Aufgabe2: Verzinkt

Team-ID: 00382

Team-Name: Sehr gutes BWINF Team

Bearbeiter/-innen dieser Aufgabe:
Paul Franosch, Joshua Benning, Lenny Schürer, Marinus Lentile

20. November 2022

Inhaltsverzeichnis

1 Lösungsidee

Ziel ist es, dass Wachstum von Kristallen nach den gegebenen Bedingungen zu simulieren. Dazu wird ein initial leeres zweidimensionales Array (Canvas) verwendet, in dem sich Keime ausbreiten. Die Wachstumsgeschwindigkeit in eine bestimmte Richtung wird dabei durch eine Wahrscheinlichkeit modelliert (Ob bei einem "Wachstumsschritt"der Keim in diese Richtung wächst). Das Wachstum erfolgt dabei rekursiv, Pixel die also durch einen wachsenden Keim eingenommen wurden, werden zu Keimen. Daten wie die Farbe und die Wachstumsgeschwindigkeiten werden dabei weitergegeben. Keime die sich nicht weiter ausbreiten können, verbleiben als gefärbte Pixel werden aber nicht weiter betrachtet.

2 Umsetzung

2.1 Leinwand und Pixel

Die Implementation erfolgt in Java. Das Canvas wird dabei durch ein zweidimensionales Pixel Array dargestellt. Ein Pixel kann dabei EMPTY, ACTIVE oder INACTIVE sein.

EMPTY entspricht dem Ursprungszustand, der Pixel ist nicht gefärbt und gehört auch keinem Kristall an. ACTIVE beschreibt einen Keim / einen Pixel der noch weiter wachsen kann, INACTIVE einen gefärbten Pixel, der nicht weiter wachsen kann.
Außerdem besitzt jeder Pixel 4 Wachstumsgeschwindigkeiten (Richtungen auf dem Raster; Norden, Westen, Süden, Osten). Diese Wachstumsgeschwindigkeiten werden durch eine Fließkommazahl zwischen 0 und 1 dargestellt und beschreiben die Wahrscheinlichkeit, dass ein Kristall / Pixel bei einer Wachstumsiteration in diese Richtung wächst.
Breitet sich ein Pixel aus, gibt er all seine Daten (Farbe und Wachstumsgeschwindigkeit) weiter.

2.2 Wachstumsprozess

Entsteht ein neuer Kristall, erhält er eine zufällige Ausrichtung von 0 bis 1. Mit der Ausrichtung als Weißanteil wird dem Kristall dann eine Farbe der Schwarz-Weiß-Skala zugeordnet. Die Wachstumsgeschwindigkeiten werden zufällig, anhand von festgelegten Bereichen, ausgewählt. Dieser neue Pixel wird dann in eine Queue hinzugefügt und nach dem Prinzip "First in first outäbgearbeitet. Landet ein Pixel P in dieser Queue an erster Stelle, wird seine Von-Neumann-Nachbarschaft (Direkte Nachbarschaft; links, rechts, oben, unten; nicht diagonal) überprüft. Wenn ein Nachbar N den Status EMPTY hat wird N (mit der entsprechenden Wachstumsgeschwindigkeit) ACTIVE und wird an die Queue angefügt. N übernimmt dabei die Farbe und Wachstumsgeschwindigkeiten von P.
Sollte die gesamte Van-Neumann-Nachbarschaft keine EMPTY Pixel enthalten, wird P INACTIVE.
Dieser Prozess wird solange durchgeführt, bis das gesamte Canvas durch Kristalle gefüllt wurde.

2.3 Anpassung der Ergebnisse

Die Bildgenerierung kann dabei durch Parameter angepasst werden

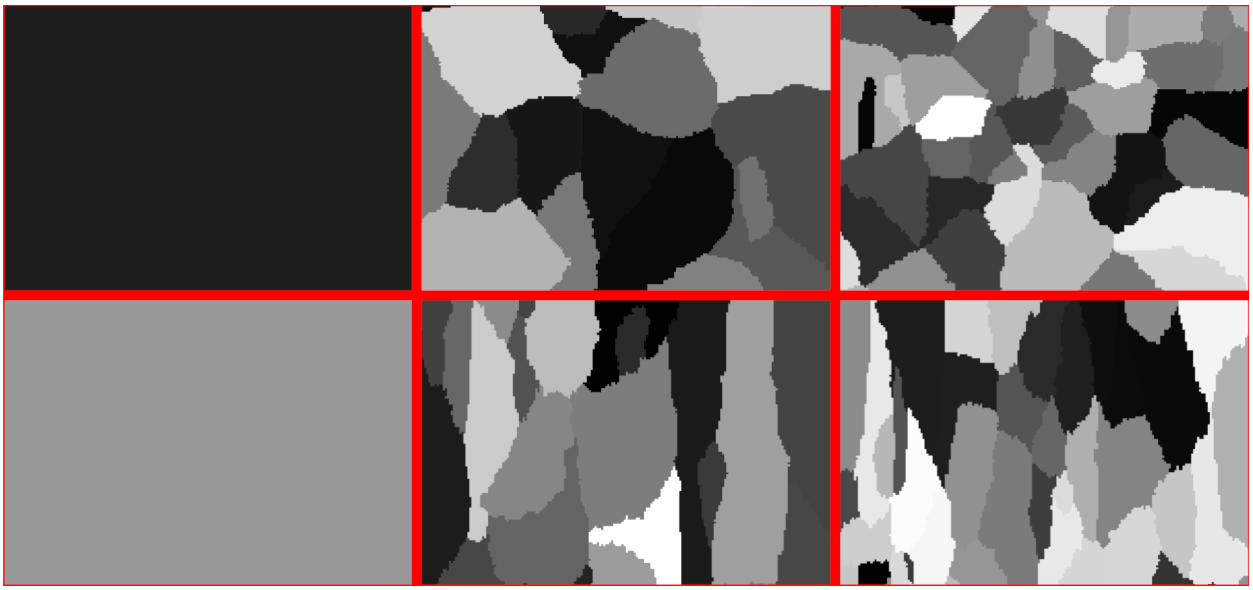
Argument	Beschreibung
-width	Ganzzahl, Breite des Canvas
-height	Ganzzahl, Höhe des Canvas
-type	'PNG' oder 'GIF', Ausgabeformat
-initialSP	Ganzzahl, Beschreibt die initiale Anzahl an Keimen
-SPperTick	Ganzzahl, Keime die pro Tick hinzugefügt werden
-tickDuration	Ganzzahl, Gibt an wie viele Wachstumsiterationen durchgeführt werden bevor -SPperTick Keime hinzugefügt werden
-minSpeedX und -maxSpeedX	Fließkommazahl, X entspricht dabei der Himmelsrichtung auf dem Canvas (N,S,W,E). Beispiel: -minSpeedN 0.1 -maxSpeedN 0.7 → Wachstumsgeschwindigkeit nach oben variiert zwischen 0.1 und 0.7
-minSpeedXY und -maxSpeedXY	Fließkommazahl, X und Y entpsprechen dabei einer Himmelsrichtung auf dem Canvas (N,S,W,E). So kann die Geschwindigkeit einer Ächseëinfacher angegeben werden Beispiel: -minSpeedNS 0.1 -maxSpeedNS 0.7

2.4 Ausgabe

Die Ausgabe der Muster erfolgt entweder als PNG oder GIF. Das Programm legt dazu einen Ordner öutän und speichert dort entsprechende Dateien ab. Im Folgenden wird auf die Einstellung der Parameter anhand der Bilder eingegangen, im Anhang finden sich aber auch GIFs mit ähnlichen Parametern wie die Beispiele.

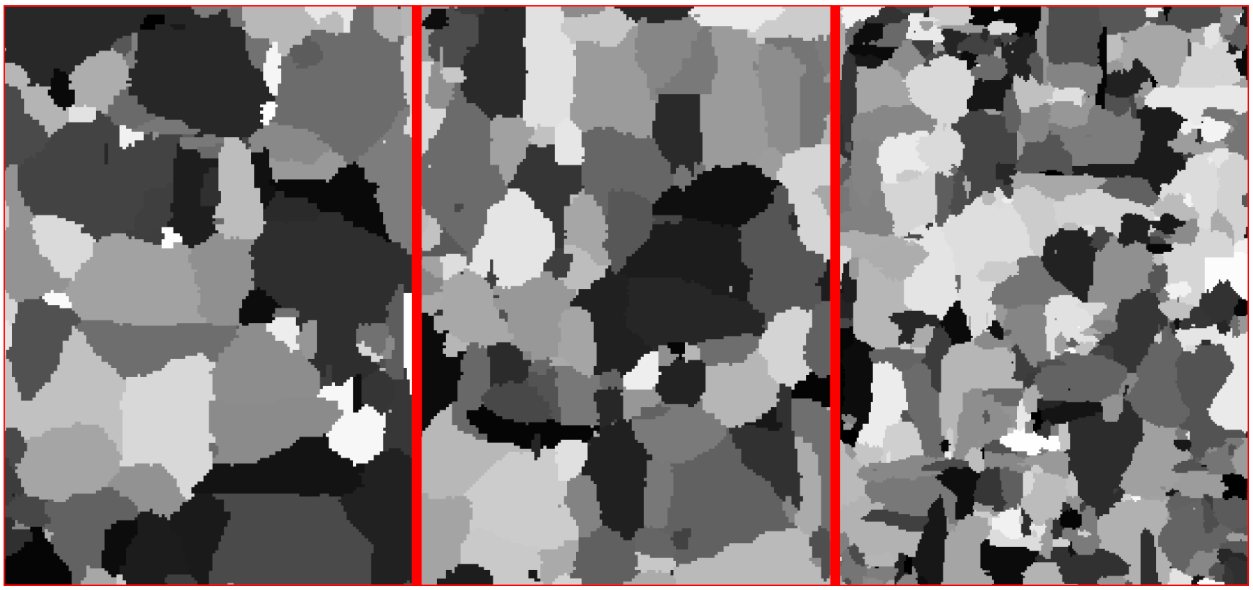
3 Beispiele

3.1 Initiale Keime und Wachstumsgeschwindigkeiten



Bei einem Canvas von 200px x 200px sind hier Muster mit links einem, in der Mitte 25 und rechts 50 initialen Keimen dargestellt. Hier wird allerdings deutlich, dass da die einzelnen Kristalle ähnlich groß sind das Referenzbild nur schwer anzunähern ist. Bei der oberen Reihe variieren dabei die Wachstumsgeschwindigkeiten in alle Richtungen von 0.0 bis 1.0. Bei der unteren wurden für die Ost-West-Achse ein Maximum von 0.1 festgelegt. Die Kristalle werden daher bis zu 10x höher als sie breit sind.

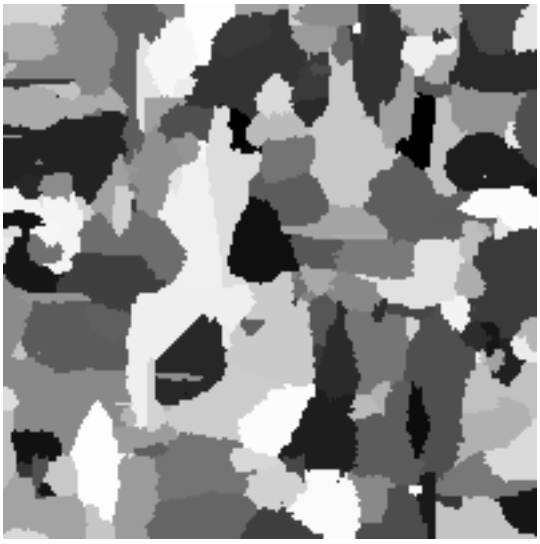
3.2 Späteres Hinzufügen von Keimen



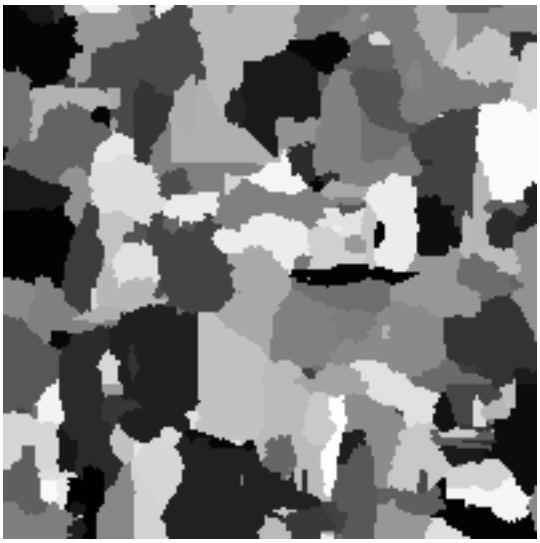
Da auf dem Referenzbild allerdings Kristalle verschiedener Formen und allgemein verschiedener Größen vorliegen, reichen die Wachstumsgeschwindigkeit und die Keimzahl nicht, um diese Varianz vollständig darzustellen. Es bietet sich daher an, Keime erst auf das Raster zu setzen nachdem andere bereits Zeit zum Wachsen hatten. In den Beispielen hier wurde wieder ein Raster von 200px x 200px verwendet. In dem Bild links wurde 1 Keim, im Mittleren wurden 15 Keime und rechts wurden 30 Keime pro 3 Wachstumsiterationen hinzugefügt. Die Bilder hier ähneln dem Referenzbild schon eher, da sie sowohl Bereiche mit Ansammlungen kleiner als auch große, durchgängige Kristalle aufweisen.

3.3 Beispiele

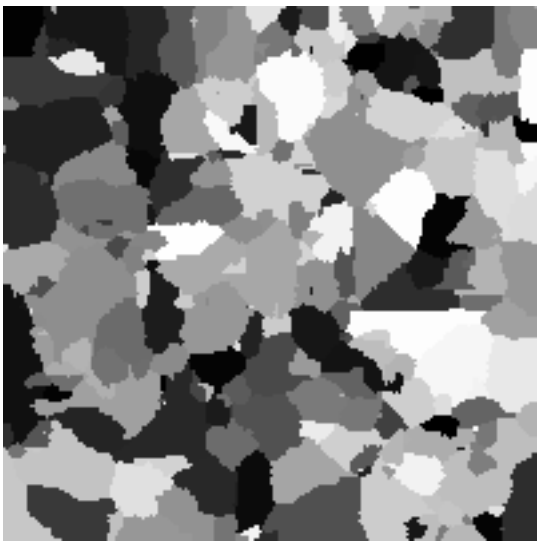
3.3.1 Beispiel 1



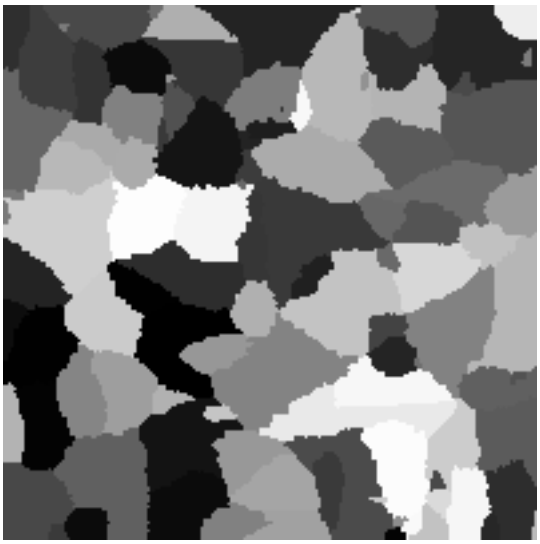
3.3.2 Beispiel 2



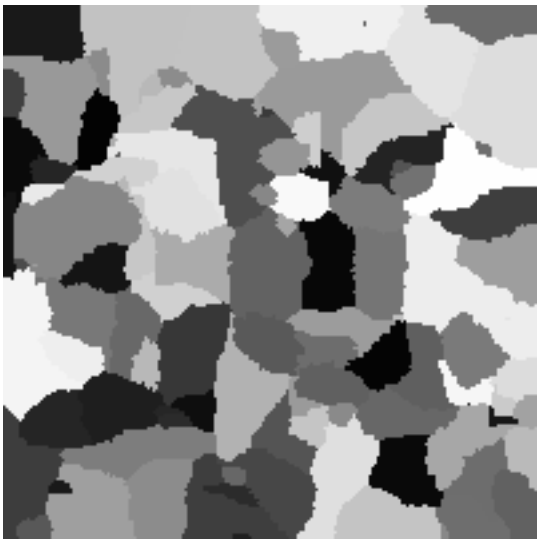
3.3.3 Beispiel 3



3.3.4 Beispiel 4



3.3.5 Beispiel 5



4 Quellcode

4.1 ImageGenerator-Klasse

```java
public class ImageGenerator {
    private final ImageGenData imageGenData;
    private final GrowSpeedGenerator growSpeedGenerator;
    private final GenerationData generationData; // Done

    private List<BufferedImage> images = new ArrayList<>();

    public void generateImage() throws IOException {
        Long start = System.currentTimeMillis();
        try {
            String d = getProgramPath() + "/out/";
            File dir = new File(d);
            if (dir.exists()) System.out.println("Output folder was found");
            if (!dir.mkdir()) System.out.println("Could not generate output folder!");
        }
        catch (Exception e) { System.out.println("Exception occured" + e);}


        Canvas canvas = new Canvas(imageGenData.getWidth(), imageGenData.getHeight(), Color.BLACK);
        Random random = new Random();
        Queue<Pixel> pixelsToExpand = new ConcurrentLinkedQueue<>(); // Pixels that can grow
        generateStartPoints(canvas, random, pixelsToExpand,  generationData.getInitialSP());
        int iterationCount = 0; // Counting iterations
        while (pixelsToExpand.size() != 0 && iterationCount < 1_000_000) {
            if (iterationCount % generationData.getTickDuration() == 0)
              generateStartPoints(canvas, random, pixelsToExpand, generationData.getSPperTick());

            Pixel poll = pixelsToExpand.poll();
            if (poll != null) poll.setPixelState(Pixel.PixelState.INACTIVE);
            pixelsToExpand.addAll(canvas.expand(poll));
            iterationCount++;

            // Saving a frame every 1000 iterations if a GIF should be generated
            if (this.imageGenData.getImageType() == ImageType.GIF && iterationCount % 1000 == 0) {
                images.add(canvas.render(false, ""));
            }
        }
    }


    private void generateStartPoints(Canvas canvas, Random random,
                                     Queue<Pixel> pixelsToExpand, int count) {
        for (int spawnPoint = 0; spawnPoint < count; spawnPoint++) {
            int w = random.nextInt(this.imageGenData.getWidth());
            int h = random.nextInt(this.imageGenData.getHeight());
            Pixel startPxl = canvas.getData()[w][h];
            if (startPxl.getPixelState() != Pixel.PixelState.EMPTY) {
                spawnPoint--;
                continue;
            }
            int rgbValue = (int) (random.nextFloat(0.0f, 1.0f) * 255);
            startPxl.setColor(new Color(rgbValue, rgbValue, rgbValue));
            startPxl.setGrowSpeed(this.growSpeedGenerator.getRandomSpeed());
            startPxl.setPixelState(Pixel.PixelState.INACTIVE);
            pixelsToExpand.add(startPxl);
        }
    }
}
```

4.2 Canvas-Klasse

```
public class Canvas {
    private final Pixel[][] data;
    private final int width;
    private final int height;

    public Canvas(int width, int height, Color defaultColor) {
        this.data = new Pixel[width][height];
        this.width = width;
        this.height = height;
        for (int x = 0; x < this.width; x++) {
            for (int y = 0; y < this.height; y++) {
                this.data[x][y] = new Pixel(x, y, Pixel.PixelState.EMPTY, defaultColor);
            }
        }
    }

    public Pixel[][] getData() {
        return data;
    }

    public HashMap<Pixel.Direction, Pixel> getNeumannNeighbours(Pixel pixel) {

        HashMap<Pixel.Direction, Pixel> neumannNeighbours = new HashMap<>();

        int x = pixel.getX();
        int y = pixel.getY();

        if (x + 1 < this.width) neumannNeighbours.put(Pixel.Direction.EAST, this.data[x + 1][y]);
        else neumannNeighbours.put(Pixel.Direction.EAST, null);
        if (x - 1 >= 0) neumannNeighbours.put(Pixel.Direction.WEST, this.data[x - 1][y]);
        else neumannNeighbours.put(Pixel.Direction.WEST, null);
        if (y + 1 < this.height) neumannNeighbours.put(Pixel.Direction.NORTH, this.data[x][y + 1]);
        else neumannNeighbours.put(Pixel.Direction.NORTH, null);
        if (y - 1 >= 0) neumannNeighbours.put(Pixel.Direction.SOUTH, this.data[x][y - 1]);
        else neumannNeighbours.put(Pixel.Direction.SOUTH, null);

        return neumannNeighbours;
    }

    public HashMap<Pixel.Direction, Pixel> getEmptyNeighbours(Pixel pixel) {
        HashMap<Pixel.Direction, Pixel> neumannNeighbours = getNeumannNeighbours(pixel);
        HashMap<Pixel.Direction, Pixel> emptyNeumannNeighbours = new HashMap<>(neumannNeighbours);
        for (Map.Entry<Pixel.Direction, Pixel> directionPixelEntry : neumannNeighbours.entrySet()) {
            Pixel value = directionPixelEntry.getValue();
            if (value != null && value.getPixelState() != Pixel.PixelState.EMPTY) {
                emptyNeumannNeighbours.remove(directionPixelEntry.getKey());
            }
        }
        return emptyNeumannNeighbours;
    }

    public List<Pixel> expand(Pixel pixel) {
        List<Pixel> newPixel = new ArrayList<>();
        Random random = new Random();
        for (Pixel.Direction value : Pixel.Direction.values()) {
            Pixel neumannNeighbour = this.getEmptyNeighbours(pixel).get(value);
            if (neumannNeighbour != null) {
                if (random.nextFloat() < pixel.getGrowSpeed().get(value)) {
                    neumannNeighbour.setColor(pixel.getColor());
                    neumannNeighbour.setPixelState(Pixel.PixelState.ACTIVE);
                    neumannNeighbour.setGrowSpeed(pixel.getGrowSpeed());
                    newPixel.add(neumannNeighbour);
                } else { // Pixel would be fillable, but was not due to the growSpeed
                    newPixel.add(pixel); // Pixel will be expanded another time
                }
            }
        }
        return newPixel;
    }

    public BufferedImage render(boolean saveFile, String fileName) {
```

```
72          BufferedImage image = new BufferedImage(this.width, this.height, BufferedImage.TYPE_INT_RGB);
            int total = 0;
74          for (int x = 0; x < this.width; x++) {
                for (int y = 0; y < this.height; y++) {
76                  if (this.data[x][y].getPixelState() == Pixel.PixelState.ACTIVE) {
                        image.setRGB(x, y, this.data[x][y].getColor().getRGB());
78                      total++;
                    } else if (this.data[x][y].getPixelState() == Pixel.PixelState.INACTIVE) {
80                      Color pixelColor = this.data[x][y].getColor();
                        image.setRGB(x, y,
82                        new Color(pixelColor.getRed(),
                                pixelColor.getGreen(),
84                              pixelColor.getBlue()).getRGB());
                        total++;
86                  }
                }
88          }
            try {
90               if (saveFile) ImageIO.write(image, "PNG", new File("./out/" + fileName + ".png"));
                return image;
92          } catch (IOException e) {
                e.printStackTrace();
94              return null;
            }
96      }
    }
```