Labyrinthe v0.1 - Les tuiles

Dans cet article, nous allons découvrir comment créer un labyrinthe en 2D avec Python et PyGame, en remplaçant les simples blocs de couleurs par des images pour un rendu visuellement plus attrayant. Nous verrons comment utiliser des images de tuiles (murs et sol) pour rendre l'affichage du labyrinthe plus immersif et interactif.

Utilisation des tuiles

Dans le développement de jeux vidéo, une **tuile** (GB ***tile***) est un **élément graphique** de petite taille, généralement carré, utilisé pour construire des environnements de jeu, comme des terrains, des labyrinthes ou des cartes. Les tuiles sont **assemblées dans une grille** (souvent appelée "**tilemap**") pour former des niveaux complets, des décors ou des objets. **Chaque tuile représente une unité d'espace dans le jeu**, que ce soit un morceau de sol, un mur, de l'eau, etc.

Utilité des tuiles

L'utilisation de tuiles permet de **faciliter la création et la gestion d'environnements complexes**. Voici quelques avantages et utilisations concrètes des tuiles dans un jeu :

1. **Modularité** : Les tuiles permettent de construire des niveaux ou des environnements à partir d'éléments réutilisables. Par exemple, une même tuile de "mur" peut être utilisée plusieurs fois dans différents endroits du jeu, ce qui simplifie la conception.

2. **Efficacité mémoire** : Au lieu de stocker une grande image représentant tout le décor d'un niveau, il suffit de stocker un petit ensemble de tuiles et de les assembler dynamiquement. Cela réduit l'utilisation de la mémoire, car on n'a pas besoin de créer une grande image unique pour chaque scène.

3. **Gestion des collisions et de la logique** : Les tuiles ne sont pas seulement visuelles. Elles sont souvent liées à des règles de jeu. Par exemple, une tuile de "mur" sera marquée comme un obstacle dans le jeu, empêchant le personnage de passer à travers. Les tuiles permettent ainsi de gérer facilement les **collisions**, les **zones de danger** (comme des piques ou de la lave), et d'autres comportements.

4. **Facilité d'édition** : Les tuiles permettent de créer des éditeurs de niveaux. En assemblant des tuiles sur une grille, les concepteurs de jeux ou même les joueurs peuvent facilement créer et modifier des niveaux.

En résumé, les tuiles sont un moyen pratique et efficace de construire des environnements de jeux vidéo en 2D, permettant non seulement de simplifier l'aspect visuel, mais aussi de gérer des comportements spécifiques comme les collisions et les interactions avec les éléments du décor.

Implémentation

Pour afficher des images à la place des blocs de couleurs dans ton labyrinthe, tu devras charger les images correspondantes aux différentes tuiles (par exemple une image pour le sol et une pour les murs) et les dessiner dans la fenêtre à la place des blocs de couleurs. Voici les étapes pour adapter ton code :

1. **Charger les images** : Utilise la fonction `pygame.image.load()` pour charger les images des murs et du sol.
2. **Afficher les images** : Utilise la fonction `blit()` pour dessiner les images dans la fenêtre à la place des `pygame.draw.rect()`.

Modifications à apporter

1. Adapte le code pour que la taille de la fenêtre de jeu s'adapte automatiquement à la taille du labyrinthe.

2. **Prépare les images** :

    - Crée ou télécharge deux images : une pour le mur et une pour le sol (par exemple, `mur.png` et `sol.png`).
    - Assure-toi que les images ont la même taille que `TAILLE_TUILE`, c'est-à-dire 32x32 pixels.
3. **Charge les images dans le code** :

- Utilise `pygame.image.load()` pour charger les images dans le programme.

4. **Remplace les rectangles colorés par les images** :
   - À la place de `pygame.draw.rect()`, utilise `fenetre.blit()` pour dessiner l'image du mur ou du sol à l'emplacement approprié.

Exemple adapté :

```python
import pygame

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

TAILLE_TUILE = 32

pygame.init()
largeur, hauteur = len(labyrinthe[0]) * TAILLE_TUILE, len(labyrinthe) * TAILLE_TUILE
fenetre = pygame.display.set_mode((largeur, hauteur))

# Charger les images pour le mur et le sol
image_mur = pygame.image.load("mur.png")
image_sol = pygame.image.load("sol.png")

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

    x = 0
    y = 0
    for ligne in labyrinthe:
        for col in ligne:
            # Choisir l'image à afficher en fonction du contenu de la cellule
            if col == 1:
                fenetre.blit(image_mur, (x, y))
            else:
                fenetre.blit(image_sol, (x, y))

            x += TAILLE_TUILE

        x = 0
        y += TAILLE_TUILE

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

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

Explications des modifications :

1. **Chargement des images** :

```
image_mur = pygame.image.load("mur.png")
image_sol = pygame.image.load("sol.png")
```

   Cela charge les images que tu as créées ou téléchargées pour représenter les murs et le sol.

2. **Remplacement du dessin des rectangles** : Au lieu d'utiliser `pygame.draw.rect()`, on utilise :

```
fenetre.blit(image_mur, (x, y))  # Pour les murs
fenetre.blit(image_sol, (x, y))  # Pour le sol
```

   Cette fonction affiche les images à la position `(x, y)` correspondant à chaque case du labyrinthe.

3. **Redimensionnement de la fenêtre** : La taille de la fenêtre est automatiquement calculée en fonction des dimensions du labyrinthe pour s'ajuster à la taille des tuiles :

```
largeur, hauteur = len(labyrinthe[0]) * TAILLE_TUILE, len(labyrinthe) * TAILLE_TUILE
```

Remarques :

- **Assure-toi que les images** (`mur.png` et `sol.png`) sont de la même taille que `TAILLE_TUILE` (32x32 pixels).
- Si tes images sont de taille différente, tu peux les redimensionner avec `pygame.transform.scale()` pour qu'elles s'adaptent à la taille des tuiles :

```
image_mur = pygame.transform.scale(pygame.image.load("mur.png"), (TAILLE_TUILE, TAILLE_TU
image_sol = pygame.transform.scale(pygame.image.load("sol.png"), (TAILLE_TUILE, TAILLE_TU
```