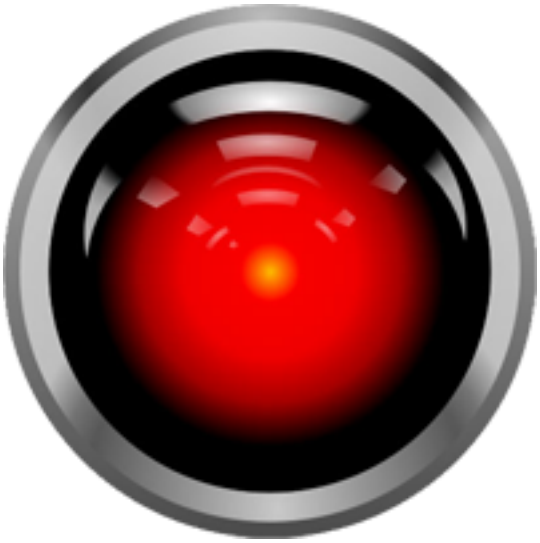
Dokumentation zur Aufgabe 3 - VortänzerChallenge

Dominic S. Meiser

Die zugehörige KI ist power2 v. 27 von meiserdo



Inhaltsverzeichnis

1 Lösungsidee

Im Prinzip geht es bei dieser Aufgabe darum, einen vorgegebenen String (= Vortanz) in möglichst wenig Zeichen umzuwandeln (zu "escapen"). Dieser Vortanz ist leider in unescapter Form übergeben; zusätzlich werden Strings, die länger als 255 Zeichen sind, abgeschnitten. Also ist in erster Linie die `escape`-Methode wichtig. Diese Methode alleine escapt einen String aber so, wie er dasteht, aus dem eigentlichen Vortanz `"FF489F..r."`, der übrigens auch in meiner Vortänzesammlung enthalten ist (siehe unten), wird zunächst der escapte Tanz `"89F..2FFr57FF...FFr48F..FF"`. Um aus diesem richtig escapten String einen kürzeren, escapten String zu machen, der (natürlich unter der Vorraussetzung, dass der unescapte Vortanz länger als 255 Zeichen war, was in diesem Fall der Fall ist) unescaped länger als 255 Zeichen ist, muss es noch eine weitere Funktion geben. Ich habe diese Funktion `compleate` genannt.

Das Ergebnis nach der `compleate`-Methode des obigen Beispiels lautet `"989F..2FFr..."`. Schon daran, dass ein Punkt zu viel enthalten ist, kann man erkennen, dass dies nicht sehr zielführend war. Deshalb wird anschließend der String in 2 Hälften aufgeteilt und anschließend jede Hälfte einzeln durch die `escape`-Methode und evtl. auch durch die `compleate`-Methode geführt. Dadurch würde im obigen Beispiel natürlich zuerst der normale String `""+"FF489F..r."` entstehen. Anschließend würde dies sich aber in `"F"+"F489F..r."`, `"FF"+"489F..r."` usw. ändern. Letzte Lösung ist am interessantesten, da in dem Fall der String sich wirklich aus beiden Hälften zusammensetzt. Den ersten Teil braucht mein Programm weder zu escapen noch zu compleaten, es bleibt also `"FF"` stehen - wie im Vortanz auch. Den 2. Teil des Vortanzes, also `"489F..r."`, würde mein Programm als `"389F..r.49F.."` escapen, wobei ich den String hier nach 255 Zeichen abgeschnitten habe, sodass ich die 2 F am Anfang ignoriert habe. Dies würde mein Programm selber natürlich nicht machen. Die `compleate`-Methode macht nun aus `"389F..r.49F.."` `"489F..r."`. Wenn man diese beiden Teile des Strings wieder zusammensetzt, erhält man nun `"FF489F..r."` - genau dasselbe wie der Vortanz.

Diese Funktion muss man aber für Vortänze wie etwa `"4rFr3rF.B."` noch etwas erweitern. Hierbei würde mein Programm bis jetzt `"42rFr.FrFB."` rauskriegen. Dies stimmt natürlich, ist aber leider länger. Deshalb muss auch noch geschaut werden, ob der String im großen und ganzen nur eine Schleife enthält. In diesem Fall wäre dies durch `"4rFrrFrFrFB."` gegeben. In dieser Schleife muss mein Programm dann auch nach 2 Teilen schauen, in diesem Fall `"rFr+3rF.B"`. Damit hat mein Programm auch diesen String so kurz wie möglich escapt.

1.1 Die `escape`-Methode

Ich habe mir überlegt, dass es am sinvollsten ist, die `escape`-Methode rekursiv zu machen. Ich schaue zuerst im übergebenen String nach `Pattern`, d.h. kleinen, öfters vorkommenden Strings, die ein Substring von 0 bis i sind. i wird in einer for-Schleife von anfänglich 1 auf maximal str.length()/2 gesetzt. Der Pattern wird anschließend weiter im String gesucht,

d.h. es wird geschaut, wie oft das Pattern hintereinander vorkommt. Diesen Wert sollte man in einer Pattern-Klasse (in meinem Fall `DancePattern`) speichern. Anschließend wird überprüft, ob es überhaupt sinvoll ist, das Pattern näher zu betrachten. Erste Möglichkeit, dass Pattern zu überspringen, ist, wenn das Auftreten im String, welches ich `appearment` genannt habe, gleich 0 ist. Wenn es nie mehr im String vorkommt, brauche ich nicht weiterzusuchen. Als zweites überprüfe ich, ob das Pattern von einem anderen ersetzt wird, also braucht man, wenn man das Pattern `"F"` schon gefunden hat, kein Pattern `"FF"` usw. mehr. Hiervon nehme ich dann das längste Pattern und escape sowohl das Pattern als auch den restlichen Teil des Strings, der nicht durch das gefundene Pattern abgedeckt wird.

Wenn hierbei kein Pattern gefunden wurde bzw nur Pattern, die es sich nicht lohnt zu escapen, wie z.B. `"FF"` ist besser als `"2F."`, wird auch rekursiv weiter-escaped. Ich schneide nun das erste Zeichen ab und escape den restlichen Teil des Strings. Dies kann bei einem String wie `"F9B."` hilfreich sein, da weder `"F"` noch `"F*B."` ein sinvoller String ist. Wenn man das `"F"` am Anfang wegschneidet, erhält man `"9B."`, was meine Methode escapen kann.

Wenn man das Ganze so laufen lässt, bekommt man anstelle von `"99F.."` ein Ergebnis wie `"81F."`. Das Programm hat hier natürlich erkannt, dass es sich um 81xF handelt, hat die 81 aber so stehen gelassen. Hier muss man dann aufpassen, dass eine Zahl wie 81 in Zahlen <=9 aufgeteilt wird, die als Produkt 81 ergeben. Hierzu gibt es eine weitere, rekursive Funktion, die alle Zahlen von 2-9 ausprobiert und eine Kombination zurückgibt, die als Produkt die gewünschte Zahl ergeben. Natürlich muss man auch hierbei aufpassen, dass man bei 82xF nicht irgendeine Kombination findet sondern `"99F..F"` zurückgibt.

1.2 Die `compleate`-Methode

Meine Idee bei der `compleate`-Methode ist es, die ersten n Zeichen des escapten Vortanz zu nehmen und erstmal abzuschneiden. Anschließend muss dies verlängert werden, d.h. ich muss überprüfen ob der Tanz sowieso schon in eine Schleife eingepackt ist, die einen kleineren Index als 9 hat, oder ob ich eine weitere Schleife hinzufügen muss. Daraufhin müssen noch Sachen wie `"9."` entfernt werden. Wenn das Ergebnis unescapt nicht dem Ausgang entspricht, wird n bis maximal 13 inkrementiert. Anschließend wird das kürzeste Ergebnis zurückgegeben.

1.3 Bewertung einer Lösung

Wenn eine oder mehrere Lösungen erstellt wurden, müssen diese bewertet werden. Aufgrund der Aufgabenstellung bietet es sich hierbei an, die Lösung mit den Strafpunkten zu bewerten. Dies ist relativ einfach. Man benötigt nur die aktuelle Bewegungsrichtung und die x- und y-Entfernung zum Startpunkt. Außerdem benötigt man sowohl den escapten als auch den unescapten String, um sowohl die Länge des Strings, die ja ebenfalls für die Strafpunkte entscheidend ist, und den Inhalt des Strings, wobei es sich hier nicht anbietet, die Schleifen eines escapten Strings zu beachten. Deshalb nehme ich am Anfang erstmal

die Länge des escapten Strings multipliziert mit 3 als Strafpunkte. Anschließend gehe ich jedes Zeichen in den beiden Strings durch (also dem Vortanz und dem eventuellen Nachtanz) und bewege es, d.h. ich addiere oder subtrahiere eine Positionsänderung oder drehe es indem ich die Bewegungsrichtung verändere. Anschließend nehme ich den Unterschied zwischen Vortanz und Nachtanz und addiere diesen hinzu. Da man auch schräg laufen kann muss man hierbei das Maximum der Unterschiede der x- bzw. y-Position nehmen.

1.4 Vortänze

Weiterhin sind die Vortänze sehr wichtig. Hierzu habe ich 6 Muster, denen ein Vortanz von mir folgt, entwickelt:

- `999F.r.-r.`
- `9Fr.99BB..`
- `FF489F..r.`
- `9rFr7rF.B.`
- `9F9Fl7F...`
- `B5F5l7B...`

Ein F kann hierbei immer durch ein B ersetzt werden, ein r immer durch ein l und manchmal ein -. Die Buchstaben werden dabei immer zufällig ausgewählt.

2 Umsetzung

Ich habe zur Umsetzung der Lösungsidee 5 Klassen benötigt. Da ich nur eine Klasse (namens AI) abgeben kann, habe ich ein einfaches Programm entwickelt, dass mir diese 5 Klassen in eine Datei verpackt und die Hauptklasse `Power2KI` in AI umbenennt. Diese Klassen sind:

- `Power2KI`
- `Solution`
- `DancePattern`
- `Compleater`
- `Escaper`

Im folgenden ist jede Klasse für sich beschrieben und es wird gezeigt inwiefern diese die Lösungsidee umsetzen.

2.1 Power2KI

Diese Klasse ist die Hauptklasse. Sie erweitert die Klasse `AbstractKI`, welche nicht viel mehr macht als die `zug`-Methode in die Methoden `vor` und `nach` aufzuteilen und evtl. den Vortanz herauszufinden. Im Prinzip handelt es sich bei dieser Klasse um die leicht veränderte Beispiel-KI, wie man sie hier finden kann.

2.1.1 Die `vor`-Methode

Diese Methode ist dafür zuständig, die Vortänze zu generieren. Sie setzt dabei den Abschnitt 4 der Lösungsidee direkt um. Ich habe hierbei aber nicht allen Mustern dieselbe Warscheinlichkeit gegeben, weil die gegnerischen KIs mit einigen Mustern schlechter umgehen können als mit anderen. Im folgenden ist ein Überblick über die Auftritswarscheinlichkeit eines Vortanzes. Diese habe ich durch sehr oftes Hintereinanderausführen der `vor`-Methode herausbekommen.

- `FF489F..r.`: 28,4 %
- `B5F5l7B...`: 25,2 %
- `9rFr7rF.B.`: 24 %
- `9F9Fl7F...`: 16,8 %
- `9Fr.99BB..`: 3,2 %
- `999F.r.-r.`: 3 %

2.1.2 Die `nach`-Methode

Die `nach`-Methode ist in erster Linie dafür da, um die anderen Methoden zu "verwalten". Selber enthält sie nur einen kleinen Teil der Lösungsidee. Sie enthält eine for-Schleife, die an der Position i den String teilt und diesen dann wie in der Lösungsidee beschrieben escaped und evtl. compleated. Damit setzt sie diesen Teil der Lösungsidee um. Dieser Abschnitt sieht etwa so aus:

```
List<Solution> solutions = new LinkedList<>();
long time = System.currentTimeMillis();
for (int i = 0; (i < tanz.length()) && (System.currentTimeMillis()-time < 700); i++)
{
      String teil1 = tanz.substring(0, i);
      String teil2 = tanz.substring(i);
      String escaped1 = Escaper.escape(teil1);
      String escaped2 = Escaper.escape(teil2);
      if ((tanz.length() == 255) && (escaped1.length() < 8))
            escaped2 = Compleater.compleate(escaped2,
                        ((escaped1.length()<7) ? 9-escaped1.length() : 10-escaped1.length()));
      String s = Escaper.escape(escaped1+escaped2);
      solutions.add(new Solution(s, tanz));
}
```

Ich bin hierbei anfangs von einer Länge von 9 Zeichen ausgegangen. Die meisten Vortänze haben eine Länge von 10 Zeichen. Da die `compleate`-Methode dieses bei Bedarf aber eh verändert, ist es am sinnvollsten, bei 9 anzufangen. Vortänze, die weniger als 9 Zeichen haben, sollten eigentlich kein Problem darstellen, da es sehr schwer ist einen Vortanz zu erstellen, der

nur 8 Zeichen lang ist, trotzdem abgeschnitten ist und dann auch noch vervollständigt werden muss. Auf dem Turnierserver existiert kein solcher String, weshalb ich Vortänze mit 8 Zeichen vernachlässigen kann.

Anschließend wird in dem escapten Vortanz noch nach einer kompletten Schleife gesucht und darin der String in 2 Teile geteilt und escapt. Dieser wird anschließend nicht compleated, da er ja durch die Schleife in seiner Länge gleichbleiben muss.

2.2 Solution

Diese Klasse dient zur Darstellung und Speicherung einer Lösung. Dazu wird der Vortanz und der Nachtanz gespeichert. Der Nachtanz muss sowohl in escapter als auch in unescapter Form vorliegen; dies kann diese Klasse aber auch selber managen. Außerdem enthält diese Klasse den 3. Teil der Lösungsidee zum Bewerten der Lösung. Dies geschieht in einem eigenen Thread. Dieser besorgt sich, sofern noch nicht geschehen, sowohl den escapten als auch den unescapten Vortanz. Dazu wird der übergebene Nachtanz, dessen Format zunächst unbekannt ist, escapt. Wenn im übergebenen String bereits Zahlen oder Punkte enthalten sind verweigert der Escaper die Arbeit und es wird nix verändert. Anschließend wird der übergebene String unescapt. Wenn der unescapte String weder Zahlen noch Punkte enthält wird der String unverändert zurückgegeben. Andernfalls erhält man den unescapten String. Es gibt jedoch auch die Möglichkeit, im Konstruktor anzugeben worum es sich handelt. Dies ist wichtig, damit ein soeben escapter String nicht hundertmal escapt wird.

2.2.1 Die Bewertung einer Lösung

Die Bewertung einer Lösung, wie in der Lösungsidee beschrieben, geschieht in der `run`-Methode dieser Klasse (`Solution`). Wie der Name schon andeutet, geschieht dies in einem eigenen Thread. Der Hauptthread wartet, sobald die Lösung verlangt wird, bis dieser Thread fertig ist, d.h. bis `rating != -1`. Dabei ist `rating` das Feld, in dem das Rating, d.h. die Strafpunkte dieser Lösung gespeichert wird.

Um die Strafpunkte zu bestimmen nehme ich zuerst die Länge des escapten Nachtanz mal 3, wie auch in der Lösungsidee beschrieben. Anschließend rufe ich die `move`-Methode auf bis diese einmal `false` zurückgibt. Sobald sowohl die `move`-Methodenaufrufe für den Vortanz und für den Nachtanz `false` zurückgegeben haben ist die `run`-Methode fertig. Nach jedem Aufruf der `move`-Methode werden die Strafpunkte wie in der Lösungsidee beschrieben addiert. Das ganze sieht so aus:

```
// Positionen berechnen und dabei Strafpunkte berechnen
boolean continuei = true;
boolean continueo = true;
while (continuei || continueo)
{
      if (continuei) continuei = move(str, imitat);
```

```
        if (continueo) continueo = move(vor, original);

        strafpunkte += Math.max(Math.abs(imitat.x-original.x),
                    Math.abs(imitat.y-original.y));
}
```

Die `move`-Methode untersucht das Zeichen an der aktuellen Position. Diese wird zusammen mit den anderen Variablen (Bewegungsrichtung, x- und y-Wert) in der Klasse `Solution.PositionDescriptor` aufbewahrt. Wenn kein weiteres Zeichen mehr da ist, wird false zurückgegeben. Andernfalls wird bei einem Bewegungszeichen wie "F" oder "B" eine weitere Methode mit demselben Namen aufgerufen. Diese dreht die übergebenen x- und y-Werte und addiert sie dem PositionDescriptor-Object hinzu. Sollte es sich bei dem aktuell untersuchten Zeichen um ein Bewegungszeichen wie "r" oder "l" handeln wird die Bewegungsrichtung im PositionDescriptor-Object verändert. Wenn diese kleiner als 0 oder größer als 4 wird, wird diese entsprechend auf 0 oder 4 gesetzt.

2.3 DancePattern

Diese Klasse ist nicht direkt notwendig, ist aber relevant zum Speichern in der `escape`-Methode. Diese Klasse habe ich in der Lösungsidee auch schon kurz angerissen. Sie dient nur dazu, einen Teilstring und sein Auftreten im String zu speichern. Dies geschieht mit dem beiden Feldern `pattern` und `appearment`. Das Auftreten sollte zudem noch im Feld `insgAppearment` abgelegt werden. Dazu siehe später. Zusätzlich gibt es noch das Feld `length`, das die Länge des Pattern, also `pattern.length()`, enthält.

Hinausgehend über die Speicherfunktion hat diese Klasse noch ein paar statische Methoden. Eine davon ist `getLongest()`, welches aus der übergebenen Liste an `DancePattern` das Längste heraussucht und es anschließend zurückgibt. Dies wird auch von der `escape`-Funktion benutzt. Eine deutlich interessantere Methode heißt ebenfalls `escape`. Dies entspricht jedoch nicht der bis jetzt angesprochene `escape`-Methode in der Klasse `Escaper`, sondern dient, um ein `appearment`>9 in mehrere Zahlen aufzuteilen, die als Produkt das entsprechende `appearment` haben. Da hierbei das `appearment` verändert wird und die `escape`-Funktion dieses später noch benötigt, wird das alte `appearment` in der Variablen `insgAppearment` gespeichert. Dadurch kann die andere `escape`-Methode das appearment später noch finden.

2.4 Compleater

Diese Klasse setzt den 2. Teil der Lösungsidee mit der `compleate`-Methode um. Diese Methode dient um escapte Strings, die länger als 255 Zeichen waren und somit von Server auf 255 Zeichen gekürzt wurden, so zu verlängern, dass sie möglichst kurz sind (d.h. aus möglichst wenig Zeichen bestehen) und abgeschnitten aber wieder genau dasselbe zurückgeben. Dies kann leider nicht garantiert werden, ist aber von der Methode so gewollt. Als erstes wird der übergebene String `escaped` bei `maxlength` abgeschnitten. Sollte dieser noch nicht mit einer

Schleife versehen sein, die den ganzen abgeschnittenen String enthält, wird dies zuerst hinzugefügt. Anschließend wird die Schleife verlängert, d.h. die Zahl der Schleife wird inkrementiert. Deshalb wurde im vorhergehenden Schritt auch nur eine "8" und keine "9" hinzugefügt. Sollte die Schleife dennoch 9 enthalten und es sich auch nicht um eine Kombination wie "98" handeln, wo man die 8 inkrementieren kann, wird eine weitere "9" vornedrangehängt und entsprechend ein Punkt am Ende hizugefügt. Durch solche Aktionen wird der String innerhalb der Schleife natürlich immer kürzer. Deshalb kann es jetzt sinvoll sein, wenn `maxlength`<13 und die unescapten Inhalte vom übergebenen String `escaped` und dem Ergebnis dieser Methode nicht übereinstimmen `maxlength` zu inkrementieren und die Methode damit nochmal aufzurufen.

2.5 Escaper

Diese Klasse stellt die schon oft angesprochene `escape`-Methode zur Verfügung. Außerdem ist es z.B. bei der `Solution`-Klasse notwendig, escapte Strings wieder unescapen zu können. Hierzu stellt diese Klasse eine `unescape`-Methode zur Verfügung. Diese Methoden sind alle statisch und können somit einfach mit den entsprechenden Argumenten aufgerufen werden.

2.5.1 Die `escape`-Methode

Diese Methode setzt den 1. Teil der Lösungsidee um, der noch nicht durch die `nach`-Methode der Klasse `Power2KI` umgesetzt wird. Zuerst wird geschaut ob der String !=null ist, nicht so kurz ist dass es ihn verlängern würde, wenn man ihn escapt, und dass er komplett unescapt ist. Letzteres geschieht mit einem regex (regulärer Ausdruck):

```
if (!Pattern.matches("[rlFB\\-]0,", str))
{
      int pos0=-1, pos1=-1;
      for (int i = 0; i < str.length(); i++)
            if (Character.isDigit(str.charAt(i)))
                  pos0 = i+1;
      for (int i = str.length()-1; i >= 0; i-)
            if (str.charAt(i) == '.')
                  pos1 = i;
      if ((pos0 != -1) && (pos1 != -1) && (pos0 < pos1))
            return str.substring(0, pos0)
                        +escape(str.substring(pos0, pos1))
                        +str.substring(pos1);
      return str;
}
```

Hierbei wird bei einem unescapten String noch versucht einen Teil zu finden, der noch nicht escapt wurde. Dies ist bei den meisten komplett escapten Strings aber nicht möglich und es

wird der übergebene String unverändert zurückgegeben.

Wenn diese Abfragen nicht gescheitert sind, wird der String nach Pattern durchsucht. Dies geschieht wie in der Lösungsidee beschrieben. Ich setze i am Anfang auf 1 und inkrementiere dies bis `str.length()/2`. Für jedes i wird ein `DancePattern` `str.substring(0, i)` gebildet. Wenn dies durch ein anderes, bereits gefundenes Pattern ersetzt werden kann, dann dieses Pattern wie ich bereits in der Lösungsidee geschrieben habe ignorieren. Dies passiert so:

```
// überprüfen, ob das DancePattern nicht ein anderes ersetzt, wie etwa bei
// einem bereits gefundenen DancePattern "Fr" wäre "FrFr" überflüssig
boolean add = true;
for (DancePattern p0 : pattern)
{
      boolean add0 = false;
      if (p.length/p0.length != 1.0*p.length/p0.length)
            continue;
      for (int j = 0; j+p0.length <= p.length; j+=p0.length)
      {
            if (!p0.pattern.equals(p.pattern.substring(j, j+p0.length)))
                  add0 = true;
      }
      if (!add0) { add = false; break; }
}
```

Nachdem alle brauchbaren Pattern gefunden worden sind, wird mithilfe der Funktion `getLongest` der Klasse `DancePattern` das beste `DancePattern` davon herausgefunden. Ist dieses null, wird, wie bereits in der Lösungsidee beschrieben, der Substring von 1 bis Ende escapt und das erste Zeichen davorgehängt. Ist dies nicht null, wird das Pattern durch die `escape`-Methode der Klasse `DancePattern` geschickt und anschließend zusammen mit dem escapten, restlichen Teil des Strings, der nicht im Pattern enthalten ist, zurückgeben.

2.5.2 Die `unescape`-Methode

Diese Methode ist zwar nicht direkt in der Lösungsidee beschrieben, ist aber trotzdem notwendig. Als allererstes wird überprüft, ob überhaupt Schleifen oder Punkte enthalten sind. Wenn nicht, wird einfach der Ausgangsstring unverändert zurückgegeben. Ansonsten arbeitet diese Methode genauso wie die `escape`-Methode rekursiv. Sie gibt dabei immer den index zurück, bei dem sie aufgehört hat zu unescapen, damit die aufrufende Methode an dieser Stelle die Arbeit vortsetzen kann. Pro Methodenaufruf wird dabei genau eine Schleife behandelt. Sollte eine weitere Schleife gefunden werden, wird die Methode erneut aufgerufen. Dadurch wird auf jeden Fall der komplette String unescapt. Wenn jedoch eine ungleiche Anzahl an Punkten und Zahlen

vorhanden ist, sehen die Ergebnisse etwas merkwürdig aus. So gibt diese Methode für die Eingabe `"9F.."` `"F.F.F.F.F.F.F.F."` zurück. Bei `"98F."` wird einfach der Ausgangsstring zurückgegeben und gar nichts escapt. Solche Fälle sollten jedoch auch nie passieren, da in der `compleate`-Methode ja extra drauf geachtet wird, dass sowas nicht passiert, und alle anderen Methoden solche Ergebnisse nie zurückliefern, da sie keine escapten Strings abschneiden.

3 Beispiele

Hier folgen ein paar Beispiel-Herausforderungen, die ich anhand der Debugausgabe meiner KI und den Bildern des Turnierservers zusammengesetzt habe. Die Beispiele sind so gewählt, dass man alle möglichen Funktionen und deren Nicht-Funktion (in einigen Fällen) sehen kann.

Die Vortänze habe ich einmal durch meine `escape`-Funktion laufen lassen, da ich sie nicht anders darstellen kann, da der String sonst aus dem Bildschirm laufen würde. Ich habe jedoch vorher überprüft, dass diese auch wirklich richtig escapt wurden usw.. Die Vortänze habe ich aber nicht durch meine `compleate`-Methode geschickt, sodass sie ganz einfach wiederherrstellbar sind.

3.1 Beispiel 1 - gegen Master und Hope

Die Herausforderung ist hier hinterlegt.

Runde	Vortanz	meine KI	Strafpunkte		
			power2	Master	Hope
1	558F.rF.F.	558F.rF.F.	30	0	30
2	-	B5F5l7B...	0	30	30
3	5r.555FF...	5r.555FF...	33	30	0
4	99rB.r9B..	99rB.r9B..	30	30	0
5	-	F5B5r7F...	0	30	30
6	l9F.79rF..r	l8F.88Fr..	30	0	30
7	-	6lFl3lF.B.	0	30	30
8	378F...r67FF..FF	9378F...r.	30	0	30
9	99FF..Fr59FF..F	999FF..Fr.	30	30	0
			183	180	180

Bei dieser Herausforderung kann man zum einen gut die Umsetzung der Vortänze sehen. Es kommen diesmal nur die beiden Vortänze mit der größten Warscheinlichkiet vor. Außerdem kann man in der 6. Runde gut den Nutzen der `nach`-Methode sehen, welchen `"l8F."` abschneidet, wodurch der String kürzer escapt werden konnte. In Runde 3 kann man jedoch leider auch sehen dass die `compleate`-Methode nicht in der Lage ist, die 555FF... kürzer zu machen, was per Hand durchaus möglich ist.

3.2 Beispiel 2 - gegen Tanzwiese und this.thinkof()

Die Herausforderung ist hier hinterlegt.

Runde	Vortanz	meine KI	Strafpunkte		
			power2	Tanzwiese	this.thinkof()
1	-	FF489F..r.	0	36	30
2	79F.9FrF..FFF	89F.9FrF..	30	33	0
3	64l9F..lF.lFF	74l9F..lF.	30	0	30
4	64l9F..lF.lFF	74l9F..lF.	30	0	30
5	9F.99FlF..	9F.99FlF.	30	33	0
6	-	9B9Bl7B...	0	42	33
7	99rB.r9B..	99rB.r9B..	30	33	0
8	-	6lFl3lF.B.	0	33	30
9	64l9F..lF.lFF	74l9F..lF.	30	0	30

Bei dieser Herausforderung kann man sehen, dass die Vortänze intelligent gewählt sind: Tanzwiese bekommt keinen meiner Vortänze optimal escapt, this.thinkof() scheitert zwar nicht bei allen, kann den Vortanz aus Runde 6 aber auch nicht optimal escapen.

4 Quelltext

Hier sind einige wichtige Quelltextausschnitte. Der komplette Quelltext ist im Ordner `power2`. Die Datei, die aus den 5 Klassen besteht, ist unter dem Namen `AI.java` enthalten.

4.1 Die `escape`- und die `unescape`-Methode aus der Klasse `Escaper`

```
/**
* Diese Klasse stellt Methoden zum escapen und unescapen von Tänzen bereit.
*
* @author Dominic S. Meiser
*/
public class Escaper
{

	/**
	* Diese Methode escaped einen Tanz.<br><br>Beispiel:
	* Aus <code>FrFrFrFrFrFrFr</code> wird <code>8Fr.</code>
	*/
	public static String escape (String str)
	{
		if (str == null)
		{
```

```
      System.err.println("CANT ESCAPE NULL");
      return "";
}

if (str.length() <= 3) return str;
if (!Pattern.matches("[rlFB\\-]{0,}", str))
{
      int pos0=-1, pos1=-1;
      for (int i = 0; i < str.length(); i++) if (Character.isDigit(str.charAt(i))) pos0 = i+1;
      for (int i = str.length()-1; i >= 0; i-) if (str.charAt(i) == '.') pos1 = i;
      if ((pos0 != -1) && (pos1 != -1) && (pos0 < pos1))
            return str.substring(0, pos0)+escape(str.substring(pos0, pos1))+str.substring(pos1);
      return str;
}

// Enthält alle gefundenen DancePattern
List<DancePattern> pattern = new LinkedList<>();

// Nach DancePattern suchen
for (int i = 1; i <= str.length()/2; i++)
{
      DancePattern p = new DancePattern();
      p.length = i;
      p.pattern = str.substring(0, i);

      // Die Anzahl der Auftritte des DancePattern suchen
      for (int j = i; j+i-1 < str.length(); j+=i)
      {
            if (str.substring(j, j+i).equals(p.pattern))
                  p.appearment++;
            else break;
      }
      p.insgAppearment = p.appearment;

      // überprüfen, ob das DancePattern nicht ein anderes ersetzt, wie etwa bei
      // einem bereits gefundenen DancePattern "Fr" wäre "FrFr" überflüssig
      boolean add = true;
      for (DancePattern p0 : pattern)
      {
            boolean add0 = false;
            if (p.length/p0.length != 1.0*p.length/p0.length) continue;
```

```
                for (int j = 0; j+p0.length <= p.length; j+=p0.length)
                {
                      if (!p0.pattern.equals(p.pattern.substring(j, j+p0.length)))
                            add0 = true;
                }
                if (!add0) { add = false; break; }
          }

          if (add) pattern.add(p);
     }

     // Das beste Pattern herausfinden und den nicht durch das Pattern abgedeckten Teil
     // des String (rekursiv) nochmal escapen.
     DancePattern bestPattern = DancePattern.getLongest(pattern);
     DancePattern best = DancePattern.escape(bestPattern);
     if (best == null) return str.substring(0, 1)+((str.length() > 1) ? escape(str.substring(1)) : "");
     else
     {
          String str0 = best.appearment+escape(best.pattern)+"."+
                      ((str.length() > 1) ? escape(str.substring(best.insgAppearment*best.length)) : "");
          if (str0 != str) str0 = escape(str0);
          return str0;
     }
}



/**
* Diese Methode unescaped einen Tanz.<br><br>Beispiel:
* Aus <code>8Fr.</code> wird <code>FrFrFrFrFrFrFr</code>
*/
public static String unescape (String escaped)
{
     if (Pattern.matches("[FBrl\\-]{0,}", escaped))
     {
          return escaped;
     }
     String str = unescape(escaped, 0);
     return str;
}
private static String unescape (String escaped, int pos)
```

```
{
	if (Pattern.matches("[FBrl\\-]{0,}", escaped)) return escaped;

	escaped = escaped.substring(pos);
	StringBuilder unescaped = new StringBuilder();

	int pos0=-1, pos1=-1, inside=0; DancePattern pattern = new DancePattern();
	for (int i = 0; i < escaped.length(); i++)
	{
		char c = escaped.charAt(i);
		if (Character.isDigit(c))
		{
			if (pos0 != -1)
				inside++;
			else
			{
				pos0 = i;
				pattern.appearment = Integer.parseInt(escaped.substring(i, i+1));
				pattern.insgAppearment = pattern.appearment;
			}
		}
		if (c == '.')
		{
			if (inside != 0)
				inside-;
			else
			{
				pos1 = i;
				pattern.pattern = unescape(escaped.substring(pos0+1, pos1), 0);
				pattern.length = pattern.pattern.length();
			}
		}
	}

	if ((pos0 != -1) && (pos1 != -1))
	{
		unescaped.append(escaped.substring(0, pos0));
		for (int i = 0; i < pattern.appearment; i++)
			unescaped.append(pattern.pattern);
		unescaped.append(unescape(escaped, pos1+1));
	}
```

```
            else unescaped.append(escaped);

            return unescaped.toString();
      }
}
```

4.2 Die `compleate`-Methode aus der Klasse `Compleater`

```
/**
* Diese Methode vervolständigt den abgeschnittenen String <code>escaped</code>
* zu einem String mit der maximalen Länge <code>maxlength</code>. Wenn da nur
* Stuß rauskommt wird evtl. ein ein wenig längerer String zurückgegebene.
* Außerdem ist nicht garantiert das das Ergebnis der Eingebe entspricht.
*/
public static String compleate (String escaped, int maxlength)
{
      // Zuerst einen String mit maximal maxlength Zeichen "abschneiden"
      String str = escaped.substring(0, ((escaped.length() > maxlength) ? maxlength : escaped.length()));

      // Diesen, falls noch nicht vorhanden, mit einer "Schleife" versehen
      if (!Character.isDigit(str.charAt(0)) || (str.charAt(str.length()-1) != '.'))
            str = "8"+str.substring(0, ((str.length() < maxlength-2) ? str.length() : maxlength-2))+".";
      // Die Anzahl der Punkte und Zahl vorrübergehend in Ordnung bringen
      for (int i = 0; i <= str.length()-2; i++)
      {
            if (Character.isDigit(str.charAt(i)) && (str.charAt(i+1) == '.'))
            {
                  System.out.println("\tremove: "+str.substring(0, i)+" |- "+str.substring(i, i+2)+" -| "+str.substring(i+2));
                  str = str.substring(0, i)+str.substring(i+2);
            }
      }

      // Den String durch eine "Schleife" verlängern
      if (Integer.parseInt(str.substring(0, 1)) < 9)
            str = (Integer.parseInt(str.substring(0, 1))+1)+str.substring(1);
      else if (Character.isDigit(str.charAt(1)) && (Integer.parseInt(str.substring(1, 2)) < 9) &&
                  (str.charAt(str.length()-2) == '.'))
            str = str.substring(0, 1)+(Integer.parseInt(str.substring(1, 2))+1)+str.substring(2);
      else
            str = "99"+str.substring(1, ((str.length()-2 < 6) ? str.length()-2 : 6))+"..";
```

```
	// Die Anzahl der Punkte und Nummern letztendlich in Ordnung bringen
	while (countDots(str) > countNums(str))
		str = str.substring(0, str.lastIndexOf('.')) + str.substring(str.lastIndexOf('.')+1);
	for (int i = str.length()-1; (countNums(str) > countDots(str)) && (i > 0); i-)
	{
		if (str.length() < maxlength)
		{
			str += ".";
			i = str.length();
		}
		else
			str = str.substring(0, i)+"."+((i != str.length()) ? str.substring(i+1) : "");
	}
	for (int i = 0; i <= str.length()-2; i++)
	{
		if (Character.isDigit(str.charAt(i)) && (str.charAt(i+1) == '.'))
		{
			System.out.println("\tremove: "+str.substring(0, i)+" |- "+str.substring(i, i+2)+" -| "+str.substring(i+2));
			str = str.substring(0, i)+str.substring(i+2);
		}
	}

	// Wenn der String viel zu lang ist maxlength evtl. etwas vergrößern
	if (((str.length() > 13) || !Escaper.unescape(str).startsWith(Escaper.unescape(escaped)))
			&& (maxlength < 13))
	{
		System.out.println(" -> compleater: increaing maxlength");
		String str0 = compleate(escaped, maxlength+1);
		System.out.println(" -> \""+str+"\" -> \""+str0+"\"");
		return str0;
	}

	System.out.println(" -> compleater returns \""+str+"\" for input \""+escaped+"\"");
	return str;
}
```

4.3 Die Klasse `Power2KI` mit dem Methoden `vor` und `nach`

```
/**
* Diese Klasse ist die KI. Sie produziert beim Vortanzen einen mehr oder wenig
* zufälligen String, der einem zufällig bestimmten Format folgt. Beim Nachtanzen
```

```
* wird zuerst der String automatisch escaped. Anschließend versucht die KI, das
* Pattern zu ergänzen, da der Server String, die länger als 255 Zeichen sind,
* abschneidet. Dadurch stehen verschiedene Solutions bereit, von denen anschließend
* die beste ausgewählt wird. Von der besten Solution werden dann noch unnötige Zeichen
* am Ende abgeschnitten, die durch ihnre Anwesenheit mehr Strafpunkte erzeugen als
* verhindern.<br>
* Beachten Sie, dass die zug()-Methode automatisch von der Superklasse in die
* beiden Methoden vor() und nach() aufgeteilt wird. Siehe {@link AbstractKI}
* <p>
* Zugehörige KI: power2 von meiserdo
* @version 26
* @author Dominic S. Meiser
*/
public class Power2KI extends AbstractKI
{
	/**
	* Diese Methode generiert einen zufälligen Vortanz. Der Tanz folgt dabei
	* einem der folgenden Muster:
	* <ul>
	* <li>999F.r.-r.
	* <li>9Fr.99BB..
	* <li>FF489F..r.
	* <li>9rFr3rF.B.
	* <li>9F9Fl7F...
	* <li>B5F5l7B...
	* </ul>
	* Das Muster wird zufällig bestimmt, wobei nicht jedesm Muster dieselbe
	* Warscheinlichkeit hat. F kann durch B, r durch l (und -) ersetzt werden.
	*
	* @param id Die Identifikationsnummer der KI.
	* @param zustand Der aktuelle Spiel.Zustand des Spiels.
	* @param zug Das Objekt, mit dem man den Spiel.Zug durchführen kann.
	*/
	public void vor (int id, Spiel.Zustand zustand, Spiel.Zug zug)
	{
		String str = "";

		if (Math.random() < 0.03)
		{
			char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
			char c1 = ((Math.random() < 4f/6f) ? 'r' : ((Math.random() < 0.75) ? 'l' : '-'));
```

```
        // Format: 999F.r.-r.
        str = "999"+c0+"."+c1+".-"+c1+".";
}

else if (Math.random() < 0.03)
{
        char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
        char c1 = ((Math.random() < 0.5) ? 'F' : 'B');
        char c2 = ((Math.random() < 4f/6f) ? 'r' : ((Math.random() < 0.75) ? 'l' : '-'));
        // Format: 9Fr.99BB..
        str = "9"+c0+""+c2+".99"+c1+""+c1+"..";
}

else if (Math.random() < 0.25)
{
        char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
        char c1 = ((c0 == 'F') ? 'B' : 'F');
        char c2 = ((Math.random() < 0.5) ? 'r' : 'l');
        int c3 = (int)(Math.random()*4+4);
        // Format: 9rFr3rF.B.
        str = c3+""+c2+""+c0+""+c2+"3"+c2+""+c0+"."+c1+".";
}

else if (Math.random() < 0.4)
{
        char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
        char c1 = ((Math.random() < 0.5) ? 'r' : 'l');
        // Format: FF489F..r.
        str = c0+""+c0+"489"+c0+".."+c1+".";
}

else if (Math.random() < 0.4)
{
        char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
        char c1 = ((Math.random() < 0.5) ? 'r' : 'l');
        // Format: 9F9Fl7F...
        str = "9"+c0+"9"+c0+""+c1+"7"+c0+"...";
}

else
{
```

```
            char c0 = ((Math.random() < 0.5) ? 'F' : 'B');
            char c1 = ((c0 == 'F') ? 'B' : 'F');
            char c2 = ((Math.random() < 0.5) ? 'r' : 'l');
            // Format: B5F5l7B...
            str = c0+"5"+c1+"5"+c2+"7"+c0+"...";
      }

      zug.ausgabe("<vor><ergebnis>"+str+"</ergebnis></vor>");
      zug.tanzen(str);
}



/**
* Diese Methode erstellt den Nachtanz des angegebenen Vortanzes. Dabei werden zuerst
* zwei sehr einfache Lösungen erstellt, die einmal den Vortanz (der automatisch escaped
* wird) und einmal einen leeren String enthält. Anschließend werden noch 2 Lösungen
* erstellt, die versuchen, einen abgeschnittenen String (der Server schneidet Strings,
* die länger als 255 Zeichen sind, ab) "wiederherzustellen", d.h. sich den abgeschnittenen
* Teil zu erschließen. Dies kann durch escapen zu einem kürzeren String führen.
*
* @param id Die Identifikationsnummer der KI.
* @param zustand Der aktuelle Spiel.Zustand des Spiels.
* @param zug Das Objekt, mit dem man den Spiel.Zug durchführen kann.
* @param tanz Der Vortanz des Vortänzers.
*/
public void nach (int id, Spiel.Zustand zustand, Spiel.Zug zug, String tanz)
{
      long insgtime = System.currentTimeMillis();

      // Wenn der Tanz nix enthält ist es am besten nix zurückzugeben
      if (tanz.equals(""))
      {
            zug.ausgabe("<nach><vortanz></vortanz><ergebnis></ergebnis></nach>");
            zug.tanzen("");
            return;
      }



      // 2 ganz einfache Lösungen hinzufügen: einmal mit nix und einmal einfach den
```

```
        // übergebenen Tanz, der sowieso automatisch escaped wird. Dazu noch eine Lösung
        // erstellen, bei der der Tanz rückwärts escaped wird, wenn der Tanz nicht abgeschnitten
        // wurde.
        Solution s0 = new Solution(tanz, tanz);
        System.out.println(tanz+": "+s0.getRating());
        Solution s1 = new Solution("", tanz);
        System.out.println(": "+s1.getRating());


        // Versuchen, die durch einen zu langen String (>255 Zeichen, d.h. der Server hat
        // den String abgeschnitten) abgeschnittenen Zeichen zu rekonstruieren
        Solution s2 = null;
        List<Solution> solutions = new LinkedList<>();
        long time = System.currentTimeMillis();
        for (int i = 0; (i < tanz.length()) && (System.currentTimeMillis()-time < 550); i++)
        {
              String teil1 = tanz.substring(0, i);
              String teil2 = tanz.substring(i);
              System.out.print(teil1+" | "+teil2);
              String escaped1 = Escaper.escape(teil1);
              String escaped2 = Escaper.escape(teil2);
              System.out.print(" = "+escaped1+""+escaped2);
              if ((tanz.length() == 255) && (escaped1.length() < 8))
                    escaped2 = Compleater.compleate(escaped2,
                                ((escaped1.length()<7) ? 9-escaped1.length() : 10-escaped1.length()));
              String s = Escaper.escape(escaped1+escaped2);
              System.out.println(" = "+s);
              solutions.add(new Solution(s, tanz));
//                if ((s.length() <= 10) && (Escaper.unescape(s).startsWith(tanz))) break;
        }
        s2 = new Solution(Solution.getBest(solutions), tanz);


        // Versuchen, einen nicht abgeschnittenen String durch Finden eines immer wiederkehrenden
        // Teils, darin 2 Teile zu finden und dies zu escapen, wenn vorher noch keine Lösung mit
        // <= 10 zeichen gefunden wurde
        Solution s3 = null;
        if ((s0.getEscapedTanz().length() > 10) && (s2.getEscapedTanz().length() > 10))
        {
              solutions = new LinkedList<>();
              DancePattern pattern = new DancePattern();
```

```
	{
		int startnums = 0;
		for (char c : s0.getEscapedTanz().toCharArray())
		{
			if (Character.isDigit(c)) startnums++;
			else break;
		}
		int enddots = 0;
		for (int i = s0.getEscapedTanz().length()-1; i > 0; i-)
		{
			if (s0.getEscapedTanz().charAt(i) == '.') enddots++;
			else break;
		}
		if ((enddots == 0) || (startnums == 0)) pattern = null;
		else
		{
			pattern.pattern = Escaper.unescape(s0.getEscapedTanz().substring(
					Math.min(startnums, enddots),
					s0.getEscapedTanz().length()-Math.min(startnums, enddots)));
			for (int i = 0; i < Math.min(startnums, enddots); i++)
				pattern.appearment *= Integer.parseInt(s0.getEscapedTanz().substring(i, i+1));
			pattern.length = pattern.pattern.length();
		}
	}
	if ((pattern != null) && Pattern.matches("[FBrl\\-]{1,}", pattern.pattern))
	{
		for (int i = 0; (i < pattern.length) && (System.currentTimeMillis()-time < 800); i++)
		{
			String teil1 = pattern.pattern.substring(0, i);
			String teil2 = pattern.pattern.substring(i);
			System.out.print(teil1+" | "+teil2);
			String escaped1 = Escaper.escape(teil1);
			String escaped2 = Escaper.escape(teil2);
			DancePattern result = new DancePattern(escaped1+escaped2, pattern.appearment);
			result = DancePattern.escape(result);
			System.out.println(" = "+result.appearment+result.pattern+".");
			solutions.add(new Solution(result.appearment+result.pattern+".", tanz));
		}
		s3 = new Solution(Solution.getBest(solutions), tanz);
	}
}
```

```
            String best = Solution.getBest(Arrays.asList(s0, s2, s3));

            // alle möglichen Zeichen abschneiden und schaun welches davon die beste Lösung ist
            List<Solution> bests = new LinkedList<>();
            bests.add(s1); // diese Lösung erst jetzt beachten
            for (int i = 0; i < best.length()-1; i++)
                  bests.add(new Solution(best.substring(0, best.length()-i), tanz, true));

            best = Solution.getBest(bests);
            if (best == null) best = "";

            // Die beste Lösung abgeben und eine Debugausgabe machen
            zug.ausgabe("<nach><vortanz>"+tanz+"</vortanz><ergebnis>"+best+"</ergebnis><time>"+
                        (System.currentTimeMillis()-insgtime)+"</time></nach>");
            zug.tanzen(best);
      }
}
```

4.4 Die Bewertung einer Lösung aus der Klasse `Solution`

```
/**
* Diese Methode berechet die Strafpunkte. Diese können anschließend über getRating()
* abgerufen werden.
*/
public void run ()
{
      // Strings escapen und unescapen
      if (escaped == null) escaped = Escaper.escape(str);
      str = Escaper.unescape(escaped);

      if (str.length() > 255) str = str.substring(0, 255);
      if (escaped.length() > 255) escaped = escaped.substring(0, 255);

      // Strafpunkte des Immitats
      int strafpunkte = escaped.length()*3;

      // Position des Imitats
      int xi = 0, yi = 0;
      // Bewegubg des Imitats im Uhrzeigersin
```

```
	int bi = 0;

	// Position des Originals
	int xo = 0, yo = 0;
	// Bewegung des Originals im Uhrzeigersinn
	int bo = 0;

	// Positionen umformen
	PositionDescriptor imitat = new PositionDescriptor (xi, yi, bi);
	PositionDescriptor original = new PositionDescriptor (xo, yo, bo);

	// Positionen berechnen und dabei Strafpunkte berechnen
	boolean continuei = true;
	boolean continueo = true;
	while (continuei || continueo)
	{
		if (continuei) continuei = move(str, imitat);
		if (continueo) continueo = move(vor, original);

		strafpunkte += Math.max(Math.abs(imitat.x-original.x), Math.abs(imitat.y-original.y));
	}

	// Strafpunkte speichern
	rating = strafpunkte;
}

/**
* Bewegt die Positionin <code>pos</code> um das Zeichen tanz.charAt(pos.mvpos) weiter.
* Siehe {@link Solution#move(int, int, PositionDescriptor)}
*/
private boolean move (String tanz, PositionDescriptor pos)
{
	if (pos.mvpos >= tanz.length()) return false;

	char c = tanz.charAt(pos.mvpos);
	switch (c)
	{
		case 'F': move(0, -1, pos); break;
		case 'B': move(0, 1, pos); break;
		case 'r': pos.b-; if (pos.b < 0) pos.b = 3; break;
		case 'l': pos.b++; if (pos.b > 3) pos.b = 0; break;
```

```
	}

	pos.mvpos++;
	return true;
}

/**
* Bewegt die Position um (x|y) weiter. Dabei wird die Richtung in pos.b beachtet.
*/
private void move (int x, int y, PositionDescriptor pos)
{
	int real_x = x;
	int real_y = y;
	switch (pos.b)
	{
		case 1: real_y = x; real_x = y*-1; break;
		case 2: real_y = y*-1; real_x = x*-1; break;
		case 3: real_y = x*-1; real_x = y; break;
	}
	pos.x += real_x;
	pos.y += real_y;
}
```