Aufgabe 2: Verzinkt

Team-ID: 00772

Team: Aino Spring

Bearbeiter/-innen dieser Aufgabe:
Aino Spring

17. November 2022

Inhaltsverzeichnis

Vorwort

Mit dieser Aufgabe bin ich nicht zufrieden, ich konnte sie nicht komplett fertigstellen, da ich erst später mit dem Wettbewerb begann und daher nur sehr begrenzte Zeit habe. Jedoch funktioniert sie und deshalb ist sie bei den Lösungen trotzdem enthalten. Diese Dokumentation wird nicht so ausführlich sein, da ich zu diesem Zeitpunkt nur sehr wenig Zeit habe.

Lösungsidee

Es wird über jede Position iteriert und wenn diese ein Kristall ist, wird für alle Richtungen überprüft, ob i (die Variable i wird jede „Runde“ um 1 inkrementiert) modulo die Geschwindigkeit dieser Richtung 0 ist. Wenn dies der Fall ist, wächst der Kristall in diese Richtung. Wenn Der Kristall auf einen anderen trifft, bleiben beide stehen.

Umsetzung

Jedes Feld besteht aus einem „Field“ struct. Es enthält einen 2-dimensionalen array mit allen Kristallen, die Größe und die längste Zeit, die ein Kristall braucht, um zu wachsen. Dies wird später noch einmal wichtig sein. Die einzelnen Kristalle können mit der „UpdateCell“ Funktion. Diese überprüft für alle Richtungen, ob die in der Lösungsidee genannte Variable i modulo die geschwindigkeit des Kristalls in diese Richtung 0 ist. Wenn dies der Fall ist, wächst der Kristall in diese Richtung. Alle Kristalle werden mir der „Update“ Funktion auf einmal aktualisiert. Darin wird für jeden Kristall, welcher nicht dieser Aktualisierung erstellt wurde, die „UpdateCell“ Funktion aufgerufen. In der „Update“ Funktion wird außerdem noch die Variable i erstellt und inkrementiert. Die „Generate“ Funktion führt die „Update“ Funktion so oft aus, bis so lang, wie der langsamste Kristall zum Wachsen braucht nichts mehr passiert ist. Dann ist die Funktion fertig.
Die „importCrystals“ Funktion importiert die „template.txt“-Datei. Diese enthält alle Kristalle. Die erste Zeile gibt an, wie groß das Feld ist. Die folgenden Zeilen enthalten alle Kristalle. Dabei sieht eine Zeile wie folgt aus:
<x>;<y>;<+x v>;<+y v>;<-x v>;<-y v>;<Orientierung>
v steht für Geschwindigkeit.
Dieses Template kann auch aus einer Bild-Datei erstellt werden. Dafür muss eine „template.png“-Datei enthalten sein. In dieser steht ein schwarzer Pixel für keinen Kristall und ein andersfarbiger für einen Kristall. Der Durchschnitt der r-g-b-Werte der Farbe (entspricht Farbe in Graustufen) wird als Orientierung des Kristalls verwendet. Die Geschwindigkeit des Kristalls wird durch eine Funktion ermittelt, welche als Parameter angegeben wurde. WICHTIG! Die template.txt-Datei wird hier mit der neuen Datei überschrienen.
Das alles wird als „Result.png“ ausgegeben. Es enthält die fertig gewachsenen Kristalle. Die Farbe ist die Orientierung. Die Datei „1.png“ zeigt die Kristalle, wie sie noch nicht gewachsen sind. Ein blauer Pixel ist kein Kristall.

Beispiele

Beispiel 1:

template.txt:

```
292;209
5;66;2;8;8;10;156
9;12;2;9;6;1;213
12;183;7;1;5;2;156
```

```
12;183;3;10;9;5;156
12;183;2;6;8;7;156
53;14;6;7;9;9;156
118;32;8;8;8;9;213
126;178;1;6;2;9;116
145;204;8;2;10;7;156
148;170;8;2;6;7;213
171;164;4;1;5;4;116
192;143;4;8;9;5;213
202;199;10;4;8;2;156
288;138;10;10;1;6;156
288;138;9;9;4;6;156
```

Beispiel 2:

template.txt:

```
292;209
5;66;82;88;48;60;156
9;148;82;19;26;41;162
9;149;57;1;95;12;162
9;12;63;90;29;75;213
12;183;12;46;38;7;156
12;183;96;67;29;59;156
12;183;48;48;88;89;156
39;106;91;16;42;9;107
53;14;88;32;30;57;156
53;181;38;32;86;27;175
53;181;14;91;95;64;175
62;59;34;48;79;25;169
88;183;60;54;58;22;87
91;95;90;100;1;6;84
96;143;89;39;4;56;51
96;144;52;11;6;57;51
118;32;67;29;62;3;213
126;178;84;47;64;77;116
141;90;3;19;48;95;59
145;204;78;64;97;21;156
148;170;24;54;38;34;213
167;100;42;60;34;44;17
```

```
171;164;92;3;79;37;116
173;99;47;8;41;4;159
181;35;53;44;6;99;119
182;35;26;52;16;58;119
192;143;88;11;11;86;213
202;199;91;33;99;54;156
227;48;92;83;85;98;107
248;149;68;38;72;95;85
254;101;27;3;82;80;127
274;20;67;71;94;87;105
288;138;20;82;53;76;156
288;138;86;11;88;50;156
```

Beisepiel 3:

template.txt:

```
292;209
5;66;82;88;48;60;97
9;148;82;19;26;41;105
9;149;57;1;95;12;105
9;12;63;90;29;75;152
12;183;12;46;38;7;97
12;90;96;67;29;59;89
12;183;48;48;88;89;97
12;183;91;16;42;9;97
26;36;88;32;30;57;89
36;60;38;32;86;27;89
37;60;14;91;95;64;89
39;106;34;48;79;25;76
49;159;60;54;58;22;100
53;14;90;100;1;6;97
53;181;89;39;4;56;89
53;181;52;11;6;57;89
57;86;67;29;62;3;89
62;59;84;47;64;77;119
88;183;3;19;48;95;89
91;95;78;64;97;21;87
96;143;24;54;38;34;89
```

```
96;144;42;60;34;44;89
105;63;92;3;79;37;94
108;108;47;8;41;4;117
108;108;53;44;6;99;117
118;32;26;52;16;58;152
118;87;88;11;11;86;89
122;119;91;33;99;54;92
122;158;92;83;85;98;79
126;178;68;38;72;95;74
136;61;27;3;82;80;80
141;90;67;71;94;87;89
142;101;20;82;53;76;92
143;136;86;11;88;50;97
143;136;29;19;85;4;97
145;204;25;48;13;33;97
148;170;17;40;41;87;152
156;102;52;77;41;52;89
157;52;45;65;6;84;89
157;52;2;91;3;59;89
158;30;68;32;79;55;71
167;76;23;24;43;9;115
167;76;44;69;67;11;115
167;100;36;41;5;63;12
171;164;58;16;72;40;74
173;99;31;14;1;60;89
173;126;21;84;71;85;115
181;35;48;11;66;63;89
182;35;30;21;49;57;89
192;143;96;67;1;57;152
194;69;30;93;32;78;182
197;60;87;21;100;63;116
197;60;48;93;89;12;116
202;199;4;89;19;57;97
206;114;20;8;58;53;75
211;88;76;82;54;96;92
227;48;18;94;71;97;76
228;12;87;33;21;61;89
243;67;23;30;62;61;129
248;149;21;80;55;65;79
251;192;61;52;82;58;89
251;192;17;1;40;38;89
253;133;34;62;5;86;89
254;101;10;16;20;15;80
274;20;41;63;41;1;78
288;138;85;74;36;44;97
288;138;22;91;75;70;97
```

Quellcode

```
type Field struct {
   field       [][]Crystal
   size        Vector
```

```
   longestTime int
}

func NewField(size Vector) (field Field) {
   field.size = size
   field.longestTime = -1
   field.field = make([][]Crystal, 0)
   for y := 0; y < size.y; y++ {
      field.field = append(field.field, make([]Crystal, size.x))
   }
   return
}

func (field *Field) Position(position Vector) *Crystal {
   return &field.field[position.y][position.x]
}

func (field *Field) SetCell(position Vector, crystal Crystal) bool {
   if !(0 <= position.x && position.x < field.size.x && 0 <= position.y &&
position.y < field.size.y) {
      return false
   }
   if positionPointer := field.Position(position); !positionPointer.notNull
{
      *positionPointer = crystal
      return true
   }
   return false
}

func (field *Field) UpdateCell(position Vector, frame int) ([]int, bool) {
   var currentlyMade = make([]int, 0)
   var changed = false
   if positionPointer := field.Position(position); positionPointer.notNull
{
      if (frame % positionPointer.seed.plusSpeed.x) == 0 {
         var currentPosition = position.Add(Vector{x: 1})
         var crystal = NewCrystal(positionPointer.seed)
         currentlyMade = append(currentlyMade, crystal.id)
         if field.SetCell(currentPosition, crystal) {
            changed = true
         }
      }
      if (frame % positionPointer.seed.plusSpeed.y) == 0 {
         var currentPosition = position.Add(Vector{y: 1})
         var crystal = NewCrystal(positionPointer.seed)
         currentlyMade = append(currentlyMade, crystal.id)
         if field.SetCell(currentPosition, crystal) {
            changed = true
         }
      }
      if (frame % positionPointer.seed.minusSpeed.x) == 0 {
         var currentPosition = position.Add(Vector{x: -1})
         var crystal = NewCrystal(positionPointer.seed)
```

```
            currentlyMade = append(currentlyMade, crystal.id)
            if field.SetCell(currentPosition, crystal) {
               changed = true
            }
         }
         if (frame % positionPointer.seed.minusSpeed.y) == 0 {
            var currentPosition = position.Add(Vector{y: -1})
            var crystal = NewCrystal(positionPointer.seed)
            currentlyMade = append(currentlyMade, crystal.id)
            if field.SetCell(currentPosition, crystal) {
               changed = true
            }
         }
      }
      return currentlyMade, changed
}

func (field *Field) AddCrystal(crystal Crystal, position Vector) {
      field.SetCell(position, crystal)
      for _, time := range []int{crystal.seed.plusSpeed.x, crys-
tal.seed.plusSpeed.y, crystal.seed.minusSpeed.x, crystal.seed.minusSpeed.y}
{
         if field.longestTime < time {
            field.longestTime = time
         }
      }
}

func (field *Field) Update(frame int) bool {
      var changed = false
      var finishedIds = make([]int, 0)
      for x := 0; x < field.size.x; x++ {
         for y := 0; y < field.size.y; y++ {
            var position = Vector{x, y}
            if positionPointer := field.Position(position); !SliceCon-
tains(finishedIds, positionPointer.id) {
               changedIds, fieldChanged := field.UpdateCell(position, frame)
               for _, id := range changedIds {
                  finishedIds = append(finishedIds, id)
               }
               changed = changed || fieldChanged
            }
         }
      }
      return changed
}

func (field *Field) Generate() {
      var noDeltaFrames = 0
      for i := 0; true; i++ {
         fmt.Printf("%v\n", i)
         var changed = field.Update(i)
         if !changed {
            noDeltaFrames++
```

```
        } else {
            noDeltaFrames = 0
        }
        if noDeltaFrames >= field.longestTime {
            break
        }
    }
}
```

```
var (
    crystalIds = 1
)

type Seed struct {
    orientation uint8
    plusSpeed   Vector
    minusSpeed  Vector
}

type Crystal struct {
    seed    Seed
    id      int
    notNull bool
}

func NewCrystal(seed Seed) (crystal Crystal) {
    crystal.seed = seed
    crystal.notNull = true
    crystal.id = crystalIds
    crystalIds++
    return
}
```

```
func importCrystals() (field Field) {
    data, err := os.ReadFile("template.txt")
    if err != nil {
        log.Panic(err)
    }
    var lines = strings.Split(strings.Replace(string(data), "\r", "", -1),
"\n")
    sizeX, err := strconv.Atoi(strings.Split(lines[0], ";")[0])
    if err != nil {
        log.Panic(err)
    }
    sizeY, err := strconv.Atoi(strings.Split(lines[0], ";")[1])
    if err != nil {
        log.Panic(err)
    }
    field = NewField(Vector{sizeX, sizeY})
    for _, line := range lines[1:] {
        var splitLine = strings.Split(line, ";")
        posX, err := strconv.Atoi(splitLine[0])
        if err != nil {
            log.Panic(err)
```

```
        }
        posY, err := strconv.Atoi(splitLine[1])
        if err != nil {
            log.Panic(err)
        }
        speedPlusX, err := strconv.Atoi(splitLine[2])
        if err != nil {
            log.Panic(err)
        }
        speedPlusY, err := strconv.Atoi(splitLine[3])
        if err != nil {
            log.Panic(err)
        }
        speedMinusX, err := strconv.Atoi(splitLine[4])
        if err != nil {
            log.Panic(err)
        }
        speedMinusY, err := strconv.Atoi(splitLine[5])
        if err != nil {
            log.Panic(err)
        }
        orientation, err := strconv.Atoi(splitLine[6])
        if err != nil {
            log.Panic(err)
        }
        var seed = Seed{
            orientation: uint8(orientation),
            plusSpeed:   Vector{speedPlusX, speedPlusY},
            minusSpeed:  Vector{speedMinusX, speedMinusY},
        }
        var crystal = NewCrystal(seed)
        field.AddCrystal(crystal, Vector{posX, posY})
    }
    return field
}
```

```
func GenerateTemplate(path string, generateSpeed func(int, int) (Vector,
Vector), mult float64) {
    imageReader, err := os.Open(path)
    if err != nil {
        log.Panic(err)
    }
    image, err := png.Decode(imageReader)
    if err != nil {
        log.Panic(err)
    }
    imageConfigReader, err := os.Open(path)
    if err != nil {
        log.Panic(err)
    }
    imageConfig, err := png.DecodeConfig(imageConfigReader)
    if err != nil {
        log.Panic(err)
    }
```

```
   var templateData = fmt.Sprintf("%v;%v", int(float64(imageCon-
fig.Width)*mult), int(float64(imageConfig.Height)*mult))
   for x := 0; x < imageConfig.Width; x++ {
      for y := 0; y < imageConfig.Height; y++ {
         r, g, b, _ := image.At(x, y).RGBA()
         r, g, b = r/255, g/255, b/255
         if int((r+g+b)/3) == 0 {
            continue
         }
         fmt.Printf("%v, %v, %v\n", r, g, b)
         plusSpeed, minusSpeed := generateSpeed(x, y)
         templateData += fmt.Sprintf("\n%v;%v;%v;%v;%v;%v",
int(float64(x)*mult), int(float64(y)*mult), plusSpeed.x, plusSpeed.y, mi-
nusSpeed.x, minusSpeed.y, (r+g+b)/3)
      }
   }
   file, err := os.Create("template.txt")
   if err != nil {
      log.Panic(err)
   }
   _, err = file.Write([]byte(templateData))
   if err != nil {
      log.Panic(err)
   }
}
```