

Aufgabe 2:

Alexandru Duca
Einsendenummer: 00222

30. März 2018

Inhaltsverzeichnis

1	Lösungsidee	2
1.1	Das Finden einer roten Linie, die am kostengünstigsten zum Errichten ist	3
1.2	Die Angabe aller Umbauarbeiten, um diese rote Linie zu errichten	4
2	eigenständige Erweiterung der Aufgabenstellung	6
3	Umsetzung	7
4	Beispiele	8
4.1	wildschwein1.txt	8
4.2	wildschwein2.txt	10
4.3	wildschwein3.txt	12
4.4	wildschwein4.txt	14
4.5	wildschwein5.txt	16
4.6	eigenesBeispiel1.txt	18
4.7	eigenesBeispiel2.txt	20
5	Quellcode	23

1 Lösungsidee

Betrachtet wird ein beliebiges Ausgangsfeld A:

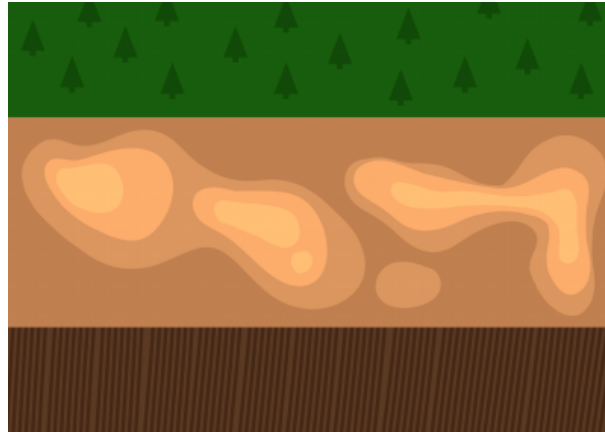


Abbildung 1: Feld A

Die Aufgabe ist Feld A durch Terraforming wildschweinundurchlässig zu machen. Das heißt in anderen Worten, dass durch eine Menge von Umbauarbeiten mindestens eine rote Linie entsteht, die von einem Rand zum anderen Rand führt und die nicht von Wildschweinen überschritten werden kann. Gäbe es eine solche Linie nach den Umbauarbeiten nämlich nicht, so wäre das Ergebnis des Terraformings wildschweindurchlässig.

Die Aufgabenstellung lässt sich somit auf zwei Teilprobleme zurückführen:

- 1.) Das Finden einer roten Linie, die am kostengünstigsten zum Errichten ist
- 2.) Die Angabe aller Umbauarbeiten, die diese rote Linie errichten

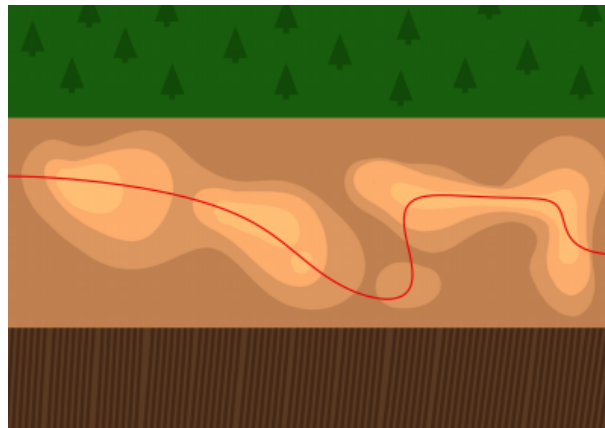


Abbildung 2: Feld A mit möglicher roter Linie

1.1 Das Finden einer roten Linie, die am kostengünstigsten zum Errichten ist

Im weiteren Verlauf werden keine stetigen Felder betrachtet, sondern welche, die in Planquadrate unterteilt sind, also so wie es die Aufgabenstellung voraussetzt. Eine rote Linie hat folglich einen eckigen Verlauf, wie es das folgende Beispiel zeigt:

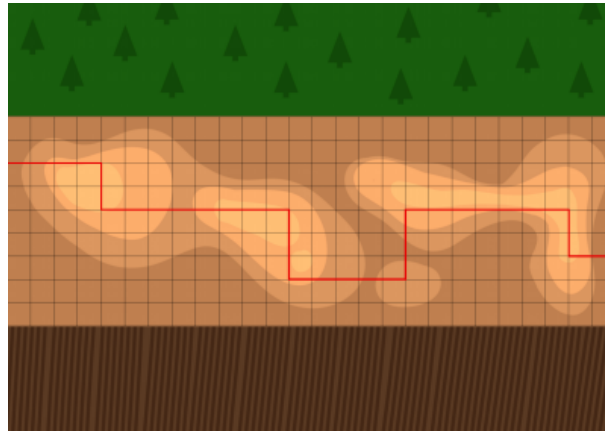


Abbildung 3: Feld A mit Raster und möglicher roter Linie

Um die rote Linie zu finden, deren Errichtung die kleinste Menge an Umbauarbeiten erfordert, wird der Dijkstra-Algorithmus verwendet: Zunächst stellt jede Ecke von mindestens einem Planquadrat ein Knoten in einem Graphen G dar. Jeder dieser Knoten ist mit höchstens vier Nachbarknoten, die falls existent unmittelbar nördlich, südlich, östlich und westlich liegen, durch ungerichtete Kanten verbunden. Zusätzlich gibt es einen Startknoten S , der mit allen Knoten des westlichen Randes des Feldes verbunden ist, und einen Endknoten E , der mit allen Knoten des östlichen Randes des Feldes verbunden ist. Es folgt eine Gewichtung der Kanten: Alle Kanten, die von S oder E ausgehen, haben die Gewichtung 0. Alle Kanten, die genau ein angrenzendes Planquadrat haben, haben ebenfalls die Gewichtung 0. Alle übrigen Kanten, die genau zwei angrenzende Planquadrate haben, sind mit der rationalen Zahl gewichtet, die die Menge an Erde beschreibt, die von einem nebenanliegendem Planquadrat zum anderem nebenanliegendem Planquadrat geschüttet werden muss, sodass die Höhendifferenz größer oder gleich 1 ist.

Auf den Graphen G soll nun der Dijkstra-Algorithmus angewendet werden, der den kürzesten Pfad von S zu E bestimmt. Dieser Pfad hat die Eigenschaft, dass, sofern alle Umbauarbeiten entlang dieses Pfades getätigt sind, keine Wildschweine diesen überqueren können, und, dass die Summe aller Umbauarbeiten minimal ist. Folglich repräsentiert dieser Pfad die rote Linie.

1.2 Die Angabe aller Umbauarbeiten, um diese rote Linie zu errichten

Da die rote Linie erfolgreich ermittelt wurde, wird nun entlang dieser Linie gelaufen und an jeder Kante die Höhendifferenz vergrößert, indem die dafür benötigte Menge an Erde von einem nebenanliegendem Planquadrat zum anderen nebenanliegendem Planquadrat geschüttet wrd, sodass die Höhendifferenz entlang der Linie immer genau 1 oder größer ist.

Es kann jedoch zu Problemen kommen, wenn die rote Linie nicht geradlinig verläuft und somit Ecken hat. Betrachtet wird im folgendem Beispiel eine Ecke im Verlauf der roten Linie:

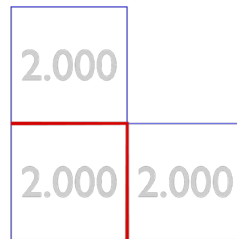


Abbildung 4: rote Linie mit Ecke

Würde der Algorithmus so vorgehen, dass er alle Kanten der roten Linie der Reihe nach unüberquerbar macht, so sieht das Ergebnis folgendermaßen aus:



Abbildung 5: erster Lösungsansatz

Zuerst werden 0.5 m Erde vom nördlichen Planquadrat zum südlichen gekippt, um die lokale Höhendifferenz zu schaffen. Anschließend reicht es 0.25 m vom östlichen Planquadrat aus nach Westen zu kippen, damit die Linie unüberquerbar wird. In der Summe wird also 0.75 m Erde verschoben. Die richtige Lösung sieht jedoch so aus:

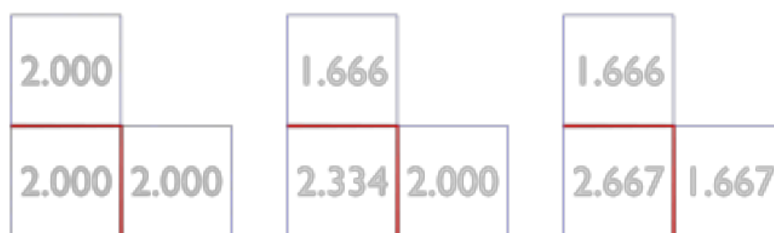


Abbildung 6: eigentliche Lösung

Es reicht nämlich bereits einen Drittel Meter - plus einen Aufrundungswert in Höhe von 0.001 m - von den äußeren Planquadraten auf das mittlere zu verschieben. Dadurch ist das mittlere Planquadrat zwei Drittel Meter höher und die anderen Planquadrate ein Drittel Meter tiefer als die Ausgangshöhe von 2 Meter. Folglich ist die Höhendifferenz 1 m bzw. 1.001 m. Die Summe der verschobenen Meter an Erde beträgt somit auch nur 0.667 m und nicht 0.75 m.

Aufgabe 2:

Aus der Betrachtung dieses Spezialfalls werden mehrere Erkenntnisse gewonnen:

- Ist ein Planquadrat in mindestens zwei Umbauarbeiten involviert, kommt es zu Überschneidungen, die zunächst keine triviale Lösung haben.
- Wenn die Höhendifferenz zwischen zwei Planquadraten zuvor kleiner als 1 m und nach den Umbauarbeiten größer als 1.001 m ist, so wurde mehr Erde verschoben als nötig war. Folglich ist das Ergebnis nicht optimal.
- Da die Ein- und Ausgabewerte der Höhen für alle Planquadrate nur auf drei Dezimalen genau angegeben werden, sind die Mengen an verschobener Erde pro Umbauarbeit auf drei Dezimalen beschränkt.

Ein Algorithmus, der für einen festgelegten Pfad die optimale Menge an Umbauarbeiten ausgibt, funktioniert somit wie folgt:

- A.) Bestimme alle möglichst langen Teilpfade der roten Linie, die an keiner Ecken der roten Linie anfangen und enden. Erhöhe entlang dieser Teilpfade alle Höhendifferenzen, die kleiner als 1 m sind, auf 1 m bzw. 1.001 m.
- B.) Bestimme alle möglichst langen Teilpfade, deren Knoten ausschließlich Ecken der roten Linie sind. Gehe nun wie folgt vor, um die Menge der Umbauarbeiten entlang dieser Teilpfade anzugeben:
 - 1.) Setze eine 'Merkliste' auf, in der alle folgenden Umbauarbeiten gespeichert werden.
 - 2.) Wiederhole solange, bis nach einem Durchlauf keine Änderung mehr gemacht werden:
 - I.) Sperre alle Kanten, an denen die Höhendifferenz 1 m oder größer ist. Über gesperrete Kanten kann keine Erde gekippt werden.
 - II.) Wiederhole solange, bis nach einem Durchlauf keine Änderung mehr gemacht werden:
 - a.) Gehe alle Kanten des Teilpfades der Reihe nach durch und kippe über jede Kante die Menge an Erde, die nötig ist, um die Höhendifferenz auf genau 1 m oder 1.001 m zu erhöhen oder zu erniedrigen. Speichere die Änderungen in der 'Merkliste'.
 - III.) Entsperre alle Kanten.
 - 3.) Rechne mit Hilfe der 'Merkliste' die Bilanz der Änderungen an jeder Kante aus.

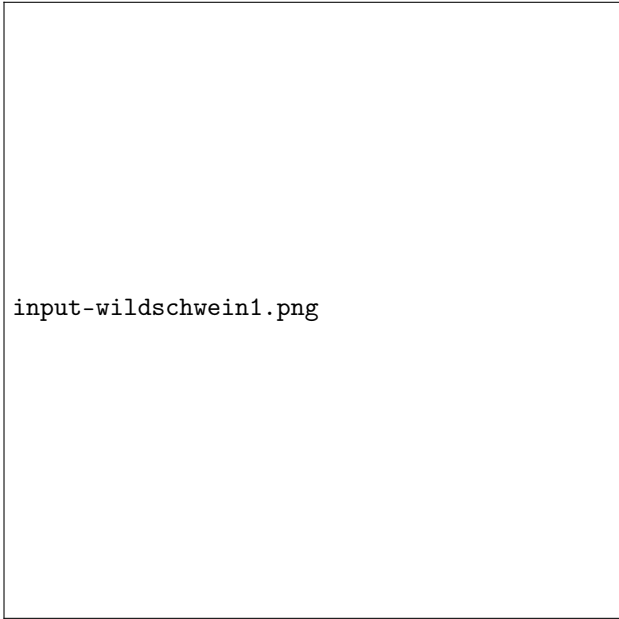
2 eigenständige Erweiterung der Aufgabenstellung

- Die Ein- und Ausgabewerte der Feldmatrix soll sowohl auf Textbasis als auch auf graphischer Ebene visualisiert und ausgegeben werden.
- Das Programm ignoriert die erste Zeile der Eingabetextdatei. Eingabefelder, die nicht quadratisch aber dennoch rechteckig sind, werden auch akzeptiert.
- Das Programm kann auch mit negativen Höhen rechnen und diese ausgeben

3 Umsetzung

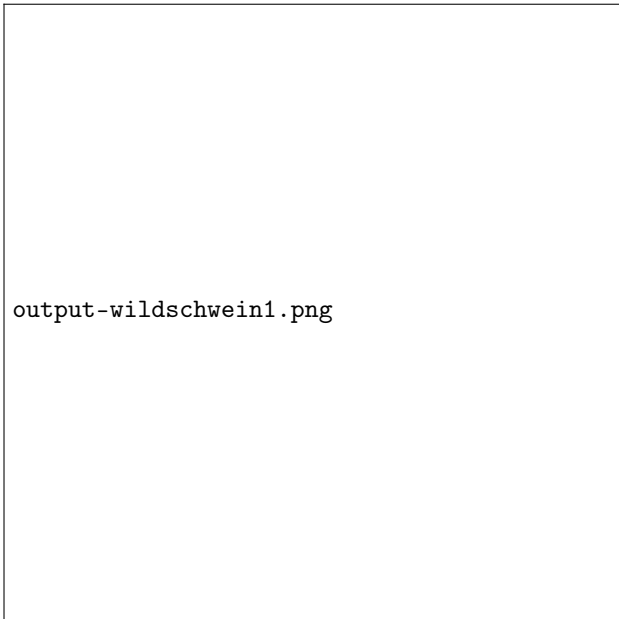
4 Beispiele

4.1 wildschwein1.txt



input-wildschwein1.png

Abbildung 7: input-wildschwein1.png



output-wildschwein1.png

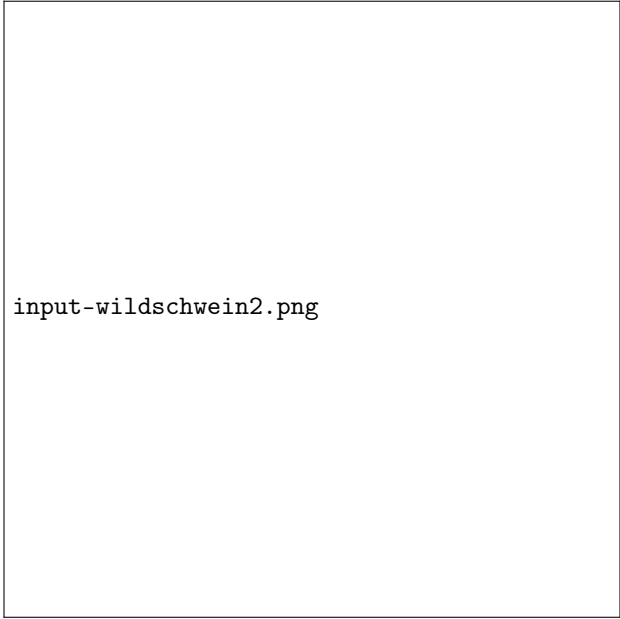
Abbildung 8: output-wildschwein1.png

Aufgabe 2:

```
1  - - - - - Programmstart - - - - -
3
5  Dateiname der Textdatei.
   >>> wildschwein1.txt
7
9  Kippe von (0,9) nach (0,8) eine Hoehe von 0.479 m .
   Kippe von (1,9) nach (1,8) eine Hoehe von 0.462 m .
11 Kippe von (2,9) nach (2,8) eine Hoehe von 0.428 m .
   Kippe von (3,9) nach (3,8) eine Hoehe von 0.358 m .
13 Kippe von (4,9) nach (4,8) eine Hoehe von 0.224 m .
   Kippe von (5,9) nach (5,8) eine Hoehe von 0.031 m .
15 Kippe von (7,9) nach (7,8) eine Hoehe von 0.039 m .
   Kippe von (8,9) nach (8,8) eine Hoehe von 0.243 m .
17 Kippe von (9,9) nach (9,8) eine Hoehe von 0.386 m .
   Kippe von (9,9) nach (10,9) eine Hoehe von 0.409 m .
19 Kippe von (10,10) nach (9,10) eine Hoehe von 0.473 m .
   Kippe von (9,11) nach (10,11) eine Hoehe von 0.473 m .
21 Kippe von (9,12) nach (10,12) eine Hoehe von 0.409 m .
   Kippe von (10,13) nach (10,12) eine Hoehe von 0.386 m .
23 Kippe von (11,12) nach (11,13) eine Hoehe von 0.243 m .
   Kippe von (12,12) nach (12,13) eine Hoehe von 0.039 m .
25 Kippe von (14,12) nach (14,13) eine Hoehe von 0.031 m .
   Kippe von (15,12) nach (15,13) eine Hoehe von 0.224 m .
27 Kippe von (16,12) nach (16,13) eine Hoehe von 0.358 m .
   Kippe von (17,12) nach (17,13) eine Hoehe von 0.428 m .
29 Kippe von (18,12) nach (18,13) eine Hoehe von 0.462 m .
   Kippe von (19,12) nach (19,13) eine Hoehe von 0.479 m .
31
   Es wurden insgesmt 7.064 m Hoehe an Erde verschoben.
33
35 Laufzeit: 1.06402 s
37
   - - - - - Programmende - - - - -
39
```

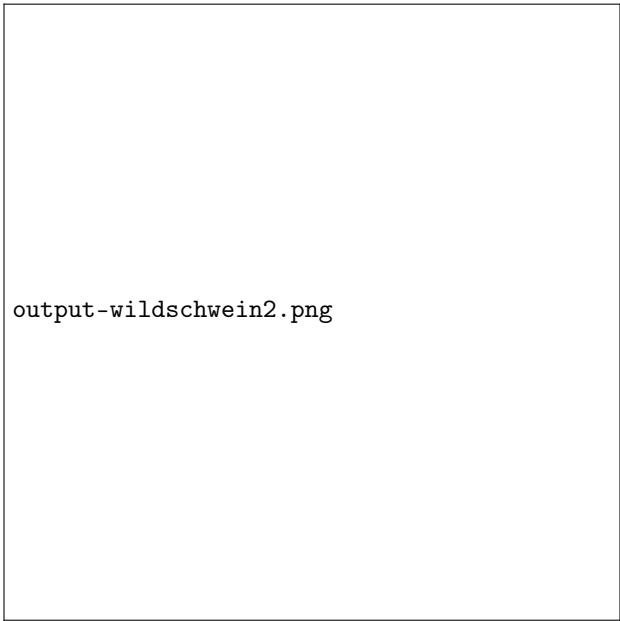
Der Input entspricht zwei Bergen, die diagonal zueinander liegen. Das Programm nutzt die Steigung um die zwei Bergen herum, um diese zu einer Mauer aufzuschütten. Lediglich eine unüberschreitbare mittlere Senkrechte zwischen den zwei Bergen muss aufgeschüttet werden. Hierfür findet das Programm ein nützliches Karomuster.

4.2 wildschwein2.txt



input-wildschwein2.png

Abbildung 9: input-wildschwein2.png



output-wildschwein2.png

Abbildung 10: output-wildschwein2.png

Aufgabe 2:

```
2 - - - - - Programmstart - - - - -
4
Dateiname der Textdatei.
6 >>> wildschwein2.txt

8
Kippe von (0,1) nach (0,0) eine Hoehe von 0.500 m .
10 Kippe von (1,1) nach (1,0) eine Hoehe von 0.500 m .
Kippe von (2,1) nach (2,0) eine Hoehe von 0.500 m .
12 Kippe von (3,1) nach (3,0) eine Hoehe von 0.500 m .
Kippe von (4,1) nach (4,0) eine Hoehe von 0.500 m .
14 Kippe von (5,1) nach (5,0) eine Hoehe von 0.500 m .
Kippe von (6,1) nach (6,0) eine Hoehe von 0.500 m .
16 Kippe von (7,1) nach (7,0) eine Hoehe von 0.500 m .
Kippe von (8,1) nach (8,0) eine Hoehe von 0.500 m .
18 Kippe von (9,1) nach (9,0) eine Hoehe von 0.500 m .
Kippe von (10,1) nach (10,0) eine Hoehe von 0.500 m .
20 Kippe von (11,1) nach (11,0) eine Hoehe von 0.500 m .

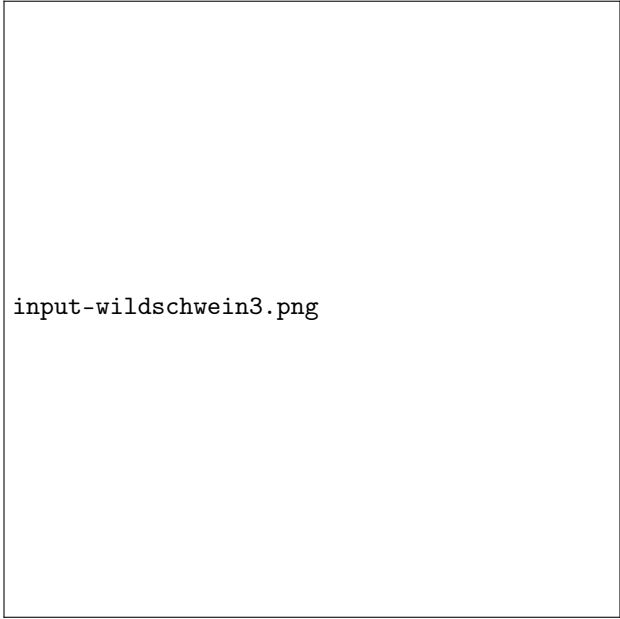
22 Es wurden insgesmt 6.000 m Hoehe an Erde verschoben.

24
Laufzeit: 0.32575099999999996 s
26

28 - - - - - Programmende - - - - -
```

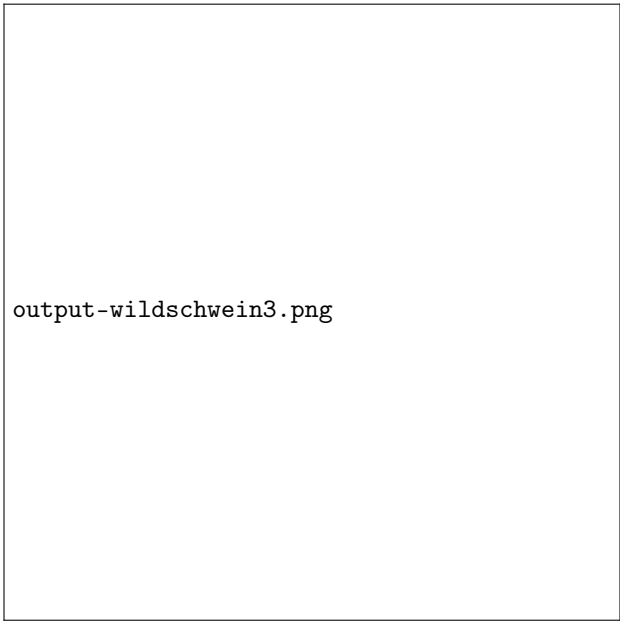
Der Input entspricht einer einfachen Ebene. Hier gibt es keine natürlichen Gegebenheiten, die geschickt genutzt werden können. Die günstigste Möglichkeit ist also das Aufschütten eines Graben.

4.3 wildschwein3.txt



input-wildschwein3.png

Abbildung 11: input-wildschwein3.png



output-wildschwein3.png

Abbildung 12: output-wildschwein3.png

Aufgabe 2:

```
2  - - - - - Programmstart - - - - -
4
Dateiname der Textdatei.
6 >>> wildschwein3.txt

8
Kippe von (0,4) nach (0,5) eine Hoehe von 0.226 m .
10 Kippe von (1,5) nach (1,4) eine Hoehe von 0.500 m .
Kippe von (2,4) nach (2,5) eine Hoehe von 0.226 m .
12 Kippe von (3,4) nach (3,5) eine Hoehe von 0.226 m .
Kippe von (4,4) nach (4,5) eine Hoehe von 0.226 m .
14 Kippe von (5,5) nach (5,4) eine Hoehe von 0.500 m .
Kippe von (6,4) nach (6,5) eine Hoehe von 0.226 m .
16 Kippe von (7,4) nach (7,5) eine Hoehe von 0.226 m .
Kippe von (8,4) nach (8,5) eine Hoehe von 0.226 m .
18 Kippe von (9,1) nach (9,0) eine Hoehe von 0.500 m .

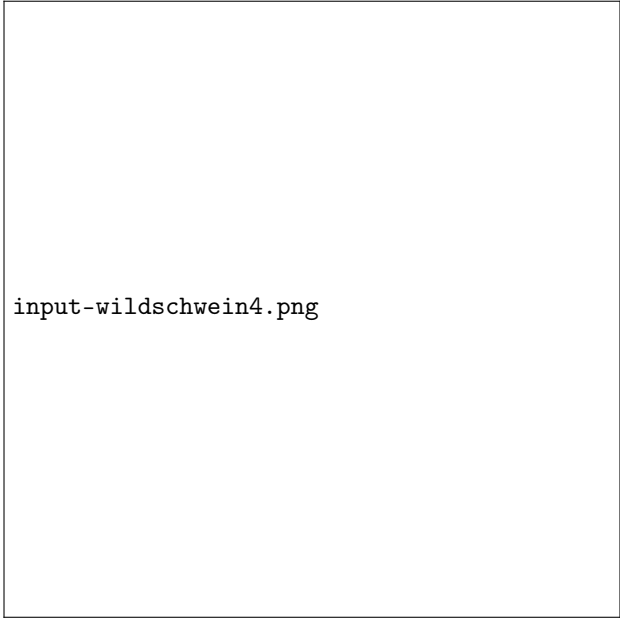
20 Es wurden insgesmt 3.083 m Hoehe an Erde verschoben.

22
Laufzeit: 0.262825 s
24

26 - - - - - Programmende - - - - -
```

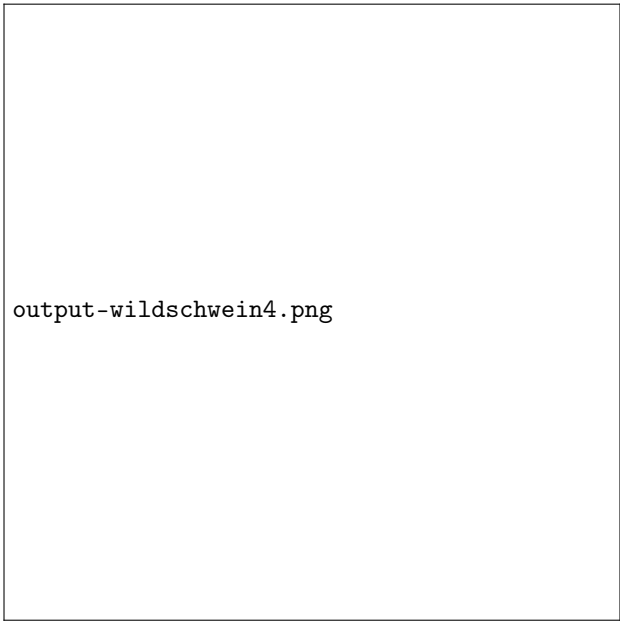
Der Input entspricht drei tiefen Gängen, die vom Wald zum Bauernhof führen. Ein Wildschwein hat also die Möglichkeit durch eines dieser drei Gänge zu laufen, oder über eines der drei Erhöhung die zwischen den Gängen bzw. am Rand liegen. Jene Erhöhungen weisen zudem Hügel auf, die zum Aufschütten einer Mauer genutzt werden können. Die Gänge sind jedoch setig auf gleicher Höhe. Hier muss für jeden Gang einzeln eine Mauer aufgeschüttet werden.

4.4 wildschwein4.txt



input-wildschwein4.png

Abbildung 13: input-wildschwein4.png



output-wildschwein4.png

Abbildung 14: output-wildschwein4.png

Aufgabe 2:

```
2 - - - - - Programmstart - - - - -
4
Dateiname der Textdatei.
6 >>> wildschwein4.txt

8
Kippe von (0,4) nach (0,5) eine Hoehe von 0.223 m .
10 Kippe von (1,4) nach (1,5) eine Hoehe von 0.253 m .
Kippe von (2,4) nach (2,5) eine Hoehe von 0.441 m .
12 Kippe von (3,5) nach (3,4) eine Hoehe von 0.101 m .
Kippe von (4,4) nach (4,5) eine Hoehe von 0.287 m .
14 Kippe von (5,5) nach (4,5) eine Hoehe von 0.187 m .
Kippe von (5,5) nach (5,6) eine Hoehe von 0.196 m .
16 Kippe von (6,6) nach (6,5) eine Hoehe von 0.199 m .
Kippe von (7,5) nach (7,6) eine Hoehe von 0.309 m .
18 Kippe von (8,6) nach (8,5) eine Hoehe von 0.097 m .
Kippe von (9,6) nach (9,5) eine Hoehe von 0.184 m .
20 Kippe von (9,6) nach (10,6) eine Hoehe von 0.198 m .
Kippe von (10,7) nach (10,6) eine Hoehe von 0.39 m .
22 Kippe von (11,6) nach (11,7) eine Hoehe von 0.209 m .
Kippe von (12,6) nach (12,7) eine Hoehe von 0.352 m .
24 Kippe von (13,7) nach (13,6) eine Hoehe von 0.106 m .
Kippe von (14,6) nach (14,7) eine Hoehe von 0.296 m .

26 Es wurden insgesmt 4.029 m Hoehe an Erde verschoben.
28

30 Laufzeit: 0.455719 s

32
- - - - - Programmende - - - - -
34
```

Der Input entspricht einer scheinbar zufällig generierten Menge von Zahlen. Das Programm findet oben links einen Vorsprung, der für einen kleinen Aufpreis zu einem Graben unfunktioniert werden kann. In der Mitte findet sich ein Karo muster aus Höhen und Tiefen. Diese können zu einer Zick-Zack-Mauer umfunktioniert werden. Für den restlichen Teil der Mauer ist es am günstigsten eine gerade Mauer zur rechten Mauer zu ziehen.

4.5 wildschwein5.txt



Abbildung 15: input-wildschwein5.png

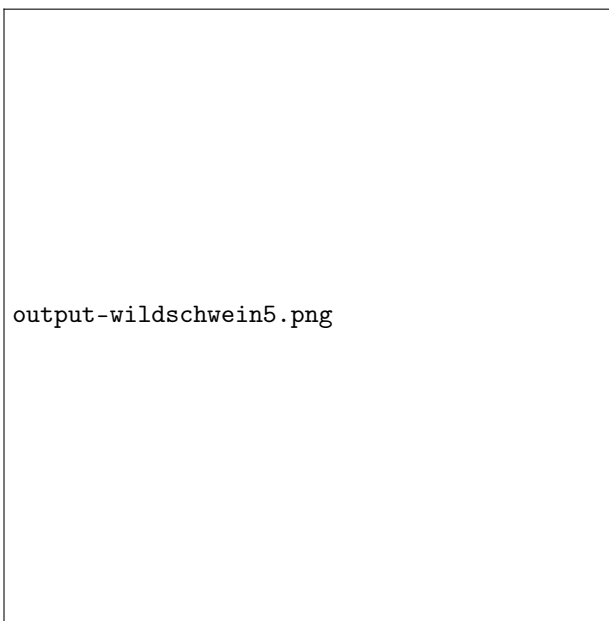


Abbildung 16: output-wildschwein5.png

Aufgabe 2:


```

1  - - - - - Programmstart - - - - -
3
5  Dateiname der Textdatei.
   >>> wildschwein5.txt
7
9  Kippe von (0,12) nach (0,13) eine Hoehe von 0.256 m .
   Kippe von (1,12) nach (1,13) eine Hoehe von 0.224 m .
11 Kippe von (2,12) nach (2,13) eine Hoehe von 0.196 m .
   Kippe von (3,12) nach (3,13) eine Hoehe von 0.17 m .
13 Kippe von (4,12) nach (4,13) eine Hoehe von 0.15 m .
   Kippe von (5,12) nach (5,13) eine Hoehe von 0.136 m .
15 Kippe von (6,12) nach (6,13) eine Hoehe von 0.129 m .
   Kippe von (7,12) nach (7,13) eine Hoehe von 0.130 m .
17 Kippe von (8,12) nach (8,13) eine Hoehe von 0.139 m .
   Kippe von (9,12) nach (9,13) eine Hoehe von 0.155 m .
19 Kippe von (10,12) nach (10,13) eine Hoehe von 0.177 m .
   Kippe von (11,12) nach (11,13) eine Hoehe von 0.204 m .
21 Kippe von (12,12) nach (12,13) eine Hoehe von 0.234 m .
   Kippe von (13,12) nach (13,13) eine Hoehe von 0.264 m .
23 Kippe von (14,13) nach (13,13) eine Hoehe von 0.117 m .
   Kippe von (14,14) nach (13,14) eine Hoehe von 0.089 m .
25 Kippe von (14,15) nach (13,15) eine Hoehe von 0.066 m .
   Kippe von (14,16) nach (13,16) eine Hoehe von 0.048 m .
27 Kippe von (14,17) nach (13,17) eine Hoehe von 0.037 m .
   Kippe von (14,18) nach (13,18) eine Hoehe von 0.034 m .
29 Kippe von (14,19) nach (14,18) eine Hoehe von 0.137 m .
   Kippe von (15,19) nach (15,18) eine Hoehe von 0.149 m .
31 Kippe von (16,19) nach (16,18) eine Hoehe von 0.168 m .
   Kippe von (17,19) nach (17,18) eine Hoehe von 0.192 m .
33 Kippe von (18,19) nach (18,18) eine Hoehe von 0.22 m .
   Kippe von (19,19) nach (19,18) eine Hoehe von 0.250 m .
35
   Es wurden insgesmt 4.071 m Hoehe an Erde verschoben.
37
39 Laufzeit: 1.031424 s
41
43  - - - - - Programmende - - - - -

```


Der Input entspricht einer Gebirgskette mit Bergen und Tälern. Das Gefälle in der Mitte der Berge kann für einen kleinen Aufpreis so steil gemacht werden, dass die Wildschwein nicht mehr hochlaufen können. Nur ein Tal unten rechts muss tiefer gegraben werden, damit in dieses keine Wildschweine gelangen können.

4.6 eigenesBeispiel1.txt



input-eigenesBeispiel1.png

Abbildung 17: input-eigenesBeispiel1.png



output-eigenesBeispiel1.png

Abbildung 18: output-eigenesBeispiel1.png

Aufgabe 2:

```

2 12
2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
4 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
6 2.000 2.000 2.000 2.000 3.000 3.000 3.000 3.000 3.000 3.000 2.000 3.000
2.000 2.000 2.000 2.000 3.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
8 2.000 2.000 2.000 2.000 3.000 3.000 3.000 3.000 3.000 2.000 2.000 2.000
2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 3.000 2.000 2.000 2.000
10 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 3.000 2.000 2.000 2.000
3.000 3.000 3.000 3.000 3.000 3.000 3.000 3.000 3.000 2.000 2.000 2.000
12 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
14 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000

```

```

2 - - - - - Programmstart - - - - -
4
Dateiname der Textdatei.
6 >>> eigenesBeispiel1.txt
8
Kippe von (10,3) nach (10,2) eine Hoehe von 0.500 m .
10
Es wurden insgesmt 0.500 m Hoehe an Erde verschoben.
12
14 Laufzeit: 0.31896499999999994 s
16
- - - - - Programmende - - - - -
18

```

Der Input entspricht einer Gebirgsschlange, die an einer Stelle eine Lücke, die gestopft werden muss, damit keine Wildschweine mehr durchkommen. Das Programm nutzt die gesamte Gebirgsschlange entsprechend.

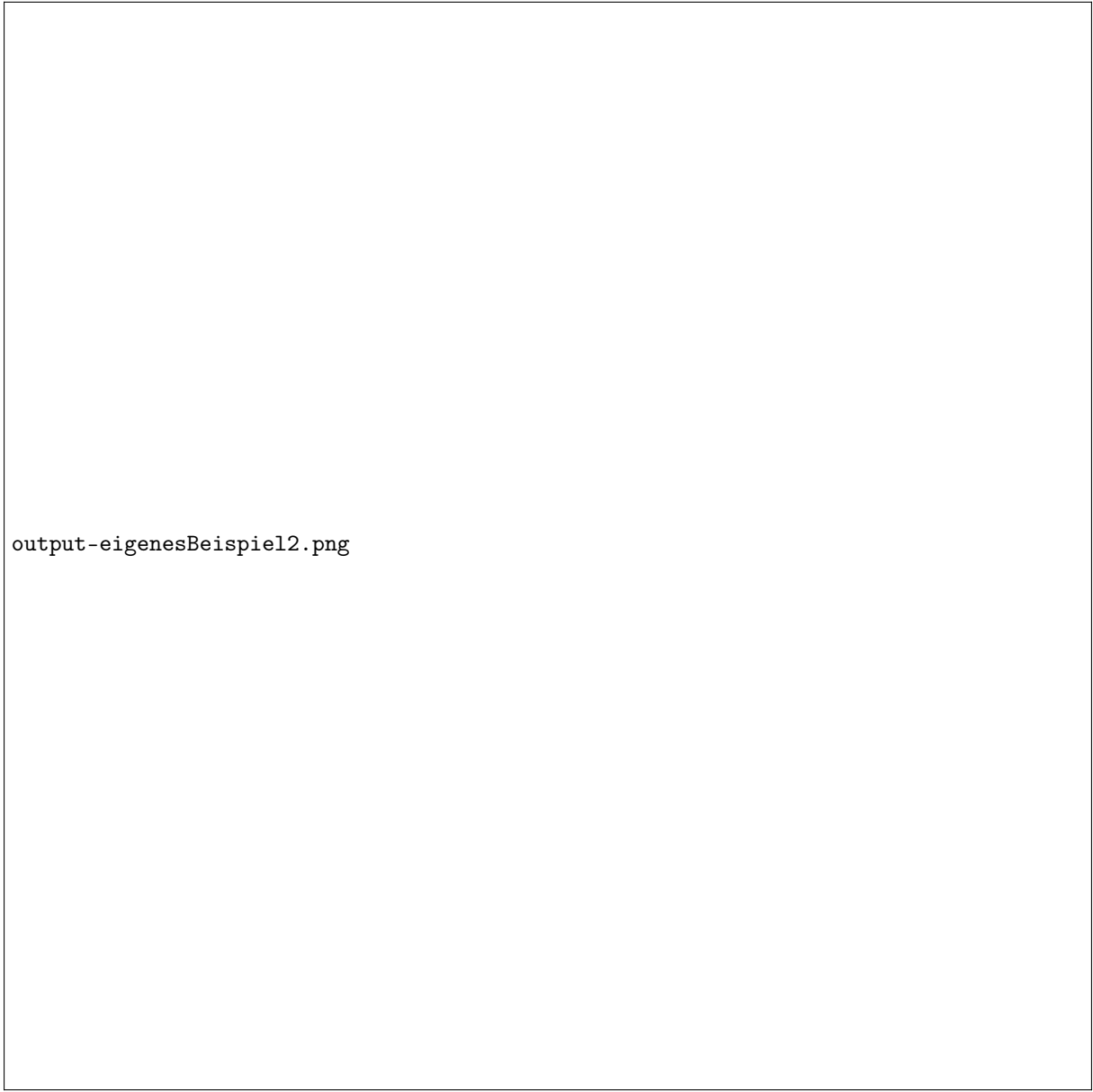
4.7 eigenesBeispiel2.txt



input-eigenesBeispiel2.png

Abbildung 19: input-eigenesBeispiel2.png

Aufgabe 2:



output-eigenesBeispiel2.png

Abbildung 20: output-eigenesBeispiel2.png

Aufgabe 2:

```

1  10x-18
3  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
   -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
5  -1  0  0  0 -1 -1  0 -1 -1 -1 -1 -1 -1  0  0  0  0 -1 0.0 -1 -1 0.0
   0 -1 -1 -1  0 -1  0 -1 -1 -1 -1 -1  0 -1 -1 -1 -1 -1 0.0 -1 -1 0.0
7  0 -1 -1 -1  0 -1  0 -1 -1 -1 -1 -1  0  0  0  0  0 -1 -1 0.0 0.0 -1
   0  0  0  0  0 -1  0 -1 -1 -1 -1 -1  0 -1 -1 -1 -1 -1 0.0 0.0 -1
9  0 -1 -1 -1  0 -1  0 -1 -1 -1 -1 -1  0 -1 -1 -1 -1 -1 0.0 -1 -1 0.0
   0 -1 -1 -1  0 -1  0  0  0  0  0 -1 -1  0  0  0  0 -1 0 -1 -1 0.0
11 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
13

```

```

2  - - - - - Programmstart - - - - -
4
   Dateiname der Textdatei.
6  >>> eigenesBeispiel2.txt
8
   Kippe von (0,3) nach (0,2) eine Hoehe von 0.500 m .
10  Kippe von (1,3) nach (1,2) eine Hoehe von 0.500 m .
    Kippe von (7,3) nach (7,2) eine Hoehe von 0.500 m .
12  Kippe von (13,7) nach (13,6) eine Hoehe von 0.500 m .
    Kippe von (19,2) nach (19,1) eine Hoehe von 0.500 m .
14  Kippe von (24,2) nach (24,1) eine Hoehe von 0.500 m .
    Kippe von (25,2) nach (25,1) eine Hoehe von 0.500 m .
16
   Es wurden insgesmt 3.500 m  Hoehe an Erde verschoben.
18
20 Laufzeit: 0.655612 s
22
   - - - - - Programmende - - - - -
24

```

Der Input entspricht meinen Initialien, die durch Berge in den Boden geschrieben wurden. Dieses Beispiel soll demonstrieren, dass als Eingabe auch eine rechteckige nicht quadratische Grundflächen akzeptiert wird. Auch soll das Beispiel beweisen, dass mein Programm auch mit negativen Zahlwerten arbeiten kann. Ebenfalls lässt sich erkennen, dass die Anzahl an Leerzeichen zwischen zwei Zahlen in der Textdatei beliebig sein kann. Der Wert der ersten Zeile wird übrigens immer ignoriert.

5 Quellcode

1
