Inhaltsverzeichnis

1 Vorhergehende Definitionen **2**

2 Lösungsidee **2**
2.1 Genereller Ansatz 2
2.2 Modellierung 3
2.3 Algorithmen 3
2.3.1 Aufbau der Adjazenzliste 3
2.3.2 Breitensuche 3

3 Implementierung **4**
3.1 Aufbau der Adjazenzliste 4
3.2 Breitensuche 4
3.3 Ausgabe 5

4 Laufzeit **5**

5 Erweiterungen **5**

6 Beispiele **7**

7 Quellen **13**

8 Quellcode **13**

1 Vorhergehende Definitionen

Ein Schritt auf dem Spielfeld des Yamyams ist das Taumeln des Yamyams von einem Feld zu einem anderen, wo es aufgrund einer Wand anhalten muss. Ein Weg von einem Feld zu einem anderen ist eine Abfolge von Schritten, die bei dem einen Feld beginnt und bei dem anderen endet. Dabei hält der Yamyam auf jedem Feld höchstens einmal an. Würde er auf einem Feld mehrfach anhalten, so muss er einen Kreis von diesem Feld zu ihm selbst zurückgegangen sein. Diesen kann er auch einfach auslassen, da sich weder das Feld noch die Möglichkeiten des Weitertaumelns durch mehrere Besuche ändern. Im Folgenden wird angenommen, dass das Spiel endet, wenn der Yamyam einen Ausgang erreicht. Daher kann ein Weg nur einmal über einen Ausgang führen, und zwar dann, wenn er auf diesem Ausgang endet.

2 Lösungsidee

2.1 Genereller Ansatz

Zuerst wird der Begriff der unsicheren Felder eingeführt. Die unsicheren Felder, sind die Felder, von denen es unmöglich ist zu einem Ausgang zu taumeln. Im ersten Beispiel der Aufgabenstellung sind dies u.a. die Felder links oben und links unten. Es ist offensichtlich, dass man von jedem der beiden Felder ausschließlich zu dem jeweils anderen gelangen kann, der Yamyam ist also auf diesen Feldern gefangen. Diese Felder sind per Definition keine sicheren Felder.

Wenn es nun von einem anderen Feld einen Weg gibt, der zu einem unsicheren Feld führt, so ist dieses Feld auch nicht sicher. Da wir uns auf endlichen Spielfeldern befinden ist dieser Weg auch endlich, da ein Weg keine Periode enthalten kann. Daher ist die Wahrscheinlichkeit P_1 positiv von diesem Feld irgendwie auf ein unsicheres Feld zu kommen. Wenn aber dieses unsichere Feld erreicht ist, kann man nicht mehr zu einem Ausgang gelangen. Daher ist die Wahrscheinlichkeit von diesem Feld einen Ausgang zu erreichen höchstens $1 - P_1$ und somit echt kleiner als 1, das Feld ist nicht sicher.

Betrachten wir nun die restlichen Felder, also die, von denen man zu einem Ausgang, aber nicht zu einem unsicheren Feld gelangen kann. Wenn sich der Yamyam von diesem Feld beliebig fortbewegt, so gibt es von jedem Feld, auf dem es zwischenzeitlich anhält, einen Weg zu einem Ausgang, andernfalls wäre das Feld unsicher. Somit ist der Yamyam stets nur einen Weg endlicher Länge von einem Ausgang entfernt. Wenn man annimmt, dass der längste solche Weg zu einem Ausgang n Schritte lang ist, dann ist der Yamyam stets maximal n Schritte von einem Ausgang entfernt. Da er sich nur in die 4 Himmelsrichtungen fortbewegen kann, ist die Wahrscheinlichkeit diesen Weg zu gehen, also bei jedem Schritt in die richtige Richtung zu gehen mindestens 0.25^n (Im Normalfall ist sie größer, da der Yamyam an jedem Anstoßpunkt nur 3 mögliche Richtungen hat, aber das ist nicht relevant). Die Wahrscheinlichkeit von seinem aktuellen Feld aus einen Weg zu einem Ausgang zu gehen, ist für das Yamyam also stets mindestens 0.25^n, da der Weg höchstens n lang ist. Daher ist die Wahrscheinlichkeit $P_{\text{Ausgang}}(n)$ den Ausgang in höchstens n Schritten zu finden mindestens 0.25^n. Die Gegenwahrscheinlichkeit den Ausgang in den nächsten n Schritten nicht zu finden

ist also auch zu jedem Zeitpunkt höchstens $1 - P_{\text{Ausgang}}(n)$. Da dieses Maximum der Wahrscheinlichkeit unabhängig von der Position des Yamyams ist, lässt sich auch annehmen, dass die Wahrscheinlichkeit den Ausgang in den nächsten $i \cdot n$ Schritten nicht zu finden, wiederum höchstens $(1 - P_{\text{Ausgang}})^i$ ist. Daher ist die Wahrscheinlichkeit den Ausgang nie zu finden

$$P_{\text{Ausgang unerreichbar}} = \lim_{i \to \infty} (1 - P_{\text{Ausgang}}(n))^i = \lim_{i \to \infty} (1 - 0.25^n)^i = 0$$

Daher ist von jedem Feld, das einen Weg zum Ausgang hat und keinen Weg zu einem unsicheren Feld besitzt, ein Ausgang mit der Wahrscheinlichkeit $1 - P_{\text{Ausgang unerreichbar}} = 1$ erreichbar, sie sind also sicher. Wie oben gezeigt sind es auch die einzigen sicheren Felder.

Daher sucht der Lösungsalgorithmus erst nach allen unsicheren Feldern und dann nach allen Feldern, die zu einem unsicheren Feld führen und gibt die restlichen Felder als sicher aus.

2.2 Modellierung

Das Spielfeld und die Wege des Yamyams lassen sich durch einen gerichteten Graphen modellieren. Dabei gibt es für jedes Feld einen Knoten und zwischen zwei Knoten $\mathfrak{u}$ und $\mathfrak{v}$ gibt es genau dann einen Kante $\mathfrak{u} \to \mathfrak{v}$, wenn der Yamyam von dem von $\mathfrak{v}$ repräsentierten Feld in eine Richtung lostaumeln kann, sodass er auf dem von $\mathfrak{u}$ repräsentierten Feld zu halten kommt. Die Kanten in diesem Graphen sind also quasi umgekehrt zu der Bewegung des Yamyams. Daher gibt es zu jedem Knoten zwischen vier und keine eingehenden Kanten. Ein Spezialfall bilden hierbei die Ausgangsfelder, diesen werden keine eingehende Kanten zugeordnet, da das Spiel, wie oben definiert, endet, sobald der Yamyam auf einem Ausgang steht. Durch diesen Aufbau des Graphen lässt sich sehr einfach herausfinden von welchen Feldern man auf ein gegebenes Feld taumeln kann.

2.3 Algorithmen

2.3.1 Aufbau der Adjazenzliste

Der Aufbau der Adjazenzliste findet während dem Einlesen des Spielfelds statt. Dazu wird das Spielfeld spalten- und zeilenweise untersucht. In jeder Spalte bzw. Zeile werden die zusammenhängende Stücke an betretbaren Feldern (freie Felder und Ausgänge) gesucht und jeweils alle freien Felder eines Stücks als von den Endpunkten des Stücks erreichbar gespeichert.

2.3.2 Breitensuche

Der Großteil der Arbeit wird durch zwei Breitensuchen verrichtet.[1] Die erste ist eine Breitensuche von den Ausgängen aus. Sie markiert somit alle Knoten, die im Graphen von einem Ausgang aus erreichbar sind. Im Spiel sind das alle Felder, von denen ein Weg für den Yamyam zu einem Ausgang existiert. Die unmarkierten Felder sind demnach unsicher. Von diesen aus wird die zweite Breitensuche ausgeführt. Dadurch werden alle Knoten markiert,

[1] Ich frage mich immer noch, ob die Bezeichnung Breiten*suche* nicht etwas übertrieben ist, wenn man nach nichts sucht.

die von einem unsicheren Knoten aus erreichbar sind. Diese entsprechen genau den Feldern von denen ein Weg zu einem unsicheren Feld besteht. Folglich sind die sicheren Felder die Felder, die von der zweiten Breitensuche nicht markiert wurden.

3 Implementierung

Das Programm ist in Python3 geschrieben und befindet sich in der Datei *aufgabe3.py*. Für die Ausgabe ist das Python3-Package Pillow (Version $\geq$ 2.1.0) von Nöten. Dieses ist ein Fork der Python Imaging Library. Zur Installation lässt sich z.B. *$ pip3 install pillow* aufrufen. Zu Beginn wird der Ausführer nach der Eingabedatei gefragt. Diese muss im Format der vorgegebenen Beispieldateien vorliegen. Nach Angabe der Eingabedatei wird die `main`-Methode aufgerufen, die zuerst das Einlesen der Eingabedatei aufruft.

3.1 Aufbau der Adjazenzliste

Der Aufbau der Adjazenzliste findet in der Methode `einlesen` statt. Nach dem Auslesen des Spielfeldes aus der Datei in die zweidimensionale Liste `lines` wird es zeilenweise durchgegangen. Für jede Zeile wird die Methode `maxconnected` aufgerufen und so alle zusammenhängende Stücke dieser Zeile gefunden. In `maxconnected` wird dazu die eingegebene Liste durchgegangen und dabei die Positionen mit Ausnahme der Ausgänge seit der letzten Wand in der Liste `inarow` gesammelt und in der Adjazenzliste für die Position direkt nach der Wand gespeichert. Sobald die nächste Wand getroffen wird, werden die gesammelten Positionen auch in der Adjazenzliste der letzten Position vor der Wand gespeichert. So wird für jedes solches Stück eine Liste aller einzelnen Positionen dieses Stück abzüglich der Ausgänge als Werte der Randpositionen im globalen Dictionary `adjlist` gespeichert. Das Gleiche wird für alle Spalten ausgeführt. So ergibt sich der oben beschriebene Graph. Zusätzlich werden alle Positionen des Feldes mit ihren jeweiligen Zeichen in das globale Dictionary `feld` eingetragen.

3.2 Breitensuche

Nach dem Aufruf von `einlesen` werden die beiden Breitensuchen durchgeführt. Dabei wird bei der ersten Breitensuche für die Menge der Ausgänge die iterative Breitensuchemethode `broad` ausgeführt. Das ist quasi ein Breitensuche, die bei einem imaginären Knoten, der eine Kante zu allen Ausgängen besitzt, begonnen wird. Diese ist eine ganz normale (iterative) Breitensuche und benötigt als solche wohl kaum Erklärungen. Erwähnenswert ist, dass sie keine Abbruchbedingung besitzt, da ja nach nichts speziellem gesucht wird, sondern erst aufhört, wenn alle von den Ausgängen erreichbaren Knoten erschöpft sind und dann alle besuchten Knoten zurück gibt. Dann sind die unsicheren Knoten die Menge aller Knoten abzüglich den gerade markierten. Diese werden in `unsicher` hinterlegt. Danach wird von allen unsicheren Feldern eine Breitensuche, wieder mittels `broad` durchgeführt. Nach dieser zweiten Breitensuche sind in `besucht2` genau die Felder die nicht sicher sind. Somit lässt sich von hier mit der Ausgabe fortfahren.

3.3 Ausgabe

Die Ausgabe erfolgt in der Methode `schreiben`. Hier werden einfach alle Positionen abgegangen und untersucht, ob das entsprechende Feld eine Wand oder einen Ausgang beherbergt. Wenn das nicht der Fall ist, wird zwischen unsicheren Feldern, normalen nicht sicheren Feldern und sicheren Feldern unterschieden. Dabei sind die ersteren in der Menge `unsicher` gespeichert, die nicht sicheren sind die, die von der letzten Breitensuche nach `besucht2` zurückgegeben und `schreiben` übergeben wurden. Die restlichen Felder sind offensichtlich sicher. Das so erstellte Spielfeld wird als Text und als Bild ausgegeben. Dabei stehen im Textformat U für ein unsicheres und S für ein sicheres Feld. In der graphischen Ausgabe sind die unsicheren Felder rötlich, während die sicheren hellgrün sind.

4 Laufzeit

Die Laufzeit des Programms ist linear abhängig von der Größe des Labyrinths. Für den Beweis wird auf die Laufzeiten einzelner Operationen aus [1] zurückgegriffen. Wenn das Labyrinth die Größe $m \times n$ hat, hat das Programm folglich die Laufzeit $\mathcal{O}(m \cdot n)$. Das ergibt sich dadurch, dass jeder einzelne Teil, das Einlesen, die Breitensuche und die Ausgabe Linearzeit benötigt. Beim Einlesen bzw. Aufbau der Adjazenzliste wird m bzw. n mal die Methode `maxconnected` ausgeführt, jeweils mit einer Zeile der Länge n bzw. m. Die Methode `maxconnected` wiederum ist linear in Abhängigkeit der Zeilenlänge. Sie besteht nämlich aus einer for-Schleife über diese Zeile, in der nur konstante Operationen durchgeführt werden. Ausgenommen ist die Erweiterung der Adjazenzliste des Endknotens (Z. 79). Diese hat lineare Laufzeit, allerdings kommt wird jede Position der Liste höchstens einmal darin benutzt, deswegen ist auch die Laufzeit über alle Iterationen linear. Die Breitensuchen haben naturgemäß eine Laufzeit von $\mathcal{O}(|E| + |V|)$ [2]. Dabei hat der Graph maximal $4 \cdot n \cdot m$ Kanten, da jede Position aus maximal 4 Richtungen erreichbar ist und somit jeder Knoten maximal 4 eingehende Kanten hat. Folglich ist $\mathcal{O}(|E| + |V|) = \mathcal{O}(4 \cdot n \cdot m + n \cdot m) = \mathcal{O}(n \cdot m)$. In der Ausgabe wird jede Position abgegangen und, bis auf die Graphikausgabe, die hier vernachlässigt wird, ist jede Operation in konstanter Zeit zu bewältigen. Folglich hat auch die Ausgabe lineare Laufzeit. Demnach ist das ganze Programm als Aneinanderreihung linearer Algorithmen linear.

5 Erweiterungen

Als kleine Erweiterung werden zusätzlich zu den sicheren Feldern auch die unsicheren Felder angezeigt, also die Felder von denen es unmöglich ist, einen Ausgang zu erreichen. Mir kamen zusätzlich drei größere Erweiterungen für die Simulation in den Sinn.

Zum einen lässt sich überlegen, dass es schade ist, dass der Yamyam einfach über Türen hinweg geht, wenn sie nicht an einer Wand sind. Würde der Yamyam hingegen durch die Türen durchfallen, wenn sie über die Türen laufen, (dies wäre der Fall, wenn die Türen z.B. Falltüren wären). Das lässt sich im Programm während der Erstellung der Adjazenzliste realisieren. Dazu wird in `maxconnected` immer, wenn ein Ausgang erreicht wird, dieser als Ende und Anfang eine zusammenhängenden Stückes behandelt. Also werden die aktuell in `inarow`

gesammelten Positionen an die Adjazenzliste des Ausgangs übergeben, eine neue Liste angefangen und die bis zur nächsten Wand oder bis zum nächsten Ausgang gefundenen Positionen auch in die Adjazenzliste des Ausgangs gespeichert.

Eine andere Idee für eine Erweiterung wäre, dass sich der Yamyam nur eine bestimmte Entfernung (gemessen in Schritten nicht in Feldern) von seinem Ursprung entfernen kann, dabei kann er insgesamt so viel in diesem Umkreis gehen, wie er will. Dann kann er nur Ausgänge in seiner Nähe erreichen, aber auch nur in unsichere Bereiche in seiner Nähe gelangen. D.h. es gibt so potentiell mehr sichere Positionen, da unter Umständen weit entfernte unsichere Positionen, in denen der Yamyam verloren gehen kann, ignoriert werden. Dies wird durch eine Ganzzahl `togo` realisiert, dies wird nach dem ersten Rand der Breitensuche in die Queue eingefügt. Danach wird jedes mal, wenn die Suche auf eine Zahl anstatt einer Position stößt diese Zahl um eins erniedrigt und wieder angehängt. Die Breitensuche stoppt wenn die Zahl 0 wird, dadurch entspricht ein negatives `togo` einer normalen Breitensuche ohne Einschränkung. Dadurch werden bei der iterativen Breitensuchen Tiefen realisiert. Somit ergeben sich maximale Bewegungsradien des Yamyams.

Wenn der Yamyam sich nur eingeschränkt bewegen kann ergeben sich mehrere Typen von unsicheren Feldern. Es gibt die klassisch unsicheren Felder, von denen der Yamyam nicht mehr zu einem Ausgang gelangen kann, sobald er sich auf sie bewegt hat. Es gibt aber auch Felder, die als Startfelder zwar unsicher sind, aber auf denen der Yamyam sich zwischendurch auch befinden kann und sich später trotzdem zu einem Ausgang bewegen kann, da sein Bewegungsradius von seinem Startfeld gemessen wird. Diese Felder sind offensichtlich zwischen `togo` und $2 \cdot$ `togo` Schritte von den Ausgängen entfernt. Daher wird, um die unsicheren Felder zu finden, die erste Breitensuche mit einem Radius von $2 \cdot$ `togo` durchgeführt, somit sind nur die wirklich unbetretbaren Felder in `unsicher` gespeichert. Nach der zweiten Breitensuche von den unsicheren Feldern aus, diese mit Radius `togo`, werden nun potentiell noch Felder als sicher bzw. nicht-unsicher angezeigt, von denen sich zwar kein klassisch unsicheres Feld aber auch kein Ausgang erreichen lässt, das ist der zweite Typ der unsicheren Felder. Da sie zwischen `togo` und $2 \cdot$ `togo` von den Ausgängen entfernt befinden, lassen sie sich als der Unterschied zwischen den bei der ersten Breitensuche gefunden Feldern und den Feldern, die bei einer Breitensuche mit dem Radius `togo` gefunden werden, bestimmen. Sie werden in der Ausgabe orange bzw. mit W für ein zu weit entferntes Feld gekennzeichnet.

Zuletzt lässt sich überlegen, dass der Yamyam nicht komplett zufällig ist und, wenn er eine direkt erreichbare Tür sieht, auf diese zugeht. Er hat also quasi einen “Riecher” für Türen. Dies lässt sich realisieren, wenn der Breitensuche noch eine Menge an Stoppern übergeben wird, über die die Breitensuche nicht hinwegkann. Dazu wird die Menge an Positionen übergeben, von denen sich direkt ein Ausgang erreichen lässt, also alle Nachbarn der Ausgänge im Graphen. Dadurch wird unterbunden, dass der Yamyam sich von diesen Feldern aus auf unsichere Felder bewegt und nicht auf den Ausgang. Dass die Felder nicht unsicher sind, wird schon dadurch garantiert, dass sie bei der ersten Breitensuche als direkte Nachbarn zu Ausgängen gefunden werden.

Alle drei Parameter, ob der Yamyam durch die Tür fallen soll, wie weit er sich bewegen

darf und ob er Türen in der Nähe erkennt, werden zu Programmstart abgefragt. Dabei ist die maximale Entfernung, die der Yamyam zurücklegen kann, der Initialwert für `togo`. Wenn diesem der Wert -1 übergeben wird, ist es stets kleiner 0 und deswegen wird die Tiefensuche nie abgebrochen, was der normalen Fragestellung ohne maximale Entfernung entspricht. Besonders interessant bzw. gut für den Yamyam ist die Kombination daraus, dass er durch Türen fallen kann und Ausgänge riechen kann, dann werden sehr viele neue Felder sicher.

6 Beispiele

Die auf der Website gegeben Beispiele ergeben folgende Outputs. Dabei ist die Ausführzeit kaum merklich.

```
yamyams0.txt:
############
#U#SSSSSS#U#
#   ##     #
#   #SSSE#U#
#       ##U#
#U       UU#
#U#E SSSE###
############

Die Ausgabe wurde nach yamyams0.out geschrieben.
Das Bild wurde nach yamyams0.png geschrieben.

yamyams1.txt:
########################
#UUUU#UU U#UUUU#UUUUUUU#
#UUUU#UU U#UUUU#UUUUUUU#
#UUUUUUU U#UUUUUUUUEUU##
#UUUUUUU UUUUUUUUUUUUU#
#UUUUUUU UUUUUU#UUUUUUU#
#UUUU##  ##UUUU#UUUUUUU#
#UUUU#UU U#UUUU#UUU#UUU#
#UUUU#UUEU#UUUU#UUUUUUU#
#   E######UUUU#UUUUUUU#
#UUUU#UUUU#UUUU#UUUUUUU#
#UUUU#UUUU#UUUU#UUUUUUU#
########################

Die Ausgabe wurde nach yamyams1.out geschrieben.
Das Bild wurde nach yamyams1.png geschrieben.

yamyams2.txt:
```

```
##################################################
# #UUU#               #         #UUU#            #
# ###U# ########### # # ##### # ###U# ### ####### #
#   #U#     #UUU#   # #   #U# #   #U#   # #       #
### #U##### #U#U# ### ### #U# ### #U### # ####### #
#E# #UUUUUU UU#U# # # #UU UU# #U# #     #       # #
#S# ###U######### # # #####U# #U# #U########### # #
#S#   #U#U#     # # #     #U# #U# #UUUUUUU#   # # #
#S### #U#U# ### # # ##### ### #U# #####U### # # # #
#SSS# #UUU#   # # #     #   # #U#     #U#   #   # #
###S# ###U### # # # ####### # #U##### ### ####### #
#   #   #U#   #   #   #     # #UUUUU#   # #       #
# #S### ### ####### # # ##### #U#U##### # # #######
# #SSS#   #       # #   #     #U#U#UUU#   # #ES SS#
# ####### ### ### # ##### # ###U###U#U##### ### ###
# #     #   #   # #     # # #UUUUUUU#UUUUU#   #   #
# # ### ### ##### ##### # # #U#######U###U### ### #
# #   #   # #   #   #   #E# #UUU#UUU#U#U#UUU#     #
# ### # ### # # # # # ##### ###U#U###U#U###U##### #
#     #     # # # # # #     #UUU#U#UUU#UUUUU#     #
# ########## # ### # # #####U###U#U######### #####
# #           #   # #   #UUU#UUUUU#UUUUU#UUU#   #U#
# ##### ######### # #####U#U#####U#####U#U#U### #U#
#       #           #UUUUU#UUUUUUUUUU#UUU#UUUU UU#
##################################################
```

```
Die Ausgabe wurde nach yamyams2.out geschrieben.
Das Bild wurde nach yamyams2.png geschrieben.
```

```
yamyams3.txt:
####################
#UUUU#UUUUUUUUUUUU#
##UUUUUUUUUUUUUUUU#
#UUUUUUUUU#UUUUUUU#
#UUUUUUUUUUUUUUUUU##
#UUUUUUUUUU###UUUU#
#UUUU#UUUUUUUUUUUU#
#UUUUUUUUUUUUUUUUU#
#UUUUUUUUUUUUUUUUU#
#UUUUUUU#UUUUUUUUEU#
##UUUUU#UUUUUUUUUU#
#UUU###UUUU#UUUUU#
#UUU#UUUUUUUUUUUU#
#UUU#UUUUUUUUUUUU#
####################
```

```
Die Ausgabe wurde nach yamyams3.out geschrieben.
Das Bild wurde nach yamyams3.png geschrieben.

yamyams4.txt:
####################
#UUUU#UUUUUU#UUUUU#
##UUU#UUUUUUUUUUU#
#UUUUUUUUU##UUUUU#
#UUUU#UUUUUUUUUUU#
#UUUU#UUUUU###UUU#
#UUEU#####UUUUUUU#
#UUUUUU#UUUUUUUUU#
#UUUUUU#UUUUUUUUU#
#UUUUUU#####UUUUU#
##UUUUU#UUUUUUUUU#
#UUU####UUUU######
#UUU#UUUUUUUUUUU#
#UUU#UUUU#UUUUUUU#
####################

Die Ausgabe wurde nach yamyams4.out geschrieben.
Das Bild wurde nach yamyams4.png geschrieben.

yamyams5.txt:
########################
#              #UUUU UU#
#              #U###   #
#              ###U  UU#
#      ### ##          #
#    ###    #          #
#    #    # #    #U  UU#
#    #  ### #    #U  UU#
#### #      #   ###    #
#E       ####    #     #
####  #        # #  #UU#
#     #     ###E    #UU#
#     #      #    ###UU#
#     ###    #         #
#   E           ###    #
#           ##    #    #
#####   #    #    #    #
#UUU#   ##   #         #
#UUU#      #    #      #
#UUU# #### ###  #      #
```

```
#UUUU UUU#       ###    #
#UUUU UUUU      EUUU  UU#
#UUUU UUUU       UUU  UU#
########################
```

```
Die Ausgabe wurde nach yamyams5.out geschrieben.
Das Bild wurde nach yamyams5.png geschrieben.
```

```
yamyams6.txt:
######
#E##S#
#SSSS#
##SSS#
#SS#S#
#S#SS#
#SSSS#
##SSS#
#SS#S#
#S#SS#
#SSSS#
##SSS#
#SS#S#
#S#SS#
#SSSS#
######
```

```
Die Ausgabe wurde nach yamyams6.out geschrieben.
Das Bild wurde nach yamyams6.png geschrieben.
```

Einige weitere interessante Beispiele für die Erweiterungen sehen folgendermaßen aus:

```
yamyams0.txt, riechen;
############
#U#SSSSSS#U#
#  S##  S  #
#  S#SSSE#U#
#  S    ##U#
#U S    SUU#
#U#ESSSSE###
############
```

```
Die Ausgabe wurde nach yamyams0,riechen.out geschrieben.
Das Bild wurde nach yamyams0,riechen.png geschrieben.
```

```
yamyams1.txt, fallen;
########################
```

```
#SSSS#    #UUUU#SSSSSSS#
#SSSS#    #UUUU#SSSSSSS#
#         #        ESS##
#SSSS           SSSSSSS#
#              #SSSSSSS#
#SSSS##  ##UUUU#SSSSSSS#
#SSSS#S  S#UUUU#SSS#SSS#
#SSSS#S ES#UUUU#SSSSSSS#
#SSSE#####UUUU#SSSSSSS#
#SSSS#UUUU#UUUU#SSSSSSS#
#SSSS#UUUU#UUUU#SSSSSSS#
########################
```

```
Die Ausgabe wurde nach yamyams1,fallen.out geschrieben.
Das Bild wurde nach yamyams1,fallen.png geschrieben.
```

```
yamyams2.txt, max13;
```

```
#####################################################
#W#UUU#WWWWWWWWWWWWW#UUUUUU UU#UUU#             #
#W###U#W###########W#W#U##### #U###U# ### ####### #
#WWW#U#WWWWW#UUU#WWW#W#UUU#U# #UUW#U#   # #       #
###W#U######W#U#U#W###W###U#U# ###W#U###W# ####### #
#E#W#UUUUUUWUU#U#W# #W#UUUUU# #U#W#WWWWW#       # #
#S#W###U##########W# #W######U# #U#W#U###########W# #
#S#WWW#U#U#UUUUU#W# #WWWWW#U# #U#W#UUUUUUU#WWW#W# #
#S###W#U#U#U###U#W# #########  #U#W#####U###W#W#W# #
#SSS#W#UUU#UUU#U#W#     #WWW# #U#WWWWW#U#WWW#WWW# #
###S#W###U###U#U#W# ########W# #U#####W###W###### #
#   #WWW#U#UUU#WWW#   #WWWWW# #UUUUU#WWW#W#       #
# #S###W###U####### #S#W##### #U#U#####W#W# #######
# #SSS#WWW#UUUUUUU# # WW#     #U#U#UUU#WWW# #ES SS#
# #######W###U###U# ##### # ###U###U#U##### ### ###
# #     #WW #UUU#U#     # # #UUUUUUU#UUUUU#   #   #
# # ### ### #####U##### # # #U#######U###U### ### #
# #   #   # #UUU#UUU#   #E# #UUU#UUU#U#U#UUU#     #
# ### # ### #U#U#U#U# ##### ###U#U###U#U###U##### #
#     #     #U#U#U#U# #     #UUU#U#UUU#UUUUU#     #
# ###########U#U###U# # #####U###U#U######### #####
# #UUUU UUUUUU#UUU#U#   #UUU#UUUUU#UUUUU#UUU#   #U#
# ##### #########U#U#####U#U#####U#####U#U#U### #U#
#       #UUUUUUUUUUU#UUUUU#UUUUUUUUUUU#UUU#UUUU UU#
#####################################################
```

```
Die Ausgabe wurde nach yamyams2,max13.out geschrieben.
Das Bild wurde nach yamyams2,max13.png geschrieben.
```

```
yamyams4.txt, fallen, riechen;
####################
#  S #UUUUUU#UUUUU#
## S #UUUUUUUUUUUU#
#  S      ##UUUUUU#
#  S #UUUUUUUUUUUU#
#  S #UUUUU###UUUU#
#SSES#####UUUUUUUU#
#  S   #UUUUUUUUUU#
#  S   #UUUUUUUUUU#
#  S   #####UUUUUU#
## S   #UUUUUUUUUU#
#S S####UUUU#######
#S S#UUUUUUUUUUUUU#
#S S#UUUU#UUUUUUUU#
####################

Die Ausgabe wurde nach yamyams4,fallen,riechen.out geschrieben.
Das Bild wurde nach yamyams4,fallen,riechen.png geschrieben.

yamyams5.txt, fallen;
########################
#SSSSSSSSSSSSSS#UUUU UU#
#SSSSSSSSSSSSSS#U###   #
#SSSSSSSSSSSSSS###U  UU#
#SSSSSS###S##         #
#SSSS###SSSS#         #
#SSSS#SSSS#S#    #U  UU#
#SSSS#SS###S#    #U  UU#
####S#SSSSSS#   ###    #
#ESSS SSS####    #     #
####  #SSS  SSS# #  #UU#
#     #     ###E    #UU#
#     #SSS  S#    ###UU#
#     ###S  S#         #
#   E    S  SS S###    #
#          ##    #    #
#####   #S  S#    #    #
#UUU#   ##   #         #
#UUU#      #SS S#      #
#UUU# #### ###  #      #
#UUUU UUU#      ###    #
#              E       #
#UUUU UUU            UU#
```

```
######################

Die Ausgabe wurde nach yamyams5,fallen.out geschrieben.
Das Bild wurde nach yamyams5,fallen.png geschrieben.
```

Die erstellten graphischen Darstellungen der Ergebnisse (Bilder) und weitere Beispiele für die Erweiterungen befinden sich im Abgabeordner. Ein Einfügen der Bilder in die Dokumentation ist aufgrund der interessanten und stark unterschiedlichen Formate der Labyrinthe weniger sinnvoll.

7 Quellen

Literatur

[1] https://www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt

[2] https://en.wikipedia.org/wiki/Breadth-first_search

8 Quellcode

In der Abgabe liegen zusätzlich zum Programm in *aufgabe3.py* noch die Programme *alle.py* und *interessant.py*, die Beispielausgaben aus den vorliegenden Beispieldateien erstellen, bei.

```
1  #!/usr/bin/env python3
2  from collections import deque
3  try:
4      from PIL import Image
5      bildout = True
6  except:
7      print("Pillow ist nicht installiert.")
8      print("Die Bildausgabe ist nun leider unmöglich.")
9      bildout = False
10
11 # Einige Konstanten um die Lesbarkeit zu verbessern
12
13 AUSGANG = 'E'
14 WAND = '#'
15 FREI = ' '
16 UNSICHER = 'U'
17 SICHER = 'S'
18 ZUWEIT = 'W'
19
20
21 if bildout:
```

```
22      LINE = 1
23      PIXEL = (20, 20)
24
25      try:
26          # Eigentlich sollten die Bilder da sein...
27          door = Image.open('pictures/door.png')
28          wall = Image.open('pictures/wall7.png')
29          INTERIM = Image.open('pictures/interim.png')
30      except:
31          #... aber man weiß ja nie
32          print("Eins oder mehrere Bilder wurden nicht gefunden.")
33          print("Fahre mit Ersatzbildern fort...")
34          door = Image.new('RGB', PIXEL, (0x0, 0xb3, 0xb3))
35          wall = Image.new('RGB', PIXEL, (0xb3, 0xb3, 0xb3))
36          # Zur Verbindung zwei nebeneinanderstehender Wände
37          INTERIM = Image.new('RGB', (LINE, PIXEL[1]), (0xb3, 0xb3, 0xb3))
38
39      BILDER = {WAND: wall,
40                FREI: Image.new('RGB', PIXEL, (0xff, 0xff, 0xff)),
41                AUSGANG: door,
42                SICHER: Image.new('RGB', PIXEL, (0x40, 0xff, 0x40)),
43                UNSICHER: Image.new('RGB', PIXEL, (0xff, 0x40, 0x40)),
44                ZUWEIT: Image.new('RGB', PIXEL, (0xff, 0xa5, 0x00))}
45
46
47  def einlesen(filename, fallen):
48      ''' Hier wird das Feld eingelesen und die Adjazenzliste aufgebaut. '''
49      with open(filename) as f:
50          lines = [i.rstrip() for i in f]
51      for i in range(len(lines)):  # Hier werden die Zeilen durchgegangen...
52          maxconnected(lines[i], y=i, fallen=fallen)
53      for i in range(len(lines[0])):  # ... und hier die Spalten
54          line = [lines[j][i] for j in range(len(lines))]
55          maxconnected(line, x=i, fallen=fallen)
56
57      global size
58      size = (len(lines), len(lines[0]))
59
60
61  def maxconnected(oneline, y=None, x=None, fallen=False):
62      ''' Hier werden die längsten zusammenhängenden Stücke freier
63      Felder gefunden '''
64      start = None
65      end = None
66      inarow = []
```

```
    for i in range(len(oneline)):
        zeichen = oneline[i]
        if zeichen == WAND:
            if end is not None:  # Dann hat gerade ein Stück aufgehört
                if not feld[end] == AUSGANG:
                    # Damit end keine Kante auf sich selbst hat
                    inarow.pop()
                if end not in adjlist:
                    adjlist.update({end: []})
                adjlist[end].extend(inarow)
                end = None
                start = None
                inarow = []
            continue
        # Damit die Methode für Spalten und Zeilen nutzbar ist
        if y is not None:
            pos = (y, i)
        else:
            pos = (i, x)
        if start is None:
            # Es beginnt demnach grade ein neues Stück
            start = pos
            if start not in adjlist:
                adjlist.update({start: []})
        end = pos
        feld.update({pos: zeichen})
        # Mit einem Dictionary lässt sich leicher arbeiten
        if zeichen == AUSGANG:
            # Auf einen Ausgang verweisen keine Kanten, da ein Weg endet,
            # wenn er zu einem Ausgang kommt
            if fallen:
                # Hier wird simuliert, dass der Yamyam durch die Tür fällt
                if pos not in adjlist:
                    adjlist.update({pos: []})
                adjlist[pos].extend(inarow)
                start = pos
                inarow = []
            ausgaenge.add(pos)
        else:
            inarow.append(pos)
            if not pos == start:
                # Damit start keine Kante auf sich selbst hat
                adjlist[start].append(pos)
```

```
112 def schreiben(filename, besucht, unsicher, zuweit):
113     ''' Die Ausgabemethode '''
114     klein = size[0] * size[1] < 10000000
115     klein = klein and bildout
116     # Das Bild sollte nicht zu groß werden
117     if klein:
118         bild = Image.new('RGB',
119                          (size[1] * (PIXEL[0] + LINE) + LINE,
120                           size[0] * (PIXEL[1] + LINE) + LINE),
121                          (0xb3, 0xb3, 0xb3))
122     out = []
123     for y in range(size[0]):
124         last = ''
125         for x in range(size[1]):
126             # Da die Wände nicht abgespeichert sind
127             zeichen = feld.get((y, x), WAND)
128             if zeichen == WAND:
129                 pass
130             elif zeichen == AUSGANG:
131                 pass
132             elif (y, x) in zuweit:
133                 zeichen = ZUWEIT
134             elif (y, x) in unsicher:
135                 zeichen = UNSICHER
136             elif (y, x) not in besucht:
137                 zeichen = SICHER
138             out.append(zeichen)
139 
140             if klein:
141                 bild.paste(BILDER[zeichen],
142                            (x * (PIXEL[0] + LINE) + LINE,
143                             y * (PIXEL[0] + LINE) + LINE))
144                 if last == WAND and zeichen == WAND:
145                     bild.paste(INTERIM,
146                                (x * (PIXEL[0] + LINE),
147                                 y * (PIXEL[1] + LINE) + LINE))
148 
149             last = zeichen
150 
151         out.append('\n')
152 
153     out = ''.join(out)
154     print(out)
155     with open(filename + '.out', 'w') as f:
156         f.write(out)
```

```
157     print('Die Ausgabe wurde nach %s.out geschrieben.' % (filename))
158     if klein:
159         bild.save(filename + '.png')
160         print('Das Bild wurde nach %s.png geschrieben.' % filename)
161
162
163 def main(filename, fallen=False, togo=-1, riechen=False):
164     ''' Der Beginn allen Übels '''
165     global feld, adjlist, ausgaenge
166     feld = {}
167     adjlist = {}
168     ausgaenge = set()
169     einlesen(filename, fallen)
170     tmp = set()
171     if riechen:
172         tmp.update(*(adjlist.get(i, set()) for i in ausgaenge))
173     # Die erste Breitensuche, von den Ausgängen aus
174     besucht = broad(ausgaenge, 2*togo)
175     unsicher = set.difference(set(feld), besucht)
176
177     # Die zweite Breitensuche, von den unsicheren Feldern aus
178     besucht2 = broad(unsicher, togo, tmp)
179     if togo > -1:
180         umfeld = broad(ausgaenge, togo)
181         entfernt = besucht.difference(umfeld)
182     else:
183         entfernt = set()
184     if filename.endswith('.txt'):  # Niemand mag .txt.out
185         filename = filename[:-4]
186     if fallen:
187         filename += ',fallen'
188     if not togo == -1:
189         filename += ',max%s'%togo
190     if riechen:
191         filename += ',riechen'
192     schreiben(filename, besucht2, unsicher, entfernt)
193
194 def broad(start, togo, stopper=set()):
195     ''' Eine iterative Breitensuche ohne Ziel '''
196     besucht = set()
197     rand = deque()
198     for i in start:
199         rand.append(i)
200         besucht.add(i)
201     togo -= 1
```

```
202     rand.append(togo)
203     while len(rand) > 0:
204         current = rand.popleft()
205         if isinstance(current, int):
206             # Nach jeder Tiefe wird eine Zahl eingefügt um so
207             # eine maximale Tiefe bestimmen zu können
208             if current == 0 or len(rand) == 0:
209                 return besucht
210             current = rand.popleft()
211             togo -= 1
212             rand.append(togo)
213         for i in adjlist.get(current, set()):
214             if i not in besucht and i not in stopper:
215                 besucht.add(i)
216                 rand.append(i)
217     return besucht
218 
219 
220 if __name__ == '__main__':
221     filename = input("Datei:\n")
222     fallen = input("Kann der Yamyam durch Türen fallen? (y/n)\n") == "y"
223     togo = int(input("Wie weit kann der Yamyam maximal gehen? (-1 =
    beliebig)\n"))
224     riechen = input("Kann der Yamyam Türen aus der Nähe erkennen? (y/n)\n")
    == "y"
225     main(filename, fallen = fallen, togo = togo, riechen = riechen)
```