Louise Martens
Olivier Elophe
Christophe Marchant

Projet Labyrinthe – première NSI

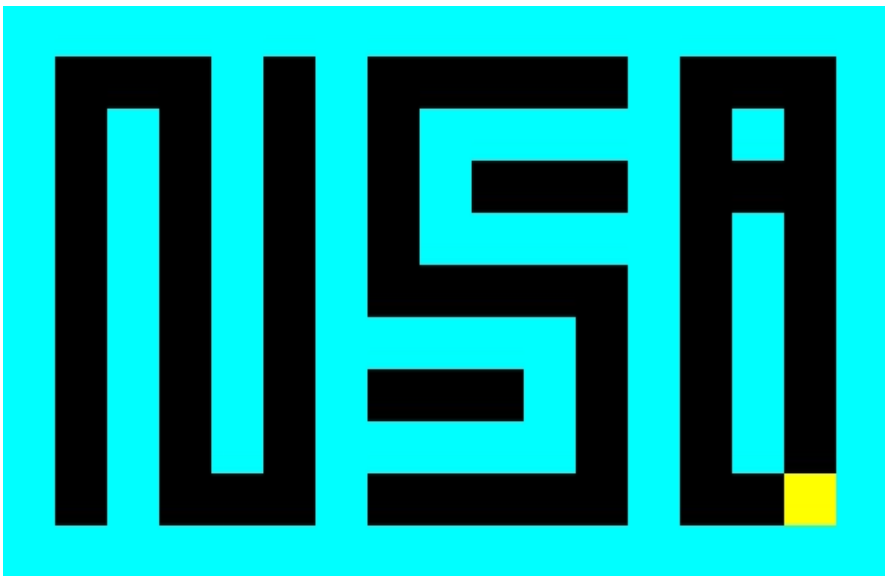
Présentation – Descriptif

Version 0.84.42 du 19 juillet 2017

Plan :

1. Objectifs visés :

- Un « projet collectif » proposé tôt dans l'année,
- introduire et travailler des éléments du programme,
- développer des compétences au programme,
- prendre en main la bibliothèque *pygame* peut-être utile pour certains projets de groupes d'élèves,
- présenter la gestion d'un projet,
- une approche fonctionnelle et modulaire de projet.



2. Lien avec le programme

2. a. Compétences liées au programme :

- Analyser et modéliser un problème en termes de flux et de traitement d'informations ;
- décomposer un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions ;
- concevoir des solutions algorithmiques ;
- traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes.

2. b. Démarche de projet et image

« Un enseignement d'informatique ne saurait se réduire à une présentation de concepts ou de méthodes sans permettre aux élèves de se les approprier en développant des projets applicatifs. »

« En classe de première comme en classe terminale, ils peuvent porter sur des problématiques issues d'autres disciplines et ont essentiellement pour but d'imaginer des solutions répondant à l'expression d'un besoin (...) Il peut s'agir d'un approfondissement théorique des concepts étudiés en commun, (...) d'un problème de traitement d'image, (…) d'un programme de jeu de stratégie, etc. »

« La gestion d'un projet inclut des points d'étape pour faire un bilan avec le professeur, valider des éléments, contrôler l'avancement du projet ou adapter ses objectifs, voire le redéfinir partiellement, afin de maintenir la motivation des élèves. »

2. c. Éléments de programme

Représentation des données : types construits

Tableau indexé	Lire et modifier les éléments d'un tableau grâce à leurs index. Utiliser des tableaux de tableaux pour représenter des matrices : notation a[i][j]. Itérer sur les éléments d'un tableau.

Traitement de données en tables

Indexation de tables	Importer une table depuis un fichier texte tabulé ou un fichier CSV.

Langages et programmation

Spécification	Prototyper une fonction. Décrire les préconditions sur les arguments. Décrire des postconditions sur les résultats.
Utilisation de bibliothèques	Utiliser la documentation d'une bibliothèque.

Algorithmique

Parcours séquentiel d'un tableau	Écrire un algorithme de recherche d'une occurrence sur des valeurs de type quelconque

3. Progression pédagogique

Généralités

La démarche de projet prend une place centrale en informatique.

Notre expérience de la gestion de projets en spécialité ISN de terminale nous amène à une conclusion : l'introduction de mini-projets ou projets collectifs dès le début de l'année est essentielle pour que les projets de fin d'année démarrent « bien ».

Ce « bien » mérite d'être développé : un bon projet de groupe, indépendamment du choix du sujet, sera sur de meilleurs rails si :

- l'objectif final est bien cadré,
- des versions intermédiaires avec objectifs limités et progressifs sont prévus,
- les « parties » de chaque membre du groupe sont délimitées,
- le travail initial « papier, stylo » de délimitation des tâches avec des fonctions bien prototypées, des constantes partagées… a été bien fait, supervisé et validé,
- les structures de données choisies sont adaptées, partagées, etc.

Ce que nous proposons dans ce travail de groupe est une succession de séances à placer de manière assez précoce en première, un fil conducteur qui aboutit à un « *serious game* » de **modélisation de sortie** dans un certain type de **labyrinthe**, lui-même **généré par un algorithme**.

La partie *glamour* du travail repose sur l'utilisation de la bibliothèque *pygame*, que les élèves utilisent massivement pour les productions ISN, au moins les deux-tiers de ce que nous évaluons au baccalauréat aujourd'hui.

Mais les objectifs prioritaires sont bien

- le travail sur des **tableaux a[i][j]**,
- le travail sur
  - la lecture et l'écriture d'un fichier texte, les conversions de types *int* et *str* liées,
  - les images numériques, les pixels et leurs positions, les couleurs,
- et **surtout l'élaboration et la programmation de deux algorithmes** :
  - génération d'un labyrinthe « creusé » avec un certain nombre de contraintes,
  - le parcours en profondeur « le chemin le plus à droite » de ce labyrinthe pour « sortir ».

Cette **proposition en six ou sept séances de TP de deux heures** prend du temps dans la progression annuelle mais permet d'**introduire des concepts et de prendre de bonnes habitudes** essentielles à l'apprentissage de l'informatique, notamment par l**a démarche de projet.**

Prolongement en terminale ?

Interprétation en terme de **graphe** et reprise du **parcours en profondeur.**

4. Descriptif détaillé des séances

Nombre de séances : 6 ou plutôt 7 : **les séances 3, 4 et 5 décrites ci-après sont prévues sur quatre ou cinq plages de TP de deux heures**
Durée d'une séance : 2h

Séance 1 :

- Prérequis ou redécouvertes :
  - codage RVB des couleurs,
  - pixels dans une image numérique matricielle,
  - liste (parcours d'une liste avec une boucle for).
- Objectifs :
  - utilité de *pygame* pour notre projet,
  - coordonnées dans une fenêtre,
  - gestion des événements souris,
  - utiliser une liste de coordonnées.
- Déroulement de la séance :
  - Le code de base est distribué (voir annexes)
  - commenter chaque ligne du programme pour le comprendre,
  - afficher les coordonnées de l'événement souris lorsque du clic,
  - modifier le programme pour afficher seulement les 100 derniers points.
- Aller plus loin :
  - gestion événements clavier pour changer la couleur de fond.
- Lors du corrigé (voir annexes) :
  - découpage d'un projet en plusieurs fichiers :
    - création d'un fichier de constantes, par convention écrites en majuscules,
    - import dans les autres fichiers.
  - Bonnes pratiques :
    - commenter le code,
    - choisir des noms de variables et/ou de fonctions explicites et adaptés,
    - camelCase ?

Séance 2:

- Prérequis:
  - ceux de la séance 1
  - les opération **%** et // en ***python***
- Objectifs:
  - tableau double entrée
- Déroulement de la séance:
  - ***Travail en îlots***
  - on montre **aux élèves au TBI** le script en action (mais pas le code) qui présente une fenêtre :
    - lorsque l'on clique un carré devient de plus en plus blanc (nuance de gris)
    - puis noir quand on dépasse 255 (en fait de 50 en 50)
  - le ***shell*** affiche des éléments qui permettent d'introduire notre liste de listes « **grille** »
  - Une seule consigne : ***mettez vous en groupes de 2 ou 3 et réalisez ce script !***
  - ***Le script reste en action pendant tout le TP au TBI et les élèves peuvent aller cliquer au TBI*** sans regarder le code, bien évidemment !
  - Les élèves peuvent se déplacer, s'aider, demander de l'aide :-)
- Erreurs à prévoir
  - la case d'abscisse x et d'ordonnée y s'accède avec **grille[y][x]**

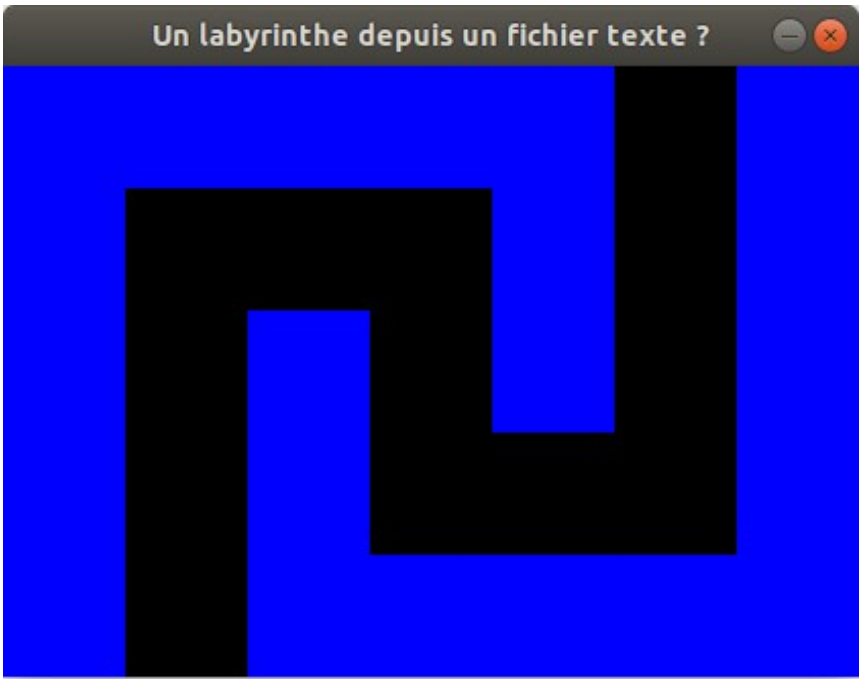
Une idée de la démonstration initiale faite aux élèves au TBI :

```
clic en (77,64)
case : (0,0)
[[50, 0, 0], [50, 50, 0], [0, 0, 0]]
clic en (285,44)
case : (1,0)
[[50, 50, 0], [50, 50, 0], [0, 0, 0]]
clic en (285,44)
case : (1,0)
[[50, 100, 0], [50, 50, 0], [0, 0, 0]]
clic en (432,159)
case : (2,1)
[[50, 100, 0], [50, 50, 50], [0, 0, 0]]
clic en (432,159)
case : (2,1)
[[50, 100, 0], [50, 50, 100], [0, 0, 0]]
clic en (432,159)
case : (2,1)
[[50, 100, 0], [50, 50, 150], [0, 0, 0]]
clic en (424,156)
case : (2,1)
[[50, 100, 0], [50, 50, 200], [0, 0, 0]]
clic en (87,258)
case : (0,2)
[[50, 100, 0], [50, 50, 200], [50, 0, 0]]
clic en (87,258)
case : (0,2)
[[50, 100, 0], [50, 50, 200], [100, 0, 0]]
clic en (87,258)
case : (0,2)
[[50, 100, 0], [50, 50, 200], [150, 0, 0]]
clic en (87,258)
case : (0,2)
[[50, 100, 0], [50, 50, 200], [200, 0, 0]]
clic en (87,258)
case : (0,2)
[[50, 100, 0], [50, 50, 200], [250, 0, 0]]
clic en (115,153)
case : (0,1)
[[50, 100, 0], [100, 50, 200], [250, 0, 0]]
```

A la page suivante, **un bon code commenté** vaut de grandes phrases ;-)

```python
import pygame
from pygame.locals import *
pygame.init()

fenetre=pygame.display.set_mode((600,300))
pygame.display.set_caption("Des cases ?")

fenetre.fill((255,255,255))

continuer=True

# notre grille "tableau à double entrée"
grille=[[0,0,0],[0,0,0],[0,0,0]]

while continuer:
    for i in range(9):
        c=i//3      # cette astuce d'affichage devra certainement être
        l=i%3       # expliquée aux élèves plusieurs fois ;-)
        # la lecture dans le tableau se fait avec la ligne d'abord
        # colonne = c et ligne = l
        gris=grille[l][c]   # on se cantonne au niveaux de gris ici
        # le dessin du rectangle en niveaux de gris
        pygame.draw.rect(fenetre,(gris,gris,gris),(c*200,l*100,200,100),0)

    #gestion des événements
    for event in pygame.event.get():
        if event.type==QUIT:
            continuer=False
        elif event.type==KEYDOWN:
            if event.key==K_ESCAPE:
                print("touche Escape")
                continuer=0
            else :
                print("autre touche")
        elif event.type==MOUSEBUTTONDOWN : # on a cliqué !
            # le pixel du clic
            (x,y)=event.pos
            print("clic en ("+str(x)+","+str(y)+")")
            # qui correspond à une des neuf cases
            c=x//200
            l=y//100
            print("case : ("+str(c)+","+str(l)+")")
            # on augmente l'intensité
            grille[l][c]+=50
            # on teste si on dépasse les bornes des limites
            # on aurait pu faire grille[l][c]=(grille[l][c]+50)%300
            if grille[l][c]>255:
                grille[l][c]=0
            # affichage de la grille pour comprendre quelque chose
            # à la choucroute
            print(grille)
    # mise à jour de l'affichage
    pygame.display.flip()
pygame.quit()
```

Séance 3 :



- Prérequis:
  - les séances précédentes et leurs pré-requis
  - les opération `%` et `//` en ***python***
  - manipuler les fichiers textes avec python
- Objectifs:
  - toujours tableau double entrée **`grille[y][x]`**
  - découpage fonctionnel
- Déroulement de la séance:
  - Premier code :
    - affichage d'une grille à l'aide d'une fonction, squelette de code fourni.
    - modification de cette grille
  - Deuxième code :
    - lecture d'un fichier texte avec des 0 et des 1 et afficher le labyrinthe,
    - modifier à la souris le labyrinthe et l'enregistrer en écrasant le fichier texte.
  - Les élèves peuvent se déplacer, s'aider, demander de l'aide :-)
- Difficultés à prévoir
  - la case d'abscisse x et d'ordonnée y s'accède avec **`grille[y][x]`**
  - open en mode r et w
  - for ligne in fichier
  - for c in ligne
  - connaître **`'\n'`**
  - conversions **`str()`** et **`int()`**

Lors du corrigé avec les élèves, on va utiliser ce TP3 pour leur montrer comment séparer en deux scripts .py :

- un pour les E/S fichiers,
- un autre qui l'importe pour pygame et la gestion de la souris

Premier Code de la séance 3

Les zones surlignées ne sont pas fournies aux élèves (voir fichiers en annexe)

On peut décrire un Labyrinthe à l'aide de 1 et de 0. On suppose que le labyrinthe est « creusé » s'il y a 0.

On peut demander aux élèves de dessiner sur une feuille à carreaux le labyrinthe décrit par la grille.

```
import pygame
from pygame.locals import *
pygame.init()

largeur,hauteur = 7,5   # dimensions de la grille
taille = 60             # taille d'une case en pixels

fenetre=pygame.display.set_mode((largeur*taille,hauteur*taille))
pygame.display.set_caption("Un labyrinthe depuis un fichier texte ?")

# notre grille initiale "tableau à double entrée"
grille=[[1,1,1,1,1,0,1],
        [1,0,0,0,1,0,1],
        [1,0,1,0,1,0,1],
        [1,0,1,0,0,0,1],
        [1,0,1,1,1,1,1]]

#fonction permettant l'affichage de la grille, envoyée en argument
def affichage(grille) :
    fenetre.fill((0,0,0))
    for l in range(hauteur):
        for c in range(largeur):
            if grille[l][c] :
                pygame.draw.rect(fenetre,(0,0,255),(c*taille,l*taille,taille,taille),0)
    pygame.display.flip()

continuer=True

while continuer:

    affichage(grille)

    #gestion des événements
    for event in pygame.event.get():
        if event.type==QUIT:
            continuer=False
        elif event.type==KEYDOWN:
            if event.key==K_ESCAPE:
                print("touche Escape")
                continuer=0
            else :
                print("autre touche")
        elif event.type==MOUSEBUTTONDOWN : # on a cliqué !
            # le pixel du clic
            (x,y)=event.pos
            print("clic en ("+str(x)+","+str(y)+")")
            # qui correspond à la case de grille en ligne l et en colonne c
            c=x//taille
            l=y//taille
            print("case : ("+str(c)+","+str(l)+")")
            # on passe de 0 à 1 ou de 1 à 0
            grille[l][c]=1-grille[l][c]
            # affichage de la grille en console
            print(grille)

pygame.quit()
```

Deuxième partie de la séance 3

On fournit aux élèves le fichier texte « **carte.txt** » qui contient :

```
111111111
100000001
111110101
101000101
101010111
100010001
111111111
```

On leur demande

1) d'écrire une fonction **lecture()** qui :
- acceptera comme argument le nom du fichier texte supposé dans le même répertoire que le script,
- renverra
  - la **grille** d'entiers (zéro ou un) comme dans le premier code,
  - la **largeur** et la **hauteur** de cette grille.

Lors de l'événement « appui sur la **touche l** » (comme **l**ecture), le script
- appelle la fonction **lecture()** et remplace donc la grille par celle lue,
- rafraîchit la **fenetre** en tenant compte de la nouvelle **largeur** et la nouvelle **hauteur**.

2) d'écrire une fonction **ecriture()** qui :
- acceptera comme arguments le nom du fichier texte à enregistrer dans le même répertoire que le script, la **grille** à enregistrer, la « **hauteur** » et la « **largeur** » de la grille,
- écrasera le fichier texte éventuellement existant par un fichier contenant des 0 et des 1 par lignes comme l'exemple fourni en lisant la **grille** d'entiers (zéro ou un).

Lors de l'événement « appui sur la **touche e** » (comme **e**nregistrement), le script appelle la fonction **ecriture()**.

Lors du corrigé, on présente deux scripts :

- le script **TP32AvecUneGrilleCliquableEtUnFichier.py** de gestion de la souris et des événements, et d'affichage de la grille,
- le script **TP32GestionDuFichier.py** contenant les fonctions **ecriture()** et la fonction **lecture()**.

Ces scripts sont en pièces jointes.

Gérer l'hétérogénéité :

en fin de séance 3, qui est peut-être le début de la quatrième plage physique de deux heures, on peut faire commencer la création de labyrinthe proposée en début de séance 4 ci-après.

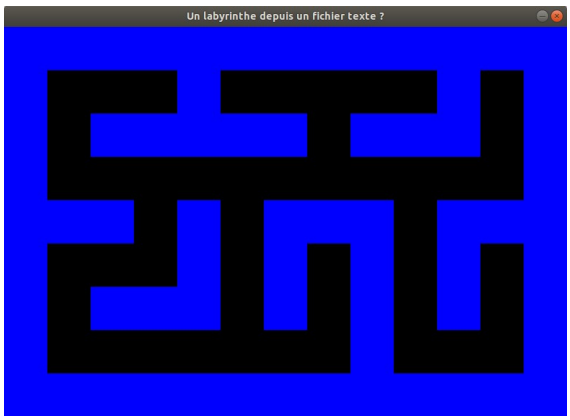
Séance 4 : creuser un labyrinthe

1) Trouver un algorithme « à la main »

Mais plutôt que d'utiliser un papier quadrillé et un crayon, on peut cliquer en utilisant les scripts du TP3 et un fichier **`carte13x9.txt`** contenant 9 lignes de 13 « un » :

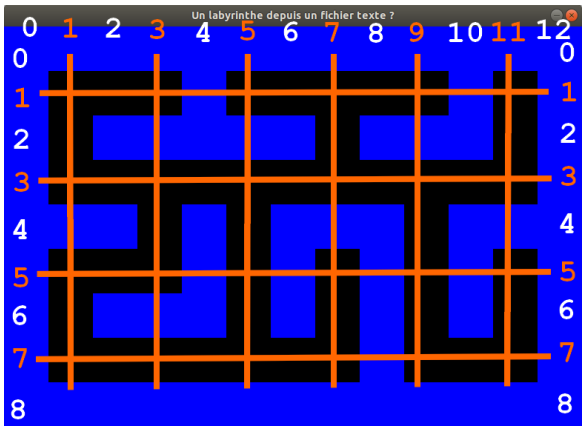
```
1111111111111
1111111111111
1111111111111
1111111111111
1111111111111
1111111111111
1111111111111
1111111111111
1111111111111
```

On demande alors de « creuser » à la souris ***un labyrinthe « optimal quand à l'espace disponible » en gardant des « murs » autour*** et on compare les productions pour aboutir à des créations de ce type :




```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

On projette ceci au TBI et on trace les lignes orange pour aboutir à la conclusion souhaitée : **les cases dont les indices `l` et `c` sont impairs doivent être creusées !!** :



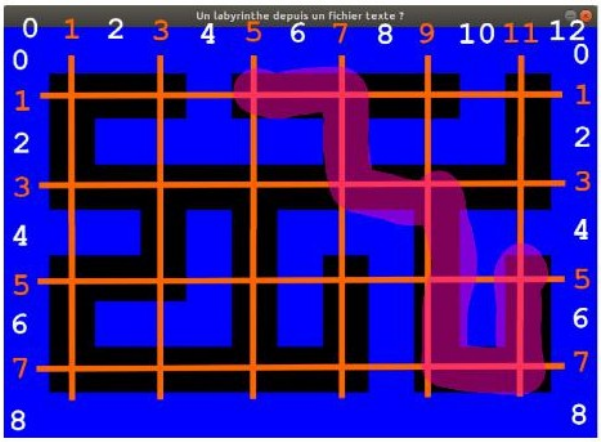

```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

2) D'où l'idée d'un algorithme élaboré avec les élèves :

- **Initialisation** → Tu pars d'**une case orange** et tu creuses, « de deux en deux », « sans sortir » un premier chemin de « longueur raisonnable »
- **Boucle** → **Tant qu'**il reste des **cases oranges** à creuser, tu en sélectionnes une (au hasard) et tu creuses jusqu'à ce que tu « rejoignes » le chemin précédemment creusé.

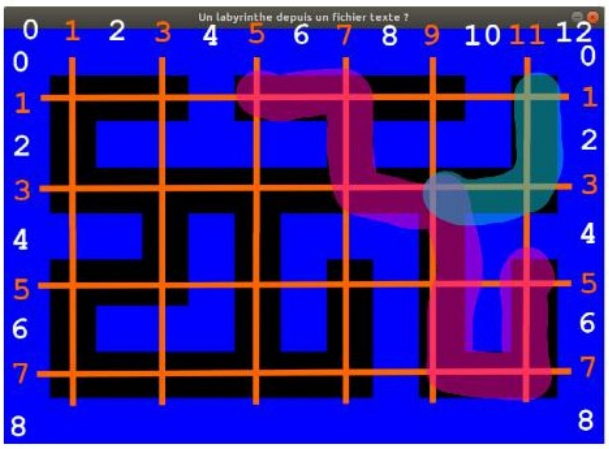
Cet algorithme donne par exemple :

Initialisation → On part d'**une case orange, ici `grille[1][5]=0` à la ligne `l=1` et la colonne `c=5`** et on creuse, « de deux en deux », « sans sortir » **ce chemin** de « longueur raisonnable » **(ici on a creusé 7 fois)**.



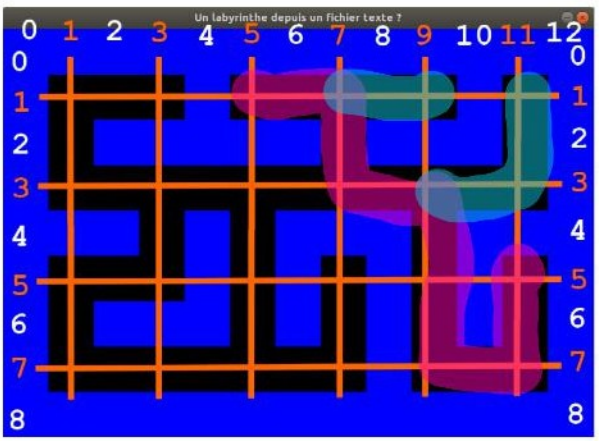

```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Il reste des cases oranges à creuser → on rentre dans le « **tant que** » → premier **tour de boucle** !
Ici on **part de tout en haut à droite `l=1` et `c=11`** et **on creuse deux fois** pour **retomber sur le chemin précédent**.



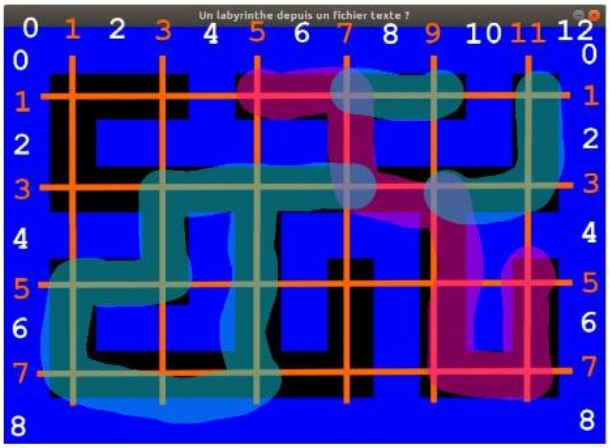

```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Il reste des cases oranges à creuser → on rentre à nouveau dans le « **tant que** » → 2ème **tour de boucle** !
Ici on **part de `l=1` et `c=9`** et **on creuse une fois** pour **retomber sur le chemin précédent**.




```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Il reste des cases oranges à creuser → on rentre à nouveau dans le « **tant que** » → 3ème **tour de boucle** !
Ici on **part de `l=3 et c=5`** et **on creuse 9 fois** pour **retomber sur le chemin précédent**.




```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Attention, ici moment clé et explication précise :

- on **part de `l=3 et c=5`** donc on pose `grille[3][5]=0`
- et on creuse « vers le bas » :
  `grille[4][5]=grille[5][5]=grille[6][5]=grille[7][5]=0`
- puis « à gauche » :
  `grille[7][4]=grille[7][3]=grille[7][2]=grille[7][1]=0`
- puis « vers le haut » :
  `grille[6][1]=grille[5][1]=0`
- puis « à droite » :
  `grille[5][2]=grille[5][3]=0`
- puis « vers le haut » :
  `grille[4][3]=grille[3][3]=0`
- puis « à droite » :
  `grille[3][4]=grille[3][5]=0`

Mais cette dernière égalité était déjà vraie, c'est notre point de départ orange dans ce « tour de boucle » !

Ceci ne saurait constituer un test d'arrêt !

La condition d'arrêt de ce « tour de boucle » dans l'algorithme est : tu creuses jusqu'à ce que tu « rejoignes » le **chemin précédemment creusé**.

On continue donc :

- puis « à droite » :

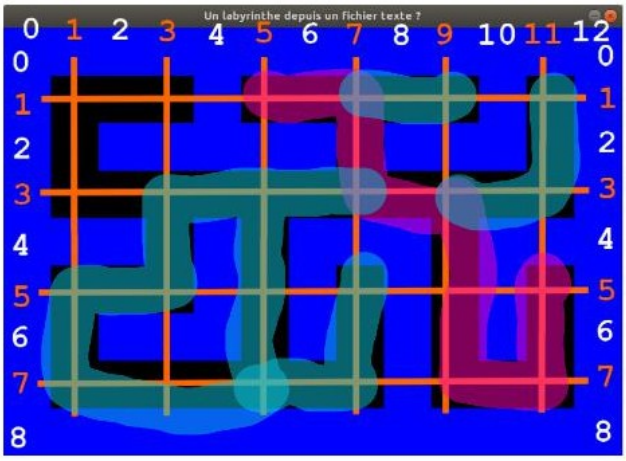
  `grille[3][6]=0` puis `grille[3][7]` mais là on est tombé **dans le chemin précédemment creusé** : **on s'arrête**. (pour ce « tour de boucle »)

Ouf. Mais ceci soulève **un sacré problème supplémentaire dans l'implémentation** : on va devoir **à chaque tour de boucle** créer une **copie de la grille précédente pour pouvoir obtenir un test d'arrêt** avec **le chemin précédemment creusé** .

En particulier, il s'agira de **copier une liste** (de listes) et pas de copier une adresse…

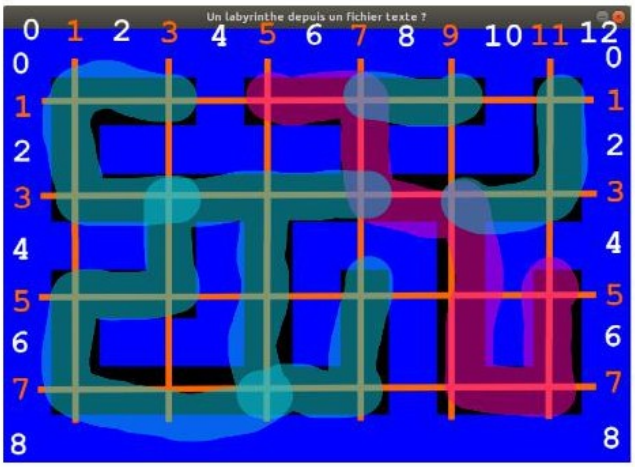
Continuons, notre algorithme n'est pas arrivé à son terme.

Il reste des cases oranges à creuser → on rentre à nouveau dans le « **tant que** » → 4ème **tour de boucle** !
Ici on **part de `l=5` et `c=7`** et **on creuse 2 fois** pour **retomber sur le chemin précédent**.




```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Il reste des cases oranges à creuser → on rentre à nouveau dans le « **tant que** » → 5ème **tour de boucle** !
Ici on **part de `l=1` et `c=3`** et **on creuse 3 fois** pour **retomber sur le chemin précédent**.




```
1111111111111
1000100000101
1011111011101
1000000000001
1110101110111
1000101010101
1011101010101
1000000010001
1111111111111
```

Il n'y a plus de cases oranges à creuser → **l'algorithme est terminé !**

Alors, c'est très beau, surtout avec les jolis dessins colorés du prof au TBI, mais il va falloir l'implémenter !

3) Implémentation

C'est ici qu'il faut un **découpage fonctionnel** de qualité.

On n'est plus du tout dans l'aspect graphique, pas de ***pygame*** ici, mais on travaille sur les **`a[i][j]`** comme le stipule le programme !

Mise en œuvre :

- On crée ensemble au TBI ou au vidéoprojecteur le squelette du script « qui creuse » le labyrinthe.
- Ensuite les élèves vont essayer d'implémenter chacune des fonctionnalités.
- Mise en commun en fin de TP.

Avec des fonctions :

- une qui crée un grille pleine de `1` prête à « creuser ».
- Une qui compte le nombre de cases oranges à creuser, c'est-à-dire les **`grille[l][c]==1` pour `l` et `c` impairs**, il suffit de **retourner la somme** des **`grille[l][c]` pour `l` et `c` impairs**.
- Une qui **retourne une copie de la grille** pour le chemin précédent (voir 3ème tour de boucle).
- Une qui prend un point de départ aléatoire dans les cases oranges disponibles.
- Deux qui cherchent les « destinations possibles » depuis une case orange sans sortir de la grille « deux cases plus loin » :
  - une qui cherche des cases « encore à creuser » donc avec un `1`,
  - une avec possibilité de trouver un zéro, pour les « tours de boucle »
- Une qui creuse le chemin initial en
  - tirant au sort un point de départ grâce à la fonction déjà créée,
  - tirant aléatoirement une direction dans les « destinations possibles » avec des `1`,
  - vérifiant que « deux cases plus loin », on est encore dans la grille et pas encore à zéro,
  - mettant des `grille[l][c]` à zéro, avant de recommencer,
  - jusqu'à ce qu'on ait la bonne « longueur raisonnable »…
- Une qui à chaque tout de boucle, creuse un chemin en
  - tirant au sort un point de départ grâce à la fonction déjà créée,
  - tirant aléatoirement une direction dans les « destinations possibles », éventuellement avec zéro,
  - mettant des `grille[l][c]` à zéro, avant de recommencer,
  - jusqu'à ce qu'on ait trouvé **un zéro de la grille précédente** !

Attention :

- Les indices vont de `0` à `hauteur-1` pour `l` et de `0` à `largeur-1` pour `c`.
- Aller « en haut », c'est ***décrémenter*** `l` !
- Il faudra **aider les élèves pour les cases oranges, `l` et `c` impairs**, il y a par exemple pour `l` :
  - `hauteur//2` possibilités,
  - de `1` à `2*(hauteur//2)-1`…
  - **le plus simple à nos yeux : `for y in range(hauteur//2) : l=2*y+1`**
- On a besoin de la bibliothèque ***random***, dont nous avons utilisé `randrange` et `choice`.
- Quid de la « longueur raisonnable » du « chemin initial » ?
  - Nous l'avons posée arbitrairement à un petit tiers du nombre de cases oranges : `t=(hauteur//2)*(largeur//2)//3`
  - C'est une longueur maximale si « on se mort la queue » !

Indécis ?

Tester le script **`TP4DemoProf.py`** dans le dossier TP4, il permet de visualiser l'algorithme en œuvre avec une animation pygame, ses qualités, et ses défauts quand on ne trouve pas rapidement le chemin initial !

Séance 5 - parcours à droite du labyrinthe (donc en profondeur)

Cette séance ludique a pour objet de créer un script qui permette de visualiser l'animation du parcours de ce labyrinthe « en profondeur », c'est à dire ici « toujours à droite ».

On repart avec une carte générée au TP4. Nous proposons de l'animer en groupe entier, avec des élèves qui manipulent pour coder au TBI, après l'approche papier/crayon animée collectivement.

Il nous manque :

- un départ et une arrivée,
- un « personnage » ou indicateur de position dans le labyrinthe : la donnée de deux coordonnées `x` et `y`.

On décide :

- de placer le départ sur la ligne zéro et sur une colonne impaire aléatoire.

  → On est alors certain de pouvoir « entrer » dans le labyrinthe en « descendant », il y a une « case orange » au dessous.
- De placer de même l'arrivée sur la ligne hauteur-1 et sur une colonne impaire aléatoire.

  → On est alors certain de pouvoir « sortir » du labyrinthe en « descendant », il y a une « case orange » au dessus.
- De coder le départ avec un 3 (impair)
- De coder l'arrivée avec un 2 (pair).
- Ainsi, on peut « déplacer le personnage dans la grille » sur une case telle que

  `grille[l][c]==0` ou `grille[l][c]==2` c'est à dire `grille[l][c]%2==0` ;-)
- On code, comme dans les directions possibles pour creuser au TP4, la direction du personnage par une variable `direction` :
  - droite → `direction = 0`,
  - bas → `direction = 1`,
  - gauche → `direction = 2`,
  - haut → `direction = 3`.
- On pose le personnage « en dessous » du départ avec la direction « bas » pour débuter.
- Et on essaie toujours d'aller le plus à droite possible donc on teste

  `direction=(direction+1)%4`
- et si ça ne va pas on teste plus à gauche

  `direction=(direction+3)%4` qui correspond à `direction-=1` en restant dans `0,1,2,3` …

Découpage fonctionnel :

- une fonction **`affichage(grille,largeur,hauteur,x,y)`** qui affiche la grille et le personnage et patiente un dixième de seconde avec **`pygame.time.delay(100)`**.
- une fonction **`newCoord(x,y,direction)`** qui accepte comme arguments les coordonnées du « personnage » et sa « **`direction`** » renvoie les nouvelles coordonnées en fonction de la **`direction`**.
- Une fonction **`deplacement(grille,l,h,x,y,d)`** qui accepte comme arguments :
  - la grille creusée **`grille`**,
  - la largeur **`l`**,
  - la hauteur **`h`**,
  - les coordonnées du personnage **`x`** et **`y`**,
  - la direction **`d`**,
- et renvoie **`d,x,y`** :
  - les nouvelles coordonnées du personnage **`x`** et **`y,`**
  - la nouvelle direction **`d`**.
- une fonction **`parcours(grille,largeur,hauteur)`** qui « déplace » ainsi le personnage en appelant **`deplacement(grille,l,h,x,y,d)`** jusqu'à l'arrivée et appelle la fonction d'affichage **`affichage(grille,largeur,hauteur,x,y)`** à chaque fois, jusqu'à ce que l'on atteigne l'arrivée.

Indécis ?

Tester le script **`TP5DemoProf.py`** dans le dossier TP4, il permet de visualiser les deux algorithmes en action !