40. Bundeswettbewerb für Informatik - Aufgabe 3 (Wortsuche)

Aufgabenstellung

Schreibe ein Programm, das eine gegebene Wortliste und die Größe des Rechtecks einliest und daraus Buchstabenrechtecke entsprechend deiner Schwierigkeitsgrade erzeugt.

Lösungsidee

Das Hauptziel des Algorithmuses ist, dass verschiedene Wortfelder nach verschiedenen Schwierigkeitsgraden generiert werden sollen.

Schwierigkeitsgrad **LEICHT**

Beim einfachsten Schwierigkeitsgrad des Algorithmuses können Wörter nur horizontal und vertikal positioniert werden und dürfen sich dabei nicht überschneiden. Des Weiteren werden die übrigen Stellen mit zufälligen Buchstaben aufgefüllt.

Schwierigkeitsgrad **MITTEL**

Bei diesem Schwierigkeitsgrad des Algorithmuses können Wörter sowohl horizontal, als auch vertikal und diagonal (nicht überschneidend) platziert werden. Die übrigen Stellen werden hierbei nur mit zufälligen Buchstaben aufgefüllt, welche in den Wörtern der Wortliste enthalten sind (können auch alle sein).

Schwierigkeitsgrad **SCHWER**

Beim letzten Schwierigkeitsgrad des Algorithmuses können Wörter horizontal, vertikal und diagonal platziert werden und können sich dabei auch überschneiden. Außerdem werden die Leerstellen mit Fragment der Wörter der Wortliste aufgefüllt (keine zufälligen Buchstaben).

Verwendung des Programmes

Das Programm zur Lösung dieses Problemes befindet sich in der Datei "Aufgabe_3.jar". Das Programm kann mit der Befehlszeile (CMD auf Windows bzw. Temrinal auf MacOS) aufgeführt werden. Dafür navigiert man zuerst in den Ordner der JAR-Datei (hier Aufgabe 3). Anschließend führt man den Befehl "java -jar "Aufgabe_3.jar <Eingabedatei> <Ausgabedatei>" aus. Die Ausgabedatei ist optional. Wenn keine Ausgabedatei angegeben ist, dann wird das Ergebnis als "output.txt" in dem Ordner der JAR-Datei gespeichert.

Implementierung

Einlesen der Beispieldateien

```
//Lesen der einzelnen Zeilen der Datei
String line;
while((line = streamReader.readLine()) != null)
```

```
contentBuilder.append(line).append("\n");

//Aufteilen des gelesenen Textes in die einzelnen Dateien
String[] contentLines = contentBuilder.toString().split("\n");

//Interpretieren der Daten
//Einlesen der Höhe und der Breite der Wortliste
height = Integer.parseInt(contentLines[0].split(" ")[0]);
width = Integer.parseInt(contentLines[0].split(" ")[1]);

//Einlesen der Wortanzahl
wordCount = Integer.parseInt(contentLines[1]);

//Einlesen der Wortliste
words.addAll(Arrays.asList(contentLines).subList(2, wordCount + 2));
```

Allgemein

Die abstrakte Klasse "PatternGenerator" beeinhaltet die Grundfunktionen/-methoden und die Hauptschleife des Algorithmuses. Die Generatoren für die verschiedenen Schwierigkeitsgrade sind von dieser Klasse abgeleitet.

```
public String generatePattern() {
    //Variablen werden initialisiert
    int placementAttempts = 0;
    pattern = new String[height][width];
    placedWords.clear();
    closedPositions.clear();

    //Wortliste wird nach der Größe absteigend sortiert und der Queue
hinzugefügt
    Collections.sort(words);
    Collections.reverse(words);
    wordQueue.addAll(words);

    //Alle Felder des Wortfeldes werden mit einem Leerzeichen aufgefüllt
    prepareEmptySpaces();

    //Hauptschleife des Algorithmus
    while(wordQueue.peek() != null) { //Solange Elemente in der Queue
enthalten sind
        placementAttempts++; //Versuchszahl wird erhöht
        if(placementAttempts > wordCount * 3) {
            //Bei zu vielen Versuchen wird der Algorithmus abgebrochen,
sodass kein Stackoverflow entsteht
            System.err.println("Failed to fit all words on the board.
Restarting...");
            return generatePattern();
        }

        //Das Wort wird aus der Queue entfernt, wenn es erfolgreich
```

```
platziert werden konnte
        if(placeWord(wordQueue.peekFirst())) wordQueue.removeFirst();
    }

    //Verbleibende Leerstellen auffüllen
    fillEmptySpaces();

    //Rückgabe des generierten Wortfeldes
    return formatMatrix(pattern);
}
```

Der Algorithmus zum Generieren eines Wortes arbeitet mit einer while-Schleife, welche ein Deque-Objekt durchläuft. In der Queue (Warteschlange) befinden sich alle Wörter, welche noch auf dem Wortfeld platziert werden sollen. Die Wörter werden der Länge nach absteigend sortiert der Queue hinzugefügt, sodass die längsten Wörter zuerst platziert werden. Die while-Schleife läuft dann, solange sich Elemente (Wörter) in der Queue befinden. Allerdings wird die Schleife automatisch abgebrochen, wenn die Schleife 3-mal so lang, wie die Wortliste eigentlich ist durchlaufen wurde. Wenn sich also 10 Wörter in der Queue befinden, dann wird der Algorithmus nach 30 Iterationen spätestens abgebrochen, da es dann sehr wahrscheinlich ist, dass die Wörter nicht alle gleichzeitig auf dem Wortfeld platziert werden können.

Der Algorithmus wählt also immer das erste Wort aus der Queue und versucht dieses über die Funktion #placeWord dem Feld hinzuzufügen. Diese Funktion wird durch die verschiedenen Klassen der verschiedenen Schwierigkeitsgrade implementiert und gibt einen boolean zurück, welcher angibt, ob das Wort erfolgreich hinzugefügt werden konnte. Wenn der Vorgang erfolgreich war, wird das Wort aus der Queue entfernt und das nächste Wort in der Warteschlange wird bearbeitet. Wenn das Wort nicht hinzugefügt werden konnte, dann verbleibt es in der Queue und wird anschließend erneut abgearbeitet.

Wenn alle Wörter dem Wortfeld hinzugefügt werden konnte, füllt der Algorithmus alle verbleibenden leeren Felder auf. Wie diese aufgefüllt werden hängt jedoch vom Schwierigkeitsgrad ab und wird folglich von den einzelnen Generatorklassen implementiert.

Zum Schluss formatiert der Algorithmus das Wortfeld von einem zweidimensionalen Array in einen String und gibt diesen zurück, sodass dieser ausgegebenen werden kann.

Schwierigkeitsgrad: Leicht

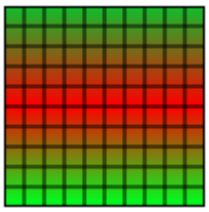
Die Klasse "EasyPatternGenerator" bildet die Grundlage für das Generieren eines Wortfeldes, da alle Klassen der höheren Schwierigkeitsklasse von dieser ableiten. Der Algorithmus zur Generierung eines solchen Feldes ist für diesen Schwierigkeitsgrad wie folgt aufgebaut:
1. Die Orientierung des Wortes wird zufällig generiert (Horizontal oder Vertikal)
2. Die Position des Wortes wird mithilfe eines weiteren Zufallsgenerators und einer bestimmten Gewichtung generiert
3. Das Wort wird an der generierten Position hinzugefügt
a. Wenn dies fehlschlägt, dann versucht der Algorithmus es erneut (maximal 3 Versuche, bevor false zurückgegeben wird)
b. Wenn der Vorgang erfolgreich war, wird true zurückgegeben, sodass das nächste Wort bearbeitet werden kann
4. Die verbleibenden leeren Felder werden mit zufälligen Buchstaben aus dem Alphabet aufgefüllt

```
//Berechnung einer zufälligen Zahl
switch(instanceRandom.nextInt(2)) {
    //Das neue Wort wird horizontal ausgerichet
    case 0:
        //Die Positionierung wird 3 mal wiederholt, falls es nicht
funktionieren sollte
        for(int i = 0; i <= maxAttempts; i++) {
            int[] position = getPositionForWord(word,
WordPosition.Orientation.Horizontal);
            if(placeWordHorizontally(word, position[0], position[1],
false))
                //Die Positionierung war erfolgreich
                return true;
        }

        //Alle drei Versuche zur Positionierung des Wortes schlugen fehl
        return false;
    case 1:
        //Die Positionierung wird 3 mal wiederholt, falls es nicht
funktionieren sollte
        for(int i = 0; i <= maxAttempts; i++) {
            int[] position = getPositionForWord(word,
WordPosition.Orientation.Vertical);
            if(placeWordVertically(word, position[0], position[1], false))
                //Die Positionierung war erfolgreich
                return true;
        }

        //Alle drei Versuche zur Positionierung des Wortes schlugen fehl
        return false;
    default: return false;
}
```

Beim ersten Schwierigkeitsgrad ergibt sich das Problem, dass unter Umständen nicht immer alle Wörter direkt auf das Wortfeld passen, wenn diese vollständig zufällig platziert werden. Daher gibt es bei diesem Schwierigkeitsgrad bestimmte Bereiche, in den die Wörter besonders häufig vorkommen.

Bei horizontaler Ausrichtung

Wenn das Wort horizontal ausgerichtet werden soll, dann ist die Wahrscheinlichkeit, dass die Wörter am Rand des Wortfeldes auftauchen höher, als in der Mitte. Dabei spielt die X-Koorindate des Wortes keine Rolle und wird zufällig generiert.
Die Y-Koorindate wird hier jedoch nach einer bestimmten Wahrscheinlichkeit generiert. Diese Verteilung wurde gewählt, um möglichst viel Platz für die anderen Wörter zu ermöglichen (vertikal bzw. hauptsächlich für diagonal). Dazu teilt der Algorithmus das Wortfeld in drei gleich große Bereiche auf, welche jeweils die Wahrscheinlichkeiten 40 %, 20 % und 40 % haben.
Nachdem der Algorithmus einen bestimmten Bereich anhand der Wahrscheinlichkeiten gewählt hat, wird die genaue Position des Wortes innerhalb dieses Bereiches zufällig generiert. Diese Position wird dann zum Schluss zurückgegeben.

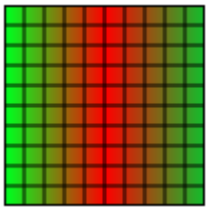


```
//X-Koorindate wird zufällig generiert
int hx = instanceRandom.nextInt(width - word.length());

int[][] verticalSections = new int[3][2];
int verticalSectionIndex = 0;

//Einteilung des Feldes in drei gleich große Bereiche
//Erstes Drittel des Feldes
verticalSections[0][0] = 0;
verticalSections[0][1] = height / 3;

//Zweites Drittel des Feldes
verticalSections[1][0] = height / 3;
verticalSections[1][1] = (height / 3) * 2;

//Letztes Drittel des Feldes
verticalSections[2][0] = (height / 3) * 2;
verticalSections[2][1] = height;

//Zufällige Auswahl des Bereiches nach den verschiedenen
Wahrscheinlichkeiten
int verticalSectionSelection = instanceRandom.nextInt(100);

//Erstes Drittel (40 %)
if(verticalSectionSelection < 40) verticalSectionIndex = 0;
//Zweites Drittel (20 %)
else if(verticalSectionSelection < 60) verticalSectionIndex = 1;
//Letztes Drittel (40 %)
else verticalSectionIndex = 2;

//Auswahl einer zufälligen Koorindate innerhalb des ausgewählten Bereiches
int hy = yGenerator.generate(verticalSections[verticalSectionIndex][1] -
verticalSections[verticalSectionIndex][0]) +
verticalSections[verticalSectionIndex][0];

//Rückgabe der berechneten Koordinaten
return new int[] { hx, hy };
```

Bei vertikaler Ausrichtung

Wenn das Wort vertikal ausgerichtet werden soll, dann ist die Wahrscheinlichkeit, dass die Wörter am Rand des Wortfeldes auftauchen höher, als in der Mitte. Dabei spielt die Y-Koorindate des Wortes keine Rolle und wird zufällig generiert.
Die X-Koorindate wird hier jedoch nach einer bestimmten Wahrscheinlichkeit generiert. Diese Verteilung

wurde gewählt, um möglichst viel Platz für die anderen Wörter zu ermöglichen (horizontal bzw. hauptsächlich diagonal).
Dazu teilt der Algorithmus das Wortfeld in drei gleich große Bereiche auf, welche jeweils die Wahrscheinlichkeiten 40 %, 20% und 40 % haben.
Nachdem der Algorithmus einen bestimmten Bereich anhand der Wahrscheinlichkeiten gewählt hat, wird die genaue Position des Wortes innerhalb dieses Bereiches zufällig generiert. Diese Position wird dann zum Schluss zurückgegeben.



```
//Y-Koorindate wird zufällig generiert
int vy = instanceRandom.nextInt(height - word.length());

int[][] horizontalSections = new int[3][2];
int horizontalSectionIndex = 0;

//Einteilung des Feldes in drei gleich große Bereiche
//Erstes Drittel des Feldes
horizontalSections[0][0] = 0;
horizontalSections[0][1] = width / 3;

//Zweites Drittel des Feldes
horizontalSections[1][0] = width / 3;
horizontalSections[1][1] = (width / 3) * 2;

//Letztes Drittel des Feldes
horizontalSections[2][0] = (width / 3) * 2;
horizontalSections[2][1] = width;

//Zufällige Auswahl des Bereiches nach den verschiedenen
Wahrscheinlichkeiten
int horizontalSectionSelection = instanceRandom.nextInt(100);

//Erstes Drittel (40 %)
if(horizontalSectionSelection < 40) horizontalSectionIndex = 0;
//Zweites Drittel (20 %)
else if(horizontalSectionSelection < 60) horizontalSectionIndex = 1;
//Letztes Drittel (40 %)
else horizontalSectionIndex = 2;

//Auswahl einer zufälligen Koordinate innerhalb des ausgewählten Bereiches
int vx = xGenerator.generate(horizontalSections[horizontalSectionIndex][1]
- horizontalSections[horizontalSectionIndex][0]) +
horizontalSections[horizontalSectionIndex][0];

//Rückgabe der berechneten Koordinaten
return new int[] { vx, vy };
```

Horizontale Platzierung der Wörter

Um ein Wort horizontal zu platzieren wird eine zufällige Position generiert, wenn dies vorher nicht geschehen ist. Anschließend überprüft der Algorithmus, ob an der Stelle im Feld überhaupt genügend platz vorhanden ist. Dazu durchläuft eine Schleife alle Positionen von der X-Koordinate bis zur Position X + Länge des Wortes. Wenn alle Positionen frei sind, dann fährt der Algorithmus fort und platziert das Wort mithilfe einer Schleife auf dem Feld. Andernfalls wird hier false zurückgegeben, da das Wort nicht platziert werden konnte.
Nachdem das Wort positioniert wurde, wird das Wort in die Liste der positionierten Wörter aufgenommen und die Position wird in eine Ausnahmeliste des Zufallsgenerators aufgenommen, sodass die gleiche Position nicht zufällig erneut ausgewählt werden kann.
Der Parameter "crossingAllowed" hat für diesen Schwierigkeitsgrad keine Bedeutung.

```
protected boolean placeWordHorizontally(String word, int optionalX, int
optionalY, boolean crossingAllowed) {
    //Initialisieren der Koordinaten
    int positionX = optionalX;
    int positionY = optionalY;

    //Berechnen der Koordinaten
    if(positionY == COORDINATE_GENERATE) positionY =
yGenerator.generate(pattern.length);
    if(positionX == COORDINATE_GENERATE) positionX =
pattern[positionY].length - word.length() != 0 ?
xGenerator.generate(pattern[positionY].length - word.length()) : 0;

    //Überprüfen der Position, in welcher das Wort platziert werden soll
    for(int i = 0; i < word.length(); i++) {
        if(!pattern[positionY][positionX + i].equals(CHARACTER_EMPTY) &&
!pattern[positionY][positionX + i].equals(String.valueOf(word.charAt(i))))
{
            return false;
        }
    }

    //Platzieren des Wortes auf dem Board
    for(int x = 0; x < word.length(); x++) pattern[positionY][positionX +
x] = String.valueOf(word.charAt(x));
    placedWords.put(word, new WordPosition(positionX, positionY,
WordPosition.Orientation.Horizontal));
    addClosedCoordinate(positionX, positionY);
    return true;
}
```

Vertikale Platzierung der Wörter

Um ein Wort vertikal zu platzieren wird eine zufällige Position generiert, wenn dies vorher nicht geschehen ist. Anschließend überprüft der Algorithmus, ob an der Stelle im Feld überhaupt genügend Platz vorhanden ist. Dazu durchläuft eine Schleife alle Positionen von der Y-Koordinate bis zur Position Y + Länge des Wortes. Wenn alle Positionen frei sind, dann fährt der Algorithmus fort und platziert das Wort mithilfe einer Schleife auf dem Feld. Andernfalls wird hier false zurückgegeben, da das Wort nicht platziert werden konnte.
Nachdem das Wort positioniert wurde, wird das Wort in die Liste der positionierten Wörter aufgenommen und die Position wird in eine Ausnahmeliste des Zufallsgenerators aufgenommen, sodass die gleiche Position nicht zufällig erneut ausgewählt werden kann.
Der Parameter "crossingAllowed" hat für diesen Schwierigkeitsgrad keine Bedeutung.

```
protected boolean placeWordVertically(String word, int optionalX, int
optionalY, boolean crossingAllowed) {
    //Initialisieren der Koordindaten
    int positionX = optionalX;
    int positionY = optionalY;

    //Berechnen der Koordinaten
    if(positionY == COORDINATE_GENERATE) positionY = pattern.length -
word.length() != 0 ? yGenerator.generate(pattern.length - word.length()) :
0;
    if(positionX == COORDINATE_GENERATE) positionX =
xGenerator.generate(pattern[positionY].length);

    //Überprüfen der Position, in welcher das Wort platziert werden soll
    for(int i = 0; i < word.length(); i++) {
        if(!pattern[positionY + i][positionX].equals(CHARACTER_EMPTY) &&
(!pattern[positionY + i]
[positionX].equals(String.valueOf(word.charAt(i))))) {
            return false;
        }
    }

    //Platzieren des Wortes auf dem Board
    for(int y = 0; y < word.length(); y++) pattern[positionY + y]
[positionX] = String.valueOf(word.charAt(y));
    placedWords.put(word, new WordPosition(positionX, positionY,
WordPosition.Orientation.Vertical));
    addClosedCoordinate(positionX, positionY);
    return true;
}
```

Schwierigkeitsgrad: Mittel

Zum Generieren von Wortfeldern des mittleren Schwierigkeitsgrades wird die Klasse "MediumPatternGenerator" verwendet, welche von der Klasse zur Generierung von leichten Wortfeldern abgeleitet ist. Daher beeinhaltet der Schwierigkeitsgrad Mittel alle Abfolgen des Schwierigkeitsgrades Leicht mit einigen Ergänzungen.

Die erste Ergänzung ist, dass Wörter hier nun auch diagonal (aufsteigend und absteigend) positioniert werden können.

Diagonal aufsteigende Positionierung eines Wortes

Um ein Wort diagonal aufsteigend zu platzieren wird eine zufällige Position generiert, wenn dies vorher nicht geschehen ist. Anschließend überprüft der Algorithmus, ob an der Stelle im Feld überhaupt genügend Platz vorhanden ist. Dazu durchläuft eine Schleife alle Positionen von der X-Koordinate zur Position X + Länge des Wortes und alle Positionen von der Y-Koordinate zur Position Y - Länge des Wortes. Wenn alle Positionen frei sind, dann fährt der Algorithmus fort und platziert das Wort mithilfe einer Schleife auf dem Feld. Andernfalls wird hier false zurückgegeben, da das Wort nicht platziert werden konnte.
Nachdem das Wort positioniert wurde, wird das Wort in die Liste der positionierten Wörter aufgenommen und die Position wird in eine Ausnahmeliste des Zufallsgenerators aufgenommen, sodass die gleiche Position nicht zufällig erneut ausgewählt werden kann. Diagonale Wörter können außerdem nur an Positionen platziert werden, welche gerade Zahlen sind (x % 2 == 0), da ansonsten Überschneidungen entstehen, welche ungültig sind. Der Parameter "crossingAllowed" hat für diesen Schwierigkeitsgrad keine Bedeutung.

```
//Berechnen der Koordinaten für die Richtung: diagonal aufsteigend
if(positionY == COORDINATE_GENERATE) positionY = pattern.length -
word.length() != 0 ? yGenerator.generate(pattern.length - word.length()) +
word.length() - 1 : pattern.length - 1;
if(positionX == COORDINATE_GENERATE) positionX = pattern[positionY].length
- word.length() != 0 ? xGenerator.generate(pattern[positionY].length -
word.length()) : 0;

//Überprüfen der Position auf dem Wortfeld
for(int i = 0; i < word.length(); i++) {
    if((positionX % 2 == 0 || positionY % 2 == 0) || (!pattern[positionY -
i][positionX + i].equals(CHARACTER_EMPTY) && (!pattern[positionY - i]
[positionX + i].equals(String.valueOf(word.charAt(i)))))) {
        return false;
    }
}

//Platzieren des Wortes auf dem Wortfeld
for(int i = 0; i < word.length(); i++) pattern[positionY - i][positionX +
i] = String.valueOf(word.charAt(i));
placedWords.put(word, new WordPosition(positionX, positionY,
WordPosition.Orientation.DiagonalUp));
```

Diagonal absteigende Positionierung eines Wortes

Um ein Wort diagonal absteigend zu platzieren wird eine zufällige Position generiert, wenn dies vorher nicht geschehen ist. Anschließend überprüft der Algorithmus, ob an der Stelle im Feld überhaupt genügend Platz vorhanden ist. Dazu durchläuft eine Schleife alle Positionen von der X-Koordinate bis zur Position X + Länge des Wortes und von der Y-Koordinate zur Position Y + Länge des Wortes. Wenn alle Positionen frei sind, dann fährt der Algorithmus fort und platziert das Wort mithilfe einer Schleife auf dem Feld. Andernfalls wird

hier false zurückgegeben, da das Wort nicht platziert werden konnte. Nachdem das Wort positioniert wurde, wird das Wort in die Liste der positionierten Wörter aufgenommen und die Position wird in eine Ausnahmeliste des Zufallsgenerators aufgenommenm sodass die gleiche Position nicht zufällig erneut ausgewählt werden kann. Diagonale Wörter können außerdem nur an Positionen platziert werden, welche gerade Zahlen sind (x % 2 == 0), da ansonsten Überschneidungen entstehen, welche ungültig sind. Der Parameter "crossingAllowed" hat für diesen Schwierigkeitsgrad keine Bedeutung.

```
 //Berechnen der Koordinaten für die Richtung: diagonal absteigend
if(positionY == COORDINATE_GENERATE) positionY = pattern.length -
word.length() != 0 ? yGenerator.generate(pattern.length - word.length()) :
0;
if(positionX == COORDINATE_GENERATE) positionX = pattern[positionY].length
- word.length() != 0 ? xGenerator.generate(pattern[positionY].length -
word.length()) : 0;

//Überprüfen der Position auf dem Wortfeld
for(int i = 0; i < word.length(); i++) {
    if((positionX % 2 == 0 || positionY % 2 == 0) || (!pattern[positionY +
i][positionX + i].equals(CHARACTER_EMPTY) && (!pattern[positionY + i]
[positionX + i].equals(String.valueOf(word.charAt(i)))))) {
        return false;
    }
}

//Platzieren des Wortes auf dem Wortfeld
for(int i = 0; i < word.length(); i++) pattern[positionY + i][positionX +
i] = String.valueOf(word.charAt(i));
placedWords.put(word, new WordPosition(positionX, positionY,
WordPosition.Orientation.DiagonalDown));
```

Auffüllen von leeren Stellen

Der Algorithmus zur Generierung der Wortfelder von diesem Schwierigkeitsgrad verwende eine andere Implementierung zur Auffüllung der Leerzeichen, welche am Ende aufgefüllt werden sollen. Im Gegensatz zu den einfachen Wortfeldern werden hier keine zufälligen Buchstaben mehr verwendete. Stattdessen verwendet der Algorithmus nur Buchstaben, welche auch in den Wörtern aus der Wortliste enthalten sind. In einigen Fällen kann dies aber auch dazu führen, dass dies das ganze Alphabet umfasst.

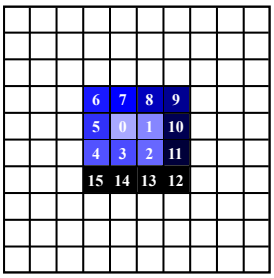
```
//Sammeln aller Buchstaben, welche in den Wörtern aus der Wortliste
enthalten sind
ArrayList<String> letterSet = new ArrayList<>();
for(String word : words) {
    for(int i = 0; i < word.length(); i++) {
        if(!letterSet.contains(String.valueOf(word.charAt(i))))
            letterSet.add(String.valueOf(word.charAt(i)));
    }
}

//Auffüllen der Leerstellen mit zufälligen Buchstaben aus der Liste
```

```
for(int x = 0; x < width; x++) {
    for(int y = 0; y < height; y++) {
        if(Objects.equals(pattern[y][x], CHARACTER_EMPTY)) {
            //Hinzufügen des Buchstabens zum Wortfeld
            pattern[y][x] =
letterSet.get(instanceRandom.nextInt(letterSet.size()));

            //Speichern des Punktes zur späteren Überprüfung der
Auffüllung
            filledPoints.add(new Point(x, y));
        }
    }
}

//Überprüfung der automatischen Auffüllung
checkFilledSpaces(letterSet);
```

Dazu sammelt der Algorithmus zuerst alle Buchstaben, welche in den Wörtern aus der Wortliste enthalten sind. Anschließend füllt er alle Leerstellen mit zufälligen Buchstaben aus dieser Liste auf. Dabei werden die Stellen, welche aufgefüllt werden für den späteren Zeitpunkt der Überprüfung gespeichert.

Überprüfung der Auffüllung

Nachdem die leeren Stellen aufgefüllt wurden, muss der Algorithmus das Feld erneut überprüfen, da es unter Umständen passieren kann, dass durch das Auffüllen Wörter aus der Wortliste entstanden sind und folglich doppelt enthalten sind.
Zur Überprüfung der Auffüllung verwendet der Algorithmus eine Schleife, welche das Wortfeld von innen nach außen spiralförmig durchläuft (siehe Abbildung).



Der folgende Code wird verwendet, um die spiralförmige Schleife zu realisieren:

```
int currentX = 0, currentY = 0; //Aktuelle Position
int deltaX = 0, deltaY = -1; //Delta der beiden Positionen, welche bei der
nächsten Iteration angewendet werden

//Höhe/Breite werden hier als Spalten/Zeilen bezeichnet, da der
zugrundeliegende Algorithmus ursprünglich zum Durchlaufen von Matrizen
entwickelt wurde.
int rows = height;
int cols = width;

//Hauptschleife, welche das Feld spiralförmig durchläuft
```

```
for(int i = 0; i < Math.pow(Math.max(rows, cols), 2); i++) {
    if(currentX >= -rows / 2 && currentX <= rows / 2 && currentY >= -cols
/ 2 && currentY <= cols / 2) {
        //Umrechnung, da currentX und currentY den Abstand vom Mittelpunkt
des Feldes angeben und nicht die absolute Position
        int actualX = (int)Math.ceil((width / 2.0) + (double)currentX);
        int actualY = (int)Math.ceil((height / 2.0) + (double)currentY);

        if(actualX > 0 && actualX < width && actualY > 0 && actualY <
height) {
            //Überprüfen der Buchstaben
            checkFilledPosition(actualX, actualY, letterSet);
        }
    }

    if((currentX == currentY) || (currentX == -currentY && currentX < 0)
|| (currentX == 1 - currentY && currentX > 0)) {
        //Richtungsumkehr
        int copy = deltaX;
        deltaX = -deltaY;
        deltaY = copy;
    }

    //Anwenden der Deltas auf die aktuelle Position
    currentX = currentX + deltaX;
    currentY = currentY + deltaY;
}
```

Die Methode #checkFilledPosition ist die eigentliche Methode, welche die Platzierung der Buchstaben überprüft und gegebenenfalls ersetzt. Diese Methode überprüft das Umfeld des Buchstabens in einem Radius von drei Einheiten und ersetzt die Buchstaben gegebenenfalls, sodass keine doppelten Wörter entstehen. Die Methode überprüft vorher, ob die Stelle, welche überprüft wird, überhaupt überprüft werden muss. Dazu verwendet sie die Liste an Positionen, welche aufgefüllt wurden. Diese Liste wurde im vorherigen Schritt (Auffüllungen) erzeugt.

```
protected void checkFilledPosition(int posX, int posY, ArrayList<String>
fullLetterSet) {
    //Überprüfen und Auffüllen der Buchstaben
    ArrayList<String> reducedLetterSet = new ArrayList<>(fullLetterSet);

    //Überprüfen aller Positionen im Umkreis von 3 Einheiten
    for(int x = -FILL_CHECK_RADIUS; x <= FILL_CHECK_RADIUS; x++) {
        for(int y = -FILL_CHECK_RADIUS; y <= FILL_CHECK_RADIUS; y++) {
            int checkX = posX + x;
            int checkY = posY + y;

            //Entfernen der Buchstaben aus dem Umfeld aus der Liste der
verfügbaren Buchstaben
            if(checkX >= 0 && checkX < width && checkY >= 0 && checkY <
height && pattern[checkY][checkX] != null)
                reducedLetterSet.remove(pattern[checkY][checkX]);
```

```
        }
    }

    //Platzieren des neuen Buchstabens innerhalb des Feldes
    if(filledPoints.contains(new Point(posX, posY)) &&
reducedLetterSet.size() == 0)
        pattern[posY][posX] = getRandomChar();
}
```

Der gesamte Ablauf zur Überprüfung der platzierten Buchstaben wird insgesamt 5-mal wiederholt, um so viele Fehler wie möglich auszuschließen und das Feld so weit wie möglich zu optimieren.

Schwierigkeitsgrad: Schwer

Zum Generieren von Wortfeldern des höchsten Schwierigkeitsgrades wird die Klasse "HardPatternGenerator" verwendet, welche von der Klasse zur Generierung von mittleren Wortfeldern abgeleitet ist. Daher beeinhaltet der Schwierigkeitsgrad Schwer alle Abfolgen des Schwierigkeitsgrades Mittel mit einigen Ergänzungen.

Dabei beeinhaltet der Algorithmus Abwandlungen aller Positionierungsfunktionen der Wörter, sodass sich diese nun überschneiden können. Dabei werden Wörter nun mit Absicht überschneidend platziert, wenn kein freier Platz gefunden werden konnte. Dies geschieht mit der Funktion #placeWordCrossing, wobei nun auch der Parameter "crossingAllowed" zum Einsatz kommt, welcher für die niederen Schwierigkeitsgrade keine Rolle spielte.

Neue Funktion zur horizontalen Positionierung

```
if(!super.placeWordHorizontally(word, optionalX, optionalY,
crossingAllowed))
    //Überschneidung Platzierung, wenn keine Position gefunden werden kann
    return crossingAllowed && placeWordCrossing(word);
else return true;
```

Neue Funktion zur vertikalen Positionierung

```
if(!super.placeWordVertically(word, optionalX, optionalY,
crossingAllowed))
    //Überschneidende Platzierung, wenn keine Position gefunden werden
kann
    return crossingAllowed && placeWordCrossing(word);
else return true;
```

Neue Funktion zur diagonalen Positionierung

```
if(!super.placeWordDiagonally(word, optionalX, optionalY, orientation,
crossingAllowed))
    //Überschneidende Platzierung, wenn keine Position gefunden werden
kann
    return crossingAllowed && placeWordCrossing(word);
else return true;
```

Überschneidende Platzierung von Wörtern

Die Funktion #placeWordCrossing sucht ein passendes Wort, welches das gegebene Wort überschneidet. Anschließend wird zufällig eine der Ausrichtung für das zu positionierende Wort ausgewählt (Horizontal, Vertikal, Diagonal). Danach wird anhand des Schnittpunktes der beiden Wörter die Position des Wortes berechnet. Zum Schluss wird das Wort an der berechneten Stelle platziert. Falls dies nicht möglich ist, gibt auch diese Funktion false zurück.

```
protected boolean placeWordCrossing(String word) {
    //Überschneidung von Wörtern
    ArrayList<Map.Entry<String, WordPosition>> positionedWords = new
ArrayList<>(placedWords.entrySet().stream().toList());
    positionedWords.sort(Comparator.comparingInt(wordA ->
wordA.getKey().length()));
    Collections.reverse(positionedWords); //Höchste Chancen bei dem
längsten Wort

    boolean matchFound = false;
    for(Map.Entry<String, WordPosition> placedWord : positionedWords) {
        //Überprüfen der Platzierungsmögichkeiten
        boolean lettersInCommon = false;

        int crossingIndexX = 0;
        int crossingIndexY = 0;

        //Bestimmen des Schnittpunktes zwischen den beiden Wörtern
        for(int a = 0; a < placedWord.getKey().length(); a++) {
            for(int b = 0; b < word.length(); b++) {

if(String.valueOf(placedWord.getKey().charAt(a)).equals(String.valueOf(wor
d.charAt(b)))) {
                    lettersInCommon = true;
                    crossingIndexX = a;
                    crossingIndexY = b;
                    break;
                }
            }
        }

        //Überspringen des momentanen Wortes, wenn kein Schnittpunkt
gefunden werden konnte
        if(!lettersInCommon) continue;
```

```
        //Bestimmen der Orientierung des neuen Wortes
        WordPosition.Orientation newOrientation =
WordPosition.randomOrientation(placedWord.getValue().orientation());

        //Berechnen der Koorindaten des kreuzenden Wortes, welches
platziert werden soll
        int possiblePositionX =
placedWord.getValue().crossWordX(crossingIndexX, crossingIndexY,
newOrientation);
        int possiblePositionY =
placedWord.getValue().crossWordY(crossingIndexX, crossingIndexY,
newOrientation);

        //Überspringen des momentanen Wortes, wenn die Koorindaten
außerhalb des Feldes liegen
        if(possiblePositionX < 0 || possiblePositionX >=
pattern[0].length) continue;
        if(possiblePositionY < 0 || possiblePositionY >= pattern.length)
continue;

        //Überspringen des momentanen Wortes, wenn das Wort nicht mehr auf
das Feld passt
        if(newOrientation.equals(WordPosition.Orientation.Vertical) &&
possiblePositionY + word.length() > pattern.length) continue;
        if(newOrientation.equals(WordPosition.Orientation.Horizontal) &&
possiblePositionX + word.length() > pattern[possiblePositionY].length)
continue;

        if(newOrientation.equals(WordPosition.Orientation.DiagonalUp) &&
(possiblePositionX + word.length() > pattern[possiblePositionY].length ||
possiblePositionY - word.length() < 0)) continue;
        if(newOrientation.equals(WordPosition.Orientation.DiagonalDown) &&
(possiblePositionX + word.length() > pattern[possiblePositionY].length ||
possiblePositionY + word.length() > pattern.length)) continue;

        //Platzieren des kreuzenden Wortes
        boolean result = switch (newOrientation) {
            case Horizontal -> placeWordHorizontally(word,
possiblePositionX, possiblePositionY, false);
            case Vertical -> placeWordVertically(word, possiblePositionX,
possiblePositionY, false);
            case DiagonalUp -> placeWordDiagonally(word,
possiblePositionX, possiblePositionY, DiagonalDirection.Ascending, false);
            case DiagonalDown -> placeWordDiagonally(word,
possiblePositionX, possiblePositionY, DiagonalDirection.Descending,
false);
        };

        //Abbrechen der Schleife, wenn das kreuzende Wort erfolgreich
platziert werden konnte
        if(result) {
            matchFound = true;
            break;
        }
```

```
        //Andernfalls wird die Schleife fortgesetzt, bis ein Schnittpunkt
gefunden wurde oder alle Möglichkeiten ausprobiert wurden
    }

    return matchFound;
}
```

Auffüllen von leeren Stellen

Dieser Schwierigkeitsgrad füllt die leeren Stellen ebenfalls auf, hierbei erweitert dieser das Vorgehen des vorherigen Schwierigkeitsgrades. Die entstehenden Lücken werden, wenn sie größer als eine Einheit groß sind, mit Fragmenten der Wörter aus der Wortliste aufgefüllt. Die verbleibenden Lücken (kleiner/gleich eine Einheit) werden widerrum mit zufälligen Buchstaben aufgefüllt, sodass die zu findenen Wörter nicht zu offensichtlich sind.
Anschließend wird der identische Algorithmus wie bei der Schwierigkeit Mittel verwendet, um die Auffüllung zu überprüfen.

Überarbeitete Funktion #fillEmptySpaces

```
//Sammeln aller Buchstaben, welche in den Wörtern aus der Wortliste
enthalten sind
ArrayList<String> letterSet = new ArrayList<>();
for(String word : words) {
    for(int i = 0; i < word.length(); i++) {
        if(!letterSet.contains(String.valueOf(word.charAt(i))))
            letterSet.add(String.valueOf(word.charAt(i)));
    }
}

//Auffüllen von bestimmten Teilen von Reihen mit Fragmenten der Wörter aus
der Wortliste
for (int y = 0; y < pattern.length; y++) {
    Map<Integer, Integer> emptySpaces = new HashMap<>();

    boolean inPart = false;
    int lastStart = 0;

    //Ermitteln von Leerstellen innerhalb einer Zeile
    for (int letter = 0; letter < pattern[y].length; letter++) {
        if (pattern[y][letter].equals(PatternGenerator.CHARACTER_EMPTY)) {
            if (!inPart) {
                inPart = true;
                lastStart = letter;
            }

            //Hinzufügen der Stelle zu der Liste
            emptySpaces.put(lastStart, letter);
        } else {
            //Leerstelle ist beendet
```

```
                inPart = false;
            }
        }

        //Auffüllen der größeren Leerstellen mit den Fragmenten der Wortliste
        for (Map.Entry<Integer, Integer> entry : emptySpaces.entrySet()) {
            Integer key = entry.getKey(); //Startposition der Leerstelle
            Integer value = entry.getValue(); //Endposition der Leerstelle

            //Leerstelle wird nur aufgefüllt, wenn diese größer, als zwei
Einheiten ist
            if (value - key > 1) {
                //Generieren eines Fragments der Wortliste
                String wordInstance =
words.get(instanceRandom.nextInt(words.size()));

                //Abbrechen der Schleife, wenn das Fragment zu kurz ist
                if(wordInstance.length() - (value - key) <= 0) continue;

                String wordFragment =
wordInstance.substring(instanceRandom.nextInt(wordInstance.length() -
(value - key)));

                //Hinzufügen des Fragmentes in das Wortfeld
                for (int i = key; i <= value; i++) {
                    pattern[y][i] = String.valueOf(wordFragment.charAt(i -
key));
                    filledPoints.add(new Point(i, y));
                }
            }
        }
    }

    //Auffüllen der übrigen Buchstaben mit zufälligen Buchstaben
    for(int y = 0; y < pattern.length; y++) {
        for(int x = 0; x < pattern[y].length; x++) {
            if(pattern[y][x].equals(PatternGenerator.CHARACTER_EMPTY)) {
                pattern[y][x] = getRandomChar();
                filledPoints.add(new Point(x, y));
            }
        }
    }

    //Überprüfung der platzierten Buchstaben
    checkFilledSpaces(letterSet);
```

Beispiele

Beispiel (worte0.txt)

Schwierigkeit: LEICHT

V	**O**	**R**	P	P
G	O	L	U	**E**
T	**O**	**R**	**F**	**V**
E	H	V	C	**A**
R	**A**	**D**	P	F

Schwierigkeit: MITTEL

T	**O**	**R**	**F**	F
A	**E**	**V**	**A**	**R**
D	**V**	**O**	**R**	**A**
Y	H	P	B	**D**
A	M	L	O	M

Schwierigkeit: SCHWER

T	**O**	**R**	**F**	D
R	**V**	A	Y	R
A	P	**O**	M	K
D	I	R	**R**	E
G	**E**	**V**	**A**	K