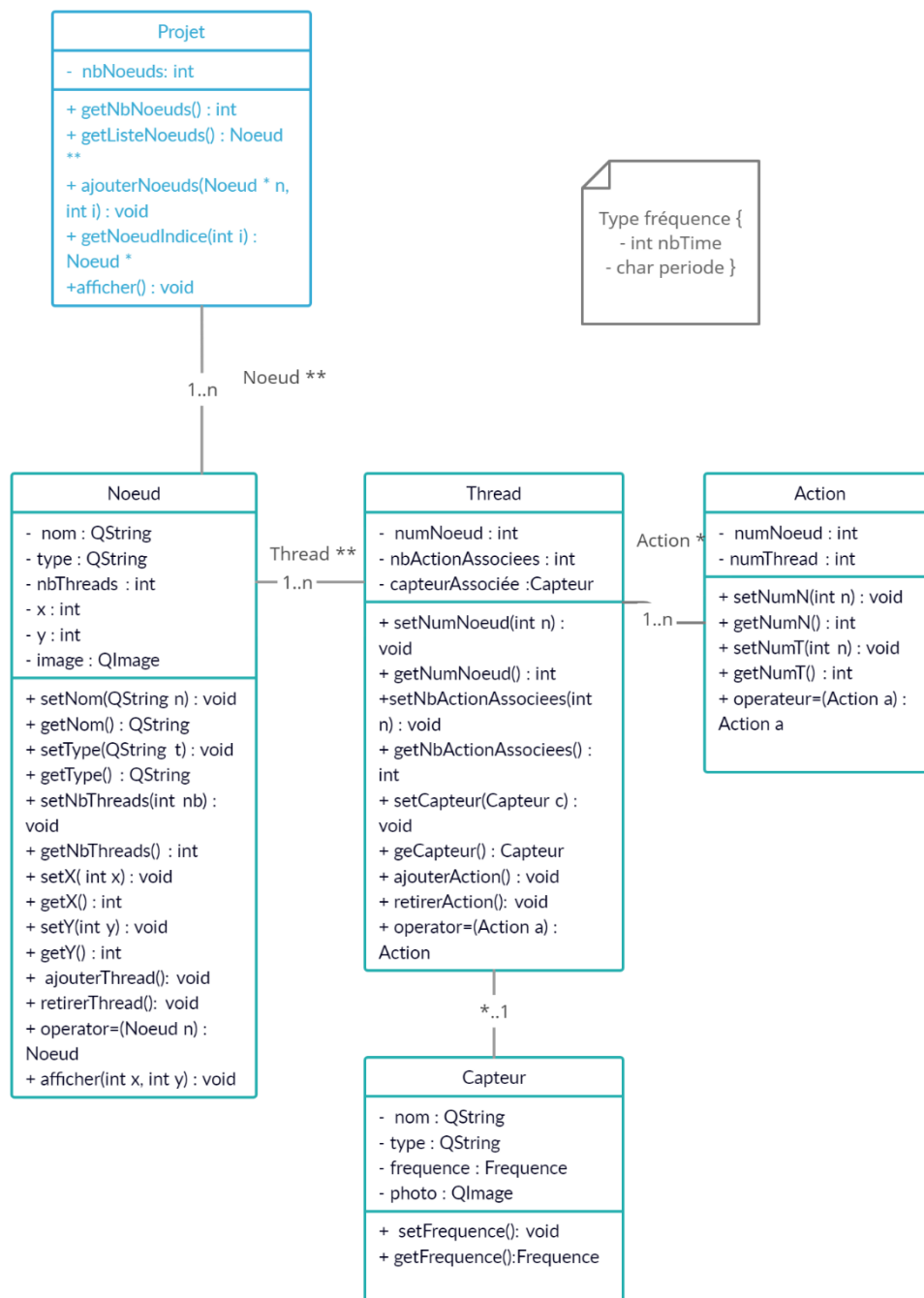


Figure 11 : Diagramme UML correspondant à l'implémentation du projet

Au stade de ce diagramme UML l'interface graphique n'est pas finie, il faudra encore certainement le modifier pour qu'il soit en adéquation avec les classes implémentées.

Je vous présenterai dans la prochaine partie les outils utilisés lors de la réalisation de ce projet.

## 4.2 - Outils utilisés

Pour réaliser l'interface graphique de SimpleEmbOS, j'ai utilisé l'environnement de développement **Qt Creator 6** (la version Community). Il est orienté pour le développement en **C++**.

La bibliothèque Qt nous permet notamment de créer des **Graphical User Interface (GUI)**, c'est pour ça que nous l'avons choisie pour ce projet. De plus Qt est **multiplateforme**, ce qui est un avantage car de cette manière les apparences des fenêtres seront adaptées selon le système d'exploitation de l'utilisateur.

En dehors du cadre de l'implémentation de l'interface, j'ai utilisé l'éditeur de graphiques vectoriels **Figma** pour concevoir le prototype et l'outil de collaboration **Creately** pour créer le diagramme UML.

J'ai également stocké le code du projet sur **Github** afin de pouvoir le partager plus facilement.

Finalement, après avoir écrit le cahier des charges, conçu le prototype et le diagramme UML, j'ai pu commencer à coder le projet. Toutes ces étapes ont rendu le début de l'implémentation plus facile car j'avais déjà en tête les classes qu'il fallait que je crée, et leurs méthodes associées.

Je détaillerai dans la prochaine partie le code du projet, en expliquant comment j'ai implémenté certaines fonctionnalités.

## 5. Présentation du code

Dans cette partie, je vous détaillerai certaines parties importantes de mon code. Je vous présenterai dans un premier temps ma classe **MainWindow**, qui correspond à l’affichage de la fenêtre principale, ainsi que les classes **parametrageNoeud** et **parametrageProjet**, qui correspondent à l’initialisation du projet et de ses nœuds. J’explicitai également rapidement les classes **popupnbnoeudsmax** et **popupnbthreadsmax**. Enfin, je vous présenterai par la suite le **constructeur de la classe Projet** et le **constructeur et la fonction affichage de Nœud**.

### 5.1 - MainWindow

La classe `MainWindow` est composée de deux méthodes, `paintEvent()` et `on_actionNouveau_projet_triggered()`.

```
24 void MainWindow::paintEvent(QPaintEvent* e) {
25     QWidget::paintEvent(e);
26     QPainter painter(this);
27
28
29     if (p != NULL){
30         p->afficher(&painter);
31     }
32 }
33
34 //méthode liée au bouton nouveau projet dans le menu
35 void MainWindow::on_actionNouveau_projet_triggered()
36 {
37     //Une fenetre s'ouvre pour régler les paramètres du projet
38     parametrageProjet f(this);
39     f.exec();
40
41     //On récupère le projet construit dans parametrageProjet
42     *p = f.getProjet();
43
44 }
```

Figure 12 : méthode de MainWindow

La méthode **paintEvent()** sert à afficher graphiquement le projet (soit ses nœuds). La méthode **on\_actionNouveau\_projet\_triggered()** s’exécute quand l’utilisateur clique sur « Nouveau Projet ». Elle déclenche l’ouverture de la fenêtre `parametrageProjet()`, et récupère le projet grâce à un accesseur de `parametrageProjet()`.

## 5.2 - ParametrageProjet

```
23 //Recuperation du nombre de noeud
24 void parametrageProjet::on_lineEdit_textEdited(const QString &arg1)
25 {
26     bool ok ;
27     int a = arg1.toInt(&ok,10);
28     if (ok){
29         nbNoeudsNonValide = a ;
30     }
31 }
32
33
34
35 //Ajout gestion des erreurs :
36 // - nbNoeuds != int
37 void parametrageProjet::on_pushButton_clicked()
38 {
39     //Si le nombre de noeud est supérieur au nombre de noeud max -> fenetre pop up
40     //On met alors le nombre de noeud au max
41     if (nbNoeudsNonValide > NB_NOEUDS_MAX){
42         popUpNbNoeudsMax p(this);
43         p.exec();
44         nbNoeudsNonValide = NB_NOEUDS_MAX ;
45         //creation du projet avec les informations
46         proj = new Projet(nbNoeudsNonValide);
47         this->close();
48     }
49     else {
50         proj = new Projet(nbNoeudsNonValide);
51         this->close();
52     }
53 }
54 }
```

Figure 13 : méthodes de parametrageProjet

La classe parametrageProjet possède une fonction **on\_lineEdit\_textEdited()** qui convertit la chaîne de caractères rentrée sur le lineEdit en entier. Ce nombre correspond au nombre de nœud dans le projet. La méthode **on\_pushButton\_clicked()** est associée au bouton OK, elle crée un projet avec le nombre de nœuds rentré. Si le nombre de nœuds est supérieur au nombre de nœud maximal, une fenêtre pop-up s'ouvre pour avertir l'utilisateur, et le nombre de nœud est automatiquement mis au plus grand (20 dans notre cas) et le projet est créé.



## 5.3 - ParametrageNoeud

```
23 //Recuperation du nom du noeud sur le lineEdit
24 void parametrageNoeud::on_lineEdit_textEdited(const QString &arg1)
25 {
26     nomNoeud = arg1 ;
27 }
28
29 //Recuperation du type du noeud sur le menu déroulant
30 void parametrageNoeud::on_comboBox_textActivated(const QString &arg1)
31 {
32     typeNoeud = arg1 ;
33 }
34
35
36 ///Recuperation du nombre de thread sur le lineedit
37 void parametrageNoeud::on_lineEdit_2_textEdited(const QString &arg1)
38 {
39     bool ok ;
40     int nbT = arg1.toInt(&ok,10);
41     if (ok){
42         nbThread = nbT ;
43     }
44 }
45
46
47 void parametrageNoeud::on_pushButton_clicked()
48 {
49     //Si le nombre de thread est supérieur au nombre max, une fenetre pop up apparait
50     //Le nombre de thread est alors directement mis au nombre max
51     if (nbThread > NB_THREADS_MAX){
52         popUpNbThreadsMax ppT(this);
53         ppT.exec();
54         nbThread = NB_THREADS_MAX ;
55         //on crée le noeud avec les informations recueillies
56         noeud = new Noeud(nomNoeud,typeNoeud,nbThread);
57         this->close();
58     }
59     else {
60         noeud = new Noeud(nomNoeud,typeNoeud,nbThread);
61         this->close();
62     }
63 }
```

Figure 14 : méthodes parametrageNoeud

La classe parametrageNoeud possède 3 méthodes qui récupèrent les saisies de l'utilisateur, une pour récupérer le nom du nœud, une pour le type (sous forme de menu déroulant) et une pour le nombre de threads du nœud. De la même manière que parametrageProjet, la méthode **on\_pushButton\_clicked** associé au bouton OK crée un nœud en fonction des informations rentrées par l'utilisateur, et vérifie que le nombre de thread n'est pas supérieur au nombre maximal.

## 5.4 - PopUpNbNoeudsMax et PopUpNbNoeudsMax

```
4  popUpNbNoeudsMax::popUpNbNoeudsMax(QWidget *parent) :  
5      QDialog(parent),  
6      ui(new Ui::popUpNbNoeudsMax)  
7  {  
8      ui->setupUi(this);  
9      setGeometry(100,100,200,100);  
10  
11      VLayout = new QVBoxLayout;  
12      label = new QLabel ;  
13      buttonOK = new QPushButton("OK",this);  
14  
15      QObject::connect(buttonOK, SIGNAL(clicked()),this, SLOT(on_pushButtonClicked()));  
16  
17      //message de la fenetre pop up  
18      msg_label = "Attention, nombre de noeuds max = " + QString::number(NB_NOEUDS_MAX) ;  
19      label->setText(msg_label);  
20  
21      VLayout->addWidget(label,Qt::AlignCenter);  
22  
23      this->setLayout(VLayout);  
24  
25  }
```

Figure 15 : méthode fenêtre pop-up

Les fenêtres pop-up PopUpNbNoeudsMax et PopUpNbthreadsMax ont une implémentation similaire, elles possèdent un constructeur, un destructeur, et une méthode associée à un bouton OK qui ferme la fenêtre. On crée dans le constructeur un layout, un label et un bouton. Dans la figure 15 le label spécifie que l'utilisateur a rentré un nombre de nœud trop grand en comparaison avec le nombre maximal. L'utilisateur a juste cliqué sur OK pour faire disparaître la fenêtre.

## 5.5 - Constructeur de Projet

```

Projet::Projet(int nbN)
{
    nbNoeuds = nbN ;

    int nb_noeuds_ligne = 0 ;
    int num_ligne = 0 ;

    listeNoeuds = new Noeud * [NB_NOEUDS_MAX];
    for (int i = 0 ; i<nbN ; i++){

        if ((80 + (i-nb_noeuds_ligne)*150)>(SIZE_X_WINDOW-150)){
            num_ligne ++ ;
            nb_noeuds_ligne = i ;
        }

        listeNoeuds[i] = new Noeud(80+(i-nb_noeuds_ligne)*150,80+ num_ligne*120);
    }
}

```

Figure 16 : constructeur de Projet

Un projet a pour attribut un nombre de nœud et une liste vers un pointeur de nœud. Pour initialiser un projet on lui initialise un nombre de nœud et ses nœuds associés, en leur donnant des coordonnées graphiques afin de pouvoir les afficher dans la fenêtre.

## 5.6 - Constructeur et affichage de Nœud

```

4  ▼ Noeud::Noeud(int xN, int yN)
5  {
6      x = xN ;
7      y = yN ;
8      image = new QImage(50,50,QImage::Format_ARGB32);
9      image->load("C:/Users/camil/OneDrive/Documents/GMM5A/Projet/Projet-5A/node.png");
10 }
11
12
13 ▼ Noeud::Noeud(QString nomNoeud, QString typeNoeud, int nbT)
14 {
15     nom = nomNoeud ;
16     type = typeNoeud ;
17     nbThreads = nbT ;
18     listeThreads = new Thread * [NB_THREADS_MAX];
19     image = new QImage(70,70,QImage::Format_ARGB32);
20     image->load("node.png");
21 ▼ for (int i = 0 ; i<nbThreads ; i++){
22     listeThreads[i] = new Thread[NB_ACTIONS_MAX]; // pas sur
23 }
24 }

```

```
98 void Noeud::afficher(QPainter * p, int x, int y){  
99  
100     QImage * image = new QImage(50,50,QImage::Format_ARGB32);  
101     image->load("C:/Users/camil/OneDrive/Documents/GMM5A/Projet/Projet-5A/node.png");  
102  
103     p->drawImage(QRect(x,y,50,50),*image);  
104  
105 }
```

Figure 17 : constructeurs et affichage de Noeud

La classe nœud possède deux constructeurs. Le premier constructeur sert quand le nœud n'est pas paramétré, on veut juste l'initialiser graphiquement. Le deuxième constructeur est utile quand on a double-cliqué sur le nœud et qu'on lui a associé un nom, un type et un nombre de thread.

Nous avons aussi une méthode afficher qui correspond à l'affichage d'un nœud afin qu'il soit paramétré. On affiche des petites icônes de nœuds de taille 50,50 suivant les coordonnées du nœud.

## 6. Résultats finaux

Je n'ai malheureusement pas pu implémenter tout ce que j'aurais voulu coder pour ce projet. Le graphisme de l'interface est donc très simple et peu développé.

Quand on exécute notre programme on obtient une première fenêtre, où on peut créer un nouveau projet (Figure 18).

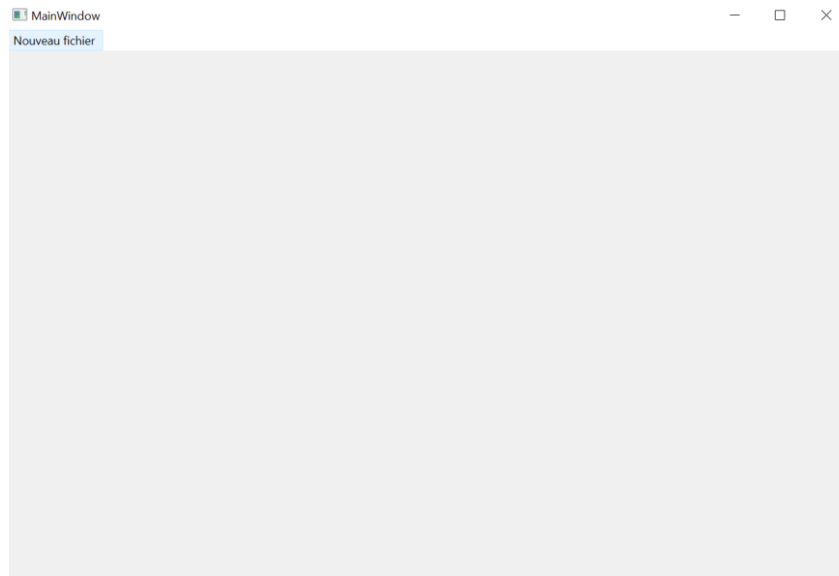


Figure 18 : fenêtre principale interface graphique

Une fenêtre pop-up s'ouvre alors pour paramétrer le projet. On peut saisir le nombre de nœud que l'on veut dans notre projet (Fig 19). Si le nombre de nœud saisi dépasse le nombre de nœud maximal, une fenêtre pop-up s'affiche (fig 20), et met automatiquement le nombre de nœud au nombre maximal.

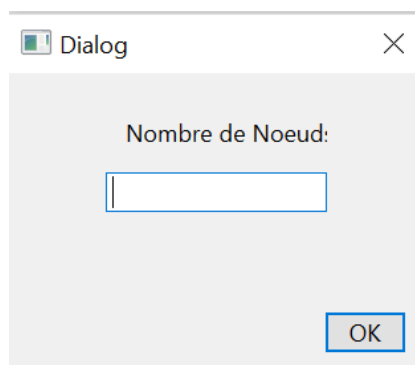


Figure 19 : fenêtre pop-up paramétrage projet

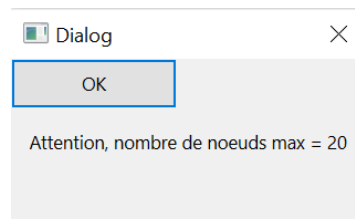


Figure 20 : fenêtre pop-up nombre de nœuds trop grand

On voit alors s'afficher autant d'icônes qu'il y a de nœud (fig 21).

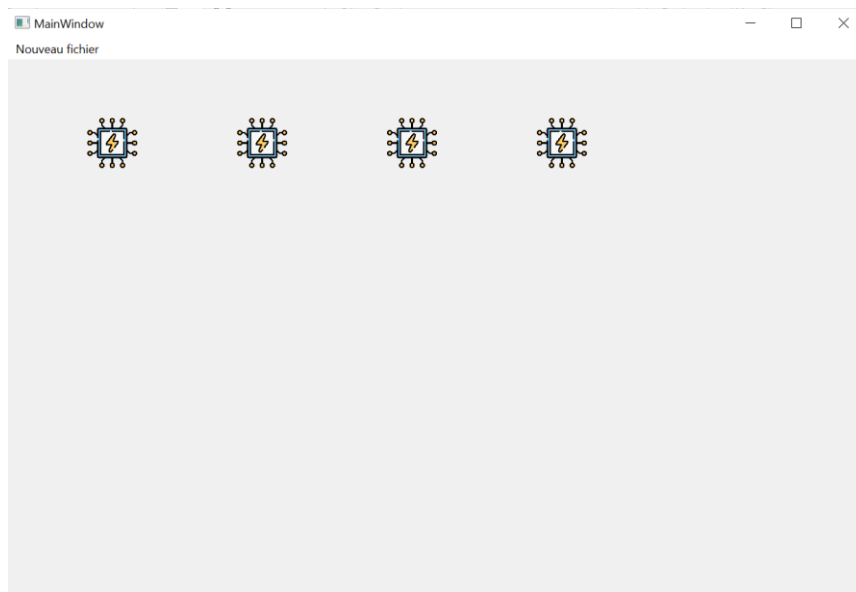


Figure 21 : fenêtre principale avec les nœuds du projet

## 7. Bilan

### 7.1 - Difficultés rencontrées

Lors du déroulement de ce projet, j'ai rencontré quelques difficultés, notamment lors de la compréhension du fonctionnement du réseau de système embarqué et de l'implémentation.

En effet, j'ai eu du mal à comprendre comment s'articulait les différents objets entre eux : les nœuds, les threads, les actions, les capteurs ... Je confondais également les nœuds et les capteurs. Pour cette raison j'ai mis du temps à finaliser le prototype de l'interface, ce qui m'a fait prendre du retard sur l'implémentation.

J'ai rapidement compris comment commencer l'implémentation grâce au diagramme UML que j'avais fait. J'ai cependant eu du mal à faire en sorte de paramétrer les nœuds en double cliquant dessus. J'ai commencé à modifier mes classes pour pouvoir déterminer le nœud cliqué en fonction de la position de la souris de l'utilisateur. Je n'ai hélas pas eu le temps de continuer son implémentation.

### 7.2 - Améliorations possibles

J'aurais aimé pouvoir finir ce projet, et ainsi avoir une interface graphique complète et fonctionnelle. J'aurais également aimé avoir une interface qui a exactement les mêmes fonctionnalités que celles imaginées dans le cahier des charges. J'aurai aussi souhaité avoir un plus joli affichage graphique.

## 8. Conclusion

Pour conclure, même si je ne suis pas totalement satisfaite du rendu final de mon code et du fait que je n'ai pas pu finir l'interface, j'ai apprécié mener ce projet. J'ai appris beaucoup de choses concernant les systèmes embarqués et les réseaux de capteurs sans fils. Même si j'ai passé beaucoup de temps sur le cahier des charges et le prototype, je suis contente de leur rendu final. Par rapport au code, même s'il n'est pas encore très abouti j'ai bien compris les méthodes que j'ai implémenté, et je pense que j'aurais moins de soucis à l'avenir pour reprendre ce code.

Ce projet vient compléter mon projet de 4A qui consistait à créer un jeu avec une interface graphique en C++.