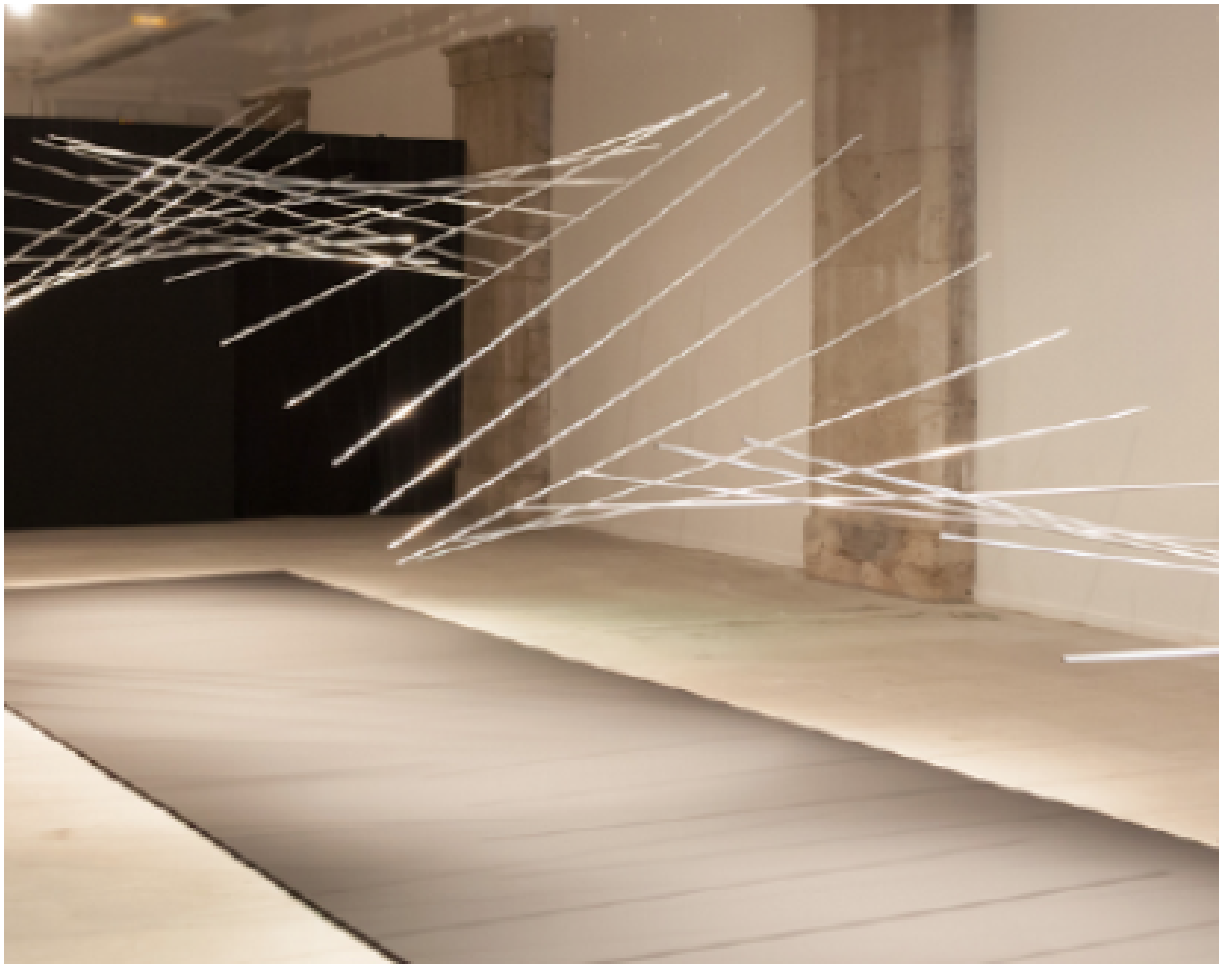
Rapport Projet Mobile
Partie Personnel IR2
DUBERNET-GLIKSTEN MATEO



AVIGNON CAMPUS DES SCIENCES ET TECHNIQUES

Sommaire

Présentation de ma partie

Le mobile doit pouvoir être commandé à distance depuis une interface de commande qui permet de choisir la vitesse des moteurs et les motifs pour le mouvement des barres du mobile. Il faudra également créer une simulation en 3D du mobile.

Ma partie dans ce projet consiste donc à la création d'une IHM, qui permet la visualisation et la commande du mobile sur une carte raspberry Pi. Cette IHM pourra être local ou distante. La version locale sera disponible en connectant un écran à la carte Raspberry Pi, quant à la version distante elle se fera au travers d'une communication sans-fil en Wi-fi. L'interface de commande sera réalisé en langage javascript avec la bibliothèque P5.js et communiquera avec un serveur web installé sur la carte raspberry Pi, la communication entre le serveur et l'interface sera une communication serveur/client en Websocket où le serveur sera bien évidemment le serveur présent sur la carte Raspberry Pi et le client sera l'interface de commande qui sera lancé depuis un navigateur internet, la communication étant réalisé en Websocket cela permettra un communication en temps réel et un échange direct entre le client et le serveur. Pour faire fonctionner cette communication correctement il faut que lors de la mise en route de la carte raspberry Pi le serveur se lance automatiquement. Pour réaliser la communication il faudra que je réalise une classe Communication en java sur processing et qui sera donc associé au serveur Web présent sur la carte Raspberry Pi.

Il me faudra aussi créer une visualisation des barres du mobile en 3D et les motifs que ces barres affichent, l'affichage sera réalisé sur l'interface de commande et sera donc programmé en javascript avec la bibliothèque P5.js également, cet affichage réalisera une simulation 3D du mobile en faisant bouger les barres 3D afficher sur l'interface en fonction du motif choisie pour le mobile, pour cela il faudra que je créer en javascript une classe Barre qui gérera l'affichage des barres sur l'interface de commande et également une classe Motif qui gérera les motifs qui seront réalisé par le mobile 3D et réel. Il faudra également que lors de la mise en marche du mobile que toutes les barres soit au point le plus bas et que l'on pourra à partir de là proposer des motifs pour le mouvement des barres.

Voilà pour ce qui est des différentes tâches à effectuer pour la partie personnelle de chaque membre du groupe. Nous allons donc maintenant passer à la présentation de l'organisation du projet où nous verrons les

différents diagrammes créer et également le planning que nous avons créé au tout début du projet.

Dans ma présentation je vais expliquer comment je m'y suis pris pour réaliser les objectifs qui m'étaient demandé, je présenterais les différentes étapes que j'ai suivies pour pouvoir atteindre mes objectifs, à travers cette présentation je dirais quel auront été les difficultés que j'ai rencontré durant mon projet et j'expliquerais les solution que j'ai trouvée pour faire face aux difficultés que j'ai rencontrés. Durant ma présentation j'expliquerais également le fonctionnement de toutes les tâches que j'ai réalisé tout en détaillant le lien entre ces différentes tâches. Je terminerais mon rapport en disant ce que travailler sur ce projet m'a apporté en termes de connaissances et de façon de travailler.

Mes objectifs

Pour accomplir les différents objectifs de ma partie j'ai dû passer par plusieurs étapes différentes. En effet pour réaliser correctement mes objectifs il m'a fallu avancer étapes par étapes pour pouvoir comprendre ou parce que j'avais besoin de ces étapes pour pouvoir réaliser mon travail. Pour pouvoir créer l'IHM il m'a fallu d'abord configurer une raspberry PI pour pouvoir la démarrer et installer le serveur web dessus. Il fallait également créer le serveur web en java sur processing pour pouvoir récupérer les informations envoyées par le client et ainsi les transmettre à la carte arduino. La configuration de la carte raspberry a été la priorité car sans la raspberry je ne pouvais pas faire la communication serveur client comme il me l'a été demandé dans le cahier des charges. En effet comme dit précédemment l'IHM se fera sur raspberry PI donc il m'a évidemment fallu d'abord configurer une raspberry. Ensuite pour réaliser cette IHM il m'a fallu créer un serveur et un client, le serveur est programmé en java sur processing et j'ai programmé le client en javascript, le client est une interface web qui commande à distance ou en locale le mobile.

Pour pouvoir programmer le serveur et le client il m'a avant tout fallu me former en java et en javascript car je n'avais aucune connaissance dans ces deux langages informatiques. Pour ce qui est de la communications serveur client il m'a également fallu faire des recherches sur le fonctionnements d'une en websocket, le cahier des charges stipulé qu'il fallait faire la communication entre le serveur et le client en websocket, j'ai donc dû comprendre comment fonctionner ce type de communication et pourquoi il m'a été demandé de

réaliser une communication en websocket. Voilà ce que j'ai dû faire pour pouvoir réaliser l'IHM.

Pour ce qui est de la partie simulation 3D des barres du mobile il m'a aussi fallu passer par différentes étapes, tout d'abord les barres étant afficher sur l'interface de commande, elles ont été programmé en javascript cependant il m'a fallu utiliser une bibliothèque javascript spécifique. La bibliothèque que j'ai utilisé est p5.js, cette bibliothèque me permet de réaliser un affichage 3D des barres, j'ai donc dû apprendre comment fonctionne cette bibliothèque et les différentes commandes qui sont disponible pour pouvoir réaliser correctement l'affichage 3D est la simulation du mobile.

Durant mon projet j'ai donc dû réaliser mes objectifs dans un certain ordre pour pouvoir accomplir les prochains objectifs, parce que en effet comme vu précédemment il y a certaines de mes tâches que je ne pouvais pas réaliser avant d'en avoir réaliser certaines. J'ai donc réalisé mes objectifs dans l'ordre ou j'ai cité les différentes étapes précédemment. Bien sûr, lors de mon travail sur ces différents objectifs qui m'étaient demandé, j'ai eu plusieurs difficultés et problèmes dans leur réalisation. Je vais donc maintenant expliquer quels ont été ces problèmes et ces difficultés que j'ai pu rencontrer durant mon projet tout en expliquant par la suite comment j'ai résolu ces problèmes et ces difficultés.

Travaux réalisé

Durant toute la période pendant laquelle j'ai travaillé sur mon projet j'ai rencontré de nombreuses difficultés et problèmes qui m'ont freiné dans mon avancement sur mes objectifs dans le projet. C'est ce dont je vais parler dans cette partie, je vais expliquer mon cheminement par rapport à mes travaux tout en expliquant les difficultés que j'ai rencontré et également les solutions que j'ai pu trouver.

Au début de mon projet j'ai commencé par configurer la carte raspberry PI sur laquelle j'allais travailler, pour pouvoir la configurer j'avais besoin d'un matériel précis, j'avais tout d'abord naturellement besoin de la carte raspberry ainsi qu'une micro carte SD sur laquelle était gravé le système d'exploitation **Raspbian** qui est nécessaire pour le démarrage de la carte raspberry PI, pour cela il suffisait d'insérer la micro carte SD dans le port SD de la raspberry, ensuite il fallait juste brancher la raspberry a un écran et mettre la carte sous alimentation, une fois cela fait l'installation se lance toute seule. Une fois que

l'installation fut terminée, il me fallait configurer la raspberry pour pouvoir l'utiliser correctement. La configuration de la raspberry permet une approche plus facile de son utilisation, en effet la configuration permet de passer le clavier de QWERTY à AZERTY, de changer la langue de la raspberry et de l'OS qui par défaut en anglais, changer le mot de passe car laisser le mot de passe par défaut entrain une faille de sécurité, comme dernière configuration il m'a fallu activer le SSH(Secure Shell Protocol) qui permet de prendre le contrôle de la raspberry à distance, ce qui me sera nécessaire pour la communication à distance entre le serveur et le client.

Une fois la configuration faite, il m'a fallu également régler la date et l'heure de la raspberry pour pouvoir installer les mises à jour de la raspberry. Pour ce qui est de la configuration de la raspberry j'ai suivi le tuto détaillé sur le site raspberry-pi.fr. Une fois les configuration de bases faite j'ai dû configurer et paramétrer la raspberry pour que le serveur se mette automatiquement en route dès l'allumage de la carte raspberry PI, pour paramétrer cette action j'ai passé pas mal de temps à rechercher sur internet comment faire, car j'ai essayé plusieurs tuto différent qui explique comment faire pour lancer un programme au démarrage de la carte raspberry mais aucun de ces tutos ne marchait, j'ai donc cherché des tutos sur youtube et j'ai fini par en trouver un qui me donner le résultat que j'attendais.

Une fois que la configuration de la raspberry et les mises à jour terminées et que tout fonctionnait correctement, j'ai commencé la création du serveur web en java avec le logiciel processing. Avant de commencer la programmation du serveur j'ai tout d'abord fait des recherches sur comment créer un serveur en Java. Pour mon apprentissage du langage java j'ai utilisé des vidéo youtube et une formation sur openclassroom. Tout d'abord il paraît approprié d'expliquer ce qu'est un serveur web et comment il fonctionne, le but d'un serveur web est de transmettre des informations/données à un client. Par exemple pour accéder à un site web il faut entrer une adresse internet dans un navigateur, dans ce cas là le navigateur envoie une requête au serveur Web et ce dernier lui envoie une réponse sous forme de page HTML, dans ce cas là le client est le navigateur et le serveur est l'appareil hébergeant le site web, dans ce cas là on parle de protocole http qui est utilisée pour tout ce qui est site web ou page internet, ce protocole consiste à ce que le client envoie une requête au serveur et que celui-ci envoie une réponse au client.

Cependant ce protocole n'est pas celui que j'utilise car il n'est pas adapté aux applications en temps réel, pour réaliser la communication entre le serveur et

le client j'utilise un protocole websocket qui permet une communication en temps réel. En effet comme vu précédemment dans une relation client/serveur, un programme (le client) demande un service ou une ressource à un autre programme. Avec le websocket la démarche fonctionne dans les deux sens le serveur peut appeler le client et le client peut appeler le serveur. Pour pouvoir réaliser la communication il faut préciser l'adresse IP du serveur et le port sur lequel s'effectue la communication et le nom de domaine.

exemple de mon programme :
"socket = new WebSocket('ws://127.0.0.1:8484/mobile');"
Je créer une variable "socket" à laquelle j'attribue l'adresse IP du serveur, ici "127.0.0.1" car la communication est faite en local, puis j'attribue le port "8484" et ensuite le nom de domaine qui est "/mobile". Cette commande est exécutée par le client, du côté du serveur il faut exécuter la commande suivante : "ws = new WebsocketServer(8484, "/mobile");" ou il faut juste définir le port de communication et le nom de domaine.

Je vais maintenant présenter le serveur que j'ai réalisé, j'expliquerais son fonctionnement et le programme qui permet de le faire fonctionner. Je vais tout d'abord présenter les programmes et les expliquer. Le serveur est constitué de quatre programmes différents dont trois classes, le premier programme regroupe les trois classes est créé une interface qui affiche la vitesse et la position de chaque moteur qui font bouger les barres du mobile.

Les trois classes sont CaseMot, communication et motif, la classe CaseMot s'occupe du rendu de l'interface ou s'affiche les informations sur la position et la vitesse des moteurs, cette classe permet d'avoir un rendu organisé de ces informations. La classe communication comme son nom l'indique s'occupe de la partie communication entre le serveur et le client, elle s'occupe également de traiter les informations reçues depuis le client et les informations envoyées au client. La dernière classe, la classe Motif a pour rôle de calculer la vitesse et la position des moteurs pour qu'ils reproduisent les motifs choisie dans l'interface web. Le premier programme que je vais présenter est le programme principal, puis je présenterai la classe CaseMot, ensuite la classe communication et pour finir la classe Motif.

Programme Principal :

```
Communications com;
PApplet pApplet;
int nMoteurs = 10;
CaseMot [] casesMot;
boolean on = false;

void setup() {
  size(800, 600, P3D);

  fill(255);
  stroke(255);
  textSize(16);

  casesMot = new CaseMot[nMoteurs];
  for (int i = 0; i<nMoteurs; i++) {
    casesMot[i] = new CaseMot(i);
  }

  pApplet = this;

  com = new Communications(pApplet);

  com.start();

}

void draw() {
  background(0);
  fill(255);
  for (int i = 0; i<nMoteurs; i++) {
      casesMot[i].maj();
    }
}

void mouseWheel(MouseEvent event) {
  float e = event.getCount();
  for (int i = 0; i<nMoteurs; i++) {
    casesMot[i].incVitesse(int(-e));
  }
}
```

```

void webSocketServerEvent(String msg) {
  com.setWsRecu(msg);
  println("recu du websocket client = \t" + msg);
}
```

En premier lieu on commence par définir les différentes variables du programmes et appeler les classes Communications et CaseMot pour pouvoir les utiliser dans ce programmes. Ensuite dans le “setup()” on crée l’interface et on définit les paramètres d’affichage du texte. On appelle également les classes CaseMot et Communications. La classe CaseMot s’occupe de personnaliser et l’affichage des informations obtenue depuis le serveur grâce à la classe Communication qui a permis la récupération de ses informations et les a transmis à la classe CaseMot. Lorsque l’on appelle la classe communication dans ce programme c’est pour pouvoir démarrer la communication entre le serveur et le client car la classe communication est une classe et ne peut pas démarrer toute seule, elle a besoin d’être implanté dans le programme principal qui sert d’hôte à la classe. Après avoir fait l’appelle des classes on affiche grâce à la fonction “draw()” la classe CaseMot que l’on répète en fonction du nombre de moteurs défini avec la variable “nMoteurs”, on la répète donc dix fois.

On exécute également la fonction “mouseWheel(MouseEvent event)” qui a pour but de faire fonctionner les événements activable avec la souris, préciser dans la classe CaseMot.
Et pour finir on exécute la fonction “webSocketServerEvent(String msg)” qui a pour rôle d’afficher dans la console les message reçu du client (le début et la fin de la communication serveur/client, le motif et le mode d’affichage, et également la vitesse de chaque moteur) lorsque les informations concernant le mobile et les moteurs sont mis à jours, grâce à la classe communication qui est rappelé dans cette fonction à travers la variable “com” préciser au début du programme.

Class CaseMot :

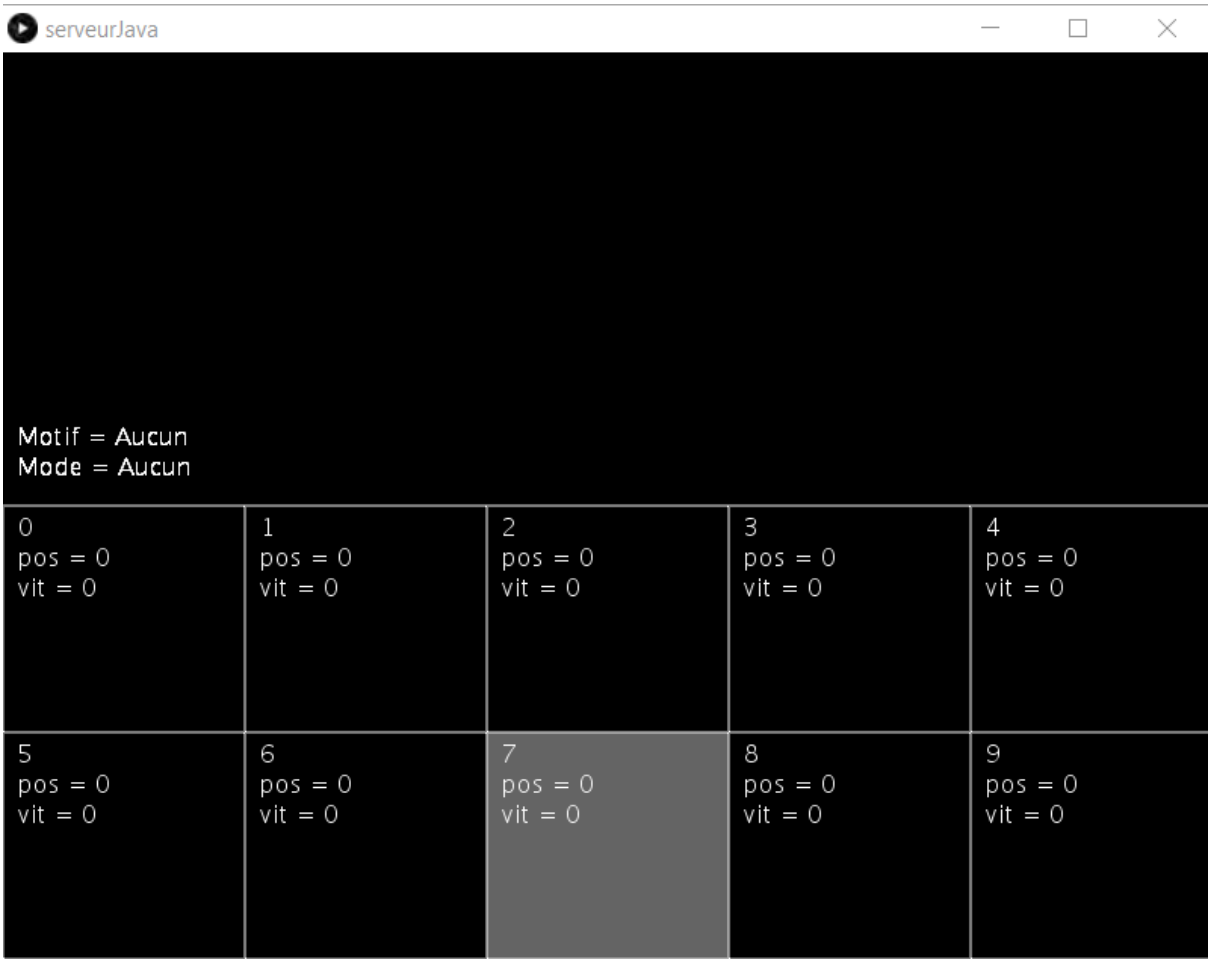
```
class CaseMot {
  private int xAff, yAff;
  private int l,h;
  private boolean sel = false;
  private int id = 0;
  private String textPos = "pos = 0";
  private String textVit = "vit = 0";
  private byte vitesse;

  CaseMot(int _id){
    vitesse = 0;
    id = _id;
    l = width/(nMoteurs/2);
    h = height/4;
    xAff = (id%(nMoteurs/2))*l;
    yAff = floor(id/(nMoteurs/2))*h+height/2;
  }

  void maj(){
    aff();
  }

  void aff(){
    if(isSel()) fill(0,180,0);
    else if (mouseOver()) fill(100);
    else fill(0);

    rect(xAff,yAff,l,h);
    fill(255);
    text(id,xAff+10,yAff+20);
    textPos = "pos = " + com.getPos()[id];
    text(textPos,xAff+10,yAff+40);
    text(textVit,xAff+10,yAff+60);
  }

  boolean mouseOver(){
    if(mouseX>xAff && mouseX<xAff+l && mouseY>yAff && mouseY<yAff+h) return true;
    else return false;
  }

  boolean isSel(){
    if(mousePressed && (mouseButton == LEFT) && mouseOver()){
      for(int i=0;i<nMoteurs;i++){
        if (i != id) casesMot[i].sel = false;
      }
      sel = true;
    }
    return sel;
  }

  void incVitesse(int e){
    if(sel) vitesse +=e;
    if(vitesse>127) vitesse = 127;
    if(vitesse<-127) vitesse = -127;
    textVit = "vit = " + vitesse;

  }

  byte getVitesse(){
    return vitesse;
  }

  void setVitesse(int _v){
    vitesse = byte(_v);
    textVit = "vit = " + vitesse;
  }
}
```

Ceci est le programme de la classe CaseMot qui gère les informations à afficher sur l'interface du serveur. Dans ce programme, je commence tout d'abord par créer la classe CaseMot en créant les différents objets de la classe et la méthode "CaseMot(int _id)" ou j'attribue des valeurs aux objets.

Ensuite je vais commencer par créer deux fonctions "maj()" et "aff()" la fonction "maj()" rappelle la fonction "aff()" qui elle est la fonction qui affiche les information sur l'interface du serveur, tout d'abord on utilise la condition "if()" qui vérifie les conditions des fonctions "isSel()" et "mouseOver()", la condition pour la fonctions "mouseOver()" est que si la souris est présente dans un certain périmètre il faut retourner cela avec la commande "return true" sinon "return false" et donc si on "return true" la condition "if()" s'active ce qui a pour conséquence de changer la couleur de fond du périmètre précisé dans la fonctions "mouseOver()".
La fonction "isSel()" quant à elle change le fond du périmètre choisie lorsque l'on appuie sur le clique gauche de la souris et permet cela pour plusieurs périmètre définie et laisse le fond lorsque la souris est relâché. Ensuite avec la commande "rect(xAff, yAff, l, h);" on créer le périmétre sur lequel on va agir et afficher du texte avec la commande "text()" que l'on répétera plusieurs fois pour pouvoir afficher plusieurs information différente. On répétera ensuite cette action 10 fois dans le programme principal pour créer 10 cases, une pour chaque moteur.

Ensuite après avoir modifié la zone d'affichage pour un rendue plus propre et organisée j'ai créer la fonction "incVitesse(int e)" qui va permettre d' incrémenter la valeur de la vitesse des moteurs enregistrer sur la variable "textVit" pour afficher la vitesse des moteurs obtenue du client, c'est la fonction "setVitesse(int _v)" qui va permettre d'attribuer une valeur à la variable "textVit", pour pouvoir utiliser la vitesse des moteurs récupérer depuis le client j'utilise la méthode "byte getVitesse()" qui permet de récupérer la variable vitesse avec les valeurs des vitesses des moteurs qui est dans la classe Communication et la commande "return vitesse;" permet de renvoyer les valeurs obtenues à l'objet vitesse définie plus tôt dans la classe CaseMot.

Je vais maintenant montrer le rendu de l'interface d'affichage des informations sur le serveur. Voici le rendu de l'interface quand il ne reçoit encore aucune information et que l'on n'interagit pas avec. Je le présenterais avec des informations provenant du client dans la partie concernant la communication serveur/client.



L’interface est construite de façon à afficher les informations pour chaque moteur séparément et également dire quel est le motif et le mode sélectionnée pour le mobile. il y a dix espaces numérotés, un pour chaque moteur avec la position et la vitesse de chaque moteur. Ici elle est à zéro car on n'interagit pas avec le client.

Voilà pour ce qui est de l’interface d’affichage du serveur Java je présenterais les deux autres classe plus tard, je vais présenter d’abord le programme du client et ensuite je présenterai le programme de la classe Communication, je présenterai la classe Motif après avoir monter et expliquer l’affichage 3D des barres du mobile. Donc pour ce qui est du client créer en javascript voici tout d’abord le programme pour créer l’interface Web de commande du mobile, je montrerais le programme complet pour la partie de l’affichage 3D qui sera compris dans ce programme là.

Client JavaScript

```
var socket;
var msg = "";
nMoteurs = 10;
let sliders = [nMoteurs]
let on = false;
let pos = []
let vit = [];
let canvas2;
let wood;

function preload(){
  wood = loadImage('texture/wood.jpg')
}
function setup() {
  createCanvas(windowWidth, windowHeight);
  canvas2 = createGraphics(460, 450, WEBGL);
  socket = new WebSocket('ws://127.0.0.1:8484/mobile');
  socket.onmessage = readMessage;
  bModeOn = createButton('ON/OFF');
  bModeOn.position(70, 330);
  bModeOn.mousePressed(function(){
    console.log(on);
    if(on){
    sendMessage("on");
    on = false;
    }
    else{
      sendMessage("off");
      on = true;
    }

  bModeRaz = createButton('RAZ');
  bModeRaz.position(19, 330);
  bModeRaz.mousePressed(function(){sendMessage("raz");raz()});
  bModeManuel = createButton('mode manuel');
  bModeManuel.position(19, 270);
  bModeManuel.mousePressed(function(){sendMessage("mode= manuel");raz()});
  bModeAuto = createButton('mode auto ....');
  bModeAuto.position(19, 300);
  bModeAuto.mousePressed(function(){sendMessage("mode= auto");raz()});
  bMotifNeutre = createButton('motif neutre');
  bMotifNeutre.position(159, 270);
  bMotifNeutre.mousePressed(function(){sendMessage("motif= neutre");});
  bMotifSinus1 = createButton('motif sinus1');
  bMotifSinus1.position(159, 300);
  bMotifSinus1.mousePressed(function(){sendMessage("motif= sinus1");});
  bMotifSinus2 = createButton('motif sinus2');
  bMotifSinus2.position(159, 330);
  bMotifSinus2.mousePressed(function(){sendMessage("motif= sinus2");});
  for (let i = 0;i<nMoteurs;i++){
    sliders[i] = createSlider(-127,127,0,1);
    sliders[i].position(375 + 200*(i%2), 140 + 30*int(i/2));
    sliders[i].style('width', '180px');
  }

}
```

```
function draw() {
  background(0);
  canvas2.background(0);
  text("on = " + on, 73,365);
  for (let i = 0;i<nMoteurs;i++){
  text("vitesse mot " + i + " : "+ sliders[i].value(),460+ 200*(i%2),140+ 30*int(i/2));
  sliders[i].mouseReleased(function (){sendMessage("vmot"+nf(i,3,0)+"="+sliders[i].value())});
  }
  fill(255);
  text(msg, 20, 20);
  tab = msg.split(',');

  if (tab[0] == "POS" && tab[nMoteurs+1] == "FIN") {
    for (let i = 0; i < nMoteurs; i++) {
    oldPos = pos[i];
    pos[i] = tab[i + 1];
    vit[i] = pos[i]-oldPos;
    }
  }
  if(tab[0] == 'INIT'){
    for (let i = 0; i < nMoteurs; i++) {
    pos[i] = 0;
    vit[i] = 0;
    }
  }
  for (let i = 0; i < nMoteurs; i++) {
    text(i + " : p=" + pos[i] + " , v=" + vit[i], 20 + 170 * (i % 2), 170 + 20 * int(i / 2));
  }
  push();
  textSize(20);
  textStyle(BOLD);
  text("Interface de Commande", windowWidth/3, 40,);
  text("Choix vitesse des Moteurs", 470, 110);
  text("Vitesse et Position des moteurs", 15, 125);
  pop();
```

```
}

function readMessage(event) {
  msg = event.data;
}
function sendMessage(msgWs) {
  socket.send(msgWs);
  console.log("msg envoy" = " + msgWs);
}
function raz(){

  for (let i = 0;i<nMoteurs;i++){
    sliders[i].value(0);
  }
}

function windowResized() {
  resizeCanvas(windowWidth, windowHeight);
}
```

En premier bien évidemment je commence par définir les variables que je vais utiliser dans mon programme, il y a cependant quelques variables sur lesquelles je reviendrais seulement dans la partie de l'affichage 3D des barres pareil pour la fonction preload qui sert à importer une texture pour les barres en 3D. La première fonction à présenter est la fonction "setup()" où je créer l'interface de commande, initialise la communication par Websocket avec le serveur et crée des sliders pour piloter la vitesse des moteurs et je crée également des boutons pour pouvoir choisir le mode de pilotage des motif (automatique ou manuelle), pour choisir le motif et pour la remise à zéro.

Pour créer l'interface j'utilise deux commande une pour créer la fenêtre où s'affiche toute les informations et création (boutons, sliders, mobile) pour cela j'utilise la commande "createCanvas(windowWidth, windowHeight);" dans cette commande il suffit de justifier la taille de l'interface, ici c'est la taille de l'écran. La deuxième commande est "canvas2 = createGraphics(460, 450, WEBGL);" qui permet de créer une seconde interface par dessus la première, cette seconde interface est là pour afficher les barres en 3D.

Puis viens la commande pour débuter la communication WebSocket avec le serveur "socket = new WebSocket('ws://127.0.0.1:8484/mobile');" avec cette commande je créer une communication en WebSocket avec le serveur en définissant son adresse IP et le port sur lequel s'effectue la communication et également le nom de domaine. L'adresse IP du serveur ici est 127.0.0.1 car je fais pour l'instant la communication en mode local, lorsque le serveur sera hébergé sur la Raspberry il suffira de mettre l'adresse IP de la Raspberry.

En suite avec la commande "socket.onmessage = readMessage;" qui permet de récupérer des messages envoyé par le serveur, pour cela on utilise également la fonction "readMessage(event)" définie plus bas dans le programme, où dans cette fonction on attribue donc les informations reçue depuis le serveur à la variable "msg".
Ensuite juste en dessous je créer un bouton qui sert pour activer l'envoie d'information vers le serveur, lorsque c'est sur "on" l'envoie d'information fonctionne et lorsque c'est "off" l'envoie d'information ne fonctionne pas, pour l'instant je n'ai pas réaliser cette étape pour définir si oui ou non les informations sont envoyé, il ya juste le bouton et lorsque l'on appuie dessus cela envoie le message "on" ou "off" au serveur selon l'état du bouton. Et juste en dessous je crée les autres boutons pour les motifs, les modes et la remise à zéro, et lorsque j'appuie sur un bouton cela envoie un message au serveur pour dire que j'ai appuyé sur tel bouton, et pareil pour les sliders que je crée

juste après qui permettent de choisir la vitesse des moteurs cependant l'envoie des donnée des sliders se fait plus bas dans la fonction "draw()".

Ce qui fait donc maintenant le parallèle avec cette fonction, dans laquelle je vais réaliser les commandes pour l'affichage du texte et la couleur de fond pour l'interface. C'est justement par quoi commence cette fonction, je défini le fond en noir et écrit le texte à afficher notament pour la valeur du bouton "on/off" et la vitesse des moteurs modifié avec les sliders que j'envoie en même temps au serveur.
Pour envoyer des informations au serveur j'utilise la fonction "sendMessage(msgWs)" qui est définie plus bas dans le programme. Ensuite de la ligne 66 à 84 le programme permet de récupérer les valeurs de la vitesse et de la position des moteurs envoyés par le serveur et ensuite de les afficher sur l'interface de commande.

Après cette partie je rajoute du texte pour définir chaque partie qui est affiché sur l'interface. Et pour finir le programme on a les fonction "readMessage(event)" et "sendMessage(msgWs)" cité précédemment ainsi que les fonction "raz()" et "windowResized()", la fonction "raz()" permet la remise à zéro de toute les données que ce soit la position, la vitesse, le motif ou le mode. La "windowResized()" quant à elle permet juste d' avoir un réajustement automatique de l'interface lorsque la taille de la fenêtre du navigateur change.

Voici un aperçu de l'interface de commande du mobile du mobile, à droite il y a ce dont je viens de parler lors de la présentation du programme et à gauche il y a la retranscription des barres du mobiles en 3D. Pour la partie droite on peut voir les sliders pour changer la vitesse des moteurs et au-dessus de chaque slider il y a le numéro du moteur que chaque slider pilote ainsi que la vitesse choisie avec le slider. On peut voir également les différents boutons, ceux pour les motifs, les modes, la remise à zéro et pour débuter la communication, il y a également la vitesse et position des moteurs qui ici n'affiche aucune valeur car la communication avec le serveur n'est pas lancée. Je vais donc revenir sur la classe Communication du serveur pour présenter le fonctionnement du programme et le résultat qu'on obtient.

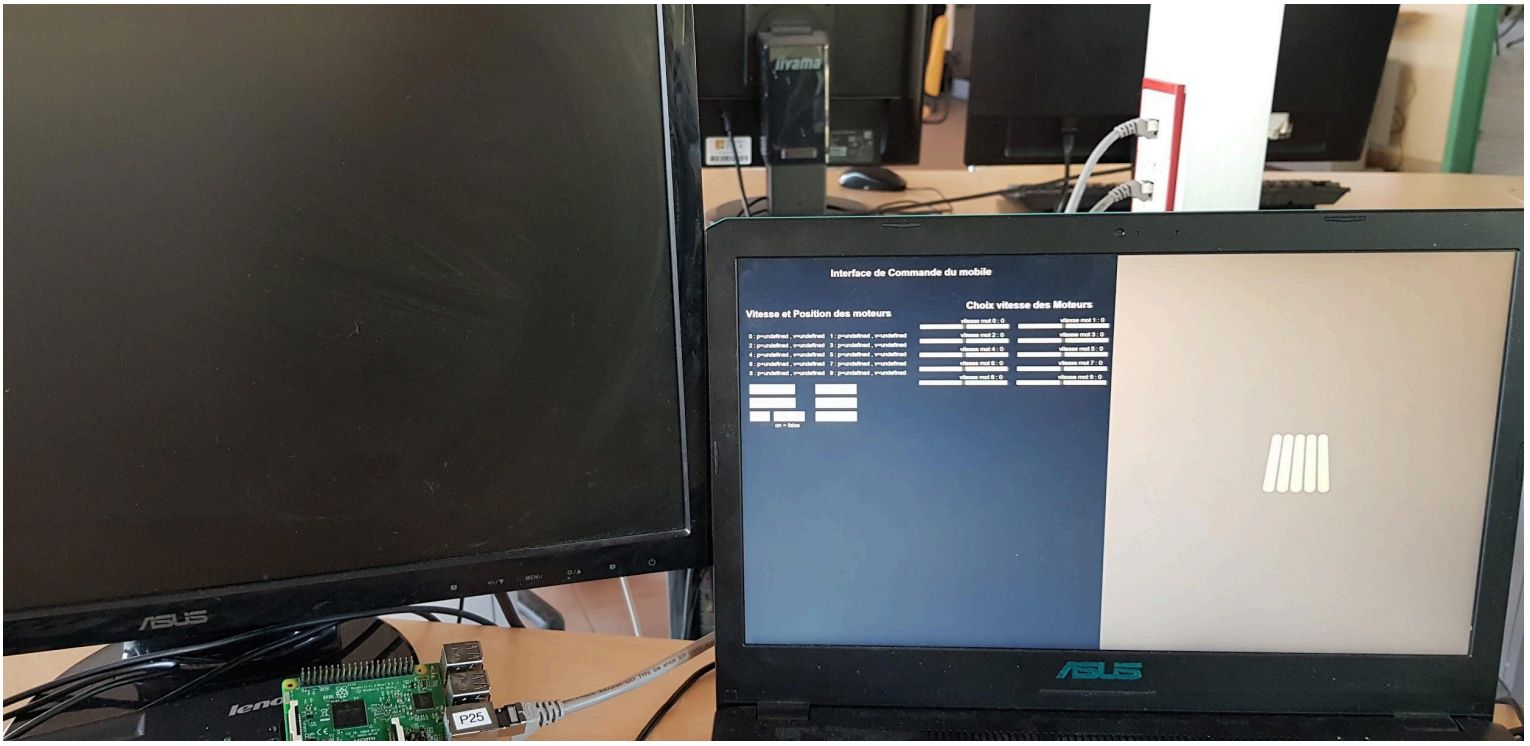
```
import websockets.*;

public class Communications extends Thread {
  Motif motif;
  int[] positions;
  byte[] vitesses;
  long maintenant, to;
  byte in = 6;

  WebsocketServer ws;
  String wsRecu = "";

  Communications(PApplet _pApplet) {
    pApplet = _pApplet;
    motif = new Motif();
    positions = new int[nMoteurs];
    vitesses = new byte [nMoteurs];
    ws = new WebsocketServer(pApplet, 8484, "/mobile");
    to=System.nanoTime();
  }

  void start() {
    super.start();
  }

  void run() {
    while (true) {
      maintenant = System.nanoTime();
      int dt = 73530*256;
      if ((maintenant - to)>dt) { // permet de simuler le comportement d'Arduino ...
        if (in == 6) {
          motif.maj(positions);
          calcVitesses();
          calcPos();
          String msg="POS,";
          for (int i = 0; i<nMoteurs; i++) {
            msg +=positions[i] + ",";
          }
          msg+="FIN";
          ws.sendMessage(msg);
        } else {
          print(hex(in) + " ");
        }
        to = to + dt;
      } else delay(1);
    }
  }


  void calcVitesses() {
    motif.maj(positions);
    vitesses = motif.getVitesse();
  }

  void calcPos() {
    for (int i =0; i<nMoteurs; i++) {
      positions[i] += vitesses[i];
    }
  }

```

```
63    byte[] getVitesses() {
64      return vitesses;
65    }
66
67    int[] getPos() {
68      return positions;
69    }
70
71    void setWsRecu(String _s) {
72      if (_s.equals("on")) on = true;
73      else if (_s.equals("off")) on = false;
74      else if (_s.equals("raz")) {
75        for (int i=0; i<nMoteurs; i++) {
76          casesMot[i].sel = false;
77          casesMot[i].setVitesse(0);
78          motif.raz();
79          positions[i] = 0;
80          vitesses[i] = 0;
81        }
82      } else if (_s.substring(0, 4).equals("vmot")) {
83        int idMot = int(_s.substring(4, 7));
84        int vMot = int(_s.substring(8, _s.length()));
85        casesMot[idMot].sel = true;
86        casesMot[idMot].setVitesse(vMot);
87      } else if (_s.substring(0, 6).equals("motif=")) {
88        wsRecu = _s.substring(6, _s.length());
89        // TO DO
90        println("nouveau motif = " + wsRecu);
91      } else if (_s.substring(0, 5).equals("mode=")) {
92        wsRecu = _s.substring(5, _s.length());
93        //TO DO
94        println("nouveau mode = " + wsRecu);
95      } else println("message ws recu inconnu = " + _s);
96    }
97
98    void resetPos() {
99      for (int i =0; i<nMoteurs; i++) {
100       positions[i] = 0;
101     }
102   }
103 }
```

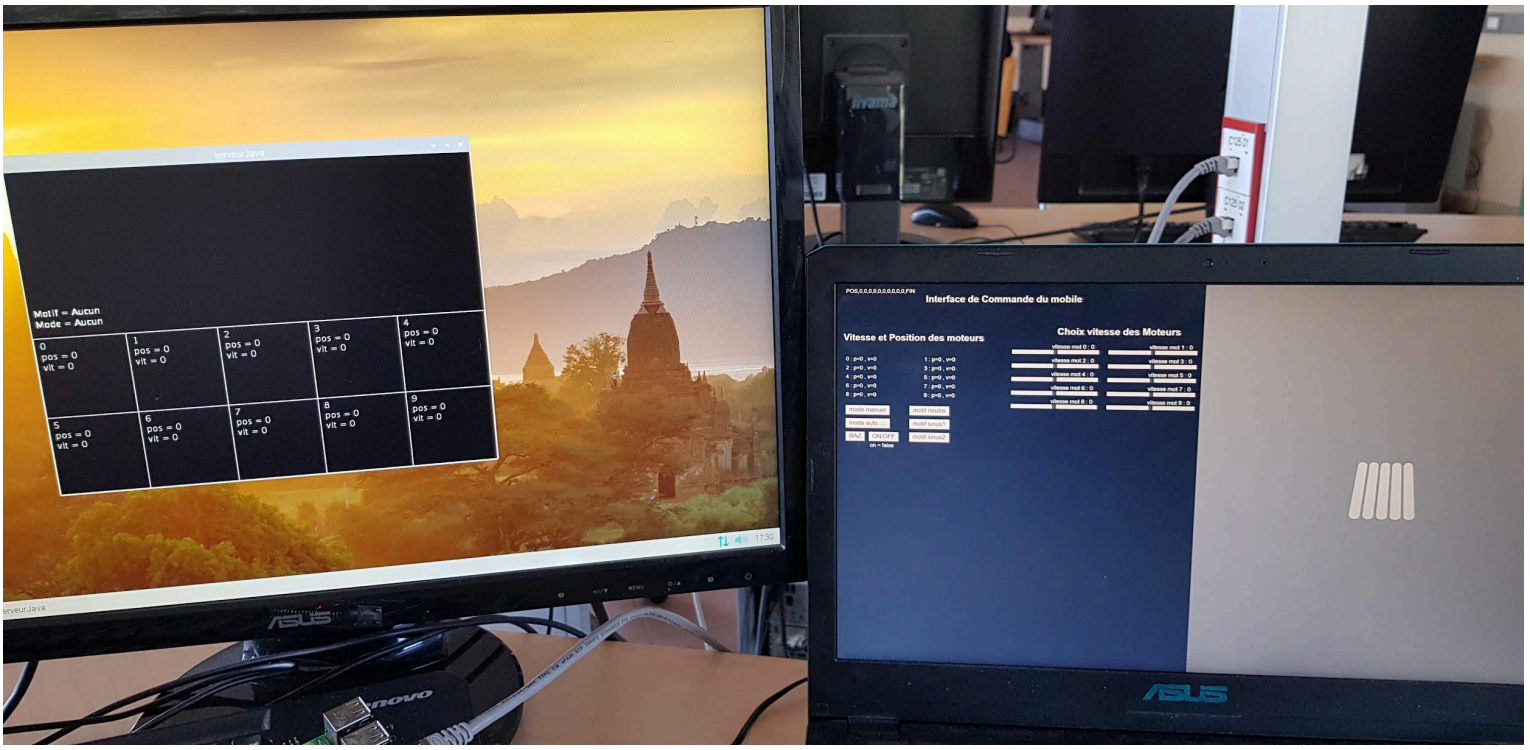
Pour commencer il faut d'abord créer la classe, les objets et les méthodes mais il faut également importer la bibliothèque Websocket pour que la communication puisse se faire en Websocket il faut utiliser des commandes spéciales seulement disponible avec cette bibliothèque.
Dans la méthode "Communications(PApplet _pApplet)" on attribue des valeurs aux objets et en initialise la communication avec le client grâce à la commande :
"ws = new WebsocketServer(pApplet, 8484, "/mobile");" où l'on dit sur quel port se fait la communication, le port à été définie dans le client et également le nom de domaine "mobile", et l'on met toute ces information dans l'objet "ws". Ensuite dans "void run()" le serveur simule le comportement de l'arduino

qui doit récupérer les informations concernant la vitesse choisie pour les moteurs, ainsi que le mode et le motif et renvoie la position de chaque moteur au serveur qui l'envoie ensuite au client. C'est également dans cette fonction que le serveur envoie la position au client grâce à la commande "ws.sendMessage(msg);".

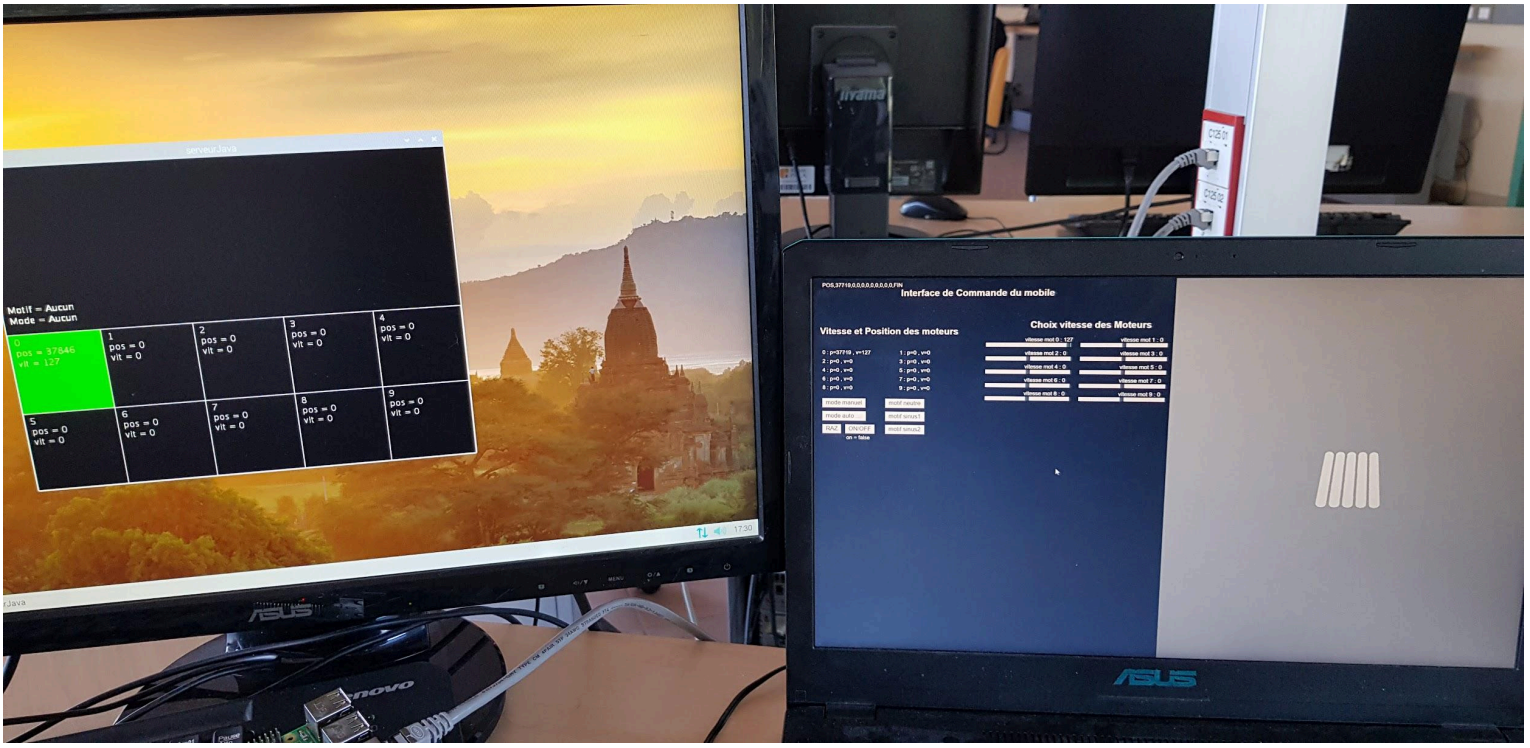
Ensuite dans la fonction "calcVitesses()" le programme récupère la vitesse qui est calculée en fonction du motif choisie dans la classe Motif, et dans la fonction "calcPos()" on calcul la position des moteurs en fonction de la vitesse, puis on renvoie les données de vitesse et de position. Et pour finir le programme dans la fonction :
"setWsRecu(String _s)" on lis les données reçues depuis le serveur et les affiches dans la console et sur l'interface à l'aide de la classe CaseMot.
Et à la toute fin en définit la fonction "resetPos()" pour pouvoir mettre les positions à zéro des moteurs. Je vais donc maintenant vous montrer le résultat obtenu grâce au programme, pour la communication entre le serveur et le client.



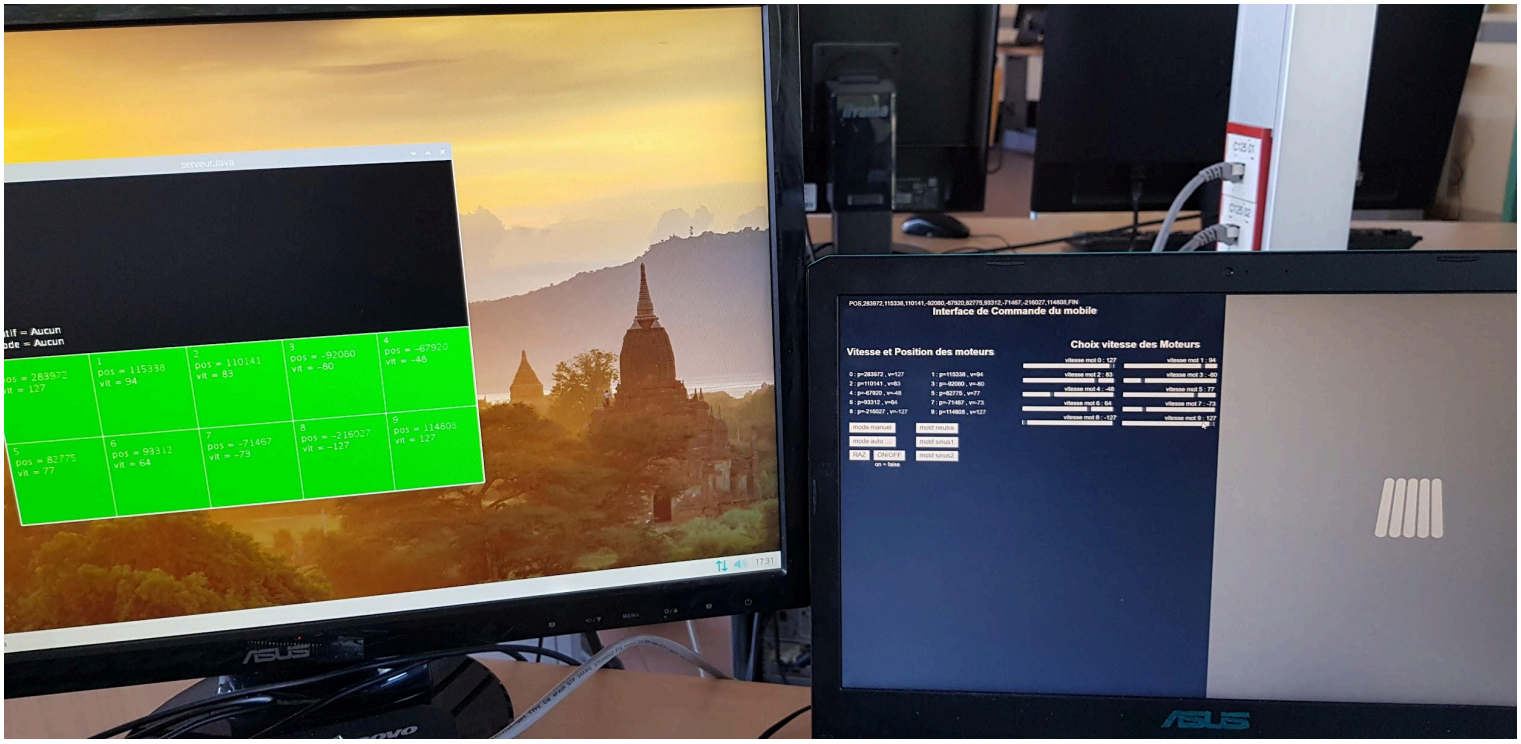
Cette image montre le résultat obtenu sur le client lorsque le serveur est éteint, pour bien visualiser la différence voici une image du rendue une fois le serveur allumé.



Ici on peut donc voir le rendue lorsque le serveur est en marche, on peut voir qu'une partie du texte à été réduit et à été remplacé par des valeurs, ces valeurs indique la positions et la vitesse de chaque moteurs, avant cela du texte s'afficher pour dire qu'il n'y avait aucune valeur définie car la communication n'était pas activé et donc le client ne recevait aucune information du serveur.
La prochaine image montre le rendu des deux interfaces lorsque des informations sont échangées.



On peut donc voir que lorsque des informations sont échangées que sur l'interface du serveur une partie est devenue verte, cette partie correspond au moteur dont la vitesse à été modifié depuis l'interface de commande, on peut également voir sur l'interface de commande que les valeurs ont changé et que le premier slider à été utilisée ce qui a donc modifié les valeurs de vitesse du moteur numéro 1.



La dernière image montre le même rendu que la précédente image à la différence que celle-ci donne un aperçu de lorsque tous les sliders sont utilisés, ce qui modifie donc toutes les cases présentes sur l'interface du serveur car on change la vitesse de chaque moteur. La seule partie qui ne fonctionne pas encore c'est l'affichage du motif et du mode choisie que j'ai décidé de rajouter il y a quelques jours et je n'ai pas encore fini de programmer cette partie.

Maintenant que j'ai expliqué comment fonctionne la communication serveur/client et le fonctionnement du serveur et du client je vais expliquer la classe Motif qui sert à reprendre les motifs envoyés par le client pour les transmettre à la carte arduino.

Class Motif

```
class Motif {
  private byte[] vitesses;
  private int[] pos;
  private boolean [] cibleAtteinte;

  long tInit;
  byte[] vCalc = new byte[nMoteurs];

  Motif() {
    vitesses = new byte[nMoteurs];
    pos = new int[nMoteurs];
    cibleAtteinte = new boolean[nMoteurs];
    tInit = System.nanoTime();
  }

  void maj(int[]_pos) {
    pos = _pos;
    vitesses = calcN();
  }

  byte[] calcN () { // calcule le nombre de pas à effectuer suivant le type de motif

    for (int i=0; i<nMoteurs; i++) {
      vCalc[i] = byte(casesMot[i].getVitesse());
    }

    return vCalc;
  }

  boolean atteindreCible(int cible, byte vit) {
    int nCiblesAtteintes = 0;
    for (int i = 0; i<nMoteurs; i++) {
      if (pos[i]==cible && cibleAtteinte[i]) {
        vCalc[i] = 0;
        nCiblesAtteintes ++;
      }

      if (!cibleAtteinte[i]) {
        if (abs(pos[i] - cible)>vit) {
          vCalc[i] = vit;
        } else {
          vCalc[i] = byte(abs(pos[i] - cible));
          cibleAtteinte[i] = true;
        }
        if (pos[i] > cible) vCalc[i]*=-1;
      }
    }
    if (nCiblesAtteintes == nMoteurs) {
      cibleAtteinte = new boolean[nMoteurs];
      return true;
    } else {
      return false;
    }
  }

  byte[] getVitesse() {
    return vitesses;
  }

  void raz() {
    for (int i=0; i<nMoteurs; i++) {
      pos[i] = 0;
      vitesses[i] = 0;
    }
  }
}
```

Pour cette classe les schéma reste le même que pour les classes précédentes, on crée la classe, on définit les objets et les méthodes de la classe. Puis vient ensuite le tour des fonctions qui définissent ce que fait le programme, ici on prend la fonction “maj()” qui permet d’attribuer les valeurs de la vitesse et de la position à des variables dont l’une récupérées depuis la variable “calcN” qui vient juste après.

Puis on créer la méthode “atteindreCible()” la calcul la position des moteurs ainsi que leur vitesse par rapport au motif choisie avec l’interface, pour obtenir une vitesse équivalente est la position de chaque moteur en fonction du motif choisie, puis vérifie si cela est correctement appliqué et renvoie c’est valeur avec la commande “return”. Et il y a également la fonction de remise à zéro pour la position et la vitesse des moteurs ce qui amènera également une remise à la position de départ des barres du mobile.

Maintenant que j’ai expliqué comment fonctionne la classe Motif je vais expliquer le fonctionnement de l’affichage 3D des barres du mobiles. Tout d’abord l’affichage des barres en 3D se fait en langages javascript car les barres seront affichées dans l’interface web pour avoir un aperçu direct. Pour pouvoir faire de la 3D en javascript il faut avant tout utiliser une bibliothèque spéciale qui permet de faire de la 3D, pour ma part j’utilise la bibliothèque p5.js. Avec cette bibliothèque je peux créer une interface en définissant une longueur et une hauteur qui pourra afficher de la 3D, je peux créer cette interface grâce à la commande “createcanvas(width, height, WEBGL);”.

La partie “WEBGL” est ce qui me permet d’afficher de la 3D, WEBGL est une API javascript qui permet une interaction 2D et 3D sur une page web. Après avoir créé l’interface il suffit de créer les barres, pour cela j’utilise la commande “cylinder(width, height);”, cette commande permet de créer un cylindre avec un diamètre et une longueur choisie. Voici maintenant la partie du programme du client pour l’affichage des barres.

Class Barre

```
class Barre {
  constructor(radius, height, rotate) {
    this.radius = radius;
    this.height = height;
    this.rotate = rotate;
  }
}

class Mouvement {
  constructor(x0, x1, x2, x3, y, z0, z1, z2){
    this.x0 = x0;
    this.x1 = x1;
    this.x2 = x2;
    this.x3 = x3;
    this.y = y;
    this.z0 = z0;
    this.z1 = z1;
    this.z2 = z2;

  }
}
```

Intégration dans le programme

```
  let cylindre = new Barre(10, 180, 180);
  let move = new Mouvement(0, -25, 75, 25, 0, 0, 0, 0);

  canvas2.push();
  canvas2.noStroke();
  canvas2.texture(wood);
  canvas2.rotateX(cylindre.rotate);
  canvas2.translate(move.x0, move.y, move.z0);
  canvas2.cylinder(cylindre.radius, cylindre.height);
  canvas2.translate(move.x1, move.y, move.z1);
  canvas2.cylinder(cylindre.radius, cylindre.height);
  canvas2.translate(move.x1, move.y, move.z1);
  canvas2.cylinder(cylindre.radius, cylindre.height);
  canvas2.translate(move.x2, move.y, move.z2);
  canvas2.cylinder(cylindre.radius, cylindre.height);
  canvas2.translate(move.x3, move.y, move.z2);
  canvas2.cylinder(cylindre.radius, cylindre.height);
  image(canvas2, windowWidth/2, 3);
  canvas2.pop();
```

Pour faire l'affichage des barres j'ai créé une classe Barre et une classe Mouvement, la classe Barre sert pour créer les barres et la classe Mouvement sert à définir leur position et les mouvements qu' elles effectueront en fonction du motif choisi.
La création d'une classe en javascript est un peu différente de en Java, pour créer une classe il faut créer un constructor qui permet de définir les objets que je vais utiliser, et ensuite j'attribue ces objet à des variables que je réutilise en leur attribuant des valeurs avec la commande "new Barre(10, 180, 180)" pour la classe Barre et "new Mouvement(0, -25, 75, 25, 0, 0, 0, 0);" pour la classe Mouvement. Cette commande attribue des valeurs aux objets définie dans le constructor qui ensuite sont définie aux variables et ensuite ces infomation sont attribuer à une variable qui représente la classe que l'on veut utiliser plus tard dans le programme.

Par exemple dans la commande "canvas2.rotateX(cylindre.rotate);" je dis d'abord que j'effectue cette action dans la fenêtre canvas2 définie plus haut dans le programme et ensuite j'indique que j'utilise la variable "cylindre" associé à la classe Barre qui va chercher la variable "rotate" associé à l'objet "rotate" qui à la valeur de "180", donc on fait tourner la barre de 180° sur l'axe des X. Ensuite on réalise cette action de différente manière en fonction des attributs de chaque commande.

Les deux commandes qui sortent du lots et qui permettent un bon fonctionnement et affichage sont tout d'abord la commande "image(canvas2, windowWidth/2, 3);" qui permet d'attribuer la fenêtre canvas2 comme une image et de pouvoir l'afficher par dessus la première fenêtre et ensuite la commande "canvas2.push(); , canvas2.pop();" qui permet d'isoler cette partie du programme pour que les axes XYZ de la deuxième fenêtre ne se mélange pas à la première et que l'on obtient pas un mouvement des barres complétements désorganiser par rapport aux axes, cela facilite donc la manipulation des mouvements des barres. Je vais donc maintenant passer à la présentation des programmes pour chaque motif, pour les motifs que le mobile doit réaliser il y en a trois différent, le motif neutre, sinus 1 et sinus 2, tout d'abord je présente les programmes et ensuite le résultat.

Motif Neutre

```
let z0 = -10;
let z1 = 0;
let z2 = 0;
let y = 0;

function draw() {
  background(0);
  noStroke();
  push();
    rotateX(180);
    fill(200, 190, 140);
    translate(0, y, z1);
    cylinder(10,180);
  pop();

push();
    rotateX(180);
    fill(200, 190, 140);
    translate(-25, y, z1);
    cylinder(10,180);
pop();

push();
rotateX(180);
fill(200, 190, 140);
translate(25, y, z1);
cylinder(10,180);
pop();

push();
rotateX(180);
fill(200, 190, 140);
translate(-50, y, z1);
cylinder(10,180);
pop();

```

```
36
37  push();
38  rotateX(180);
39  fill(200, 190, 140);
40  translate(50, y, z1);
41  cylinder(10,180);
42  pop();
43  if(z1 === -40){
44    z1 = z1 + 80;
45  }
46  else {
47    z1 = z1 - 1;
48   }
49  }
```

Je commence par créer des variables pour les positions des barres sur les axes YZ ce qui permettra de faire bouger leur position. Ensuite je crée les barres séparément grâce aux commandes “push() et pop()” qui permettent d’isoler des parties du programme sans qu'elles affectent les valeurs des autres parties du programme. Car pour faire bouger les positions des barres sur l’axe Z j’ai besoin d’incrémenter la valeur de leur position, et lorsque je défini une position sur un axe pour une barre, la position de la barre suivante prendra en compte sa position par rapport à la barre précédente. Par exemple si avec une barre je définit sa position sur l’axe Z à “Z= 10” cette barre se positionnera à 10 sur l’axe Z, mais si je créer une seconde barre avec également sa position à Z= 10, cette seconde barre prendra comme point de départ la position de la barre précédente et sera à la valeur 20 sur l’axe Z. Alors que si je définit leur position avec les commandes “push() et pop()” pour délimiter la zone d’action cela n’aura aucun effet sur les autres barres. Pour clôturer le programme je définis la condition “if()” qui dit que si la position “z1” atteint “-40” elle obtiendra la position de “80” et sinon elle incrémente de “1”. Donc maintenant voici un aperçu de ce que réalise ce programme du motif Neutre.

Motif Sinus 1

```
let z0 = 0;
let z1 = 0;
let z2 = 0;
let y = 0;

function draw() {
  background(0);
  noStroke();
  push();
    rotateX(180);
    fill(200, 190, 140);
      translate(-15, y, z0);
      cylinder(10,180);
  pop();

push();
    rotateX(180);
    fill(200, 190, 140);
      translate(-40, y, z1);
      cylinder(10,180);
      translate(-25, y, z1);
      cylinder(10,180);
      if(z1 === 20){
        z1 = z1 - 40;
      }
      else {
        z1 = z1 + 1;
      }
pop();
```

```
push();
  rotateX(180);
  fill(200, 190, 140);
    translate(10, y, z2);
    cylinder(10,180);
    translate(25, y, z2);
    cylinder(10,180);
    if(z2 === -20){
      z2 = z2 + 40;
    }
    else {
      z2 = z2 - 1;
    }
pop();
}
```

Pour ce qui est du Motif Sinus1 le procédé du programme est quasiment le même que pour le Motif Neutre, il y a juste les valeurs qui changent et une deuxième position sur l'axe Z attribuer aux barres ce qui permet de créer des mouvement différent pour certaines barres par rapport à d'autres.

Motif Sinus 2

```
let z0 = 0;
let z1 = 0;
let z2 = 0;
let y = 0;

function draw() {
  background(0);
  noStroke();

push();
rotateX(180);
fill(200, 190, 140);
translate(0, y, z1);
cylinder(10,180);
pop();

push();
rotateX(180);
fill(200, 190, 140);
translate(-50, y, z1);
cylinder(10,180);
pop();

push();
rotateX(180);
fill(200, 190, 140);
translate(-100, y, z1);
cylinder(10,180);
pop();

```

```
  push();
  rotateX(180);
  fill(200, 190, 140);
  translate(-25, y, z2);
  cylinder(10,180);
  pop();

  push();
  rotateX(180);
  fill(200, 190, 140);
  translate(-75, y, z2);
  cylinder(10,180);
  pop();

  if(z1 === -40){
    z1 = z1 + 80;
  }
  else {
    z1 = z1 - 1;
  }

  if(z2 === 40){
    z2 = z2 - 80;
  }
  else {
    z2 = z2 + 1;
  }
   }
```

Pour ce qui est du Motif Sinus2 le procédé est le même que le Sinus1 avec les valeurs qui changent pour la position des barres, pour pouvoir créer un pattern de mouvement différent.

Après avoir créé les barres et les motifs, il m'a fallu associer leur mouvement aux boutons de l'interface pour que le motif choisi soit réalisé par les barres. Puis ensuite transmettre les informations pour le mouvement des barres au serveur. Cependant je n'ai toujours pas réaliser cette partie, je suis actuellement entrain de la faire et ne peut donc pas la présenter dans ce rapport. Je vais donc présenter les différents problèmes et difficultés que j'ai pu rencontrer durant la réalisation de mon projet, je vais tout d'abord commencer par la configuration de la raspberry, ensuite le serveur, puis le client et enfin l'affichage 3D.

Pour ce qui était de faire la configuration de la raspberry et du paramétrage de celle-ci pour lancer le serveur automatiquement, je n'ai pas eu de grande difficulté, le seul point qui m'a posé problème c'est que je ne savais pas comment la configurer, installer l'OS Raspberry sur la carte Raspberry PI, la paramétré pour lancer un serveur automatiquement, j'ai donc dû rechercher sur internet comment réaliser ces différentes tâches pour avoir une raspberry Pi correctement opérationnel, cependant cela à été assez compliqué de trouver des tutos corrects qui en les suivants me donner le rendus que je voulais.

Pour le serveur Web c'est l'une des parties sur laquelle j'ai eu le plus de difficultés, car c'est la partie qui m'a demandé le plus de recherche possible pour comprendre le fonctionnement d'un serveur web et d'une communication client/serveur, ainsi que de l'apprentissage sur le langage Java, lors de mes recherches je me suis efforcé à comprendre du mieux que possible le fonctionnement d'un serveur web et d'une communication client/serveur, pour cela je me suis beaucoup documenté et regardé beaucoup de vidéo expliquant comment créer cela en java sur processing.

Grâce à ses recherches j'ai pu grandement m'inspirer par rapport aux différents programmes que j'avais vu sur internet, la partie pour créer l'interface du serveur à été la partie la plus simple lors de la programmation du serveur car j'ai facilement compris comment faire une interface sur processing avec les différentes explications présente sur le site officiel, ce qui ma demandé le plus de travail c'est la classe Communication où le premier problème à été la récupération des informations depuis le client car il fallait que je réalise cette classe en parallèle du client et que j'arrive à associer les données envoyé et reçu depuis le client, pour réussir cela j'ai eu l'aide de mon professeur qui ma montré la marche à suivre pour réaliser la réception et l'envoie des données que ce soit sur le serveur où le client, ensuite le second problèmes à été de relier la classe Communication et les informations reçu depuis cette classe au reste du programme.
Pour cela j'ai trouvé une vidéo sur youtube qui explique parfaitement comment relier une classe et des fonctions à un autre programme, ce qui m'a permis de réussir et de finir enfin le serveur Web.

Et donc au final j'ai pu correctement comprendre le rôle d'un serveur Web et comment fonctionner une communication client/serveur et donc la nécessité qu'il était de créer un serveur pour ce projet et également la nécessité d'utiliser une communication en Websocket lorsque que le client est une

application Web, j’ai d’ailleur préciser le fonctionnement et l’utilité de ces deux outils précédemment dans mon rapport.

Maintenant que j’ai expliqué quelles ont été les difficultés auxquelles j’ai pu faire face pour la partie serveur et communication, je vais maintenant présenter les difficultés et problèmes que j’ai rencontrés pour la création du client en javascript.

Ce qui m’a posé problème lorsque j’ai débuté la création du client c’est le fait que je n’avais aucune connaissance en langage javascript, cependant j’ai pu résoudre ce problème assez rapidement en suivant des tutos sur youtube de la chaîne “The Coding Train”, qui est une chaîne proposant de nombreux tuto très instructifs sur la bibliothèque p5.js que j’ai du utiliser pour créer l’interface web et je me suis également aidé du site p5js.org qui regroupe toute les fonctionnalité et commandes utilisables avec la bibliothèque p5.js, j’ai donc pu comprendre rapidement les bases du langages. Ce qui m’a ensuite posé problème c’est la partie pour pouvoir communiquer avec le serveur, lorsque je me suis trouvé confronté pour la première fois à la création de la communication serveur/client depuis le client je ne savais pas par quoi commencer et de quel manière je devais réaliser cela.

Pour résoudre mon problème j’ai donc fait des recherches sur comment faire pour envoyer et recevoir des informations depuis un client en javascript, j’ai donc lu plusieurs document expliquant quels type de commandes je pouvais utiliser et leur utilité, mais j’ai aussi regardé des vidéos expliquant comment procéder pour utiliser les commandes nécessaire, j’ai également eu l’aide de mon professeur pour m’expliquer comment structurer mon programme avec c’est commandes que j’avais apprise, j’ai pu donc grâce à ces différent support résoudre mon problème concernant l’envoie et la réception d’information depuis un client en javascript.

Je vais donc maintenant parler des difficultés que j’ai rencontrées pour la simulation 3D des barres du mobile.

Pour cette partie de mon projet le premier problème que j’ai rencontré à été lorsque je souhaitai faire bouger les barres de façon à ce qu'elles réalisent un mouvement prédéfinie en boucle, pour cela j’ai tout d’abord essayé d’utiliser les conditions “if(), while() et for()” qui sont des condition permettant de créer une boucle. Pour que les positions de chaque barre soit incrémenter en atteignant une certaine valeur puis décrémenter an en atteignant une autre,

cependant les conditions “while() et for()” ne proposait pas la manière de faire que je chercher et j’ai donc dû me rabattre sur la condition “if()”, cependant lorsque je voulais incrémenter et décrémenter les positions cela s’annuler en effet je voulais incrémenter de 1 et de décrémenter de -1, je n’ai pas trop compris le problème cependant lorsque j’ai fait cela les barres ne bougeait pas. J’ai donc à la place définie une position à laquelle les barres doivent retourner puis être incrémenter ou décrémenter.

Ensuite il y a aussi la création des classes barre et mouvement, pour créer c’est classe j’ai du mal car je ne comprenais pas le fonctionnement d’une classe et son but précis, mais avec du travaille sur la compréhension du fonctionnement d’une classe en regardant plusieurs vidéos j’ai finis par correctement comprendre le comment fonctionne une classe et ai donc pu créer ces deux classes. Voilà donc pour les problèmes et difficultés rencontrés durant mon projet et ce qui conclut ce rapport sur le projet du mobile que j’ai eu à réaliser durant cette deuxième années de bts. Et pour conclure ce rapport je dirais que travailler sur ce projet m’a beaucoup appris au niveau de la programmation Web, notamment avec l’apprentissage de nouveau langages informatique, la compréhension de l’utilité et du fonctionnement d’un serveur web et également de l’apprentissage sur le fonctionnement d’une communication serveur/client, tous ces apprentissage vont me permettre d’avoir dans le futur une meilleur approche de ces différents thèmes et me permettront d' être plus efficace dont la formation en développement web que je souhaite réaliser après mon bts.

Liens des sources utilisée :

UML (informatique) — Wikipédia (wikipedia.org)

Diagramme de classes — Wikipédia (wikipedia.org)

Appliquez le principe du Domain-Driven Design à votre application - OpenClassrooms

https://raspberry-pi.fr/installer-raspbian-premier-demarrage-configuration/
Guide complet : Mise à l'Heure Automatique d'un Raspberry Pi – Raspberry Tips
raspberry-pi - Comment mettre à Jour la Date et l'Heure de la Raspberry Pi Avec Internet (askcodez.com)

https://pi.processing.org/download/

https://www.raspberrypi.org/forums/viewtopic.php?t=187613

https://www.raspberrypi.org/forums/viewtopic.php?t=153499

https://raspberry-pi.fr/executer-programme-demarrage/#:~:text=Pour%20lancer%20un%20programme%20au,n%27ai%20fini%20de%20booter.

https://raspberry-projects.com/pi/pi-operating-systems/raspbian/scripts

https://discourse.processing.org/t/executing-sketch-via-linux-command-line/6839

https://github.com/processing/processing/wiki/Command-Line

https://www.youtube.com/watch?v=IVBhd3gQc2o

WebSocket — Wikipédia (wikipedia.org)

WebSockets - Référence Web API | MDN (mozilla.org)

Socket java : Créer une application de chat Client/Serveur (codeurjava.com)

Passez à l'échange de données en temps réel avec WebSockets (journaldunet.com)

https://www.journaldunet.com/web-tech/developpeur/1147869-passez-a-l-echange-de-donnees-en-temps-reel-avec-websockets/

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications

https://developer.mozilla.org/en-US/docs/Web/API/WebSocket/onmessage

https://www.ionos.fr/digitalguide/sites-internet/developpement-web/quest-ce-que-le-websocket/

https://openclassrooms.com/forum/sujet/java-websocket

https://openclassrooms.com/forum/sujet/websocket-js-client

https://javascript.developpez.com/actu/83882/L-API-Websockets-ce-que-c-est-et-comment-l-utiliser/

https://www.youtube.com/watch?v=o8Fn9BgqoQM&t=529s

https://www.youtube.com/watch?v=FduLSXEHLng

https://www.youtube.com/watch?v=2Nt-ZrNP22A

https://p5js.org/reference/

https://p5js.org/reference/#/p5/function

https://p5js.org/reference/#/p5/class

https://p5js.org/reference/#/p5/cylinder

https://p5js.org/reference/#/p5/rotate

https://p5js.org/reference/#/p5/translate

https://p5js.org/reference/#/p5/createCanvas

https://p5js.org/reference/#/p5/createGraphics

https://p5js.org/reference/#/p5/loadImage

https://p5js.org/reference/#/p5/createImage

https://p5js.org/reference/#/p5/loadFont

https://p5js.org/reference/#/p5/textFont

https://www.youtube.com/watch?v=nqiKWXUX-o8&list=PLRqwX-V7Uu6bPhi8sS1hHJ77n3zRO9FR_

https://www.youtube.com/watch?v=6TPVoB4uQCU&list=PLRqwX-V7Uu6bPhi8sS1hHJ77n3zRO9FR_&index=2

https://www.youtube.com/watch?v=k2FguXvqp60&list=PLRqwX-V7Uu6bPhi8sS1hHJ77n3zRO9FR_&index=3

https://www.youtube.com/watch?v=O1mYw-3WI_Q&list=PLRqwX-V7Uu6bPhi8sS1hHJ77n3zRO9FR_&index=4

https://www.youtube.com/watch?v=3tTZITq4Cxs&list=PLRqwX-V7Uu6bPhi8sS1hHJ77n3zRO9FR_&index=6

https://developer.mozilla.org/fr/docs/Learn/Getting_started_with_the_web/JavaScript_basics

https://openclassrooms.com/fr/courses/6173501-debutez-la-programmation-avec-java

https://www.jmdoudoux.fr/java/dej/chap-net.htm

https://gfx.developpez.com/tutoriel/java/network/

https://processing.org/tutorials/data/

https://processing.org/tutorials/text/

https://processing.org/reference/

https://github.com/alexandrainst/processing_websockets