Aufgabe 3: Wortsuche

Team-ID: 00861

Team: Team YEET

Bearbeiter/-innen dieser Aufgabe:
Robin Schupp & Leo Kling

18. November 2021

Inhaltsverzeichnis

Lösungsidee **1**

Umsetzung **2**

Beispiele **3**

Beispiel 0 3

Beispiel 1 4

Beispiel 4 5

Quellcode **6**

Überprüfen einer möglichen Position 6

Auffüllen des Spielfeldes 7

Lösungsidee

Um Buchstabenrechtecke in drei verschiedenen Schwierigkeitsstufen zu generieren, haben wir uns zuerst Eigenschaften überlegt, die es schwieriger machen, ein Wort zu finden.

So sind wir auf die Schwierigkeitsstufen “einfach”, “mittel” und “schwer” gekommen. Jede dieser Stufen fügt eine oder mehrere Schwierigkeiten zur vorherigen Stufe hinzu.

Der Schwierigkeitsgrad “einfach” beinhaltet nur Wörter in horizontaler und vertikaler Ausrichtung, während “mittel” auch diagonale Wörter erlaubt. Bei beiden Schwierigkeitsgraden sind im Hintergrund zufällige Buchstaben.

“Schwer” wird um zwei Eigenschaften erweitert: Wörter können auch in umgekehrter Reihenfolge erscheinen und im Hintergrund werden Buchstaben angezeigt, die in den gesuchten Wörtern enthalten sind. Dadurch heben sich die gesuchten Wörter nicht mehr so deutlich vom Hintergrund ab.

Umsetzung

Das “Spielfeld” können wir relativ simpel in einem zweidimensionalen Array darstellen. Wir beginnen damit, dieses einfach mit Leerzeichen zu füllen. Basierend auf dem Schwierigkeitsgrad fangen wir nun an, Wörter in das Gitter einzufügen.

Dafür suchen wir uns eine zufällige Position und eine Rotation des Wortes aus und überprüfen, ob dieses an diese Position und mit dieser Ausrichtung Platz findet. Das wiederholen wir solange, bis wir einen Ort und eine Rotation gefunden haben, an dem das Wort eingefügt werden kann und tun dies im nächsten Schritt. All das wird für jedes Wort durchgeführt das später zu suchen sein soll.

Das Alphabet im Hintergrund wird wie oben erwähnt abhängig vom Schwierigkeitsgrad generiert. Für den schwierigen Spielmodus wird dafür die Liste an Wörtern in einen String konvertiert und mit zufälligen Buchstaben aufgefüllt, bis er eine Länge von 26 Buchstaben hat.
Die zusätzlichen Buchstaben werden angehängt, um eine gleichmäßigere Verteilung zu gewährleisten. Für die anderen beiden Schwierigkeitsgrade ist einfach ein String mit Buchstaben von A bis Z vordefiniert.

Zuletzt werden alle Leerzeichen im Array bzw. Spielfeld mit zufälligen Buchstaben des zuvor generierten Alphabets ersetzt.

Beispiele

Beispiel 0

```
R A D Y Z
K E V U P
Y V O H F
V A R P L
T O R F F

E T O R F
V P V J H
A R C O N
H A A G R
M X H D A

D N Z A E
T D A R A
V O R D E
R O R L V
T V R F A
--------------------
Gesuchte Wörter:
Vor
Rad
Eva
Torf
```

Bei den gesuchten Wörtern “Vor”, “Rad”, “Eva” und “Torf” und einer Größe von 5x5, sieht die Ausgabe des Programms wie links abgebildet aus.

Die Schwierigkeitsstufe nimmt dabei von oben nach unten zu.
Die Schwierigkeit wird besonders beim letzten Suchbild deutlich, in welchem einem (im Gegensatz zu den vorherigen) die Wörter nicht sofort ins Auge springen, da Buchstaben wie das R oder E häufiger vorkommen.

Beispiel 1

```
N R E B U N
B E O I N A
K I I N D B
G N P F N D
E R M O C U
A E D A V I

D E D N U B
A I I O W X
E N U N D T
R A I N F O
T D U C F L
S E L P O N

U U E N E O
D N D U E R
U N D A Q S
N S I N U E
D N E E R R
S I N F O Q
--------------------
Gesuchte Wörter:
Ein
Da
Er
Info
Du
Und
```

Dieses Beispiel ist ähnlich zu Beispiel 0, allerdings zeigt es den einzigen Schönheitsfehler des Programms auf.

Im dritten Rätsel lassen sich einige Wörter (“Er”, “Da” und “Du”) mehrfach finden. Natürlich gibt es auch Rätsel, in denen dies bewusst der Fall ist, und natürlich könnten wir auch behaupten, dass es das Rätsel nur schwerer macht (Nach dem Motto: It’s not a bug, it’s a feature) aber wir wollten es erwähnen, da es eigentlich nicht gedacht war.

Dieses Problem besteht, da die leeren Stellen im Gitter mit zufälligen Buchstaben aufgefüllt werden, ohne zu überprüfen, ob dadurch neue Wörter entstehen.
Diese Überprüfung konnte aus zeitlichen Gründen nicht mehr implementiert werden.

Beispiel 4

```
A H S U S T A U A Z B G G T Z A Z U U E U B N A A T S A U T D B
A L C U B A B E W I K T I O N A R Y E R R L T R B D L T T E R A
W R E T N E C D H D B X F S U Z N F G D H L R H L T B A A X W A
I T Z U A A L B D Ö A A O U C A H F U H L U E A B A B A C T E A
K B B R H B A T A F H A B B A U M R L Z A L T D A A D A T U B H
I U A Z B G R H L U D A G E O U T A A K A T U T I A H D M A A T
D A R A A B U U Z H H A G M L X T S M G B A D G U G F N U H R Z
A N E T S A K T A U N T A R A A A D A T N B L A L A T A L A C A
T U I N F O R M A T I O N M N R C T T B T A L H A B T L T S H A
A A N N C A N F A T A U F U A O A O U L L H Z L T L A O I A I B
B E L E G E U T C I S U M L L A I A O S N O M M O C S O L G V A
C U T S U N A A L A N G L T M U A T A R M S A D C C T B I M U U
B B I B R E C O R D Ö T L T N T D T A L D F U Z A H R T N T U S
I T U B H L A S L U T S Z T A S D C G T B I S L A A A O G U A T
W O G T T A U K C I B T T T A C T A L B I D N U A R H B U C A E
I T D A A C L A L A Z E Z E D F S H B B F Z M A U T C V A T E L
K F A F L T N T C R T E G R R F F N T N S C A I T S K I L B L L
I C A U O T T E A C A B N R U R R M O G L A A U D E I H C A L E
S B E L F L C G M H G H P Z I T E G L M U A B A T Z S C F G E G
O H A G N M R O L I U N B E U F A I R S M S S B A C U R N N U G
U A S F I A B R B V U L T A R M F R C Z A O A S T M M A H U Q A
R G G L B U A I A I T D B S B S S S E H U S C A A L A A T T T U
C A N U A T L E T E R S F A C R O T K T B A D T S I B T N H E L
E V I H C R A G F R O N M L A M U N E L I E T Z U F L B T C N U
B M R M E U Z R B U S H S I B F U F E L Ä L Z U T U A A M A R F
Z A A U L H A A A N U T C C L B A H D N L R A O F T D B E A E B
T M A A L U U P B G S G N I S E D R F A L U U A G A U H D U T C
M C L A E A L H A U D C T D Z T Y L N L T E N N S E A D A U N B
A B F L U D E T R A K S N O I T I S O P G U I G G A N F I C I E
L P A A Q C O O R D I N A T E M A P A T T C M S A A B O L L R D
L T I T O B A L U C T U U A U A C C A A L A O D T B U L L A G N
R M F D E T S N O M M O C O N Z C M U U U U U L A E M G E S E E
A A S N G A C H A T H C E R N E P P A W A A U C A H H E N A R G
U P I N G D B G N A B E N U T Z E R C T A B Z T L A L N S U U E
N R S S I E W N I H S G N U R Ä L K S F F I R G E B T L P L C L
U U C Z B A T U S A T U D L I T S R E D N E L A K P G E I A T B
F N F U C E T S I E L S N O I T A G I V A N L M A R M I E U A R
U F U S S B A L L D A T E N L D B L U A L A U M T O U S G A A A
A L A A U R M A S D L Z U T A U T O A R C H I V M U U T E C U F
E M A R F V A N S A N E T I E S S N O I S S U K S I D E L A A Z
```

Dieses Beispiel existiert nur, um zu zeigen, dass das Programm auch nicht-quadratische Spielfelder und sehr große Spielfelder mit vielen Worten problemlos generieren kann.

Um Platz zu sparen, wurden hier die gesuchten Wörter und der einfache und mittlere Schwierigkeitsgrad nicht mit abgedruckt.

Quellcode

Überprüfen einer möglichen Position

```
def is_word_possible(pos, word, grid, direction):
    """Überprüft, ob es möglich ist, ein Wort an gegebener Position einzufügen.

    Argumente:
    pos -- Tupil mit x und y Koordinaten des Wortes
    word -- Das zu überprüfende Wort als String
    grid -- Das Gitter in dem das Wort platziert werden soll
    direction -- Ausrichtung des Wortes im Gitter

    Rückgabe:
    True || False -- Das Wort kann (nicht) eingefügt werden
    """
    x, y = pos
    space = len(word)
    for i in range(space):
        if direction == WordDirection.vertical:
            if i + y >= height:
                return False
            if grid[y + i][x] != " ":
                return False
        elif direction == WordDirection.horizontal:
            if i + x >= width:
                return False
            if grid[y][x + i] != " ":
                return False
        elif direction == WordDirection.diagonal:
            if i + y >= height or i + x >= width:
                return False
            if grid[y + i][x + i] != " ":
                return False
    return True
```

Die Methode nimmt die Position eines Wortes als x und y Koordinate, das Wort selber und die Rotation des Wortes und überprüft, ob an dieser Stelle genügend Platz wäre um ein Wort einzufügen (oder ob das Wort über das Gitter hinausragen würde) und ob das Wort ein anderes Wort überschneiden würde.

Dafür iterieren wir über jeden Buchstaben des Wortes und überprüfen, ob die Koordinate des Buchstabens größer als die Breite bzw. Länge des Feldes ist. Sollte sie größer sein, bedeutet es, dass sich Wort an dieser Stelle über das Feld hinaus erstrecken würde und die Methode gibt False zurück.
Sollte das Wort horizontal im Spielfeld liegen, wird nur die x-Koordinate überprüft, bei vertikaler Lage nur die y-Koordinate und bei diagonaler Lage beide Komponenten.

Die Methode gibt außerdem False zurück, wenn an einer Stelle, an der ein Buchstabe eingefügt werden soll, kein Leerzeichen ist (also bereits ein anderer Buchstabe die Position belegt). In allen anderen Fällen gibt die Methode True zurück.

Auffüllen des Spielfeldes

```python
def populate_grid(in_grid, alphabet, difficulty, width, height, words, possible_word_directions):
    """Füllt das Gitter mit Wörtern und leere Stellen mit Buchstaben aus dem generierten Alphabet auf.

    Argumente:
    in_grid -- Das Gitter, das gefüllt werden soll
    alphabet -- In `generate_alphabet()` generierte Zeichenkette
    difficulty -- Gewollte  Schwierigkeitsstufe des Gitters
    width -- Breite des Gitters
    height -- Höhe des Gitters
    words -- Liste mit einzufügenden Wörtern
    possible_word_directions -- Mögliche Ausrichtungen eines Wortes

    Rückgabe:
    grid -- Das aufgefüllte Gitter
    """

    grid = in_grid
    for word in words:
        x = random.randint(0, width - 1)
        y = random.randint(0, height - 1)
        direction = random.choice(possible_word_directions)
        reversed = random.choice(
            [True, False]) if difficulty == Difficulty.hard else False
        while not is_word_possible((x, y), word, grid, direction):
            x = random.randint(0, width - 1)
            y = random.randint(0, height - 1)
            direction = random.choice(possible_word_directions)
        grid = place_word((x, y), word, grid, direction, reversed)

    # Füllen des Gitters mit zufälligen Buchstaben des generierten Alphabets
    for i in range(0, height):
        for k in range(0, width):
            if grid[i][k] == " ":
                grid[i][k] = alphabet[random.randint(0, 25)]

    return grid
```

Diese Methode übernimmt die eigentliche Aufgabe des Erstellens eines Buchstabenrechtecks unter Berücksichtigung des Schwierigkeitsgrades, der Größe des Gitters und der zu suchenden Worte.

Der Prozess ist recht simpel. Wir fügen zuerst jedes Wort in das Gitter ein. Dafür überprüfen wir solange mit der zuvor beschriebenen Methode zufällige Positionen und Rotationen für ein Wort, bis wir eine valide Position gefunden haben und das Wort mit der Methode *“place_word”* in das Gitter einfügen können.

Dann überprüfen wir für jeden Platz im Gitter, ob ein Leerzeichen vorhanden ist. Sollte dies der Fall sein, ersetzen wir es durch einen zufälligen Buchstaben aus unserem generiertem Alphabets.