




14 JUIN 2019

**AMBI-LIKE**  
RAPPORT TECHNIQUE P2

JONATAN BAUMGARTNER  
JULIEN CHAPPUIS  
TÉO SCHAFFNER



## Table des matières

Introduction.....	2
Rappel du cahier des charges .....	2
Gestion de projet.....	2
Planification .....	2
Utilisation git .....	2
Conception .....	3
Choix de l'architecture matériel .....	3
Conception électronique.....	3
Conception prototype de test .....	4
Conception logiciel.....	4
Implémentation .....	5
Gui.....	5
Fenêtre principale.....	5
Fenêtre de configuration.....	5
Fenêtre de création de mode .....	5
Les différents computations .....	6
Ambi-like.....	6
Modes personnalisés.....	6
Couleur fixe.....	6
Les divers sender .....	6
rmiSender .....	6
testSender .....	6
previewSender .....	7
Modes personnalisés .....	7
Mise en zone de notification .....	7
Commande des leds.....	8
Délivrable .....	9
Création de l'installateur et exécutable .....	9
Images du Raspberry pi .....	9
Conclusion .....	9
Annexe :.....	9
Planning.....	10
Installation du Raspberry pi.....	11
Schéma de classe .....	13

## Introduction

Ce rapport décrit les différentes étapes de notre projet p2 java, de la création du cahier des charges aux livrables en passant par la conception et l'implémentation, il montre également l'organisation de l'équipe et la gestion du projet en général.

## Rappel du cahier des charges

Selon plusieurs études scientifiques, la lumière bleue émise par nos écrans est néfaste pour nos yeux et les fatigue. C'est pour diminuer cet effet que Philips a breveté dès 2017 une technologie nommée ambilight qui permet de projeter un prolongement de l'image affichée par l'écran sur le mur derrière celui-ci. Cette technique permet de moins fatiguer les yeux car elle diminue le contraste entre un écran lumineux et une pièce sombre.

Nous avons donc voulu imiter cet effet sur nos ordinateurs en se servant de leds adressables et des éléments affichés sur nos écrans. À partir de là, le cahier des charges était assez clair, nous avons choisi d'utiliser un Raspberry pi pour commander les leds afin de rester sur le langage de programmation de ce module, à savoir java. Nous avons également décidé d'ajouter des modes couleurs fixes et personnalisés afin de permettre d'émettre juste une lumière d'ambiance. Il nous fallait également une interface de prévisualisation et de configuration de notre programme.

## Gestion de projet

### Planification

La planification est un outil crucial pour jauger la ponctualité du projet. L'estimation des échéances se base notamment sur notre expérience acquise lors de projets précédents. Notre planning est disponible dans les annexes.

Dans le cas de Ambi-Like, les jalons correspondaient à notre avancement. L'estimation présumée des délais au début du projet était correcte.

### Utilisation git

Nous avons utilisé les diverses possibilités offertes par gitlab pour notre gestion de projet.

Nous avons utilisé et mis à jour le wiki durant le semestre et on y retrouve l'essentiel de notre documentation.

Nous avons également beaucoup utilisé les issues et les milestones afin de suivre l'évolution du projet et la correction des bugs.

Nous avons également mis en place de l'intégration continue. Ce système permet de vérifier à chaque commit que le projet compile et permet de télécharger les jars correspondants à chaque commit pour pouvoir les tester facilement. Pour plus de détails, voir les fichiers `.gitlab-ci.yml` et `Dockerfile` qui ont été commentés sur le dépôt git.

Afin d'être sûr d'avoir en permanence une version fonctionnelle sur la branche master, elle était protégée et nous devons faire des merge request pour la mettre à jour. L'application de ces merge request nécessitait l'approbation des trois membres du groupe ce qui permet à chaque personne de vérifier le travail effectué.

## Conception

### Choix de l'architecture matérielle

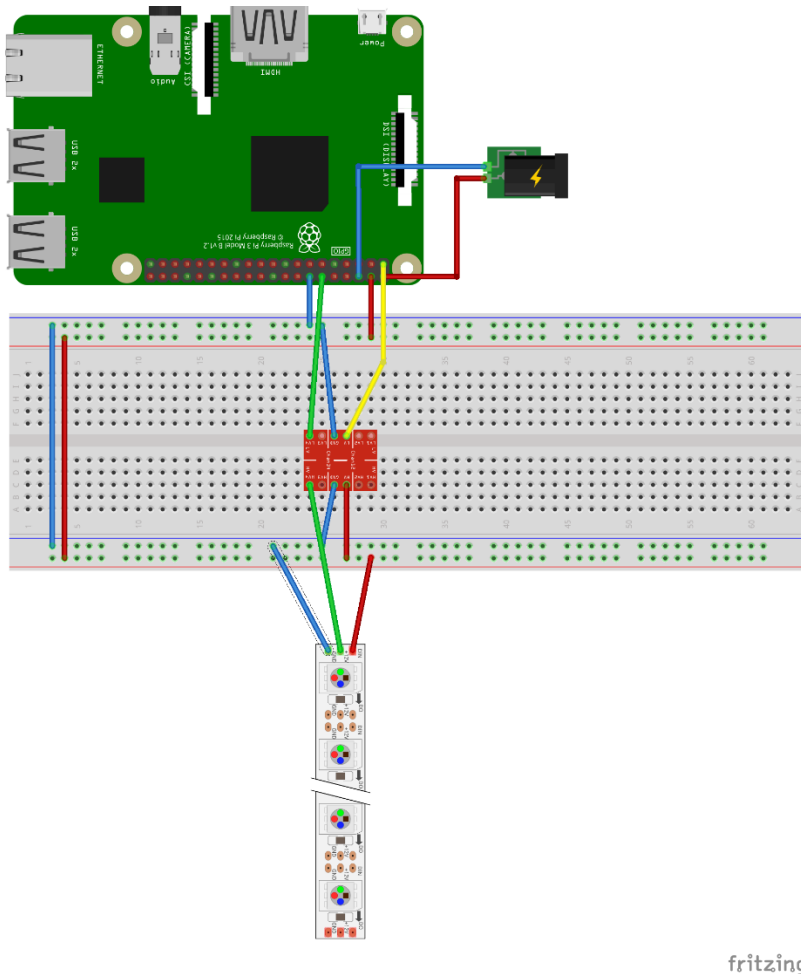
Les leds ne pouvant pas être commandées par ordinateur directement, nous avons le choix entre un microcontrôleur ou un ordinateur mono-carte. Le choix du Raspberry pi s'est fait assez naturellement afin de pouvoir faire un programme java et utiliser RMI. Nous avons vu par la suite qu'un microcontrôleur serait une meilleure solution afin de limiter les latences.

### Conception électronique

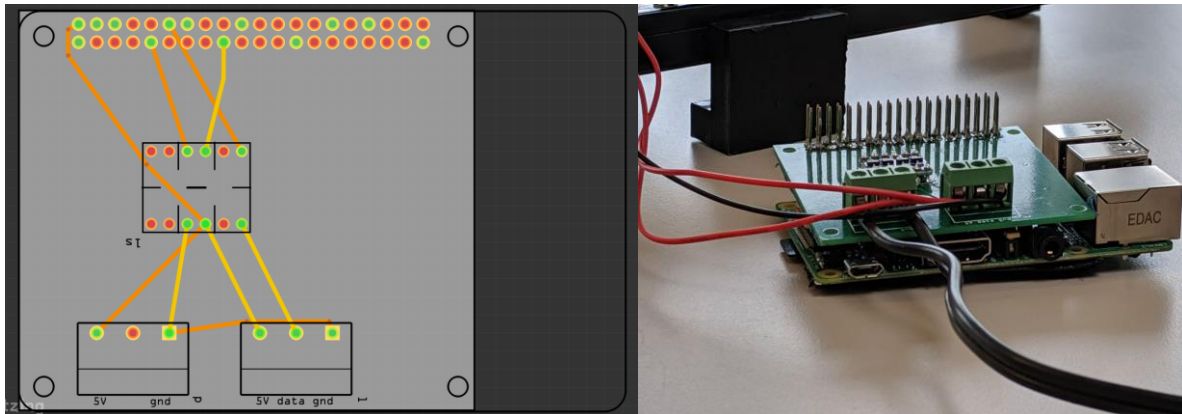
Les bandes de leds rgb existent en deux grandes familles, les analogiques avec lesquelles on fixe la tension des canaux rgb pour toute la bande, on a donc la même couleur sur toute la bande, et les numériques avec lesquelles on peut fixer la couleur de chaque led individuellement. Pour notre projet, nous nous sommes orientés sur des bandes de leds numérique de type Ws2812b. On peut utiliser d'autres types de led en changeant un paramètre dans le code et en recompilant la partie Raspberry pi du programme.

Ensuite, il fallait convertir la tension utilisée par les gpio du Raspberry pi (3.3V) dans la tension utilisée par les leds (5V). Pour ce faire, il existe des circuits prêts à l'emploi avec 4 canaux bidirectionnels même si nous n'en n'utilisons qu'un et de manière unidirectionnelle.

Nous avons donc un montage volant dont le schéma est le suivant :



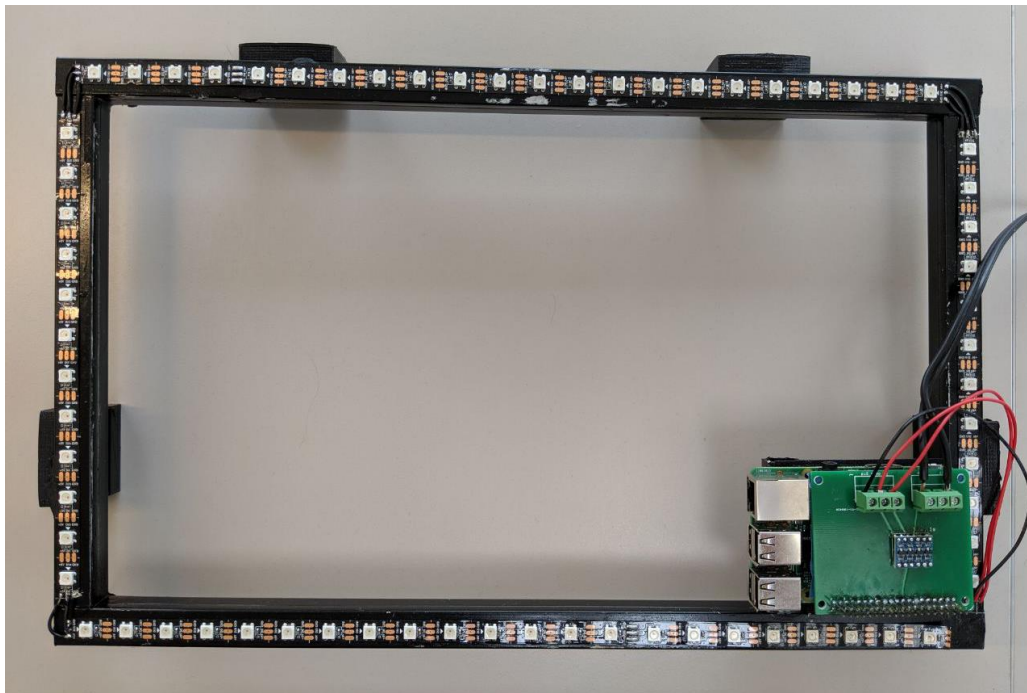
Afin d'avoir un résultat plus propre et de simplifier nos tests, nous avons conçu une carte électronique à l'aide du logiciel fritzing. Cette carte possède un connecteur afin de venir enficher la carte sur le Raspberry.



### Conception prototype de test

Afin de faciliter nos tests, nous avons conçu un cadre, avec les leds et le Raspberry pi fixés dessus. Ce cadre a été conçu afin de venir se placer sur l'écran de nos ordinateurs portables.

Nous avons commencé par mesurer nos pcs et trouvé des dimensions qui conviennent. Le cadre a ensuite été créé en découpant des morceaux de canal électrique collés ensemble avec de la colle pvc. Les crochets qui viennent se placer sur l'écran ont été imprimés en 3D. Le résultat final est le suivant :



### Conception logicielle

Le schéma de classe se trouve à la fin du document. Le langage de programmation est java.

Le programme PC est composé de trois grandes parties. La partie computation représentée par une interface permet la gestion du système et des calculs de couleurs à envoyer aux leds. Les senders implémentent le routage des valeurs calculées à la bonne place. Finalement, la partie gui composée de trois fenêtres est décrite en détail ci-dessous.

Toutes les parties sont indépendantes les unes des autres. Au lancement du programme, la partie gui n'existe pas. Le programme place simplement une icône dans la zone de notification. Un clic droit sur cette icône offre l'éventualité d'ouvrir la fenêtre principale.

La gestion des calculs se situe dans la partie computation. Ce choix stratégique induit la séparation des fonctionnalités. La motivation principale était d'améliorer la compréhension du code ainsi que le couplage interclasse.

Les leds sont représentées par la classe "Pixel". Cette classe comporte les mêmes attributs que les leds. Les attributs comme la luminosité et les couleurs rgb sont présents. Cette classe implémente l'interface serializable.

La classe Config est un Singleton, une seule instance existe au maximum. Ce raisonnement est pratique car cela permet de simplifier l'accès aux valeurs de configuration dans tout le programme. La gestion de la sauvegarde et du chargement de celle-ci s'avère beaucoup plus simple.

## Implémentation

### Gui

L'interface graphique se base sur Swing. Il s'agit principalement de fenêtres de configurations qui sont cachées au lancement du programme.

### Fenêtre principale

La fenêtre principale s'articule autour d'un Box-layout horizontal. Celui-ci contient deux panels respectivement PanelPreview et PanelChoice. Les éléments graphiques sont encapsulés dans des panels qui sont eux-même ajoutés dans les panels cités précédemment. Différents listeners java ont été mis en place afin d'animer notre gui.

Le PanelPreviewScreen est le panel le plus compliqué. En effet, celui-ci s'occupe d'afficher la prévisualisation. Pour cela, java2D est utilisé pour dessiner l'écran et les leds. La difficulté réside dans le fait qu'il est nécessaire d'effectuer un rafraîchissement de la zone lorsque les leds changent. Pour cela, ce panel possède une méthode `setPixelAt(int index, Pixel pixel)` qui est appelée dans un composant externe de type PreviewSender. Avec le but de préserver les performances, un compteur s'incrémente à chaque appel de `setPixelAt(..)`. Lorsque la valeur du compteur correspond au nombre de leds disponibles, `repaint()` est appelé pour actualiser la zone de prévisualisation.

### Fenêtre de configuration

La fenêtre de configuration permet de régler le nombre de led de chaque côté de l'écran, la luminosité maximale et l'adresse IP du Raspberry.

### Fenêtre de création de mode

Cette fenêtre comporte deux JPanels principaux : Celui de droite s'occupe d'afficher une représentation du mode en cours d'édition et celui de gauche permet de sélectionner une couleur à l'aide d'un élément JColorChooser.

L'affichage de la représentation du mode fonctionne de manière similaire à celui du PanelPreviewScreen de la fenêtre principale. La différence principale réside dans le fait que l'affichage n'est actualisé que lorsque l'utilisateur modifie la couleur d'une led en cliquant dessus ou lorsqu'il charge un mode depuis un fichier.

## Les différents computations

Les computations sont les composants du logiciel responsable du calcul des couleurs à envoyer aux leds, ils implémentent tous la même interface ce qui permet de les gérer de la même manière selon le design pattern template method. Notre programme peut ainsi accueillir à l'avenir de nombreux autres modes d'affichage très facilement.

### Ambi-like

Ce mode est notre implémentation de la technologie ambilight, présentée en introduction de ce document.

En premier lieu, différentes zones en bordure de l'écran sont établies et stockées, selon la figure ci-contre. Ces zones définissent quelles parties de l'écran doivent reproduire leur couleur sur quelle led.

Des captures d'écran sont effectuées à intervalles réguliers et utilisées pour déterminer quelle couleur doit être envoyée à quelle led.

Ces captures sont traitées par plusieurs workers (leur nombre étant déterminé par le nombre de cœurs disponibles sur l'ordinateur) qui s'occupent chacun leur tour d'une zone en bordure de l'écran, et passent à la prochaine zone qui n'a pas été traitée lorsqu'ils en ont terminé une.



### Modes personnalisés

Dans ce mode, on envoie les différentes couleurs de chaque led à la suite depuis le mode personnalisé passé au constructeur. On doit répéter cette opération de manière régulière puisque le programme sur le Raspberry éteint toutes les leds s'il ne reçoit pas de commande pendant un certain temps.

### Couleur fixe

Dans ce mode, on envoie la couleur du pixel reçu dans le constructeur à chaque led, on répète ensuite cette action régulièrement.

## Les divers sender

Afin d'envoyer les données des computations au bon endroit, nous avons mis en place un design pattern strategy. Nous avons donc créé une interface sender qui permet d'envoyer une couleur à une led. Nous avons ensuite réalisé les trois implémentations suivantes :

### rmiSender

RMI sender permet d'envoyer les commandes par RMI, il est implémenté comme un singleton pour qu'il n'y ait pas de risque de se connecter plusieurs fois au Raspberry. La première fois que l'on fait `getInstance`, il se connecte de manière bloquante au pi. Ensuite il garde cette connexion jusqu'à l'arrêt de la jvm. C'est également à ce moment-là que l'on recalcule la luminosité de chaque led selon la luminosité maximale configurée par l'utilisateur.

### testSender

Ce sender a été utilisé pour les tests sans Raspberry et affiche simplement les commandes dans la console.

### previewSender

Ce sender est utilisé par la fenêtre de prévisualisation et met à jour le panel de prévisualisation.

### Modes personnalisés

Les modes personnalisés sont stockés dans une classe qui contient le nom du mode et un vecteur de pixel qui représente chaque led. Nous avons choisi le vecteur car il est thread-safe et il n'y a ainsi pas de risque d'avoir des problèmes de concurrence. Ces modes peuvent être sauvegardés sous forme sérialisé. On peut ensuite les ouvrir afin de les utiliser ou de les modifier. Cette liberté a apporté son lot de problème, notamment si un changement du nombre de led intervient entre la création et l'ouverture du fichier, nous avons donc dû implémenter une vérification du nombre de led avant/après et afficher un message d'erreur le cas échéant.

```
public static ModePersonnalise getMode(String file) {
    try {
        File f = new File(file);
        if (f.exists() && !f.isDirectory()) {
            FileInputStream fi = new FileInputStream(f);
            ObjectInputStream oi = new ObjectInputStream(fi);
            ModePersonnalise mp = (ModePersonnalise) oi.readObject();
            fi.close();
            //vérification du nombre de led
            if(mp.l.size() != Config.getConfig().getNbLedTotal()){
                JOptionPane.showMessageDialog(null, "Le nombre de LEDs"+
                    "de votre mode personnalisé ne correspond pas à la"+
                    "configuration actuelle.", "Erreur", JOptionPane.INFORMATION_MESSAGE);
                mp = new ModePersonnalise(mp.name);
            }

            return mp;
        } else {
            throw new Exception("File doesn't exist");
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        return new ModePersonnalise();
    }
}
```

### Mise en zone de notification

Nous voulions que notre programme démarre en arrière-plan et soit accessible par la zone de notification de Windows. Pour ce faire, on commence par vérifier que la zone de notification est disponible sur l'hôte :

```
if (!SystemTray.isSupported()) {
    System.out.println("SystemTray is not supported");
    return;
}
```

On crée ensuite les éléments que le menu va proposer, une ligne pour ouvrir l'interface graphique et une ligne pour quitter le programme :



```
MenuItem messageItem = new MenuItem("Configuration");
messageItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new FrameMainWindow();
    }
});
menu.add(messageItem);

MenuItem closeItem = new MenuItem("Fermer");
closeItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
menu.add(closeItem);
```

On crée ensuite notre icône :

```
TrayIcon icon = new TrayIcon(image, "Ambi-like", menu);
icon.setImageAutoSize(true);
```

Et on l'ajoute finalement à notre zone de notification :

```
try {
    tray.add(icon);
} catch (AWTException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    final JPanel panel = new JPanel();
    JOptionPane.showMessageDialog(panel, "cannot add system tray icon, quit",
        "Error", JOptionPane.ERROR_MESSAGE);
    System.exit(1);
}
```

## Commande des leds

La gestion des leds a été considérée comme le point le plus épineux du projet. En effet, Java ne permet pas d'accéder à des composants matériels de manière native. Il est nécessaire de recourir à des wrappers comme Swig. La condition sine qua non de ce projet était la commande des leds grâce à une bibliothèque. Heureusement, la perle rare a été trouvée. Un individu a mis à disposition sur GitHub l'outil requis. La bibliothèque `rpi-ws-281x-java` a permis de faciliter grandement le travail.

Sur le Wiki se trouve la procédure pour l'installation de cette dépendance externe.

L'utilisation de la bibliothèque est très simple. Il suffit d'inclure le jar dans l'IDE de notre choix. L'initialisation du composant s'effectue comme la capture ci-dessous le montre.

```
//initialisation of the ledStrip
Ws281xLedStrip led = new Ws281xLedStrip(150, 18, 800000, 10, 255, 0, false, LedStripType.WS2811_STRIP_GRB, false);
led.setStrip(0, 0, 0);
led.render();
```

La gestion des leds est facile. L'accès aux leds s'effectue avec un index. Le bandeau implémente ce mécanisme avec TTL. Initialement, le TTL correspond à l'index de la led. Lors de chaque passage de led,

on décrémente le TTL. Lorsque le paquet arrive à zéro, la led sait que celui-ci lui est destiné. La javadoc de cette librairie est fournie dans le guide développeur.

## Délivrable

### Création de l'installateur et exécutable

Les JAR exécutables sont encapsulés dans un exécutable grâce à launch4j. Ceci nous permet de lancer notre programme au démarrage de Windows.

L'installateur, créé avec Inno Setup, permet de sélectionner l'emplacement d'installation du programme et certaines options supplémentaires, comme le démarrage automatique, géré grâce à une entrée dans la base de registre.

Il n'est pas nécessaire d'installer java avant d'installer notre programme, car l'installateur dispose d'une machine virtuelle java en version 11.0.3. La version 11 de java est la version minimum requise pour notre programme.

### Images du Raspberry pi

Afin de faciliter la mise en place pour les utilisateurs, nous avons créé une image pour le Raspberry pi, il suffit de copier cette dernière sur la carte sd et on a un système avec une IP fixe préconfigurée (192.168.100.100) et le programme qui se lance automatiquement au démarrage. Nous avons laissé les identifiants par défaut(pi/raspberry). Si une plus grande personnalisation est nécessaire, nous avons créé des images à chaque point de la configuration du Raspberry pi qui se trouve en annexe. De plus, s'il est nécessaire de changer l'adresse IP préconfigurée, une procédure détaillée est disponible dans le guide utilisateur.

## Conclusion

Globalement, ce projet a contribué à la pratique du langage de programmation java. Les plus-values sont nombreuses. Tout d'abord, la création d'une interface graphique en Swing a permis de découvrir des fonctionnalités intéressantes.

La gestion de la concurrence représente un gros morceau de ce projet. En effet, afin d'optimiser ce programme, le recours aux threads a été quasi obligatoire. Ce choix technologique a œuvré à la mise en pratique des connaissances acquises lors du cours de programmation concurrente.

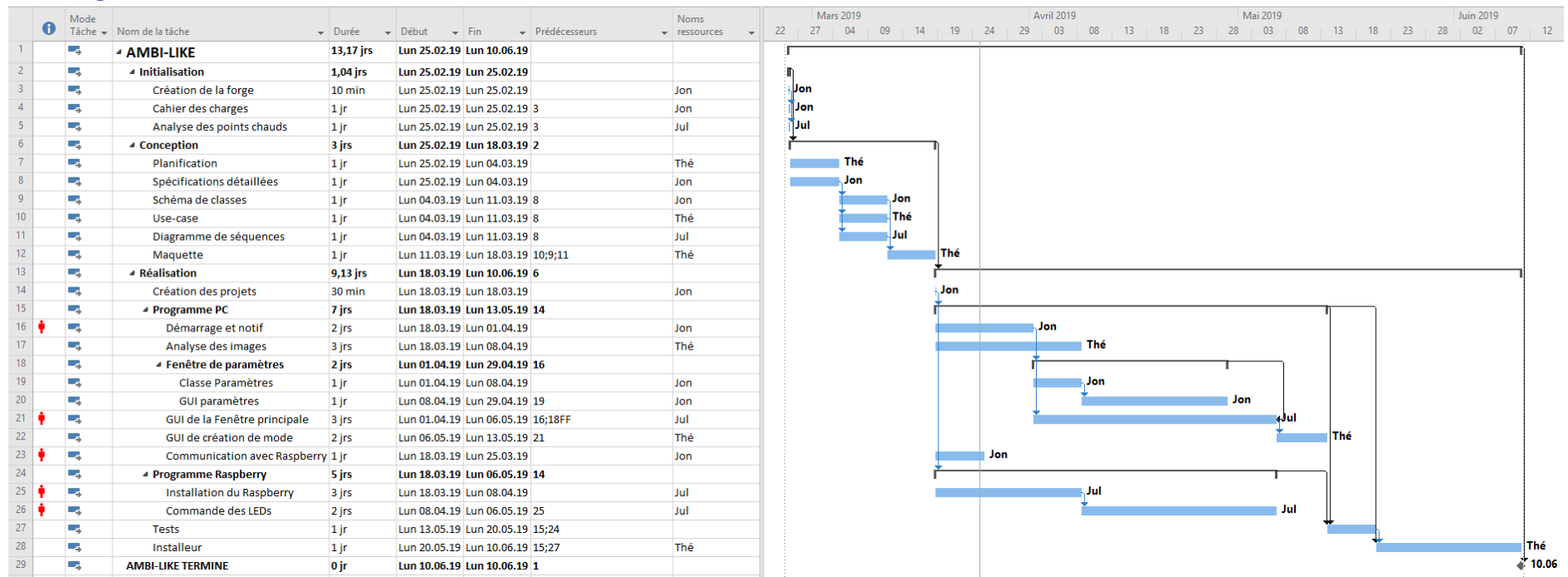
En outre, la communication entre le PC et le Raspberry Pi s'est fait avec la technologie RMI. Cela a permis au développeur en question de gagner de l'expérience.

Finalement, ce projet a permis de mettre en pratique les connaissances Java dans un cadre concret. Personnellement, nous sommes satisfaits du résultat et sommes tout ouïs à des améliorations futures.

### Annexe :

1. Planning
2. Préparation du Raspberry pi

## Planning



## Installation du Raspberry pi

Pour voir les liens vers les images, se rendre sur le wiki du git

### Préparation RaspberryPi

Last edited by [jonatan.baumgart](#) just now

### Mise en place du raspberry pi

cette page regroupe les information utiles à la mise en place du Raspberry pi, avec des images à chaque étape (carte 8go minimum).

### Préparation du systeme

Télécharger la dernière version de raspbian lite depuis [le site officiel](#).

Utiliser [BalenaEtcher](#) ou autre pour flasher la carte sd.

ajouter un fichier `ssh` vide sur la partition `boot`.

Brancher la raspberry pi à une source d'alimentation et au réseau. Trouver son adresse ip (fing, serveur dhcp ou autre)

se connecter en ssh avec le compte pi et le mdp raspberry et effectuer les commandes suivantes:

```
sudo raspi-config
```

dans Advanced option faire Expand Filesystem. ressortir en faisant finish et reboot.

[image](#)

### mise en place d'une adresse ip fixe (optionelle)

Editer `/etc/dhcpd.conf`

Décommenter les lignes

```
#interface eth0
#static ip_address=192.168.0.10/24
#static routers=192.168.0.1
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1
```

changer ces lignes pour correspondre à la config désiré.

[image](#)

### Installation de java

```
sudo apt-get update && sudo apt-get upgrade
```

```
sudo apt-get install openjdk-11-jdk
```

[image](#)

### Installation de Swig

```
sudo apt-get install swig
```

[image](#)

### Installation Graddle

```
apt-get install gradle
```

[image](#)

## Installation de Git

---

```
apt-get install git
```

[image](#)

## Installation RPI

---

```
git clone https://github.com/rpi-ws281x/rpi-ws281x-java.git
cd rpi-ws281x-java
```

executer le script suivant

```
chmod +x src/scripts/createNativeLib.sh src/scripts/createNativeLib.sh
```

```
./gradlew assemble -x signArchives
```

[image image avec le programme final](#)

