Lilian BUZER

LABYRINTHE

Nous allons étudier un classique du jeu : un personnage qui cherche la sortie d'un donjon et qui lors de son périple ramasse quelques trésors sur son passage. Nous allons en profiter pour présenter en python comment stocker des données.

LISTE, TABLEAU, DICTIONNAIRE

Nous allons vous présenter une syntaxe pour lire et écrire des informations :

LISTE 1D :

```
(1)   L = ['titi', 5, 'toto']
      print(L)          →   [' titi', 5, 'toto']
      print(L[0])       →   'titi'
      L[1] += 1
      L[2] = 'tata '
      print(L)          →   ['titi',  6,  'tata']
      len(L)            →   3
```

TABLEAU nD :

```
(3)   T = np.zeros((2,2))          // import numpy as np
      T[0,0] = 1
      T[0,1] = 2
      T[1,1] = 3
(2)   print(T)          →   array([[ 1., 2.], [ 0., 3.]])
      T.shape[0]        →   2
```

Pourquoi des listes et des tableaux ? Une liste fonctionne avec un seul indice, le tableau peut avoir 1 à plusieurs indices. La liste peut stocker des éléments de différents types alors que le tableau contient des éléments de même type (2). La liste peut s'initialiser rapidement en donnant l'ensemble des éléments séparés par une virgule (1). Il n'y a pas d'équivalent pour les tableaux, on peut cependant initialiser toutes les valeurs du tableau grâce à la fonction zeros((…)) (3).

Il existe aussi les dictionnaires. Ils se comportent comme les listes sauf qu'ils peuvent être indexés sur tout type de données comme notammaent des chars ou des strings : D['age_du_capitaine'] = 44 :

Lilian BUZER

DICTIONNAIRE

```
D = { }
D['mami']  = 76
D['mami']  += 1
print(D['mami'])        → 77
D['papi']   = 74
print(D)                → {'mami': 77, 'papi': 74}
```

En résumé :

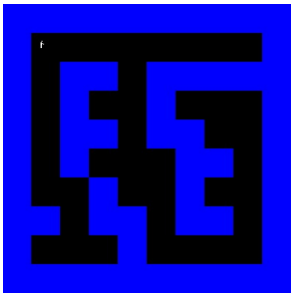
- J'ai plusieurs dimensions à gérer → Tableau
- J'ai une dimension indexée sur des string/char → Dictionnaire
- Sinon → Liste devrait convenir

Présentation

Pour nous simplifier la vie, nous stockons le plan du labyrinthe dans une liste de string :

```
plan = [ 'BBBBBBBBBB',
         'B        B',
         'B BB BBBBB',
         'B B  B   B',
         'B BB BB  B',
         'B B   BB B',
         'B  B  B  B',
         'BB BB BB B',
         'B   B    B',
         'BBBBBBBBBB' ]
```

→



Cela nous permet de représenter le plan directement dans le code. L'idée ici est que chaque lettre représente la couleur de la case du labyrinthe. Nous avons créé un dictionnaire palette, qui associe à chaque lettre une couleur :

```
# crée une palette de couleurs
palette = {} # initialise un dictionnaire
palette['B'] =  [  0,   0, 255]   # BLUE
palette[' '] =  [  0,   0,   0]   # BLACK
palette['W'] =  [255, 255, 255]   # WHITE
palette['G'] =  [  0, 255,   0]   # GREEN
palette['R'] =  [255,   0,   0]   # RED
```

Ainsi, palette['B'] donne la couleur bleu et palette[' '] la couleur noire. Nous convertissons ensuite notre « plan » vers un tableau nommé LABY. Ainsi, à la fin de cette conversion, LABY[x,y] désigne la couleur de la case [x,y] du labyrinthe.

```
# remplissage du tableau du labyrinthe
LABY  = np.zeros((NBcases,NBcases,3))
for y in range(NBcases):
    ligne = plan[y]
    for x in range(NBcases):
        c = ligne[x]
        LABY[x,y] = palette[c]
```

Nous procédons exactement de la même manière pour définir le sprite de notre aventurier :

```
persl= [ '   RRR    ',
         '  RRWWR   ',
         '   RRR    ',
         '   YY     ',
         '   YYY    ',
         '   YY YG  ',
         '   GG     ',
         '   CC     ',
         '   CC     ',
         '  C  C    ',
         '  C  C    ' ]
```

→ 

La fonction ToSprite(…) permet de convertir une liste de string en sprite pygame. Son principe est identique au fonctionnement du remplissage du tableau.

TODOLIST

- Faites en sorte que l'aventurier se déplace grâce aux flèches du clavier : ← →
- Utilisez le deuxième sprite de l'aventurier. Deux fois par seconde, alternez les deux sprites pour donner l'impression que le personnage marche en permanence.
  - ♣ int(pygame.time.get_ticks()/500) augmente de 1 toutes les 500ms
  - ♣ La fonction modulo % devrait vous aider

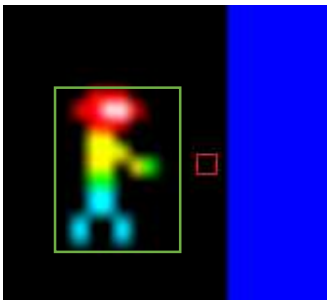
Lilian BUZER

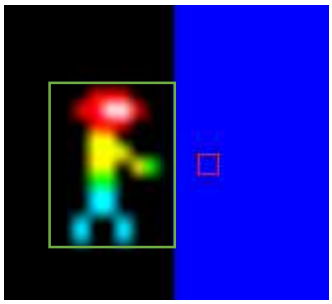
- Créez un sprite trésor (clef, porte..) qui symbolise la sortie, affichez le dans le jeu sur une case vide

- Lorsque la distance entre la sortie et le sprite de l'aventurier est inférieur à 5, affichez « WIN » en gros au milieu de l'écran. Nous vous rappelons comment calculer une distance :

  $$d((x1,y1) , (x2,y2))^2 = (x1-x2)^2 + (y1-y2)^2$$

- Notre aventurier a tendance à traverser les murs. Ce n'est pas acceptable ! Par exemple s'il avance vers la droite, nous vous proposons de tester sa collision en regardant la couleur du pixel à sa droite :





  Son sprite est symbolisé par un rectangle vert. Nous regardons la couleur du pixel devant le sprite à mi-hauteur. Si la couleur est noire, le joueur peut avancer, si la couleur est bleue, cela veut dire que le joueur est face à un mur et qu'il ne doit pas avancer dans cette direction.

  ♣ On doit lire la couleur du pixel après avoir dessiné le labyrinthe, pas avant !

  ♣ La fonction screen.get_at((x,y)) retourne la couleur du pixel de l'écran

  ♣ Les crochets [0] [1] et [2] permettent de récupérer les composantes R,V,B de la couleur

  ♣ Les fonctions xxx.get_width() et xxx.get_height() donnent les dimensions d'un sprite

- Gérez les collisions dans les 4 directions de déplacement

- Créez un labyrinthe 20x20. Pour cela, diminuez la taille en pixel des cases par deux. Ainsi la fenêtre de jeu ne change pas de taille. Pensez à créer un parcours complexe avec plusieurs chemins possibles et des emplacements pour des pièges. La sortie n'est pas forcément en bas à droite !

Lilian BUZER

- Créez un sprite piège : mine, trappe, flamme, bombe. Positionnez un piège dans le jeu

- Lorsque le joueur passe sur le piège, remettez-le à la position de départ

- L'animation de la marche a tendance à clignoter, en effet, nous alternons l'affichage avec deux sprites uniquement, c'est insuffisant. Créez de nouveaux sprites pour l'animation de la marche, il suffit d'améliorer le jeu de jambe du personnage ☺ Modifiez la boucle d'animation en conséquence

- Créez un sprite trésor (coffre, gemme, pièce d'or). Ajoutez plusieurs trésors dans le parcours

- Lorsque le jouer passe à proximité d'un trésor, celui-ci disparaît et le joueur gagne 100 points. Affichez le score en haut de l'écran

- Faites en sorte de positionner plusieurs pièges