Aufgabe 3: Wortsuche

Team-ID: 00105

Team: Lennart Enns

Bearbeiter dieser Aufgabe:

Lennart Enns

Inhaltsverzeichnis

Lösungsidee

Da es mindestens 3 verschiedene auszuwählende Schwierigkeitsgrade geben soll, habe ich mir zuerst überlegt, von welchen Kriterien diese abhängen sollen. Neben der Häufigkeit von Fragmenten der gesuchten Wörter im Buchstabengitter können auch möglichen Ausrichtungen eines Wortes eine Rolle spielen, z.B. nur vertikale und horizontale Wörter bei einem einfacheren Grad und sonst auch diagonale. Außerdem könnte auch die Wahrscheinlichkeit jedes einzelnen Wortes, sich mit einem anderen an einer bestimmten Stelle mit demselben Buchstaben zu „verbinden“, mit zunehmender Schwierigkeit sinken.

Im Programm soll also zuerst die zu öffnende Textdatei und der Schwierigkeitsgrad abgefragt werden. Nachdem die entsprechende Textdatei nun ausgelesen und ihr relevanter Inhalt in praktischerer Form gespeichert wurde, soll das Programm beginnen, die zu suchenden Wörter in einer zufällig ausgewählten, dem Schwierigkeitsgrad entsprechenden Richtung nacheinander in das Gitter einzufügen. Sobald dieser Vorgang beendet wurde, könnten zufällige Wortfragmente bzw. andere Buchstabengruppen ausgewählt

werden, um die übrigen Lücken aufzufüllen. Am Ende werden schließlich das fertige Buchstabengitter und die zu suchenden Wörter ausgegeben.

Umsetzung

Schwierigkeitsgrade:

Ich habe zur Umsetzung die Programmiersprache Python verwendet. Bevor ich allerdings mit der Umsetzung angefangen habe, habe ich mir überlegt, welcher Schwierigkeitsgrad welche Eigenschaften haben soll. Dabei habe ich mir folgendes überlegt:

Bei Stufe 1 sind keine Wörter diagonal ausgerichtet, Verbindungen zwischen Wörtern sind wahrscheinlicher und es werden keine ablenkenden Wortfragmente aktiv ausgewählt.

Stufe 2 ist genau so, nur dass hier die Hälfte der Fragmente den Zielwörtern ähneln.

Bei Stufe 3 kann ein Wort alle möglichen Ausrichtungen haben und Verbindungen sind unwahrscheinlicher. Auch hier sind die Hälfte der Fragmente Zielwort-orientiert.

Bei Stufe 4 ist der einzige Unterschied zu Stufe 3, dass alle Fragmente den zu suchenden Wörtern ähneln, was diese besonders schwer zu finden macht.

Das Programm:

Zuerst werden im Programm die benötigten Module importiert. Ich verwende „numpy“ zum Erstellen eines 2-dimensionalen Arrays als Gitter. Dazu das Modul „os“, um einfacher auf die Beispieldateien zugreifen zu können und „random“ zum Generieren von Zufallszahlen und mischen von Listen, um möglichst viel Variabilität in die Erstellung des Rätsels zu bringen. Weil die deutschen Umlaute von Python manchmal nicht richtig eingelesen werden, habe ich das Modul „codecs“ benutzt und zuletzt noch „string“, um hierraus das Alphabet in Großbuchstaben zu importieren und anschließend die Umlaute hinzuzufügen.

```
import numpy as np
import os
import random
import codecs
from string import ascii_uppercase
alphabet = list(ascii_uppercase)+["Ä", "Ö", "Ü"]
```

Um die Abfrage der benötigten Werte übersichtlicher zu machen, habe ich eine Funktion dazu definiert. Sie benötigt als Parameter die zu stellende Frage nach dem Wert, die Bedingung, bei der die Antwort nicht gültig ist und eine entsprechende Antwort, die in diesem Fall ausgegeben wird. Dabei wird die Fehlerbedingung als String übergeben und erst in der Funktion umgewandelt, damit es außerhalb dieser nicht zu einem Programmfehler wegen unbekannter Variablen kommt.

```
def abfrage_zahl(frage, fehler, fehler_antwort):
    while True:
        try:
            ausgabe = int(input(frage))
            if eval(fehler):
                raise Exception
            break
        except:
            print(fehler_antwort+"\n")
    return ausgabe
```

Vor der Wertabfrage habe ich in der Variablen „leer“ noch ein Platzhalterzeichen für leere Stellen im Gitter definiert und die Anzahl an Beispieldateien ermittelt.

```
leer = "."
n_Dateien = len(os.listdir("Beispieldateien"))
```

Dann wird der Index der zu öffnenden Textdatei mit der vorher definierten Funktion und der entsprechenden Frage und Fehlerantwort als Argumente abgefragt. Wenn keine Beispieldatei mit dem angegebenen Index existiert, ist die Fehlerbedingung erfüllt.

```
datei_nummer = abfrage_zahl("Welche Textdatei soll verwendet werden? ",
"not ausgabe in range(0, n_Dateien)", "Bitte gebe eine Zahl von 0 bis "+str(n_Dateien-1)+"
ein.")
```

Nachdem die Schwierigkeitsgrade und ihre Eigenschaften zur Information für den Benutzer ausgegeben wurden, wird auch die Schwierigkeit abgefragt, wobei die Fehlerbedingung ist, dass die Antwort nicht im Bereich vom 1 bis 4 liegt.

```
stufe = abfrage_zahl("Was ist der gewünschte Schwierigkeitsgrad (Zahl von 1 bis 4)? ",
```

```
                    "not ausgabe in range(1, 5)", "Bitte gebe eine Zahl von 1 bis 4
ein.")
```

Jetzt wird die ausgewählte Beispieldatei geöffnet. Dabei habe ich die Funktion open aus dem codecs-Modul verwendet und utf-8 als Codierung ausgewählt, weil so auch die Umlaute richtig eingelesen werden.

```
with codecs.open("Beispieldateien\\"+os.listdir("Beispieldateien")[datei_nummer],
 "r", "utf-8") as file:
```

Als nächstes wird eine Liste aus den Zeilen der Datei erstellt und von jeder Zeile das Zeilenende entfernt. Die erste Zeile der Datei, welche die Maße des Gitters enthält, wird in eine Liste umgewandelt und die Werte in Integer-Werte konvertiert.

```
Zeilen = file.readlines()
for n in range(len(Zeilen)):
    Zeilen[n] = Zeilen[n].strip()
maße = Zeilen[0].split(" ")
maße = [int(maße[0]), int(maße[1])]
```

Dann wird das Array „gitter“ mit dem Datentyp String initialisiert und mit den Platzhalter-Strings aufgefüllt. Die zweite Zeile (Anzahl der Wörter) wird in einen Integer konvertiert und gespeichert und weil anschließend die ersten beiden Zeilen nicht mehr benötigt werden, werden sie gelöscht.

```
gitter = np.empty(maße, str)
gitter.fill(leer)
n_wörter = int(Zeilen[1])
del Zeilen[:2]
```

Nun wird eine Liste „wörter“ erstellt, die aus den jeweils in Großbuchstaben umgewandelten Wörtern aus der Datei besteht. Die zu findenden Wörter werden dann von Kommata getrennt ausgegeben.

```
wörter = list(wort.upper() for wort in Zeilen)
print("\nZu findende Wörter:")
print(", ".join(wörter)+"\n")
```

Die Wort-Klasse:

Danach habe ich die Klasse „Wort“ definiert, von der später jedes Wort im Gitter ein Objekt sein soll. Das erleichtert das Einfügen, Verbinden und Interagieren der Wörter zwischeneinander.

```
class Wort():
```

Der Konstruktor benötigt die Parameter „typ“ und „string“. Wenn „typ“ auf False gesetzt ist, repräsentiert das Objekt ein Wortfragment oder eine zufällige Buchstabengruppe, bei True ein zu findendes Wort. „string“ enthält den String des Objektes, z.B. „BAUSTELLE“. Der String wird zusätzlich zum Attribut „string“ auch im Attribut „string_echt“ gespeichert, weil der String, nachdem seine Länge gespeichert wurde, zu einer Wahrscheinlichkeit von 50% invertiert wird. Das ermöglicht es, nur die Hälfte aller möglichen Ausrichtungen berücksichtigen zu müssen und trotztdem alle zu ermöglichen. Die Attribute „richtung“, „anfang“ und „pos“ werden erst einmal mit None initialisiert und später geändert.

```
    def __init__(self, typ, string):
        self.typ = typ
        self.string = string
        self.string_echt = string
        self.len = len(self.string)
        if random.randint(0, 1):
            self.string = self.string[::-1]
        self.richtung = None
        self.anfang = None
        self.pos = None
```

Die Klassenmethode „einfügen“ fügt den String des Wortes an der angegebenen Stelle und in der angegebenen Richtung ein.

```
    def einfügen(self, richtung, anfang):
```

In der Methode wird zuerst die Funktion „ungültig“ definiert, die aus zwei for-Schleifen besteht. Mit ihr soll nach Festlegung der Positionen der einzelnen Buchstaben des Wortes im Gitter überprüft werden, ob das Wort an dieser Stelle eingefügt werden darf.

In der ersten for-Schleife wird für jede der Positionen überprüft, ob sie leer ist oder der Buchstabe dort dem dort einzufügenden Buchstaben entspricht. Trifft beides nicht zu, würde ein Einfügen andere Wörter überschreiben und ist somit nicht zulässig.

In der zweiten for-Schleife wird für jeden Wert der Positionen überprüft, ob er negativ ist, weil das beim anschließenden Ausmachen der Positionen vorkommen kann. Auch hier wird in diesem Fall eine Exception hervorgerufen.

```
        def ungültig(pos_z, pos_s):
            for i in range(len(pos_z)):
                if gitter[pos_z[i], pos_s[i]] != leer and gitter[pos_z[i],
                pos_s[i]] != self.string[i]:
                    raise Exception
            for i in list(pos_z)+list(pos_s):
                if abs(i) != i:
                    raise Exception
```

Wenn die angegebene Richtung 0 ist, was einer horizontalen Ausrichtung entspricht, ist die Zeilenposition für jeden Buchstaben die Gleiche und die Spaltenposition erhöht sich für jeden weiteren Buchstaben um 1. Bei Richtung 1 (=vertikal) ist es genau umgekehrt. Bei Richtung 2, welche bedeutet, dass das Wort von oben links nach unten rechts diagonal gehen soll, wird mit einer Erhöhung des Buchstabenindexes um 1 sowohl der Zeilen- als auch der Spaltenwert um 1 erhöht. Und bei Richtung 3 schließlich, was für eine Ausrichtung von unten links nach oben rechts steht, verringert sich der Zeilenwert mit Erhöhung des Indexes, während sich der Spaltenwert erhöht.

```
        if richtung == 0:
            pos_z = self.len*[anfang[0]]
            pos_s = range(anfang[1], anfang[1]+self.len)
        elif richtung == 1:
            pos_z = range(anfang[0], anfang[0]+self.len)
            pos_s = self.len*[anfang[1]]
        elif richtung == 2:
            pos_z = range(anfang[0], anfang[0]+self.len)
            pos_s = range(anfang[1], anfang[1]+self.len)
        else:
            pos_z = range(anfang[0], anfang[0]-self.len, -1)
            pos_s = range(anfang[1], anfang[1]+self.len)
```

Nachdem die Positionen auf diese Weise festgelegt wurden, wird ihre Gültigkeit mit der vorher erstellten Funktion überprüft und der Rest der Methode wird nur ausgeführt, wenn das Wort eingefügt werden darf. Der String des Objekts wird dann an den Positionen im Gitter eingefügt und die Attribute „richtung“ und „anfang“ werden auf die anfangs als Argumente angegebenen Werte gesetzt. Die Zeilen- und

Spaltenpositionen werden für das Attribut „pos“ passend umgewandelt, sodass für jeden Index die entsprechende Zeilen- und Spaltenposition in einer Liste steht, die wiederrum ein Element der Positionsliste ist.

```
        ungültig(pos_z, pos_s)
        gitter[pos_z, pos_s] = list(self.string)
        self.richtung = richtung
        self.anfang = anfang
        self.pos = list([pos_z[i], pos_s[i]] for i in range(self.len))
```

Die Methode „verbinden“ ist dafür zuständig, das Wort mit einem anderen, bereits eingefügten Wort zu verbinden, indem die beiden Wörter sich an einer Stelle denselben Buchstaben teilen. Als Parameter hat sie den Index vom String des die Methode ausführenden Wortes, an dem die Verbindung erfolgen soll und die Position, an der sich ein Buchstabe geteilt werden soll. In einer for-Schleife werden Alle möglichen Ausrichtungen in zufälliger Reihenfolge durchgegangen, also 0 bis 1 bei Stufe 1 und 2 und ansonsten 0 bis 3.

```
    def verbinden(self, index, pos):
        ausrichtungen = [0,1]
        if stufe > 2:
            ausrichtungen = [0,1,2,3]
        random.shuffle(ausrichtungen)
        for r in ausrichtungen:
```

Für jede Ausrichtung wird erst einmal das Attribut „anfang“ auf eine Kopie des Parameters „pos“ gesetzt. Wenn die Richtung nicht 1 ist, die Spaltenpositionen also beim Einfügen nicht alle die gleiche wären, wird überprüft, ob der linke Teil des Wortes von der Verbindungsstelle aus nicht über den Rand hinausgehen würde. Wenn er das würde, wird ein Fehler hervorgerufen, ansonsten wird von der Anfangsposition des Wortes die Länge des linken Teils abgezogen, um sie entsprechend nach links zu verschieben. Wenn die Richtung 1 oder 2 ist, wird das auch für den von der Verbindungsstelle aus gesehen oberen Teil überprüft. Bei Richtung 3 muss allerdings die Länge des Wortteils nach dem Index der Verbindungsstelle abgezogen werden, weil der Index eines Buchstaben im String dann mit niedrigerer Zeilenposition zunehmen würde.

```
            self.anfang = pos.copy()
            if r in [0, 2, 3]:
                if self.anfang[1]-len(self.string[:index]) < 0:
                    raise Exception
                self.anfang[1] -= len(self.string[:index])
            if r in [1, 2]:
                if self.anfang[0]-len(self.string[:index]) < 0:
                    raise Exception
                self.anfang[0] -= len(self.string[:index])
            if r == 3:
                if self.anfang[0]-len(self.string[index+1:]) < 0:
                    raise Exception
                self.anfang[0] += len(self.string[:index])
```

Wenn die Verbindung nun mit der aktuell überprüften Ausrichtung möglich ist, wird die Methode „einfügen“ verwendet, um das Wort im Gitter mit dieser Ausrichtung und dem ermittelten Anfangspunkt einzufügen. Wenn dies keinen Fehler erzeugt, wird die Methode beendet, ansonsten läuft die Schleife für die nächste Ausrichtung weiter und es wird ein Fehler erzeugt, falls bei keiner Ausrichtung eine Verbindung möglich war. Weil in dieser Methode viele Fehler absichtlich erzeugt werden können, wird sie im Folgenden nur in try-except-Blöcken ausgeführt.

```
            try:
                self.einfügen(r, self.anfang)
                return
            except:
                pass
        raise Exception
```

Die Methode „poyByIndex“, die als Parameter einen Index im String des Wortes benötigt, gibt für diesen Index den Positionswert seines Buchstabens im Gitter aus, welcher nach dem Setzen des Wortes im Attribut „pos“ bei diesem Index gespeichert ist.

```
    def posByIndex(self, index):
        return self.pos[index]
```

Die letzte Methode der Klasse Wort ist „yx-Bereich“. Sie gibt eine Liste mit den minimalen und den maximalen Positionswerten zurück, an denen das Wort in seiner aktuellen Ausrichtung eingefügt werden kann, ohne über den Gitterrand hinauszugehen. Dafür werden erstmal

die maximalen Werte mit den Gittermaßen und die minimalen Werte mit 0 initialisiert.

```
    def yx_bereich(self):
        max = maße.copy()
        min = [0, 0]
```

Anschließend wird, wenn die Ausrichtung diagonal ist, die Länge des Strings von der maximalen Zeilenposition abgezogen bzw. die Länge minus 1 zur minimalen Zeilenposition hinzuaddiert. Dazu wird die Stringlänge von der maximalen Spaltenposition abgezogen. Bei Richtung 1 oder 2 wird dementsprechend die Länge des Strings von der maximalen Zeilen- bzw. Spaltenposition abgezogen und vom anderen maximalen Positionswert 1 subtrahiert, weil der höchste Index um 1 kleiner ist als die Anzahl der Zeilen/Spalten.

```
        if self.richtung in [2, 3]:
            if self.richtung == 2:
                max[0] -= self.len
            else:
                min[0] += self.len-1
                max[0] -= 1
            max[1] -= self.len
        else:
            max[abs(self.richtung-1)] -= self.len
            max[self.richtung] -= 1
        return [min, max]
```

Weitere Funktionen:

Außerhalb der Wort-Klasse habe ich noch weitere Funktionen definiert. Unter Anderem die Funktion „verbindungen“, welche nach Übergabe von zwei Strings eine Liste aus Indexpaaren ausgibt, an denen diese denselben Buchstaben haben.

```
def verbindungen(wort1, wort2):
    gleich = []
    for i in range(len(wort1)):
        for i2 in range(len(wort2)):
            if wort1[i] == wort2[i2]:
                gleich.append((i, i2))
    return gleich
```

Die Funktion „wörterMitPos“, welche als Parameter einen Positionswert braucht, geht alle Wort-Objekte durch und fügt sie zu einer am Ende ausgegebenen Liste hinzu, falls sie im Gitter an einer Stelle die übergebene Position haben.

```
def wörterMitPos(pos):
    obj = []
    for i in wörter_obj:
        if pos in i.pos:
            obj.append(i)
    return obj
```

Die Funktionen „verbindenTrue“ und „verbindenFalse“ werden später in der Schleife verwendet, in der alle Wörter und Fragmente in das Gitter eingefügt werden. Sie beide nehmen als Parameter einen den Index eines Wort-Objektes in der Liste „wörter_obj“ an, die später noch initialisiert wird. In der Funktion „verbindenTrue“ wird zuerst eine Liste erstellt aus allen bereits eingefügten Wörtern, die mindestens einen Buchstaben mit dem am übergebenen Index stehenden Wort gemeinsam haben.

```
def verbindenTrue(n):
    fertig = False
    möglich = list(w for w in wörter_obj if w.pos != None and sum(
        c in wörter_obj[n].string for c in w.string))
```

Dann werden in einer for-Schleife die Objekte der Wörter, mit denen sich das erste Wort verbinden kann, in zufälliger Reihenfolge durchgegangen und jedes Mal mit der Funktion „verbindungen“ die Verbindungsstellen zwischen dem ersten und dem zweiten Wort ermittelt. Sofern welche vorhanden sind, werden sie ebenfalls in zufälliger Reihenfolge durchgegangen und es wird versucht, die Wörter an dieser Stelle miteinander zu verbinden. Wenn das klappt, wird die aktuelle Schleife und auch die äußere Schleife abgebrochen und der Wahrheitswert True ausgegeben, ansonsten False.

```
    for wort2 in random.sample(möglich, len(möglich)):
        gleich = verbindungen(wörter_obj[n].string, wort2.string)
        if len(gleich):
            for i in random.sample(gleich, len(gleich)):
                try:
```

```
                    wörter_obj[n].verbinden(i[0], wort2.posByIndex(i[1]))
                    fertig = True
                    break
                except:
                    pass
        if fertig:
            break
    return fertig
```

In der Funktion „verbindenFalse“, welche ein Wort unabhängig im Gitter einfügt, wird zuerst eine Liste aus den möglichen Ausrichtungen in zufälliger Reihenfolge erstellt. Diese werden dann in einer for-Schleife nacheinander durchgegangen, jeweils der Bereich ermittelt, in dem das Wort eingefügt werden kann. Dann werden alle Zeilen- und Spaltenwerte in diesem Bereich gemischt und durchgegangen und falls das Einfügen an der aktuellen Position klappt, werden die Schleifen unterbrochen und es wird True ausgegeben, sonst False.

```
def verbindenFalse(n):
    if stufe > 2:
        richtungen = random.sample([0, 1, 2, 3], 4)
    else:
        richtungen = random.sample([0, 1], 2)
    fertig = False
    for r in richtungen:
        wörter_obj[n].richtung = r
        bereich = wörter_obj[n].yx_bereich()
        zeilen = random.sample(
            list(range(bereich[0][0], bereich[1][0]+1)), bereich[1][0]-bereich[0][0]+1)
        spalten = random.sample(
            list(range(bereich[0][1], bereich[1][1]+1)), bereich[1][1]-bereich[0][1]+1)
        for z in zeilen:
            for s in spalten:
                try:
                    wörter_obj[n].einfügen(r, [z, s])
                    fertig = True
                    break
                except:
                    pass
            if fertig:
                break
        if fertig:
            break
    return fertig
```

Die Funktion „reset“ kommt zum Einsatz, wenn beim Einfügen der Wörter eventuell ein Neustart erforderlich ist. Sie leert das Gitter,

erstellt neue Wort-Objekte, mischt deren Liste neu und hängt die selben Fragmente wie zuvor an.

```
def reset():
    gitter.fill(leer)
    random.shuffle(wörter)
    wörter_obj.clear()
    for wort in wörter:
        wörter_obj.append(Wort(True, wort))
    wörter_obj.extend(Fragmente)
```

Erstellung des Rätsels:

Nach der Definition dieser Funktionen beginnt der eigentliche Prozess der Erstellung des Rätsels. Es wird die bereits erwähnte Liste wörter_obj initialisiert, mit Objekten der Wörter in der Wörterliste gefüllt und gemischt. Für die spätere Verwendung wird noch die Funktion n_leer() definiert, die die Anzahl der leeren Felder im Gitter zählt und ausgibt. Dann folgt die Erstellung der Wortfragmente. Wenn die Schwierigkeit unter 4 liegt, werden zwischen der Anzahl der leeren Felder geteilt durch 2 und der geteilt durch 3 zufällige Buchstabengruppen zur Liste „Fragmente“ hinzugefügt, die jeweils 1 bis 3 Buchstaben lang sind.

```
Fragmente = []
if stufe < 4:
    for n in range(random.randint(round(n_leer()/3), round(n_leer()/2))):
        Fragmente.append(Wort(False, "".join(random.choice(alphabet)
                        for n in range(random.randint(1, 3)))))
```

Wenn die Stufe größer als 1 ist, also nicht alle Buchstabengruppen zufällig sein sollen, wird eine Variable F_Anzahl deklariert, die angibt, wie lang die Liste „Fragmente“ am Ende sein soll. Ist die Schwierigkeit kleiner als 4, Wird diese Liste um die Hälfte dezimiert, sodass später die Hälfte zufällige Buchstabengruppen sind und die andere den Zielwörtern ähnelnde Fragmente. Die gewünschte Länge der Liste ist dann ihre ursprüngliche, bei Stufe 4 ist es die Anzahl an Zielwörtern.

```
if stufe > 1:
    if stufe < 4:
        F_Anzahl = len(Fragmente)
        Fragmente = Fragmente[:round(len(Fragmente)/2)]
```

```
    else:
        F_Anzahl = len(wörter)
```

Solange nun die Länge der Fragmentliste kleiner ist als die gewünschte, werden die Strings der zu findenden Wörter durchgegangen und von jedem String zu einer Wahrscheinlichkeit von 50% ein Fragment erstellt. Dafür wird zuerst die Länge dieses Fragments festgelegt. Es soll mindestens zwei Buchstaben lang sein und maximal die Länge des aktuellen Wortes minus 2, um Duplikaten der zu suchenden Wörter vorzubeugen.

```
    w_strings = list(w.string for w in wörter_obj)
    while len(Fragmente) < F_Anzahl:
        for s in w_strings:
            if random.randint(0, 1):
                if len(s) >= 5:
                    f_länge = random.randint(2, len(s)-2)
                else:
                    f_länge = min(2, len(s))
```

Als nächstes wird ein zufälliger Startwert so festgelegt, dass auch der Index Startwert plus Fragmentlänge innerhalb des Wortes liegt. Dann wird ab diesem Startwert ein Teil des Wortes mit der entsprechenden Länge verwendet und mit einer Wahrscheinlichkeit von 2/3 durchgemischt. Wenn das Fragment sowohl rückwärts als auch vorwärts weder in der Fragmentliste, noch in der Wörterliste vorhanden ist, wird ein Wortobjekt davon mit dem Typ Fragment (False) an einer zufälligen Stelle in die Fragmentliste eingefügt und falls sie die Ziellänge erreicht hat, wird die for-Schleife abgebrochen.

```
                start = random.randint(0, len(s)-f_länge)
                fragment = s[start:start+f_länge]
                if random.randint(0,2):
                    fragment="".join(random.sample(fragment,len(fragment)))
                if not fragment in w_strings+Fragmente and not fragment[::-1] in
w_strings+Fragmente:
                    Fragmente.insert(random.randint(
                        0, len(Fragmente)), Wort(False, fragment))
                    if len(Fragmente) >= F_Anzahl:
                        break
```

Nun werden die verschiedenen Buchstaben ermittelt, die in der Gesamtheit aller Fragmente vorkommen. Wenn es weniger als 3 verschiedene Buchstaben gibt, wodurch die Wörter zu einfach zu finden wären, müssen noch einige zufällige Buchstabengruppen hinzugefügt werden. Dabei dürfen diese jedoch keine Buchstaben enthalten, die Teil eines Wortes sind, das außer diesem Buchstaben nur welche enthält, die bereits in den Fragmenten enthalten sind. Dann würden nämlich mit hoher Wahrscheinlichkeit zu viele Duplikate auftreten. Diese Buchstaben werden also zur Liste „raus“ hinzugefügt und schließlich wird, solange weniger als 3 verschiedene Buchstaben in der Gesamtheit der Fragmente vorkommen, eine zufällige Buchstabengruppe der Länge 1 bis 2 erstellt, die keinen der ausgeschlossenen Buchstaben enthält, und ein Objekt von ihr zu den Fragmenten hinzugefügt.

```
    b_enthalten = list(set("".join(list(f.string for f in Fragmente))))
    if len(b_enthalten) < 3:
        raus = []
        for w in wörter:
            for b in alphabet:
                if b in w and b not in b_enthalten:
                    if len(w)-sum(w.count(i) for i in b_enthalten) == w.count(b):
                        raus.append(b)
                        break
        while len(b_enthalten) < 3:
            f = "".join(random.choice(alphabet) for n in range(random.randint(1, 2)))
            while sum(i in raus for i in f):
                f = "".join(random.choice(alphabet) for n in range(random.randint(1, 2)))
            Fragmente.append(Wort(False, f))
            b_enthalten = list(set("".join(list(f.string for f in Fragmente))))
```

Die Fragment-Objekte werden dann zur Liste der Wörterobjekte hinzugefügt.

```
wörter_obj.extend(Fragmente)
```

Zum Einfügen der Wörter und Fragmente ins Gitter werden diese, solange kein Neustart erforderlich ist, durchgegangen und zuerst je nach Stufe zu einer Wahrscheinlichkeit von 1/3 bzw. 1/2 die Variable „verbinden“ auf True gesetzt.

```
gelungen = False
while not gelungen:
    neustart = False
    for n in range(len(wörter_obj)):
        if stufe > 2:
            verbinden = not bool(random.randint(0, 2))
        else:
            verbinden = bool(random.randint(0, 1))
```

Wenn das aktuelle Wort nicht das erste ist, das eingefügt wird, wird die der Variable „verbinden“ entsprechende Funktion ausgeführt. Wenn die Funktion erfolglos war, wird die entsprechend andere (verbindenTrue bzw. verbindenFalse) ausgeführt und wenn diese auch erfolglos war und das aktuelle Wort kein Fragment ist, da diese vernachlässigbar sind, wird ein Neustart durchgeführt.

```
        if n > 0:
            func = [verbindenFalse, verbindenTrue]
            fertig = func[int(verbinden)](n)
            if not fertig:
                fertig = func[int(not verbinden)](n)
                if not fertig and n < len(wörter):
                    reset()
                    neustart = True
                    break
```

Wenn das Wort als erstes eingefügt wird, wird es einfach mithilfe der Funktion „verbindenFalse“ unabhängig im Gitter eingefügt. Wenn nach dem Durchlaufen der for-Schleife kein Neustart erforderlich ist, wird die äußere while-Schleife abgebrochen.

```
        else:
            verbindenFalse(n)
    if not neustart:
        gelungen = True
```

Nach diesem Vorgang bleiben normalerweise noch Lücken im Gitter übrig, in denen immer noch das Platzhalterzeichen steht. Also wird so lange jeweils ein zufällig ausgewähltes Fragment aus der Liste eingefügt, bis es keine leeren Felder mehr gibt. Wenn dieses Einfügen 20-mal hintereinander fehlgeschlagen ist, was darauf hindeuten könnte, dass die Fragmente alle zu lang für die restlichen Lücken sind, werden die restlichen Lücken mit einzelnen Buchstaben aus den Fragmenten gefüllt.

```
f_counter = 0
while n_leer() > 0:
    index = random.randint(len(wörter), len(wörter_obj)-1)
    if not verbindenFalse(index):
        f_counter+=1
        if f_counter >= 20:
            for z in range(maße[0]):
                for s in range(maße[1]):
                    if gitter[z,s] == leer:
                        gitter[z,s]=random.choice(random.choice(Fragmente).string)
    else:
        f_counter = 0
```

Alle nicht initialisierten Fragmente werden jetzt aus der Liste wörter_obj entfernt, um das Auftreten eines Fehler bei der späteren Verwendung von „wörterMitPos“ zu vermeiden.

```
rem = []
for f in wörter_obj[len(wörter):]:
    if f.pos == None:
        rem.append(f)
for i in rem:
    wörter_obj.remove(i)
```

Dann wird eine Funktion „gitter_ausgeben“ definiert, die das Gitter Zeile für Zeile mit jeweils zwei Leerzeichen zwischen den einzelnen Buchstaben ausgibt. Diese wird direkt danach ausgeführt, um das fertige Gitter auszugeben.

```
def gitter_ausgeben():
    for zeile in gitter:
        print(" ".join(zeile))
gitter_ausgeben()
```

Zum Schluss wird noch angeboten, die gesuchten Wörter aufzudecken. Wenn der Benutzer das will, werden alle Buchstaben im Gitter, die keinem der gesuchten Wörter angehören, durch das Platzhalterzeichen ersetzt und das Gitter wird wieder ausgegeben. Zum Schluss wird eine while-Schleife unendlich lange durchlaufen, sodass der Benutzer sich die Ergebnisse durchlesen und ggf. kopieren kann, ohne dass sich das Konsolenfenster schließt.

```
input("\nZum Aufdecken der gesuchten Wörter Enter drücken. ")
for z in range(maße[0]):
    for s in range(maße[1]):
        if not sum(int(w.typ) for w in wörterMitPos([z,s])):
            gitter[z,s]=leer
gitter_ausgeben()

while True:
    pass
```

Beispiele

Zusätzlich zu den Beispieldateien habe ich noch die Textdatei „worte6.txt“ erstellt, die ein 30-mal-30-Gitter mit 50 Wörtern vorgibt. Um diese 50 zufälligen Wörter zu generieren, habe ich folgende Webseite verwendet: https://www.palabrasaleatorias.com/zufallige-worter.php

Außerdem habe ich ein kleines Hilfsprogramm geschrieben, um mir diese Wörter direkt Zeile für Zeile ausgeben zu lassen und sie in die Datei einzufügen:

```
Wörter="Siamese,Harfe,Fenster,Zahlen,Tentakel,Lupe,Radio,Lenker,Verbindung,Mantel,Eintauch
en,Klar,Farbe,\
Abspielen,Schlüsselbund,Prime,Teenager,Gazelle,Infektion,Rauch,Zoll,Charakter,Balance,Sold
at,Hof,Bienen,Schluchzen,Birne,Abschleppen,\
Frankreich,Staatsanwalt,Insel,Seebeben,Kasserolle,Population,Wurm,Hurrikan,Pose,Weit,\
Tanga,Cousin,Aufsteigen,Brüder,Vergessen,Schlagloch,Antworten,Asphalt,Urne,Briefe,Automati
sch"
Wörter=Wörter.split(",")
Wörter.sort()
print(len(Wörter))
print("\n".join(Wörter))
```

Im Folgenden zeige ich für jede Beispieldatei inklusive meiner eigenen jeweils für einen verschiedene Schwierigkeitsgrad ein vom Programm generiertes Buchstabenrätsel und seine aufgedeckte Version. Es hängt von den Einstellungen der den Text anzeigenden Software und der Schrift ab, ob das Gitter, wenn es Zeile für Zeile ausgegeben wird, sichtbar zu sehr gestreckt oder gestaucht erscheint, aber ich halte zwei Leerzeichen zwischen den Buchstaben für einen guten Mittelwert.

worte0.txt:

Stufe 1:

```
Zu findende Wörter:
VOR, RAD, EVA, TORF

C  E  E  G  I
V  W  R  Ü  Q
O  D  A  R  A
R  O  J  J  V
F  R  O  T  E

Zum Aufdecken der gesuchten Wörter Enter drücken.
.  .  .  .  .
V  .  .  .  .
O  D  A  R  A
R  .  .  .  V
```

worte1.txt:

Stufe 2:

```
Zu findende Wörter:
EIN, DA, ER, INFO, DU, UND

N  Y  D  N  U  R
U  Z  D  A  N  I
I  Q  U  N  U  X
N  I  N  Ö  K  E
```

F E I N R M

O R M I N U

Zum Aufdecken der gesuchten Wörter Enter drücken.

. . D N U .

. . D A . .

I . U . . .

N . . . . .

F E I N . .

O R . . . .

worte2.txt:

Stufe 3:

Zu findende Wörter:
MAUS, TASTATUR, BILDSCHIRM, FESTPLATTE, USB, COMPUTER

C P Ü Ä Z O G B Z S

O T Ä P Z M A U S U

M A Z R S P C B U U

P S U N R T B E Ü J

U A M S W G A U G P

T A M D A J Z T P P

E A U Z Ä T Y Ä U M

R U R U T A T S A T

F E S T P L A T T E

B I L D S C H I R M

Zum Aufdecken der gesuchten Wörter Enter drücken.

C . . . . . B . .

O . . . . M A U S .

M . . . . . U

P . . . . . .

U . . . . . .

T . . . . . .

E . . . . . .

R . R U T A T S A T

```
F  E  S  T  P  L  A  T  T  E
B  I  L  D  S  C  H  I  R  M
```

worte3.txt:

Stufe 4:

Zu findende Wörter:

INTUITION, INFEKTION, MONOGRAMM, REVOLUTION, KONJUNKTUR, CHRONIK, EMISSION, DEKORATION, LEGITIMATION, EMPATHIE, REFERAT, VERS

```
N U R E U A T H I E U N I N I T N A T R T R E I
A I A T R T T I T R R M I T I A T O U I U A N R
T M T E A H A H T E O G T N E I U N I N I U T I
R I R I R I T K I N O R H C K E M I S S U A R R
I T T I A E R T O E A T R I O M N R T I S R T S
N I R T T N A G I T I T N T R I I U E I T I S R
M A I E R I R T A R E F E R A S U T R E I I M R
A T I E T A T R R E E K U U T S I K O G M T O E
T N R I M U M R E K R E O G R I T N G E R D N G
R R M M R R R I T V E K O R A T I U R O M E N I
U A T H I E I I T R O L N E A A I J O G I K O R
S R E V G N O N T I E L I R U T T N A G T O U G
U R U O E N N A T G A H U I E R N O T E I R O R
I R R I N A I N I U T T R T E U R K R E A A G G
U T H G I T T T I A I A T R I U I R U R T T I T
A T O N I I I R T I T T N E E O T O I U R I G A
A O G M N M E M P A T H I E M A N G T N I O E T
I T T I A I E M I S S N R O R I T I I N U N M R
N I H T N I T U G A T R A O N T S R I T R E I N
N I I I N E K O R A T R K T N E I S T I O O S I
U O O G E K O R A T R E E O R I T T N I G G S T
N R E M I T I A T R R E O G G I T U I T I I T N
```

Zum Aufdecken der gesuchten Wörter Enter drücken.

```
. . . . . N . . . . . .
. . . . . M . . . . . O . . . . . .
. . . . . O . . . . . I . . . . . .
. . . . . K I N O R H C . . . . . S . . . . .
```

```
. . . . . O . . . . I . . . R . . S . . .
. . . . . G . . . . N . . . . U . . . I . .
. . . . . R T A R E F E R . . . T . . . . M .
. . . . . A . R . . E . . . . . . K . . . . . E
. . . . M . . . E K . . . . . . . . N . . . D . .
. . . M . . . . . T V . . . . . . . . U . . . E . .
. . . . . I I . . O L . . . . . . J . . . K . .
S R E V . . O N . . E L . . . . . N . . . O . .
. . . . . N . . T G . . U . . . . O . . . R . .
. . . . . I U . . . T . . . K . . . A . .
. . . . . T . . I . . . I . . . . . T . .
. . . . . I . . . . T . . . O . . . . . I . .
. . . . . M E M P A T H I E . . N . . . . O . .
. . . . A . . . . . . . . . O . . . . . . . . N . .
. . . T . . . . . . . . . . N . . . . . . . . . .
. . I . . . . . . . . . . . . . . . . . . . . . .
. O . . . . . . . . . . . . . . . . . . . . . . .
N . . . . . . . . . . . . . . . . . . . . . . . .
```

worte4.txt:

Stufe 4:

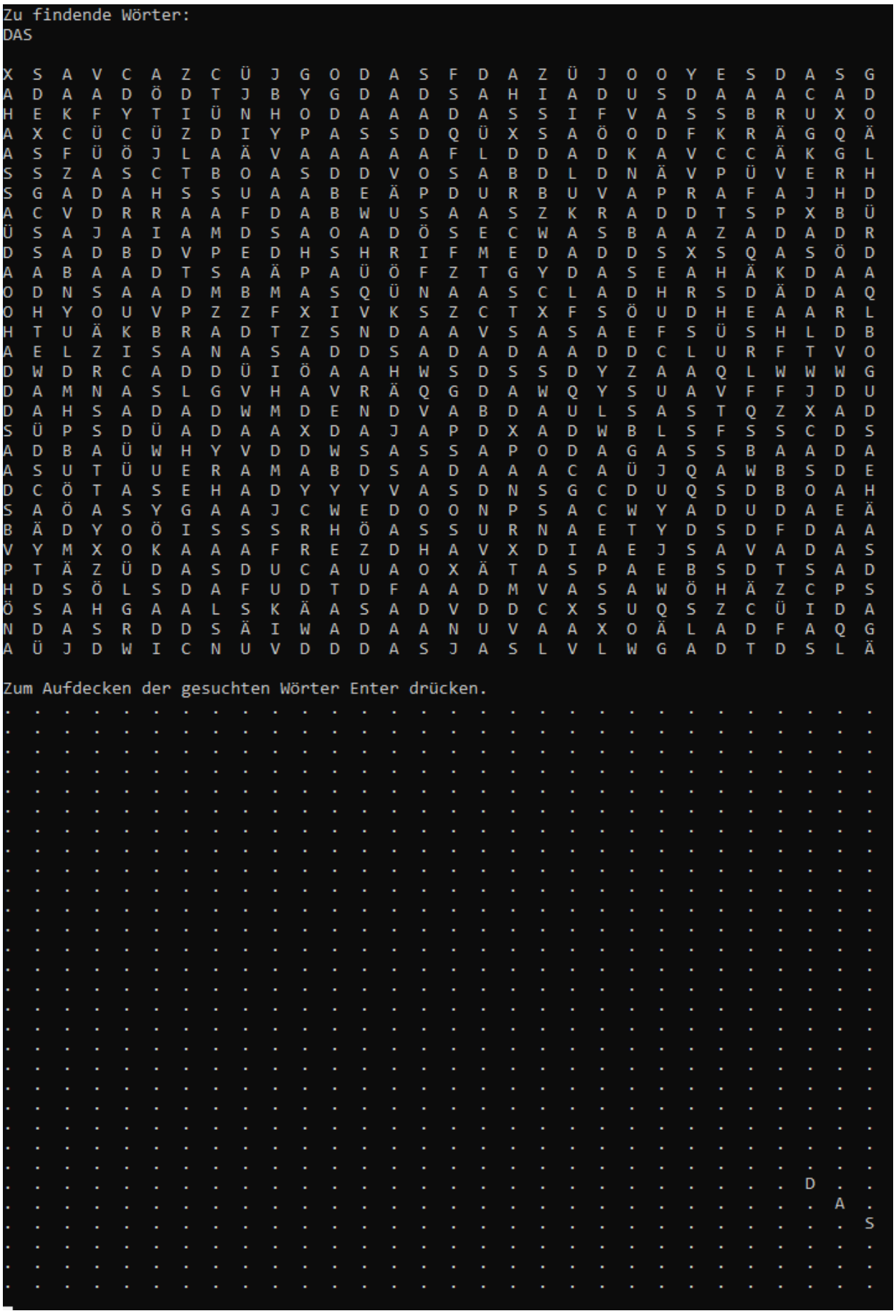
Zu findende Wörter:

ABRUFDATUM, ABSATZ, ACHTUNG, ALLMUSIC, ARCHIV, ARCHIVBOT, ARCHIVIERUNG, AUS, AUT, AUTOARCHIV, BABEL, BAUSTELLE, BBKL, BEGRIFFSKLÄRUNG, BEGRIFFSKLÄRUNGSHINWEIS, BEL, BELEGE, BENUTZER, BGR, BIBRECORD, BOOLAND, CAN, CENTER, CHARTS, COL, COMMONS, COMMONSCAT, COORDINATE, COORDINATEMAP, DDB, DEU, DISKUSSIONSSEITE, DOI, ERLEDIGT, FARBLEGENDE, FILM, FN, FNZ, FOLGENLEISTE, FRA, FUSSBALLDATEN, GEOQUELLE, GER, GNIS, HÖHE, IMDB, INFO, INFORMATION, INTERNETQUELLE, IPA, KALENDERSTIL, KASTEN, KATEGORIEGRAPH, LANG, LITERATUR, LIZENZUMSTELLUNG, MEDAILLENSPIEGEL, MULTILINGUAL, MUSIKCHARTS, NAVFRAME, NAVIGATIONSLEISTE, NOCOMMONS, ÖSTERREICHBEZOGEN, PERSONENLEISTE, PING, POSITIONSKARTE, PRO, SMILEY, SORT, TAXOBOX, TEXT, WAPPENRECHT, WEBARCHIV, WIKIDATA, WIKISOURCE, WIKTIONARY, ZITATION


U E L L T N S B P K I L E N E B N E T S A K V I H C R A O T U A
Z A L A O B K I E S A G P A S D E G I R F E S G K I U L F Ä R L
L E R M L G N B Z T Z T B V B F N G N S M O R E I V I H C R O D
N T M K I G N R T R I E E I A B F E O Z U C L X O B O X A T B L
S C I L S I S E T A T Z R G B E R I G G T M I G N O M S O M I N
O E I C G M N C I G N T A A O C G B A E A M E L E P A E K T G Z
L G A E I L O O A N B E U T H R N B T E D O G K C N R N E E G K
R O U L E N M R Z A G S G I F A I R I L F N E B A E L R O D R N
E E E L N E M D F L T B V O H P R E T R U N E B Z R A E E A B Z
D Y O U T Z O I L E D B G N Z R I C G C R Z N T B T K Z I Z N D
S R N E E T C A L R A N U S B E C F H R B O U S U A N O O S T Z
I H V A I R C L A E R S O L B E B E S U A N G R F G D B A L T M
E T T I A Z E I T N C R A E E D L H G B E P S N D F M T A U E E
W G G F S E N G A U H R E I L F R L C B D S H O L E A N E D R R
N T R N B O E U S L I A K S L N W A E I G U B V I H C R A Z O O
I A W A M L I F T E V P T T I I A R A U E N O I T A T I Z C T M
H B T I R S P I R T I I G E K I M D B S Q R A N N A L L V R O R
S E A L K C Z A A L E N A T E M A E T A R T R G A L I I L S U Z
G R C T T I A Z B R R F I S F F G R H R O B E E E T H L M E C E
N N S O A R D O D E U O L W K R A U A Ö G A N N T C E O G N A G
U I N G R I F A L Q N A A B I H I H R C H R S O R S N M N U R A
R L O R P L H L T A G P W F C E F G C B E P I A N E Ö B A L L C
Ä E M E L L E R R A P E F K G N B I Y T I G R F E S T R A E I D
L N M S U U K Y C E B S I E L L U E N E S B E L E G E N O T Z D
K N O A Q O L G N A K S N P T L L E G Y U E T E E Z G F I L E I
S S C O R A N R R L U D C O I R C E G A R A L X N E U B D R N S
F I E G N U E C Ä M E R R S E E L L U E L O R L E S D D K E Z K
F G R C T C H R B K H R C I D L S I E S F L A C S T P N A Q U U
I S I H H I U A T I E O O T D B L C L U A R M B H C O L K C M S
R U C T V N U O E G O N L I B D R U Z E R C A U L E Y Z O A S S
G A Z B G A B S T R A N E O D U P T E C B L R C S D A O N L T I
E N N S O V D D D B A S L N O A I L H A L B D L E I R O L K E O
B E G U I F D I K R A P L S N Z E A B D E D G N T D C T C E L N
U D E H T R N L T B D B I K O B R E A C G S Z T I O Z R L K L S
B A C G A A T A S T O K L A L T L T I L E O L N M E G T K S U S
C R D B T M S A U O I E P R S G E Z T U N R A M F N C R S T N E
A R D E I E T B L W E A L T A N D O P A D T O R I T A S C U G I
G C M G F Z R A E G G I Z E T S I E L N E N O S R E P I B A L T
B A L A U G N I L I T L U M Y P G L I T S R E D N E L A K B P E
P G E E L D I N O G A T I N O I T A M R O F N I N Z I T G E E N
Zum Aufdecken der gesuchten Wörter Enter drücken.
. . . . . B P K . . . N . . N E T S A K V I H C R A O T U A
. . . . . I . . A . . A . . . . . F . . . .
. . . . . N B . . . T . V . . . . . M O . . . .
. . . . . G . R . . . . E I . . . . . U . L X O B O X A T . L
. . . . . S E . . . . . G B . . . . . T . . G . . . . . I .
. . . . . M N C . G N . . A O . . . . . A . . L E . . . . T . .
. . . . I . O O . N . E U T . R . . . . D . . K . N R . E . . .
. . U L . . M R . A . S G I . . I . . . F . . B . E L R . . . .
. E E . . . M D . L T . . O . . . E . . U . . B Z . A E . . . .
D Y . . . . O . . E . . . N Z . . . G . R . . T . T . . I . . .
S . . . . . C . L . A . . S . E . . . R B . U . U . . O . S . .
I . . . . . L . . R . . L . E B . S U A N . R . . D . . . T M
E . . . . . E . . . C . . E . . L H . . E P . . . . . E E
W . . F . . . . . H . . I . F . L C B . . H . . . . N . D . .
N . R . . . . . I . . S . N W . E I . . . . . A . . .
I A W . M L I F . . V . . T . I . . . U E N O I T A T I Z C . .
H . T I . . . . . I I . E K I M D B S Q R . . . . L . V . . .
S . A . K . . . . . E N . T . . . E T . . T R . . L . I . . . .
G . C . . I . . . . R F I . . . G R H . . . E E E . H . . . . .
N . S . . . D . . E U O . W . R A . . Ö . . . N T C . . . . . .
U . N . . . . A L . N . A . I H . . R . H R S . R S . . . . . .
R . O R P . . L T A G P W F C . . G . . E P . A . E Ö . . . L .
Ä . M . . . E . R A P E F K . . B . . T I . . . . . T . . . I .
L . M . . U . Y . E B S I . . . . . N E . B E L E G E N . . Z D
K . O . Q . . G N A K S . P . . . E G . . . T . . . . F I . E I
S . C O . . N R R L U . . O . . C E . A . . . X . . U . . . N S
F . E . . U E C Ä M . . . S . . L . . . L . . . E S . . . . Z K
F G . . T C H R . . . R C I . . . . E . F L . . S T . . . . U U
I . . H H I U . T . E O . T . . . C . . A . M B . C O L . C M S
R . C T V N . O . G O . . I . . R . . . R . A U . . . . O . S S
G A . . G A B . . R . . . O . U . . . C B L . . S . . O N . T I
E . . . . V D . D . A . . N O . . L H A L . . . . I R O . . E O
B . . . I F D I . . A P . S . . E A B D E . . . . D C . . . L N
. . . H . R N . . B . B I K . B R E A . G S Z . I O . . . . L S
. . C . . A . . S . O K . A . T L T . . E O . N M . G . . . U S
. R . . T M . A . O I . . R S . E . . . N R A M F N . . . T N E
A . . E . E T . L W . . . T . N D . . . D T O . I . . . . U G I
. . M . . Z . A . . . . . E T S I E L N E N O S R E P . . A . T
. A L A U G N I L I T L U M . . G L I T S R E D N E L A K . . E
P . . . . D . . . . N O I T A M R O F N I . . . .

worte5.txt:

Stufe 3:



worte6.txt:

Stufe 2:

Zu findende Wörter:

ABSCHLEPPEN, ABSPIELEN, ANTWORTEN, ASPHALT, AUFSTEIGEN, AUTOMATISCH, BALANCE, BIENEN, BIRNE, BRIEFE, BRÜDER, CHARAKTER, COUSIN, EINTAUCHEN, FARBE, FENSTER, FRANKREICH, GAZELLE, HARFE, HOF, HURRIKAN, INFEKTION, INSEL, KASSEROLLE, KLAR, LENKER, LUPE, MANTEL, POPULATION, POSE, PRIME, RADIO, RAUCH, SCHLAGLOCH, SCHLUCHZEN, SCHLÜSSELBUND, SEEBEBEN, SIAMESE, SOLDAT, STAATSANWALT, TANGA, TEENAGER, TENTAKEL, URNE, VERBINDUNG, VERGESSEN, WEIT, WURM, ZAHLEN, ZOLL

Quellcode

Hier ist der unkommentierte Quellcode:

```
import numpy as np
import os
import random
import codecs
from string import ascii_uppercase
alphabet = list(ascii_uppercase)+["Ä", "Ö", "Ü"]

def abfrage_zahl(frage, fehler, fehler_antwort):
    while True:
        try:
            ausgabe = int(input(frage))
            if eval(fehler):
                raise Exception
            break
        except:
            print(fehler_antwort+"\n")
    return ausgabe

leer = "."
n_Dateien = len(os.listdir("Beispieldateien"))
datei_nummer = abfrage_zahl("Welche Textdatei soll verwendet werden? ", "not ausgabe in
range(0, n_Dateien)",
                            "Bitte gebe eine Zahl von 0 bis "+str(n_Dateien-1)+" ein.")

print("\nSchwierigkeitsgrade:")
print("Stufe 1: Ohne diagonale Ausrichtung + Wörter einfach zu finden")
print("Stufe 2: Ohne diagonale Ausrichtung + einige wiederholte Fragmente")
print("Stufe 3: Alle Richtungen + einige wiederholte Fragmente + Verbindungen
unwahrscheinlicher")
print("Stufe 4: Alle Richtungen + sehr viele wiederholte Fragmente + Verbindungen
unwahrscheinlicher\n")

stufe = abfrage_zahl("Was ist der gewünschte Schwierigkeitsgrad (Zahl von 1 bis 4)? ",
                     "not ausgabe in range(1, 5)", "Bitte gebe eine Zahl von 1 bis 4
ein.")

with codecs.open("Beispieldateien\\"+os.listdir("Beispieldateien")[datei_nummer], "r",
"utf-8") as file:
    Zeilen = file.readlines()
    for n in range(len(Zeilen)):
        Zeilen[n] = Zeilen[n].strip()
    maße = Zeilen[0].split(" ")
    maße = [int(maße[0]), int(maße[1])]
    gitter = np.empty(maße, str)
    gitter.fill(leer)
    n_wörter = int(Zeilen[1])
    del Zeilen[:2]
```

```
    wörter = list(wort.upper() for wort in Zeilen)
    print("\nZu findende Wörter:")
    print(", ".join(wörter)+"\n")

class Wort():
    def __init__(self, typ, string):
        self.typ = typ
        self.string = string
        self.string_echt = string
        self.len = len(self.string)
        if random.randint(0, 1):
            self.string = self.string[::-1]
        self.richtung = None
        self.anfang = None
        self.pos = None

    def einfügen(self, richtung, anfang):
        def ungültig(pos_z, pos_s):
            for i in range(len(pos_z)):
                if gitter[pos_z[i], pos_s[i]] != leer and gitter[pos_z[i], pos_s[i]] !=
self.string[i]:
                    raise Exception
            for i in list(pos_z)+list(pos_s):
                if abs(i) != i:
                    raise Exception
        if richtung == 0:
            pos_z = self.len*[anfang[0]]
            pos_s = range(anfang[1], anfang[1]+self.len)
        elif richtung == 1:
            pos_z = range(anfang[0], anfang[0]+self.len)
            pos_s = self.len*[anfang[1]]
        elif richtung == 2:
            pos_z = range(anfang[0], anfang[0]+self.len)
            pos_s = range(anfang[1], anfang[1]+self.len)
        else:
            pos_z = range(anfang[0], anfang[0]-self.len, -1)
            pos_s = range(anfang[1], anfang[1]+self.len)
        ungültig(pos_z, pos_s)
        gitter[pos_z, pos_s] = list(self.string)
        self.richtung = richtung
        self.anfang = anfang
        self.pos = list([pos_z[i], pos_s[i]] for i in range(self.len))

    def verbinden(self, index, pos):
        ausrichtungen = [0,1]
        if stufe > 2:
            ausrichtungen = [0,1,2,3]
        random.shuffle(ausrichtungen)
        for r in ausrichtungen:
            self.anfang = pos.copy()
            if r in [0, 2, 3]:
                if self.anfang[1]-len(self.string[:index]) < 0:
```

```
                    raise Exception
                self.anfang[1] -= len(self.string[:index])
            if r in [1, 2]:
                if self.anfang[0]-len(self.string[:index]) < 0:
                    raise Exception
                self.anfang[0] -= len(self.string[:index])
            if r == 3:
                if self.anfang[0]-len(self.string[index+1:]) < 0:
                    raise Exception
                self.anfang[0] += len(self.string[:index])
            try:
                self.einfügen(r, self.anfang)
                return
            except:
                pass
        raise Exception

    def posByIndex(self, index):
        return self.pos[index]

    def yx_bereich(self):
        max = maße.copy()
        min = [0, 0]
        if self.richtung in [2, 3]:
            if self.richtung == 2:
                max[0] -= self.len
            else:
                min[0] += self.len-1
                max[0] -= 1
            max[1] -= self.len
        else:
            max[abs(self.richtung-1)] -= self.len
            max[self.richtung] -= 1
        return [min, max]

def verbindungen(wort1, wort2):
    gleich = []
    for i in range(len(wort1)):
        for i2 in range(len(wort2)):
            if wort1[i] == wort2[i2]:
                gleich.append((i, i2))
    return gleich

def wörterMitPos(pos):
    obj = []
    for i in wörter_obj:
        if pos in i.pos:
            obj.append(i)
    return obj

def verbindenTrue(n):
    fertig = False
```

```
    möglich = list(w for w in wörter_obj if w.pos != None and sum(
        c in wörter_obj[n].string for c in w.string))
    for wort2 in random.sample(möglich, len(möglich)):
        gleich = verbindungen(wörter_obj[n].string, wort2.string)
        if len(gleich):
            for i in random.sample(gleich, len(gleich)):
                try:
                    wörter_obj[n].verbinden(i[0], wort2.posByIndex(i[1]))
                    fertig = True
                    break
                except:
                    pass
        if fertig:
            break
    return fertig

def verbindenFalse(n):
    if stufe > 2:
        richtungen = random.sample([0, 1, 2, 3], 4)
    else:
        richtungen = random.sample([0, 1], 2)
    fertig = False
    for r in richtungen:
        wörter_obj[n].richtung = r
        bereich = wörter_obj[n].yx_bereich()
        zeilen = random.sample(
            list(range(bereich[0][0], bereich[1][0]+1)), bereich[1][0]-bereich[0][0]+1)
        spalten = random.sample(
            list(range(bereich[0][1], bereich[1][1]+1)), bereich[1][1]-bereich[0][1]+1)
        for z in zeilen:
            for s in spalten:
                try:
                    wörter_obj[n].einfügen(r, [z, s])
                    fertig = True
                    break
                except:
                    pass
            if fertig:
                break
        if fertig:
            break
    return fertig

def reset():
    gitter.fill(leer)
    wörter_obj.clear()
    for wort in wörter:
        wörter_obj.append(Wort(True, wort))
    random.shuffle(wörter_obj)
    wörter_obj.extend(Fragmente)

wörter_obj = []
```

```
for wort in wörter:
    wörter_obj.append(Wort(True, wort))
random.shuffle(wörter_obj)

def n_leer(): return sum(sum(s == leer for s in z) for z in gitter)

Fragmente = []
if stufe < 4:
    for n in range(random.randint(round(n_leer()/3), round(n_leer()/2))):
        Fragmente.append(Wort(False, "".join(random.choice(alphabet)
                        for n in range(random.randint(1, 3)))))
if stufe > 1:
    if stufe < 4:
        F_Anzahl = len(Fragmente)
        Fragmente = Fragmente[:round(len(Fragmente)/2)]
    else:
        F_Anzahl = len(wörter)
    w_strings = list(w.string for w in wörter_obj)
    while len(Fragmente) < F_Anzahl:
        for s in w_strings:
            if random.randint(0, 1):
                if len(s) >= 5:
                    f_länge = random.randint(2, len(s)-2)
                else:
                    f_länge = min(2, len(s))
                start = random.randint(0, len(s)-f_länge)
                fragment = s[start:start+f_länge]
                if random.randint(0,2):
                    fragment="".join(random.sample(fragment,len(fragment)))
                if not fragment in w_strings+Fragmente and not fragment[::-1] in
w_strings+Fragmente:
                    Fragmente.insert(random.randint(
                        0, len(Fragmente)), Wort(False, fragment))
                    if len(Fragmente) >= F_Anzahl:
                        break
    b_enthalten = list(set("".join(list(f.string for f in Fragmente))))
    if len(b_enthalten) < 3:
        raus = []
        for w in wörter:
            for b in alphabet:
                if b in w and b not in b_enthalten:
                    if len(w)-sum(w.count(i) for i in b_enthalten) == w.count(b):
                        raus.append(b)
                        break
        while len(b_enthalten) < 3:
            f = "".join(random.choice(alphabet) for n in range(random.randint(1, 2)))
            while sum(i in raus for i in f):
                f = "".join(random.choice(alphabet) for n in range(random.randint(1, 2)))
            Fragmente.append(Wort(False, f))
            b_enthalten = list(set("".join(list(f.string for f in Fragmente))))
wörter_obj.extend(Fragmente)
```

```
gelungen = False
while not gelungen:
    neustart = False
    for n in range(len(wörter_obj)):
        if stufe > 2:
            verbinden = not bool(random.randint(0, 2))
        else:
            verbinden = bool(random.randint(0, 1))
        if n > 0:
            func = [verbindenFalse, verbindenTrue]
            fertig = func[int(verbinden)](n)
            if not fertig:
                fertig = func[int(not verbinden)](n)
                if not fertig and n < len(wörter):
                    reset()
                    neustart = True
                    break
        else:
           verbindenFalse(n)
    if not neustart:
        gelungen = True

f_counter = 0
while n_leer() > 0:
    index = random.randint(len(wörter), len(wörter_obj)-1)
    if not verbindenFalse(index):
        f_counter+=1
        if f_counter >= 20:
            for z in range(maße[0]):
                for s in range(maße[1]):
                    if gitter[z,s] == leer:
                        gitter[z,s]=random.choice(random.choice(Fragmente).string)
    else:
        f_counter = 0

rem = []
for f in wörter_obj[len(wörter):]:
    if f.pos == None:
        rem.append(f)
for i in rem:
    wörter_obj.remove(i)

def gitter_ausgeben():
    for zeile in gitter:
        print("  ".join(zeile))
gitter_ausgeben()

input("\nZum Aufdecken der gesuchten Wörter Enter drücken. ")
for z in range(maße[0]):
    for s in range(maße[1]):
        if not sum(int(w.typ) for w in wörterMitPos([z,s])):
            gitter[z,s]=leer
```

```
gitter_ausgeben()

while True:
    pass
```