Bundeswettbewerb Informatik
Runde 1

---

2 Verzinkt

Zur Lösungs dieses Problems wird zunächst eine Klasse `Seed` definiert. Sie hält die in der Aufgabenstellung definierten Charakteristika eines Kristallisationskeims. Sie entspricht folgender Form:

Seed
x_positive_speed : int y_positive_speed : int x_negative_speed : int y_negative_speed : int color_value : double

Die Attribute der Klasse werden über das `random` Modul zufällig im Konstruktor der Klasse zugeordnet. Das Attribut `color_value` c wird aus dem Winkel des Vektors, der sich aus der Addition Wachstumsgeschwindigkeiten ergibt, zu einem vordefinierten Normalvektor bestimmt:

$$\vec{g} = \begin{pmatrix} x_positive_speed - x_negative_speed \\ y_positive_speed - y_negative_speed \end{pmatrix}$$

$$\vec{n} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$c = \frac{arccos(\frac{\vec{n} * \vec{g}}{|\vec{n}| * |\vec{g}|})}{\pi}$$

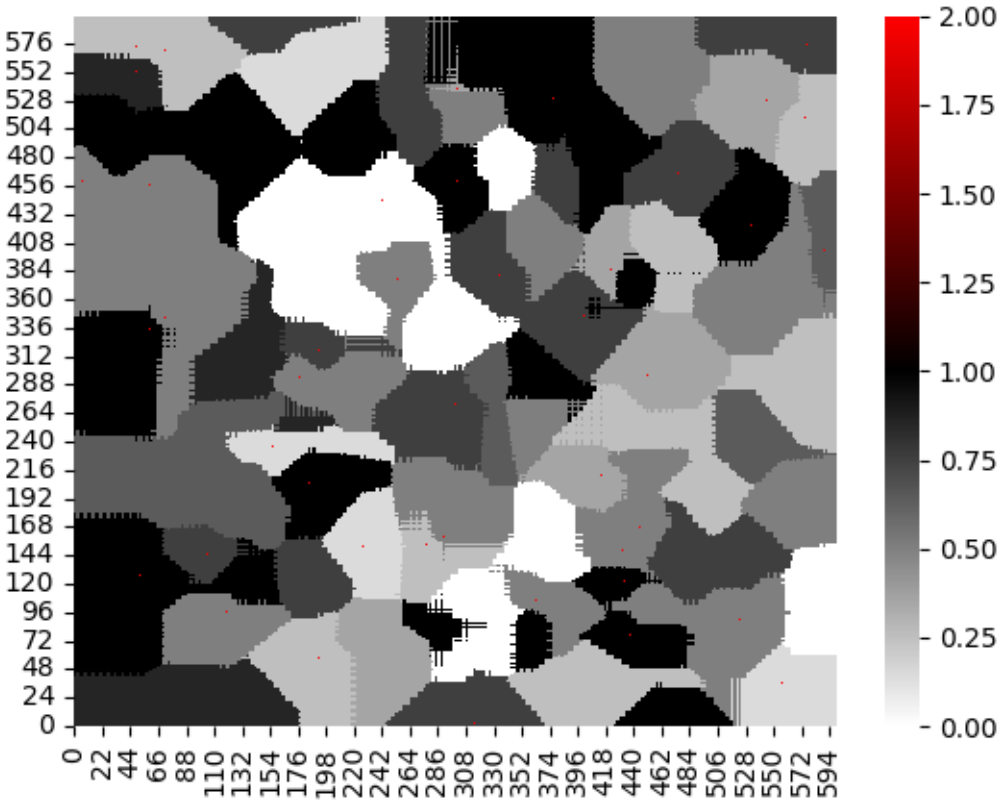
Jeder generierte Kristallisationskeim wird in einem Tupel mit dem Frame seiner Entstehung an einer zufälligen Stelle im zweidimensionalen Array `img` platziert. Alle leeren Positionen des Arrays erhalten das Tupel `(0,0)`.

$$img = \begin{pmatrix} (0,0) & (0,S_1) & \cdots & (0,0) \\ (0,0) & (0,0) & \cdots & (0,1) \\ \vdots & \vdots & \ddots & \vdots \\ (0,0] & (0,0) & \cdots & (0,S_2) \end{pmatrix}$$

Um alle Keime reproduzieren zu lassen wird die Methode `generate_next_frame(img` aufgerufen. Sie iteriert über jedes Element aus `img` und prüft, ob das erste Tupelelement vom Typ Seed ist und ob dieser sich in dieser Iteration reproduzieren soll.
Ist dies der Fall, geht das Programm in eine for-Schleife, die den Wert `dx` über das Intervall

{`-x_neg_speed`, `x_pos_speed`} iterieren lässt. Nun wird für jeden Wert `dx` geprüft, ob sich im Punkt `img[x+dx][y]` bereits ein Kristall befindet. Ist dies nicht der Fall wird im Punkt `img[x+dx][y]` ein neuer Tupel, bestehend aus dem Kristallisationskeim und dem Frame seiner Entstehung eingesetzt. Dieser Keim reproduziert sich im Folgenden Frame.
Nachdem die Iteration von `dx` abgeschlossen ist, wird der Vorgang in Y-Richtung wiederholt.
Es wird solange ein neues Frame produziert, bis das neue Frame dem alten gleicht, was bedeutet, dass jeder Bildpixel einen Kristall enthält und keine weiteren Änderungen mehr geschehen.

Mit einer Höhe und Breite von 600px, 100 zufällig, im ersten Frame generierten Kristallisationskeimen, Wachstumsgeschwindigkeiten im Intervall $\{4, 6\}$ ergibt sich folgendes Bild (Ursprünge der Kristalle sind rot markiert).



Da Ort und Wachstumsgeschwindigkeit eines Kristallisationskeims über den beschriebenen Ansatz bereits zufällig sind muss das bestehende Programm nicht erweitert werden, um diese Parameter zu variieren. Der Entstehungszeitpunkt der Keime ist allerdings noch auf das erste Frame begrenzt. Dies lässt sich durch die Einführung einer Wahrscheinlichkeit p zur Entstehung eines neuen Kristallisationskeims beim Rendern eines neuen Frames lösen. Hierzu wird der Hauptschleife der `generate_next_frame(img)` Methode, die jeden Pixel auf das Bestehen eines Kristalls hin überprüft, ein weiterer Prüffall hinzugefügt. Hierbei wird eine Zufallszahl im Intervall $\{0, \frac{1}{p}$ erzeugt und geprüft, ob diese 0 ist. Ist dies der Fall, so entsteht an dieser Stelle ein neuer, zufällig generierter Kristallisationkeim.

```
for y in range(0,HEIGHT):
  for x in range(0,WIDTH):
    if(isinstace(img[x][y][1], Seed) and img[x][y][0] == frame):
      //Reproduzieren des Kristalls wie beschrieben
    elif rd.randint(0,1/p) == 0:        //---NEU---//
          img[x][y] = (frame+1, Seed()) //---NEU---//
```

Um den das Ergebnis näher an das Bild der Aufgabenstellung zu bringen wurden alle alle Grautöne um einen Faktor von 2.5 aufgehellt. Mit dem Reduzieren der initialen Zahl an Kristallisationskeimen und einer Wahrscheinlichkeit von $\frac{1}{5000}$ zur Entstehung neuer Kristalle ergibt sich folgendes Bild.

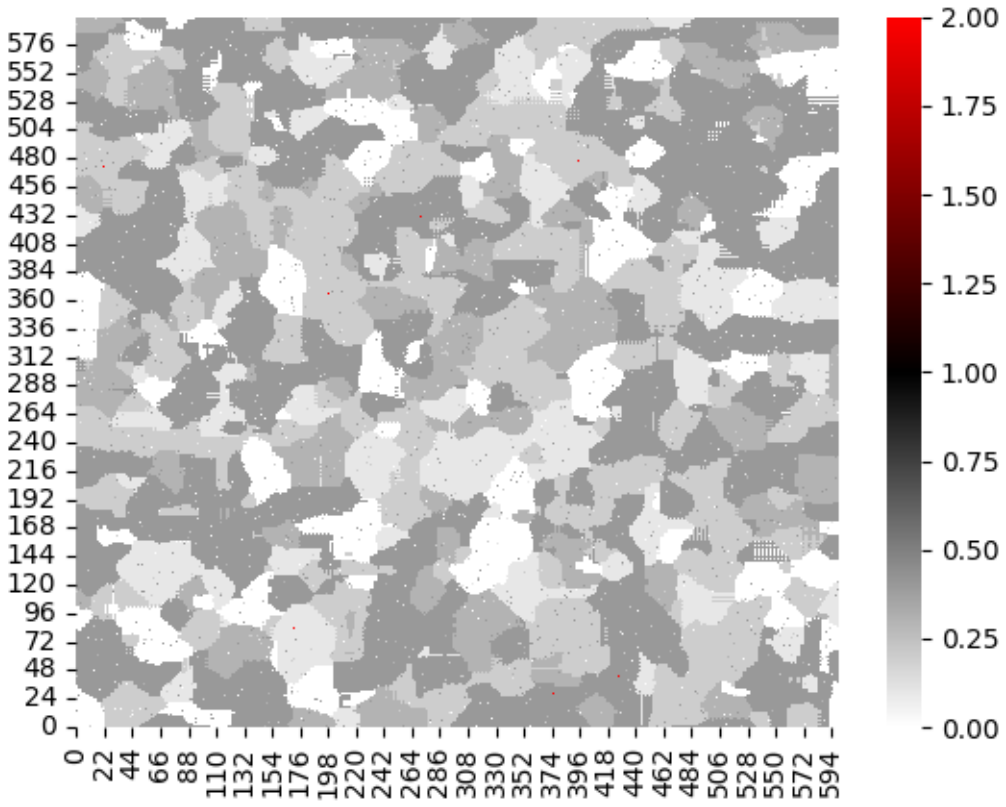


Abbildung 1: Finaler Render (#Seeds=20, p=1/5000)