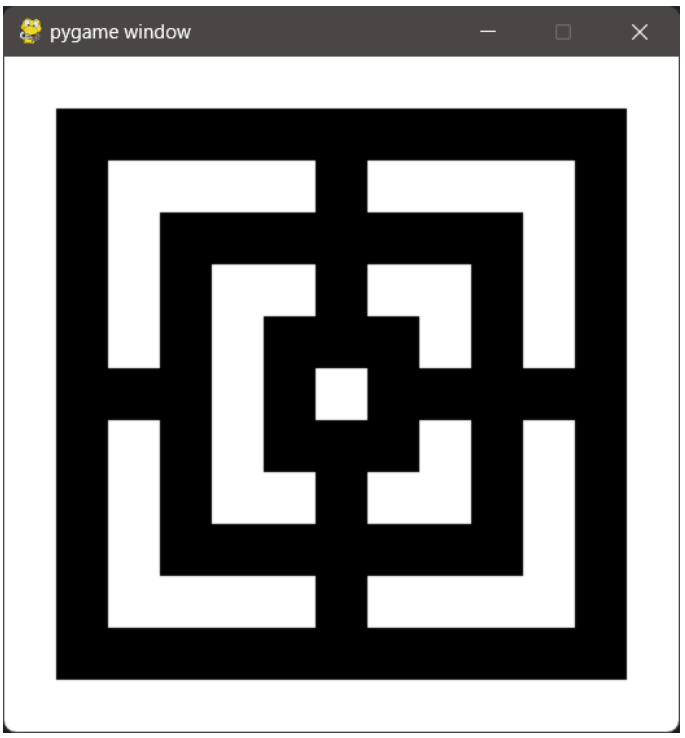
Labyrinthe v0.0 - Affichage

Dans cet exercice, vous allez utiliser **Pygame** pour afficher un labyrinthe simple, où chaque tuile représente soit un mur, soit un sol. Le labyrinthe est fourni sous forme de matrice, où les `1` représentent des murs et les `0` représentent des tuiles de sol.

L'objectif principal de cet exercice est de créer une fonction qui permet d'afficher visuellement le labyrinthe à l'écran à l'aide de Pygame.



1. Contexte de l'exercice

Le labyrinthe est représenté par une matrice 2D où chaque élément de la matrice correspond à une "tuile" dans la grille :

- `1` représente un **mur**.
- `0` représente le **sol**.

Par exemple, voici un labyrinthe sous forme de matrice 5x5 :

```
labyrinthe = [
    [1, 1, 1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 0, 1, 0, 1],
    [1, 0, 0, 0, 1],
    [1, 1, 1, 1, 1]
]
```

But de l'exercice : Créer une fonction qui utilise Pygame pour afficher visuellement ce labyrinthe. Vous allez devoir représenter chaque cellule de la matrice comme une tuile à l'écran.

---

2. Étapes à suivre pour l'exercice

Étape 1 : Initialiser Pygame

Pygame est une bibliothèque qui permet de créer des jeux 2D. La première chose à faire est d'initialiser Pygame et de créer une fenêtre de la taille de votre labyrinthe.

Pour cela, utilisez les méthodes suivantes :

- `pygame.init()` : Cette méthode initialise tous les sous-modules de Pygame. Elle doit être appelée avant d'utiliser toute autre fonctionnalité de Pygame.
- `pygame.display.set_mode((largeur, hauteur))` : Crée une fenêtre de jeu avec la taille spécifiée. La largeur et la hauteur doivent correspondre à la taille du labyrinthe multipliée par la taille de chaque tuile (par exemple, 32x32 pixels par tuile).

Exemple :

```
import pygame

pygame.init()
largeur, hauteur = 320, 320  # Ajustez selon la taille du labyrinthe et des tuiles
fenetre = pygame.display.set_mode((largeur, hauteur))
```

Étape 2 : Définir les couleurs pour les tuiles

Dans Pygame, vous pouvez définir des couleurs en utilisant des triplets RGB (rouge, vert, bleu). Vous devrez définir deux couleurs :

- Une couleur pour le **mur**.
- Une couleur pour le **sol**.

Voici un exemple de définition des couleurs :

```
BLANC = (255, 255, 255)  # Couleur pour le mur
NOIR = (0, 0, 0)         # Couleur pour le sol
```

Étape 3 : Parcourir la matrice avec des boucles imbriquées

Pour afficher chaque tuile de la matrice, vous allez devoir parcourir la matrice avec une double boucle (une pour les lignes et une pour les colonnes). Vous allez utiliser une tuile de taille fixe (par exemple, 32x32 pixels) pour représenter chaque élément du labyrinthe.

Étape 4 : Utiliser la méthode `pygame.draw.rect()` pour dessiner les tuiles

Pour dessiner chaque tuile, utilisez la méthode `pygame.draw.rect()`.

Syntaxe :

```
pygame.draw.rect(fenetre, couleur, (x, y, largeur_tuile, hauteur_tuile))
```

- `fenetre` : La fenêtre dans laquelle vous dessinez.
- `couleur` : La couleur de la tuile (par exemple, `BLANC` pour le sol, `NOIR` pour le mur).
- `x, y` : Les coordonnées en haut à gauche où la tuile sera placée.
- `largeur_tuile, hauteur_tuile` : Les dimensions de la tuile (en pixels).

Étape 5 : Calculer les positions des tuiles

Pour chaque tuile, vous devez calculer sa position en fonction de ses indices dans la matrice. Si vous utilisez des tuiles de 32x32 pixels, alors la position de chaque tuile dans la fenêtre sera donnée par :

© 2025 Mr. Lambrechts

```
x = colonne * largeur_tuile
y = ligne * hauteur_tuile
```

Ainsi, pour chaque élément de la matrice, vous dessinez un rectangle correspondant à sa position.

Étape 6 : Mettre à jour l'affichage avec `pygame.display.flip()`

Après avoir dessiné toutes les tuiles, vous devez mettre à jour la fenêtre pour que les changements soient visibles à l'écran.

Méthode :

```
pygame.display.flip()  # Actualise l'affichage
```

Étape 7 : Gérer la boucle d'événements Pygame

Enfin, vous devez ajouter une boucle d'événements pour garder la fenêtre ouverte et permettre à l'utilisateur de la fermer correctement. Voici une boucle d'exemple :

```
continuer = True
while continuer:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            continuer = False

pygame.quit()  # Quitter Pygame proprement
```

Calculer les coordonnées x et y

Imagine que ton écran est comme une grille de papier avec des cases, et chaque case représente un carré dans ton jeu, un peu comme un quadrillage. Chaque case a un numéro de ligne et de colonne, comme dans une bataille navale. Par exemple, la ligne 1 et la colonne 1 te donnent la première case en haut à gauche.

Pour afficher les choses dans le jeu, on ne parle pas en lignes et colonnes, mais en **pixels**. Un pixel est un tout petit point sur l'écran. Ton écran est fait de milliers de pixels qui sont alignés les uns à côté des autres.

Pourquoi transformer les lignes et colonnes en pixels ?

Les jeux vidéo utilisent des **pixels** pour savoir exactement où dessiner les objets à l'écran. Par exemple, si tu veux placer un mur ou un personnage dans ton jeu, il faut dire à l'ordinateur combien de **pixels** ils doivent être placés depuis le haut (pour la position verticale, "y") et depuis la gauche (pour la position horizontale, "x").

Comment transformer les lignes/colonnes en pixels ?

Si chaque case de ta grille fait 32 pixels de large et de haut, alors pour savoir où dessiner une case, tu multiplies simplement le numéro de la ligne ou de la colonne par 32 :

- **x (horizontale)** = colonne × 32
- **y (verticale)** = ligne × 32

Par exemple, pour la case qui est à la ligne 2, colonne 3 :

- x = 3 × 32 = 96 pixels depuis la gauche
- y = 2 × 32 = 64 pixels depuis le haut

Ainsi, l'ordinateur saura où dessiner exactement chaque objet dans ton labyrinthe !

© 2025 Mr. Lambrechts

3. Plan de la fonction d'affichage du labyrinthe

Résumé des étapes pour créer la fonction

1. **Initialisez Pygame** avec `pygame.init()` et créez une fenêtre.
2. **Parcourez la matrice** avec des boucles imbriquées.
3. **Pour chaque élément de la matrice** :
    - Si c'est un `1`, dessinez un mur (par exemple, un rectangle noir).
    - Si c'est un `0`, dessinez une tuile de sol (par exemple, un rectangle blanc).
4. **Utilisez** `pygame.display.flip()` pour mettre à jour l'affichage.
5. **Gérez la boucle d'événements** pour maintenir la fenêtre ouverte jusqu'à ce que l'utilisateur la ferme.

Conseils :

- **Modularité** : Placez le code d'affichage dans une fonction que vous appellerez après avoir chargé votre labyrinthe.
- **Évitez le code répétitif** : Utilisez des conditions pour choisir la couleur de la tuile en fonction de la valeur dans la matrice (par exemple, `if labyrinthe[i][j] == 1:`).
- **Expérimentez avec les tailles de tuiles** : Vous pouvez ajuster la taille des tuiles pour avoir un rendu plus ou moins grand en fonction de la taille de votre labyrinthe.

Conversion des coordonnées

On gère l'écran de jeu comme une grille de cases, un peu comme un échiquier, où chaque case a un numéro de **ligne** et de **colonne**. Dans un jeu vidéo, on veut placer des objets (comme des personnages ou des murs) dans certaines cases de cette grille. Pour cela, on utilise les coordonnées de **ligne** et **colonne** pour savoir où placer ces objets. Mais le truc, c'est que l'ordinateur ne comprend pas directement ces numéros de ligne/colonne — il travaille en **pixels**, qui sont les minuscules points lumineux qui forment l'écran.

Pourquoi transformer les lignes et colonnes en pixels ?

Quand tu veux afficher un objet à l'écran, tu dois dire à l'ordinateur exactement où il doit le dessiner. Or, l'écran ne fonctionne pas avec des lignes et colonnes, mais avec des **pixels**, qui sont les points qui composent ton écran. Chaque objet de ton jeu, comme un personnage ou un mur, doit être placé quelque part en fonction de ces pixels. Donc, il faut **convertir les lignes et colonnes en pixels** pour que l'ordinateur sache où dessiner.

Comment transformer les lignes/colonnes en pixels ?

Chaque case de ta grille a une **taille définie**, par exemple, 32 pixels de large et 32 pixels de haut (on appelle ça la **taille d'une tuile**). C'est le concepteur du jeu qui choisit la taille de ses tuiles. Dans le cadre de ce cours, on utilise des tuiles de 32x32 car on peut facilement trouver des jeux de tuiles GB ***tilesets*** gratuit dans ce format.

Pour trouver la position exacte d'un objet sur l'écran, on fait un petit calcul.

- Si tu connais la **colonne**, tu multiplies ce numéro de colonne par la taille d'une tuile (32 pixels) pour obtenir la position **x** (c'est la position horizontale, combien de pixels depuis la gauche de l'écran).
- Si tu connais la **ligne**, tu multiplies ce numéro de ligne par la taille d'une tuile pour obtenir la position **y** (c'est la position verticale, combien de pixels depuis le haut de l'écran).

Exemple concret

Si tu veux dessiner un mur dans la case qui est à la ligne 3 et à la colonne 4, tu fais :

- Pour **x** : colonne 4 × 32 pixels = 128 pixels → donc, tu commences à 128 pixels depuis la gauche.
- Pour **y** : ligne 3 × 32 pixels = 96 pixels → donc, tu commences à 96 pixels depuis le haut.

Cela permet à l'ordinateur de savoir exactement où placer l'objet, parce que tu lui as donné la position exacte en **pixels**.

Le procédé que nous venons de décrire s'appelle la **transformation de coordonnées** ou **conversion de coordonnées**. Dans le contexte des jeux vidéo et de la programmation graphique, il s'agit de convertir les

positions en **lignes** et **colonnes** de la grille (qui représentent la position logique ou abstraite dans le jeu) en positions en **pixels** sur l'écran (qui sont les positions physiques ou concrètes où les éléments doivent être dessinés).

Cette transformation permet au programme de savoir où afficher chaque élément du jeu sur l'écran. En multipliant le numéro de la ligne et de la colonne par la taille d'une tuile (par exemple, 32 pixels), on obtient les coordonnées exactes en pixels (**x** et **y**) où l'objet doit être dessiné. Cela permet de faire correspondre la structure logique du jeu (la grille) avec la représentation visuelle sur l'écran (les pixels).

© 2025 Mr. Lambrechts