Dario Porcaro

Fabrique - et jeux - de labyrinthes auto-générés et auto-transformés

Lycée Louis Barthou Pau - MP 2023-2024 - TIPE

Avant Propos

Python est un langage impératif permettant l'usage de la récursivité, OCaml un langage fonctionnel permettant l'usage de la programmation impérative. En classe Préparatoire, on a essentiellement étudié Ocaml. Néanmoins, tous mes programmes sont rédigés en Python, d'une part parce qu'ils sont strictement impératifs (excepté un seul cas de possible appel récursif), d'autre part compte tenu de la grande flexibilité de ce langage et de ses nombreux modules disponibles. Pour autant, il serait tout à fait possible de rendre ces programmes récursifs.

Entre les versions 1 et 2 des générateurs et des solveurs, le principe de l'algorithme reste le même, seul son implémentation change. Dans les v1, on parcourt systématiquement l'ensemble des sommets présents dans le graphe, alors que dans les v2, on ne parcourt que les sommets sur lesquels on va agir. De plus, en terme d'application, le générateur a été traduit en C# pour pouvoir être utilisé sous Unity (qui m'était déjà familier en visual scripting), ce qui m'a donc permis de commencer l'apprentissage de ce langage.

Voici donc l'ensemble des ces programmes.

Sommaire

1 - Convertisseur de Labyrinthe en Graphe

```python
from PIL import Image
import numpy as np
import pickle


NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))
NAMENR=str(input("Nom du fichier contenant les informations relatives aux portails : "))
NameFin=str(input("Nom du fichier du graphe à sauvegarder : "))

def ckoa2(pix):     #Reconnaissance mur/chemin.
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "chemin"
    elif pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "chemin"
    elif 100>pix[0]:
        return "mur"
    return "chemin"

def converter(g,tab): #Crée un graphe g sous forme de dictionnaire où les clés sont les
coordonnées des points "chemin" et les valeurs sont leurs listes de points "chemin" adjacents.
    temp=[]
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            if ckoa2(tab[i][j])=="chemin":
                temp=[]
                if i>0 and ckoa2(tab[i-1][j])=="chemin":
                    temp.append((i-1,j))
                if i<len(tab)-1 and ckoa2(tab[i+1][j])=="chemin":
                    temp.append((i+1,j))
                if j>0 and ckoa2(tab[i][j-1])=="chemin":
                    temp.append((i,j-1))
                if j<len(tab[0])-1 and ckoa2(tab[i][j+1])=="chemin":
                    temp.append((i,j+1))
                g[(i,j)]=temp
                temp=[]
    return g

def enrichiG(g,NAMENR):          #Ajoute les données relatives aux portails dans le graphe.
    file = open(NAMENR, "rb")
    enr = pickle.load(file)
    for c in enr:
        for e in enr[c]:
            if not e in g[c]:
                g[c].append(e)
    file.close()
    return g

def saveG(g,NameFin):
    FinalFile = open(NameFin, "wb")
    pickle.dump(g,FinalFile)
```

```
51      FinalFile.close()
52
53  g={}
54  if not NAME=="":
55      im=Image.open(NAME)
56      tab=np.array(im)
57      im.close()
58      g=converter(g,tab)
59  if not NAMENR=="":
60      enrichiG(g,NAMENR)
61
62  saveG(g,NameFin)
```

2 - Convertisseur de Graphe en Labyrinthe

```python
import numpy as np
from PIL import Image
from copy import *
from time import *
import imageio
import pickle


NAMEFILE=input("Nom du fichier pickle contenant le graphe à 'labyrinthiser' (avec l'extension) : ")
NAMSAV=input("Nom du fichier où sera sauvegardée l'image du labyrinthe créé : ")
NAMSAVTXT=input("Nom du fichier où sera enregistré le dictionnaire des portails : ")


def taille(gres):
    t=0
    for c in g:
        if t<max(len(gres[c][0]),len(gres[c][1])):
            t=max(len(gres[c][0]),len(gres[c][1]))
    return t

def entsortdic(g,INIT):   # Le format est {sommets: (liste des sommets desquels on peut arriver, liste des sommets vers lesquels on peut aller)}.
    gres={}
    nxt=g[INIT]
    cur=[]
    for c in g:
        gres[c]=([],[])
    for c in g:
        tmp=g[c]
        for i in tmp:
            gres[i][0].append(c)
        gres[c]=gres[c][0],deepcopy(tmp)
        nxt=nxt+tmp
    return gres

def motif(entsort):
    mot=np.full((6, 4, 3), [0, 0, 0], dtype=np.uint8)
    mot[5][2]=[254,254,254]
    return mot

def dicoord(dictidsom, g):
    dicaret={}
    for som in g:
        dicaret[(5,4*dictidsom[som]+2)]=[]
    for som in g:
        adj=g[som]
        for ar in adj:
            dicaret[(5,4*dictidsom[som]+2)].append((5,4*dictidsom[ar]+2))
    return dicaret
```

```
51
52  def GEN(g,NAMSAV,NAMSAVTXT):
53      INIT=next(iter(g))
54      gres=entsortdic(g,INIT)
55      Imax=taille(gres)+10
56      Jmax=len(g)*4
57      tab=np.full((Imax, Jmax, 3), [0, 0, 0], dtype=np.uint8)
58      n=0
59      dictidsom={}
60      id=0
61      for c in gres:
62          dictidsom[c]=id
63          id+=1
64          mot=motif(gres[c])
65          for i in range(len(mot)):
66              for j in range(len(mot[0])):
67                  tab[i][j+n*4]=mot[i][j]
68          n+=1
69      coord=dicoord(dictidsom,g)
70      imageio.imwrite(NAMSAV+".png", tab)
71      tf = open(NAMSAVTXT, "wb")
72      pickle.dump(coord,tf)
73      tf.close()
74
75
76  file = open(NAMEFILE, "rb")
77  g = pickle.load(file)
78  file.close()
79  GEN(g,NAMSAV,NAMSAVTXT)
```

3 - Convertisseur de Graphe de Labyrinthe en Labyrinthe - "Reformer"

```python
import numpy as np
from PIL import Image
import imageio
import pickle


NAMEFILE=input("nom du fichier pickle contenant le graph du jeu a 'labyrinthiser' (avec l'extension)      ")
NAMSAV=input("nom du fichier ou sera sauvegardée l'image du labyrinthe créé      ")


#Le "reformeur" prend en entrée un fichier graphe généré par le script ayant l'effet inverse appliqué sur un labyrinthe sans portail.

def taille(g):
    i,j=0,0
    for c in g:
        if i<c[0]:
            i=c[0]
        if j<c[1]:
            j=c[1]
    return i,j

def GEN(NAMEFILE,NAMSAV,Ie=0,Je=0,Is=None,Js=None):
    file = open(NAMEFILE, "rb")
    g = pickle.load(file)
    file.close()
    Imax,Jmax=taille(g)
    if Is==None:
        Is=Imax
    if Js==None:
        Js=Jmax
    ta=np.full((Imax+1, Jmax+1, 3), [0, 0, 0], dtype=np.uint8)
    for c in g:
        i,j=c
        ta[i][j][0]=255
        ta[i][j][1]=255
        ta[i][j][2]=255
    ta[Ie][Je][0]=255
    ta[Ie][Je][1]=0
    ta[Ie][Je][2]=0
    ta[Is][Js][0]=0
    ta[Is][Js][1]=255
    ta[Is][Js][2]=0
    imageio.imwrite(NAMSAV+".png", ta)

GEN(NAMEFILE,NAMSAV)
```

4 - Solveur v1

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio

print("WARNING /!\ ce script détecte également les cycles, il n'est utilisable que sur des labyrinthes du type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce programme donne toute les solutions possibles")


SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)"))
RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)"))
SAVTEMP=SAVTEMPin=="True"
RETOURS=RETOURSin=="True"
MAXSAVEDINTERETAP=int(input("Maximum d'étapes intermédiaires sauvegardées en images : "))
NOMFINAL=str(input("Nom du fichier de sauvegarde du labyrinthe résolu : "))
NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))


t0=time.time()
im=Image.open(NAME)
tab=np.array(im)

#Recherche de l'entrée et de la sortie.
azert=0
for i in range(len(tab)):
    for j in range(len(tab[0])):
        if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
            if azert>0:
                azert=2
                Is,Js=i,j
                break
            else:
                azert=1
                Ie,Je=i,j
    if azert==2:
        break


tab[Ie][Je]=[255,0,0]     #Marquage de l'entrée et de la sortie.
tab[Is][Js]=[0,255,0]

def ckoa(pix):     #Reconnaissance de mur/chemin/entrée/sortie.
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "start"
    if pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "end"
    if 100>pix[0]:
        return "mur"
    return "chemin"
```

```
53
54  n=len(tab)
55  m=len(tab[0])
56
57
58
59  def converter(tab): #Crée un graphe g sous forme de dictionnaire, où les clés sont les
60  coordonnées des points et les valeurs sont les listes des sommets adjacents.
61      g={}
62     temp=[]
63      for i in range(len(tab)):
64          for j in range(len(tab[0])):
65              temp=[]
66              if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
67  1][j])=="start"):
68                  temp.append([i-1,j])
69              if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
70  ckoa(tab[i+1][j])=="start"):
71                  temp.append([i+1,j])
72              if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
73  ckoa(tab[i][j-1])=="start"):
74                  temp.append([i,j-1])
75              if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
76  ckoa(tab[i][j+1])=="start"):
77                  temp.append([i,j+1])
78              if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
79  ckoa(tab[i][j])=="start"):
80                  g[(i,j)]=temp
81                  temp=[]
82      print("completion de la convertion en " +str(floor(time.time()-t0))+" secondes")
83      return g
84
85  def solv(g):
86      a=0
87      at=time.time()
88      z = True
89      while z:                    #Tant que le labyrinthe n'est pas résolu.
90          z = False
91          to_delete = []
92          for c in g:               #On supprime les chemins où il n'y a qu'un seul voisin
93  adjacent ---> cul-de-sac = mur.
94              if len(g[c]) < 1:
95                  if not (c == (Ie, Je) or c == (Is, Js)):
96                      to_delete.append(c)
97                      z = True
98              if len(g[c]) ==1:
99                  if not (c == (Ie, Je) or c == (Is, Js)):
100                     z = True
101                     i, j = c
102                     x = [i, j]
103                     to_delete.append(c)
104                     k=0
105                     for pos in g[c]:
106                         t=pos[0],pos[1]
107                         if t in g:
108                             while x in g[t]:
109                                 g[t].remove(x)
```

```
110         for key in to_delete:
111             del g[key]
112         a+=1
113         if RETOURS:
114             print("complétion de l'étape " +str(a)+" en " +str(floor(time.time()-t0))+"
115 secondes")
116             at=time.time()
117         if SAVTEMP:
118             at=time.time()
119             savetap(g,a)
120     print("résolu en " +str(floor(time.time()-t0))+" secondes")
121     return g
122 
123 def savetap(g,a):
124     ta=deepcopy(tab)
125     for c in g:
126         i,j=c
127         ta[i][j]=[0,0,255]
128     for i in range(len(tab)):
129         for j in range(len(tab[0])):
130             if ckoa(tab[i][j])=="mur":
131                 ta[i][j]=[0,0,0]
132     ta[Ie][Je]=[255,0,0]
133     ta[Is][Js]=[0,255,0]
134     if a<MAXSAVEDINTERETAP:
135         imageio.imwrite("etape_"+str(a)+".png", ta)
136     else:
137         imageio.imwrite("etape_"+str(MAXSAVEDINTERETAP)+".png", ta)
138     print("étape "+str(a)+" rendue en " +str(floor(time.time()-t0))+" secondes")
139 
140 def end(g):       #Applique la solution sur le tableau de l'image.
141     ta=deepcopy(tab)
142     for c in g:
143         i,j=c
144         ta[i][j]=[0,0,255]
145     ta[Ie][Je]=[255,0,0]
146     ta[Is][Js]=[0,255,0]
147     print("solution rendue en " +str(floor(time.time()-t0))+" secondes")
148     return ta
149 
150 def finitions():
151 ##    for i in range(len(tab)):
152 ##        for j in range(len(tab[0])):
153 ##            if ckoa(tab[i][j])=="mur": #Repasse les murs en noir : artefact datant
154                                                 des problèmes de compression JPG, n'a
155 rien résolu.
156 ##                ta[i][j]=[0,0,0]
157     imageio.imwrite(NOMFINAL+".png", ta)
158     print("solution enregistree en " +str(floor(time.time()-t0))+" secondes")
159     (Image.fromarray(ta)).show()
160 
161 
162 g=converter(tab)  #Lancement de la résolution.
163 solv(g)
164 ta=end(g)
165 finitions()
```

5 - Solveur v2

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
import threading
import pickle
print("WARNING /!\ ce script détecte également les cycles, il n'est utilisable que sur des labyrinthes du type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce programme donne toute les solutions possibles")


SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)"))
RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)"))
SAVTEMP=SAVTEMPin=="True"
RETOURS=RETOURSin=="True"
MAXSAVEDINTERETAP=int(input("Maximum d'étapes intermédiaires sauvegardées en images : "))
INRVALCAP=int(input("Intervalle d'étape entre chaque image intermédiaire : "))
NOMFINAL=str(input("Nom du fichier de sauvegarde du labyrinthe résolu : "))
NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))
NAMENR=str(input("Nom du fichier contenant les informations relatives aux portails : "))


t0=time.time()
im=Image.open(NAME)
tab=np.array(im)
print("chargé")
#Recherche de l'entrée et de la sortie.

#On recherche deux façons de couvrir les deux modèles d'enregistrement.
azert=0
for i in range(len(tab)):
    for j in range(len(tab[0])):
        if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
            if azert>0:
                azert=2
                Is,Js=i,j
                break
            else:
                azert=1
                Ie,Je=i,j
    if azert==2:
        break

for i in range(len(tab)):
    for j in range(len(tab[0])):
        if tab[i][j][0]==255 and tab[i][j][1]==0 and tab[i][j][2]==0:
            Ie,Je=i,j
        if tab[i][j][0]==0 and tab[i][j][1]==255 and tab[i][j][2]==0:
```

```
            Is,Js=i,j

tab[Ie][Je]=[255,0,0]     #Marquage de l'entrée et de la sortie.
tab[Is][Js]=[0,255,0]
print('entree-sortie trouvee')
def ckoa(pix):     #Reconnaissance de mur/chemin/entrée/sortie.
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "start"
    if pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "end"
    if 100>pix[0]:
        return "mur"
    return "chemin"


def converter(tab): #Crée un graphe g sous forme de dictionnaire, où les clés sont les
coordonnées des points et les valeurs sont les listes des sommets adjacents.
    g={}
    temp=[]
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            temp=[]
            if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
1][j])=="start"):
                temp.append((i-1,j))
            if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
ckoa(tab[i+1][j])=="start"):
                temp.append((i+1,j))
            if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
ckoa(tab[i][j-1])=="start"):
                temp.append((i,j-1))
            if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
ckoa(tab[i][j+1])=="start"):
                temp.append((i,j+1))
            if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
ckoa(tab[i][j])=="start"):
                g[(i,j)]=temp
                temp=[]
    if not NAMENR=="":
        file = open(NAMENR, "rb")
        enr = pickle.load(file)
        for dep in enr: #enrichissement de g par les portailes
            for dest in enr[dep]:
                g[dep].append(dest)
    print("completion de la convertion en " +str(floor(time.time()-t0))+" secondes")
    return g

#L'algorithme utilisé est le suivant : "Si tu es un cul-de-sac, alors tu es un mur". On
applique cette transformation à chaque point du labyrinthe jusqu'à ce que le labyrinthe soit
résolu.
#Les chemins ne menant nulle part régresseront et disparaîtront jusqu'à ce qu'il ne reste que
le bon chemin.

def solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab):
    a=0
    b=0
    cur,nxt=[],[]     #cur[ent] et n[e]xt contiennent les chemins à supprimer respectivement à
```

```
108 l'instant courant et à l'instant suivant.
109     for i in g:
110         if len(g[i])<2 and not (i == (Ie, Je) or i == (Is, Js)):  #Création de l'état initial
111 des éléments à détruire.
112             nxt.append(i)
113     while not nxt==[]:      #Tant qu'il reste des éléments à détruire.
114         cur,nxt=nxt,[]       #On inverse les listes de sorte que le suivant devienne le courant
115 et on vide le suivant.
116         for i in cur:
117             if i in g:
118                 t=g[i]
119                 for j in t:        #On indexe les sommets adjacents qui deviennent "à détruire"
120 dans la liste "suivant" si on retire le sommet i.
121                     l=g[j]
122                     l.remove(i)
123                     g[j]=l
124                     if len(l)<2 and not (i == (Ie, Je) or i == (Is, Js)):
125                         nxt.append(j)
126                 del g[i]           # Puis, on supprime le sommet i.
127         a+=1
128         if RETOURS:
129             print("étape "+str(a)+" complétée en "+str(floor(time.time()-t0))+" seconde")
130         if SAVTEMP and b<MAXSAVEDINTERETAP and a%INRVALCAP==0:
131             b+=1
132             sav(g,str(a),tab)
133 ##            thread=threading.Thread(target=sav, args=(deepcopy(g),str(a)))
134 ##            thread.start()
135     print("completion du labyrinthe en "+str(floor(time.time()-t0))+" secondes")
136 
137 def sav(g,nom,tab):
138     ta=deepcopy(tab)
139     for c in g:
140         i,j=c
141         ta[i][j]=[0,0,255]
142     ta[Ie][Je]=[255,0,0]
143     ta[Is][Js]=[0,255,0]
144     imageio.imwrite(nom+".png", ta)
145     print("fichier suvegardé après " +str(floor(time.time()-t0))+" secondes")
146 
147 g=converter(tab)
148 solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab)
149 sav(g,NOMFINAL,tab)
```

6 - Solveur v2 - Multi-Threads

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
import threading
print("WARNING /!\ ce script détecte également les cycles, il n'est utilisable que sur des labyrinthes du
type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce programme donne toute les solutions possibles")


t0=time.time()


def ckoa(pix):     #Reconnaissance de mur/chemin/fin/début.
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "start"
    if pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "end"
    if 100>pix[0]:
        return "mur"
    return "chemin"

def converter(tab):
    g={}
    temp=[]
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            temp=[]
            if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-1][j])=="start"):
                temp.append((i-1,j))
            if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or ckoa(tab[i+1][j])=="start"):
                temp.append((i+1,j))
            if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or ckoa(tab[i][j-1])=="start"):
                temp.append((i,j-1))
            if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or ckoa(tab[i][j+1])=="start"):
                temp.append((i,j+1))
            if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or ckoa(tab[i][j])=="start"):
                g[(i,j)]=temp
                temp=[]
    print("completion de la convertion en " +str(floor(time.time()-t0))+" secondes")
    return g

```

```
51  def solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab):
52      a=0
53      b=0
54      cur,nxt=[],[]
55      for i in g:
56          if len(g[i])<2 and not (i == (Ie, Je) or i == (Is, Js)):
57              nxt.append(i)
58      while not nxt==[]:
59          cur,nxt=nxt,[]
60          for i in cur:
61              if i in g:
62                  t=g[i]
63                  for j in t:
64                      l=g[j]
65                      l.remove(i)
66                      g[j]=l
67                      if len(l)<2 and not (i == (Ie, Je) or i == (Is, Js)):
68                          nxt.append(j)
69                  del g[i]
70          a+=1
71          if RETOURS:
72              print("étape "+str(a)+" complétée en "+str(floor(time.time()-t0))+" seconde")
73          if SAVTEMP and b<MAXSAVEDINTERETAP and a%INRVALCAP==0:
74              b+=1
75              sav(g,str(a),tab)
76  ##            thread=threading.Thread(target=sav, args=(deepcopy(g),str(a)))
77  ##            thread.start()
78      print("completion du labyrinth en "+str(floor(time.time()-t0))+" secondes")
79
80  def sav(g,nom,tab):
81      ta=deepcopy(tab)
82      for c in g:
83          i,j=c
84          ta[i][j]=[0,0,255]
85      ta[Ie][Je]=[255,0,0]
86      ta[Is][Js]=[0,255,0]
87      imageio.imwrite(nom+".png", ta)
88      print("fichier suvegardé apres " +str(floor(time.time()-t0))+" secondes")
89
90
91  Ie,Je=0,0
92  Is,Js=99,99
93  def solvm(i):
94      im=Image.open(str(i)+'.png')
95      tab=np.array(im)
96      g=converter(tab)
97      solv(g,False,False,1,1,tab)
98      sav(g,str(i)+' Soluce',tab)
99
100 for i in range(1000):
101     thread = threading.Thread(target=solvm, args=(i,))
102     thread.start()
```

7 - Solveur v2.5 - "Low Memory"

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio

def ckoa2(pix):     #Reconnaissance de mur/chemin.
    if pix[0]==0 and pix[1]==0 and pix[2]==0:
        return "mur"
    return "chemin"

def count(i,j):
    temp=[]
    if (ckoa2(tab[i][j])=="chemin"):
        c=0
        if i>0 and (ckoa2(tab[i-1][j])=="chemin"):
            c+=1
        if i<len(tab)-1 and (ckoa2(tab[i+1][j])=="chemin"):
            c+=1
        if j>0 and (ckoa2(tab[i][j-1])=="chemin"):
            c+=1
        if j<len(tab[0])-1 and (ckoa2(tab[i][j+1])=="chemin"):
            c+=1
        return c
    return 0

def voisins(i,j):
    temp=[]
    if (ckoa2(tab[i][j])=="chemin"):
        c=[]
        if i>0 and (ckoa2(tab[i-1][j])=="chemin"):
            c.append((i-1,j))
        if i<len(tab)-1 and (ckoa2(tab[i+1][j])=="chemin"):
            c.append((i+1,j))
        if j>0 and (ckoa2(tab[i][j-1])=="chemin"):
            c.append((i,j-1))
        if j<len(tab[0])-1 and (ckoa2(tab[i][j+1])=="chemin"):
            c.append((i,j+1))
        return c
    return []

def solv(tab):
    cur,nxt=[],[]
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            if count(i,j)<2 and not ((i,j) == (Ie, Je) or (i,j) == (Is, Js)):
                nxt.append((i,j))
    print("nxt initial créé")
    while not nxt==[]:
```

```
        cur,nxt=nxt,[]
        for cs in cur:
            i,j=cs
            temp=voisins(i,j)
            tab[i][j][0]=0
            tab[i][j][1]=0
            tab[i][j][2]=0
            for ic,jc in temp:
                if count(ic,jc)<2 and not ((ic,jc) == (Ie, Je) or (ic,jc) == (Is, Js)):
                    nxt.append((ic,jc))
    print("completion du labyrinth apres "+str(floor(time.time()-t0))+" secondes")
import cv2



NAME="3.png"
NAMEFIN="3 - Soluce.png"
t0=time.time()

im = cv2.imread(NAME)
tab=np.array(im)
print("fichier chargé après " +str(floor(time.time()-t0))+" secondes")


Ie,Je=0,0
Is,Js=len(tab)-1,len(tab[0])-1

tab[Is][Js]=[255,0,0]

solv(tab)
print("labyrinthe résolu après " +str(floor(time.time()-t0))+" secondes")

imageio.imwrite("3 - Solution seule.png", tab)

im = cv2.imread(NAME)
ta=np.array(im)

for i in range(len(tab)):
    for j in range(len(tab[0])):
        if not tab[i][j][0]==0:
            ta[i][j][0]=0
            ta[i][j][1]=0
            ta[i][j][2]=255
ta[Ie][Je]=[255,0,0]
ta[Is][Js]=[0,255,0]
print("fichier prêt a la sauvegarde après " +str(floor(time.time()-t0))+" secondes")

imageio.imwrite(NAMEFIN, ta)
print("fichier sauvegardé après " +str(floor(time.time()-t0))+" secondes")
```

8 - Générateur v1

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *

print("WARNING /!\ ce script prend en entree une image toute noire, ou un labyrinthe
incomplet, il doit y avoir au moins une case blanche ou grise au départ et si on veut
restreindre le programme a un template, il faut mettre des limites en bleu")
#Ce programme est capable de reprendre pour compléter une génération incomplète sans poser de
questions.
#La sortie doit être placée manuellement.

SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)    "))
RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)    "))
SAVTEMP=SAVTEMPin=="True"
RETOURS=RETOURSin=="True"
MAXSAVEDINTERETAP=int(input("Combien au maximum d'étapes intermédiaires souhaitez-vous
sauvegarder en images ?    "))
NOMFINAL=str(input("Quel nom souhaitez-vous donner au fichier de sauvegarde du labyrinthe
généré ?    "))
NAME=str(input("Nom du fichier du template à ouvrir (image de pixels noirs ([0,0,0]) avec des
délimitations si voulues en bleu ([0,0,255]) et une entrée en gris ([100,100,100])) avec
l'extension jpg ou png (pas testé sur autre chose) :"))
PROB=int(input("Probabilité qu'un mur soit converti en chemin : un chiffre plus grand implique
un
labyrinthe plus aléatoire et un temps de génération plus long (par exemple, un mur a une
chance de se convertir de environ 1/nombre entré).    "))

t0=time.time()
im=Image.open(NAME)
tab=np.array(im)
locked={}
chemins={}
todel=[]
FOUND=False
#Recherche de l'entrée.
azert=0
for i in range(len(tab)):
    for j in range(len(tab[0])):
        if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
            azert=1
            Ie,Je=i,j
            FOUND=True
            break
    if azert==1:
        break

```

```
51  if FOUND:
52      tab[Ie][Je]=[255,0,0]     #Marquage de l'entrée.
53
54  def ckoa(pix):     #Reconnaissance de mur/chemin/hypermur (ces derniers sont les limites du
55  labyrinthe).
56      if pix[0]==255 and pix[1]==0 and pix[2]==0:
57          return "start"
58      if pix[0]==0 and pix[1]==0 and pix[2]==255:
59          return "hypermur"
60      if 100>pix[0]:
61          return "mur"
62      return "chemin"
63
64  n=len(tab)
65  m=len(tab[0])
66
67
68  for i in range(len(tab)):
69      for j in range(len(tab[0])):
70          if ckoa(tab[i][j])=="chemin":
71              chemins[(i,j)]=1
72
73  def converter(tab): #Création d'un graphe g sous forme de dictionnaire, où les clés
74  représentent les coordonnées des points et les valeurs représentent les listes des chemins
75  adjacents.
76      g={}
77      temp=[]
78      for i in range(len(tab)):
79          for j in range(len(tab[0])):
80              temp=[]
81              if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="start"):
82                  temp.append([i-1,j])
83              if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="start"):
84                  temp.append([i+1,j])
85              if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="start"):
86                  temp.append([i,j-1])
87              if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or
88  ckoa(tab[i][j+1])=="start"):
89                  temp.append([i,j+1])
90              if not (ckoa(tab[i][j])=="hypermur"):
91                  g[(i,j)]=temp
92                  temp=[]
93      print("completion de la convertion en " +str(floor(time.time()-t0))+" secondes")
94      return g
95
96  def cheminadkey():
97      for c in g:
98          if ckoa(tab[c])=="chemin":
99              chemins[c]=1
100
101 def gen(g):
102     a=0
103     at=time.time()
104     z = True
105     while z:                    #Tant qu'il peut se passer des choses.
106         z = False
107         for c in g:             #Les murs avec un chemin adjacent peuvent devenir des chemins
```

```
avec une probabilité de 1/PROB.
            if len(g[c]) > 1:
                locked[c]=1
                todel.append(c)
            if not c in locked and len(g[c]) ==1:
                if not c in chemins:
                    z = True
                    if randint(0,PROB)==1:
                        i, j = c
                        x = [i, j]
                        chemins[c]=1
                        if i>0 and not (ckoa(tab[i-1][j])=="hypermur") and not ((i-1,j) in
locked) and not (i-1,j) in chemins:
                            g[i-1,j].append(x)
                        if i<len(tab)-1 and  not (ckoa(tab[i+1][j])=="hypermur") and not
((i+1,j) in locked) and not (i+1,j) in chemins:
                            g[i+1,j].append(x)
                        if j>0 and  not (ckoa(tab[i][j-1])=="hypermur") and not ((i,j-1) in
locked) and not (i,j-1) in chemins:
                            g[i,j-1].append(x)
                        if j<len(tab[0])-1 and not (ckoa(tab[i][j+1])=="hypermur") and not
((i,j+1) in locked) and not (i,j+1) in chemins:
                            g[i,j+1].append(x)
                        g[c].append([0,0])
                        todel.append(c)
        for c in todel:
            if c in g:
                del g[c]
        a+=1
        if RETOURS:
            print("completion de l'étape " +str(a)+" en " +str(floor(time.time()-t0))+"
secondes")
            at=time.time()
        if SAVTEMP:
            at=time.time()
            savetap(g,a)
    print("genere en " +str(floor(time.time()-t0))+" secondes")
    return g

def savetap(g,a):                       #Cette version du programme a été abandonnée avant la
réalisation des tests de cette fonction.
    ta=deepcopy(tab)
    for c in g:
        if len(g[c])==0:
            i,j=c
            ta[i][j]=[0,0,0]
    for c in chemins:
            i,j=c
            ta[i][j]=[255,255,255]
    if FOUND:
        tab[Ie][Je]=[255,0,0]
    if a<MAXSAVEDINTERETAP:
        imageio.imwrite("etape_"+str(a)+".png", ta)
    else:
        imageio.imwrite("etape_"+str(MAXSAVEDINTERETAP)+".png", ta)
    print("étape "+str(a)+" rendue en " +str(floor(time.time()-t0))+" secondes")

```

```
def end(g):       #Génère le tableau de l'image du labyrinthe stocké dans le graphe et le dictionnaire

des chemins.
    ta=deepcopy(tab)
    for c in g:
        if len(g[c])==0:
            i,j=c
            ta[i][j]=[0,0,0]
    for c in chemins:
            i,j=c
            ta[i][j]=[255,255,255]
    for c in locked:
            i,j=c
            ta[i][j]=[0,0,0]
    if FOUND:
        tab[Ie][Je]=[255,0,0]
    print("labyrinthe finale rendue en " +str(floor(time.time()-t0))+" secondes")
    return ta



def finitions():
    for i in range(len(ta)):
        for j in range(len(ta[0])):
            if ckoa(ta[i,j])=="hypermur":
                ta[i,j]=[0,0,0]
    if FOUND:
        ta[Ie][Je]=[100,100,100]
    imageio.imwrite(NOMFINAL+".png", ta)         #Artefact du moment où je n'étais pas au courant que PIL peut enregistrer en PNG : sauvegarde et affichage de l'image.
    (Image.fromarray(ta)).show()



g=converter(tab)  #Lancement de la génération.
gen(g)
cheminadkey()
ta=end(g)
finitions()

print("C'est moche, mais c'est un labyrinthe parfait (reconnu par l'autre script), mais il n'y a pas encore de sortie à placer manuellement. La sortie est un pixel [100,100,100].")
```

9 - Générateur v2 - Sans Issue

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *

print("WARNING /!\ Ce script génère un labyrinthe parfait où un pixel blanc est un chemin et un pixel noir est un mur.")

SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)    "))
RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)    "))
SAVTEMP=SAVTEMPin=="True"
RETOURS=RETOURSin=="True"
MAXSAVEDINTERETAP=int(input("Combien au maximum d'étapes intermédiaires souhaitez-vous sauvegarder en images ?   "))
NOMFINAL=str(input("Quel nom souhaitez-vous donner au fichier de sauvegarde du labyrinthe généré ?   "))
SIZEP=int(input("Taille (abscisse) en pixels du labyrinthe généré :   "))
SIZEQ=int(input("Taille (ordonnée) en pixels du labyrinthe généré :   "))
PROB=int(input("Probabilité qu'un mur soit converti en chemin : un chiffre plus grand implique un
labyrinthe plus aléatoire et un temps de génération plus long (par exemple, un mur a une chance de se convertir de environ 1/nombre entré).   "))

g={}


def fus(ta,p,q):
    for i in range(ta):
        for j in range(ta[0]):
            tab[i+p][j+q]=ta[i][j]

def ckoa(pix):     #Reconnaissance de mur/chemin.
    if pix[0]==0 and pix[1]==0 and pix[2]==255:
        return "hypermur"
    if 100>pix[0]:
        return "mur"
    return "chemin"

tab=[[(0,0,0)*SIZEP] for _ in range(SIZEQ)]

def adj(i,j):
    a=0
    if (i-1,j) in g:
        a+=1
    if (i+1,j) in g:
        a+=1
    if (i,j-1) in g:
```

```
51          a+=1
52      if (i,j+1) in g:
53          a+=1
54      return a
55
56  def possible(i,j,p,q):
57      return i>0 and j>0 and i<p and j<q and not ((i,j) in g) and adj(i,j)==1
58
59  sta=0,0
60  def gen(p,q,Ie,Je):     #Propage l'arbre du labyrinthe à partir d'un seul point.
61      cu,nxt=[],[]
62      g={}
63      if possible(Ie,Je+1,p,q):
64          nxt.append(Ie,Je+1)
65      if possible(Ie,Je-1,p,q):
66          nxt.append(Ie,Je-1)
67      if possible(Ie+1,Je,p,q):
68          nxt.append(Ie+1,Je)
69      if possible(Ie-1,Je,p,q):
70          nxt.append(Ie-1,Je)
71      g[Ie,Je]=1
72      while not nxt==[]:
73          cu,nxt=nxt,[]
74          for k in cu:
75              i,j=k
76              if possible(i,j,p,q) and randint(0,PROB)==1:
77                  g[k]=1
78                  if possible(i,j+1,p,q):
79                      nxt.append(i,j+1)
80                  if possible(i,j-1,p,q):
81                     nxt.append(i,j-1)
82                  if possible(i+1,j,p,q):
83                      nxt.append(i+1,j)
84                  if possible(i-1,j,p,q):
85                      nxt.append(i-1,j)
86      return g
```

10 - Générateur v2

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *
import os
import sys
print("WARNING /!\ Ce script génère un labyrinthe parfait où un pixel blanc est un chemin et un pixel noir est un mur.")

PROB=1

t0=time.time()


def fus(tab,ta,p,q):
    for i in range(ta):
        for j in range(ta[0]):
            tab[i+p][j+q]=ta[i][j]

def ckoa(pix):     #reconnaissance de mur/chemin/sortie/entrée
    if pix[0]==0 and pix[1]==0 and pix[2]==255:
        return "hypermur"
    if 100>pix[0]:
        return "mur"
    return "chemin"

def merge(g1,g2):
    for c in g2:
        g1[c]=g2[c]
    return g1

def adj(g,i,j):
    a=0
    if (i-1,j) in g:
        a+=1
    if (i+1,j) in g:
        a+=1
    if (i,j-1) in g:
        a+=1
    if (i,j+1) in g:
        a+=1
    return a

def possible(g,i,j,p,q):
    return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1

def gen(p,q,Ie,Je,Is,Js):     #Génère à partir d'une entrée et d'une sortie.
    g={}
    cu1,nxt1=[],[]
```

```
    cu2,nxt2=[],[]      #"cu" et "nxt" contiennent les chemins possibles respectivement à
l'instant courant et à l'instant suivant.
    g1={}
    g1[Ie,Je]=1
    g2={}
    g2[Is,Js]=1
    g[Ie,Je]=1
    g[Is,Js]=1
    if possible(g,Ie,Je+1,p,q):#On indexe les murs adjacents à l'entrée et à la sortie qui sont
susceptibles de se transformer en chemin.
        nxt1.append((Ie,Je+1))
    if possible(g,Ie,Je-1,p,q):
        nxt1.append((Ie,Je-1))
    if possible(g,Ie+1,Je,p,q):
        nxt1.append((Ie+1,Je))
    if possible(g,Ie-1,Je,p,q):
       nxt1.append(Ie-1,Je)
    if possible(g,Is,Js+1,p,q):
        nxt2.append((Is,Js+1))
    if possible(g,Is,Js-1,p,q):
        nxt2.append((Is,Js-1))
    if possible(g,Is+1,Js,p,q):
        nxt2.append((Is+1,Js))
    if possible(g,Is-1,Js,p,q):
        nxt2.append((Is-1,Js))
    while not (nxt1==[] and nxt2==[]):
        cu1,nxt1=nxt1,[]
        cu2,nxt2=nxt2,[]
        for k in cu1:                #On fait "pousser" "l'arbre" de l'entrée.
            i,j=k
            if possible(g,i,j,p,q) and randint(0,PROB)==1:
                g[k]=1
                g1[k]=1
                if possible(g,i,j+1,p,q):
                    nxt1.append((i,j+1))
                if possible(g,i,j-1,p,q):
                   nxt1.append((i,j-1))
                if possible(g,i+1,j,p,q):
                    nxt1.append((i+1,j))
                if possible(g,i-1,j,p,q):
                    nxt1.append((i-1,j))
            else:
                if possible(g,i,j,p,q):
                    nxt1.append((i,j))
        for k in cu2:                    #On fait "pousser" "l'arbre" de la sortie.
            i,j=k
            if possible(g,i,j,p,q) and randint(0,PROB)==1:
                g[k]=1
                g2[k]=1
                if possible(g,i,j+1,p,q):
                    nxt2.append((i,j+1))
                if possible(g,i,j-1,p,q):
                   nxt2.append((i,j-1))
                if possible(g,i+1,j,p,q):
                    nxt2.append((i+1,j))
                if possible(g,i-1,j,p,q):
                    nxt2.append((i-1,j))
```

```
            else:
                if possible(g,i,j,p,q):
                    nxt2.append((i,j))
    l=[]
    print("Les deux 'arbres' ont fini de 'pousser'.")
    for i in range(p):                    #On fusionne les deux "arbres" sur un point unique.
        for j in range(q):
            nb=0
            b1=False
            b2=False
            if (i-1,j) in g:
                nb+=1
            if (i-1,j) in g1:
                b1=True
            if (i-1,j) in g2:
                b2=True
            if (i+1,j) in g:
                nb+=1
            if (i+1,j) in g1:
                b1=True
            if (i+1,j) in g2:
                b2=True
            if (i,j+1) in g:
                nb+=1
            if (i,j+1) in g1:
                b1=True
            if (i,j+1) in g2:
                b2=True
            if (i,j-1) in g:
                nb+=1
            if (i,j-1) in g1:
                b1=True
            if (i,j-1) in g2:
                b2=True
            if b1 and b2 and nb==2:
                l.append((i,j))
    if len(l)>0:
        n=randint(0,len(l)-1)
        g[l[n]]=1
    else:
        gen(p,q,Ie,Je,Is,Js)
    return g

def genere(nom,p,q,Ie,Je,Is,Js):
    ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
    g=gen(p,q,Ie,Je,Is,Js)
    for c in g:
        i,j=c
        ta[i][j]=[255,255,255]
    ta[Ie][Je]=[255,0,0]
    ta[Is][Js]=[0,255,0]
    imageio.imwrite(nom+".png", ta)


genere("1000 par 1000",100,100,0,0,99,99)
print(time.time()-t0)
```

11 - Générateur v2 - Multi-Thread

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *
import threading
import os
import sys
print("WARNING /!\ Ce script génère automatiquement un grand nombre de labyrinthes, chacun sur un thread différent.")

PROB=2


t0=time.time()

def ckoa(pix):     #Reconnaissance de mur/chemin/fin/départ.
    if pix[0]==0 and pix[1]==0 and pix[2]==255:
        return "hypermur"
    if 100>pix[0]:
        return "mur"
    return "chemin"

def merge(g1,g2):
    for c in g2:
        g1[c]=g2[c]
    return g1

def adj(g,i,j):
    a=0
    if (i-1,j) in g:
        a+=1
    if (i+1,j) in g:
        a+=1
    if (i,j-1) in g:
        a+=1
    if (i,j+1) in g:
        a+=1
    return a

def possible(g,i,j,p,q):
    return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1

"""On fait "pousser" deux arbres de chemins différents que l'on rejoint en un unique point de leur frontière."""
def gen(p,q,Ie,Je,Is,Js):     #Génère à partir d'une entrée et d'une sortie.
    g={}
    cu1,nxt1=[],[]
    cu2,nxt2=[],[]      #"cu" et "nxt" contiennent les chemins possibles respectivement à l'instant courant et à l'instant suivant.
```

```
53      g1={}
54      g1[Ie,Je]=1
55      g2={}
56      g2[Is,Js]=1
57      g[Ie,Je]=1
58      g[Is,Js]=1
59      if possible(g,Ie,Je+1,p,q):         #On indexe les murs adjacents à l'entrée et à la sortie
60  qui sont susceptibles de se transformer en chemin.
61          nxt1.append((Ie,Je+1))
62      if possible(g,Ie,Je-1,p,q):
63          nxt1.append((Ie,Je-1))
64      if possible(g,Ie+1,Je,p,q):
65          nxt1.append((Ie+1,Je))
66      if possible(g,Ie-1,Je,p,q):
67         nxt1.append(Ie-1,Je)
68      if possible(g,Is,Js+1,p,q):
69          nxt2.append((Is,Js+1))
70      if possible(g,Is,Js-1,p,q):
71          nxt2.append((Is,Js-1))
72      if possible(g,Is+1,Js,p,q):
73          nxt2.append((Is+1,Js))
74      if possible(g,Is-1,Js,p,q):
75          nxt2.append((Is-1,Js))
76      while not (nxt1==[] and nxt2==[]):
77          cu1,nxt1=nxt1,[]
78          cu2,nxt2=nxt2,[]
79          for k in cu1:                   #On fait "pousser" "l'arbre" de l'entrée.
80              i,j=k
81              if possible(g,i,j,p,q) and randint(0,PROB)==1:
82                  g[k]=1
83                  g1[k]=1
84                  if possible(g,i,j+1,p,q):
85                      nxt1.append((i,j+1))
86                  if possible(g,i,j-1,p,q):
87                     nxt1.append((i,j-1))
88                  if possible(g,i+1,j,p,q):
89                      nxt1.append((i+1,j))
90                  if possible(g,i-1,j,p,q):
91                      nxt1.append((i-1,j))
92              else:
93                  if possible(g,i,j,p,q):
94                      nxt1.append((i,j))
95          for k in cu2:                   #On fait "pousser" "l'arbre" de la sortie.
96              i,j=k
97              if possible(g,i,j,p,q) and randint(0,PROB)==1:
98                  g[k]=1
99                  g2[k]=1
100                 if possible(g,i,j+1,p,q):
101                     nxt2.append((i,j+1))
102                 if possible(g,i,j-1,p,q):
103                    nxt2.append((i,j-1))
104                 if possible(g,i+1,j,p,q):
105                     nxt2.append((i+1,j))
106                 if possible(g,i-1,j,p,q):
107                     nxt2.append((i-1,j))
108             else:
109                 if possible(g,i,j,p,q):
```

```
                    nxt2.append((i,j))
    l=[]
    for i in range(p):                    #On fusionne les deux "arbres" sur un point unique.
        for j in range(q):
            nb=0
            b1=False
            b2=False
            if (i-1,j) in g:
                nb+=1
            if (i-1,j) in g1:
                b1=True
            if (i-1,j) in g2:
                b2=True
            if (i+1,j) in g:
                nb+=1
            if (i+1,j) in g1:
                b1=True
            if (i+1,j) in g2:
                b2=True
            if (i,j+1) in g:
                nb+=1
            if (i,j+1) in g1:
                b1=True
            if (i,j+1) in g2:
                b2=True
            if (i,j-1) in g:
                nb+=1
            if (i,j-1) in g1:
                b1=True
            if (i,j-1) in g2:
                b2=True
            if b1 and b2 and nb==2:
                l.append((i,j))
    if len(l)>0:
        n=randint(0,len(l)-1)
        g[l[n]]=1
    else:
        gen(p,q,Ie,Je,Is,Js)
    return g

def genere(nom,p,q,Ie,Je,Is,Js):
    ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
    g=gen(p,q,Ie,Je,Is,Js)
    for c in g:
        i,j=c
        ta[i][j]=[255,255,255]
    ta[Ie][Je]=[255,0,0]
    ta[Is][Js]=[0,255,0]
    imageio.imwrite(nom+".png", ta)

print(time.time()-t0)

for i in range(10):
    tread=threading.Thread(target=genere, args=((str(i),100,100,0,0,99,99)))
    tread.start()
print("launched")
```

12 - Générateur de Labyrinthe en C# sur Unity

```
1   using System.Collections.Generic;
2   using UnityEngine;
3   using UnityEngine.SceneManagement;
4   using Random = UnityEngine.Random;
5
6   public class labgen : MonoBehaviour
7   {
8       public int i; // Champ pour i
9       public int j; // Champ pour j
10      public int SIZEP;
11      public int SIZEQ;
12      public int PROB;
13      public float hauteur;
14      public float hauteurbas;
15      public labgen generat;
16
17      private Dictionary<Vector2, bool> g;
18
19      public void Update()
20      {
21          Vector2 sta = new Vector2(i, j);
22          Generate(SIZEP, SIZEQ, i, j);
23
24          generat.enabled = false;
25
26      }
27
28      private int Adj(int i, int j, Dictionary<Vector2, bool> g)
29      {
30          int a = 0;
31          if (g.ContainsKey(new Vector2(i - 1, j))) a++;
32          if (g.ContainsKey(new Vector2(i + 1, j))) a++;
33          if (g.ContainsKey(new Vector2(i, j - 1))) a++;
34          if (g.ContainsKey(new Vector2(i, j + 1))) a++;
35          return a;
36      }
37
38      private bool Possible(int i, int j, int p, int q, Dictionary<Vector2, bool> g)
39      {
40          return i >= 0 && j >= 0 && i <= p && j <= q && !g.ContainsKey(new Vector2(i, j)) &&
41  Adj(i, j, g) == 1;
42      }
43
44      private void Generate(int p, int q, int Ie, int Je)
45      {
46          string name="";
47          GameObject foundObject =null;
48
49
50
```

```
51          for (int i = 0; i < p; i++) //reseter la grille
52          {
53              for (int j = 0; j < q; j++)
54              {
55                  if (!(i==Ie && j==Je))
56                  {
57                      name = i.ToString() + "," + j.ToString();
58                      // Trouver l'objet par son nom
59
60                      foundObject = GameObject.Find(name);
61
62
63
64                      // Vérifier si l'objet a été trouvé
65                      if (foundObject != null)
66                      {
67                          // Récupérer le composant Transform de l'objet
68                          Transform objectTransform = foundObject.transform;
69                          // Appliquez la nouvelle position à l'objet
70                          objectTransform.position =
71                               new Vector3(objectTransform.position.x, hauteur,
72  objectTransform.position.z);
73
74                          GameObject colone = GameObject.Find(name+"S");
75                          GOCIBL blocscr = colone.GetComponent<GOCIBL>();
76                          blocscr.go=true;
77                      }
78
79
80                  }
81          }
82          }
83
84
85
86
87
88          List<Vector2> cu = new List<Vector2>();
89          List<Vector2> nxt = new List<Vector2>();
90          Dictionary<Vector2, bool> g1 = new Dictionary<Vector2, bool>();
91
92          g1[new Vector2(Ie, Je)] = true;
93
94
95          if (Possible(Ie, Je + 1, p, q, g1))
96          {
97              nxt.Add(new Vector2(Ie, Je + 1));
98          }
99
100         if (Possible(Ie, Je - 1, p, q, g1))
101         {
102             nxt.Add(new Vector2(Ie, Je - 1));
103         }
104
105         if (Possible(Ie + 1, Je, p, q, g1))
106         {
107             nxt.Add(new Vector2(Ie + 1, Je));
```

```
        }

        if (Possible(Ie - 1, Je, p, q, g1))
        {
            nxt.Add(new Vector2(Ie - 1, Je));
        }


        while (nxt.Count != 0)
        {
            cu = new List<Vector2>(nxt);
            nxt.Clear();



            foreach (var k in cu)
            {
                int i = (int)k.x;
                int j = (int)k.y;




                if (Possible(i, j, p, q, g1) && Random.Range(0, PROB) == 1)
                {
                    g1[new Vector2(i, j)] = true;
                    //propager l'info au bloc i,j

////////////////////////////////////////////////////////////////////////////
                    name = i.ToString() + "," + j.ToString();
                    // Trouver l'objet par son nom

                    foundObject = GameObject.Find(name);



                    // Vérifier si l'objet a été trouvé
                    if (foundObject != null)
                    {
                        // Récupérer le composant Transform de l'objet
                        Transform objectTransform = foundObject.transform;
                        // Appliquez la nouvelle position à l'objet
                        objectTransform.position =
                          new Vector3(objectTransform.position.x, hauteurbas,
objectTransform.position.z);

                        GameObject colone = GameObject.Find(name+"S");
                        GOCIBL blocscr = colone.GetComponent<GOCIBL>();
                        blocscr.go=true;



                    }
////////////////////////////////////////////////////////////////////////////

                    if (Possible(i, j + 1, p, q, g1))
                        nxt.Add(new Vector2(i, j + 1));
```

```
165
166                     if (Possible(i, j - 1, p, q, g1))
167                         nxt.Add(new Vector2(i, j - 1));
168
169                     if (Possible(i + 1, j, p, q, g1))
170                         nxt.Add(new Vector2(i + 1, j));
171
172                     if (Possible(i - 1, j, p, q, g1))
173                         nxt.Add(new Vector2(i - 1, j));
174                 }
175                 else
176                 {
177                     if (Possible(i, j, p, q, g1))
178                         nxt.Add(new Vector2(i, j));
179                 }
180             }
181         }
182     }
183 }
```

13 - Outils Utiles

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *

#La fusion écrase les images l'une sur l'autre en ne collant que les zones non noires.

def fus(Name1,Name2):
    im=Image.open(Name1)
    im2=Image.open(Name2)
    tab=np.array(im)
    tab2=np.array(im2)
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            if not (tab2[i][j][0]==0 and tab2[i][j][1]==0 and tab2[i][j][2]==0):
                tab[i][j]=tab2[i][j]
    print(tab)
    imageio.imwrite("done"+".png", tab)
    (Image.fromarray(tab)).show()

def agrandis(Name,multi,multj):
    im=Image.open(Name)
    tab=np.array(im)
    ta=np.full((len(tab)*multi, len(tab[0])*multj, 3), [0, 0, 0], dtype=np.uint8)
    for i in range(len(ta)):
        for j in range(len(ta[0])):
            ta[i][j]=tab[i//multi][j//multj]
    imageio.imwrite(Name+"grand"+".png", ta)
    (Image.fromarray(ta)).show()

def ckoa(pix):
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "start"
    if pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "end"
    if 100>pix[0]:
        return "mur"
    return "chemin"

def clean(Name):                    #Nettoie les fichiers JPG et les rend pixel perfect et
couleur perfect pour enregistrement PNG : chemin = 255,255,255; mur = 0,0,0; départ = 0,255,0;
arrivée = 255,0,0.
    im=Image.open(Name)
    tab=np.array(im)
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            if ckoa(tab[i][j])=="mur":
```

```
                tab[i][j][0]=0
                tab[i][j][1]=0
                tab[i][j][2]=0
            else:
                tab[i][j][0]=255
                tab[i][j][1]=255
                tab[i][j][2]=255
    tab[0][0][0]=0
    tab[0][0][1]=255
    tab[0][0][2]=0
    tab[len(tab)-2][len(tab[0])-2][0]=255
    tab[len(tab)-2][len(tab[0])-2][1]=0
    tab[len(tab)-2][len(tab[0])-2][2]=0
    imageio.imwrite(Name+"clean"+".png", tab)
    (Image.fromarray(tab)).show()
```

14 - Générateur, Solveur, Agrandisseur de Labyrinthe - "Auto-Former"

```
from PIL import Image
import numpy as np
from copy import deepcopy
import time
from math import *
import imageio
from random import *
import os
import sys
import threading


PROB=2

def adj(g,i,j):
    a=0
    if (i-1,j) in g:
        a+=1
    if (i+1,j) in g:
        a+=1
    if (i,j-1) in g:
        a+=1
    if (i,j+1) in g:
        a+=1
    return a

def possible(g,i,j,p,q):
    return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1

def gen(g,p,q,Ie,Je,Is,Js):
    cu1,nxt1=[],[]
    cu2,nxt2=[],[]
    g1={}
    g1[Ie,Je]=1
    g2={}
    g2[Is,Js]=1
    g[Ie,Je]=1
    g[Is,Js]=1
    if possible(g,Ie,Je+1,p,q):
        nxt1.append((Ie,Je+1))
    if possible(g,Ie,Je-1,p,q):
        nxt1.append((Ie,Je-1))
    if possible(g,Ie+1,Je,p,q):
        nxt1.append((Ie+1,Je))
    if possible(g,Ie-1,Je,p,q):
       nxt1.append(Ie-1,Je)
    if possible(g,Is,Js+1,p,q):
        nxt2.append((Is,Js+1))
    if possible(g,Is,Js-1,p,q):
        nxt2.append((Is,Js-1))
```

```
    if possible(g,Is+1,Js,p,q):
        nxt2.append((Is+1,Js))
    if possible(g,Is-1,Js,p,q):
        nxt2.append((Is-1,Js))
    while not (nxt1==[] and nxt2==[]):
        cu1,nxt1=nxt1,[]
        cu2,nxt2=nxt2,[]
        for k in cu1:
            i,j=k
            if possible(g,i,j,p,q) and randint(0,PROB)==1:
                g[k]=1
                g1[k]=1
                if possible(g,i,j+1,p,q):
                    nxt1.append((i,j+1))
                if possible(g,i,j-1,p,q):
                   nxt1.append((i,j-1))
                if possible(g,i+1,j,p,q):
                    nxt1.append((i+1,j))
                if possible(g,i-1,j,p,q):
                    nxt1.append((i-1,j))
            else:
                if possible(g,i,j,p,q):
                    nxt1.append((i,j))
        for k in cu2:
            i,j=k
            if possible(g,i,j,p,q) and randint(0,PROB)==1:
                g[k]=1
                g2[k]=1
                if possible(g,i,j+1,p,q):
                    nxt2.append((i,j+1))
                if possible(g,i,j-1,p,q):
                   nxt2.append((i,j-1))
                if possible(g,i+1,j,p,q):
                    nxt2.append((i+1,j))
                if possible(g,i-1,j,p,q):
                    nxt2.append((i-1,j))
            else:
                if possible(g,i,j,p,q):
                    nxt2.append((i,j))
    l=[]
    for i in range(p):
        for j in range(q):
            nb=0
            b1=False
            b2=False
            if (i-1,j) in g:
                nb+=1
            if (i-1,j) in g1:
                b1=True
            if (i-1,j) in g2:
                b2=True
            if (i+1,j) in g:
                nb+=1
            if (i+1,j) in g1:
                b1=True
            if (i+1,j) in g2:
                b2=True
```

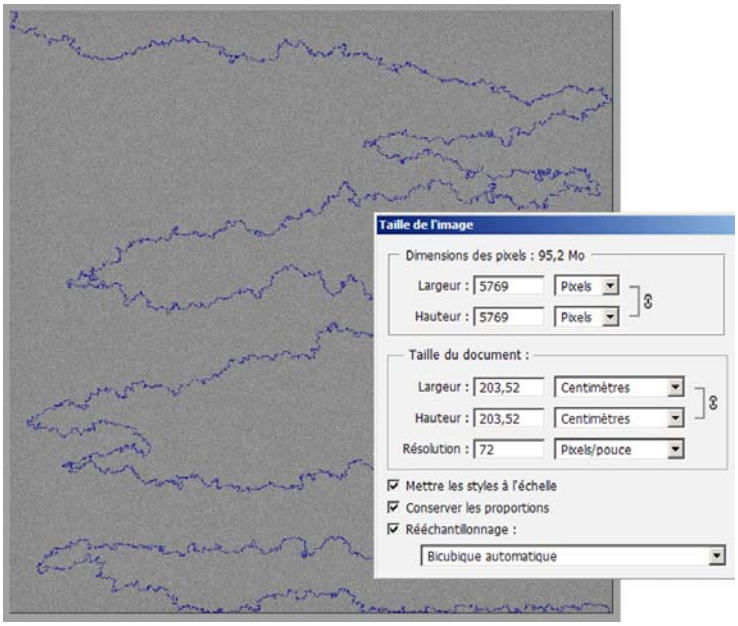
```
            if (i,j+1) in g:
                nb+=1
            if (i,j+1) in g1:
                b1=True
            if (i,j+1) in g2:
                b2=True
            if (i,j-1) in g:
                nb+=1
            if (i,j-1) in g1:
                b1=True
            if (i,j-1) in g2:
                b2=True
            if b1 and b2 and nb==2:
                l.append((i,j))
    if len(l)>0:
        n=randint(0,len(l)-1)
        g[l[n]]=1
    else:
        gen({},p,q,Ie,Je,Is,Js)
    return g

def ckoa(pix):
    if pix[0]==255 and pix[1]==0 and pix[2]==0:
        return "start"
    if pix[0]==0 and pix[1]==255 and pix[2]==0:
        return "end"
    if 100>pix[0]:
        return "mur"
    return "chemin"

def converter(tab):
    g={}
    temp=[]
    for i in range(len(tab)):
        for j in range(len(tab[0])):
            temp=[]
            if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
1][j])=="start"):
                temp.append((i-1,j))
            if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
ckoa(tab[i+1][j])=="start"):
                temp.append((i+1,j))
            if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
ckoa(tab[i][j-1])=="start"):
                temp.append((i,j-1))
            if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
ckoa(tab[i][j+1])=="start"):
                temp.append((i,j+1))
            if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
ckoa(tab[i][j])=="start"):
                g[(i,j)]=temp
                temp=[]
    return g

def solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab,Ie,Je,Is,Js):
    a=0
    b=0
```

```
    cur,nxt=[],[]
    for i in g:
        if len(g[i])<2 and not (i == (Ie, Je) or i == (Is, Js)):
            nxt.append(i)
    while not nxt==[]:
        cur,nxt=nxt,[]
        for i in cur:
            if i in g:
                t=g[i]
                for j in t:
                    l=g[j]
                    l.remove(i)
                    g[j]=l
                    if len(l)<2 and not (i == (Ie, Je) or i == (Is, Js)):
                        nxt.append(j)
                del g[i]
        a+=1

def sav(g,nom,tab,Ie,Je,Is,Js):
    ta=deepcopy(tab)
    for c in g:
        i,j=c
        ta[i][j]=[0,0,255]
    ta[Ie][Je]=[255,0,0]
    ta[Is][Js]=[0,255,0]
    imageio.imwrite(nom+".png", ta)

def genere(nom,p,q,Ie,Je,Is,Js):
    ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
    g=gen({},p,q,Ie,Je,Is,Js)
    for c in g:
        i,j=c
        ta[i][j]=[255,255,255]
    ta[Ie][Je]=[255,0,0]
    ta[Is][Js]=[0,255,0]
    imageio.imwrite(nom+".png", ta)

def AUTOFORM1(nom,p,q,Ie,Je,Is,Js):
    genere(nom,p,q,Ie,Je,Is,Js)
    im=Image.open(nom+".png")
    tab=np.array(im)
    g=converter(tab)
    solv(g,False,False,1,1,tab,Ie,Je,Is,Js)
    sav(g,nom+" solution",tab,Ie,Je,Is,Js)

def agrandis(Name,multi,multj):
    im=Image.open(Name)
    tab=np.array(im)
    ta=np.full((len(tab)*multi, len(tab[0])*multj, 3), [0, 0, 0], dtype=np.uint8)
    for i in range(len(ta)):
        for j in range(len(ta[0])):
            ta[i][j]=tab[i//multi][j//multj]
    imageio.imwrite(Name, ta)

def Auto1(nom,p,q,multi,multj):
    AUTOFORM1(str(nom),p,q,0,0,p-1,q-1)
    agrandis(str(nom)+".png",multi,multj)
```

```
222     agrandis(str(nom)+" solution"+".png",multi,multj)
223 
224 def AUTOFOMULT(NOMLIST,p,q,multi,multj):
225     for nom in NOMLIST:
226         Auto1(nom,p,q,multi,multj)
227         #thread = threading.Thread(target=Auto1, args=(nom,p,q,multi,multj,))
228         #thread.start()
229 
230 
231 AUTOFOMULT(["Labyrinthe "+str(i) for i in range(1,2)],441,784,5,5)
```

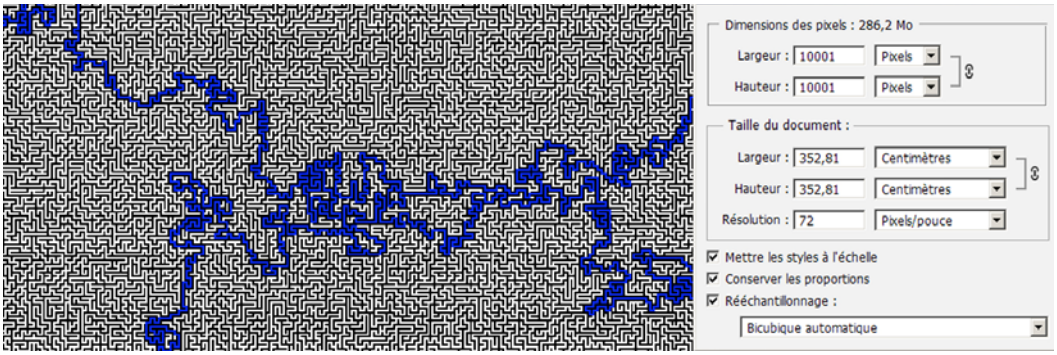
15 - Labyrinthes d'Internet

Parmi les labyrinthes qu'il a trouvé sur Internet, Walter D. Pullen (www.astrolog.org/labyrnth/algrithm.htm) estime que les trois suivants sont les plus gros. Par ordre de difficulté croissante, les voici, ici, résolus.

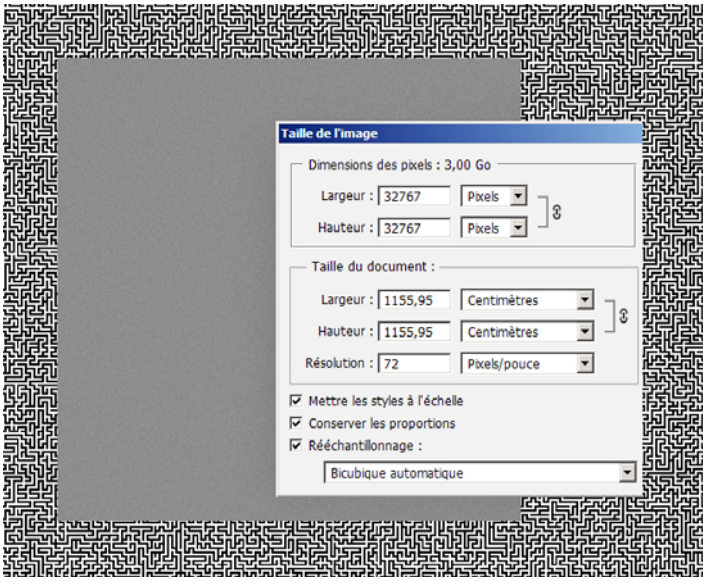


Ce premier labyrinthe
est le plus facile des trois.
Le Fichier image obtenu
est relativement léger.
Ce labyrinthe
de 5769x5769 cases
a une solution.

Le Fichier image obtenu
de ce second labyrinthe
est moyennement lourd.
Ce labyrinthe
de 10001x10001 cases
a une solution du même
type que celle du premier.



Ce dernier labyrinthe
est le plus grand des trois,
le plus grand d'internet
selon Walter D. Pullen.
Le Fichier image obtenu
est extrêmement lourd
pour un fichier image.
Ce labyrinthe de
32767x32767 cases
n'a pas de solution.



On peut observer
au second plan
le labyrinthe
dans son intégralité
et en arrière plan
une toute petite partie
agrandie de ce dernier.