

TP : IoRMatrix

La Matrice Connectée

Remarque : La partie Java de ce TP ne doit pas être programmée sous un IDE. Il faut créer un nouveau dossier au nom de **iormatrix** et créer les fichiers sources Java à l'intérieur. Pour la compilation, il faut créer et compiler uniquement le fichier Main.java en utilisant la commande :

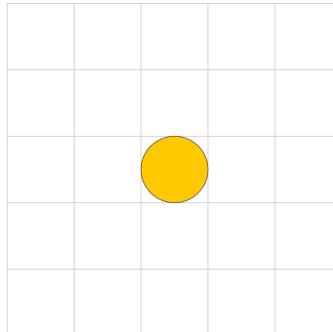
`javac Main.java`

Pour exécuter :

`java Main (sans l'extension)`

Épisode 1

Partie 1 : La matrice (Java) – Local



Cette partie permet de préparer la matrice à commander à distance. Il s'agit d'une grille de 5x5 contenant un disque jaune. Le disque se déplace d'une case à une autre en fonction des messages suivants :

- Le message **a** : se déplacer vers la case à droite
- Le message **b** : se déplacer vers la case à gauche
- Le message **c** : se déplacer vers la case en bas
- Le message **d** : se déplacer vers la case en haut
- Le message **afficher** : permet d'afficher le disque
- Le message **cacher** : permet de cacher le disque
- Le message **init** : permet ramener le disque au centre de la matrice

Pour récupérer le code Java de la matrice, visitez le lien suivant :

<http://labsticc.univ-brest.fr/~bounceur/cours/android/tps/iormatrix/java/MyMatrix.java>

Pour tester :

<http://labsticc.univ-brest.fr/~bounceur/cours/android/tps/iormatrix/java/Main.java>

Modifiez le code de sorte à obtenir une matrice 8x8, avec des cases de taille de 60x60. Remplacer les commandes a, b, c et d par d, g, b, h (d pour droite, g pour gauche, b pour bas, h pour haut).

Faites de sorte que la commande **exit** quitte définitivement l'application en fermant la fenêtre de la matrice (méthode dispose).

Faites de sorte que si le disque dépasse une des bornes de la matrice elle apparaît de l'autre côté.

Partie 2 : La matrice (Java) – Réseau (Socket)

Dans cette partie la matrice devient un serveur. Nous allons créer un client (le disque jaune) qui sera commandé à distance via le réseau. Pour la simplicité, nous supposons dans cet exemple qu'un seul client (disque) peut se connecter à la fois.

Le code Java (Main_Serveur.java) de la matrice Serveur peut être récupéré du lien suivant :

http://labsticc.univ-brest.fr/~bounceur/cours/android/tps/iormatrix/java/Main_Serveur.java

Le code Java (Main_Client.java) du client ‘disque’ peut être récupéré ici :

http://labsticc.univ-brest.fr/~bounceur/cours/android/tps/iormatrix/java/Main_Client.java

Testez ... (Lancez deux terminaux séparément et lancez ensuite le serveur en premier dans un terminal puis le client dans l'autre).

(Voir la démo youtube : <https://www.youtube.com/watch?v=iYB5JxARwtg>)

Partie 3 : La commande Android – Réseau (Socket)

Dans cette partie, nous allons remplacer le client (Main_Client.java) de la partie 2 précédente par un autre client permettant de commander la matrice à partir d'une application Android (voir l'image ci-dessous). Pour ce faire, nous allons utiliser les Sockets. Inspirez-vous du code Java pour compléter cette partie. Pensez surtout aux permissions et à l'utilisation des Threads/AsyncTask.



Pour dessiner les flèches, il faut utiliser Android Studio : à partir d'un clic droit sur le dossier res → new → Image Asset (Icon Type : Action Bar and Tab Icons). Notez que le localhost (127.0.0.1) sous Android doit être spécifié par : 10.0.2.2

Écrivez l'application Android de la commande !

Épisode 2

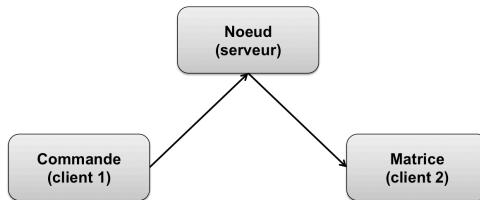
Partie 1 : Architecture Nœud (Java) – Local

Que pensez-vous de l'architecture précédente, si on suppose que l'utilisateur final qui doit voir la matrice ne se trouve pas au niveau du serveur ?

L'architecture précédente est sous la forme suivante :



Pour pouvoir voir la matrice à partir d'un endroit différent de celui du serveur, nous proposons l'architecture suivante :



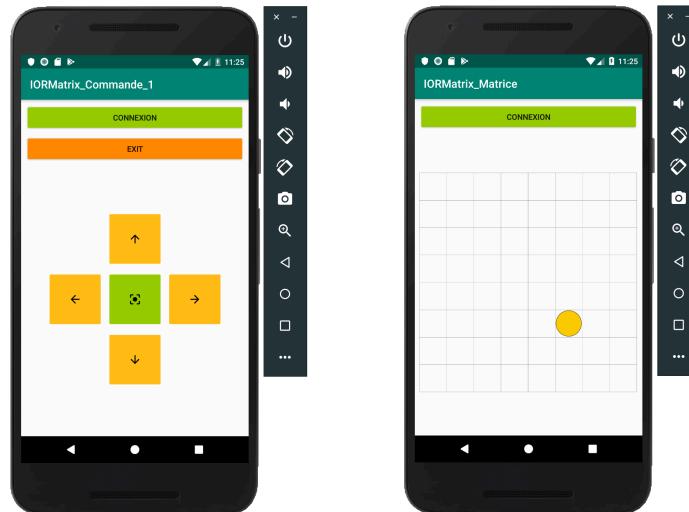
Modifiez le code du serveur (Matrice) pour le transformer en un client et ajoute une classe Noeud.java permettant de jouer le rôle du serveur. Ce dernier servira d'un intermédiaire entre la commande et la matrice. Chaque message envoyé par la commande (client 1) doit être reçu par le Noeud (serveur) qui le transfèrera ensuite (i.e., le même message) à la matrice (client 2). Dans le cas du code donné ci-dessus, il faut lancer le client 1 en premier ensuite le client 2.

Partie 2 : Commande Android / Matrice Android (Noeud Java)

Dans cette partie, nous allons garder le même nœud de la partie 1 précédente. Nous allons utiliser aussi la même commande Android de la partie 3 (Épisode 1) précédente. Ce qui est demandé est de reprogrammer la matrice sous Android. Reprenez le code du client 2 précédent pour l'adapter à une application Android. Le code source de la matrice Android (objet View) peut être téléchargé à partir du lien suivant :

<http://labsticc.univ-brest.fr/~bounceur/cours/android/tps/iormatrix/android/MyMatrix.java>

Testez (à noter que la connexion de la commande doit être lancée avant celle de la matrice).



Partie 3 : Commande Android / Matrice Android (MQTT)

Le code précédent permet de connecter 2 Smartphones indépendants uniquement si le Nœud Java est accessible depuis l'extérieur de votre ordinateur. Dans cette partie, nous proposons d'utiliser le protocole MQTT. Réalisez la même application en utilisant le protocole MQTT.

Désormais, il est possible d'utiliser deux smartphones totalement indépendants. Testez avec votre binôme (un binôme installe la commande et l'autre la matrice) ...

Partie 4 : Commande Android / Matrice Android (NodeJS)

Dans cette partie, nous allons remplacer le nœud Java par un nœud NodeJS. Le code de ce dernier est donné comme suit (le **xx** est un numéro qui sera donné pour chaque étudiant) :

```

var app = require('express')();
var server = require('http').createServer(app);
var io = require('socket.io').listen(server);
io.sockets.on('connection', function (socket) {
    socket.on('a', function (id, message) {
        io.emit('b', id, message);
    });
});

```

```
});  
});  
server.listen(8080);
```

Ce nœud permet s'il reçoit d'un client un id et un message de type **a** de les envoyer en broadcast (à tous les clients connectés) en type **b**. Il est accessible via l'URL : <http://iotserveur.univ-brest.fr:8080>

Ecrivez les codes Android de la commande et de la matrice qui communiquent par l'intermédiaire de ce serveur NodeJS.

Partie 4 : Commande Android / Matrice JS (NodeJS)

Dans cette partie, nous allons afficher la matrice dans une page web à l'aide d'un code JavaScript. En effet, il est possible de voir la matrice à la fois sur un Smartphone ainsi que sur un navigateur web. Le lien suivant permet de récupérer le code permettant de dessiner la matrice IoRMatrix en Javascript et de la commander en Javascript via un serveur NodeJS. Modifiez le code pour afficher une matrice 8x8 et commander le point rouge avec la commande Android de la partie 2 précédente.

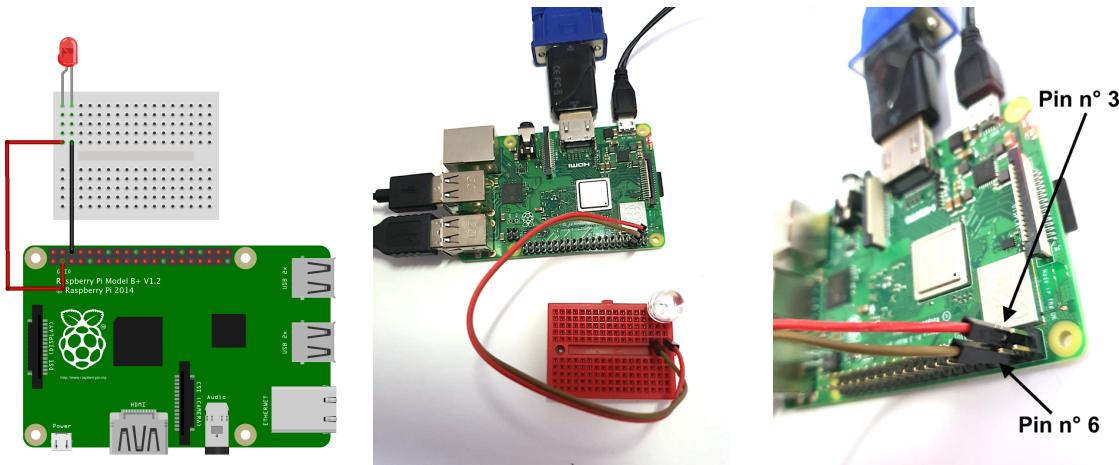
Code source :

http://labsticc.univ-brest.fr/-bounceur/cours/android/tps/iormatrix/nodejs/source_2_4.zip

Episode 3

Partie 1 : Commande d'une LED par le système (Raspberry PI)

Dans cette partie nous allons contrôler une LED directement à partir du système d'exploitation d'une carte Raspberry. Branchez le pin + d'une LED sur le GPO8 (pin 3 de la ligne 2, colonne 1) et le pin – sur le Ground (pin 6 de la ligne 3, colonne 2) d'une carte Raspberry comme le montre la figure suivante (les numéros représentent l'ordre du pin à partir du haut, cf. le cours) :

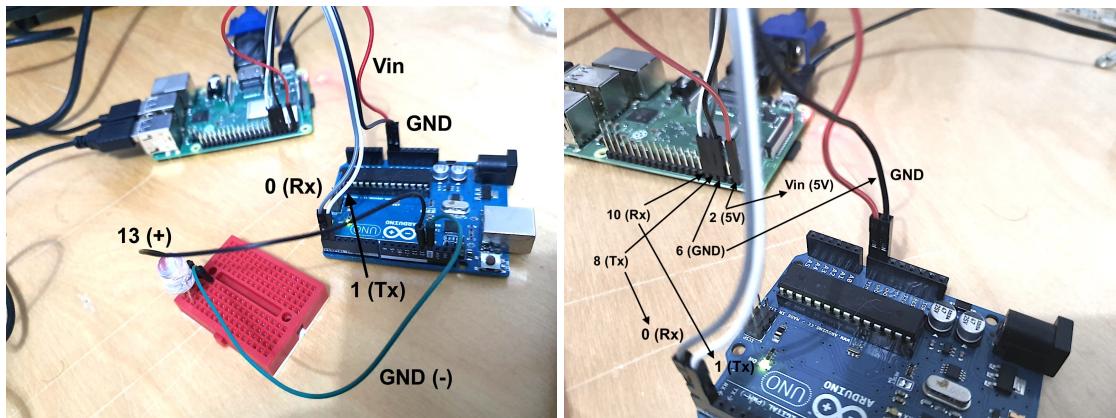
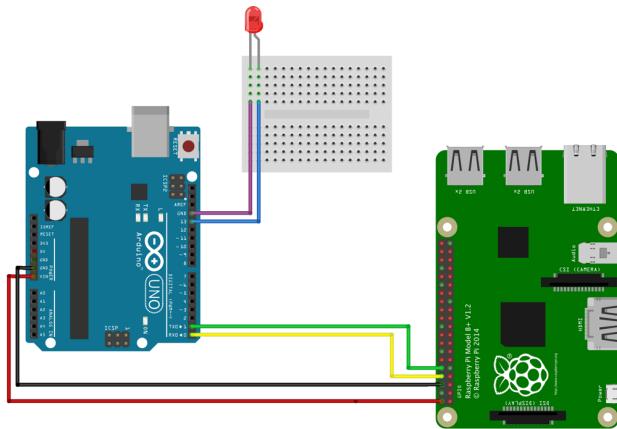


Cette LED peut être allumée ou éteinte directement à partir du système Raspberry en utilisant les commandes suivantes :

```
gpio mode 8 out  
gpio write 8 1  
gpio write 8 0
```

Partie 2 : Commande d'une LED via le port série (Raspberry PI/Arduino)

Dans cette partie, nous allons utiliser la communication série afin d'envoyer des commandes d'une carte Raspberry vers une carte Arduino. Cette dernière allumera et éteindra une LED branchée en fonction des commandes reçues. Pour ce faire, nous allons utiliser le minicom (à installer) de Raspberry. Une fois configuré (port /dev/ttymA0 et fréquence de 9600 bauds), nous allons contrôler la LED d'une Arduino programmée comme dans la partie 1 de l'épisode 3 et la brancher sur la Raspberry comme suit (l'Arduino ne doit pas être alimentée, c'est la Raspberry qui l'alimente via le Vin) :



Avant de reprogrammer la carte Arduino, il faut tout d'abord enlever les câbles branchés sur les pins 0 (Rx) et 1 (Tx). Une fois programmée, il faut remettre ces deux câbles. Ensuite, à partir du minicom de la Raspberry, appuyez sur les touches **a** et **b** pour allumer et éteindre la LED.

Partie 3 : Commande d'une LED par le port série/NodeJS (Raspberry PI/Arduino)

Cette fois-ci, nous allons écrire le code NodeJS permettant de contrôler la LED de l'Arduino via une communication série. Le code suivant permet d'envoyer le message **a** sur un port série.

```
var SerialPort = require('serialport');
var port = new SerialPort('/dev/ttyAMA0', {baudRate: 9600});
port.on("open", () => {
    console.log('serial port open');
    port.write('a', (err) => {
        if(err) {
            console.log(err);
        }
        else {
            console.log('message écrit');
        }
    });
});
```

L'exécution de ce code permettra d'allumer la LED. Changer le caractère **a** par **b** et ré-exécuter le code. La LED doit s'éteindre. Le code suivant permet de faire clignoter la LED. On utilisera la fonction `setInterval` de NodeJS permettant d'exécuter une fonction périodiquement.

```
var SerialPort = require('serialport');
var port = new SerialPort('/dev/ttyAMA0', {baudRate: 9600});
var c = 'a';
port.on("open", () => {
    console.log('Port série ouvert');
    setInterval(mafonction, 1000)
});
```

```
function mafonction() {
    port.write(c, (err) => {
        if(err) {console.log(err);}
        else {
            if(c=='a') c='b';
            else c='a';
        }
    });
}
```

Partie 4 : Commande d'une LED par le port série/Serveur NodeJS (Raspberry PI/Arduino)

Dans ce qui suit, nous allons créer un serveur NodeJS dans une Raspberry permettant de recevoir des messages (commandes) via le réseau afin d'allumer/éteindre une LED Arduino connectée en série à la Raspberry. Le rôle du serveur est tout simplement d'envoyer sur le port série les messages de type 'a' reçus du réseau :

```
var SerialPort = require('serialport');
var port = new SerialPort('/dev/ttyAMA0', {baudRate: 9600});
var app = require('express')();
var server = require('http').createServer(app);
var io = require('socket.io').listen(server);

var SerialPort = require('serialport');
var port = new SerialPort('/dev/ttyAMA0', {baudRate: 9600});
port.on("open", () => {
io.sockets.on('connection', function (socket) {
    socket.on('a', function (id, message) {
        if(id==xx) {
            port.write(message, (err) => {
                if(err) {
                    console.log(err);
                }
                else {
                    console.log('message écrit');
                }
            });
        }
    });
});
server.listen(8080);
}
```

Ecrivez le code Android permettant de commander la LED de l'Arduino via le serveur NodeJS de la Raspberry. Pour cette tâche, il faut connaître l'adresse IP de la carte Raspberry.

Partie 5 : Commande d'une LED par le port série/Clients/Serveur NodeJS (Raspberry PI/Arduino)

Dans cette partie, on suppose que nous avons déjà un serveur en ligne (<http://iotserver.univ-brest.fr:8080>) avec le code suivant :

```
var app = require('express')();
var server = require('http').createServer(app);
var io = require('socket.io').listen(server);
io.sockets.on('connection', function (socket) {
    socket.emit('b', 'Connexion établie avec succès');
    socket.on('a', function (id, message) {
        io.emit('b', id, message);
    });
});
server.listen(8080);
```

Ce serveur permet à chaque réception d'un message de type **a** de l'envoyer en broadcast en un message de type **b**.

On va écrire deux clients NodeJS. Un client PC et un client Raspberry. Le client PC va envoyer chaque seconde les message **u** et **v** (en type **a**) de sorte à ce que le client de la Raspberry fasse clignoter la LED Arduino. En d'autres termes, le client de la Raspberry allume la LED s'il reçoit le message **u** et l'éteint s'il reçoit le message **v**.

Code du client PC :

```
var socket = require('socket.io-client')('http://iotserver.univ-brest.fr:8080');
socket.on('b', function (id, message) {
    if(id == xx) console.log(message);
});
setInterval(maf, 1000);
var c = 'u';
function maf() {
    socket.emit('a', xx, c);
    if(c=='u')
        c = 'v';
    else
        c = 'u';
}
```

Le client Raspberry :

```
var socket = require('socket.io-client')('http://iotserver.univ-brest.fr:8080');
var SerialPort = require('serialport');
var port = new SerialPort('/dev/ttyAMA0', {baudRate: 9600});
port.on("open", () => {
    console.log('Port série ouvert');
    socket.on('b', function (id, message) {
        if(id==xx) {
            port.write(message, (err) => {
                if(err) {
                    console.log(err);
                }
            });
        }
    });
});
```

Partie 6 : Commande d'une LED par le port série/Client/Serveur NodeJS (Raspberry PI/Arduino) et Client Android

Dans cette partie, nous allons remplacer le client PC de la partie 5 précédente par une application Android (client) avec 2 boutons pour allumer/éteindre la LED Arduino de la Raspberry.

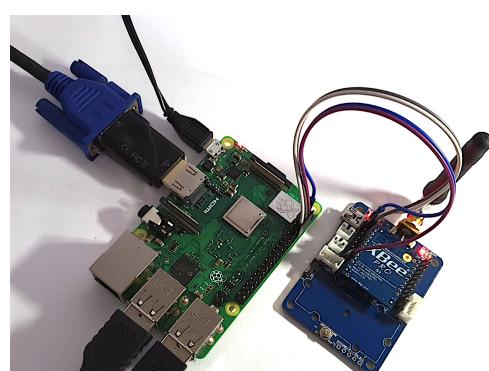
Partie 7 : Commande d'une Matrice à LEDs par le port série/Client/Serveur NodeJS (Raspberry PI/Arduino) et Client Android

Reprendre la commande Android des parties précédentes et le code Arduino de la matrice à LEDs pour bouger le point à distance.

Partie 8 : La sans-fil !



La Matrice



La Raspberry

Episode 4 → Le JEU

Dans cette partie, il faut programmer un jeu avec la matrice. Commencer par le programmer avec la matrice Android (**Partie 1 : utilisation de couleurs différentes pour les points**), ensuite faites-le avec la matrice à LEDs (**Partie 2 : utilisation d'une seule couleur pour les points**). Ce jeu se joue à 2. Son idée est d'afficher sur la matrice 3 points. Un point sera contrôlé par le premier joueur, un autre point sera contrôlé par le deuxième joueur. Le troisième point sera considéré comme point cible. Chaque joueur doit l'attendre en essayant de se déplacer sur sa case (même ligne, même colonne). Le premier joueur ayant atteint cet objectif gagnera un point, ensuite le point cible change de place de manière aléatoire et les deux joueurs continuent à le suivre jusqu'à atteindre un certain niveau de points (par exemple 100) pour gagner. On suppose que dans ce jeu, on n'affiche pas le score jusqu'à ce que un des joueurs gagne. (**Partie 3 : version 2 → limite de temps et autres propositions**).

Vos propositions sont les bienvenues
Bon TP !
Ahcène Bounceur