Année 2022/2023
Résolution de Problèmes
Master 1, MIV

# Série de Travaux Pratiques N°1
# Modélisation et résolution du jeu Sokoban (Suite)

**TP 3 :**

Proposer un solveur pour le jeu Sokoban basé sur l'algorithme de recherche de type A. Dans cet algorithme, le choix du nœud à développer est guidé par une fonction d'évaluation $f$ définie par l'équation suivante (Equation 1):

$$f(n) = g(n) + h(n) \quad (1)$$

Où $g(n)$ représente le coût du chemin parcouru du nœud initial jusqu'à le nœud courant $n$, et $h(n)$ est une estimation du coût du chemin qui reste à parcourir du nœud courant $n$ vers le nœud objectif. Pour le jeu Sokoban :

- Le coût $g(n)$ est égal à 1 pour chaque déplacement du joueur.
- Le coût $h(n)$ est estimé à l'aide d'une heuristique.

1. En se basant sur le pseudo code donné dans le cours, faire l'implémentation de l'algorithme de recherche de type A, en utilisant les trois heuristiques suivantes :

- **Heuristique 1 :** Le nombre de caisses que le joueur n'a pas encore placées dans les cases cibles (Equation 2) :

$$h1(n) = nb\_left\_blocks(n) \quad (2)$$

- **Heuristique 2 :** Dans cette heuristique, on va rajouter à l'heuristique $h1$, une estimation du nombre de poussées requises pour déplacer chacune des caisses vers la cible la plus proche.

Pour estimer le nombre de poussées nécessaires pour déplacer une caisse $b$ vers une cible $s$, on va calculer la distance de Manhattan entre la caisse $b$ et la cible $s$. Cette distance est donnée par l'équation suivante (Equation 3) :

$$Manhattan\_distance_{b,s} = |x_b - x_s| + |y_b - y_s| \quad (3)$$

Où $(x_b, y_b)$ représente la position de la caisse $b$ et $(x_s, y_s)$ représente la position de la case cible $s$. La distance de Manhattan entre une caisse et une cible est illustrée dans la Figure 1 (avec les deux lignes en rouge).

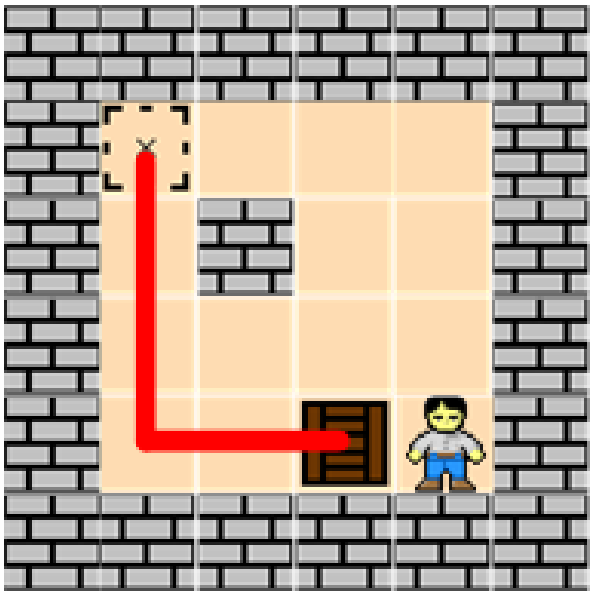

*Figure 1 Illustration de la distance de Manhattan entre une caisse et une cible*

D'où, l'heuristique 2 est donnée par l'équation suivante (Equation 4) :

$$h2(n) = 2 * nb\_left\_blocks(n) + \sum_{b \in B} \min_{s \in S} Manhattan\_distance_{b,s}(n) \quad (4)$$

Ici, on a multiplié le nombre de caisses que le joueur n'a pas encore placées dans les cases cibles par 2, pour lui donner plus de poids dans l'heuristique.

- **Heuristique 3 :** On peut remarquer que dans l'heuristique $h2$, on n'a pas pris en considération le nombre de déplacements que le joueur doit effectuer pour atteindre une caisse, afin qu'il puisse par la suite la pousser vers une cible. D'où la nécessité de rajouter à l'heuristique précédente $h2$, une estimation du nombre de déplacements requis par le joueur pour atteindre la caisse la plus proche. Ceci peut aussi être fait à l'aide de la distance de Manhattan entre le joueur R et une caisse b.
  Donc, l'heuristique 3 est donnée par l'équation suivante (Equation 5) :

$$\begin{aligned} h3(n) = 2 * nb\_left\_blocks(n) \\ + \sum_{b \in B} \min_{s \in S} Manhattan\_distance_{b,s}(n) \\ + \min_{b \in B} Manhattan\_distance_{R,b}(n) \end{aligned} \quad (5)$$

2. Effectuer des tests de l'algorithme de recherche de type A sur les exemples de la Figure 2. Donner la solution retournée pour chaque exemple, ainsi que le nombre d'itérations nécessaires pour atteindre cette solution. Comparer les résultats obtenus avec ceux de l'algorithme de recherche « en largeur d'abord ».

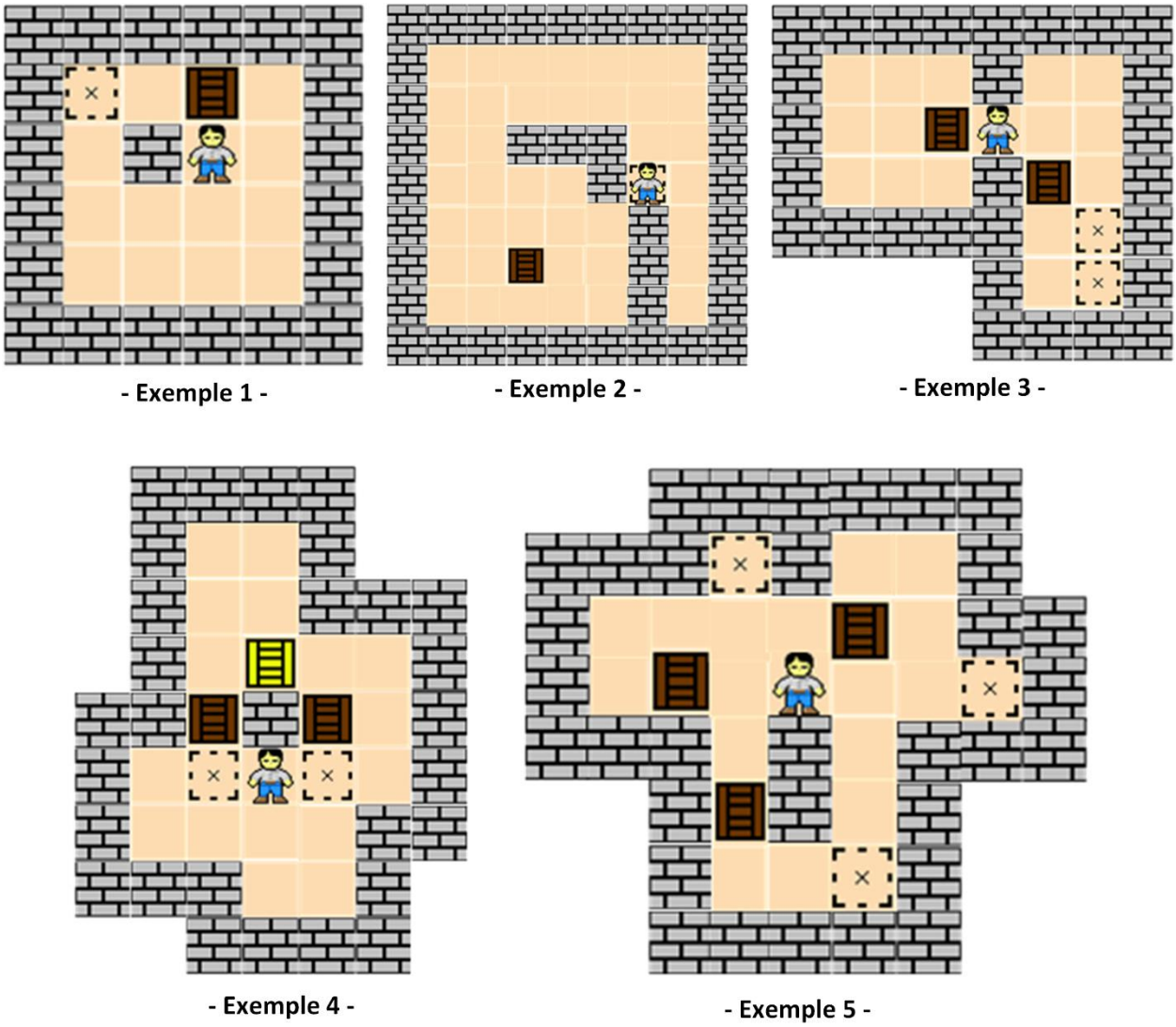


*Figure 2 Exemples de test*

**TP 4 :**

Sokoban est un jeu dans lequel on peut être confrontés à ce que nous appelons des deadlocks. Un deadlock est un état de l'arbre de recherche à partir duquel aucun état solution ne pourra être trouvé. De manière générale, un deadlock est provoqué par un sous-ensemble de caisses de l'état qui provoque une situation dans laquelle il est impossible de pousser toutes les caisses sur des cibles.

Si nous parcourons l'arbre de recherche en ignorant ces deadlocks, il est fort probable que nous obtenions un nombre très conséquent de sous-arbres condamnés à ne jamais aboutir à une solution. La taille de l'arbre va alors augmenter de manière importante. C'est pourquoi la détection des deadlocks est une étape indispensable dans le jeu Sokoban.

- **Deadlock en coin :** un deadlock en coin est provoqué par une caisse coincée sur une case qui n'est pas une cible et qui est cernée par deux murs de manière à former un coin. Des exemples d'un deadlock en coin sont représentés dans la Figure 3.

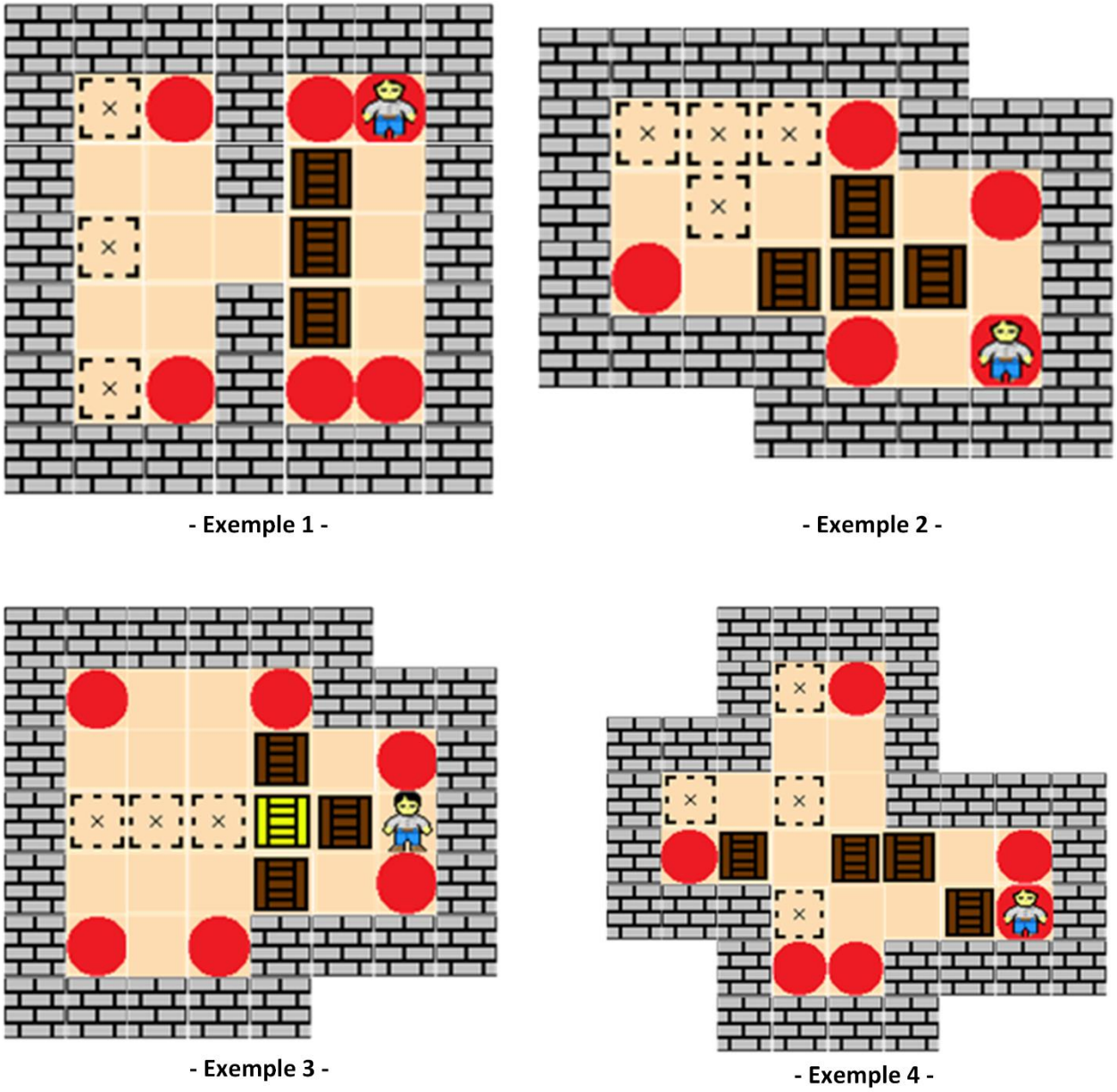


*Figure 3 Les positions en rouge provoquent un deadlock en coin si une caisse s'y trouve*

- **Deadlock en ligne :** un deadlock en ligne est provoqué par une caisse qui se situe contre un mur et qui est incapable de s'en dégager pour se diriger vers une cible. Des exemples d'un deadlock en ligne sont représentés dans la Figure 4. Les deadlocks en ligne sont provoqués par des murs qui ne possèdent pas de possibilité d'échappement, c'est-à-dire un passage par lequel le joueur pourrait repousser la caisse dans l'autre sens. La méthode utilisée pour détecter les lignes problématiques consiste à utiliser les positions de deadlock en coin. En se servant de l'une de ces positions pour en joindre une deuxième via une ligne droite, si l'un des deux côtés de cette ligne droite est exclusivement formé de murs, alors nous pouvons considérer toutes les positions de cette ligne comme des deadlocks.

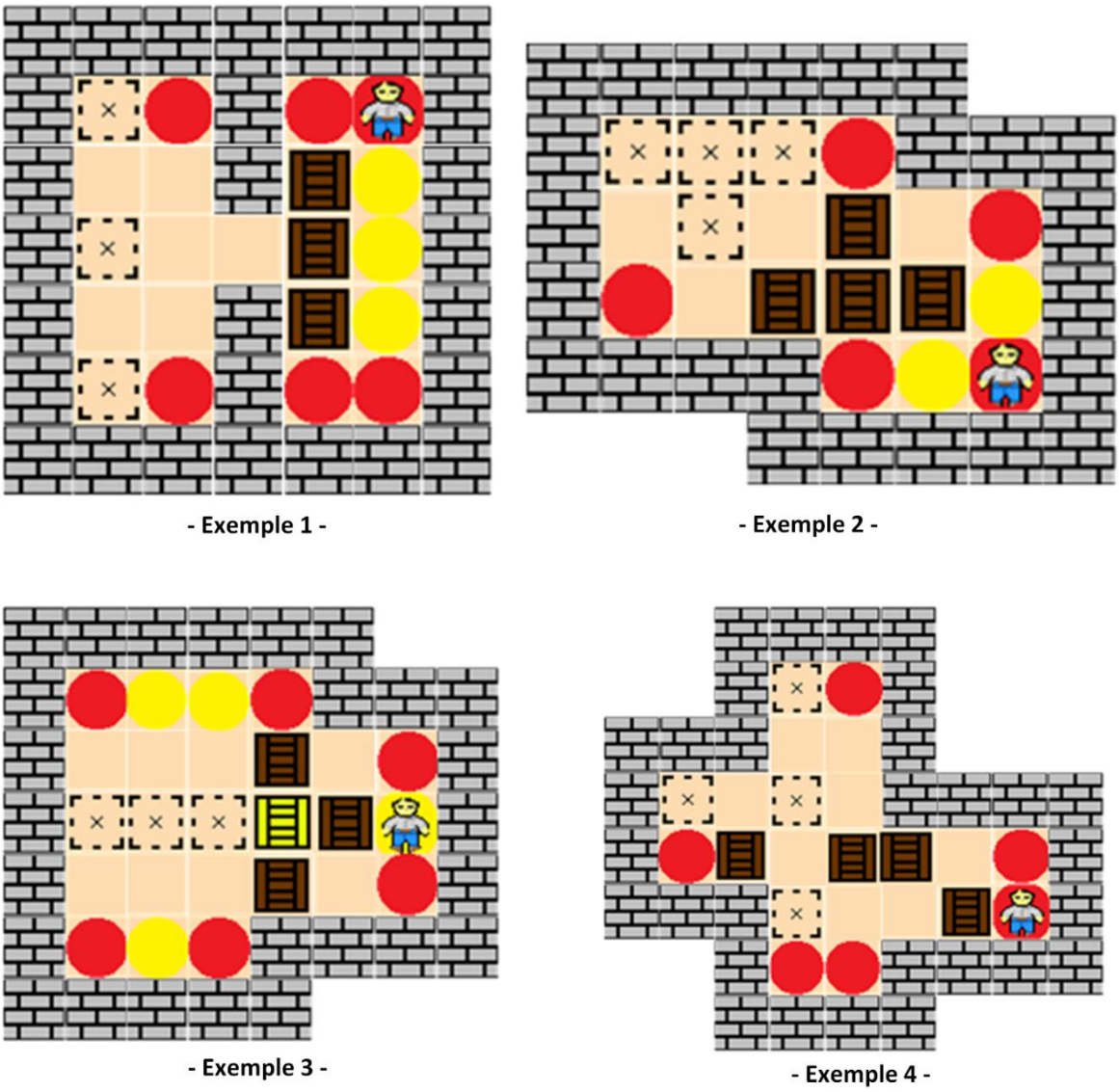


*Figure 4 Les positions en jaune provoquent un deadlock en ligne si une caisse s'y trouve*

1. Ecrire une fonction qui va déterminer pour un niveau Sokoban, toutes les positions qui peuvent provoquer un deadlock en coin ou un deadlock en ligne.
2. Ecrire une fonction qui va, pour un nœud donné, déterminer si son état contient des caisses sur des positions de deadlock (en coin ou en ligne).
3. Utiliser la fonction précédente pour améliorer l'algorithme de recherche A. Cette fonction va aider l'algorithme A à ne pas développer les sous-arbres condamnés à ne jamais aboutir à une solution.
4. Effectuer des tests de l'algorithme A amélioré sur les exemples de la Figure 5 en utilisant l'heuristique 3. Donner la solution retournée pour chaque exemple, ainsi que le nombre d'itérations nécessaires pour atteindre cette solution. Comparer les résultats obtenus avec les algorithmes de recherche précédents.

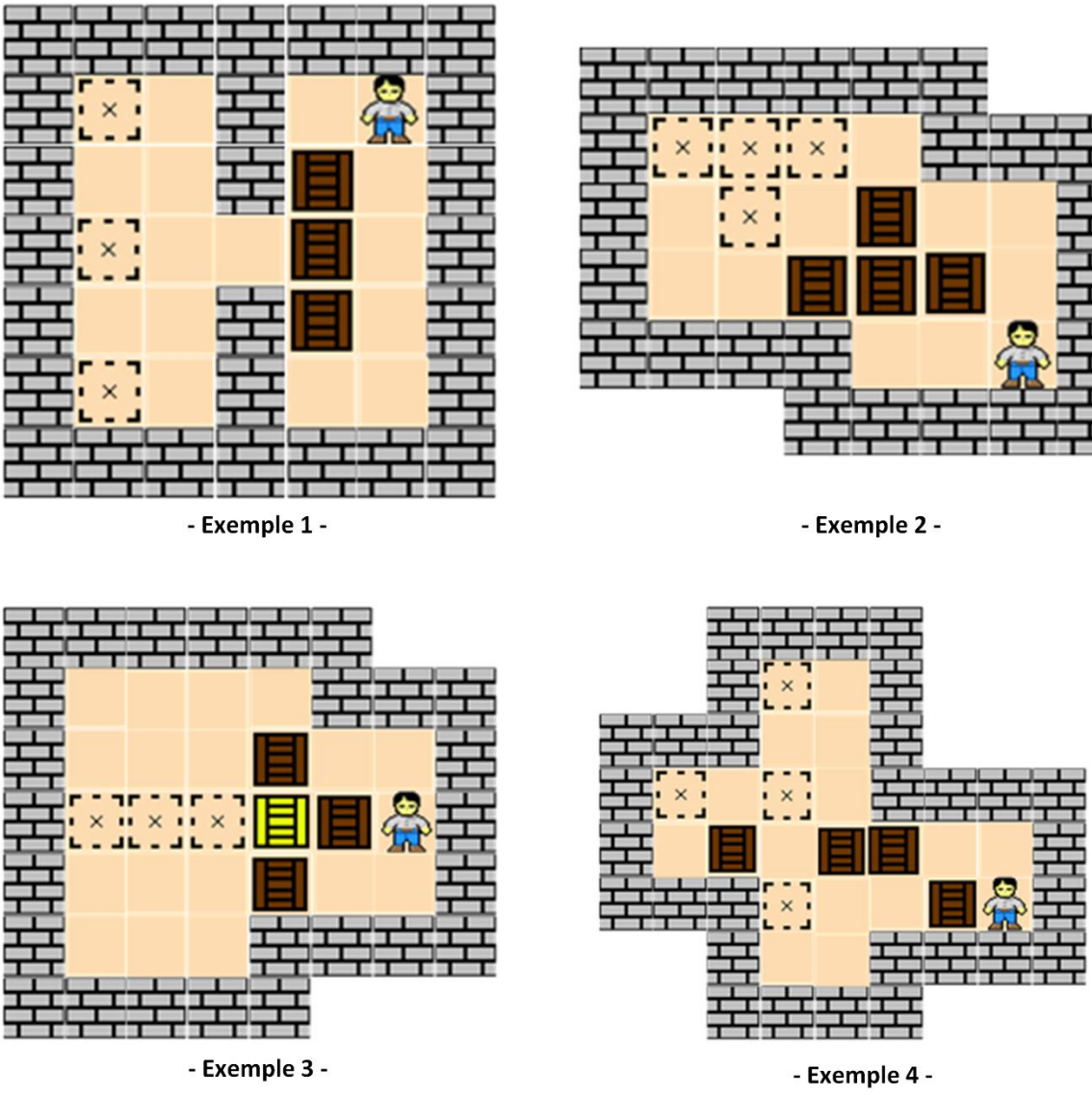


*Figure 5 Exemples de test*