C'est une idée fascinante, un peu comme le concept de la "Bibliothèque de Babel", mais pour les images.

Générer une image de 1000x1000 pixels demande de calculer 1 million de pixels à chaque mouvement de souris. C'est une tâche lourde pour le processeur.

Voici deux approches en Processing. Choisissez celle qui correspond le mieux à votre vision de "l'infini".

Option 1 : L'Infini "Organique" (Bruit de Perlin)
C'est l'option la plus esthétique. Elle utilise le Bruit de Perlin (un générateur de nombres aléatoires cohérents). Chaque mouvement de souris vous fait "voyager" dans une texture infinie de nuages ou de paysages abstraits. Ce n'est pas du pur hasard, c'est un hasard structuré.

Java

```
// Dimensions de l'image
void setup() {
  size(1000, 1000);
  noLoop(); // On arrête la boucle draw() continue pour économiser le CPU
}

// Cette fonction s'active uniquement quand la souris bouge
void mouseMoved() {
  redraw(); // On demande à Processing de redessiner une seule fois
}

void draw() {
  loadPixels(); // On charge le tableau des pixels pour y accéder directement (plus rapide)

  // L'échelle du bruit (plus c'est petit, plus c'est "zoomé")
  float noiseScale = 0.01;

  // On utilise la position de la souris pour décaler le bruit (le voyage dans l'infini)
  float xOffset = mouseX * 0.05;
  float yOffset = mouseY * 0.05;

  for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {

      // On calcule la couleur basée sur le bruit de Perlin
      // On ajoute xOffset et yOffset pour que l'image change avec la souris
      float r = noise((x * noiseScale) + xOffset, (y * noiseScale) + yOffset) * 255;
      float g = noise((x * noiseScale) + xOffset + 100, (y * noiseScale) + yOffset + 100) * 255; //
+100 pour décaler les couleurs
      float b = noise((x * noiseScale) + xOffset + 200, (y * noiseScale) + yOffset + 200) * 255;

      // On assigne la couleur au pixel courant
      // La formule (x + y * width) trouve l'index correct dans le tableau à 1 dimension
      pixels[x + y * width] = color(r, g, b);
    }
  }
```

```
  updatePixels(); // On met à jour l'écran
}
```
Option 2 : L'Infini "Mathématique" (Chaos Total)
Ceci est l'interprétation littérale de votre demande. Chaque pixel est totalement aléatoire. Théoriquement, si vous bougez la souris assez longtemps (quelques milliards d'années), vous verrez apparaître une photo de vous-même, la Joconde, ou n'importe quelle image possible. Attention : Visuellement, cela ressemble à de la "neige" de télévision (bruit blanc).

Java

```
void setup() {
  size(1000, 1000);
  noLoop();
}

void mouseMoved() {
  redraw();
}

void draw() {
  loadPixels();

  // On parcourt tous les pixels de l'image (1 million)
  for (int i = 0; i < pixels.length; i++) {
    // Chaque pixel reçoit une couleur R, V, B totalement aléatoire
    pixels[i] = color(random(255), random(255), random(255));
  }

  updatePixels();
}
```
Comment utiliser ce code :
Téléchargez et installez Processing (processing.org).

Copiez l'un des blocs de code ci-dessus.

Collez-le dans l'éditeur.

Appuyez sur le bouton Play (le triangle).

Note technique : Comme 1000x1000 représente beaucoup de données, j'ai utilisé la fonction noLoop() et mouseMoved(). Cela signifie que l'ordinateur ne calcule l'image que lorsque vous bougez la souris, évitant ainsi de faire surchauffer votre machine inutilement.

Souhaitez-vous que j'ajoute une fonctionnalité pour sauvegarder l'image si vous tombez sur une variation qui vous plaît ?

oui pourquoi pas ! mais y a - t-il assez de temps entre 2 mouvements de souris ?

C'est une excellente question technique. La réponse courte est : pour l'affichage, oui, c'est juste ; pour la sauvegarde, non.

Voici l'explication détaillée de ce qui se passe dans la "boîte noire" de l'ordinateur :

Le Calcul (CPU) : Générer 1 million de pixels aléatoires (méthode "Chaos") prend quelques millisecondes. Un ordinateur moderne peut le faire environ 30 à 60 fois par seconde. C'est fluide pour l'œil.

L'Affichage (GPU) : Envoyer ces pixels à l'écran est très rapide.

La Sauvegarde (Disque Dur) : C'est là que ça coince. Écrire une image de 1000x1000 pixels sur le disque dur est une opération "lente" (en temps informatique). Si on essayait de sauvegarder à chaque micromouvement de souris, l'ordinateur gèlerait complètement car il passerait son temps à écrire des fichiers plutôt qu'à calculer la suite.

La Solution : Dissocier l'exploration de la capture.

On va laisser l'ordinateur calculer l'image à toute vitesse quand vous bougez la souris (mode exploration), mais on ne déclenchera la sauvegarde (opération lente) que sur une action volontaire (clic de souris ou touche du clavier).

Voici le code complet, optimisé et sécurisé :

Java

```
// Programme : Le Chercheur d'Infini
// Instructions :
// 1. Bougez la souris pour explorer le bruit blanc
// 2. Appuyez sur la touche 's' ou CLIQUEZ pour sauvegarder l'image actuelle

void setup() {
  size(1000, 1000);
  noLoop(); // Important : on ne calcule que si nécessaire
}

void draw() {
  loadPixels();

  // BOUCLE OPTIMISÉE
  // On remplit le tableau de pixels directement
  for (int i = 0; i < pixels.length; i++) {
    // Note : color(r,g,b) est un peu lent en Java pur dans une boucle intensive.
    // Ici on utilise une astuce binaire pour aller plus vite :
    // On génère un entier qui contient l'alpha (FF) et le RGB aléatoire d'un coup.
    // 0xFF000000 est le masque pour l'opacité totale.
    pixels[i] = 0xFF000000 | (int)random(0xFFFFFF);
  }

  updatePixels();
}

// Cette fonction s'exécute à chaque fois que la souris bouge
void mouseMoved() {
  redraw(); // Demande un nouveau dessin immédiatement
}
```

```
// Cette fonction s'exécute quand on clique
void mousePressed() {
  sauvegarderImage();
}

// Cette fonction s'exécute quand on appuie sur une touche
void keyPressed() {
  if (key == 's' || key == 'S') {
    sauvegarderImage();
  }
}

void sauvegarderImage() {
  // On crée un nom de fichier unique basé sur la date et l'heure
  // ex: image-20231027-153022.png
  String nomFichier = "infini-" + year() + month() + day() + "-" + hour() + minute() + second() +
millis() + ".png";

  saveFrame(nomFichier);
  println("Image sauvegardée : " + nomFichier); // Petit message dans la console
}
```

Ce qui a changé pour la performance :

L'astuce binaire (0xFF000000 | ...) : Au lieu de demander à Processing de calculer des objets "Couleur" complexes trois fois (R, G, B) un million de fois par image, je génère directement la valeur brute du pixel en hexadécimal. C'est beaucoup plus rapide pour le processeur.

Nommage intelligent : J'ai ajouté l'heure et les millisecondes au nom du fichier. Ainsi, vous pouvez sauvegarder plusieurs images à la suite sans qu'elles ne s'écrasent les unes les autres.

Essayez ce code. Vous verrez que l'image change de manière fluide quand vous bougez la souris, et il suffit d'un clic pour immortaliser un moment de ce "bruit" unique.