Correction des exercices sur le labyrinthe

Antonin Dudermel Ambroise Poulet Matthias Goffette

Nous avons ici un algorithme créant un labyrinthe parfait de dimension n. Nous voulons le modifier, d'abord pour enegistrer le labyrinthe en une image, puis pour construire le chemin entre les cases $(0,0)$ *et* $(n-1,n-1)$.

Les fonctions originales sont les suivantes :

```
def visiter(c):
    (x,y) = c
    if x < 0 or x>= n or y<0 or y>=n:
        return
    atteinte[x][y] = True

def est_atteinte(c):
    (x,y) = c
    if x < 0 or x>= n or y<0 or y>=n:
        return True
    return atteinte[x][y]

def choix(c):
    (x,y) = c
    r = []
    def ajouter(p):
        if not est_atteinte(p):
            r.append(p)
    ajouter((x-1,y))
    ajouter((x+1,y))
    ajouter((x,y-1))
    ajouter((x,y+1))
    return r

def tirage(L):
    m = len(L)
    assert m > 0
    return L[random.randint(0, m-1)]

def labyrinthe():
    pile = creer_pile(n*n)
    empiler(pile, (0,0))
    visiter((0,0))
    while not est_vide(pile):
        cellule = depiler(pile)
        print(cellule)
        c = choix(cellule)
        if len(c) > 0:
            suivante = tirage(c)
            # c'est ici qu'on relie les cases cellule et suivante
            visiter(suivante)
            empiler(pile, cellule)
            empiler(pile, suivante)

atteinte = [[False] * n for i in range(n)]
```

Tout d'abord, nous allons réaliser l'enregistrement du labyrinthe au format PGM. Ce format permet d'enegistrer facilement des images en nivau de gris. Pour ce faire, nous avons besoin de

plusieurs modifications :

- En premier lieu, il faut compléter la fonction labyrinthe, en ajoutant un tableau `lislab`, de dimension $(2n+1)\times(2n+1)$ qui contiendra la représentation du labyrinthe. Il est initialisé de manière à ce que chaque case soit isolée des autres.
- Ensuite, la fonction `save_lab` permet de sauvegarder le labyrinthe au format désiré. Le tableau `lislab` est d'abord traité de manière à ce que tous les éléments soient du type `str`. Puis viens l'enregistrement proprement dit. Un premier `f.write` permet d'écrire l'en-tête de l'image, dans laquelle il faut indiquer `P2`, puis les dimensions et enfin la valeur codant le blanc, ici `cb`.

```
cb = 15 #couleur d'une case visitée

def labyrinthe():
        """Créé le tableau du labyrinthe."""
    # Création de lislab, tableau de dimensions (2*n+1) contenant la repré
        sentation du labyrinthe
    lislab = [[0]*(2*n+1) for i in range(2*n + 1)]
    for i in range(n):
        for j in range(n):
            lislab[2*i+1][2*j+1] = 1

    pile = creer_pile(n*n)
    empiler(pile, (0,0))
    visiter((0,0))
    while not est_vide(pile):
        cellule = depiler(pile)
        #print(cellule)
        c = choix(cellule)
        if len(c) > 0:
            suivante = tirage(c)
            lislab[cellule[0] +1+ suivante[0]][cellule[1]+1 +suivante[1]] = cb
                # Lie les deux cases
            visiter(suivante)
            empiler(pile, cellule)
            empiler(pile, suivante)
    return lislab

def save_lab(fichier, lislab):
    """Enregistre le labyrinthe en pgm"""
    #conversion des éléments de lislab en str
    for i in range(2*n+1):
        for j in range(2*n+1):
            lislab[i][j] = str(lislab[i][j])
    with open(fichier + '.pgm', "w") as f:
        f.write("P2\n"+str(2*n+1)+' '+str(2*n+1)+'\n'+str(cb)+'\n')
        for i in lislab:
            f.write(' '.join(i)+'\n')
```

La question suivante consiste à construire le chemin reliant les cases $(0,0)$ et $(n-1,n-1)$. Nous avons besoin d'une nouvelle couleur, `cc` pour coder le chemin. Le point clé pour construire le chemin consiste à remarquer que `labyrinthe` construit naturellement le chemin lorsque la case $(n-1,n-1)$ est visitée. Ainsi, si l'état de la pile est le suivant, alors les cases C_{n-1} et C_n sont adjacentes.

Par récurrence, si la pile est

C_n
C_{n-1}
...
C_1
C_0

alors, le chemin allant de c0 a cn est [c0,c1,...,cn] Or $C_0 = (0,0)$, donc pour $C_n = (n-1,n-1)$ le chemin est la pile.

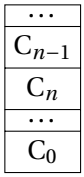

Figure 1: lejeu

- Là aussi, nous avons besoin d'une fonction auxilliaire, `revdup`. Elle prend en argument une pile et retourne une pile contenant les mêmes éléments, mais retournés.
- Nous introduisons une nouvelle variable pour le labyrinthe : `chemin`. Elle est initialisée à `False`, mais contiendra la liste des cases du chemin à la fin. Lors de l'exécution du `while` principal, un premier cas se présente. En effet, si la case $(n-1, n-1)$ est atteinte, c'est alors qu'un chemin a été construit. Nous vérifions alors que `chemin` est encore vide, puis la valeur du chemin est stocké dans `chemin` sous forme de pile.
- En sortie du `while`, le code suivan permet de modifier `lislab` pour changer la couleur du chemin.

```
        xp,yp = depiler(chemin)
    lislab[2*xp+1][2*yp+1] = cc
    while not est_vide(chemin):
        (x,y) = depiler(chemin)
        lislab[2*x+1][2*y+1] = cc
        lislab[x+xp+1][y+yp+1] = cc
        xp,yp = x,y
    return lislab
```

Nous obtenons donc le code suivant :

```
## Variables importantes

cc = 12 #quelques couleurs...
cb = 15

def revdup(p):
    """Prend en argument la pile p
    retourne une pile similaire à p et
    une pile contenant p retournée"""
    n = taille(p)
    s,t = creer_pile(n),creer_pile(n)
    for i in range(n):
        v = depiler(p)
        empiler(s,v)
        empiler(t,v)
    for j in range(n):
        empiler(p,depiler(s))
    return t

def labyrinthe():
    # Création de lislab, tableau de dimensions (2*n+1)
    #contenant la représentation du labyrinthe

    lislab = [[0]*(2*n+1) for i in range(2*n + 1)]

    for i in range(n):
        for j in range(n):
            lislab[2*i+1][2*j+1] = cb
```

```
    pile = creer_pile(n*n)
    chemin = False

    empiler(pile, (0,0))
    visiter((0,0))

    while not est_vide(pile):
        cellule = depiler(pile)

        #construction du chemin à la volée
        if cellule == (n-1,n-1) and not(chemin):
            empiler(pile,cellule)
            chemin = revdup(pile)
            cellule = depiler(pile)
            print(taille(chemin))

        #print(cellule)
        c = choix(cellule)
        if len(c) > 0:
            suivante = tirage(c)
            # Lie les deux cases
            xc,yc = cellule
            xs,ys = suivante
            #lislab[cellule[0] +1+ suivante[0]][cellule[1]+1 +suivante[1]] =
                cb
            lislab[xc+xs+1][yc+ys+1]=cb
            visiter(suivante)
            empiler(pile, cellule)
            empiler(pile, suivante)
        # fin du while

    #ajout du chemin a lislab
    xp,yp = depiler(chemin)
    lislab[2*xp+1][2*yp+1] = cc
    while not est_vide(chemin):
        (x,y) = depiler(chemin)
        lislab[2*x+1][2*y+1] = cc
        lislab[x+xp+1][y+yp+1] = cc
        xp,yp = x,y
    return lislab
```