

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.....	2
2 - Convertisseur de Graphe en Labyrinthe.....	4
3 - Convertisseur de Labyrinthe en Graphe - "Reformer"	6
4 - Solveur v1.....	7
5 - Solveur v2.....	10
6 - Solveur v2 - Multi-Threads	13
7 - Solveur v2 - Low Memory	15
8 - Générateur v1	17
9 - Générateur v2 - Sans Issue	21
10 - Générateur v2	23
11 - Générateur v2 - Multi-Threads.....	26
12 - Générateur de Labyrinthe en C# sur Unity.....	29
13 - Outils Utiles.....	33
14 - Générateur, Solveur, Agrandisseur de Labyrinthe - "Auto-Former"	35
15 - Labyrinthes d'Internet.....	40

1 - Convertisseur de Labyrinthe en Graphe

```

1  from PIL import Image
2  import numpy as np
3  import pickle
4
5
6  NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))
7  NAMENR=str(input("Nom du fichier contenant les informations relatives aux portails : "))
8  NameFin=str(input("Nom du fichier du graphe à sauvegarder : "))
9
10 def ckoa2(pix):      #Reconnaissance mur/chemin.
11     if pix[0]==255 and pix[1]==0 and pix[2]==0:
12         return "chemin"
13     elif pix[0]==0 and pix[1]==255 and pix[2]==0:
14         return "chemin"
15     elif 100>pix[0]:
16         return "mur"
17     return "chemin"
18
19 def convertir(g,tab): #Crée un graphe g sous forme de dictionnaire où les clés sont les
20 coordonnées des points "chemin" et les valeurs sont leurs listes de points "chemin" adjacents.
21     temp=[]
22     for i in range(len(tab)):
23         for j in range(len(tab[0])):
24             if ckoa2(tab[i][j])=="chemin":
25                 temp=[]
26                 if i>0 and ckoa2(tab[i-1][j])=="chemin":
27                     temp.append((i-1,j))
28                 if i<len(tab)-1 and ckoa2(tab[i+1][j])=="chemin":
29                     temp.append((i+1,j))
30                 if j>0 and ckoa2(tab[i][j-1])=="chemin":
31                     temp.append((i,j-1))
32                 if j<len(tab[0])-1 and ckoa2(tab[i][j+1])=="chemin":
33                     temp.append((i,j+1))
34                 g[(i,j)]=temp
35                 temp=[]
36     return g
37
38 def enrichiG(g,NAMENR):      #Ajoute les données relatives aux portails dans le graphe.
39     file = open(NAMENR, "rb")
40     enr = pickle.load(file)
41     for c in enr:
42         for e in enr[c]:
43             if not e in g[c]:
44                 g[c].append(e)
45     file.close()
46     return g
47
48 def saveG(g,NameFin):
49     FinalFile = open(NameFin, "wb")
50     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

```

1  import numpy as np
2  from PIL import Image
3  from copy import *
4  from time import *
5  import imageio
6  import pickle
7
8
9  NAMEFILE=input("Nom du fichier pickle contenant le graphe à 'labyrinthiser' (avec l'extension)
10 : ")
11 NAMSAV=input("Nom du fichier où sera sauvegardée l'image du labyrinthe créé : ")
12 NAMSAVTXT=input("Nom du fichier où sera enregistré le dictionnaire des portails : ")
13
14
15 def taille(gres):
16     t=0
17     for c in g:
18         if t<max(len(gres[c][0]),len(gres[c][1])):
19             t=max(len(gres[c][0]),len(gres[c][1]))
20     return t
21
22 def entsortdic(g,INIT): # Le format est {sommets: (liste des sommets desquels on peut
23 arriver, liste des sommets vers lesquels on peut aller)}.
24     gres={}
25     nxt=g[INIT]
26     cur=[]
27     for c in g:
28         gres[c]=([],[])
29     for c in g:
30         tmp=g[c]
31         for i in tmp:
32             gres[i][0].append(c)
33         gres[c]=gres[c][0],deepcopy(tmp)
34         nxt=nxt+tmp
35     return gres
36
37 def motif(entsort):
38     mot=np.full((6, 4, 3), [0, 0, 0], dtype=np.uint8)
39     mot[5][2]=[254,254,254]
40     return mot
41
42 def dicoord(dictidsom, g):
43     dicaret={}
44     for som in g:
45         dicaret[(5,4*dictidsom[som]+2)]=[]
46     for som in g:
47         adj=g[som]
48         for ar in adj:
49             dicaret[(5,4*dictidsom[som]+2)].append((5,4*dictidsom[ar]+2))
50     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"

```

1  import numpy as np
2  from PIL import Image
3  import imageio
4  import pickle
5
6
7  NAMEFILE=input("nom du fichier pickle contenant le graph du jeu a 'labyrinthiser' (avec
8  l'extension)      ")
9  NAMSAV=input("nom du fichier ou sera sauvegardée l'image du labyrinthe créé      ")
10
11
12  #Le "reformeur" prend en entrée un fichier graphe généré par le script ayant l'effet inverse
13  appliqué sur un labyrinthe sans portail.
14
15  def taille(g):
16      i,j=0,0
17      for c in g:
18          if i<c[0]:
19              i=c[0]
20          if j<c[1]:
21              j=c[1]
22      return i,j
23
24  def GEN(NAMEFILE,NAMSAV,Ie=0,Je=0,Is=None,Js=None):
25      file = open(NAMEFILE, "rb")
26      g = pickle.load(file)
27      file.close()
28      Imax,Jmax=taille(g)
29      if Is==None:
30          Is=Imax
31      if Js==None:
32          Js=Jmax
33      ta=np.full((Imax+1, Jmax+1, 3), [0, 0, 0], dtype=np.uint8)
34      for c in g:
35          i,j=c
36          ta[i][j][0]=255
37          ta[i][j][1]=255
38          ta[i][j][2]=255
39      ta[Ie][Je][0]=255
40      ta[Ie][Je][1]=0
41      ta[Ie][Je][2]=0
42      ta[Is][Js][0]=0
43      ta[Is][Js][1]=255
44      ta[Is][Js][2]=0
45      imageio.imwrite(NAMSAV+".png", ta)
46
47  GEN(NAMEFILE,NAMSAV)

```


4 - Solveur v1

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7
8  print("WARNING /\ ce script détecte également les cycles, il n'est utilisable que sur des
9  labyrinthes du type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce
10 programme donne toute les solutions possibles")
11
12
13 SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)"))
14 RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)"))
15 SAVTEMP=SAVTEMPin=="True"
16 RETOURS=RETOURSIn=="True"
17 MAXSAVEDINTERETAP=int(input("Maximum d'étapes intermédiaires sauvegardées en images : "))
18 NOMFINAL=str(input("Nom du fichier de sauvegarde du labyrinthe résolu : "))
19 NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))
20
21
22 t0=time.time()
23 im=Image.open(NAME)
24 tab=np.array(im)
25
26 #Recherche de l'entrée et de la sortie.
27 azert=0
28 for i in range(len(tab)):
29     for j in range(len(tab[0])):
30         if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
31             if azert>0:
32                 azert=2
33                 Is,Js=i,j
34                 break
35             else:
36                 azert=1
37                 Ie,Je=i,j
38         if azert==2:
39             break
40
41
42 tab[Ie][Je]=[255,0,0]      #Marquage de l'entrée et de la sortie.
43 tab[Is][Js]=[0,255,0]
44
45 def ckoa(pix):      #Reconnaissance de mur/chemin/entrée/sortie.
46     if pix[0]==255 and pix[1]==0 and pix[2]==0:
47         return "start"
48     if pix[0]==0 and pix[1]==255 and pix[2]==0:
49         return "end"
50     if 100>pix[0]:
51         return "mur"
52     return "chemin"

```

```

53
54 n=len(tab)
55 m=len(tab[0])
56
57
58
59 def convertir(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 coa(tab[i-1][j])=="end" or coa(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 coa(tab[i+1][j])=="end" or
70 coa(tab[i+1][j])=="start"):
71                 temp.append([i+1,j])
72             if j>0 and (ckoa(tab[i][j-1])=="chemin" or coa(tab[i][j-1])=="end" or
73 coa(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 coa(tab[i][j+1])=="end" or
76 coa(tab[i][j+1])=="start"):
77                 temp.append([i,j+1])
78             if (ckoa(tab[i][j])=="chemin" or coa(tab[i][j])=="end" or
79 coa(tab[i][j])=="start"):
80                 g[(i,j)]=temp
81                 temp=[]
82             print("completion de la conversion 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 enregistrée en " +str(floor(time.time()-t0))+ secondes")
159     (Image.fromarray(ta)).show()
160
161
162 g=convertir(tab)  #Lancement de la résolution.
163 solv(g)
164 ta=end(g)
165 finitions()

```

5 - Solveur v2

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  import threading
8  import pickle
9  print("WARNING /\ ce script détecte également les cycles, il n'est utilisable que sur des
10 labyrinthes du type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce
11 programme donne toute les solutions possibles")
12
13
14 SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False)"))
15 RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False)"))
16 SAVTEMP=SAVTEMPin=="True"
17 RETOURS=RETOURSin=="True"
18 MAXSAVEDINTERETAP=int(input("Maximum d'étapes intermédiaires sauvegardées en images : "))
19 INRVALCAP=int(input("Intervalle d'étape entre chaque image intermédiaire : "))
20 NOMFINAL=str(input("Nom du fichier de sauvegarde du labyrinthe résolu : "))
21 NAME=str(input("Nom du fichier du labyrinthe avec l'extension jpg ou png : "))
22 NAMENR=str(input("Nom du fichier contenant les informations relatives aux portails : "))
23
24
25 t0=time.time()
26 im=Image.open(NAME)
27 tab=np.array(im)
28 print("chargé")
29 #Recherche de l'entrée et de la sortie.
30
31 #On recherche deux façons de couvrir les deux modèles d'enregistrement.
32 azert=0
33 for i in range(len(tab)):
34     for j in range(len(tab[0])):
35         if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
36             if azert>0:
37                 azert=2
38                 Is,Js=i,j
39                 break
40             else:
41                 azert=1
42                 Ie,Je=i,j
43     if azert==2:
44         break
45
46 for i in range(len(tab)):
47     for j in range(len(tab[0])):
48         if tab[i][j][0]==255 and tab[i][j][1]==0 and tab[i][j][2]==0:
49             Ie,Je=i,j
50         if tab[i][j][0]==0 and tab[i][j][1]==255 and tab[i][j][2]==0:

```

```

51         Is,Js=i,j
52
53     tab[Ie][Je]=[255,0,0]        #Marquage de l'entrée et de la sortie.
54     tab[Is][Js]=[0,255,0]
55     print('entree-sortie trouvee')
56     def ckoa(pix):               #Reconnaissance de mur/chemin/entrée/sortie.
57         if pix[0]==255 and pix[1]==0 and pix[2]==0:
58             return "start"
59         if pix[0]==0 and pix[1]==255 and pix[2]==0:
60             return "end"
61         if 100>pix[0]:
62             return "mur"
63         return "chemin"
64
65
66     def convertir(tab):          #Crée un graphe g sous forme de dictionnaire, où les clés sont les
67     coordonnées des points et les valeurs sont les listes des sommets adjacents.
68         g={}
69         temp=[]
70         for i in range(len(tab)):
71             for j in range(len(tab[0])):
72                 temp=[]
73                 if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
74 1][j])=="start"):
75                     temp.append((i-1,j))
76                 if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
77 ckoa(tab[i+1][j])=="start"):
78                     temp.append((i+1,j))
79                 if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
80 ckoa(tab[i][j-1])=="start"):
81                     temp.append((i,j-1))
82                 if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
83 ckoa(tab[i][j+1])=="start"):
84                     temp.append((i,j+1))
85                 if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
86 ckoa(tab[i][j])=="start"):
87                     g[(i,j)]=temp
88                     temp=[]
89                 if not NAMENR=="":
90                     file = open(NAMENR, "rb")
91                     enr = pickle.load(file)
92                     for dep in enr: #enrichissement de g par les portails
93                         for dest in enr[dep]:
94                             g[dep].append(dest)
95                 print("completion de la conversion en " +str(floor(time.time()-t0))+ " secondes")
96                 return g
97
98     #L'algorithme utilisé est le suivant : "Si tu es un cul-de-sac, alors tu es un mur". On
99     applique cette transformation à chaque point du labyrinthe jusqu'à ce que le labyrinthe soit
100     résolu.
101     #Les chemins ne menant nulle part régresseront et disparaîtront jusqu'à ce qu'il ne reste que
102     le bon chemin.
103
104     def solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab):
105         a=0
106         b=0
107         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

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  import threading
8  print("WARNING /\ ce script détecte également les cycles, il n'est utilisable que sur des
9  labyrinthes du
10 type 'lab.jpg', la sortie et l'entrée peuvent cependant être changé de place, ce programme
11 donne toute les solutions possibles")
12
13
14 t0=time.time()
15
16
17 def ckoa(pix):      #Reconnaissance de mur/chemin/fin/début.
18     if pix[0]==255 and pix[1]==0 and pix[2]==0:
19         return "start"
20     if pix[0]==0 and pix[1]==255 and pix[2]==0:
21         return "end"
22     if 100>pix[0]:
23         return "mur"
24     return "chemin"
25
26 def convertir(tab):
27     g={}
28     temp=[]
29     for i in range(len(tab)):
30         for j in range(len(tab[0])):
31             temp=[]
32             if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
33 1][j])=="start"):
34                 temp.append((i-1,j))
35             if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
36 ckoa(tab[i+1][j])=="start"):
37                 temp.append((i+1,j))
38             if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
39 ckoa(tab[i][j-1])=="start"):
40                 temp.append((i,j-1))
41             if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
42 ckoa(tab[i][j+1])=="start"):
43                 temp.append((i,j+1))
44             if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
45 ckoa(tab[i][j])=="start"):
46                 g[(i,j)]=temp
47                 temp=[]
48     print("completion de la conversion en " +str(floor(time.time()-t0))+ " secondes")
49     return g
50

```

```

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"

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7
8  def ckoa2(pix):      #Reconnaissance de mur/chemin.
9      if pix[0]==0 and pix[1]==0 and pix[2]==0:
10         return "mur"
11         return "chemin"
12
13 def count(i,j):
14     temp=[]
15     if (ckoa2(tab[i][j])=="chemin"):
16         c=0
17         if i>0 and (ckoa2(tab[i-1][j])=="chemin"):
18             c+=1
19         if i<len(tab)-1 and (ckoa2(tab[i+1][j])=="chemin"):
20             c+=1
21         if j>0 and (ckoa2(tab[i][j-1])=="chemin"):
22             c+=1
23         if j<len(tab[0])-1 and (ckoa2(tab[i][j+1])=="chemin"):
24             c+=1
25         return c
26     return 0
27
28 def voisins(i,j):
29     temp=[]
30     if (ckoa2(tab[i][j])=="chemin"):
31         c=[]
32         if i>0 and (ckoa2(tab[i-1][j])=="chemin"):
33             c.append((i-1,j))
34         if i<len(tab)-1 and (ckoa2(tab[i+1][j])=="chemin"):
35             c.append((i+1,j))
36         if j>0 and (ckoa2(tab[i][j-1])=="chemin"):
37             c.append((i,j-1))
38         if j<len(tab[0])-1 and (ckoa2(tab[i][j+1])=="chemin"):
39             c.append((i,j+1))
40         return c
41     return []
42
43 def solv(tab):
44     cur,nxt=[],[]
45     for i in range(len(tab)):
46         for j in range(len(tab[0])):
47             if count(i,j)<2 and not ((i,j) == (Ie, Je) or (i,j) == (Is, Js)):
48                 nxt.append((i,j))
49     print("nxt initial créé")
50     while not nxt==[]:

```

```

51         cur,nxt=nxt,[]
52         for cs in cur:
53             i,j=cs
54             temp=voisins(i,j)
55             tab[i][j][0]=0
56             tab[i][j][1]=0
57             tab[i][j][2]=0
58             for ic,jc in temp:
59                 if count(ic,jc)<2 and not ((ic,jc) == (Ie, Je) or (ic,jc) == (Is, Js)):
60                     nxt.append((ic,jc))
61         print("completion du labyrinthe apres " +str(floor(time.time()-t0))+" secondes")
62 import cv2
63
64
65
66 NAME="3.png"
67 NAMEFIN="3 - Soluce.png"
68 t0=time.time()
69
70 im = cv2.imread(NAME)
71 tab=np.array(im)
72 print("fichier chargé après " +str(floor(time.time()-t0))+" secondes")
73
74
75 Ie,Je=0,0
76 Is,Js=len(tab)-1,len(tab[0])-1
77
78 tab[Is][Js]=[255,0,0]
79
80 solv(tab)
81 print("labyrinthe résolu après " +str(floor(time.time()-t0))+" secondes")
82
83 imageio.imwrite("3 - Solution seule.png", tab)
84
85 im = cv2.imread(NAME)
86 ta=np.array(im)
87
88 for i in range(len(tab)):
89     for j in range(len(tab[0])):
90         if not tab[i][j][0]==0:
91             ta[i][j][0]=0
92             ta[i][j][1]=0
93             ta[i][j][2]=255
94 ta[Ie][Je]=[255,0,0]
95 ta[Is][Js]=[0,255,0]
96 print("fichier prêt a la sauvegarde après " +str(floor(time.time()-t0))+" secondes")
97
98 imageio.imwrite(NAMEFIN, ta)
99 print("fichier sauvegardé après " +str(floor(time.time()-t0))+" secondes")

```

8 - Générateur v1

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8
9  print("WARNING /\ ce script prend en entree une image toute noire, ou un labyrinthe
10 incomplet, il doit y avoir au moins une case blanche ou grise au départ et si on veut
11 restreindre le programme a un template, il faut mettre des limites en bleu")
12 #Ce programme est capable de reprendre pour compléter une génération incomplète sans poser de
13 questions.
14 #La sortie doit être placée manuellement.
15
16 SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False) "))
17 RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False) "))
18 SAVTEMP=SAVTEMPin=="True"
19 RETOURS=RETOURSin=="True"
20 MAXSAVEDINTERETAP=int(input("Combien au maximum d'étapes intermédiaires souhaitez-vous
21 sauvegarder en images ? "))
22 NOMFINAL=str(input("Quel nom souhaitez-vous donner au fichier de sauvegarde du labyrinthe
23 généré ? "))
24 NAME=str(input("Nom du fichier du template à ouvrir (image de pixels noirs ([0,0,0]) avec des
25 délimitations si voulues en bleu ([0,0,255]) et une entrée en gris ([100,100,100])) avec
26 l'extension jpg ou png (pas testé sur autre chose) :"))
27 PROB=int(input("Probabilité qu'un mur soit converti en chemin : un chiffre plus grand implique
28 un
29 labyrinthe plus aléatoire et un temps de génération plus long (par exemple, un mur a une
30 chance de se convertir de environ 1/nombre entré). "))
31
32 t0=time.time()
33 im=Image.open(NAME)
34 tab=np.array(im)
35 locked={}
36 chemins={}
37 todel=[]
38 FOUND=False
39 #Recherche de l'entrée.
40 azert=0
41 for i in range(len(tab)):
42     for j in range(len(tab[0])):
43         if 50<tab[i][j][0]<200 and 50<tab[i][j][1]<200 and 50<tab[i][j][2]<200:
44             azert=1
45             Ie,Je=i,j
46             FOUND=True
47             break
48     if azert==1:
49         break
50

```

```

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 convertir(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 conversion 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

```

```

108 avec une probabilité de 1/PROB.
109     if len(g[c]) > 1:
110         locked[c]=1
111         todel.append(c)
112     if not c in locked and len(g[c]) ==1:
113         if not c in chemins:
114             z = True
115             if randint(0,PROB)==1:
116                 i, j = c
117                 x = [i, j]
118                 chemins[c]=1
119                 if i>0 and not (ckoa(tab[i-1][j])=="hypermur") and not ((i-1,j) in
120 locked) and not (i-1,j) in chemins:
121                     g[i-1,j].append(x)
122                 if i<len(tab)-1 and not (ckoa(tab[i+1][j])=="hypermur") and not
123 ((i+1,j) in locked) and not (i+1,j) in chemins:
124                     g[i+1,j].append(x)
125                 if j>0 and not (ckoa(tab[i][j-1])=="hypermur") and not ((i,j-1) in
126 locked) and not (i,j-1) in chemins:
127                     g[i,j-1].append(x)
128                 if j<len(tab[0])-1 and not (ckoa(tab[i][j+1])=="hypermur") and not
129 ((i,j+1) in locked) and not (i,j+1) in chemins:
130                     g[i,j+1].append(x)
131                 g[c].append([0,0])
132                 todel.append(c)
133     for c in todel:
134         if c in g:
135             del g[c]
136     a+=1
137     if RETOURS:
138         print("completion de l'étape " +str(a)+" en " +str(floor(time.time()-t0))+"
139 secondes")
140         at=time.time()
141     if SAVTEMP:
142         at=time.time()
143         savetap(g,a)
144     print("genere en " +str(floor(time.time()-t0))+" secondes")
145     return g
146
147 def savetap(g,a):
148     #Cette version du programme a été abandonnée avant la
149     réalisation de tests de cette fonction.
150     ta=deepcopy(tab)
151     for c in g:
152         if len(g[c])==0:
153             i,j=c
154             ta[i][j]=[0,0,0]
155     for c in chemins:
156         i,j=c
157         ta[i][j]=[255,255,255]
158     if FOUND:
159         tab[Ie][Je]=[255,0,0]
160     if a<MAXSAVEDINTERETAP:
161         imageio.imwrite("etape_"+str(a)+".png", ta)
162     else:
163         imageio.imwrite("etape_"+str(MAXSAVEDINTERETAP)+".png", ta)
164     print("étape "+str(a)+" rendue en " +str(floor(time.time()-t0))+" secondes")

```

```

165 def end(g):          #Génère le tableau de l'image du labyrinthe stocké dans le graphe et le
166 dictionnaire
167
168 des chemins.
169     ta=deepcopy(tab)
170     for c in g:
171         if len(g[c])==0:
172             i,j=c
173             ta[i][j]=[0,0,0]
174     for c in chemins:
175         i,j=c
176         ta[i][j]=[255,255,255]
177     for c in locked:
178         i,j=c
179         ta[i][j]=[0,0,0]
180     if FOUND:
181         tab[Ie][Je]=[255,0,0]
182     print("labyrinthe finale rendue en " +str(floor(time.time()-t0))+ " secondes")
183     return ta
184
185
186
187 def finitions():
188     for i in range(len(ta)):
189         for j in range(len(ta[0])):
190             if ckoa(ta[i,j])=="hypermur":
191                 ta[i,j]=[0,0,0]
192     if FOUND:
193         ta[Ie][Je]=[100,100,100]
194     imageio.imwrite(NOMFINAL+".png", ta)          #Artefact du moment où je n'étais pas au
195 courant que PIL peut enregistrer en PNG : sauvegarde et affichage de l'image.
196     (Image.fromarray(ta)).show()
197
198
199
200
201
202 g=convertir(tab)  #Lancement de la génération.
203 gen(g)
204 cheminadkey()
205 ta=end(g)
206 finitions()
207
208 print("C'est moche, mais c'est un labyrinthe parfait (reconnu par l'autre script), mais il n'y
209 a pas encore de sortie à placer manuellement. La sortie est un pixel [100,100,100].")

```

9 - Générateur v2 - Sans Issue

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8
9  print("WARNING /\ Ce script génère un labyrinthe parfait où un pixel blanc est un chemin et un
10 pixel noir est un mur.")
11
12 SAVTEMPin=(input("Doit-on sauvegarder les étapes intermédiaires ? (True ou False) "))
13 RETOURSin=(input("Doit-on recevoir des retours dans la console ? (True ou False) "))
14 SAVTEMP=SAVTEMPin=="True"
15 RETOURS=RETOURSin=="True"
16 MAXSAVEDINTERETAP=int(input("Combien au maximum d'étapes intermédiaires souhaitez-vous
17 sauvegarder en images ? "))
18 NOMFINAL=str(input("Quel nom souhaitez-vous donner au fichier de sauvegarde du labyrinthe
19 généré ? "))
20 SIZEP=int(input("Taille (abscisse) en pixels du labyrinthe généré : "))
21 SIZEQ=int(input("Taille (ordonnée) en pixels du labyrinthe généré : "))
22 PROB=int(input("Probabilité qu'un mur soit converti en chemin : un chiffre plus grand implique
23 un
24 labyrinthe plus aléatoire et un temps de génération plus long (par exemple, un mur a une chance
25 de se convertir de environ 1/nombre entré). "))
26
27 g={}
28
29
30 def fus(ta,p,q):
31     for i in range(ta):
32         for j in range(ta[0]):
33             tab[i+p][j+q]=ta[i][j]
34
35 def ckoa(pix): #Reconnaissance de mur/chemin.
36     if pix[0]==0 and pix[1]==0 and pix[2]==255:
37         return "hypermur"
38     if 100>pix[0]:
39         return "mur"
40     return "chemin"
41
42 tab=[[ (0,0,0)*SIZEP for _ in range(SIZEQ)]
43
44 def adj(i,j):
45     a=0
46     if (i-1,j) in g:
47         a+=1
48     if (i+1,j) in g:
49         a+=1
50     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

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8  import os
9  import sys
10 print("WARNING /\ Ce script génère un labyrinthe parfait où un pixel blanc est un chemin et un
11 pixel noir est un mur.")
12
13 PROB=1
14
15 t0=time.time()
16
17
18 def fus(tab,ta,p,q):
19     for i in range(ta):
20         for j in range(ta[0]):
21             tab[i+p][j+q]=ta[i][j]
22
23 def ckoa(pix):      #reconnaissance de mur/chemin/sortie/entrée
24     if pix[0]==0 and pix[1]==0 and pix[2]==255:
25         return "hypermur"
26     if 100>pix[0]:
27         return "mur"
28     return "chemin"
29
30 def merge(g1,g2):
31     for c in g2:
32         g1[c]=g2[c]
33     return g1
34
35 def adj(g,i,j):
36     a=0
37     if (i-1,j) in g:
38         a+=1
39     if (i+1,j) in g:
40         a+=1
41     if (i,j-1) in g:
42         a+=1
43     if (i,j+1) in g:
44         a+=1
45     return a
46
47 def possible(g,i,j,p,q):
48     return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1
49
50 def gen(p,q,Ie,Je,Is,Js):      #Génère à partir d'une entrée et d'une sortie.
51     g={}
52     cul,nxtl=[],[]

```

```

53     cu2,nxt2=[],[]          #"cu" et "nxt" contiennent les chemins possibles respectivement à
54 l'instant courant et à l'instant suivant.
55     g1={}
56     g1[Ie,Je]=1
57     g2={}
58     g2[Is,Js]=1
59     g[Ie,Je]=1
60     g[Is,Js]=1
61     if possible(g,Ie,Je+1,p,q):#On indexe les murs adjacents à l'entrée et à la sortie qui sont
62 susceptibles de se transformer en chemin.
63         nxt1.append((Ie,Je+1))
64     if possible(g,Ie,Je-1,p,q):
65         nxt1.append((Ie,Je-1))
66     if possible(g,Ie+1,Je,p,q):
67         nxt1.append((Ie+1,Je))
68     if possible(g,Ie-1,Je,p,q):
69         nxt1.append((Ie-1,Je))
70     if possible(g,Is,Js+1,p,q):
71         nxt2.append((Is,Js+1))
72     if possible(g,Is,Js-1,p,q):
73         nxt2.append((Is,Js-1))
74     if possible(g,Is+1,Js,p,q):
75         nxt2.append((Is+1,Js))
76     if possible(g,Is-1,Js,p,q):
77         nxt2.append((Is-1,Js))
78     while not (nxt1==[] and nxt2==[]):
79         cul1,nxt1=nxt1,[]
80         cu2,nxt2=nxt2,[]
81         for k in cul1:                #On fait "pousser" "l'arbre" de l'entrée.
82             i,j=k
83             if possible(g,i,j,p,q) and randint(0,PROB)==1:
84                 g[k]=1
85                 g1[k]=1
86                 if possible(g,i,j+1,p,q):
87                     nxt1.append((i,j+1))
88                 if possible(g,i,j-1,p,q):
89                     nxt1.append((i,j-1))
90                 if possible(g,i+1,j,p,q):
91                     nxt1.append((i+1,j))
92                 if possible(g,i-1,j,p,q):
93                     nxt1.append((i-1,j))
94             else:
95                 if possible(g,i,j,p,q):
96                     nxt1.append((i,j))
97         for k in cu2:                #On fait "pousser" "l'arbre" de la sortie.
98             i,j=k
99             if possible(g,i,j,p,q) and randint(0,PROB)==1:
100                 g[k]=1
101                 g2[k]=1
102                 if possible(g,i,j+1,p,q):
103                     nxt2.append((i,j+1))
104                 if possible(g,i,j-1,p,q):
105                     nxt2.append((i,j-1))
106                 if possible(g,i+1,j,p,q):
107                     nxt2.append((i+1,j))
108                 if possible(g,i-1,j,p,q):
109                     nxt2.append((i-1,j))

```

```

110         else:
111             if possible(g,i,j,p,q):
112                 nxt2.append((i,j))
113     l=[]
114     print("Les deux 'arbres' ont fini de 'pousser'.")
115     for i in range(p):                                #On fusionne les deux "arbres" sur un point unique.
116         for j in range(q):
117             nb=0
118             b1=False
119             b2=False
120             if (i-1,j) in g:
121                 nb+=1
122             if (i-1,j) in g1:
123                 b1=True
124             if (i-1,j) in g2:
125                 b2=True
126             if (i+1,j) in g:
127                 nb+=1
128             if (i+1,j) in g1:
129                 b1=True
130             if (i+1,j) in g2:
131                 b2=True
132             if (i,j+1) in g:
133                 nb+=1
134             if (i,j+1) in g1:
135                 b1=True
136             if (i,j+1) in g2:
137                 b2=True
138             if (i,j-1) in g:
139                 nb+=1
140             if (i,j-1) in g1:
141                 b1=True
142             if (i,j-1) in g2:
143                 b2=True
144             if b1 and b2 and nb==2:
145                 l.append((i,j))
146     if len(l)>0:
147         n=randint(0,len(l)-1)
148         g[l[n]]=1
149     else:
150         gen(p,q,Ie,Je,Is,Js)
151     return g
152
153 def genere(nom,p,q,Ie,Je,Is,Js):
154     ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
155     g=gen(p,q,Ie,Je,Is,Js)
156     for c in g:
157         i,j=c
158         ta[i][j]=[255,255,255]
159     ta[Ie][Je]=[255,0,0]
160     ta[Is][Js]=[0,255,0]
161     imageio.imwrite(nom+".png", ta)
162
163
164 genere("1000 par 1000",100,100,0,0,99,99)
165 print(time.time()-t0)

```

11 - Générateur v2 - Multi-Thread

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8  import threading
9  import os
10 import sys
11 print("WARNING /\ Ce script génère automatiquement un grand nombre de labyrinthes, chacun sur
12 un thread différent.")
13
14 PROB=2
15
16
17 t0=time.time()
18
19 def ckoa(pix):      #Reconnaissance de mur/chemin/fin/départ.
20     if pix[0]==0 and pix[1]==0 and pix[2]==255:
21         return "hypermur"
22     if 100>pix[0]:
23         return "mur"
24     return "chemin"
25
26 def merge(g1,g2):
27     for c in g2:
28         g1[c]=g2[c]
29     return g1
30
31 def adj(g,i,j):
32     a=0
33     if (i-1,j) in g:
34         a+=1
35     if (i+1,j) in g:
36         a+=1
37     if (i,j-1) in g:
38         a+=1
39     if (i,j+1) in g:
40         a+=1
41     return a
42
43 def possible(g,i,j,p,q):
44     return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1
45
46 """On fait "pousser" deux arbres de chemins différents que l'on rejoint en un unique point de
47 leur frontière."""
48 def gen(p,q,Ie,Je,Is,Js):      #Génère à partir d'une entrée et d'une sortie.
49     g={}
50     cu1,nxt1=[],[]
51     cu2,nxt2=[],[]             #"cu" et "nxt" contiennent les chemins possibles respectivement à
52 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         cul,nxt1=nxt1,[]
78         cu2,nxt2=nxt2,[]
79         for k in cul:                      #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):

```

```

110             nxt2.append((i,j))
111     l=[]
112     for i in range(p):                #On fusionne les deux "arbres" sur un point unique.
113         for j in range(q):
114             nb=0
115             b1=False
116             b2=False
117             if (i-1,j) in g:
118                 nb+=1
119             if (i-1,j) in g1:
120                 b1=True
121             if (i-1,j) in g2:
122                 b2=True
123             if (i+1,j) in g:
124                 nb+=1
125             if (i+1,j) in g1:
126                 b1=True
127             if (i+1,j) in g2:
128                 b2=True
129             if (i,j+1) in g:
130                 nb+=1
131             if (i,j+1) in g1:
132                 b1=True
133             if (i,j+1) in g2:
134                 b2=True
135             if (i,j-1) in g:
136                 nb+=1
137             if (i,j-1) in g1:
138                 b1=True
139             if (i,j-1) in g2:
140                 b2=True
141             if b1 and b2 and nb==2:
142                 l.append((i,j))
143     if len(l)>0:
144         n=randint(0,len(l)-1)
145         g[l[n]]=1
146     else:
147         gen(p,q,Ie,Je,Is,Js)
148     return g
149
150 def genere(nom,p,q,Ie,Je,Is,Js):
151     ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
152     g=gen(p,q,Ie,Je,Is,Js)
153     for c in g:
154         i,j=c
155         ta[i][j]=[255,255,255]
156     ta[Ie][Je]=[255,0,0]
157     ta[Is][Js]=[0,255,0]
158     imageio.imwrite(nom+".png", ta)
159
160 print(time.time()-t0)
161
162 for i in range(10):
163     tread=threading.Thread(target=genere, args=((str(i),100,100,0,0,99,99)))
164     tread.start()
165 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     private int Adj(int i, int j, Dictionary<Vector2, bool> g)
28     {
29         int a = 0;
30         if (g.ContainsKey(new Vector2(i - 1, j))) a++;
31         if (g.ContainsKey(new Vector2(i + 1, j))) a++;
32         if (g.ContainsKey(new Vector2(i, j - 1))) a++;
33         if (g.ContainsKey(new Vector2(i, j + 1))) a++;
34         return a;
35     }
36
37     private bool Possible(int i, int j, int p, int q, Dictionary<Vector2, bool> g)
38     {
39         return i >= 0 && j >= 0 && i <= p && j <= q && !g.ContainsKey(new Vector2(i, j)) &&
40 Adj(i, j, g) == 1;
41     }
42
43     private void Generate(int p, int q, int Ie, int Je)
44     {
45         string name="";
46         GameObject foundObject =null;
47
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));

```



```

108     }
109
110     if (Possible(Ie - 1, Je, p, q, gl))
111     {
112         nxt.Add(new Vector2(Ie - 1, Je));
113     }
114
115
116     while (nxt.Count != 0)
117     {
118         cu = new List<Vector2>(nxt);
119         nxt.Clear();
120
121
122
123         foreach (var k in cu)
124         {
125             int i = (int)k.x;
126             int j = (int)k.y;
127
128
129
130
131             if (Possible(i, j, p, q, gl) && Random.Range(0, PROB) == 1)
132             {
133                 gl[new Vector2(i, j)] = true;
134                 //propager l'info au bloc i,j
135
136                 //////////////////////////////////////
137                 name = i.ToString() + "," + j.ToString();
138                 // Trouver l'objet par son nom
139
140                 foundObject = GameObject.Find(name);
141
142
143
144                 // Vérifier si l'objet a été trouvé
145                 if (foundObject != null)
146                 {
147                     // Récupérer le composant Transform de l'objet
148                     Transform objectTransform = foundObject.transform;
149                     // Appliquez la nouvelle position à l'objet
150                     objectTransform.position =
151                         new Vector3(objectTransform.position.x, hauteurbas,
152 objectTransform.position.z);
153
154                     GameObject colone = GameObject.Find(name+"S");
155                     GOCIBL blocscr = colone.GetComponent<GOCIBL>();
156                     blocscr.go=true;
157
158
159
160                 }
161                 //////////////////////////////////////
162
163                 if (Possible(i, j + 1, p, q, gl))
164                     nxt.Add(new Vector2(i, j + 1));

```

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

13 - Outils Utiles

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8
9  #La fusion écrase les images l'une sur l'autre en ne collant que les zones non noires.
10
11 def fus(Name1,Name2):
12     im=Image.open(Name1)
13     im2=Image.open(Name2)
14     tab=np.array(im)
15     tab2=np.array(im2)
16     for i in range(len(tab)):
17         for j in range(len(tab[0])):
18             if not (tab2[i][j][0]==0 and tab2[i][j][1]==0 and tab2[i][j][2]==0):
19                 tab[i][j]=tab2[i][j]
20     print(tab)
21     imageio.imwrite("done"+" .png", tab)
22     (Image.fromarray(tab)).show()
23
24 def agrandis(Name,multi,multj):
25     im=Image.open(Name)
26     tab=np.array(im)
27     ta=np.full((len(tab)*multi, len(tab[0])*multj, 3), [0, 0, 0], dtype=np.uint8)
28     for i in range(len(ta)):
29         for j in range(len(ta[0])):
30             ta[i][j]=tab[i//multi][j//multj]
31     imageio.imwrite(Name+"grand"+" .png", ta)
32     (Image.fromarray(ta)).show()
33
34 def ckoa(pix):
35     if pix[0]==255 and pix[1]==0 and pix[2]==0:
36         return "start"
37     if pix[0]==0 and pix[1]==255 and pix[2]==0:
38         return "end"
39     if 100>pix[0]:
40         return "mur"
41     return "chemin"
42
43 def clean(Name):
44     #Nettoie les fichiers JPG et les rend pixel perfect et
45     couleur perfect pour enregistrement PNG : chemin = 255,255,255; mur = 0,0,0; départ = 0,255,0;
46     arrivée = 255,0,0.
47     im=Image.open(Name)
48     tab=np.array(im)
49     for i in range(len(tab)):
50         for j in range(len(tab[0])):
51             if ckoa(tab[i][j])=="mur":

```

```
51         tab[i][j][0]=0
52         tab[i][j][1]=0
53         tab[i][j][2]=0
54     else:
55         tab[i][j][0]=255
56         tab[i][j][1]=255
57         tab[i][j][2]=255
58 tab[0][0][0]=0
59 tab[0][0][1]=255
60 tab[0][0][2]=0
61 tab[len(tab)-2][len(tab[0])-2][0]=255
62 tab[len(tab)-2][len(tab[0])-2][1]=0
63 tab[len(tab)-2][len(tab[0])-2][2]=0
64 imageio.imwrite(Name+"clean"+" .png", tab)
65 (Image.fromarray(tab)).show()
```

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

```

1  from PIL import Image
2  import numpy as np
3  from copy import deepcopy
4  import time
5  from math import *
6  import imageio
7  from random import *
8  import os
9  import sys
10 import threading
11
12
13 PROB=2
14
15 def adj(g,i,j):
16     a=0
17     if (i-1,j) in g:
18         a+=1
19     if (i+1,j) in g:
20         a+=1
21     if (i,j-1) in g:
22         a+=1
23     if (i,j+1) in g:
24         a+=1
25     return a
26
27 def possible(g,i,j,p,q):
28     return i>=0 and j>=0 and i<p and j<q and not ((i,j) in g) and adj(g,i,j)==1
29
30 def gen(g,p,q,Ie,Je,Is,Js):
31     cul,nxt1=[],[]
32     cu2,nxt2=[],[]
33     g1={}
34     g1[Ie,Je]=1
35     g2={}
36     g2[Is,Js]=1
37     g[Ie,Je]=1
38     g[Is,Js]=1
39     if possible(g,Ie,Je+1,p,q):
40         nxt1.append((Ie,Je+1))
41     if possible(g,Ie,Je-1,p,q):
42         nxt1.append((Ie,Je-1))
43     if possible(g,Ie+1,Je,p,q):
44         nxt1.append((Ie+1,Je))
45     if possible(g,Ie-1,Je,p,q):
46         nxt1.append((Ie-1,Je))
47     if possible(g,Is,Js+1,p,q):
48         nxt2.append((Is,Js+1))
49     if possible(g,Is,Js-1,p,q):
50         nxt2.append((Is,Js-1))

```

```

51     if possible(g,Is+1,Js,p,q):
52         nxt2.append((Is+1,Js))
53     if possible(g,Is-1,Js,p,q):
54         nxt2.append((Is-1,Js))
55     while not (nxt1==[] and nxt2==[]):
56         cu1,nxt1=nxt1,[]
57         cu2,nxt2=nxt2,[]
58         for k in cu1:
59             i,j=k
60             if possible(g,i,j,p,q) and randint(0,PROB)==1:
61                 g[k]=1
62                 g1[k]=1
63                 if possible(g,i,j+1,p,q):
64                     nxt1.append((i,j+1))
65                 if possible(g,i,j-1,p,q):
66                     nxt1.append((i,j-1))
67                 if possible(g,i+1,j,p,q):
68                     nxt1.append((i+1,j))
69                 if possible(g,i-1,j,p,q):
70                     nxt1.append((i-1,j))
71             else:
72                 if possible(g,i,j,p,q):
73                     nxt1.append((i,j))
74         for k in cu2:
75             i,j=k
76             if possible(g,i,j,p,q) and randint(0,PROB)==1:
77                 g[k]=1
78                 g2[k]=1
79                 if possible(g,i,j+1,p,q):
80                     nxt2.append((i,j+1))
81                 if possible(g,i,j-1,p,q):
82                     nxt2.append((i,j-1))
83                 if possible(g,i+1,j,p,q):
84                     nxt2.append((i+1,j))
85                 if possible(g,i-1,j,p,q):
86                     nxt2.append((i-1,j))
87             else:
88                 if possible(g,i,j,p,q):
89                     nxt2.append((i,j))
90     l=[]
91     for i in range(p):
92         for j in range(q):
93             nb=0
94             b1=False
95             b2=False
96             if (i-1,j) in g:
97                 nb+=1
98             if (i-1,j) in g1:
99                 b1=True
100             if (i-1,j) in g2:
101                 b2=True
102             if (i+1,j) in g:
103                 nb+=1
104             if (i+1,j) in g1:
105                 b1=True
106             if (i+1,j) in g2:
107                 b2=True

```

```

108         if (i,j+1) in g:
109             nb+=1
110         if (i,j+1) in g1:
111             b1=True
112         if (i,j+1) in g2:
113             b2=True
114         if (i,j-1) in g:
115             nb+=1
116         if (i,j-1) in g1:
117             b1=True
118         if (i,j-1) in g2:
119             b2=True
120         if b1 and b2 and nb==2:
121             l.append((i,j))
122     if len(l)>0:
123         n=randint(0,len(l)-1)
124         g[l[n]]=1
125     else:
126         gen({},p,q,Ie,Je,Is,Js)
127     return g
128
129 def ckoa(pix):
130     if pix[0]==255 and pix[1]==0 and pix[2]==0:
131         return "start"
132     if pix[0]==0 and pix[1]==255 and pix[2]==0:
133         return "end"
134     if 100>pix[0]:
135         return "mur"
136     return "chemin"
137
138 def converter(tab):
139     g={}
140     temp=[]
141     for i in range(len(tab)):
142         for j in range(len(tab[0])):
143             temp=[]
144             if i>0 and (ckoa(tab[i-1][j])=="chemin" or ckoa(tab[i-1][j])=="end" or ckoa(tab[i-
145 1][j])=="start"):
146                 temp.append((i-1,j))
147             if i<len(tab)-1 and (ckoa(tab[i+1][j])=="chemin" or ckoa(tab[i+1][j])=="end" or
148 ckoa(tab[i+1][j])=="start"):
149                 temp.append((i+1,j))
150             if j>0 and (ckoa(tab[i][j-1])=="chemin" or ckoa(tab[i][j-1])=="end" or
151 ckoa(tab[i][j-1])=="start"):
152                 temp.append((i,j-1))
153             if j<len(tab[0])-1 and (ckoa(tab[i][j+1])=="chemin" or ckoa(tab[i][j+1])=="end" or
154 ckoa(tab[i][j+1])=="start"):
155                 temp.append((i,j+1))
156             if (ckoa(tab[i][j])=="chemin" or ckoa(tab[i][j])=="end" or
157 ckoa(tab[i][j])=="start"):
158                 g[(i,j)]=temp
159             temp=[]
160     return g
161
162 def solv(g,RETOURS,SAVTEMP,MAXSAVEDINTERETAP,INRVALCAP,tab,Ie,Je,Is,Js):
163     a=0
164     b=0

```

```

165     cur,nxt=[],[]
166     for i in g:
167         if len(g[i])<2 and not (i == (Ie, Je) or i == (Is, Js)):
168             nxt.append(i)
169     while not nxt==[]:
170         cur,nxt=nxt,[]
171         for i in cur:
172             if i in g:
173                 t=g[i]
174                 for j in t:
175                     l=g[j]
176                     l.remove(i)
177                     g[j]=l
178                     if len(l)<2 and not (i == (Ie, Je) or i == (Is, Js)):
179                         nxt.append(j)
180                 del g[i]
181         a+=1
182
183 def sav(g,nom,tab,Ie,Je,Is,Js):
184     ta=deepcopy(tab)
185     for c in g:
186         i,j=c
187         ta[i][j]=[0,0,255]
188     ta[Ie][Je]=[255,0,0]
189     ta[Is][Js]=[0,255,0]
190     imageio.imwrite(nom+".png", ta)
191
192 def genere(nom,p,q,Ie,Je,Is,Js):
193     ta=np.full((p, q, 3), [0, 0, 0], dtype=np.uint8)
194     g=gen({},p,q,Ie,Je,Is,Js)
195     for c in g:
196         i,j=c
197         ta[i][j]=[255,255,255]
198     ta[Ie][Je]=[255,0,0]
199     ta[Is][Js]=[0,255,0]
200     imageio.imwrite(nom+".png", ta)
201
202 def AUTOFORM1(nom,p,q,Ie,Je,Is,Js):
203     genere(nom,p,q,Ie,Je,Is,Js)
204     im=Image.open(nom+".png")
205     tab=np.array(im)
206     g=converter(tab)
207     solv(g,False,False,1,1,tab,Ie,Je,Is,Js)
208     sav(g,nom+" solution",tab,Ie,Je,Is,Js)
209
210 def agrandis(Name,multi,multj):
211     im=Image.open(Name)
212     tab=np.array(im)
213     ta=np.full((len(tab)*multi, len(tab[0])*multj, 3), [0, 0, 0], dtype=np.uint8)
214     for i in range(len(ta)):
215         for j in range(len(ta[0])):
216             ta[i][j]=tab[i//multi][j//multj]
217     imageio.imwrite(Name, ta)
218
219 def Auto1(nom,p,q,multi,multj):
220     AUTOFORM1(str(nom),p,q,0,0,p-1,q-1)
221     agrandis(str(nom)+".png",multi,multj)

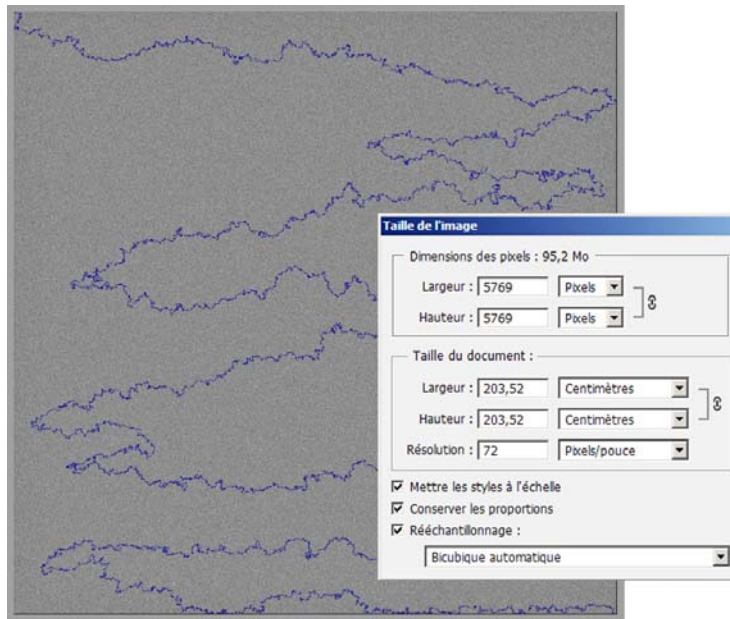
```



```
222     agrandis(str(nom)+" solution"+" .png",multi,multj)
223
224 def AUTOFOMULT(NOMLIST,p,q,multi,multj):
225     for nom in NOMLIST:
226         Autol(nom,p,q,multi,multj)
227         #thread = threading.Thread(target=Autol, 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)
```

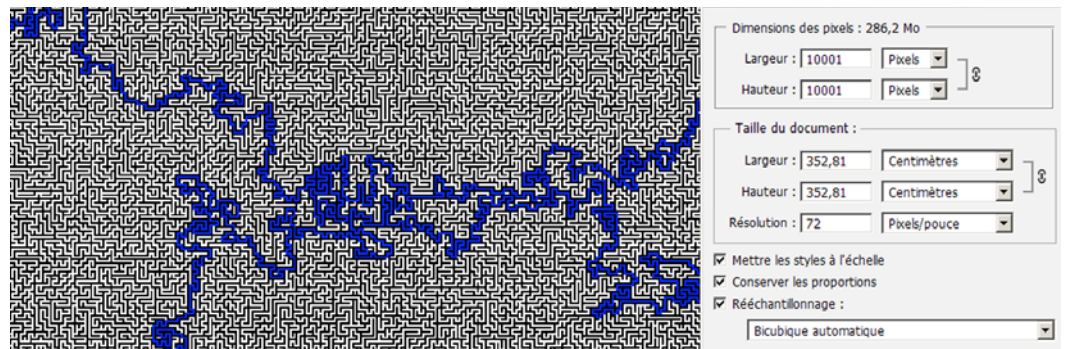
15 - Labyrinthes d'Internet

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.

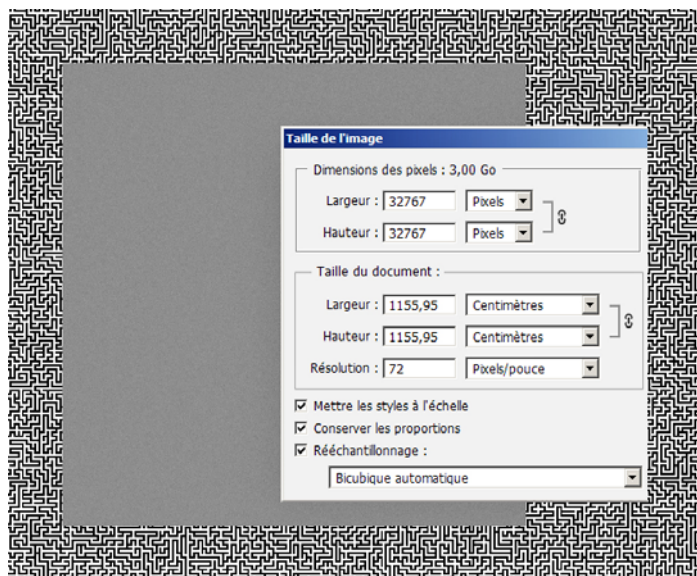


Ce premier labyrinthe est le plus facile des trois. Le Fichier image obtenu est relativement léger. Ce labyrinthe de 5769x5769 cases a une solution.

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.