



# // CAHIER DE RECHERCHES //

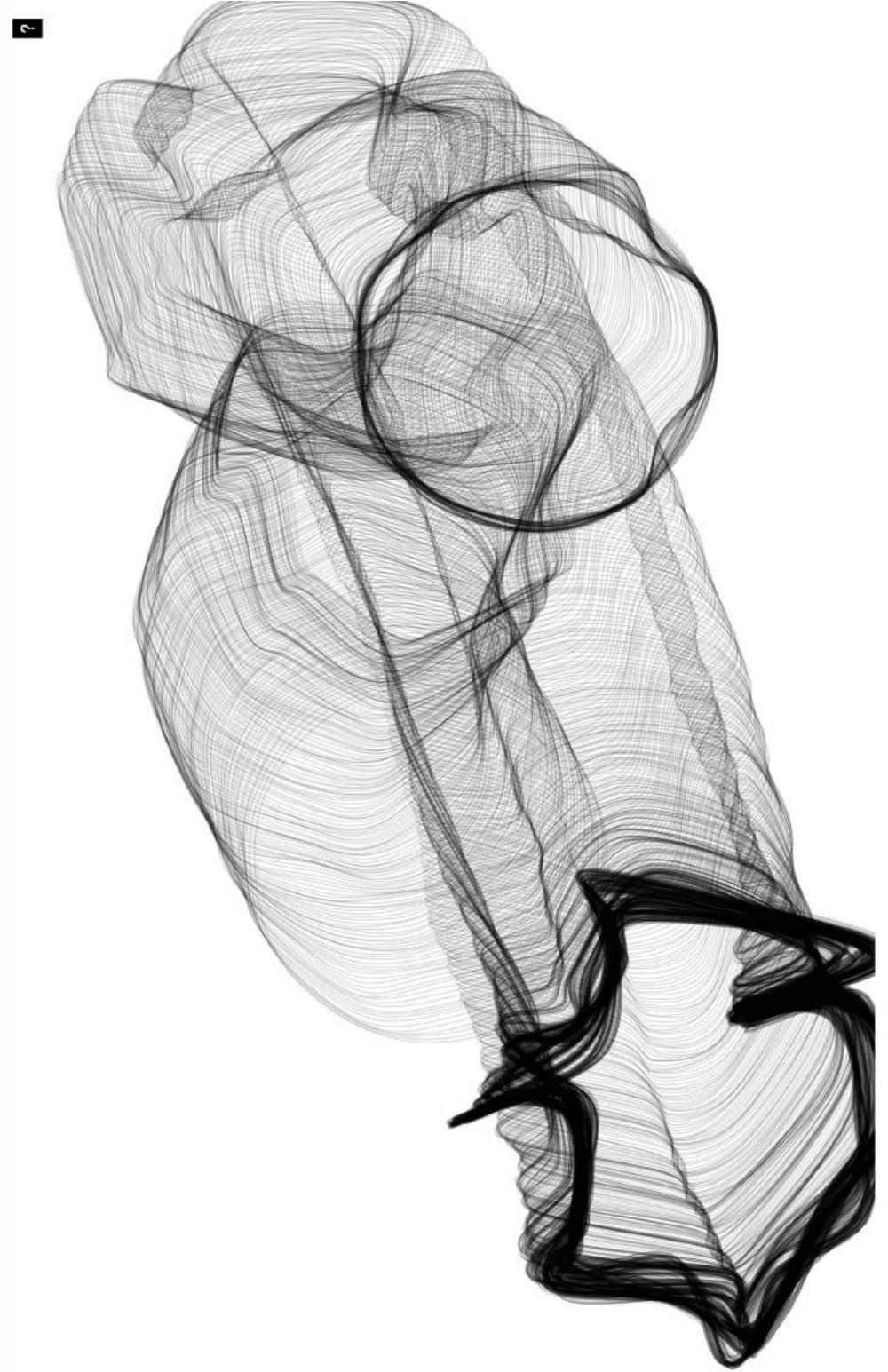
**Robin Exbrayat**

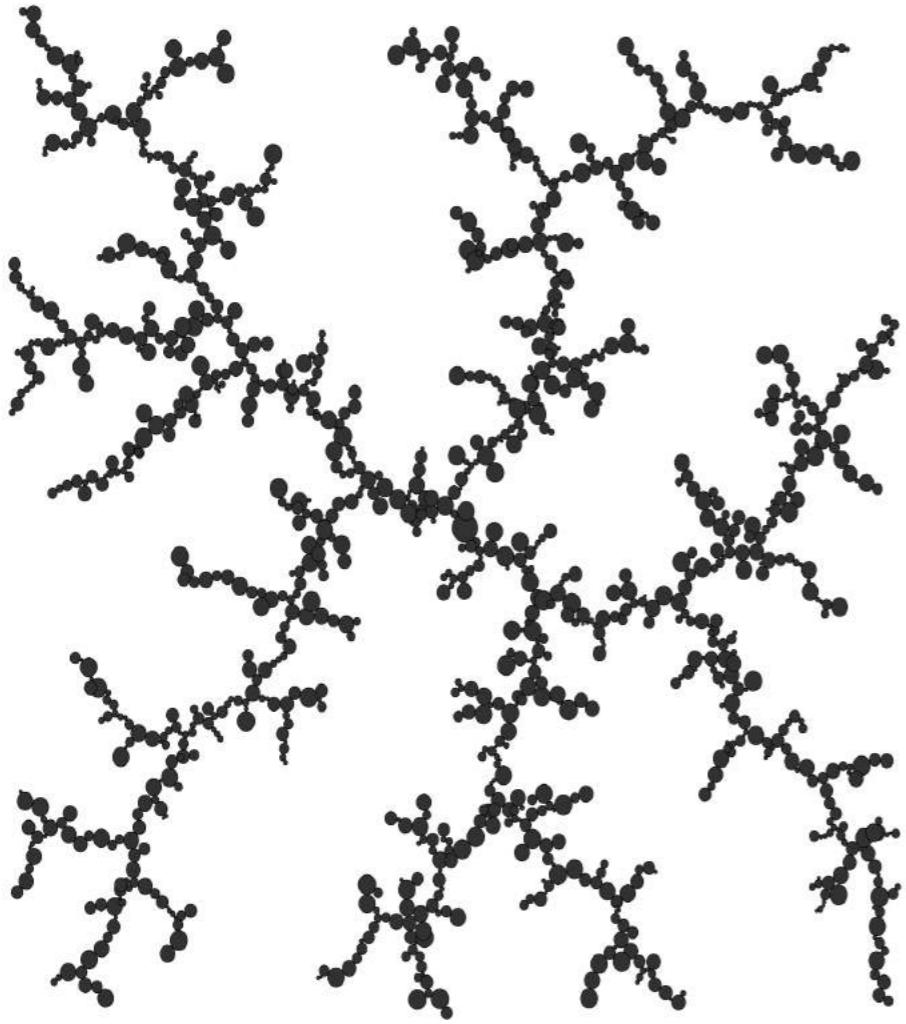
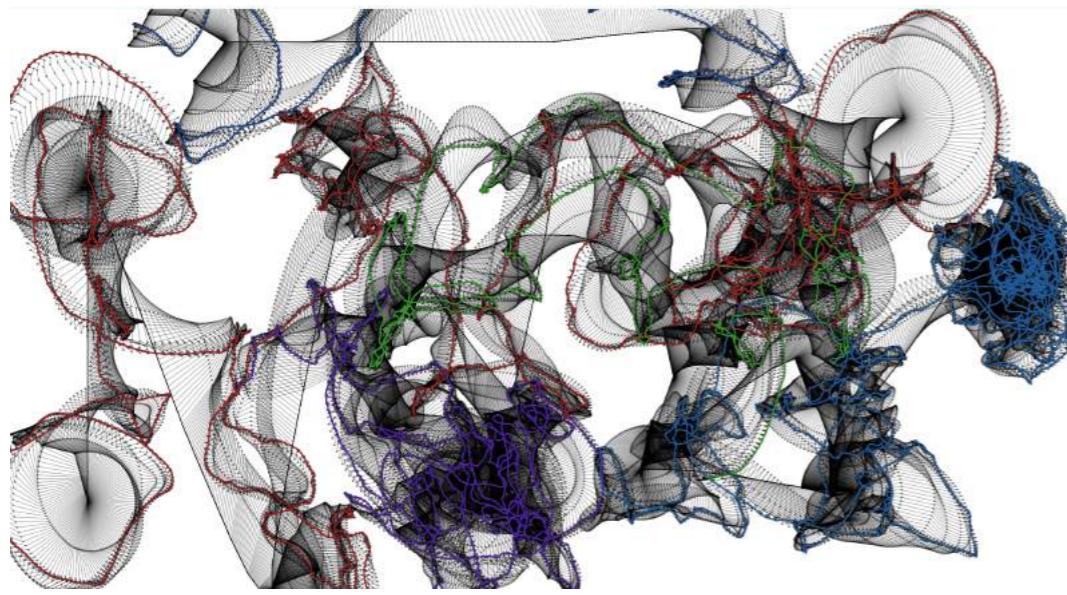
**Elliot Chevalier**

**Raphaël Perraud**

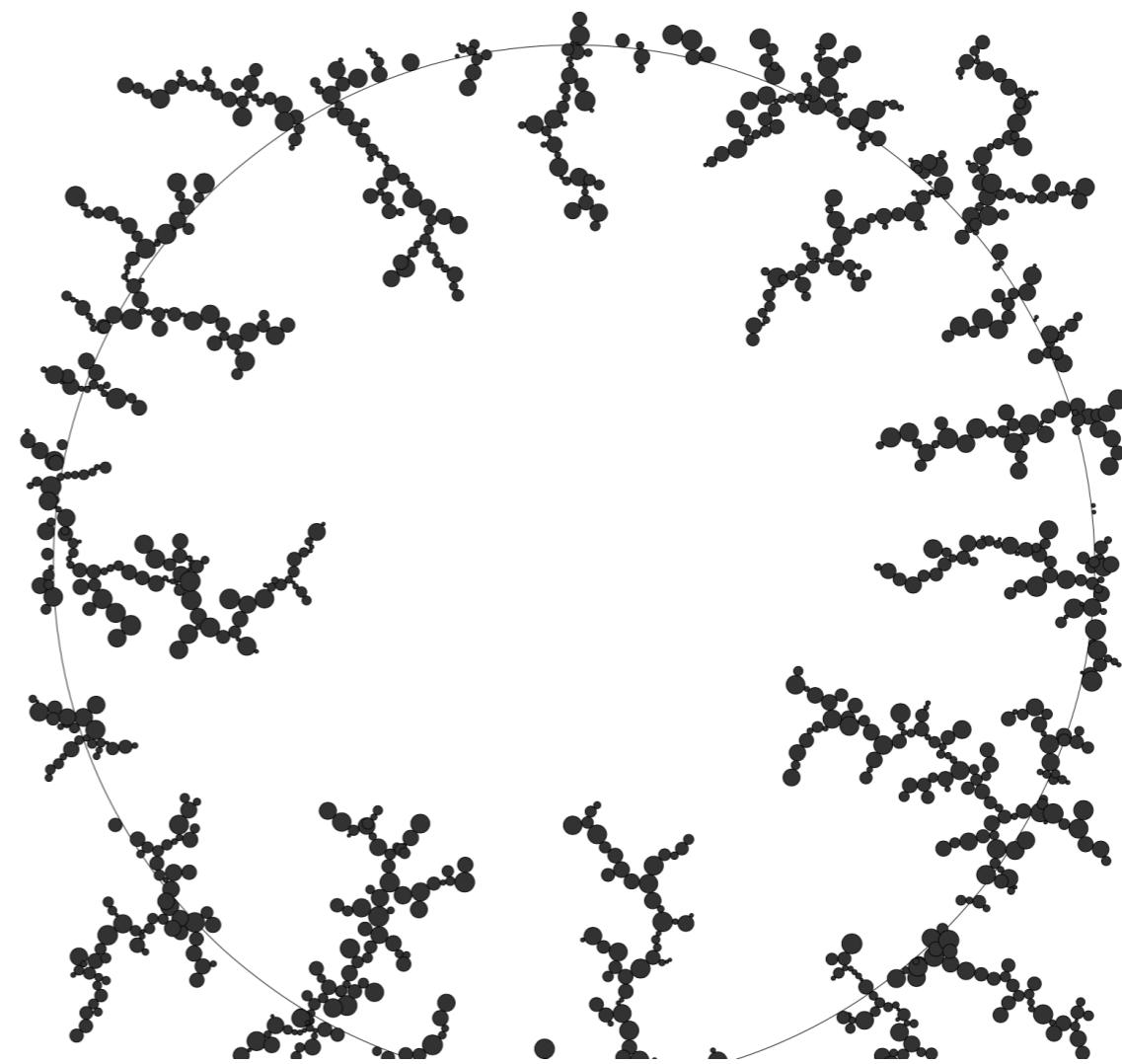
**Theo Geiller**

# // VEILLE // Patterns



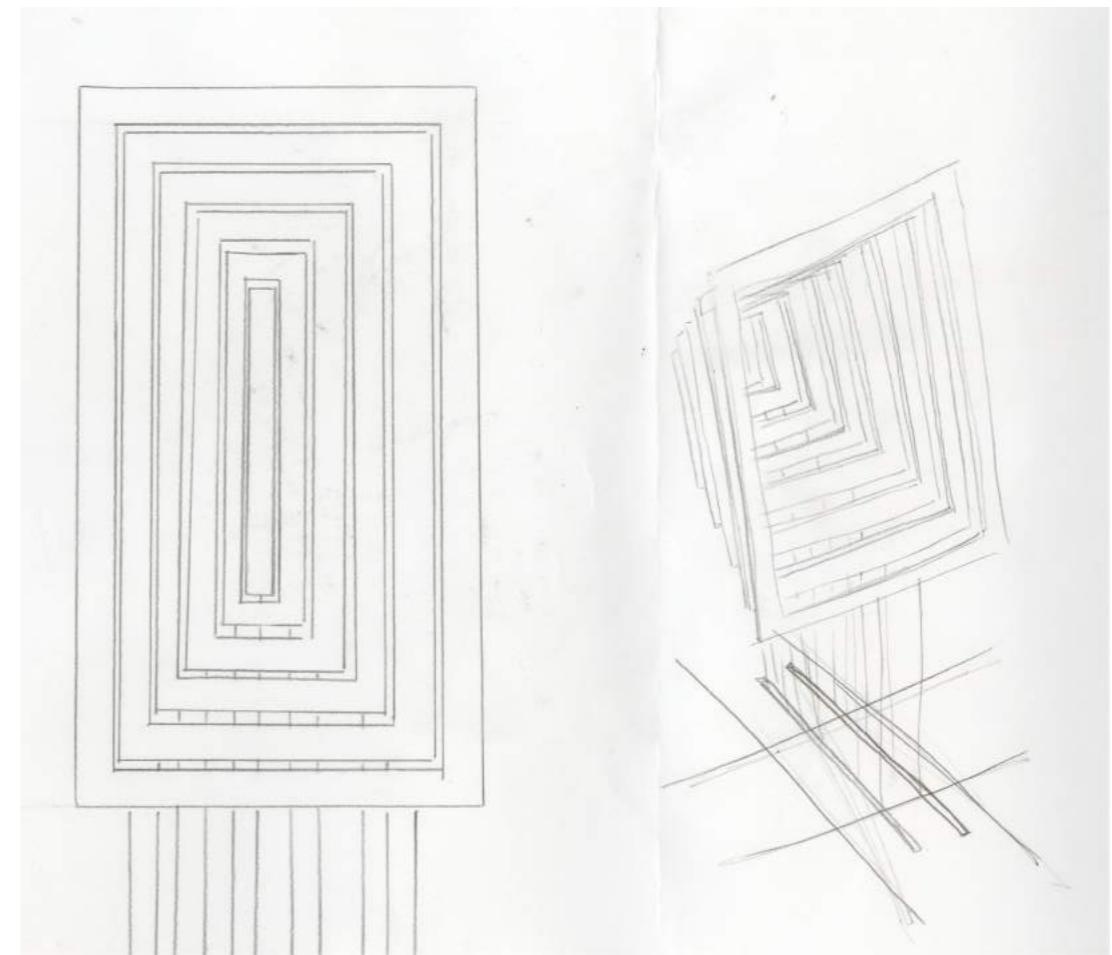
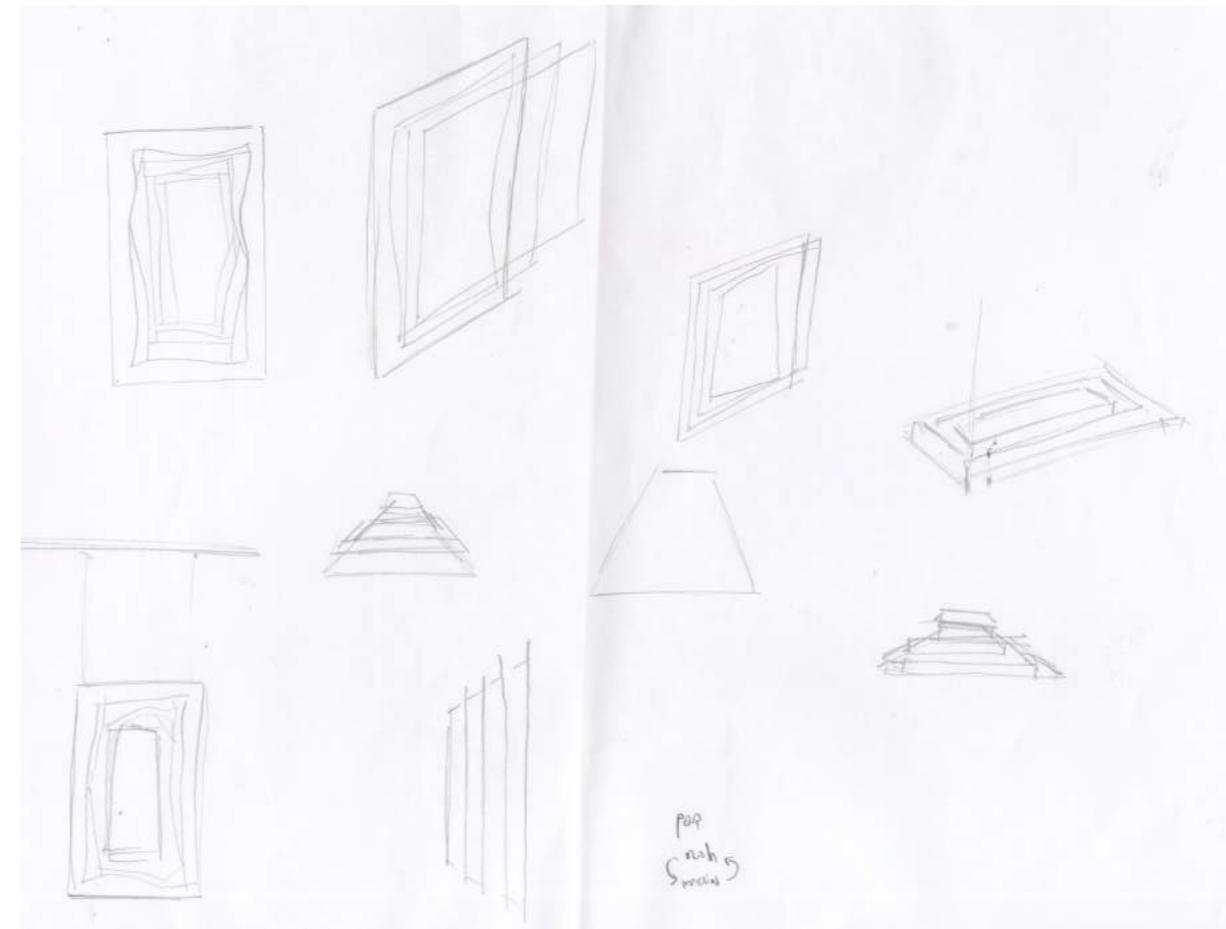
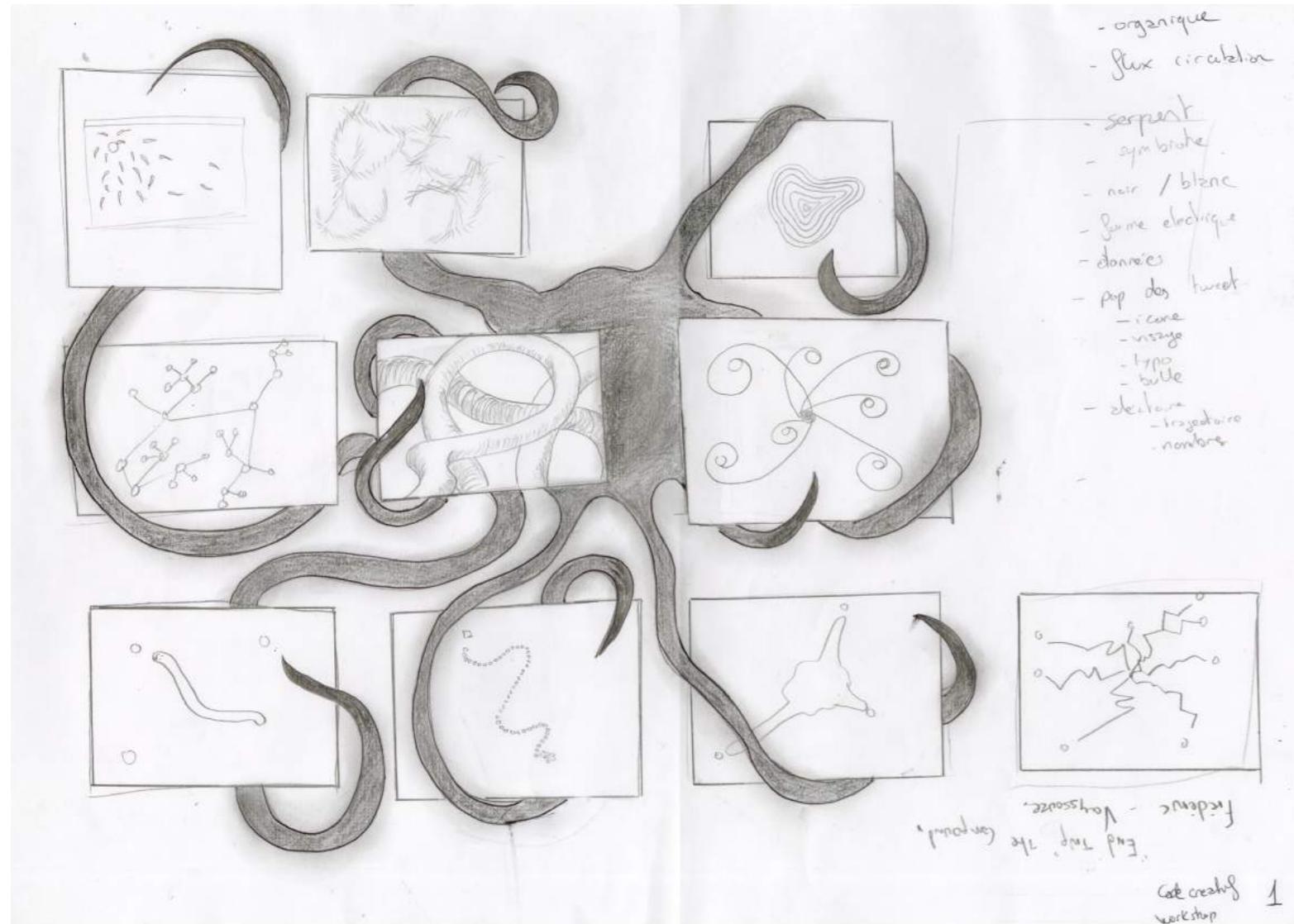


// Nous voulons prendre en compte l'évolution, le déplacement autonome, la progression dans notre travail. Cette notion correspond bien à notre univers //



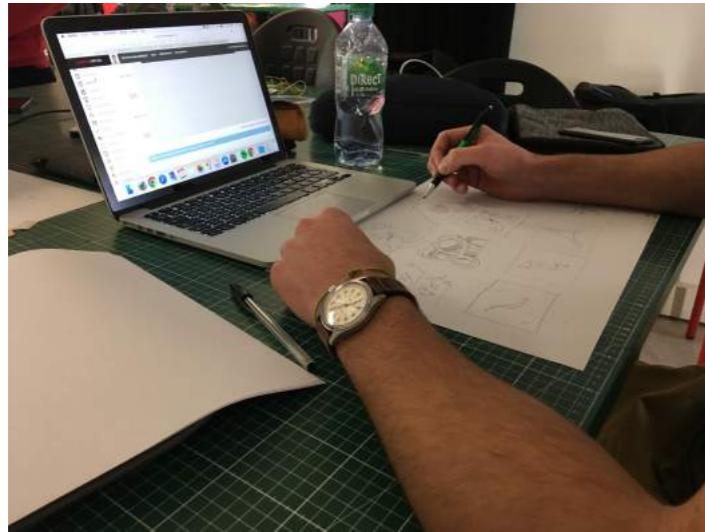
// Les formes et les graphismes générés par ces codes peuvent faire penser à de la data qui évolue//

# //SKETCHS //



//Premières idées posées au crayon sur une feuille, Mind map Schéma + essais scéno définition de l'univers //

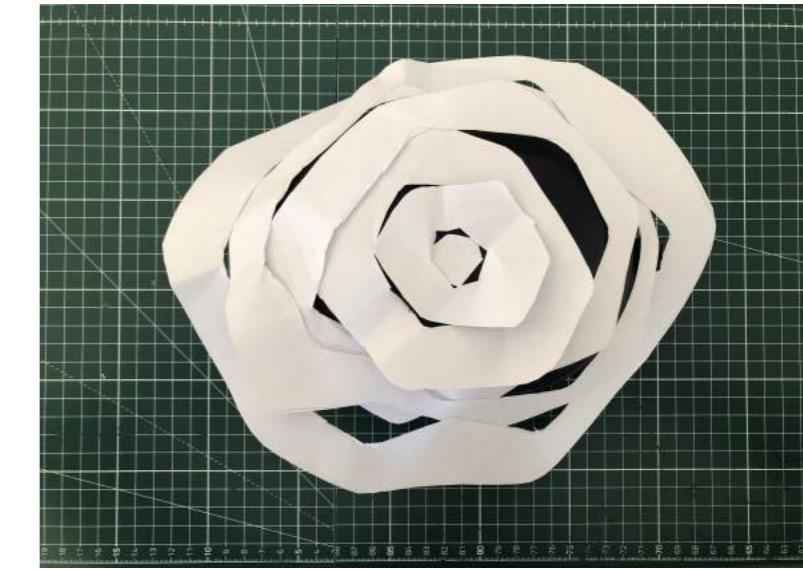
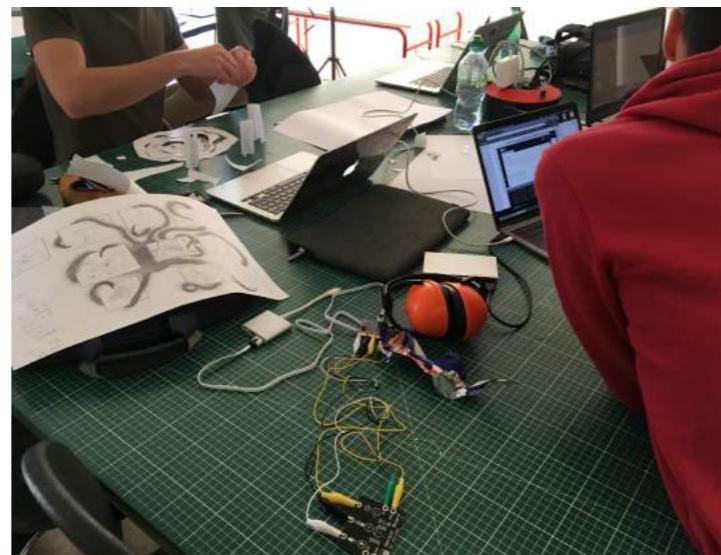
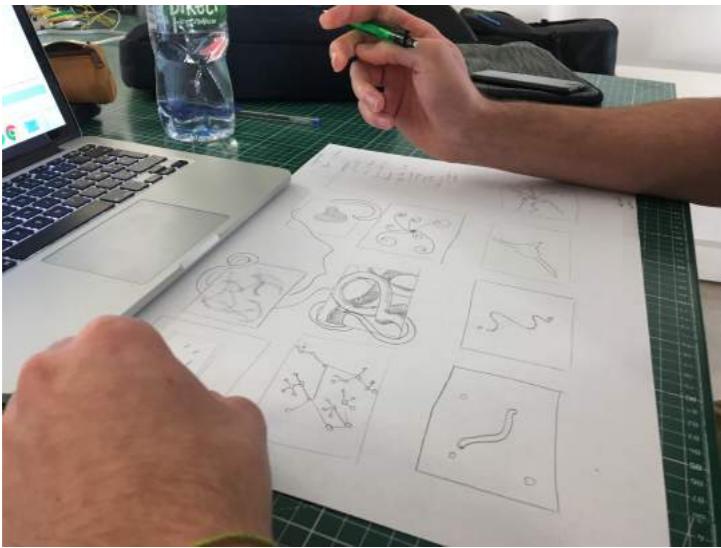
# // CARNET DE BORD //



// -Recherche d'idées, beaucoup de réflexion à l'oral.

-Transmettre l'idée d'effet de masse ( 1 information / notification attire l'attention )

-Premières idées : travailler autour de la data, Snake mangeur / agents. //

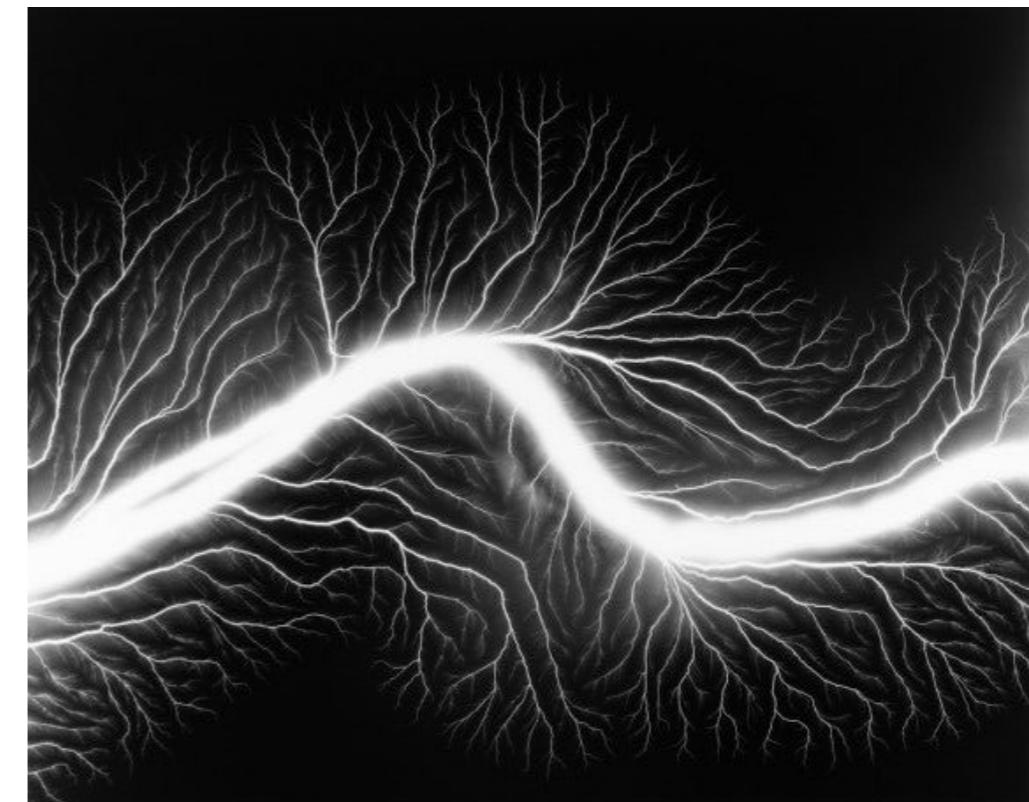
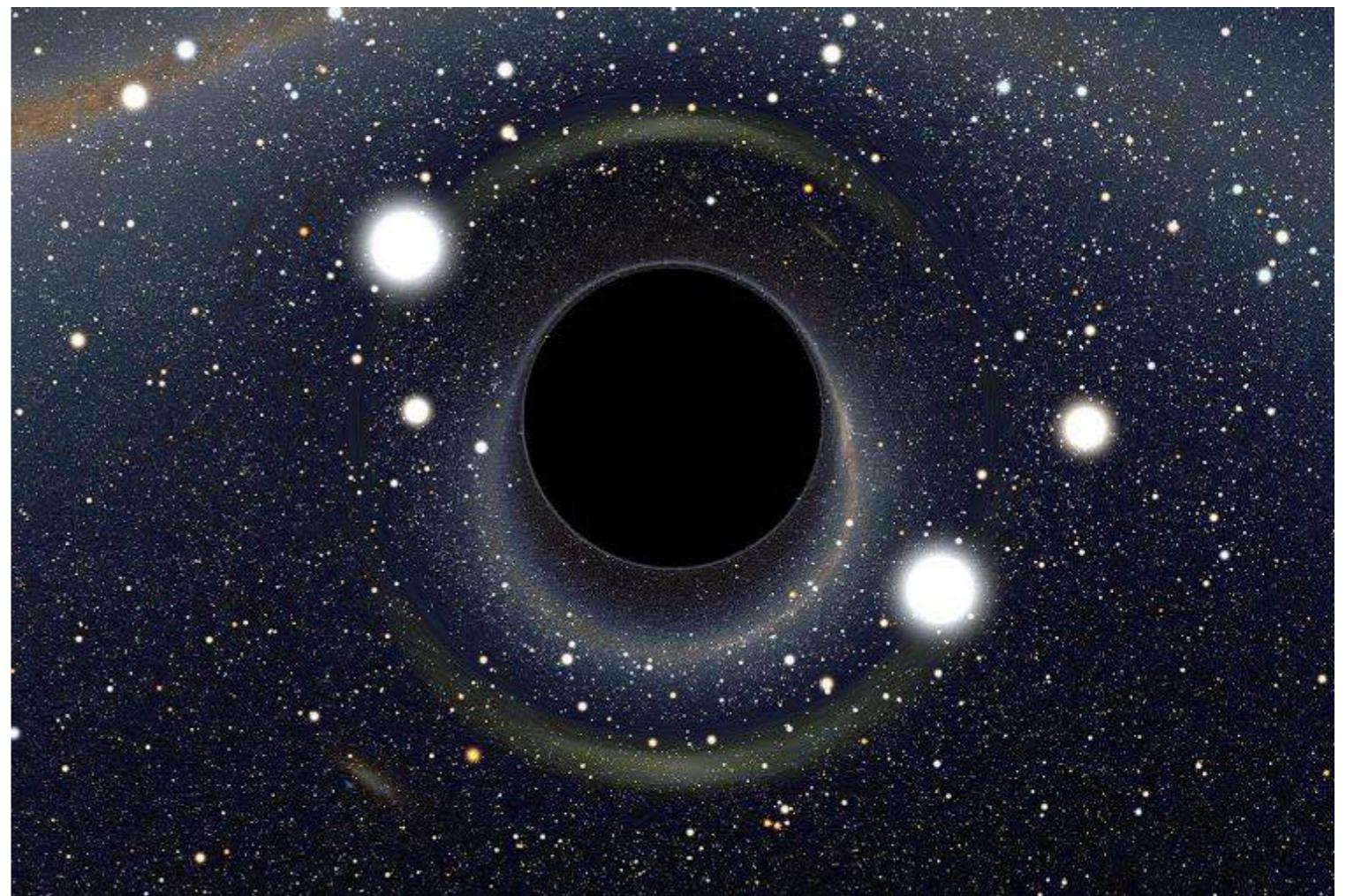


// -Recherches autour de l'univers et de l'identité du projet.

-Découverte du code pour récupérer les de API Twitter/ Instagram/ Facebook/ Snapchat  
Carnivore + OSC Hmap + nombre d'adresses IP.

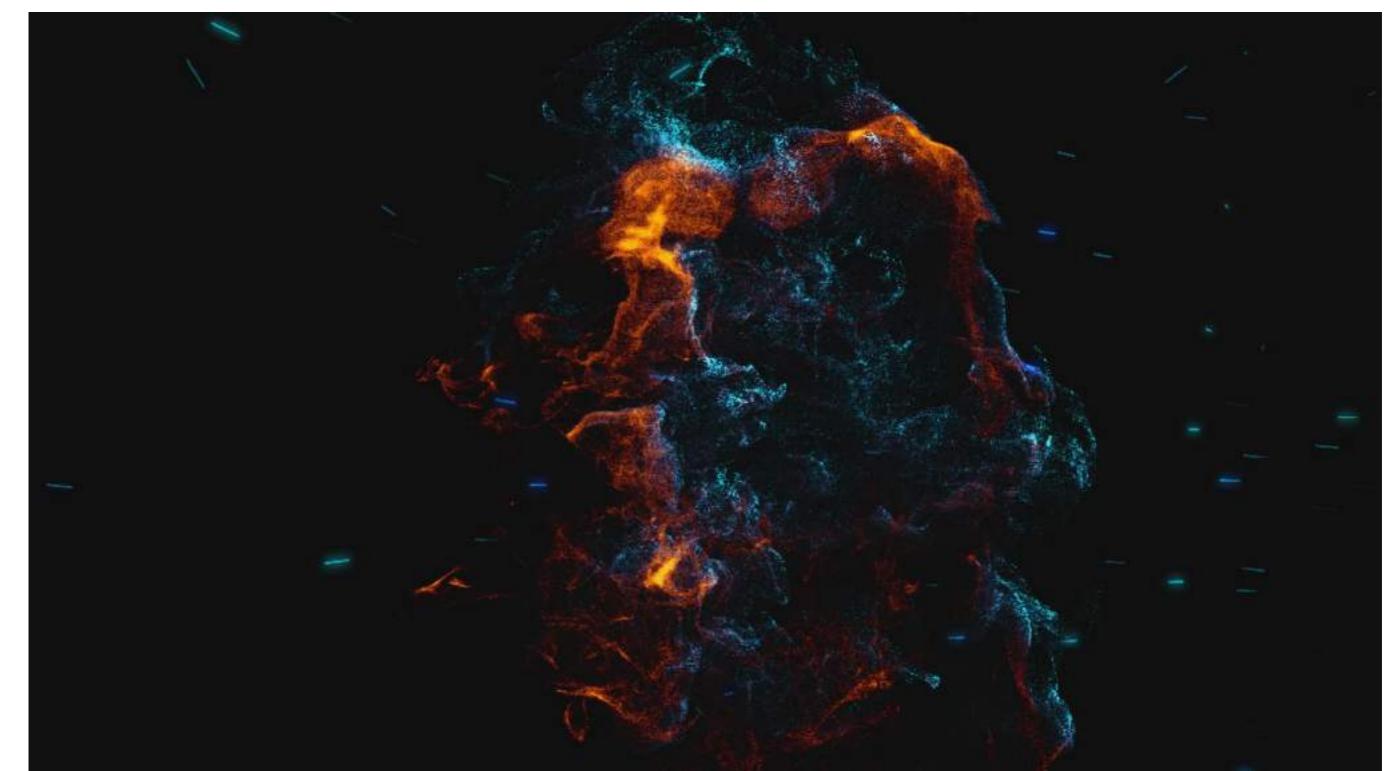
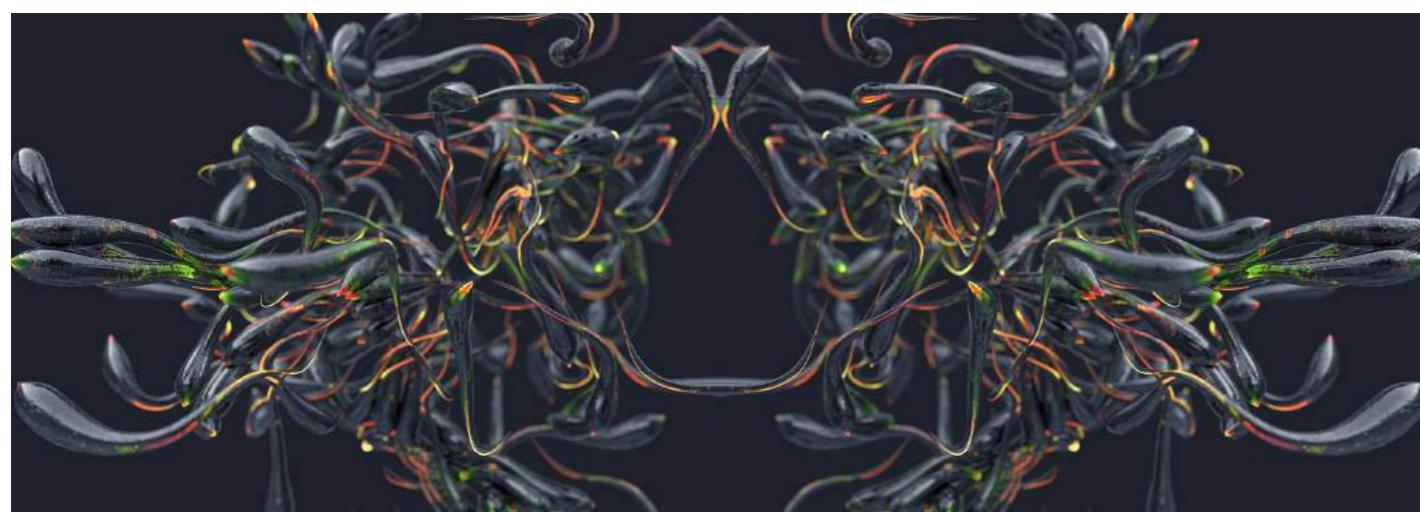
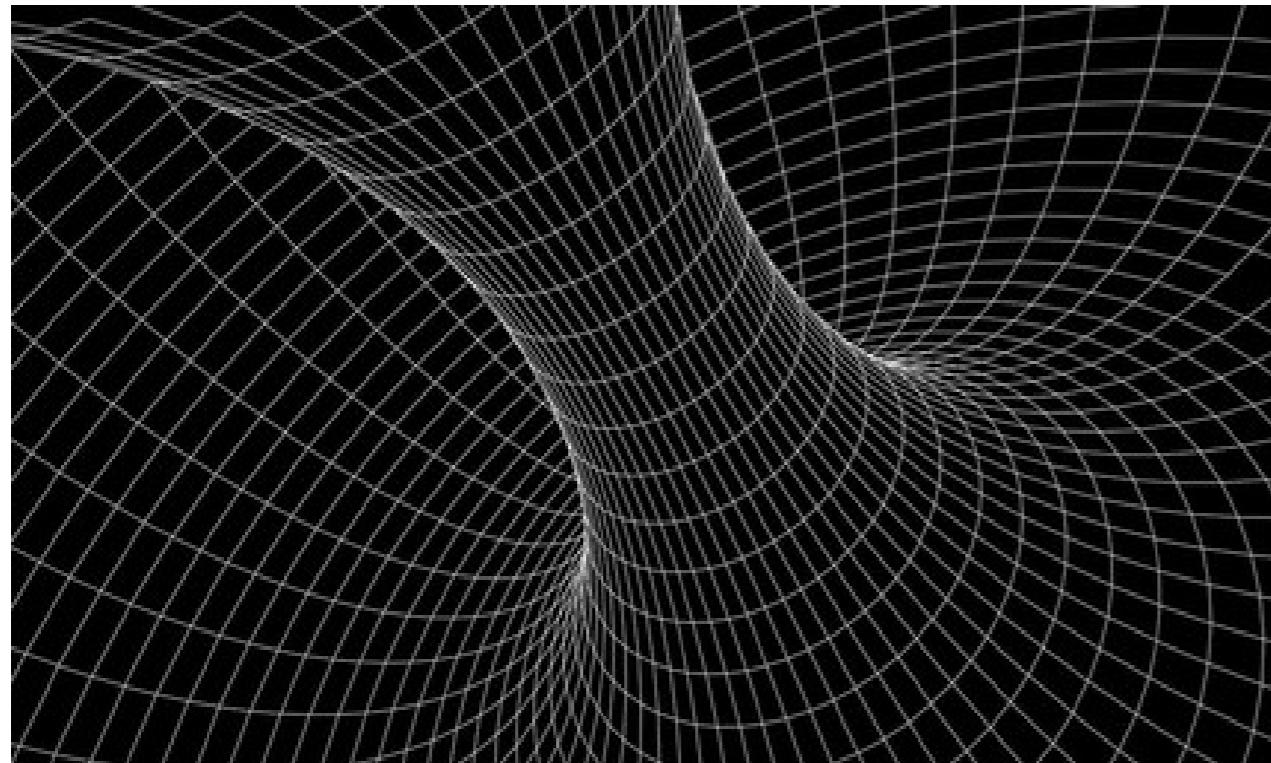
-Premières idées : travailler autour de la data, Snake mangeur / agents. //

# Univers



// Nous avons opté pour un univers sombre, en noir et blanc .  
Inspiré de jeux comme Limbo ou du trou noir et des particules //

# Univers



// Particules, trou noir, trou de ver, effets optiques  
mouvements aléatoires //

# // PISTES //

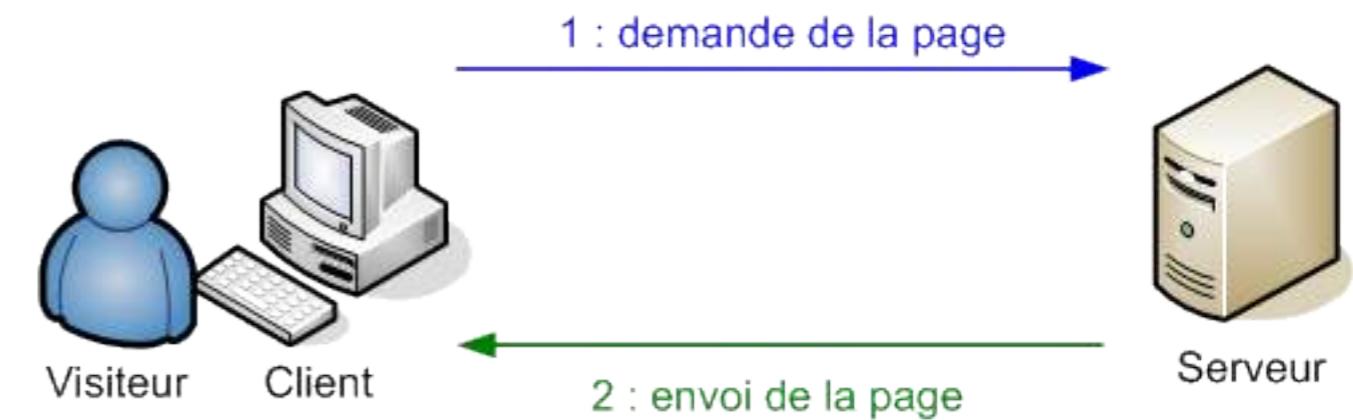


CARNIVORE IS A SURVEILLANCE TOOL FOR DATA NETWORKS.  
AT THE HEART OF THE PROJECT IS CARNIVOREPE, A SOFTWARE APPLICATION THAT LISTENS TO ALL INTERNET TRAFFIC (EMAIL, WEB SURFING, ETC.) ON A SPECIFIC LOCAL NETWORK. THEN CARNIVOREPE SERVES THIS DATA STREAM TO INTERFACES CALLED "CLIENTS." THESE CLIENTS ARE DESIGNED TO ANIMATE, DIAGNOSE, OR INTERPRET THE NETWORK TRAFFIC IN VARIOUS WAYS.

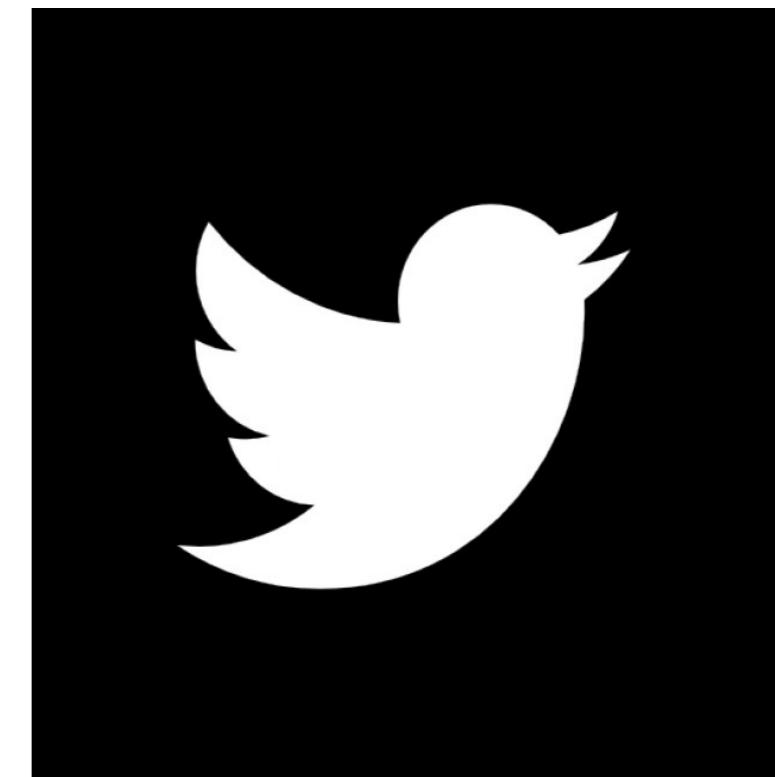
USE CARNIVOREPE TO RUN CARNIVORE CLIENTS FROM YOUR OWN DESKTOP, OR USE IT TO MAKE YOUR OWN CLIENTS.

DOWNLOAD CLIENTS REFERENCE SUPPORT ABOUT

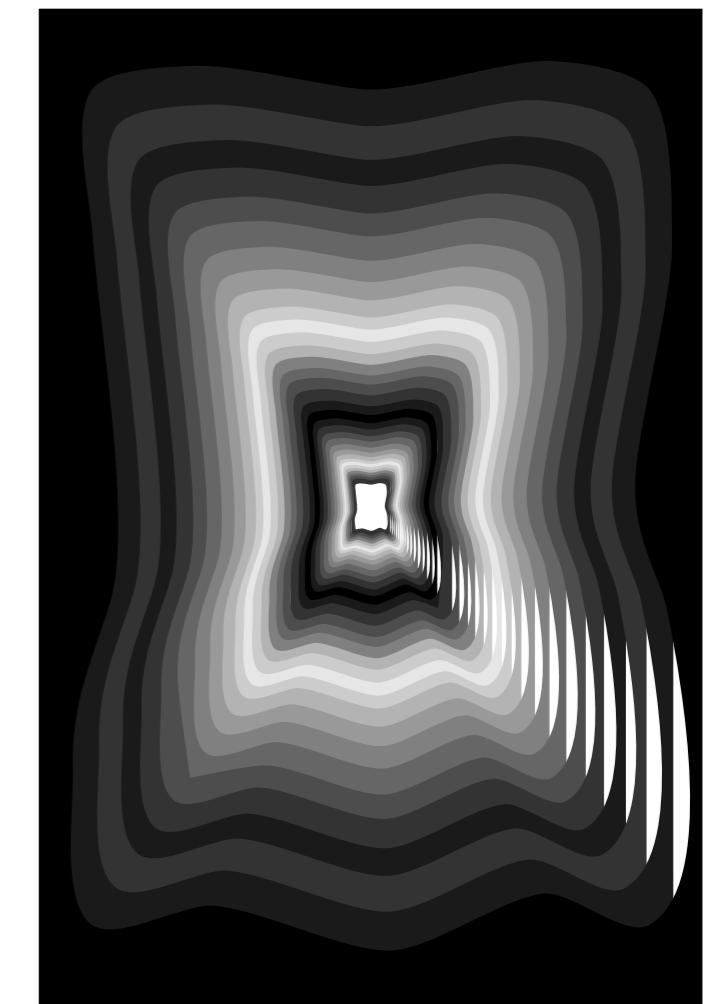
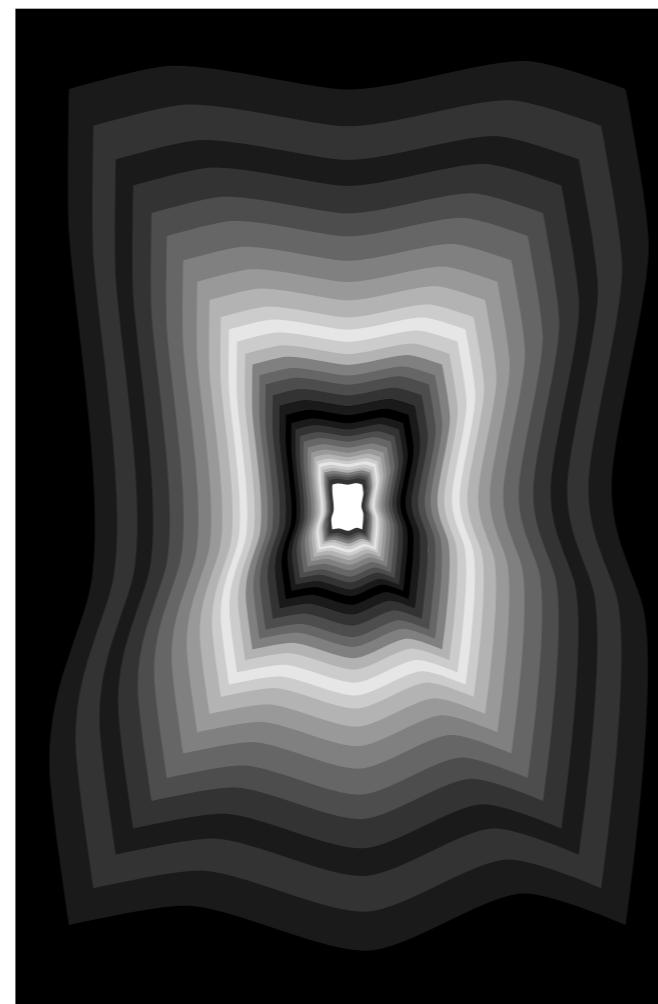
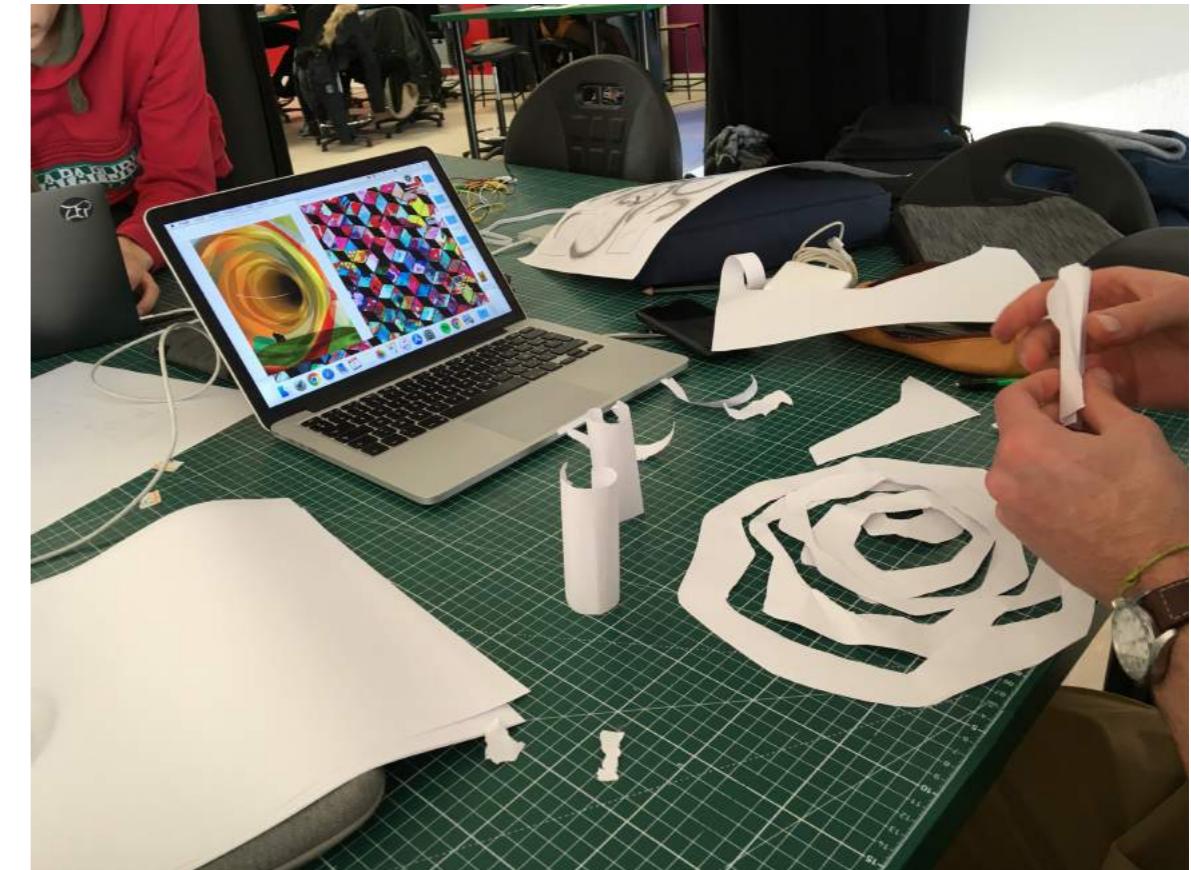
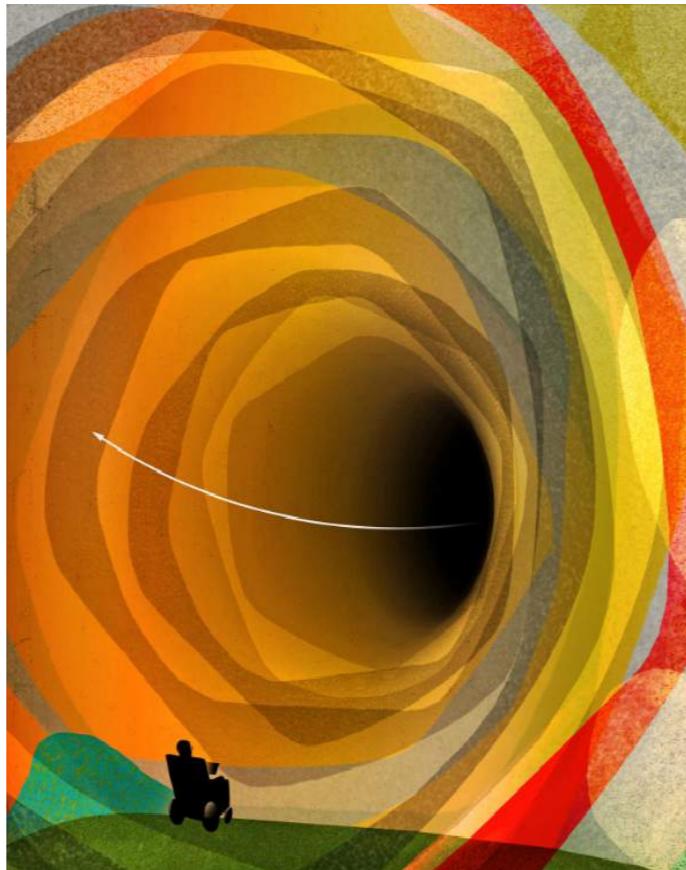
Built using the Carnivore Library for Processing | View files and source code



// La Première piste consistait à utiliser la bibliothèque processing carnivore et à créer un «Snake» qui se servirait de la data pour grossir. //



// Idée de layers et de profondeur. Projeter le code sur une surface à plusieurs couches pour créer de la profondeur //



# // PROJET // Sound



// Tous les sons du projet ont été créés avec Ableton 9 L'univers est inspiré de Slender man et de l'Ender Man de Minecraft //

# DAY ONE : LIBRAIRIE OSC

Journée consacrée au test des librairies Osc et Carnivore

```
osc = new OscP5(this, 8000); //start liste  
port = new NetAddress("127.0.0.1", 9000);
```

Test avec la bibliothèque Osc qui permet de faire communiquer deux appareils, donc potentiellement d'écouter l'échange. On déclare un port de transit des paquets et l'IP visée

```
void sendMessage() {  
    activity = "sending: ";  
    value = mouseX;  
  
    OscMessage myMessage = new OscMessage("/1/fader3");  
    myMessage.add(value);  
    osc.send(myMessage, port);  
}
```

Les exemples de la bibliothèque ont une fonction ajoutant à la String Message le contenu, et l'envoie sur le port défini auparavant. On a testé en s'échangeant les positions de souris pour dessiner des cercles sur un autre terminal.

```
/* incoming osc message are forwarded to the oscEvent method  
void oscEvent(OscMessage theOscMessage) {  
    // only react on this addresspattern  
    if (theOscMessage.checkAddrPattern("/1/fader3")==true) {  
        // check if the value is a float  
        if (theOscMessage.checkTypeTag("f")) {  
            activity = "receiving: ";  
            value = theOscMessage.get(0).floatValue();  
        }  
    }  
}
```

Les exemples fournissent aussi la fonction événementielle **oscEvent** comprise dans la library. Vérification du contenu en regardant les premiers caractères de la chaîne reçue. Si l'information est correcte, on parse le message reçu pour récupérer les données émises par l'autre sketch.

Sketch disponible sur le repo Github :

emetteur.pde

receveur.pde

[https://github.com/Theojkydbz/Workshopcodecreatif/  
tree/master/WORKSHOPDAY01](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/WORKSHOPDAY01)

## CONCLUSION :

Les possibilités de la library ne sont pas celles recherchées. Son usage nécessite un effort au spectateur et lui ajoute une couche de complexité que l'on voudrait éviter ( configurer sa connexion pour ça, chose que nombreux utilisateurs de smartphone ne savent pas faire ).

# DAY ONE : LIBRAIRIE CARNIVORE

Journée consacrée au test des librairies Osc et Carnivore

```
// Called each time a new packet arrives
void packetEvent(CarnivorePacket p) {
    //println("(" + p.strTransportProtocol + " packet) " + p.send
    if (tab.size() == 0) {
        tab.add(p.senderSocket());
    }
    for (int i=0; i <= tab.size(); i++) {
        // println(p.senderSocket() +"=" + (String)tab.get(i));
        if (p.senderSocket().compareTo((String)tab.get(i)) == 0) {
            println("okzopkzeza");
            break;
        }
        else{
            tab.add(p.senderSocket());
            println("pas là");
            println("longeur tableau = " + tab.size());
        }
    }
}
```

Test avec la bibliothèque Carnivore, plus complexe à utiliser. Elle permet de sniffer directement le réseau sur lequel on est connecté. Elle intègre même une fonction événementielle `packetEvent` qui est appelée à chaque émission de paquet. Elle renvoie une chaîne de caractères complexes mais complète.  
On a travaillé par dessus les résultats qu'elle nous renvoyait, et simplement testé avec des `print` quand est-ce que la conditionnelle était activée.

## CONCLUSION :

Beaucoup plus intéressante pour notre objectif, Carnivore peut permettre d'identifier le nombre d'appareils connectés, les nouvelles requêtes, leur contenu (crypté) et le port par lequel elles transitent, ce qui nous laisse la possibilité d'interpréter de quel type de requête il s'agit.

```
String[] parts = p.senderSocket().split(":");
String codec = p.ascii();

codecnumber+=codec.length();
println(codecnumber);

hm.put(parts[0], 1);

for (Map.Entry me : hm.entrySet()) {

    String value = me.getKey().toString();
    value.substring(0, 3);
    if (value.substring(0, 3).equals(testPrefix)) {
        n++;
        //print(me.getKey() + " is " + me.getValue());
    }
}
if (oldNumber < n) {
    println("Nouvelle connexion !");
    oldNumber = n;
}
println("Appareils connectés : ", n);
n = 0;
println("hm.size = ", hm.size());
```

Il faut donc parser ce que l'on reçoit. La chaîne reçue se définit selon la structure suivante :

[PROTOCOLE] : [IP] : [codec]

On s'intéresse uniquement à ceux qui utilisent le protocole **UDP** car `ascii()` permet de récupérer le codec du paquet  
`hm` stocke toutes les nouvelles IP.

On vérifie donc les métas ; sur un réseau wifi, on a pu observer que le préfixe des adresses «privées» (internes au réseau) était toujours **172**. Ca nous permet ainsi de différencier les requêtes émanant du réseau interne de celles émises par les appareils externes sondant l'environnement wifi. (voir page suivante)  
On incrémente donc notre compteur à connexions `n` à chaque nouvelle connexion en deux temps : puisque l'on a pas de moyens pour vérifier directement le nombre d'appareils connectés, on compte chaque requête émanant d'un interne, puis on stocke ce nombre dans `oldNumber` et reset `n`. Le reset de `n` est nécessaire car il s'incrémente à chaque requête sans distinction par rapport aux IP qui ont déjà été notées auparavant.

Sketch disponible sur le repo Github :

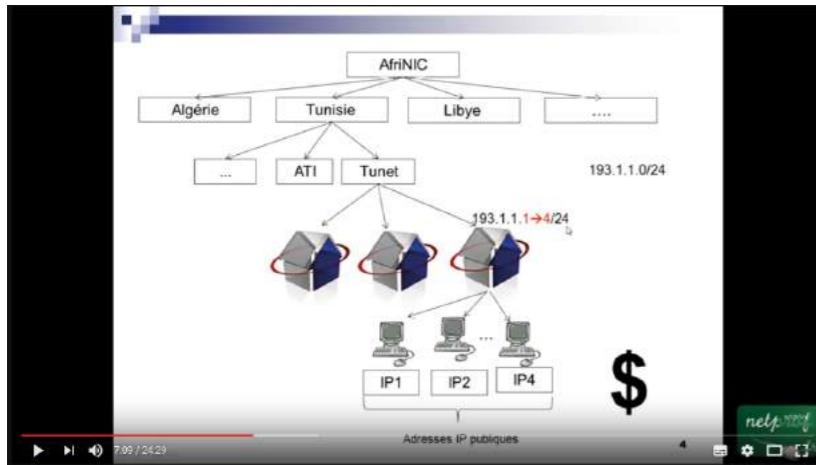
DAY01.pde

DAY01hashmap.pde

[https://github.com/Theojkydbz/Workshopcodecreatif/  
tree/master/WORKSHOPDAY01](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/WORKSHOPDAY01)

# DAY ONE : ADRESSES IP & RÉSEAUX

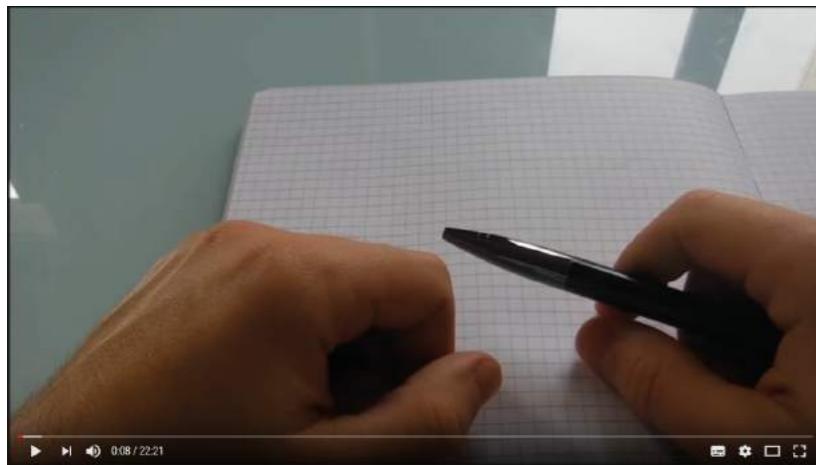
Journée consacrée au test des librairies Osc et Carnivore



[https://www.youtube.com/watch?v=i7v\\_URamto0&feature=share](https://www.youtube.com/watch?v=i7v_URamto0&feature=share)

En regardant les retours console, on s'est aperçus que certains «patterns» revenaient souvent dans les IP, on s'est donc documentés sur la façon dont elles étaient déterminées pour pouvoir différencier les appareils derrière.

Nous avons dû nous pencher sur la notion de masque de sous-réseau et comprendre ce que signifiait les valeurs d'une IP :  
L'addition (en binaire) du masque de sous réseau et de l'adresse IP permettent de retrouver l'adresse du réseau.  
Un réseau a une capacité maximale de sous-réseaux et d'appareils qui sont directement liées : plus un réseau laisse des places pour accueillir des appareils, moins il pourra créer de sous-réseau.  
Cela est simplement lié au fait que l'adresse du réseau est codée sur 4 octets, or la valeur maximale d'un octet est 255.  
L'adresse IP d'un appareil est donc séparée en deux parties : l'une qui est identique pour tous les appareils connectés, et l'autre qui permet au réseau de différencier chaque terminal.



<https://www.youtube.com/watch?v=RnpSaDSSjR4&feature=share>

Sur une adresse IP type (192.168.156.2 par exemple), c'est en fait le dernier chiffre qui correspond à un appareil réel.  
Solution facile pour déterminer le nombre d'appareils connectés, il suffit de récupérer la plus grande valeur des IP connectées. Mais les mobiles ayant une adresse dynamique, le comportement de ce dernier chiffre était fluctuant, et nous n'avons pas su déterminer pourquoi, ce qui nous a amené à compter les appareils d'une autre manière (comme précisé précédemment).

**Sketch disponible sur le repo Github :**  
DAY01hashmap.pde  
<https://github.com/Theojkydbz/Workshopcodecreatif>

## CONCLUSION :

On comprend donc comment différencier les requêtes externes des internes. Les appareils qui se connecteront seront très majoritairement des téléphones, on a donc trouvé une solution de rechange pour les comptabiliser même si elle est un peu sale.

# DAY TWO : AGENT INDÉPENDANT & PRÉSENTATION GRAPHIQUE

Journée consacrée à la création des agents et au traitement des données captées en sniffant le réseau

```
x[0]= noise(xoff) * width;  
y[0]= noise(yoff) * height;
```

Réadaptation d'un ancien sketch que l'on avait sur nos machines.  
Le **noise** servira à définir le mouvement des agents : l'incrémentation  
du paramètre est faite avec un **random** pour que chaque agent ait  
une amplitude et une direction de déplacement différente.

```
for(int i=1; i<1000; i++){  
    //Distance  
    float d = dist(x[0],y[0], x[i], y[i]);  
    //Probabilité de rencontre  
    float p = random(20);  
    if(d <= 20 && p<1){  
        line(x[0], y[0],x[i - 1], y[i - 1]);  
    }  
}
```

On vérifie pour un nombre conséquent de points si ils sont à la bonne  
proximité (**20**) ; et pour éviter d'avoir un pâté de traits, on pioche au  
hasard seulement quelques uns de ceux - ci.

Sketch disponible sur le repo Github :  
agent\_reliant\_autonome.pde  
<https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/WORKSHOPDAY02>

```
for(int i = y.length-1 ; i > 0 ; i-- ){  
    x[i] = x[i-1];  
    y[i] = y[i-1];  
}
```

A chaque itération, on écrase les coordonnées de passage  
enregistrées dans les tableaux **x** et **y** pour ne pas avoir un rendu  
graphique trop surchargé ou l'agent tracerait finalement des traits  
partout sur le sketch.

RENDU VISUEL :



# DAY TWO : MOUVEMENT DES AGENTS INDÉPENDANTS

Journée consacrée à la création des agents et au traitement des données captées en sniffant le réseau

```
void stayHereBro() {  
    //ensure that the particles remain on screen  
    if (location.x > width || location.x < 0) {  
        vitesse.x = -vitesse.x;  
        //println("sortie X");  
    }  
    if (location.y > height || location.y < 0) {  
        vitesse.y = -vitesse.y;  
        // println("sortie Y");  
    }  
  
    //dégagez ! c'est le blackhole ici !  
    float trou = dist(width/2, height/2, location.x, location.y);  
    if (trou < 300) {  
        vitesse.x = - vitesse.x ;  
        vitesse.y = - vitesse.y;  
    }  
  
    xoff[i] += random(0.08);  
    yoff[i] += random(0.01);  
    // float bou = map (noise(xoff[i],i*5,50), 0, 1, 0, width/2);  
    // float lou = map (noise(yoff[i],i*23,15), 0, 1, -5, 5);  
    // println(bou," ",lou);  
    // xT = location.x + bou ;  
    // yT = location.y + lou ;  
    xT = noise (xoff[i]) * width ;  
    yT = noise (yoff[i]) * height ;  
    target = new PVector(xT, yT);  
}  
  
//test en attendant : lorsqu'un nouvel objet apparaît, diriger le:  
if (ok == true) {  
    target = new PVector(width/2, height/2);  
    xT = target.x; //update new values of my points  
    yT = target.y;  
    fill(255, 2, 2);  
}
```

Intégration d'une classe (que l'on avait déjà écrite avant également) dans le sketch précédent. Elle permet de déplacer les agents non plus en changeant directement leurs positions mais plutôt en appliquant les lois de Newton (version basique : vecteurs de position, vitesse, accélération, attraction et coefficient de friction). Tests sur des points seulement.

Cette première version a considérablement modifié le mouvement des agents : plus lents, beaucoup plus directs et plutôt retors : ils s'échappent de l'écran et peuvent rentrer dans la zone interdite correspondant au trou central malgré les conditionnelles qui doivent les repousser.

On définit donc une position **target** pour définir le vecteur de déplacement et le passer dans les fonctions calculant le vecteur de vitesse, d'accélération et de friction pour pouvoir déterminer sa nouvelle **position**.

Si un évènement se produit (en version le test clic de souris), **target** devient alors le centre du centre, pour simuler l'attraction vers une bulle de requête qui aurait popped.

Problème : en faisant comme ça, les positions target sont toujours très proche de la position actuelle ce qui rend le déplacement très lent car toute la distance n'est même pas parcourue par l'agent à cause de la friction. De plus, avec des positions cibles toujours proches, cela force le déplacement à rester sensiblement dans la même direction.

Sketch disponible sur le repo Github :

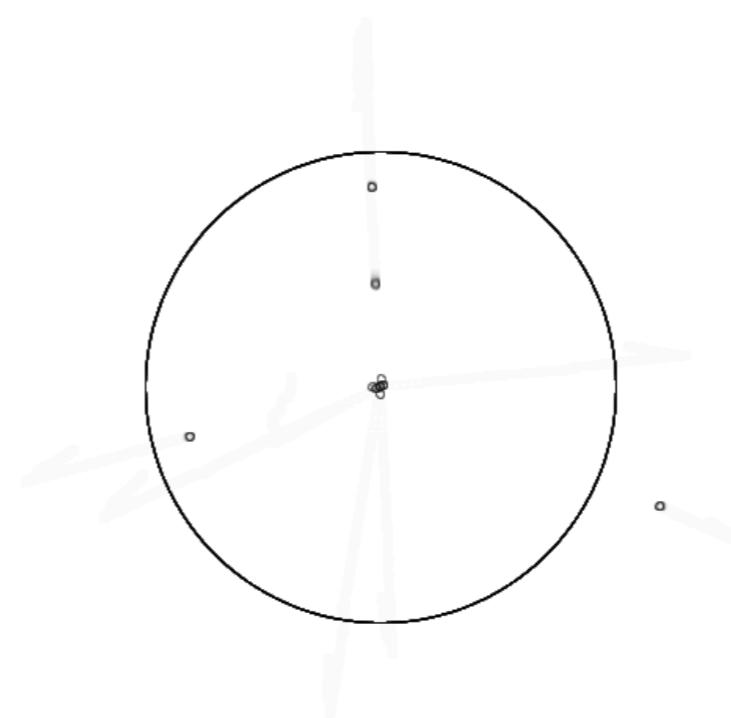
agent\_reliant\_autonome.pde

<https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/WORKSHOPDAY02>

## RENDU VISUEL :

(mode attraction)

Le cercle visualise la zone interdite correspondant au trou central de l'installation



# DAY TWO : MISE Á NIVEAU DU COMPORTEMENT DES AGENTS

Journée consacrée à la création des agents et au traitement des données captées en sniffant le réseau

```
void stayHereBro() {  
    PVector testZone = location; // vecteur temporaire pour faire les tests  
    testZone.add(vitesse);  
  
    .....  
    void degagedela(float _zoneX, float _zoneY, int _dir) {  
        //dégagez ! c'est le blackhole ici !  
        float trou = dist(width/2, height/2, _zoneX, _zoneY);  
  
        if (trou < 71 && ok == false) { //si ils sont à X dista  
            int dir2 = (int(random(0, 2)) < 1) ? 1: -1; //a  
            numDist = 5000 ; //en augmentant le numérateur du rap  
            xT = _zoneX + ( 50 * _dir ); // forcer une ejecti  
            yT = _zoneY + ( 50 * dir2 );  
            applyForces();  
  
            //tracés fantômes ; d'où sortent-ils ?  
            pushStyle();  
            fill(255, 0, 0);  
            noStroke();  
            ellipse(_zoneX, _zoneY, 5, 5);  
            popStyle();  
        }  
    }  
}
```

On crée un vecteur test pour faire les vérifications avant de actualiser la position de l'agent. Si il se trouve au centre, on lui attribuera un nouveau vecteur complètement différents de l'ancien.

En donnant une marge sur les contraintes du bord du sketch, on diminue le nombre de sorties du cadre mais certains arrivent quand même à s'échapper. Au centre, on essaie de forcer leur déplacement en leur imposant de nouveaux vecteurs de déplacements plus importants que ceux qu'ils peuvent avoir en dehors.

En regardant la console, on observe au passage que les valeurs de **zoneX** et **zoneY** correspondant aux positions potentielles des points se positionnent toutes sur la diagonale de la fenêtre. Etrange.

**Sketch disponible sur le repo Github :**  
Agent\_autonome\_waiting.pde  
[https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/noisyAgent\\_autonome\\_waiting](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/noisyAgent_autonome_waiting)

```
void killCowards(int index) {  
    if (xT < 0 || xT > width || yT < 0 || yT > height) {  
        //reset noise values to respawn it in the screen  
        xoff[index] = 0;  
        yoff[index] = 0;  
        if (xT<0) xT = random(11, (width/2) - 51);  
        if (xT>width) xT = random((width/2) + 51, width -11);  
        if (yT<0) yT = random(11, (height/2)- 51);  
        if (yT>height) yT = random((height/2)+51, height - 11);  
        applyForces();  
    }  
}
```

Si des fuyards forcent le passage, on reset leurs **noise** et attribue de nouvelles valeurs comprises dans l'écran

```
void toileAraignee(float _x, float _y, float _maxLines) {  
    //virer les anciennes valeurs  
    if (toiles.size() > 1000) toiles.remove(0);
```

Puisque qu'on range les anciennes positions dans un **ArrayList** **Toile** cette fois-ci, plus besoin de réindexer toutes les valeurs pour supprimer les trop vieilles. Un simple **remove** suffit et l'**arraylist** se réindexe dynamiquement.

# DAY TWO : MISE Á NIVEAU DU COMPORTEMENT DES AGENTS

Journée consacrée à la création des agents et au traitement des données captées en sniffant le réseau

```
if (ok == true) {  
    ...  
    distance = distance * 0.73 ; // également pour diminuer la valeur pour accélérer  
    if ( timer > t_stamp ) numDist = 30; // particules qui s'excitent avant de se jeter  
}  
  
float m = numDist /(distance * distance); //formule de gravité pour calculer la force  
//formula de gravité pour calculer la force
```

```
maxLines = map(vitesse.mag(), 0, 5, 600, 998);  
maxLines = constrain(maxLines,600,998); // util  
if (toiles.size() >= 1000) {  
    // println(_maxLines);  
    for (int i = 1; i < _maxLines; i++) {
```

Le **timer** définit le temps durant lequel les particules restent plus ou moins sur place en frétiltant avant de foncer sur l'objectif au centre.

Formule d'attraction simplifiée pour modifier le déplacement des agents :

**numDist** change en fonction de la position de l'agent : dans une zone interdite, elle augmente.

La **distance** calculée est également minorée pour simuler un état d'excitation car il n'est pas dans son milieu. Cela permet de pallier au problème de franchissement des frontières, en donnant une cohérence dans leur comportement.

Cela leur donne une nouvelle façon de se déplacer : plus vraiment fluide, le déplacement est plus saccadé et ressemble à celui des blattes. On considère que c'est plus en lien avec le propos : on amène la personnification des usagers à avoir un comportement de parasite en quelque sorte.

Lorsque les particules se déplace très peu, elles tracent alors trop de traits, ce qui donne un rendu «gros pâté». On contraint donc le nombre maximal de **lines** qu'il pourra tracer en mappant sa vitesse.

- 
- 
- 
- 
- 
- 
- 
- 

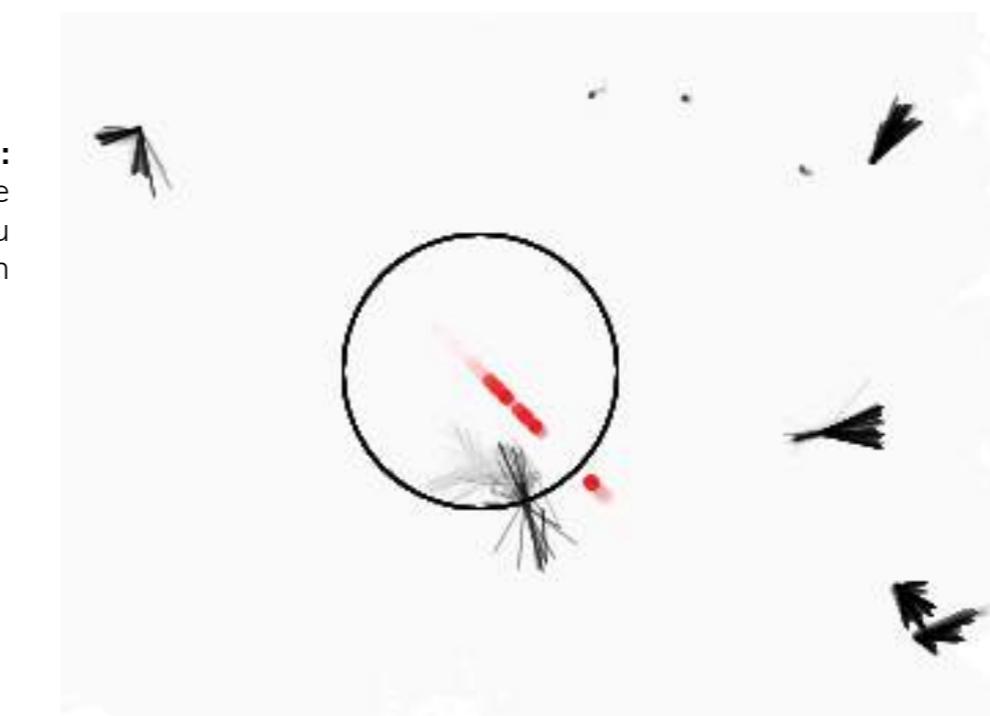
Sketch disponible sur le repo Github :

agent\_reliant\_autonome.pde

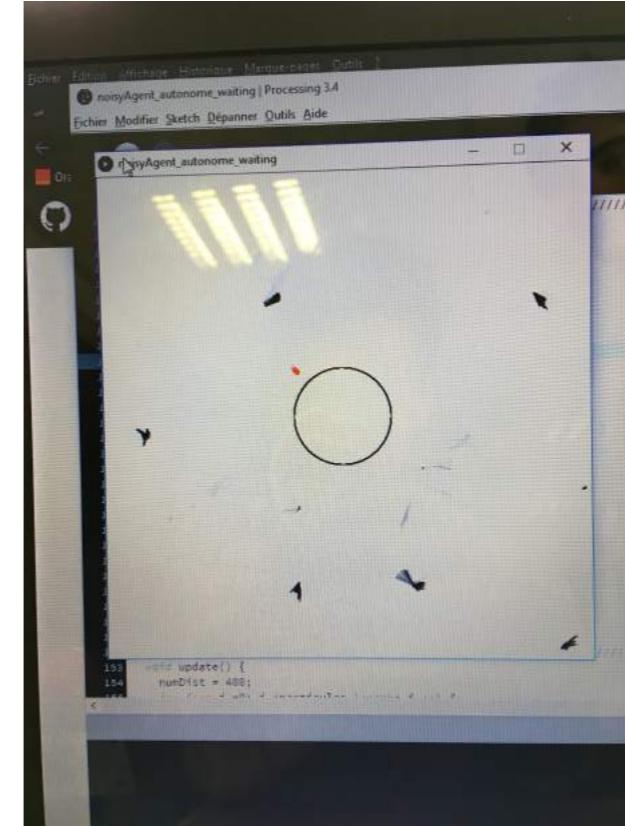
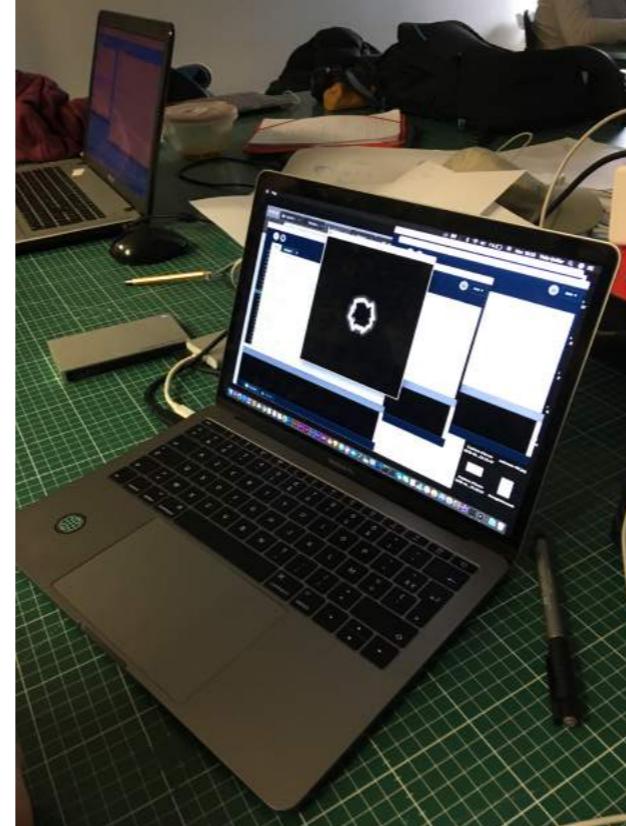
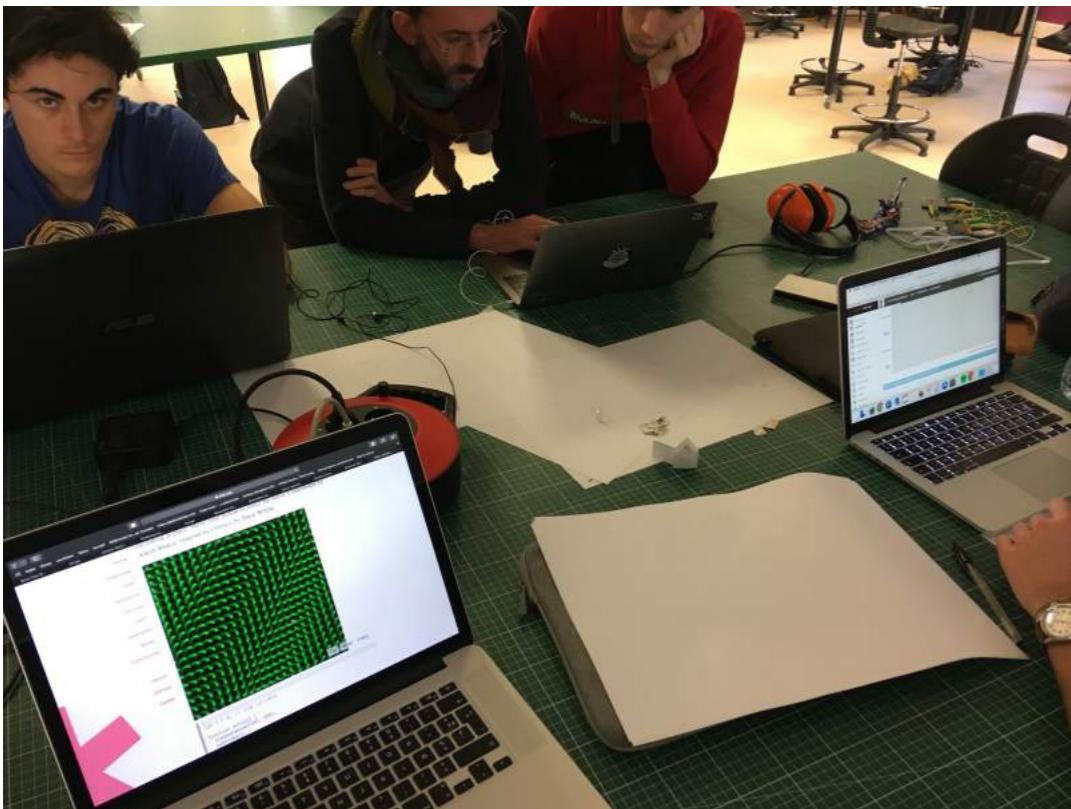
<https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/WORKSHOPDAY02>

- 
- 
- 
- 
- 
- 
- 
- 

**RENDU VISUEL :**  
Le cercle visualise la zone interdite correspondant au trou central de l'installation



# // CARNET DE BORD // DAY 2



// -Décryptage, Carnivore, solution du hotspot « partage de connexion » pour héberger notre projet

-Changement d'orientation scénographique du projet. (inspiration autour des trous noirs trous de ver) forme circulaire, layers abandonnés.

-Agents qui se déplacent seuls et se relient avec d'anciens points (problème tous les agents vont vers un seul point)

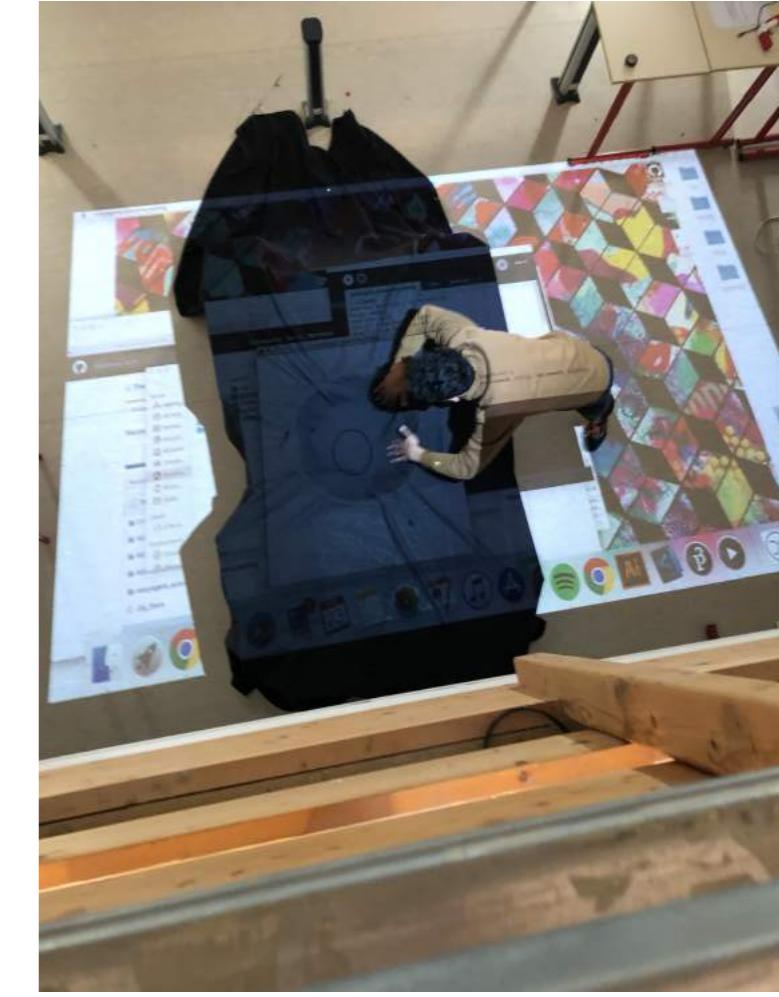
-Test projecteur et scénographie avec des tables, un cerceau, des draps noirs

-Création des sons et de l'ambiance sur Ableton ( inspirations Slender man / Ender man)

-Création d'un QR code à flasher redirigeant sur une page web avec plusieurs liens complétant le projet (Pinterest, vidéos, liens divers et variés) Création d'une ambiance autour du projet.

-Avancée du carnet de recherche. //

# // CARNET DE BORD // DAY 2



# DAY THREE : TEXTURE DU BACKGROUND

Journée consacrée à la maj des agents et création des autres éléments graphiques du sketch

```
float speed = random(0.005);
float tpos=0;
float radius = random(150, 400);
float size = random(3, 8);
float sens= random(0,1);

void update() {
  if (sens < 0.5) {
    tpos += speed;
  } else {
    tpos -= speed;
  }
  float sinval = sin(tpos);
  float cosval = cos(tpos);

currSize+=((maxSize-currSize)*ease);
void drawBlackHole(float x, float y, float size) {
  //centers pen
  translate(x, y);
  //loops through 10 times
  for (int i=0; i<10; i++) {
    //rotates semi-randomly to add variation to the
    rotation += random(0.05, 0.1);
    rotate(rotation);
    noFill();
    stroke(20, 20, 254);
    ellipse(0, 0, random(1, 300*size), 400*size);
    rotate(rotation+90);
    ellipse(0, 0, random(1, 300*size), 400*size);
```

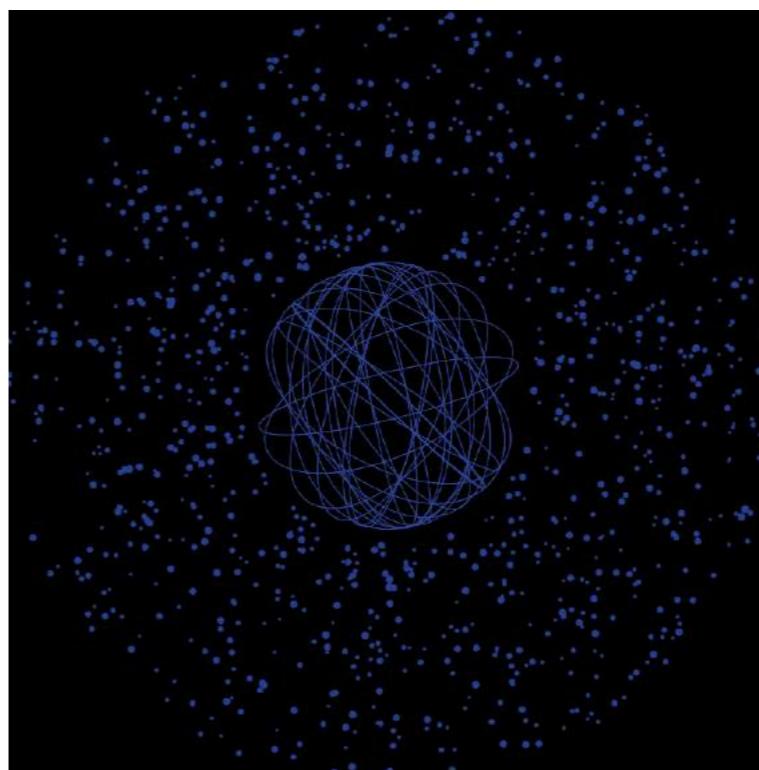
En prévision de l'intégration de tous les sketches tests, tout est fait sous forme de **classe** pour faciliter le rassemblement. Les particules rangées dans un **ArrayList Back** du background font donc le tour du trou central.

On update leurs positions directement ici, en augmentant à chaque itération l'angle **tpos** grâce à **speed**. Un **random** entre **0 et 1** pour définir leur sens de rotation grâce à une conditionnelle.

Sketch disponible sur le repo Github :  
back.pde  
[https://github.com/Theojkydbz/Workshopcodecreatif/  
tree/master/background\\_circlezone](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/background_circlezone)

En version test avec la souris, une boule grossit rapidement en l'incrémentant légèrement à chaque itération.  
Ce sont des ellipses dont les rayons sont donnés par un **random** qui ont subi une rotation.  
La taille **currSize** des ellipse croît avec un effet d'easing.

**RENDU VISUEL :**  
(mode déployé et pop)



# DAY THREE : TEXTURE DU BACKGROUND & VIE

Journée consacrée à la maj des agents et création des autres éléments graphiques du sketch

```
void life() {  
    //food  
  
    //hunger  
    if (frameCount%2==0&&lvl<700) {  
        lvl+=1;  
    }  
    rect(width*0.3, lvl, width*0.4, height);  
    //println(lvl);  
}  
  
void keyPressed() {  
    if (lvl > 0) lvl -= 10;  
}  
  
void update(int index) {  
    radiusvoidrev+=1;  
    if (sensvoidrev < 0.5) {  
        tposvoidrev += speedvoidrev;  
    } else {  
        tposvoidrev -= speedvoidrev;  
    }  
    float sinvalvoidrev = sin(tposvoidrev);  
    float cosvalvoidrev = cos(tposvoidrev);  
    xposvoidrev = (width / 2) + (cosvalvoidrev * radiusvoidrev);  
    yposvoidrev = (height / 2) + (sinvalvoidrev * radiusvoidrev);  
    if (radiusvoidrev > 150) {  
        if (sensvoidrev < 0.5) {  
            tposvoidrev += speedvoidrev;  
        } else {  
            tposvoidrev -= speedvoidrev;  
        }  
        float sinvalvoidrev = sin(tposvoidrev);  
        float cosvalvoidrev = cos(tposvoidrev);  
        xposvoidrev = (width / 2) + (cosvalvoidrev * radiusvoidrev);  
        yposvoidrev = (height / 2) + (sinvalvoidrev * radiusvoidrev);  
    }  
}
```

Ajout de la notion de vie des particules, en mode test avec les touches du clavier. Elle décroît rapidement toutes les deux **frames**, et augmente à chaque nouvel évènement (ici un **keyPressed**).

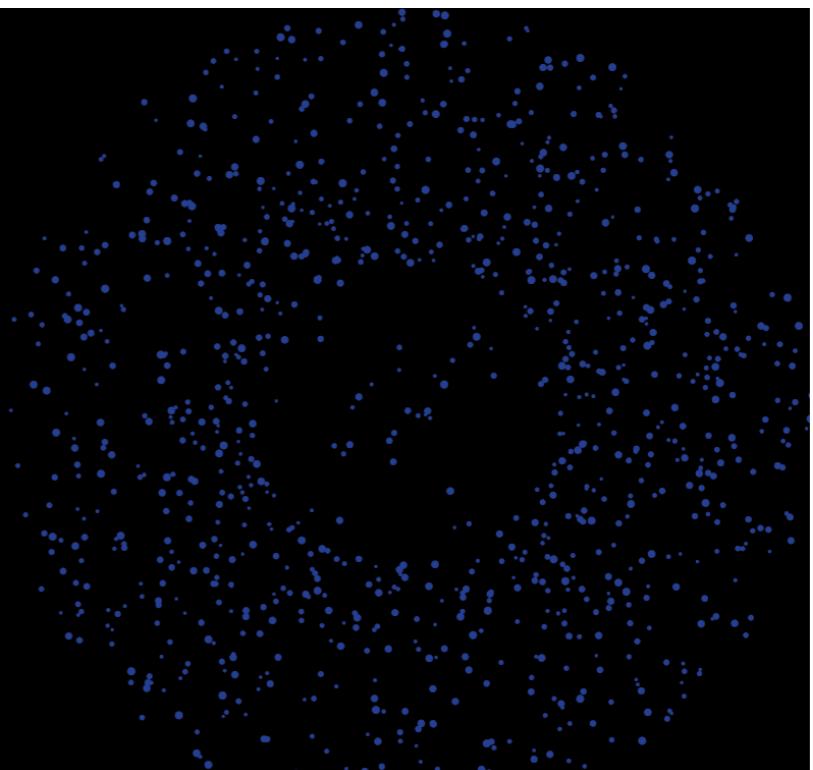
- 
- 
- 
- 
- 
- 
- 
- 

Ajout d'une nouvelle feature au **background** : aspiration vers le centre de quelques particules. Pour faciliter la chose, c'est une autre **classe** qui crée d'autres particules, plutôt que modifier le comportement de seulement quelques unes. On introduit un **radiusvoid** qui décroît fortement à chaque itération. On a fait une version **reverse** pour visualiser le meilleur rendu. A l'écran, le reverse (apparition de quelques particules depuis le centre) rend mieux.

Sketch disponible sur le repo Github :  
background\_circlezone\_backvoid\_backvoidreverse.pde  
[https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/background\\_circlezone\\_backvoid\\_backvoidreverse](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/background_circlezone_backvoid_backvoidreverse)

- 
- 
- 
- 
- 
- 
- 
- 

RENDUS VISUELS :



# DAY THREE : PRÉSENTATION GRAPHIQUE DES REQUÊTES

Journée consacrée à la maj des agents et création des autres éléments graphiques du sketch

```
for (int i=0; i<formResolution; i++) {  
    xoff+=0;  
    stepSize=noise(xoff)*2;  
    xbeg[i] += random(-stepSize,stepSize);  
    ybeg[i] += random(-stepSize,stepSize);  
    // ellipse(x[i], y[i], 5, 5);  
}  
  
// only these points are drawn  
for (int i=0; i<formResolution; i++) {  
    curveVertex(xbeg[i]+centerX, ybeg[i]+centerY);  
}  
curveVertex(xbeg[0]+centerX, ybeg[0]+centerY);  
  
// end controlpoint  
curveVertex(xbeg[1]+centerX, ybeg[1]+centerY);  
endShape();  
  
  
float angle = radians(360/float(formResolution));  
for (int i=0; i<formResolution; i++) {  
    xbeg[i] = cos(angle*i) * initRadius;  
    ybeg[i] = sin(angle*i) * initRadius;  
    xstart[i] =xbeg[i];  
    ystart[i] =ybeg[i];  
}  
void poPing() {  
    float size=100;  
    for (float i =0; i<1200; i+=0.5) {  
        size+=1;  
        stroke(255);  
        noFill();  
        ellipse(mouseX, mouseY, size, size);  
        ellipse(0,0,50,50);  
    }  
}
```

Inspiré d'un sketch du livre Generative Design : [http://www.generative-gestaltung.de/2/sketches/?01\\_P/P\\_2\\_2\\_3\\_01](http://www.generative-gestaltung.de/2/sketches/?01_P/P_2_2_3_01)

Pour créer un cercle qui se déforme au **noise**, on boucle en deux temps : le premier pour définir toutes les nouvelles positions des points, et le second pour lisser le tout avec des **curveVertex**.

Sketch disponible sur le repo Github :

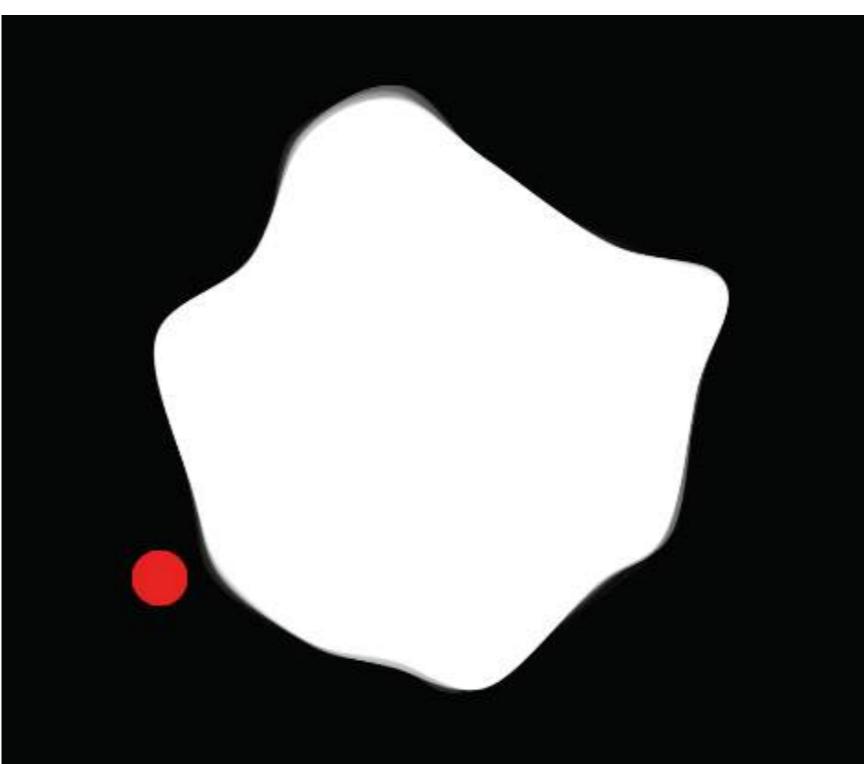
back.pde

[https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/background\\_circlezone](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/background_circlezone)

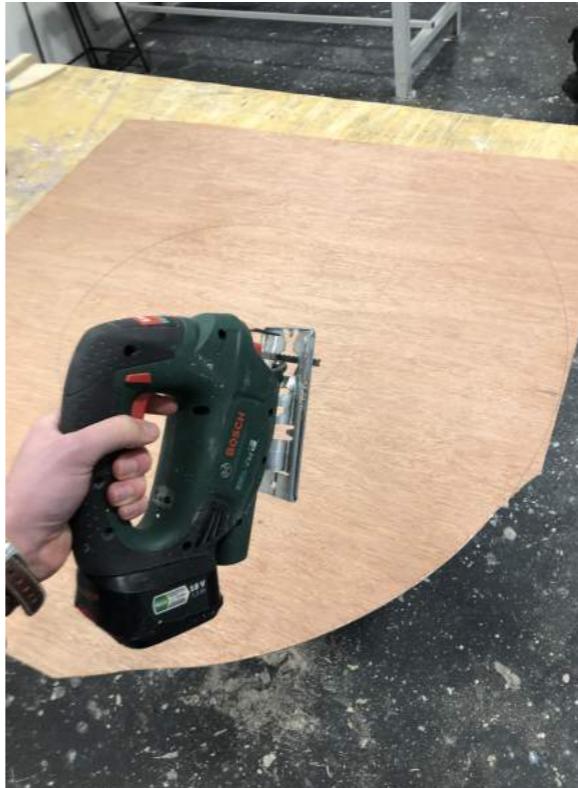
Essai de créer une onde à un évènement (ici en test **keyPressed**) mais le rendu n'est pas bon. On conservera finalement le grossissement de la boule de **circleZone**.

On intègre au passage la possibilité de revenir à un état normal pour le cercle, car le **noise** le déforme trop relativement rapidement.

RENDU VISUEL :  
(début de déformation)



# // CARNET DE BORD // DAY 3



//-Carnet de recherche + Carnet Photographique.

-informations / pages complémentaires (Pinterest, liens vers d'autres artistes, enrichissement du rendu )

-Création d'un micro organisme indépendant  
( système de vie, et nourriture) le réel enjeu ici est de créer plusieurs organismes qui ont l'air autonomes et qui possèdent tous des caractéristiques propres

-Création de la scénographie, Découpe dans une planche en contre plaqué pour créer un anneau qui est notre surface de projection.

-Essais projection, Modifications du sketch aux vues du rendu de la projection ( trop pixellisé, pas assez opaque).

Relier les codes :

- celui qui capte les requêtes et le nombre d'appareils connectés sur un hotspot
- celui qui gère les différents agents
- Le spawn des données

-Modification du QR code ( Il renvoie sur une page random dans tous les liens que l'on a sélectionné, créant des packages de data + ou - lourds). //

# DAY FOUR : INTÉGRATION GÉNÉRALE & MAPPING

Journée consacrée à la finalisation de certains éléments et de l'installation

```
.....
if (oldNumber < appareilsCo) {
particules.add(new Particle(x, y, 255));

boolean faim() {
//convertir en valeur déca pour poser les seuils
float seuilLife = map ( life, 0, 255, 0, 1);
if (seuilLife < 0.2 && openDoor == true ) _faim = true ;
if (seuilLife > 0.8) {
degagedela(location.x, location.y); //si il dépasse les
_faim = false ;
}
if (openDoor == false) {
degagedela(location.x, location.y);
_faim = false ; //pas forcément exact, mais plus simple
}
return _faim;
}
```

A chaque nouvelle connexion, un agent est rajouté avec sa notion de **vie**.

Pour faciliter la lecture, chaque action bien définie est séparée en fonction. La notion de faim est déterminée par un **boolean** en fonction de la vie restante de l'agent. 3 états :  
Il a **faim** et il y a de la nourriture (**openDoor**) = **faim** sera **true**  
Il n'a plus **faim** et est dans la **zone centrale** = **faim** **false** et l'agent est repoussé  
Il a **faim** mais il n'y a pas de nourriture = **faim** sera **false**

```
if ( particules.size() == 3){
for ( int i = 0 ; i < particules.size() ; i++){
Particle part = particules.get(i);
if ( part.life < 150) {
life = lerp(life,255,0.8);
}
}
```

Pour qu'il y ait toujours un minimum de mouvement, on conserve artificiellement **3 agents** en réaugmentant progressivement leur vie.

Sketch disponible sur le repo Github :  
GUDULOPROJMAP.pde  
<https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/GUDULOMAP/GUDULOPROJMAP>

```
void OnALaDalle(float _x, float _y) {
if (openDoor == true) {
numDist = 7000;
float cibleProche = width;
float xproche = width / 2;
float yproche = height / 2;
for (int i = 0; i < hamburger.size(); i++) {
Hamburger sandwich = hamburger.get(i) ;
float x = sandwich posX;
float y = sandwich posY;
float cible = dist(x, y, _x, _y);

if (cible < cibleProche) { //déterminer la
cibleProche = cible ;
xproche = x;
yproche = y;
}
}
```

Lorsqu'une requête fait popper de la nourriture, on parcourt l'**arraylist Hamburger** pour déterminer la cible la plus proche (une demande génère souvent plusieurs **paquets** donc plusieurs cibles).

# DAY FOUR : INTÉGRATION GÉNÉRALE & MAPPING

Journée consacrée à la finalisation de certains éléments et de l'installation

```
constrain(codecnumber, 0, 3000);  
float energie = map (codecnumber, 0, 13135, 0, 40)
```

En vérifiant la **taille** du codec de la requête, on map la valeur au passage pour déterminer l'**énergie** correspondante qui sera utilisée pour la taille des pop et pour la **quantité de vie** qu'absorbera l'agent qui mangera ce pop.  
Les valeurs changeant beaucoup trop pour qu'on ait des extrêmes fiables, cette version ne faisait que tester ; cela sera stabilisé sur la version finale du projet.

```
void nourrir(float _x, float _y) {  
    for (int i = 0; i < hamburger.size(); i++) {  
        Hamburger sandwich = hamburger.get(i);  
        float x = sandwich posX;  
        float y = sandwich posY;  
        float nrj = sandwich kcal;  
        float cible = dist(x, y, _x, _y);  
  
        if (cible < 10) {  
            hamburger.remove(i); //cible mangée  
            if (life + nrj <= 255) life += nrj; // regen  
        }  
    }  
}
```

On vérifie à chaque **update** si l'agent touche l'un des pop ; et on lui attribue sa valeur «**calorique**» le cas échéant en retirant l'**hamburger** de **l'arraylist**.

```
void playSound(AudioPlayer sound) {  
    if (sound.isPlaying() == true) {  
        sound.pause();  
        sound.rewind();  
        sound.play();  
    } else {  
        sound.rewind();  
        sound.play();  
    }  
  
    if (Pop.position() < Pop.length() && Pop.isPlaying()) {  
        Circlezone.updater();  
    } else Circlezone.reset();  
}
```

On utilise la library **Minim** pour jouer les sons qui ont été faits sur Ableton. Pour éviter la saturation lorsque des requêtes créent un grand nombre de pops, on arrête le son joué pour le relancer et créer l'effet de «**spam**» de manière sonore.

**Minim** était nécessaire pour avoir accès aux fonctions basiques d'analyse du son : le **currentTime** et la **durée** du son. On se sert de celles-ci pour écrire la condition qui régit l'apparition de la boule centrale **Circlezone**.

Sketch disponible sur le repo Github :  
GUDULOPROJMAP.pde  
<https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/GUDULOMAP/GUDULOPROJMAP>

# DAY FOUR : INTÉGRATION GÉNÉRALE & MAPPING

Journée consacrée à la finalisation de certains éléments et de l'installation

```
cp5.addSlider("sliderValue")
    .setPosition(100, 50)
    .setRange(-0.7878, 0.79)
;
cp5.addSlider("sliderValueY")
    .setPosition(100, 100)
    .setRange(1, 1.07)
;
cp5.addSlider("sliderMinX")
    .setPosition(100, 150)
    .setRange(0.5, 1.06)
;
cp5.addSlider("sliderMinY")
    .setPosition(100, 200)
    .setRange(0.5, 1.39)
;

float donuts = dist(testVect.x, testVect.y, width /2, height /2)

if (donuts > 400) {
```

Le vidéoprojection n'étant pas parallèle au sol, le sketch était déformé à la **projection**, et une partie du flux de points bleus sortait de l'installation.

En utilisant les sliders de **ControlP5**, on a pu déterminer les coefficients précis pour déformer nos ellipses et trajectoires.

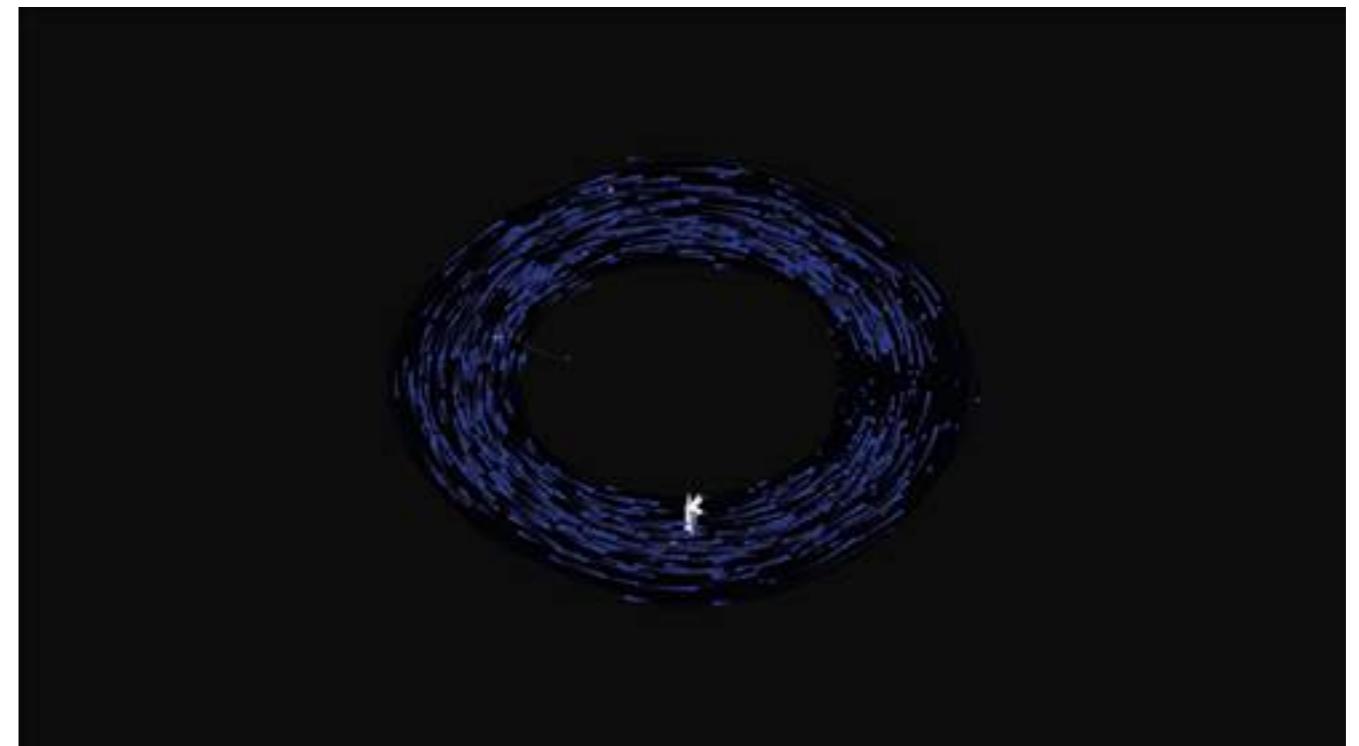
Délimiter une **zone ellipsoïdale** étant relativement compliqué, on a opté pour une solution plus simple qui n'a pas d'impact visuel : les limites sont en réalité celle d'une cercle dont le rayon correspond au rayon de l'**axe mineur** de l'ellipse en question.

**Sketch disponible sur le repo Github :**

GUDULOPROJMAP.pde

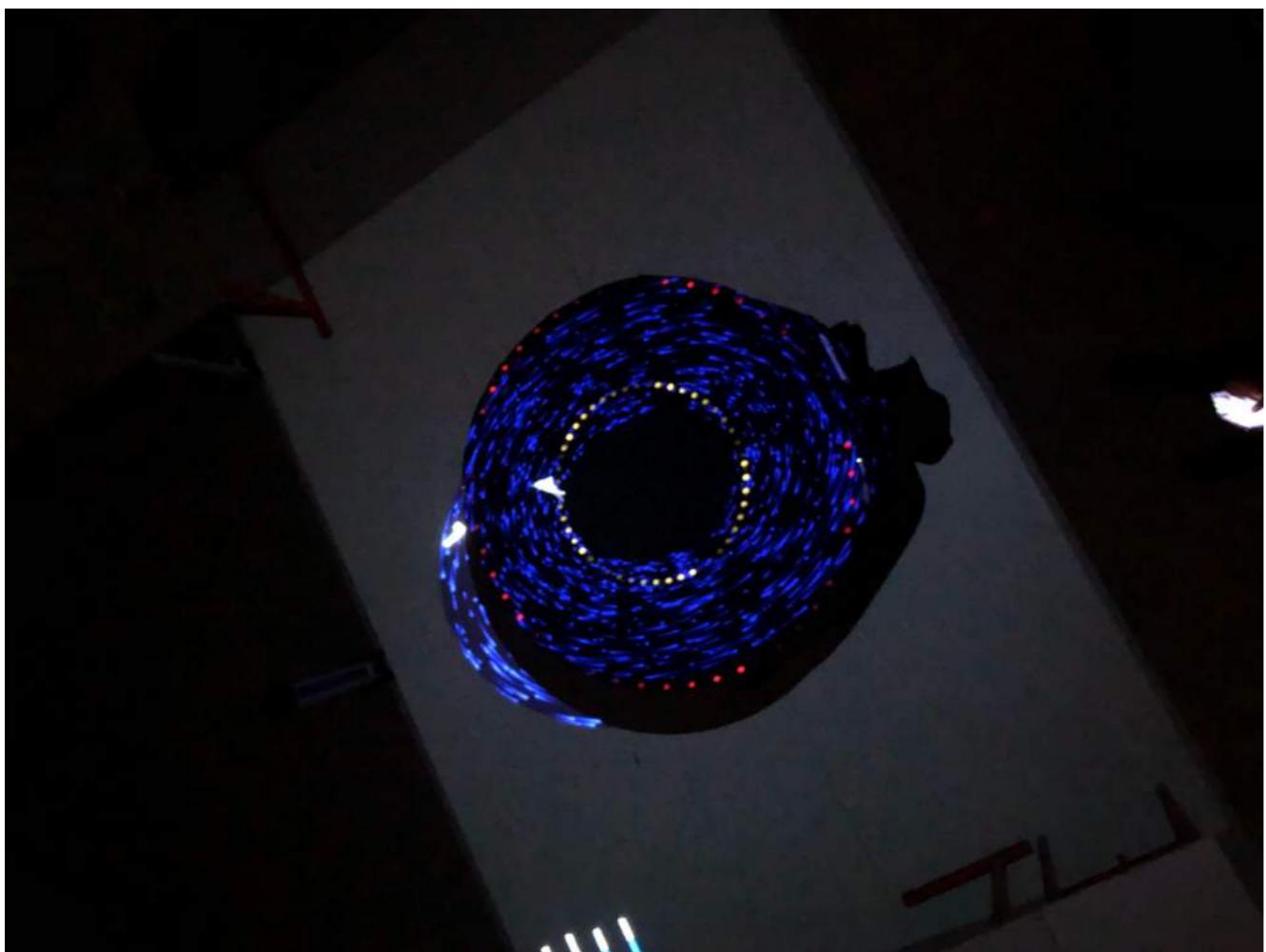
[https://github.com/Theojkydbz/Workshopcodecreatif/  
tree/master/GUDULOMAP/GUDULOPROJMAP](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/GUDULOMAP/GUDULOPROJMAP)

### **RENDU VISUEL :**



# // CARNET DE BORD // DAY 4

- // -Creation background du projet + animation apparition données // -ajout des sons avec la bibliothèque « minime »
- Assemblage du code (son, requêtes, classes, animations, food) ` -QR codes imprimés et placés
- Modification d'une des pistes son // -Réglages précis video projecteur //



- // -Branchements système son  
-mapping via le code avec la bibliothèque P5 control  
- finalisation du mapping //



- // -Création d'autres son de Pop de données  
-Création effet d'attraction au centre de la zone de projection  
-commentaire + débug du code final //

# DAY FOUR/FIVE : VERSION FINALE

Journée consacrée à la finalisation de certains éléments et de l'installation

```
//constrain(codecnumber, 0, 300);
energie = map (codecnumber, 150, 2000, 5, 30)
if (energie > 25) energie = 25 ;
```

```
AudioPlayer choosePopSound() {
    int lotterie = int(random(9));
    if ( lotterie == 9) lotterie = 8; //à ne

    if (lotterie == 0) soundPop = pophard;
    if (lotterie == 1) soundPop = pophard8;
    if (lotterie == 2) soundPop = pophard2;
    if (lotterie == 3) soundPop = pophard3;
    if (lotterie == 4) soundPop = pophard4;
    if (lotterie == 5) soundPop = pophard5;
    if (lotterie == 6) soundPop = pophard6;
    if (lotterie == 7) soundPop = pophard7;
    if (lotterie == 8) soundPop = pophard9;

    return soundPop ;
}

playSound(pophard);
```

Correction de certains bugs ou maj pour améliorer la projection et ajout d'un mode débug pour tester rapidement la viabilité de l'installation avant la présentation. Modification des sons.

Après avoir faits plusieurs observations, on note que les taille du **codec** dépassent rarement 2000 caractères, et si la limite est franchie on constrain la valeur du **map** en forçant avec une condition car la méthode **constrain** n'était visiblement pas efficace, en entrée comme

Tentative de dernière minute (qui n'a pas fonctionné) d'instaurer une **palette de son** de pop pour qu'il puisse avoir une variation sonore. Nous n'avons pas réussi à ranger tous les sons dans un **tableau** ou un **arraylist**, on a donc tenté de passer par une **string** pour donner le nom du fichier à la fonction **choosePopSound**. On a finalement opté pour une série de if qui était elle, fonctionnelle.

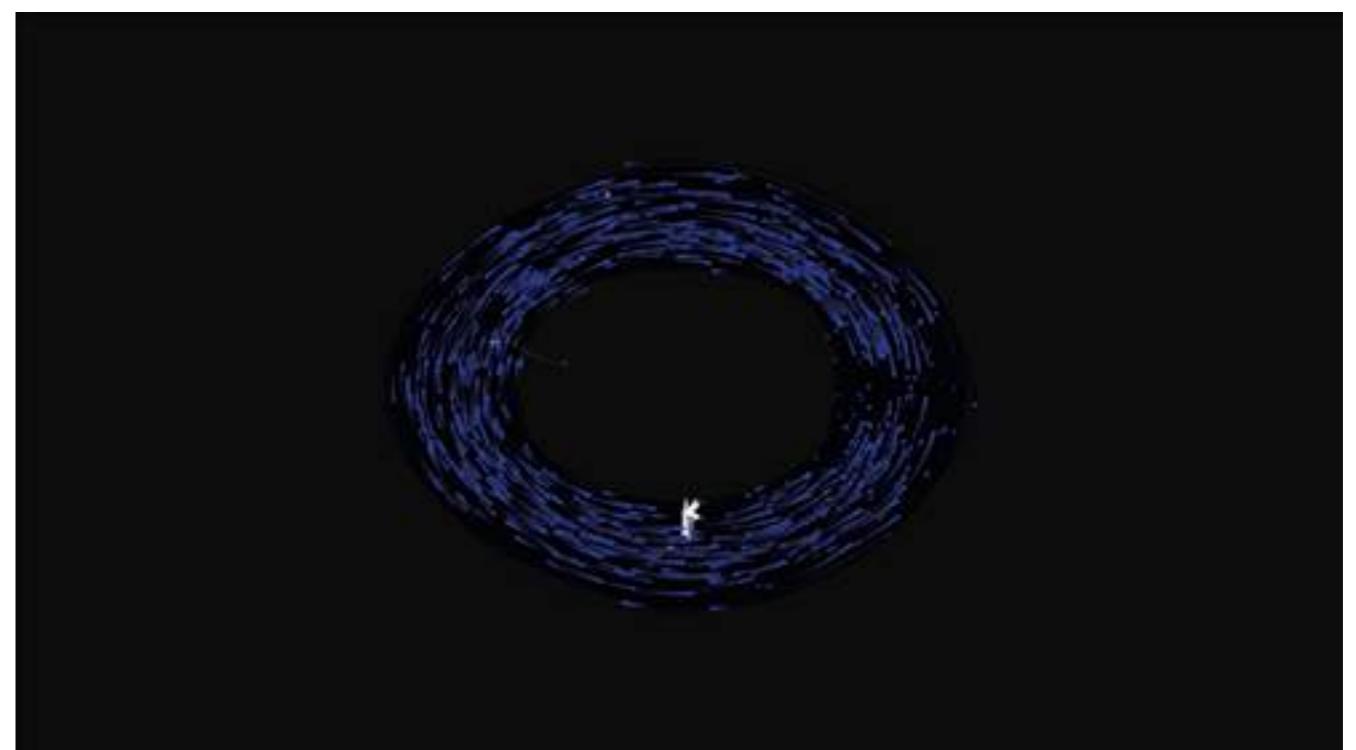
Mais l'argument de **playSound** avait besoin d'être défini sur un son précis dès le début du sketch ; on ne pouvait pas laisser un paramètre encore non défini.

Sketch disponible sur le repo Github :

GUDULOPROJMAP.pde

[https://github.com/Theojkydbz/Workshopcodecreatif/  
tree/master/VOID/GUDULOPROJ](https://github.com/Theojkydbz/Workshopcodecreatif/tree/master/VOID/GUDULOPROJ)

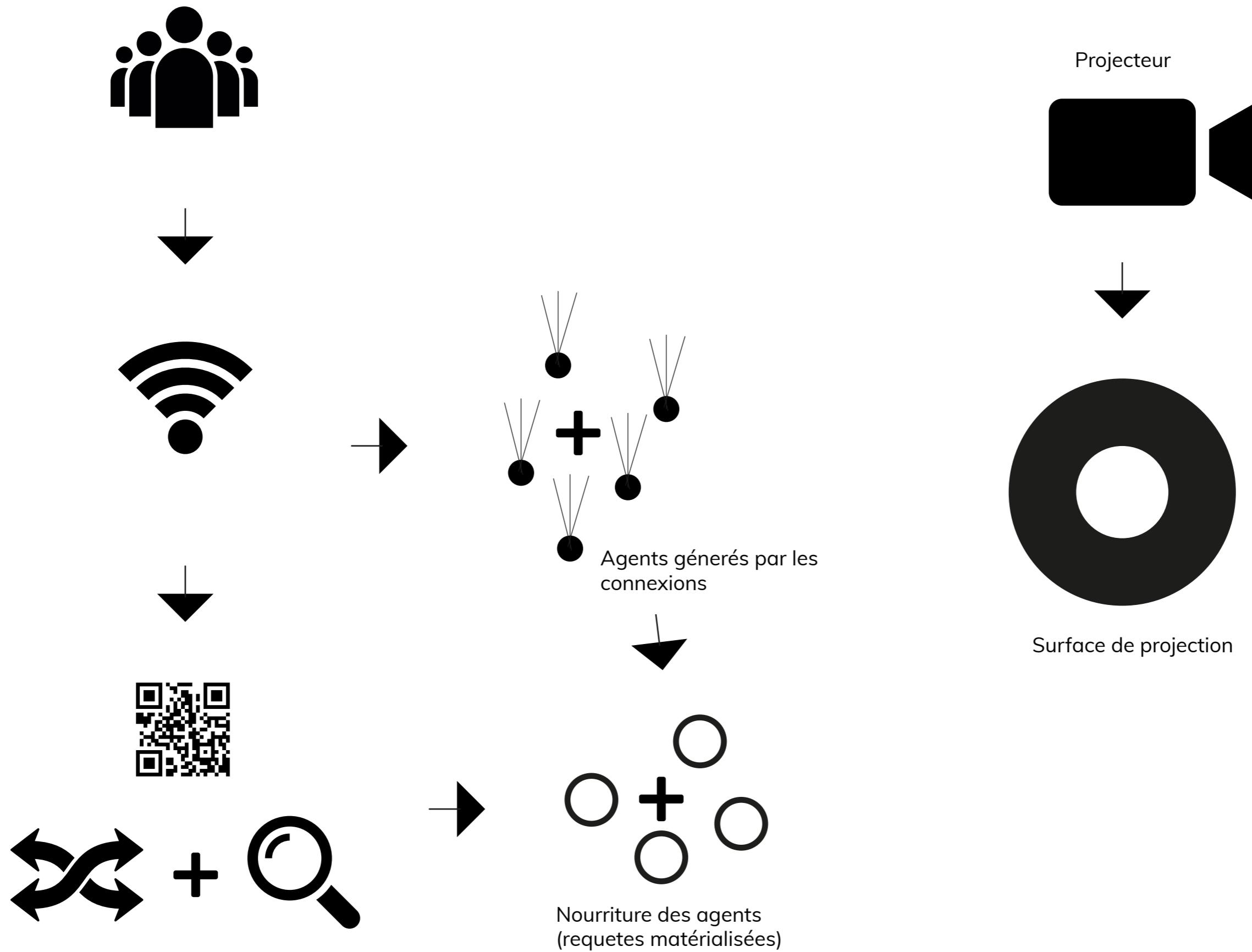
**RENDU VISUEL :**  
(visu écran après déploiement)



// LOGO //



# //SCHÉMA DE PRINCIPE //



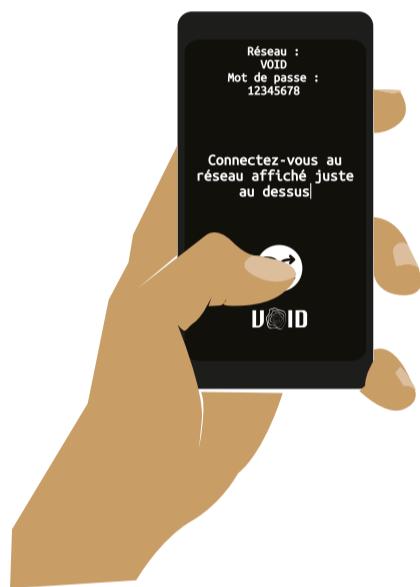
# // SCÉNARIO D'USAGE //



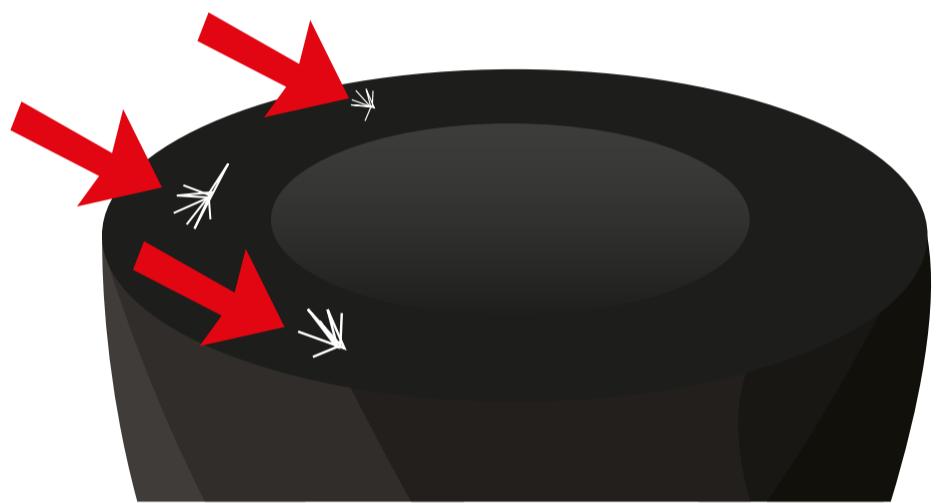
Se positionner autour de l'installation



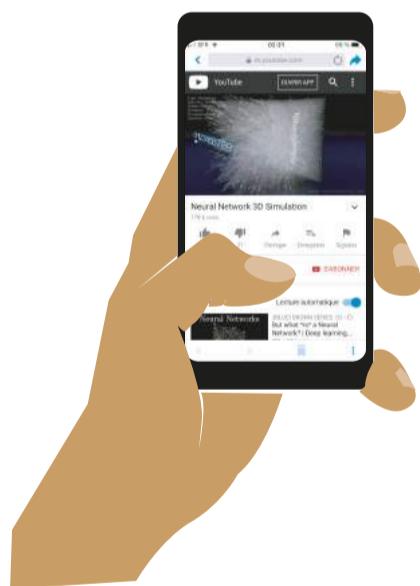
Utiliser le Qr-code pour aller sur le site



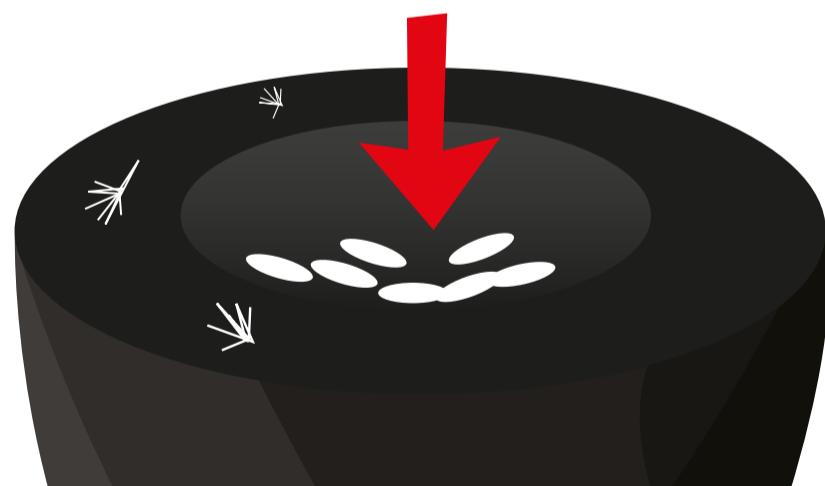
Se connecter au reseau «VOID»



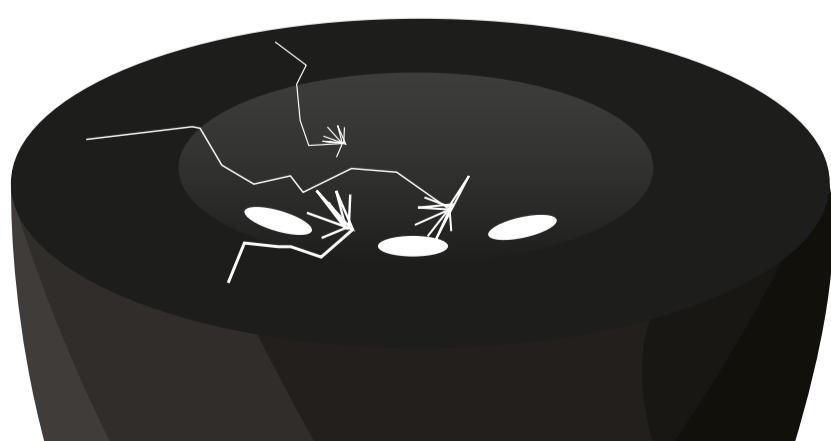
Pour chaque connection un agent apparaît sur l'installation



Faire des recherches sur internet ou regarder des vidéos pour générer des données



Les données générées vont se transformer en nourriture pour les agents



La vie des agents diminuant au cours du temps, ils doivent se nourrir pour survivre