Année 2022/2023
Résolution de Problèmes
Master 1, MIV

Série de Travaux Pratiques N°1
Modélisation et résolution du jeu Sokoban

Le Sokoban (Pushbox) est un jeu de puzzle classique, originaire du Japon et datant du début des années 80. Dans ce jeu un gardien d'entrepôt (le joueur) se déplace dans un entrepôt, représenté par une grille de cases. Ces cases sont de cinq types : JOUEUR, MUR, CASE VIDE, CAISSE, CASE CIBLE (voir Figure 1). Le but de ce jeu est de ranger toutes les caisses dans les cases cibles.

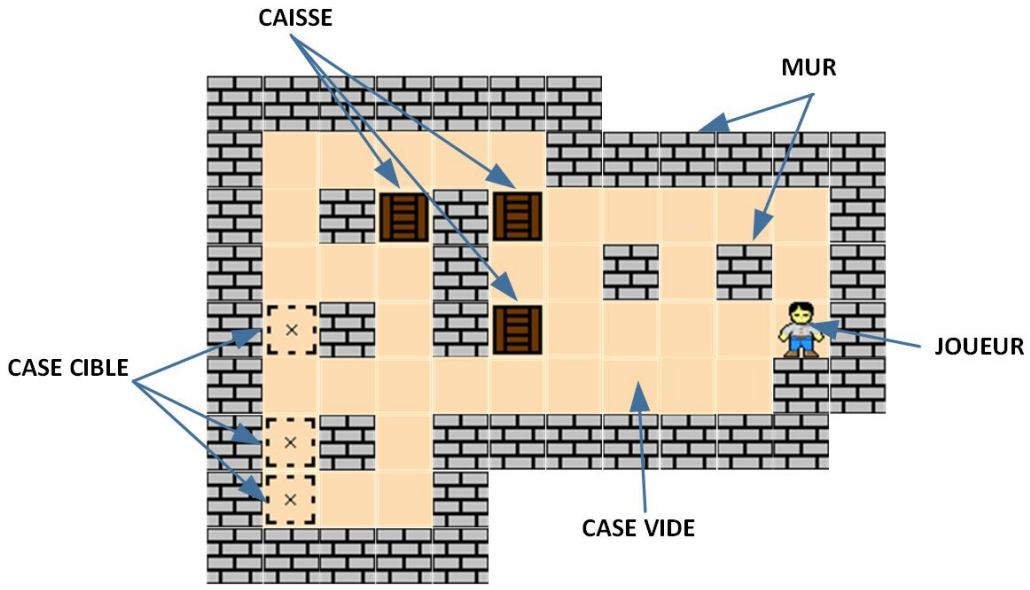

Figure 1 Exemple d'un niveau Sokoban

Les règles du jeu sont très simples :

- La structure de l'entrepôt et le nombre de caisses varient d'un niveau à un autre ;
- L'entrepôt dans lequel il faut ranger les caisses a rarement la forme d'un simple rectangle, mais c'est plutôt un labyrinthe, plus ou moins compliqué. Il y a des murs sur tous les bords de ce labyrinthe ;
- On peut avoir une ou plusieurs caisses, et il y a toujours autant de cases cibles que de caisses ;
- Le jeu contient qu'un seul joueur, qui peut se déplacer dans les quatre directions cardinales ;

- Le joueur a uniquement le droit de se déplacer vers une case inoccupée (et qui n'est pas un mur) ou de pousser une caisse et de se déplacer ainsi sur la case libérée. Il n'a pas le droit de tirer une caisse ni de passer par-dessus ;
- Pour pousser une caisse, la case adjacente de la caisse dans la direction de poussé doit être libre. Le joueur ne peut pas pousser deux caisses en même temps ;
- Le joueur gagne lorsque toutes les caisses sont sur les cases cibles, dans n'importe quel ordre. Il n'y a pas de caisses assignées à des points cibles particuliers.

L'objectif du projet est dans un premier temps de réaliser une modélisation du jeu, puis d'implémenter un solveur basé sur les différents algorithmes de recherche, et enfin de créer une GUI pour ce jeu.

TP 1 :

Proposer une modélisation du jeu Sokoban, et faire son implémentation en Python.

1. **Modélisation d'un état :** pour pouvoir modéliser un état, il faut suivre les instructions suivantes :

- Le jeu se présente sous forme d'une grille à deux dimensions. On doit pouvoir représenter les éléments suivants : les murs, les cases vides et les cases cibles qui sont des éléments statiques, le joueur et les caisses qui sont des éléments dynamiques. Dans cette grille, on peut avoir des cases avec les configurations suivantes (voir Figure 2):

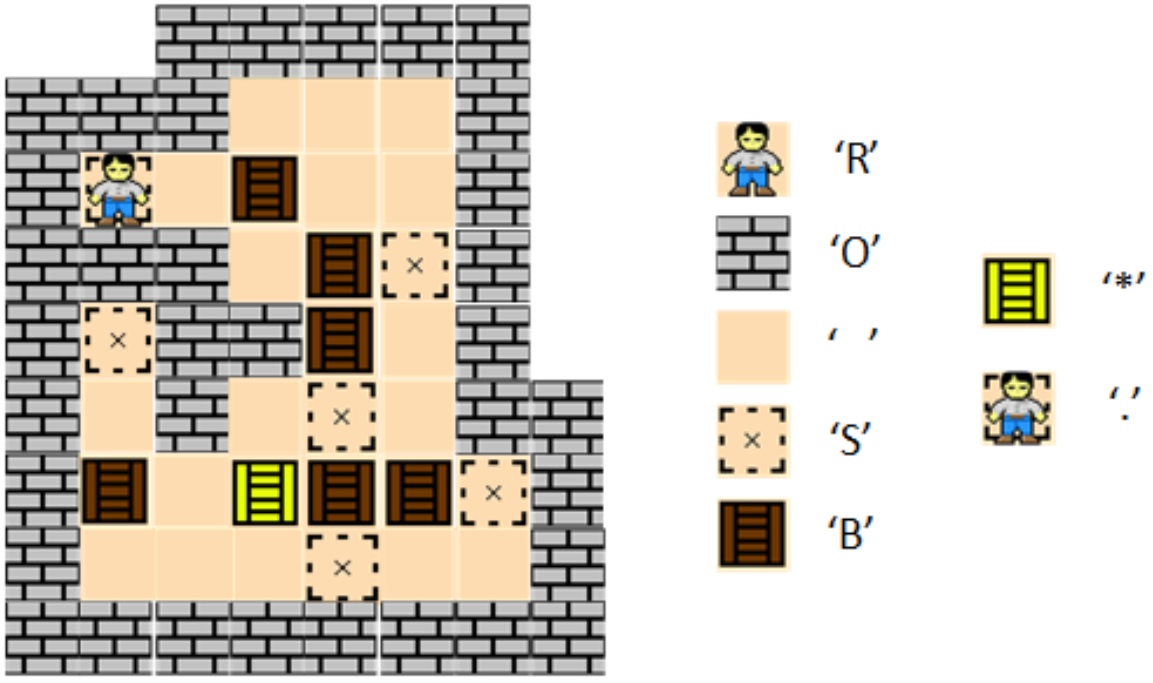

Figure 2 Configurations des cases Sokoban

- Joueur (Robot) : représenté par ‘R’
- Mur (Obstacle) : représenté par ‘O’
- Case vide : représenté par ‘ ’
- Case cible (Storage) : représenté par ‘S’
- Caisse (Block) : représenté par ‘B’
- Joueur sur une case cible : ‘.’
- Caisse sur une case cible : ‘*’

- Pour passer d’un état à un autre, le joueur doit se déplacer dans la grille. Quatre déplacements sont permis :

- UP : (-1, 0)
- DOWN : (1, 0)
- LEFT : (0, -1)
- RIGHT : (0, 1)

Afin qu’il soit valide, un déplacement doit respecter les règles du jeu (voir la description du jeu).

- L’état objectif est atteint lorsque toutes les caisses sont dans les cases cibles.

2. **Modélisation d’un nœud :** un nœud doit contenir les éléments suivants :

- Un état ;

- Un lien vers le nœud père ;

- L’action (le déplacement) qui a permis de générer le nœud courant à partir du nœud père ;

- Une fonction qui permet de générer les successeurs d’un nœud ;

- Une fonction qui permet de vérifier si le nœud courant est un nœud objectif, et de reconstruire la solution du nœud initial vers le nœud objectif, si c’est le cas.

TP 2 :

Proposer un solveur pour le jeu Sokoban basé sur la méthode de recherche « en largeur d’abord »

- En se basant sur le pseudo code donné dans le cours, faire l’implémentation de la méthode de recherche « en largeur d’abord », tout en l’adaptant à la problématique du jeu Sokoban.
- Effectuer des tests de cette méthode sur les exemples de la Figure 3. Donner la solution retournée pour chaque exemple, ainsi que le nombre d’itérations nécessaires pour atteindre cette solution.

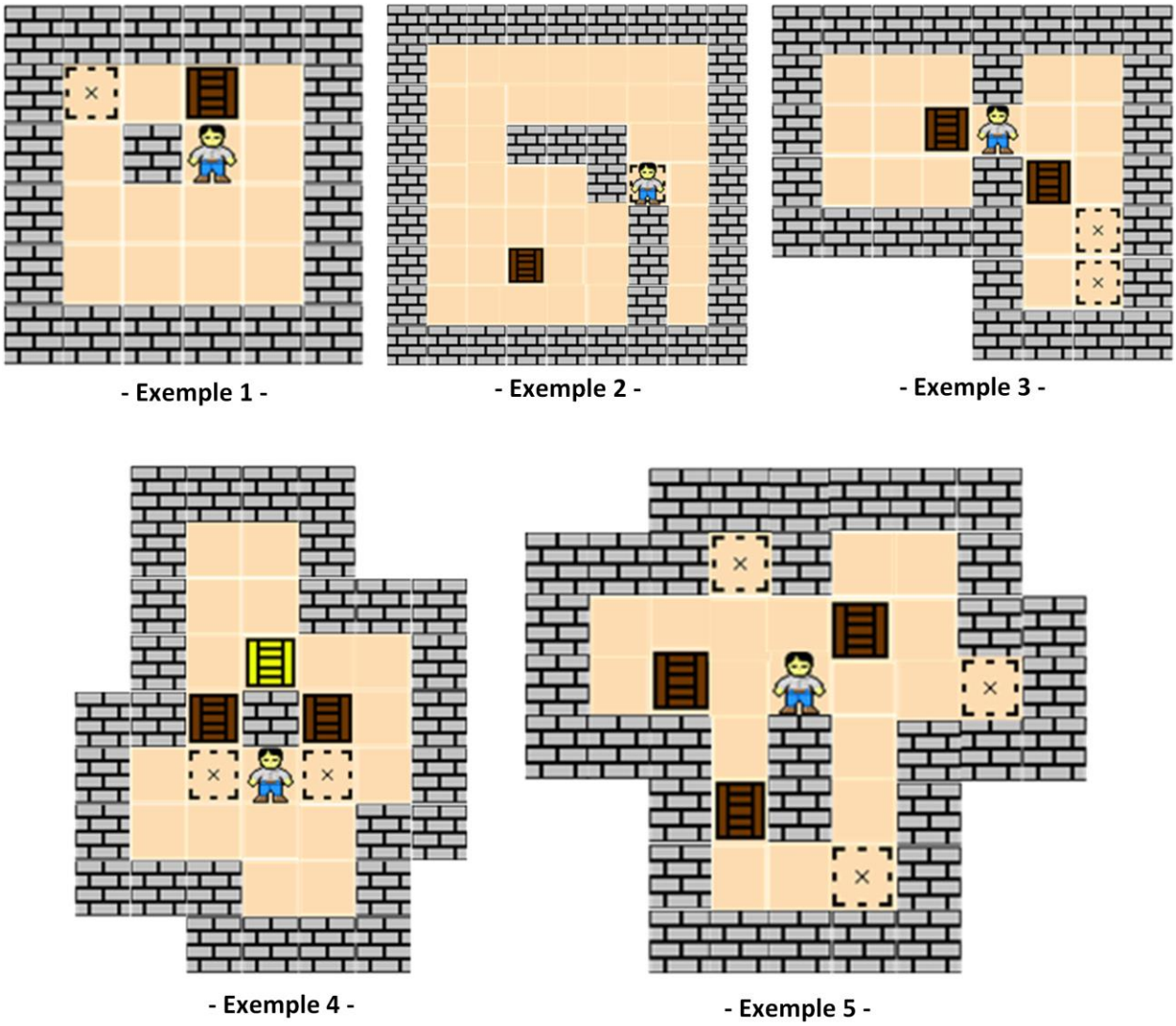

Figure 3 Exemples de test