

Aufgabe 1: Hopsitexte

Team-ID: 00178

Team-Name: Team-Name

Bearbeiter/-innen dieser Aufgabe:
Matthew Greiner

November 17, 2024

Contents

| | | |
|----------|---|----------|
| 1 | Lösungsidee / Ansatz | 1 |
| 1.1 | Anforderungen an das Programm | 1 |
| 2 | Umsetzung | 2 |
| 3 | Beispiele | 3 |
| 3.1 | Beispiele BwInf-Aufgabenangaben | 3 |
| 3.2 | Eigene Beispiele / Edgecases | 3 |
| 4 | Quellcode | 5 |

1 Lösungsidee / Ansatz

1.1 Anforderungen an das Programm

In dieser Aufgabe geht es darum, ein Programm zu erstellen, das Zara hilft, deutsche Hopsitexte zu erstellen. Um ein Programm zu gestalten, damit Zara besonders gut damit arbeiten kann, müssen meiner Meinung nach, folgende Kriterien für das Programm erfüllt sein:

1. Das Programm muss ein graphisches Nutzerinterface mit einem Textfeld haben, in dem Zara ihren Text tippen kann.
2. Die Stellen, an denen die Sprünge stattgefunden haben, müssen wie in den Beispieltexten, rot für den ersten Spieler und blau für den zweiten Spieler, gefärbt sein.
3. Da es ein deutscher Text sein soll, müssen Umlaute wie "ä, ö, ü" und "ß" auch unterstützt werden.
4. Wenn der Text kein Hopsitext ist, muss die Stelle, an dem sich die Spieler treffen, farblich markiert sein, damit Zara weiß, dass sie Änderungen vornehmen muss.
5. Es muss anzeigen, ob der eingegebene Text ein "Hopsitext" ist oder nicht.
6. Die Ausgabe des Programmes und die farbliche Markierungen des Eingabetextes müssen dynamisch und automatisch mit dem Tippen von Zara geschehen und Zara kann an beliebigen Stellen im Eingabetext Änderungen vornehmen (Quality of life).

Grobe Struktur

Auf der linken Seite des Programmes soll Zara ihren Text tippen können und auf der rechten Seite soll dieser Text an den Sprüngen der Spieler farblich markiert, - wie im Beispielttext, angezeigt werden. So sieht Zara beim tippen, ob ihre aktuelle Eingabe ein Hopsitext ist oder nicht. Zusätzlich soll oberhalb des Textfeldes deutlich angezeigt werden, ob es sich um einen Hopsitext handelt. Wenn diese Kriterien erfüllt sind, hat Zara ein Programm, in der sie einen Text schreiben kann, das ihr beim tippen anzeigt, ob ihre aktuelle Eingabe ein Hopsitext ist oder nicht. Falls es kein Hopsitext ist, sieht Zara das sofort und kann ihren Text bzw. einfach die letzten Worte mit dem Sprung ändern.

2 Umsetzung

Um dieses Programm zu erstellen, habe ich JavaFX als GUI Library verwendet, welches mir grundlegende Bausteine für das Interface zur Verfügung stellt. Zur Implementierung werden die in (1.1) angesprochenen Anforderungen einzeln umgesetzt:

1. Zu Beginn erstelle ich mit JavaFX Modulen ein Textfeld ("TextArea") auf der linken Seite des Programms, eine formatierte Textanzeige ("Textflow") auf der rechten Seite, in der die Hopsisprünge markiert werden und einen Statustext ("Text") mit dem Ergebnis ganz oben. Das Textfeld bekommt einen Listener, der jedes mal gerufen wird, wenn sich der Text im Textfeld ändert. So wird jedes mal das Ergebnis und der formatierte Text auf der rechten Seite geupdated, sobald sich die Texteingabe ändert.
2. Eine Methode wird erstellt, die die Texthops-Sprünge ab einer Startposition simuliert. Diese Methode wird zweimal aufgerufen, jeweils einmal für die Roten und die Blauen Sprünge. Die Methode funktioniert wie folgt:
 1. Der Eingabetext als String und die Farbe, mit der gesprungen werden soll, wird übergeben.
 2. Die ersten Zeichen, die keine Buchstaben sind, werden bei der Simulation der Texthops-Sprünge übersprungen und bei dem ersten Zeichen, das ein Buchstabe ist, wird begonnen.
 3. Dieses Zeichen wird mit der übergebenen Farbe gefärbt und zu einer Collection aus Zeichen (TextFlow) hinzugefügt. Die Position des Buchstabens im Alphabet wird sich gemerkt; z.B. als n .
 4. Dann werden die nächsten n Buchstaben übersprungen. Alle Zeichen, die keine Buchstaben sind, werden ignoriert. Alle Zeichen, inklusive der Nicht-Buchstaben werden ungefärbt zu dem TextFlow hinzugefügt.
 5. Anschließend wird der Buchstabe bei dem wir nach den Sprüngen angekommen sind, mit der übergebenen Farbe gefärbt und zum Textflow hinzugefügt. Diese Schleife (3. - 5.) wird wiederholt, bis der Inputtext vollständig geparkt wurde.
 6. Das resultierende TextFlow enthält nun den gesamten Inputtext, wobei die Hopsisprünge farblich markiert sind. Dieses Textflow wird zurückgegeben.
3. Um die Sprungweite eines Buchstabens gleich weit wie ihre Position im Alphabet zu machen und Umlaute auch zu beachten, muss eine Methode erstellt werden, die die zugehörige Sprungweite von einem übergebenen Zeichen zurück gibt:

Wenn das Zeichen ein deutsches Sonderzeichen ist, wird eine festgelegte Position zurückgegeben. Wenn es sich um einen normalen großen Buchstaben handelt, kann man um die Position im Alphabet zu bekommen, den ASCII-Wert von dem Zeichen mit dem ASCII-Wert von "A" subtrahieren und 1 addieren. Somit ist A=1, B=2, etc. Kleinbuchstaben funktionieren analog, nur wird hier der ASCII-Wert von "a" abgezogen. Alle anderen Zeichen geben -1 zurück.
4. Um die Stelle zu markieren, an der sich beide Spieler treffen (z.B. in **lila**), wird die Methode aus (2.) verwendet. Sie wird aufgerufen, um die Sprünge von beiden Spielern zu berechnen und in einem resultierendem TextFlow farblich zu markieren. Nun müssen diese beiden TextFlows verglichen werden. Wenn an einer bestimmten Stelle beide Buchstaben in beiden TextFlows farblich markiert sind, bedeutet das, dass sich die Spieler an dieser Stelle treffen. Daher muss für dieses Teilproblem eine Methode erstellt werden, die diese beiden TextFlows vereinigt und wenn sich die Spieler treffen, farblich markiert (lila), und dies als Ergebnis dem Nutzer anzeigt.

Dies geht indem man über Zeichen in den beiden TextFlows iteriert und überprüft, ob ein Buchstabe in beiden TextFlows farblich markiert ist. Wenn ja, ist an diesem Buchstaben eine Kollision von den Sprüngen, und wird im Ergebnistextflow **lila** markiert. Wenn keine Kollision an einem Buchstaben vorherrscht, wird der farblich markierte Buchstaben aus einem der beiden TextFlows übernommen. Wenn an einer Stelle kein Buchstabe farblich markiert ist, wird das Zeichen ungefärbt zum Resultattextflow hinzugefügt.

5. Wenn die Methode aus (4.) an beliebiger Stelle lila anzeigt, ist der übergebenen Text kein Hopsitext mehr. Dies wird über dem Nutzereingabefeld bekenntlich gemacht.
6. Das von oben genannte wird jedes mal ausgeführt, wenn sich das Eingabefeld ändert.

3 Beispiele

3.1 Beispiele BwInf-Aufgabenangaben

Hier ist das Programm mit den Angabetexten der Junioraufgabe 2 und der Aufgabe 1 als Eingabetext.

Angabe aus Junioraufgabe 2

| Dieser Text ist ein Hopsitext! | |
|--|--|
| <p>Bela und Amira ist im Deutschunterricht oft langweilig. Daher haben sie sich ein neues Spiel ausgedacht: Texthopsen. Sie suchen sich einen Text im Schulbuch aus und hopsen darin um die Wette. Das geht so: Jeder Buchstabe hat eine Sprungweite zugeordnet (siehe Tabelle). Bela fängt beim ersten, Amira beim zweiten Buchstaben des Textes an. Von dieser Startposition springen sie jeweils so viele Positionen weiter, wie die Sprungweite des Buchstabens vorgibt. Alles was kein Buchstabe ist, überspringen sie dabei einfach. An der neuen Position lesen sie dann jeweils den nächsten Buchstaben mit einer neuen Sprungweite. Das Hopsen wiederholen sie abwechselnd so lange, bis sie aus dem Text herauspringen. Wer dies als Erstes schafft, hat gewonnen.</p> | <p>Bela und Amira ist im Deutschunterricht oft langweilig. Daher haben sie sich ein neues Spiel ausgedacht: Texthopsen. Sie suchen sich einen Text im Schulbuch aus und hopsen darin um die Wette. Das geht so: Jeder Buchstabe hat eine Sprungweite zugeordnet (siehe Tabelle). Bela fängt beim ersten, Amira beim zweiten Buchstaben des Textes an. Von dieser Startposition springen sie jeweils so viele Positionen weiter, wie die Sprungweite des Buchstabens vorgibt. Alles was kein Buchstabe ist, überspringen sie dabei einfach. An der neuen Position lesen sie dann jeweils den nächsten Buchstaben mit einer neuen Sprungweite. Das Hopsen wiederholen sie abwechselnd so lange, bis sie aus dem Text herauspringen. Wer dies als Erstes schafft, hat gewonnen.</p> |

Angabe aus Aufgabe 1

| Dieser Text ist ein Hopsitext! | |
|--|--|
| <p>Zara hat das Texthopsen schon programmiert. Dabei ist ihr aufgefallen, dass die beiden Spieler erstaunlich häufig von derselben Stelle aus dem Text herauspringen. Sie versucht einen möglichst langen Text zu schreiben, der diese Eigenschaft nicht hat. Das heißt, die Sprungsequenzen, die am ersten und zweiten Buchstaben des Textes beginnen, sollen nicht zur selben Endposition führen. So einen Text nennt sie einen 'Hopsitext'. Dieser Aufgabentext ist zum Beispiel ein solcher Hopsitext.</p> | <p>Zara hat das Texthopsen schon programmiert. Dabei ist ihr aufgefallen, dass die beiden Spieler erstaunlich häufig von derselben Stelle aus dem Text herauspringen. Sie versucht einen möglichst langen Text zu schreiben, der diese Eigenschaft nicht hat. Das heißt, die Sprungsequenzen, die am ersten und zweiten Buchstaben des Textes beginnen, sollen nicht zur selben Endposition führen. So einen Text nennt sie einen 'Hopsitext'. Dieser Aufgabentext ist zum Beispiel ein solcher Hopsitext.</p> |

3.2 Eigene Beispiele / Edgecases

Kurze Texte

| Zu kurz um ein Hopsitext zu sein! | |
|-----------------------------------|---|
| A | A |

| | |
|--|----|
| Dieser Text ist leider noch kein Hopsitext :(| |
| AB | AB |

| | |
|---------------------------------------|----|
| Dieser Text ist ein Hopsitext! | |
| Xy | Xy |

Random Input

| | |
|---------------------------------------|----------------|
| Dieser Text ist ein Hopsitext! | |
| 123123123bgsef | 123123123bgsef |

| | |
|--|------------------------------------|
| Dieser Text ist leider noch kein Hopsitext :(| |
| 231@lofawfjöli3298jf2-2jff'aaaaaaa | 231@lofawfjöli3298jf2-2jff'aaaaaaa |

| | |
|--|---|
| Dieser Text ist leider noch kein Hopsitext :(| |
| föaoi98 iua90l 9<)ß#+hfH Nfasdfohnahsfd bbbhw23 | föaoi98 iua90l 9<)ß#+hfH Nfasdfohnahsfd bbbhw23 |

Lange Texte

| | |
|---|---|
| Dieser Text ist leider noch kein Hopsitext :(| |
| <p>This page attempts to draw a comparison between Arch Linux and other popular GNU/Linux distributions and UNIX-like operating systems. The summaries that follow are brief descriptions that may help a person decide if Arch Linux will suit their needs. Although reviews and descriptions can be useful, first-hand experience is invariably the best way to compare distributions.</p> <p>For a more complete comparison, see Wikipedia:Comparison of operating systems and Wikipedia:Comparison of Linux distributions.</p> <p>In all of the following, only Arch Linux is compared with other distributions. Community ports that support architectures other than x86_64 can be found listed among the Arch-based distributions.</p> | <p>This page attempts to draw a comparison between Arch Linux and other popular GNU/Linux distributions and UNIX-like operating systems. The summaries that follow are brief descriptions that may help a person decide if Arch Linux will suit their needs. Although reviews and descriptions can be useful, first-hand experience is invariably the best way to compare distributions.</p> <p>For a more complete comparison, see Wikipedia:Comparison of operating systems and Wikipedia:Comparison of Linux distributions.</p> <p>In all of the following, only Arch Linux is compared with other distributions. Community ports that support architectures other than x86_64 can be found listed among the Arch-based distributions.</p> |

| | |
|---|---|
| Dieser Text ist leider noch kein Hopsitext :(| |
| <p>Einsatzgebiete der Strömungsmesstechnik sind die Forschung und Entwicklung, wo es gilt, Strömungsvorgänge zu untersuchen oder zu optimieren. Die Strömungsmesstechnik ist aber auch eine wesentliche Komponente für die Prozessführung in industriellen Anlagen der Chemie- oder Energiewirtschaft. Verlässliche Informationen über Eigenschaften turbulenter Strömungen können nur durch die Strömungsmesstechnik erhalten werden.</p> <p>Von besonderem Interesse sind die grundlegenden Größen Geschwindigkeit, Druck und Temperatur. Messungen können mit in die Strömung eingebrachten Messsonden aufgenommen werden. Staudrucksonden messen im Fluid den Gesamtdruck, aus dem indirekt auf die Geschwindigkeit rückgeschlossen werden kann. Die Thermische Anemometrie stellt eine weitere indirekte Geschwindigkeitsmessmethode dar. Der Nachteil an diesen indirekten Messmethoden ist, dass das Messsignal nicht allein von der Geschwindigkeit, sondern auch von anderen Zustandsgrößen abhängt, die also bekannt sein müssen.</p> <p>Verfahren wie die Particle Image Velocimetry und Laser-Doppler-Anemometrie (siehe Bild) gestattet die direkte und lokale Geschwindigkeitsmessung ohne Sonden. Insbesondere in der Aeroakustik interessieren nicht die Durchschnittswerte, sondern die Schwankungswerte des Drucks, insbesondere die spektrale Leistungsdichte, die durch weitere Signalverarbeitung erhalten wird.</p> <p>Numerische Strömungsmechanik → Hauptartikel: Numerische Strömungsmechanik Visualisierung einer CFD-Simulation der Boeing X-43 bei Mach 7</p> <p>Die Leistungsfähigkeit der Computer gestattet es, die Grundgleichungen in wirklichkeitsnahen Randwertproblemen zu lösen und die erzielten, realitätsnahen Resultate haben dazu geführt, dass die numerische Strömungsmechanik ein wichtiges Werkzeug in der Strömungsmechanik wurde. In der aerodynamischen Auslegung und Optimierung haben sich die numerischen Methoden etabliert, denn sie gestatten einen detaillierten Einblick in die Strömungsvorgänge, siehe Bild, und Untersuchung von Modellvarianten.</p> <p>Die aus der angewandten Mathematik bekannten Methoden zur Lösung gewöhnlicher Differentialgleichungen versehen vorbereitend das Strömungsgebiet mit einem „numerischen Gitter“. Potentialströmungen verlangen den geringsten Aufwand und auch die Euler-Gleichungen erlauben relativ grobe Gitter. Die bei Anwendung der Navier-Stokes-Gleichungen bedeutsamen Grenzschichten und Turbulenzen erfordern eine hohe räumliche Auflösung des Gitters. In drei Dimensionen steigt die Anzahl der Freiheitsgrade mit der dritten Potenz der Abmessung, so dass auch noch im 21. Jahrhundert der Aufwand für die Direkte Numerische Simulation bei Anwendungen in der Fahrzeugentwicklung nicht vertretbar ist. Daher kommen Turbulenzmodelle zum Einsatz, die die notwendige Auflösung zu reduzieren gestatten. Trotzdem sind oftmals Systeme mit mehreren zehnmillionen Gleichungen für mehrere tausend Iterations- oder Zeitschritte zu lösen, was eines Rechnerverbunds und effizienter Programmierung bedarf.</p> | <p>Einsatzgebiete der Strömungsmesstechnik sind die Forschung und Entwicklung, wo es gilt, Strömungsvorgänge zu untersuchen oder zu optimieren. Die Strömungsmesstechnik ist aber auch eine wesentliche Komponente für die Prozessführung in industriellen Anlagen der Chemie- oder Energiewirtschaft. Verlässliche Informationen über Eigenschaften turbulenter Strömungen können nur durch die Strömungsmesstechnik erhalten werden.</p> <p>Von besonderem Interesse sind die grundlegenden Größen Geschwindigkeit, Druck und Temperatur. Messungen können mit in die Strömung eingebrachten Messsonden aufgenommen werden. Staudrucksonden messen im Fluid den Gesamtdruck, aus dem indirekt auf die Geschwindigkeit rückgeschlossen werden kann. Die Thermische Anemometrie stellt eine weitere indirekte Geschwindigkeitsmessmethode dar. Der Nachteil an diesen indirekten Messmethoden ist, dass das Messsignal nicht allein von der Geschwindigkeit, sondern auch von anderen Zustandsgrößen abhängt, die also bekannt sein müssen.</p> <p>Verfahren wie die Particle Image Velocimetry und Laser-Doppler-Anemometrie (siehe Bild) gestattet die direkte und lokale Geschwindigkeitsmessung ohne Sonden. Insbesondere in der Aeroakustik interessieren nicht die Durchschnittswerte, sondern die Schwankungswerte des Drucks, insbesondere die spektrale Leistungsdichte, die durch weitere Signalverarbeitung erhalten wird.</p> <p>Numerische Strömungsmechanik → Hauptartikel: Numerische Strömungsmechanik Visualisierung einer CFD-Simulation der Boeing X-43 bei Mach 7</p> <p>Die Leistungsfähigkeit der Computer gestattet es, die Grundgleichungen in wirklichkeitsnahen Randwertproblemen zu lösen und die erzielten, realitätsnahen Resultate haben dazu geführt, dass die numerische Strömungsmechanik ein wichtiges Werkzeug in der Strömungsmechanik wurde. In der aerodynamischen Auslegung und Optimierung haben sich die numerischen Methoden etabliert, denn sie gestatten einen detaillierten Einblick in die Strömungsvorgänge, siehe Bild, und Untersuchung von Modellvarianten.</p> <p>Die aus der angewandten Mathematik bekannten Methoden zur Lösung gewöhnlicher Differentialgleichungen versehen vorbereitend das Strömungsgebiet mit einem „numerischen Gitter“. Potentialströmungen verlangen den geringsten Aufwand und auch die Euler-Gleichungen erlauben relativ grobe Gitter. Die bei Anwendung der Navier-Stokes-Gleichungen bedeutsamen Grenzschichten und Turbulenzen erfordern eine hohe räumliche Auflösung des Gitters. In drei Dimensionen steigt die Anzahl der Freiheitsgrade mit der dritten Potenz der Abmessung, so dass auch noch im 21. Jahrhundert der Aufwand für die Direkte Numerische Simulation bei Anwendungen in der Fahrzeugentwicklung nicht vertretbar ist. Daher kommen Turbulenzmodelle zum Einsatz, die die notwendige Auflösung zu reduzieren gestatten. Trotzdem sind oftmals Systeme mit mehreren zehnmillionen Gleichungen für mehrere tausend Iterations- oder Zeitschritte zu lösen, was eines Rechnerverbunds und effizienter Programmierung bedarf.</p> |

4 Quellcode

Simuliert Hopsitext Sprünge - Entspricht der Methode aus (2.2)

```

1  /**
2   * Simulates Hopsitext-jumps
3   *
4   * @param input - Inputtext as String
5   * @param startIndex - Starting Position of Player
6   * @param color - Color of Player
7   * @return TextFlow
8   */
9  private TextFlow jump(String input, int startIndex, Color color) {
10     // Textflow is a collection of textnodes
11     TextFlow tf = new TextFlow();
12     int nextIndex = 0;
13     char currentChar = ' ';
14     // Textnode is a single character, because characters need to be colored
15     // individually
16     Text textNode = new Text();
17
18     // Skip over first non-alphabetic characters
19     int i = 0;
20     for (; i < input.length(); i++) {
21         if (getAlphabetPosition(input.charAt(nextIndex)) == -1) {
22             textNode = new Text(String.valueOf(input.charAt(nextIndex)));
23             tf.getChildren().add(textNode);
24             nextIndex++;
25         } else {
26             break;
27         }
28     }
29     nextIndex += startIndex;
30
31     if (nextIndex == (1 + i) && !input.isEmpty()) {
32         textNode = new Text(String.valueOf(input.charAt(0)));
33         tf.getChildren().add(textNode);
34     }
35
36     // Main loop
37     while (nextIndex < input.length()) {
38         currentChar = input.charAt(nextIndex);
39         textNode = new Text(String.valueOf(currentChar));
40
41         // Character that needs to be colored, because jump starts on this char
42         if (getAlphabetPosition(currentChar) != -1) {
43             // Display character in specified color
44             textNode.setFill(color);
45             tf.getChildren().add(textNode);
46
47             int alphabetPosition = getAlphabetPosition(currentChar);
48             int count = 0;
49
50             // Skip non-alphabetic characters and move forward by the alphabet position
51             while (count < alphabetPosition) {
52                 nextIndex++;
53                 if (nextIndex >= input.length()) {
54                     break;
55                 }
56                 // If Character is an alphabetic character
57                 if (getAlphabetPosition(input.charAt(nextIndex)) != -1) {
58                     count++;
59                     // Add uncolored character
60                     if (count != alphabetPosition) {
61                         textNode = new Text(String.valueOf(input.charAt(nextIndex)));
62                         tf.getChildren().add(textNode);
63                     }
64                     // If Character is non alphabetic -> skip it and don't increment
65                     loop counter
66                 } else {
67                     // Add uncolored character
68                     textNode = new Text(String.valueOf(input.charAt(nextIndex)));

```

```

        tf.getChildren().add(textNode);
    }
} else {
    textNode = new Text(String.valueOf(input.charAt(nextIndex)));
    // textNode.setFill(color);
    System.out.println("If this gets executed, then there is something wrong");
    tf.getChildren().add(textNode);
    nextIndex++;
}
}
return tf;
}

```

Vereinigt zwei TextFlows - Entspricht der Methode aus (2.4)

```

/**
 * Merges two TextFlows into one with merged colors of two players. Jump
 * collisions are colored purple
 *
 * @param tf1 - first player jumps
 * @param tf2 - second player jumps
 * @return TextFlow
 */
private TextFlow mergeTextFlows(TextFlow tf1, TextFlow tf2) {
    TextFlow result = new TextFlow();
    int count = 0;

    // Loops over every character
    for (int i = 0; i < tf1.getChildren().size(); i++) {
        Text text1 = (Text) tf1.getChildren().get(i);
        Text text2 = (Text) tf2.getChildren().get(i);

        Color color1 = (Color) text1.getFill();
        Color color2 = (Color) text2.getFill();

        Text mergedText = new Text(text1.getText());

        // Result Color of current character
        Color finalColor = determineColor(color1, color2);
        if (finalColor.equals(Color.PURPLE)) {
            output.setText("Dieser Text ist leider noch kein Hopsitext.");
            count = 1;
        } else {
            if (count == 1) {
                output.setText("Dieser Text ist leider noch kein Hopsitext.");
            } else {
                output.setText("Dieser Text ist ein Hopsitext!");
            }
        }
        mergedText.setFill(finalColor);

        // Display result to user
        result.getChildren().add(mergedText);
    }
    return result;
}

```

Sprungweite aus Buchstaben ermitteln - Entspricht der Methode aus (2.3)

```

1  /**
2  * Return the position in the alphabet of the character. Upper and lowercase
3  * characters are treated the same. German "Umlaute" also return a valid
4  * position. Everything else returns -1
5  *
6  * @param c - Character to check
7  * @return int - Position in the alphabet
8  */
9  private int getAlphabetPosition(char c) {
10     switch (c) {
11         case 'ä':
12         case 'Ä':
13             return 27;
14         case 'ö':
15         case 'Ö':
16             return 28;
17         case 'ü':
18         case 'Ü':
19             return 29;
20         case 'ß':
21             return 30;
22         default:
23             break;
24     }
25
26     if (Character.isUpperCase(c) && upperCase.contains(c)) {
27         return c - 'A' + 1; // A=1, B=2, ..., Z=26
28     } else if (Character.isLowerCase(c) && lowerCase.contains(c)) {
29         return c - 'a' + 1; // a=1, b=2, ..., z=26
30     } else {
31         return -1;
32     }
33 }

```

Nutzung der oben gezeigten Methoden

```

1  /**
2  * Receives Text input from the use and displays the processed version with
3  * correct colors for the jumps
4  *
5  * @param input - Text from the input field
6  */
7  private void processText(String input) {
8     textFlow.getChildren().clear();
9
10     if (input.isEmpty()) {
11         output.setText("Ist dein Text ein Hopsitext? Tippe in das Textfeld.");
12     } else if (input.length() == 1) {
13         textFlow.getChildren().add(new Text(input));
14         output.setText("Zu kurz um ein Hopsitext zu sein!");
15     } else {
16         textFlow.getChildren()
17             .addAll(mergeTextFlows(jump(input, 0, Color.RED), jump(input, 1,
18                 Color.BLUE)).getChildren());
19     }
20 }

```