2. Exemple pratique 1 : Approche inspirée des jeux vidéo (40 min)

Objectif : Utiliser la récursivité pour résoudre un labyrinthe en jeu vidéo

Dans ce projet, nous allons résoudre un problème classique inspiré des jeux vidéo : aider un héros à s'échapper d'un labyrinthe. Le labyrinthe est modélisé comme une grille 2D où certaines cases sont des murs et d'autres représentent des chemins libres. Le but est de guider notre héros vers la sortie en utilisant la récursivité pour explorer toutes les directions possibles.

Contexte : Un héros piégé dans un labyrinthe

- **Murs** : Les murs sont des obstacles que le héros ne peut pas traverser.
- **Chemins libres** : Les cases sur lesquelles le héros peut marcher.
- **Position de départ (S)** : Le point où commence l'exploration.
- **Sortie (E)** : Le point que le héros doit atteindre.

Le héros peut se déplacer dans **quatre directions** : **haut**, **bas**, **gauche** et **droite**.

Étape 1 : Modélisation du labyrinthe

Le labyrinthe est représenté par une matrice 2D où chaque élément correspond à une case :

- `1` représente un **mur**.
- `0` représente un **chemin libre**.
- `S` représente la **position de départ**.
- `E` représente la **sortie**.

Voici un exemple simple d'un labyrinthe en Python :

```
labyrinthe = [
    ['S', 1, 0, 0, 1],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 'E'],
    [1, 1, 0, 0, 1],
    [0, 0, 0, 1, 0]
]
```

Visualisation de ce labyrinthe :

```
S  1  0  0  1
0  1  0  1  0
0  0  0  1  E
1  1  0  0  1
0  0  0  1  0
```

Le héros commence en `S` (position `(0, 0)`) et doit atteindre `E` (position `(2, 4)`).

Étape 2 : Fonction récursive pour explorer le labyrinthe

L'idée est de créer une fonction qui explore toutes les directions possibles à partir de chaque position. La fonction va :

1. **Vérifier les limites du labyrinthe** pour ne pas sortir des bords.

2. **Vérifier si la case est un mur** ou déjà visitée.
3. **Chercher la sortie** : si la position actuelle est la sortie, l'exploration s'arrête avec succès.
4. **Explorer récursivement** les quatre directions (haut, bas, gauche, droite).
5. **Marquer les cases visitées** pour éviter de repasser par les mêmes cases.

Voici la fonction récursive qui permet de résoudre le labyrinthe :

```
def explorer_labyrinthe(labyrinthe, x, y, visitees):
    # Vérifier les limites du labyrinthe
    if x < 0 or x >= len(labyrinthe) or y < 0 or y >= len(labyrinthe[0]):
        return False

    # Vérifier si la case est un mur ou déjà visitée
    if labyrinthe[x][y] == 1 or (x, y) in visitees:
        return False

    # Si on a trouvé la sortie
    if labyrinthe[x][y] == 'E':
        print(f"Sortie trouvée à la position ({x}, {y})")
        return True

    # Marquer la case comme visitée
    visitees.add((x, y))

    # Appeler la fonction récursive dans les 4 directions
    if (explorer_labyrinthe(labyrinthe, x+1, y, visitees) or  # Bas
        explorer_labyrinthe(labyrinthe, x-1, y, visitees) or  # Haut
        explorer_labyrinthe(labyrinthe, x, y+1, visitees) or  # Droite
        explorer_labyrinthe(labyrinthe, x, y-1, visitees)):   # Gauche
        return True

    # Si aucune direction ne fonctionne, on retourne False
    return False
```

Explication détaillée :

- **Vérification des limites** :

  ```
  if x < 0 or x >= len(labyrinthe) or y < 0 or y >= len(labyrinthe[0]):
      return False
  ```

  Cette condition permet de vérifier que le héros ne sort pas des bords du labyrinthe.

- **Vérification des murs ou des cases déjà visitées** :

  ```
  if labyrinthe[x][y] == 1 or (x, y) in visitees:
      return False
  ```

  Si le héros rencontre un mur (`1`) ou s'il revient sur une case déjà visitée, on arrête la récursion dans cette direction.

- **Vérification de la sortie** :

  ```
  if labyrinthe[x][y] == 'E':
      print(f"Sortie trouvée à la position ({x}, {y})")
      return True
  ```

  Si le héros atteint la sortie `E`, la fonction renvoie `True` et l'exploration s'arrête.

- **Exploration récursive** :

```
if (explorer_labyrinthe(labyrinthe, x+1, y, visitees) or  # Bas
    explorer_labyrinthe(labyrinthe, x-1, y, visitees) or  # Haut
    explorer_labyrinthe(labyrinthe, x, y+1, visitees) or  # Droite
    explorer_labyrinthe(labyrinthe, x, y-1, visitees)):   # Gauche
    return True
```

Le héros explore récursivement dans les quatre directions. Si l'une des directions mène à la sortie, la fonction renvoie `True` et stoppe l'exploration.

Étape 3 : Lancer la simulation

Pour démarrer l'exploration du labyrinthe à partir de la position de départ `(0, 0)`, nous devons appeler la fonction en fournissant les coordonnées de départ et un ensemble vide pour suivre les cases visitées :

```
# Position de départ (0, 0)
position_depart = (0, 0)

# Ensemble des cases visitées
cases_visitees = set()

# Lancer la recherche de la sortie
explorer_labyrinthe(labyrinthe, position_depart[0], position_depart[1],
cases_visitees)
```

Objectif atteint :

- La fonction explore récursivement toutes les directions à partir de la position de départ.
- Chaque appel récursif se déplace dans une nouvelle direction jusqu'à ce que la sortie soit trouvée.
- Si aucune direction ne fonctionne, la fonction retourne `False`, signifiant qu'il n'y a pas de chemin viable dans cette direction.

Illustration du déroulement de la récursivité :

Prenons l'exemple du héros démarrant à la position `(0, 0)` :

1. La fonction teste la direction **bas** `(1, 0)` (chemin libre).
2. Elle teste ensuite encore **bas** `(2, 0)` (chemin libre).
3. Puis **droite** `(2, 1)` (chemin libre), puis `(2, 2)` (chemin libre), jusqu'à atteindre la sortie `(2, 4)`.

Le héros atteint la sortie avec succès et le message `Sortie trouvée à la position (2, 4)` s'affiche.

Améliorations possibles :

- **Affichage du chemin parcouru** : Vous pouvez conserver un historique des déplacements pour afficher le chemin que le héros a emprunté.
- **Gestion des impasses** : Actuellement, le héros explore aveuglément toutes les directions. On peut ajouter de l'optimisation pour éviter de revenir sur des chemins déjà reconnus comme impasses.

Cet exemple montre comment utiliser la récursivité pour résoudre des problèmes de type **exploration** dans un jeu vidéo. C'est un bon modèle pour comprendre comment la récursion peut être utilisée pour explorer des espaces, comme les labyrinthes, dans le développement de jeux.