Aufgabe 2: Verzinkt (lib/task2.dart)

Problem: Es soll die Kristallisierung von Keimen simuliert werden. Jeder Keim bzw. Kristall hat ein bestimmtes Wachstum in alle 4 Richtungen, eine Orientierung (Farbe), eine Spawn-Zeit (Zeit bis zum Spawn) und eine Position (x, y Koordinate). Kristalle dürfen sich nicht gegenseitig berühren oder überdecken.
Lösungsidee: Jeder Kristall bzw. Keim ist eine Klasse, welche die nötigen Informationen (Wachstum etc.) enthält. Eine sich nach einer bestimmten Zeit (Tick-Dauer) wiederholende Methode leitet bei jedem Keim das Wachstum ein. Ein Wachstum eines Kristalls ist dabei die Kreation eines neuen Kristalls, der anschließend auch weiter wächst und neue Kristalle spawnt.

Ich habe als erstes die bereits erwähnte Klasse (Crystal) konstruiert. Der Nutzer hat die Möglichkeit Grund-Kristalle oder Grund-Keime zu bestimmen. Der zentrale Bestandteil der Simulation ist die `List<List<int>> simulationMap`. Dies ist eine Liste, welche 500 Listen mit jeweils 500 Elementen hält. Sie fungiert wie ein Koordinatensystem mit jeweils 500 Elementen auf der x und y Achse. Die Position eines Kristalles ist in diesem Raster festgelegt. Dabei wird nur die Orientierung also der spätere Grau-Ton der Kristalls dem Raster hinzugefügt. Das Set `Set<Crystal> aktuelleKristalle` beinhaltet die aktuellen Kristalle, also die Kristalle, welche von der Simulation aktiv simuliert werden. Die vom Benutzer zu Beginn erstellten Keime sind auch Kristalle dieses Sets.
Wenn die Simulation gestartet wird, wird die `runTick(1)` Methode ausgeführt. Ein Tick entspricht einem Durchlauf der Simulation bzw. einem Ausführen der Methode und kann mit dem `tickDuration` Parameter zeitlich (also die Dauer) festgelegt werden.
Tick 0: Bei Tick 0 werden nur die vom Nutzer festgelegten Keime dargestellt und noch nicht simuliert. Die Simulation startet ab **Tick 1** und setzt sich solange fort, bis keine Kristalle mehr im Set `aktuelleKristalle` sind. In der `runTick` Methode wird durch das Set `aktuelleKristalle` iteriert und bei jedem Kristall wird die Methode `neueKristalle` ausgeführt:

```
      Set<'Crytal>  neueKirstalle =
          kristall.neueKristalle(simulationMap, tickNummer);
      if (neueKirstalle.length == 1 &&
          neueKirstalle.first.creationStatus == 4) {
        aktuelleKristalle.remove(kristall);
      } else {
        aktuelleKristalle.addAll(neueKirstalle);
      }
```

Jeder Kristall enthält die Variable `creationStatus`. Diese sagt aus, wie oft aus dem Kristall schon neue Kristalle hervorgegangen sind. Wenn der Kristall schon 4 Kristalle erzeugt hat also in alle Richtungen gewachsen ist, so wird dieser Kristall vom Set gelöscht. Die neu erzuegten Kristalle werden dem Set hinzugefügt. Die Methode `neueKristalle` funktioniert wie folgt:

```
 if (creationStatus == 4) {
      ausgabe.add(this);.
      return ausgabe;
 }
```

Es wird zunächst überprüft, ob der Kristall schon in alle Richtungen gewachsen ist (creationStatus == 4). Wenn ja wird der Kristall als Set zurückgegeben.

```
if (positionY > 0) {
      if (wachstumsGeschwindigkeiten[0] + startTickNummer <= currentTick) {
        /// wenn die Bedingung erfüllt ist, kann ein Kristall wachsen (oben)
        /// überprüfen, ob an dieser Stelle bereits ein Kristall exisitiert:
        if (simulationMap[positionY - 1][positionX] == -1) {
          /// frei
          /// in der simulationMap wird der Platz mit der Orientierung belegt
          simulationMap[positionY - 1][positionX] = orientierung;

          Crystal neuerKristall = Crystal(currentTick, orientierung, positionX,
              positionY - 1, wachstumsGeschwindigkeiten);
          ausgabe.add(neuerKristall);
        }else{
          if(simulationMap[positionY - 1][positionX] != orientierung){
            creationStatus += 1;
          }
        }
      }
    }
```

Als nächstes wird das "Wachsen" in alle 4 Richtungen probiert. Im Code oben handelt es sich um das Wachstum nach oben. Es wird zunächst überprüft, ob der Kristall, welcher durch das Wachstum eine neue Position im Raster hat, immernoch innerhalb des Rasters ist. Anschließend wird überprüft, ob ein Kristall (hier: nach oben) bei diesem Tick wachsen kann:
Jeder Kristall hat eine Liste mit der Anzahl an Ticks pro Richtung, welche für ein Wachstum erforderlich sind.
`wachstumsGeschwindigkeiten[0]` entspricht hier der Anzahl an Ticks, die erfolderlich sind, damit ein Kristall oben entsteht. Die Anzahl an Ticks (`startTickNummer`) bei der Erstellung des Kristalls wird als Grundwert genommen. D.h.
`wachstumsGeschwindigkeiten[0] + startTickNummer` entspricht dem Tick, ab dem das Wachstum möglich ist. Als nächstes wird überprüft, ob die Position, an der der neue Kristalle (der neu gewachsene Kristall) entstehen würde, in dem Raster (`simulationMap`) schon vergeben ist. Wenn nicht, wird an der Position der Orientierungs-Wert hinzugefügt. Nun wird der neue Kristall (`Crystal()`) erstellt und dem Set, das später ausgegeben wird, hinzugefügt. Nachdem alle Richtungen auf Wachstum überprüft wurden, werden ein Set mit neuen Kristallen zurückgegeben. Die neuen Kristalle haben hierbei die gleiche Orientierung und Wachstumsparamter, wie der ursprüngliche Kristall. Es ändern sich Position und die `startTickNummer` (zu `currentTick`). Wenn ein neuer Kristall erzeugt wurde oder wenn das Erstellen eines neuen Kristalls auf Grund von einem bereits dort existierenden Wert nicht möglich war, so wird der `creationStatus` um 1 erhöht (je nach Richtung, also maximal um 4).

Ein wichtiger Bestandteil der Simulation ist die Methode `sendToUI` :

```
img.Image image = img.Image.rgb(500, 500);
    for (var i = 0; i < simulationMap.length; i++) {
      for (var j = 0; j < simulationMap[i].length; j++) {
        final c = simulationMap[i][j];
        if (c == -1) {
          image.setPixelSafe(j, i, hexOfRGB(255, 255, 255));
        } else {
```

```
          image.setPixelSafe(j, i, hexOfRGB(c, c, c));
        }
      }
    }
```

Dort wird die `simulationMap` in ein Bild umgewandelt. Dieses Bild wird anschließend in die UI (mit `setState`) übertragen. Nach dem Druchlauf von `runTick()` wird die runTick Methode mit der Ticknummer + 1 als Paramter eingeleitet (Rekursion).

Ich habe ungefähr 30 zufällige Keime mit dem Programm erstellt und die Simulation ergibt ungefähr ein ähnliches Bild wie in der Aufgabenstellung. Das Programm kann frei für jede beispielhafte Simulation genutzt werden.