UNIVERSITÉ PARIS SUD | IUT DE CACHAN

INTERFACE POUR GESTION DE COMMANDE

CARREZ Nicolas – DESERT Louis

JUIN 2018

Vue d'ensemble (Intitulé)

Il s'agit de mettre en place une solution qui assiste un employé pour la gestion des commandes d'un magasin en ligne. L'employé doit avoir le choix entre travailler avec une tablette Android ou bien avec un bracelet équipé d'un écran LCD pour mettre en place des paniers correspondant aux commandes, paniers qui seront ensuite envoyés par colis. En pratique, les commandes des clients initialement sur PC sont dans un premier temps transférées dans le système embarqué à mettre en place ou dans la tablette Android. L'écran affiche une image des différents articles commandés (avec la quantité désirée) pour aider l'employé à remplir le panier. A chaque article placé dans le panier, l'utilisateur scanne un code barre, afin de vérifier que le produit est le bon, et de passer au produit suivant jusqu'à la fin de la commande.

Objectifs

- Créer une solution assistant les employés,
- Créer une interface affichant la liste des produits à ajouter dans le carton de commande,
- Configurer une connexion (Bluetooth) entre l'interface et une douchette afin de vérifier le contenu du colis.

Caractéristiques

L'application doit afficher les différents produits compris dans la commande.

Matériel :

Une plateforme sous Android (4.0.3/6.0.0/8.0.0)
Une carte STM32 +Module Bluetooth Slave
Une douchette Bluetooth

Deux ordinateur de développement (Android Studio / KeilµVision)

SOMMAIRE

Table des matières

Partie I : Générale

Vue d'ensemble (Intitulé) 2

Objectifs 2

Caractéristiques 2

Matériel : 2

Définition du fonctionnement 4

Design de l'UI 5

Partie II : Android

Introduction 7

Programmation de l'UI 7

Programmation du comportement 7

Gestion de la liste de course 8

Bluetooth 8

Partie III : STM32

Introduction 10

Programmation de l'UI (Se référer au code) 10

Programmation du comportement (Se référer au code) 10

Gestion dynamique des listes 11

Liaison UART 11

Définition du fonctionnement

Pour fonctionner, le système a besoin de données en entrées, et renverra des données à un serveur.

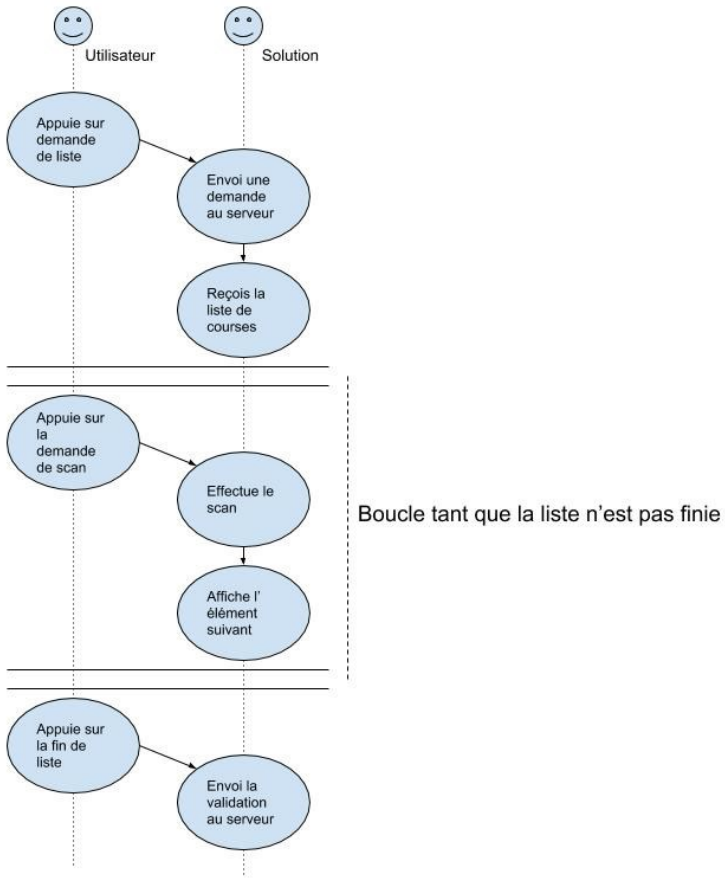
Entrées : La solution aura besoin d'avoir accès aux listes de courses des différents clients. Ces informations doivent être disponibles chez l'utilisateur de la solution. Ainsi, un protocole de communication doit être défini. Un protocole utilisant le Bluetooth semble efficace, pour permettre permettrait aux utilisateurs de ne pas avoir à se déplacer jusqu'au serveur.

(Exemple de protocole : [*numéro de commande*][*taille de la commande*] +
[*nom de l'article*][*code barre de l'article*][*informations sur l'article*]…)

Sorties : Il faut signaler au serveur que la commande a bien été traitée. Avant de "demander" une nouvelle commande au serveur, solution envoie un signal au serveur.

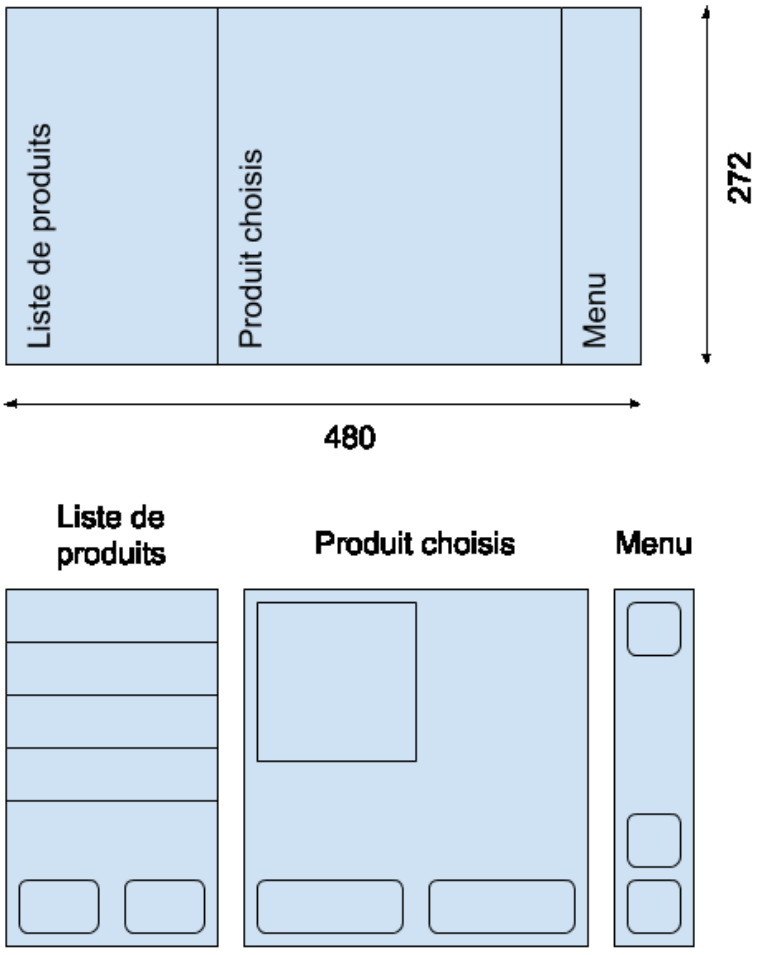
(Exemple de protocole : [*numéro de commande*][*statut de la commande*])

Un scénario d'utilisation a été créé afin de présenter le fonctionnement du système :

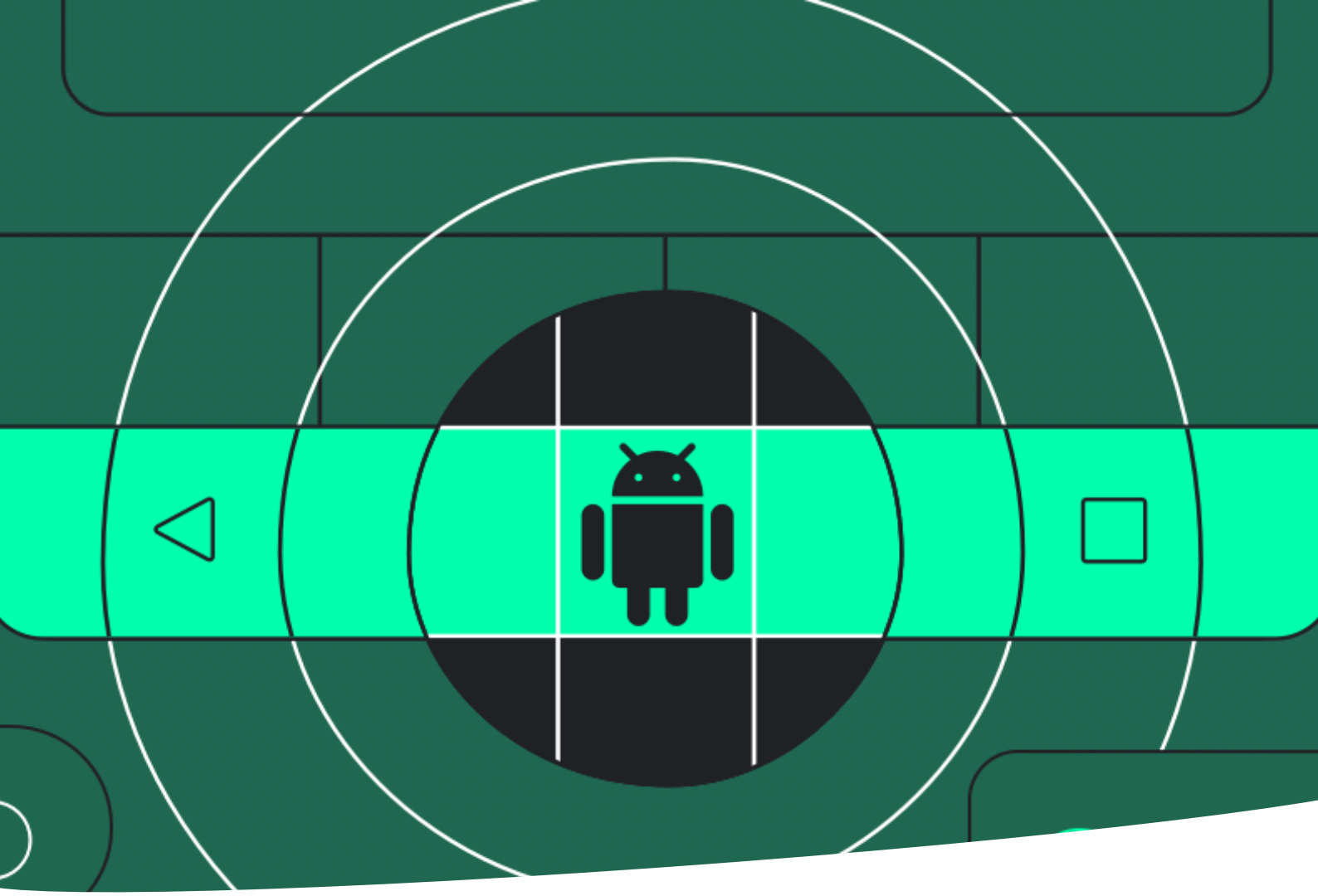


Design de l'UI

L'Interface Utilisateur a été divisée en trois parties, une partie liste où l'on retrouve la liste des produits de la commande que l'on peut faire défiler, une partie produit où sont regroupés le nom, la description, l'image, la quantité restante et l'emplacement du produit ainsi que les deux boutons scan et sold-out, enfin on retrouve une dernière partie constituée des boutons finalisation de commandes, options et bluetooth.



Quelques différences peuvent apparaitre entre la version Android et STM32 mais la disposition reste la même.



PARTIE I :
ANDROID

Introduction

Une interface est design pour Android grâce à Android Studio (tests sous la version 6.0 :Marshmallow).

Utiliser Android Studio a de nombreux avantages, en particulier l'abondance de bibliothèques et la possibilité de designer l'UI directement qui facilite la programmation et le design.

Programmation de l'UI

Sous Android Studio, le design d'UI peut se faire avec l'outil de design intégré. Ce dernier génère automatiquement un fichier .xml correspondant à l'UI designée.

Programmation du comportement

Le comportement est programmé à l'aide de onClickListener(), sans boucle while(). Les boutons utilisés sont :

```
final Button button1 = findViewById(R.id.button1); //Boutons de liste
final Button button2 = findViewById(R.id.button2);
final Button button3 = findViewById(R.id.button3);
final Button button4 = findViewById(R.id.button4);
```

```
Button buttonRight = findViewById(R.id.buttonRight); //Boutons défile liste
Button buttonLeft = findViewById(R.id.buttonLeft);
```

```
Button buttonScan = findViewById(R.id.buttonScan); //Boutons Vente
Button buttonSoldOut = findViewById(R.id.buttonSoldOut);
```

```
Button buttonGetList = findViewById(R.id.buttonGetList); //Boutons Autres
Button buttonBluetooth = findViewById(R.id.buttonBluetooth);
Button buttonFinalise = findViewById(R.id.buttonFinalise);
```

Voici par exemple le onClickListener de buttonLeft pour dérouler la liste jusqu'à 4 produits suivants.

```
buttonLeft.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (page >= 1) {
            page--;

            printProductButton(page*4,button1);
            printProductButton((page*4)+1,button2);
            printProductButton((page*4)+2,button3);
            printProductButton((page*4)+3,button4);
        }
    }
});
```

Gestion de la liste de course

Pour la liste de course on utilise la bibliothèque ArrayList de Android studio, qui nous permet d'avoir un tableau dynamique sans taille fixe à la création, auquel on peut ajouter et retirer des éléments à notre guise.

```
// Product list declaration
private ArrayList<Product> product = new ArrayList<>();
```

Pour les produits une classe « Product » a été créée qui regroupe toutes les informations dont nous avons besoin sur chaque produit : Nom, Description, Quantité requise, Emplacement dans le magasin, son code barre, s'il est encore en stock ou non et enfin une image. A noter que pour l'instant l'image redirige sur une image présente dans le projet de base, l'objectif final serait de pouvoir rajouter les images en même temps que l'on rajoute des produits à la liste.

```
//Constructeur de la classe Product
Product(int quantity, int barcode, int imgId, String name, String
description, String place) {
    this.quantity = quantity;
    this.barcode = barcode;
    this.imgId = imgId;
    this.name = name;
    this.description = description;
    this.place = place;
    this.soldOut = false;
}
```

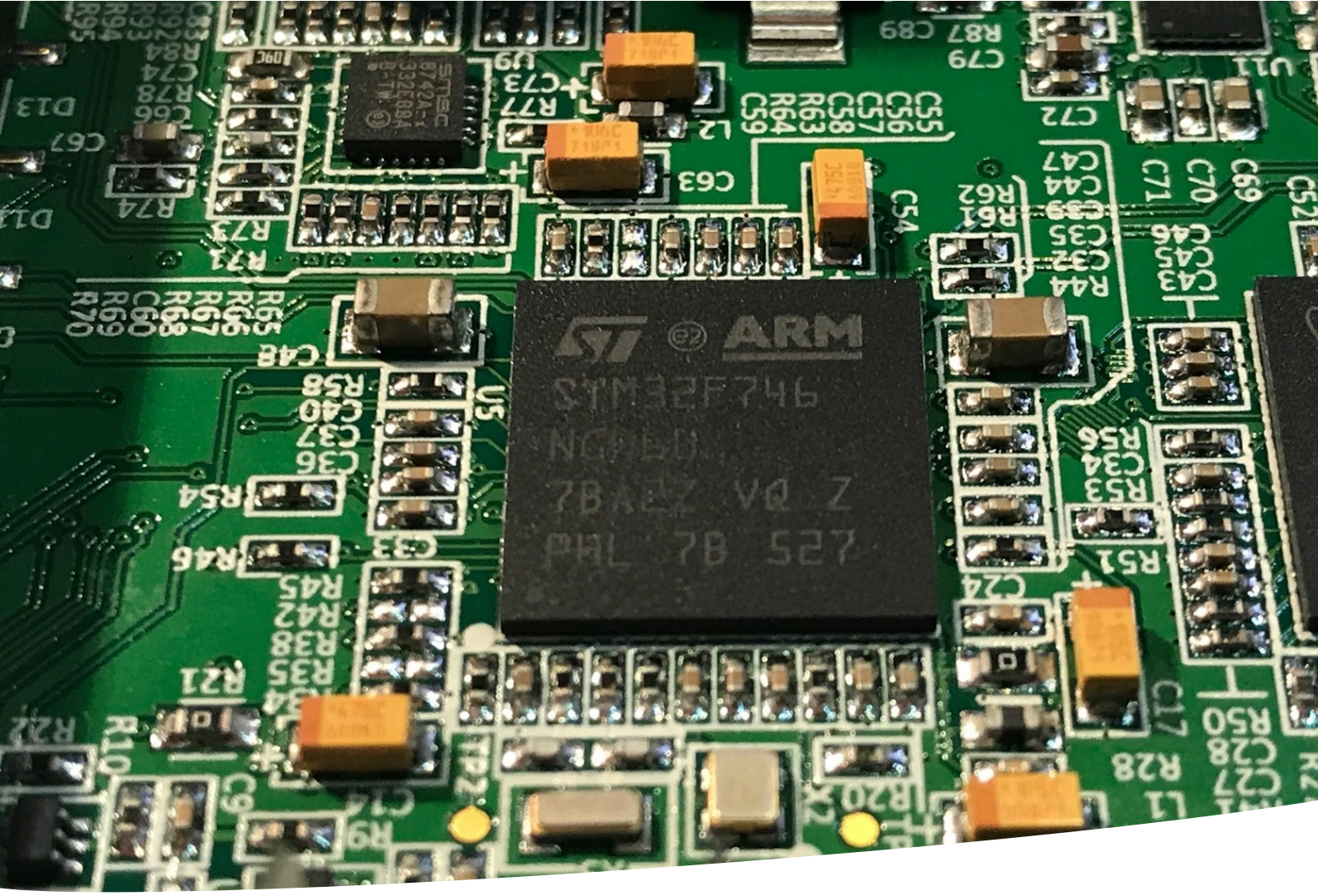
La fonction « ReadList » permet d'aller chercher un fichier command_list.txt dans la mémoire du téléphone et de le déchiffrer pour avoir les informations sur chaque produit demandé. Un produit est écrit de la manière suivante dans le fichier texte :

<product>
name=
quantity=
description=
place=
barcode=
</product>

A noter que pour l'instant même si le décryptage de fichier fonctionne il y a un problème d'accès aux fichiers du téléphone.

Bluetooth

Pour scanner les code-barres, on prévoit d'utiliser une douchette Bluetooth de Wasp Barcode. Plusieurs fonctions ont été écrites pour utiliser cette douchette, cependant la connexion refusant de s'établir, elles ont été mises de côté pour le moment.



PARTIE II :
STM32 - DISCO

Introduction

Afin de mener à bien le projet sur la STM32, il a fallu effectuer un choix technique au niveau du logiciel de développement.

Dans le cadre du projet, l'environnement de développement intégré KeilµVision semblait être la solution la plus correct. En effet, KeilµVision dispose de nombreux exemples. De plus, la communauté en ligne n'est, certes, pas très développée, mais est présente tout de même.

(Avant d'aller plus loin, merci de lire la partie globale)

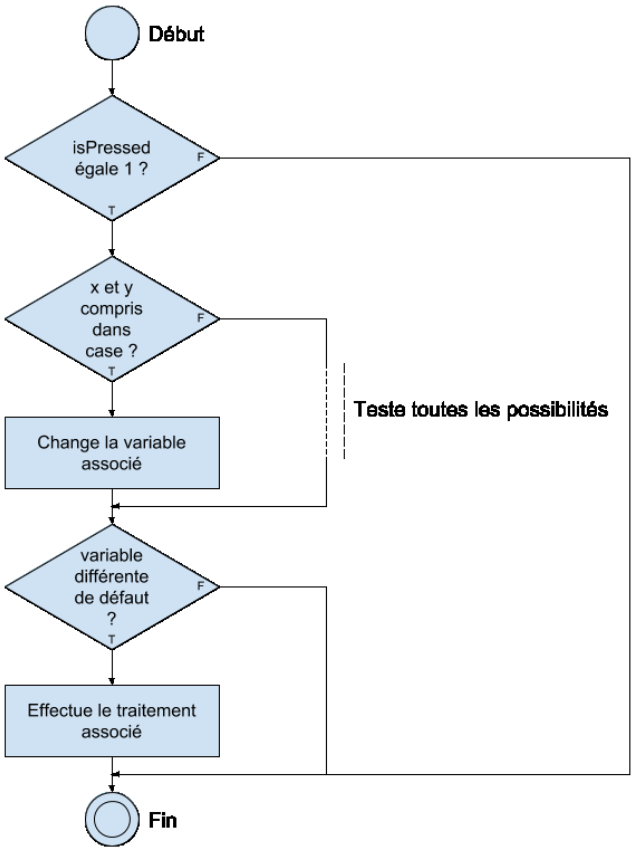
Programmation de l'UI (Se référer au code)

Pour pouvoir réaliser le design de la UI sur la STM, plusieurs fonctions sont utilisées.

```
GUI_SetColor(GUI_BLUE);                 //Définit la couleur d'écriture sur la UI.
GUI_SetFont(&GUI_Font16_1);             //Définit la taille de police d'écriture.
GUI_SetPenSize(1);                      //Définit la taille de dessin des formes.
GUI_DrawBitmap(&bm{nomfichier}, x, y);  //Dessine une BitMap depuis la position {x, y}.
(Attention les images sont considérées comme des objets et doivent donc être « dessinées » en permanence)
GUI_DrawLine(x1, y1, x2, y2);           //Dessine une ligne de la position {x1, y1} à la position {x2, y2}.
GUI_DrawRect(x1, y1, x2, y2) ;          //Dessine un rectangle de la position {x1, y1} à la position {x2, y2}.
GUI_DispStringAt(const char*, x, y);    //Ecrit une chaine de caractère sur la UI à la position {x, y}.
```

Programmation du comportement (Se référer au code)

Les interactions avec l'utilisateur se faisant avec le LCD, les fonctions de comportements doivent être dans une boucle *while(1)*, pour être exécutées en permanence. Dans ce projet, la partie comportement est gérée par la fonction *doScan(x ,y, isPressed)*;. Son algorithme :



Gestion dynamique des listes

L'objectif étant d'avoir un programme qui peut actualiser sa liste de courses, une liste dynamique est nécessaire. Pour cela, une structure existe : la structure *product* :

```
40 struct product {
41   char name[20];
42   const GUI_BITMAP *image;
43   int  quantity;
44   char barcode[20];
45   char location[20];
46   int  status;
47 };
48
```

La structure contient le nom du produit, son image, la quantité demandée, ainsi que, le code barre et la position dans le hangar du produit.

Enfin, l'entier *status* permet de définir le statut du produit
(Exemple : 0 = En traitement, 1 = Ok, -1 = Sold Out, etc…)

Le compilateur de Keil étant un compilateur C90, l'utilisation de la structure est quelque peu différente :

1. Créer un pointeur vers la structure.
2. Définir la taille de la structure.
3. Allouer l'espace mémoire à la structure.
4. Initialiser les valeurs de la structure.

```
 59  struct product *productList;
 60  int listSize = 6;
251   productList = (struct product*) malloc(listSize * sizeof(struct product));
252   for(i=0;i<listSize;i++) {
253     sprintf(name,"Product %d",i+1);
254     strcopy(name,(productList+i)->name);     //Init Product Name
255     (productList+i)->quantity = i+1;         //Init Product Quantity
256     sprintf(barc,"0112233445");
257     strcopy(barc,(productList+i)->barcode); //Init Product Barcode
258     sprintf(loca,"%d°Allee",i+1);
259     strcopy(loca,(productList+i)->location);//Init Product Location
260     (productList+i)->status = 0;             //Init Product Status
261   }
```

Liaison UART

Une liaison UART est nécessaire pour communiquer avec le module Bluetooth HC-05.

Une initialisation de la liaison est nécessaire (Cf UART_init()). Une fois initialisée, cette connexion permet de communiquer librement avec le module Bluetooth.

Il faut ensuite paramétrer le module à l'aide de commande.

*(**Attention** Toutes les commandes et les retours se terminent par les caractères '\r' et '\n')*

COMMANDE	RETOUR	UTILITE
AT	OK	Ping
AT+ADDR ?	+ADDR :(valeur)	Renvoie l'adresse Bluetooth du module
AT+ROLE ?	+ROLE=(valeur) OK	Renvoie le mode du module 0 – Slave 1 – Master

(Pour plus d'informations se référer aux liens dans le dossier *DOCUMENTATION*)