Résolution de niveaux du Sokoban

PoulpoGaz, darth-mole

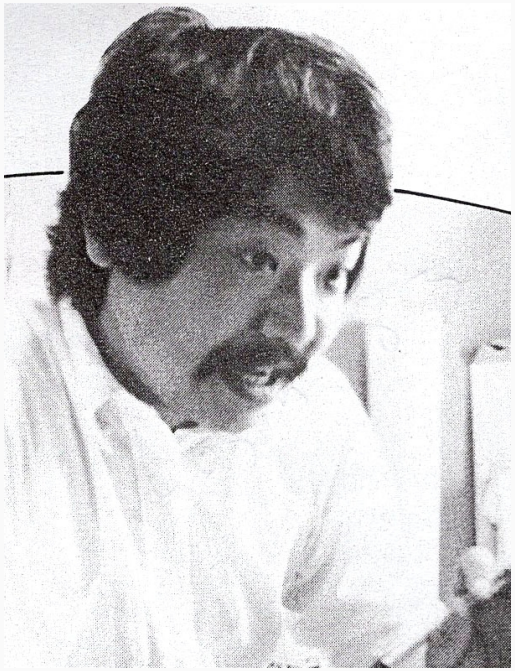
16 mai 2023

Candidat n° 012345

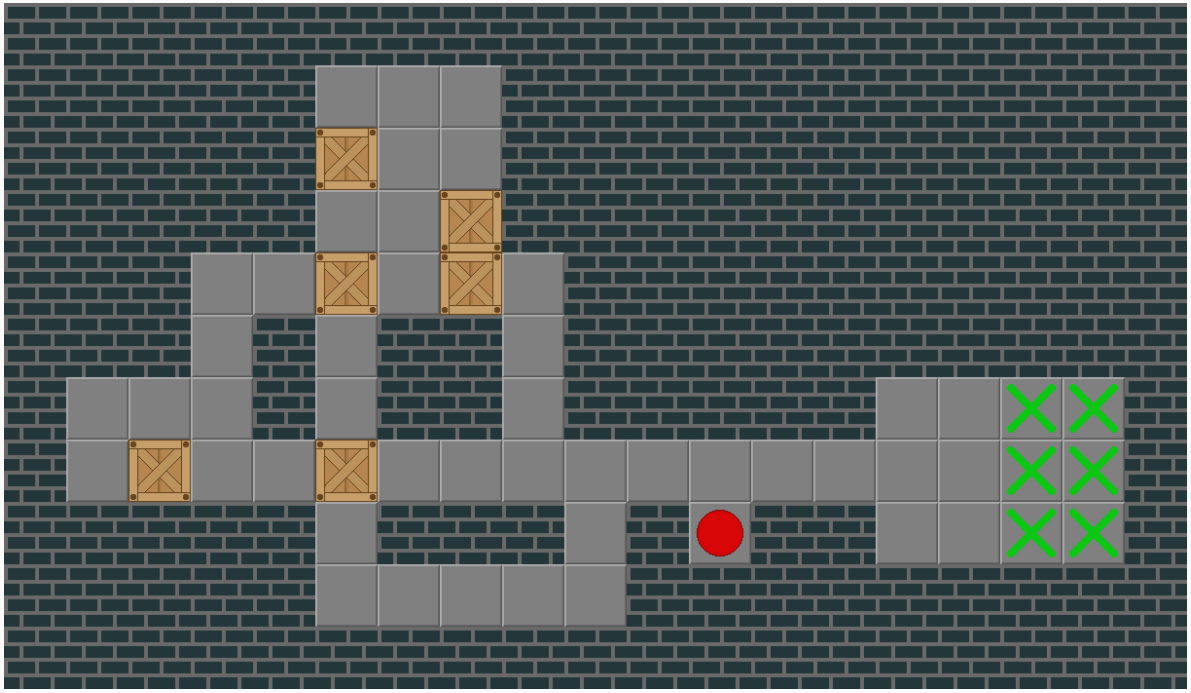
Plan

Le jeu du Sokoban

Le jeu du Sokoban

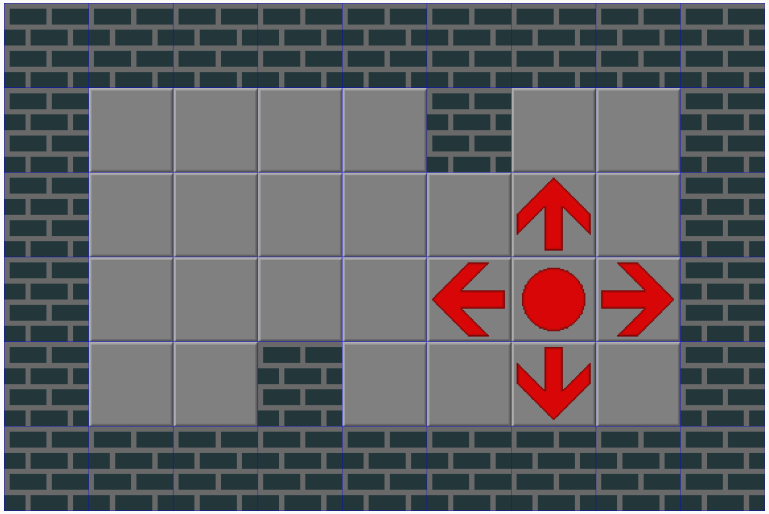


Hiroyuki Imabayashi



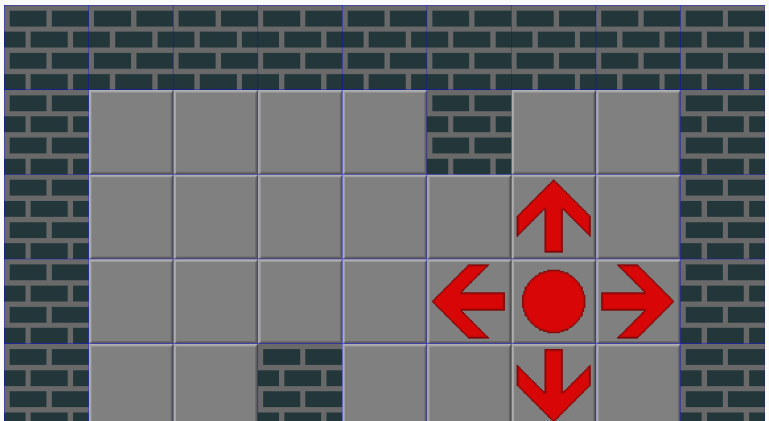
XSokoban

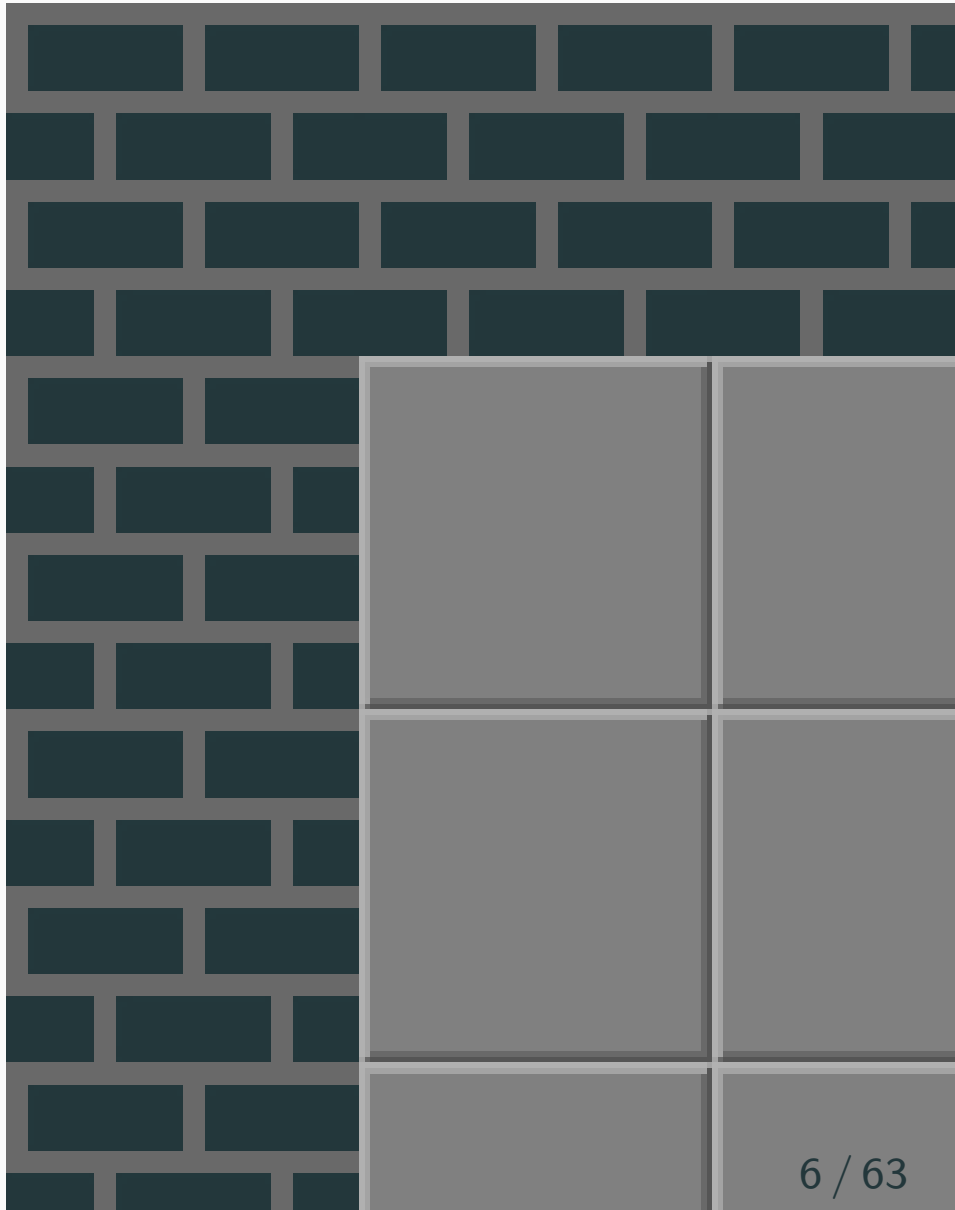
Règles



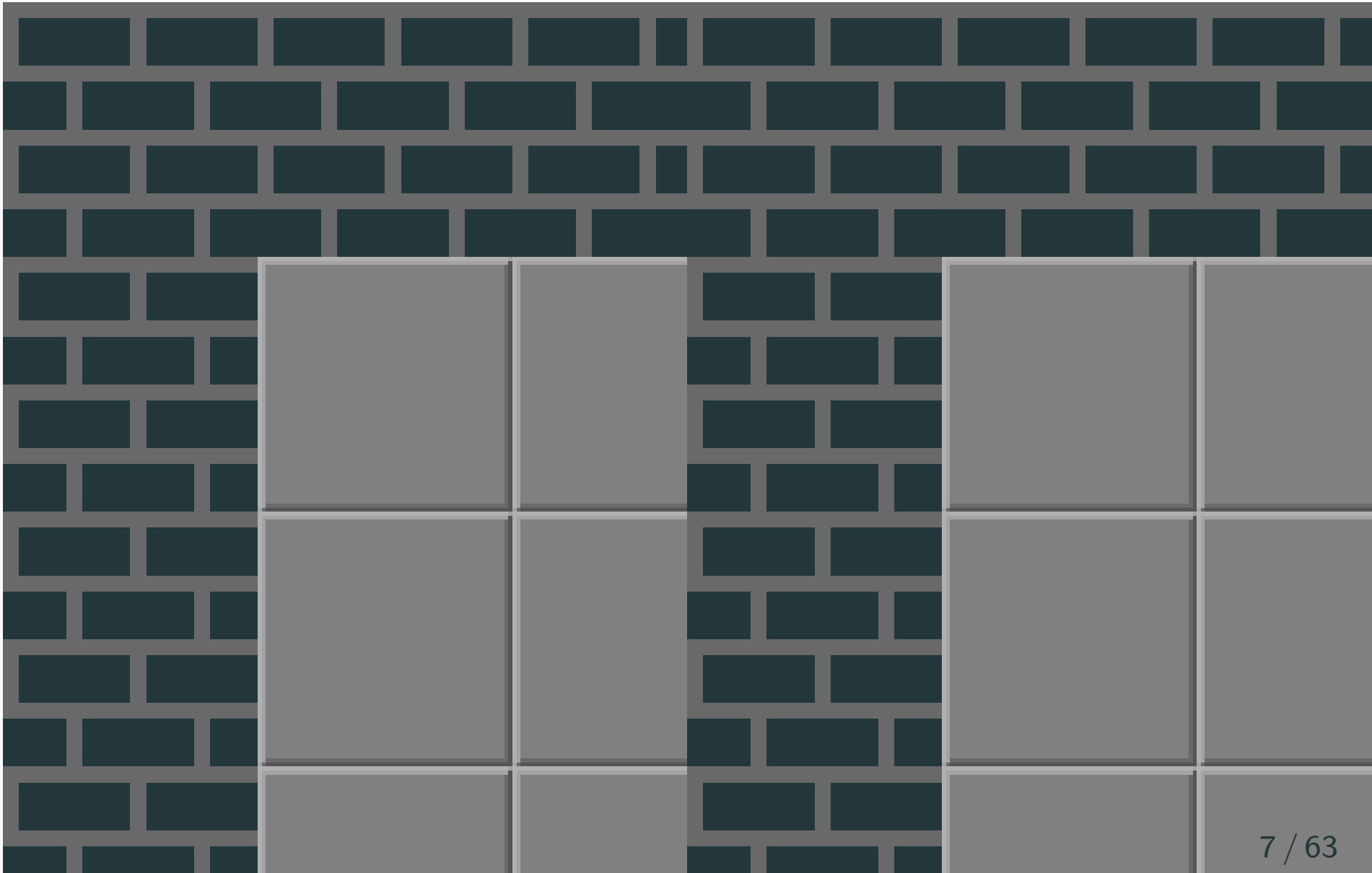
Déplacements autorisés

Règles

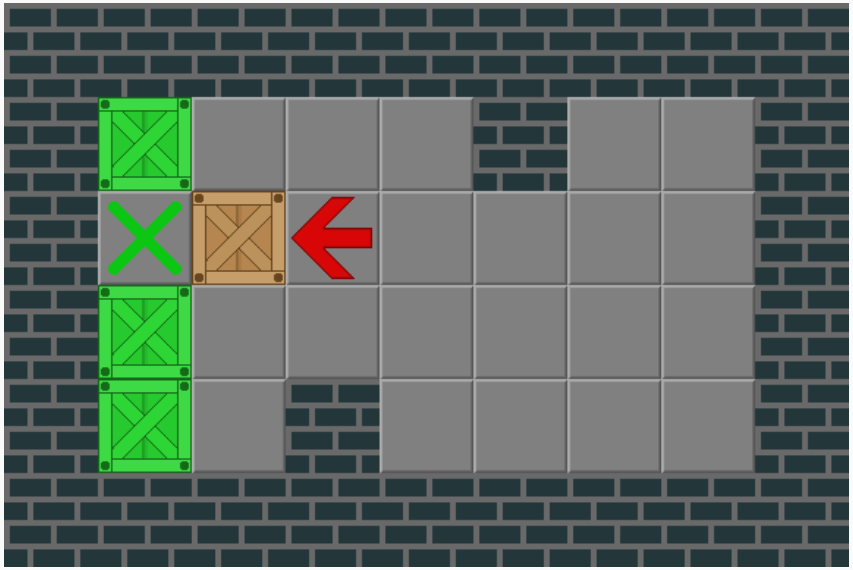


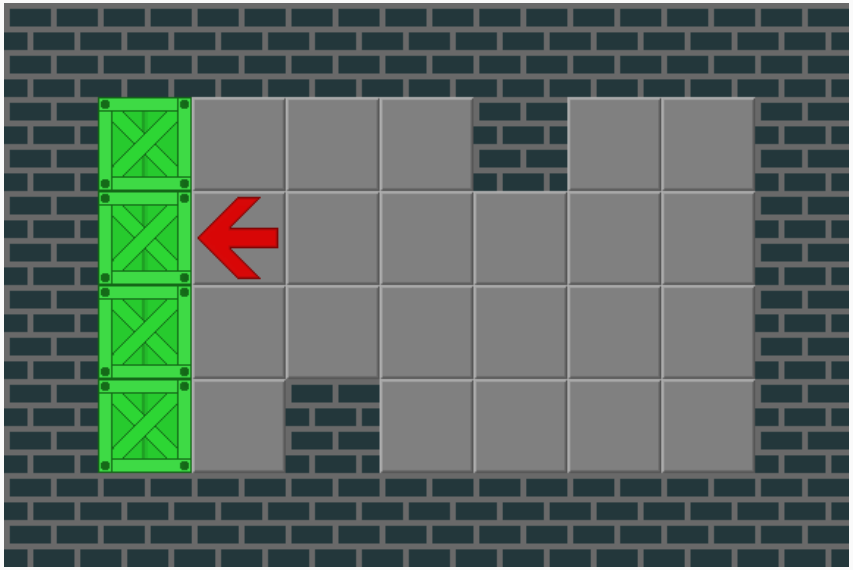


Règles



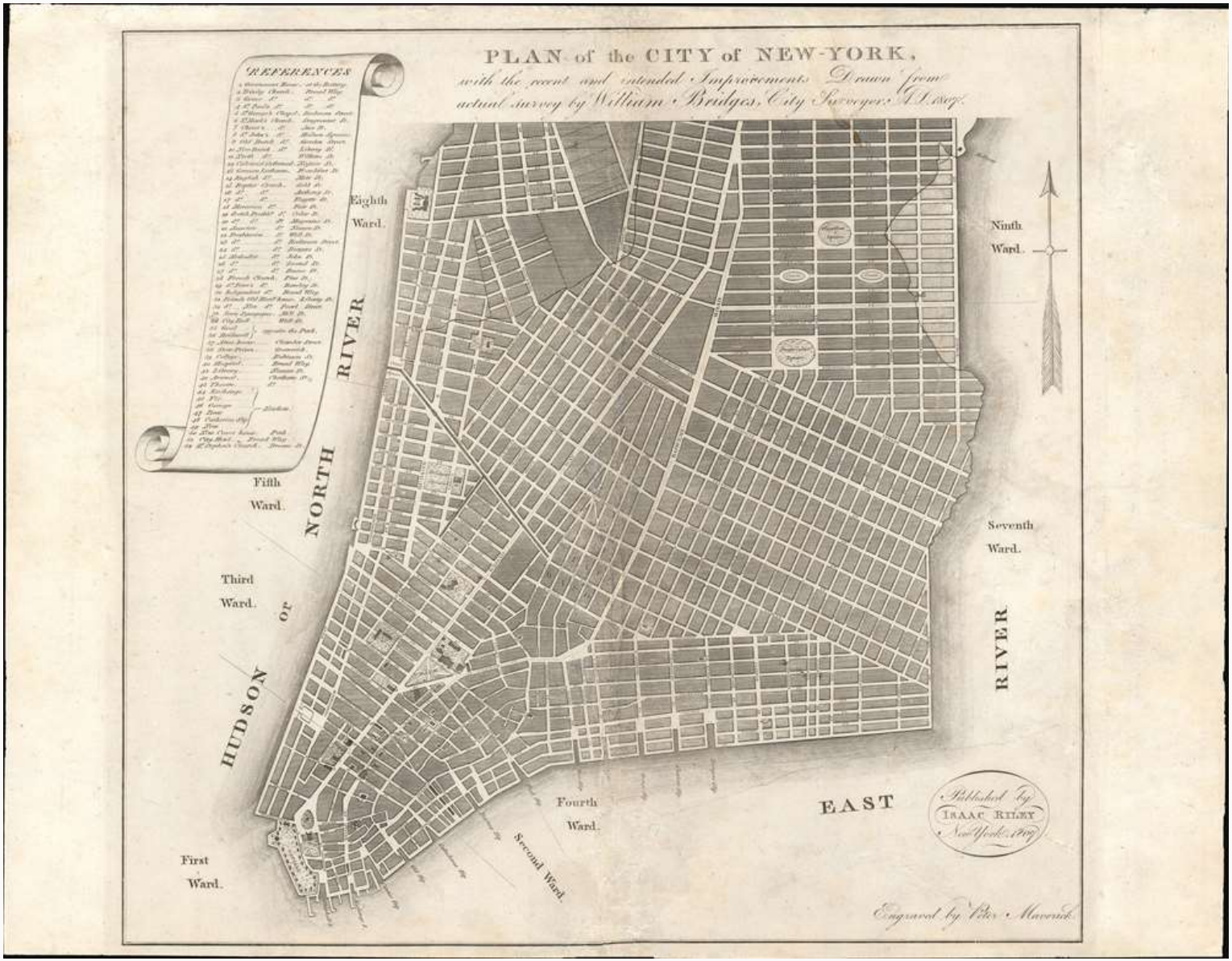
But du jeu





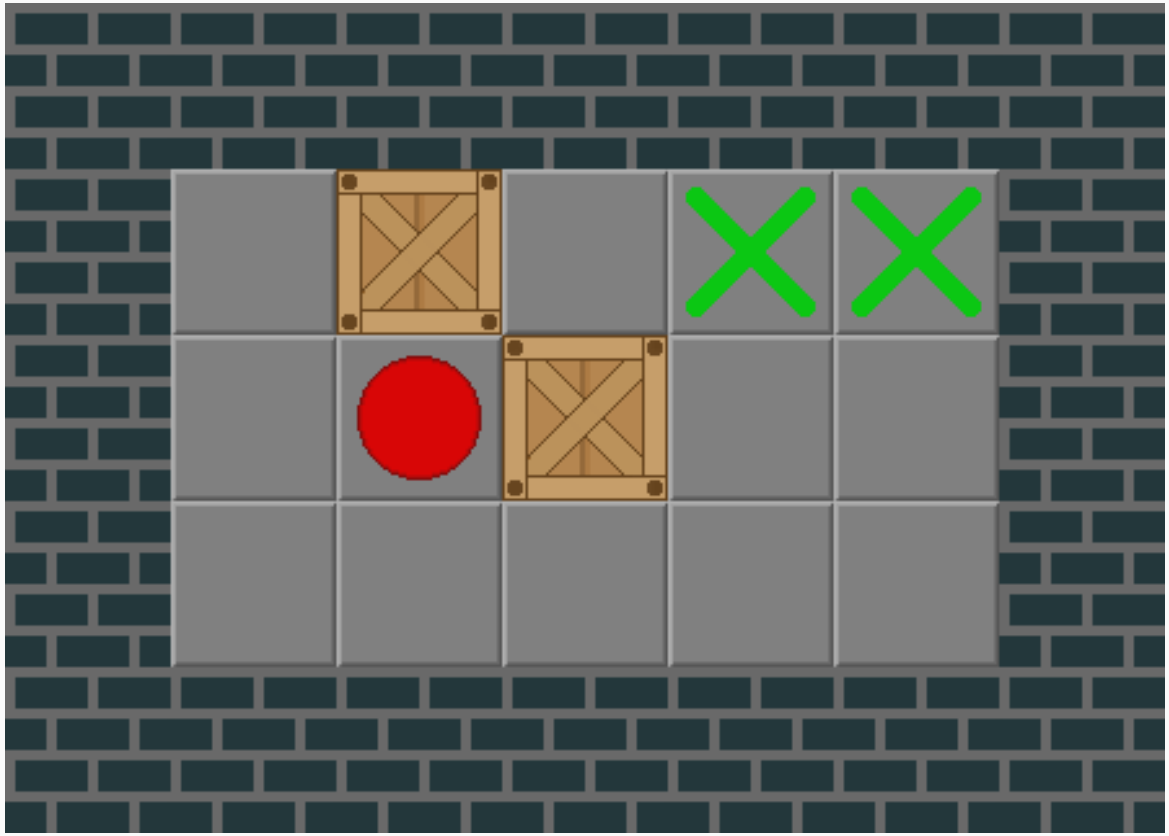
Lien avec le thème de l'année

Lien avec le thème de l'année

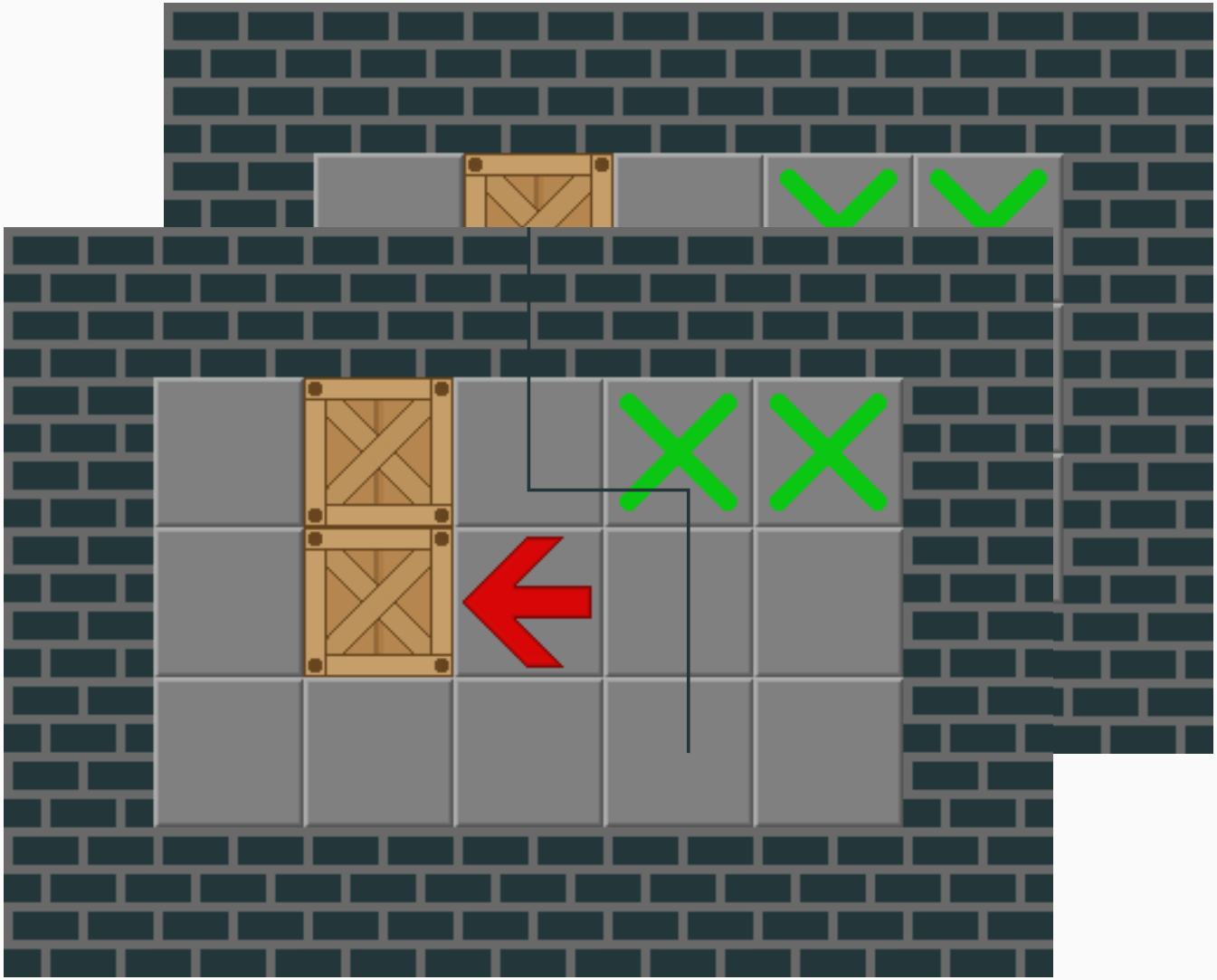


Principe de résolution

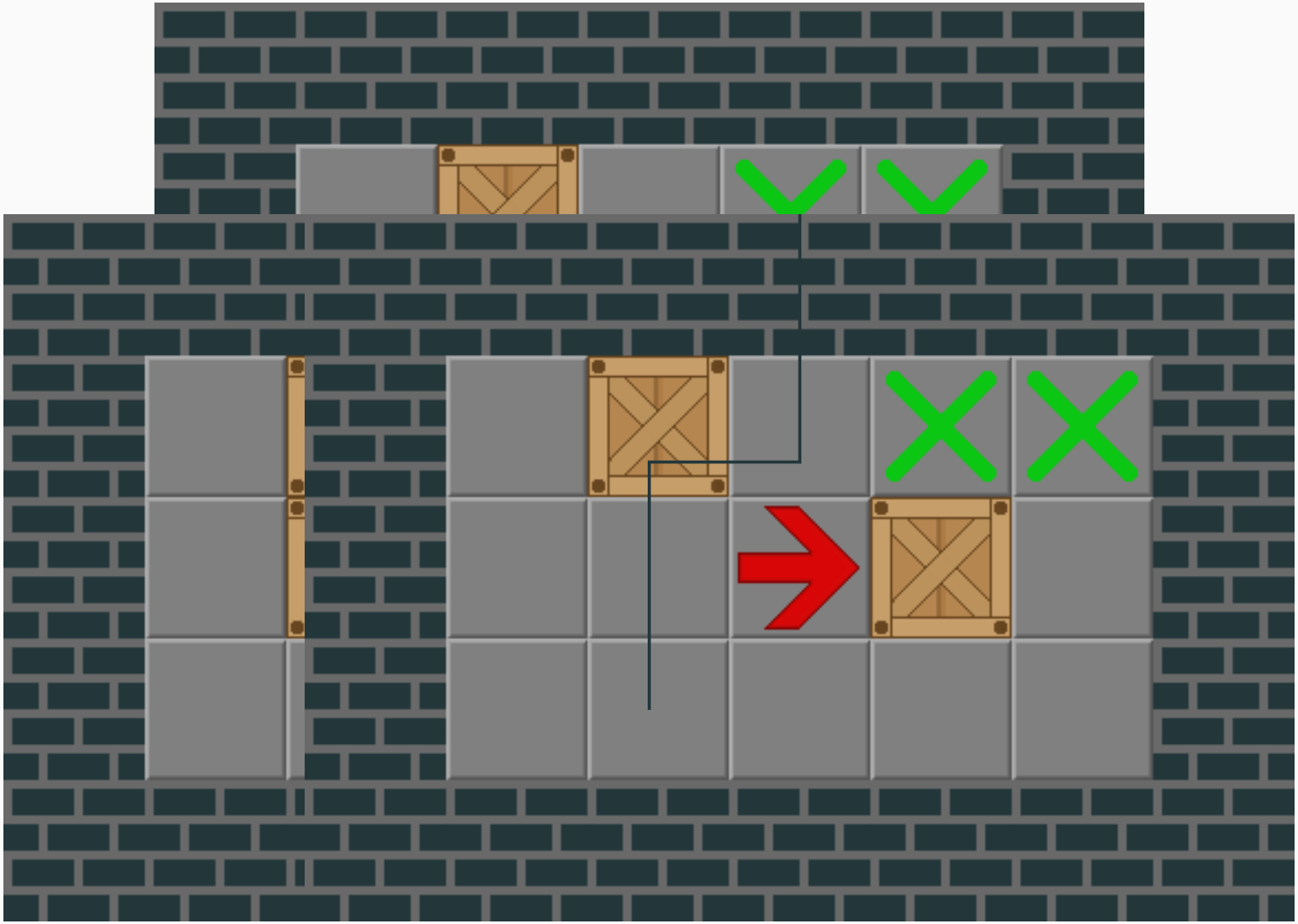
Arbre des états



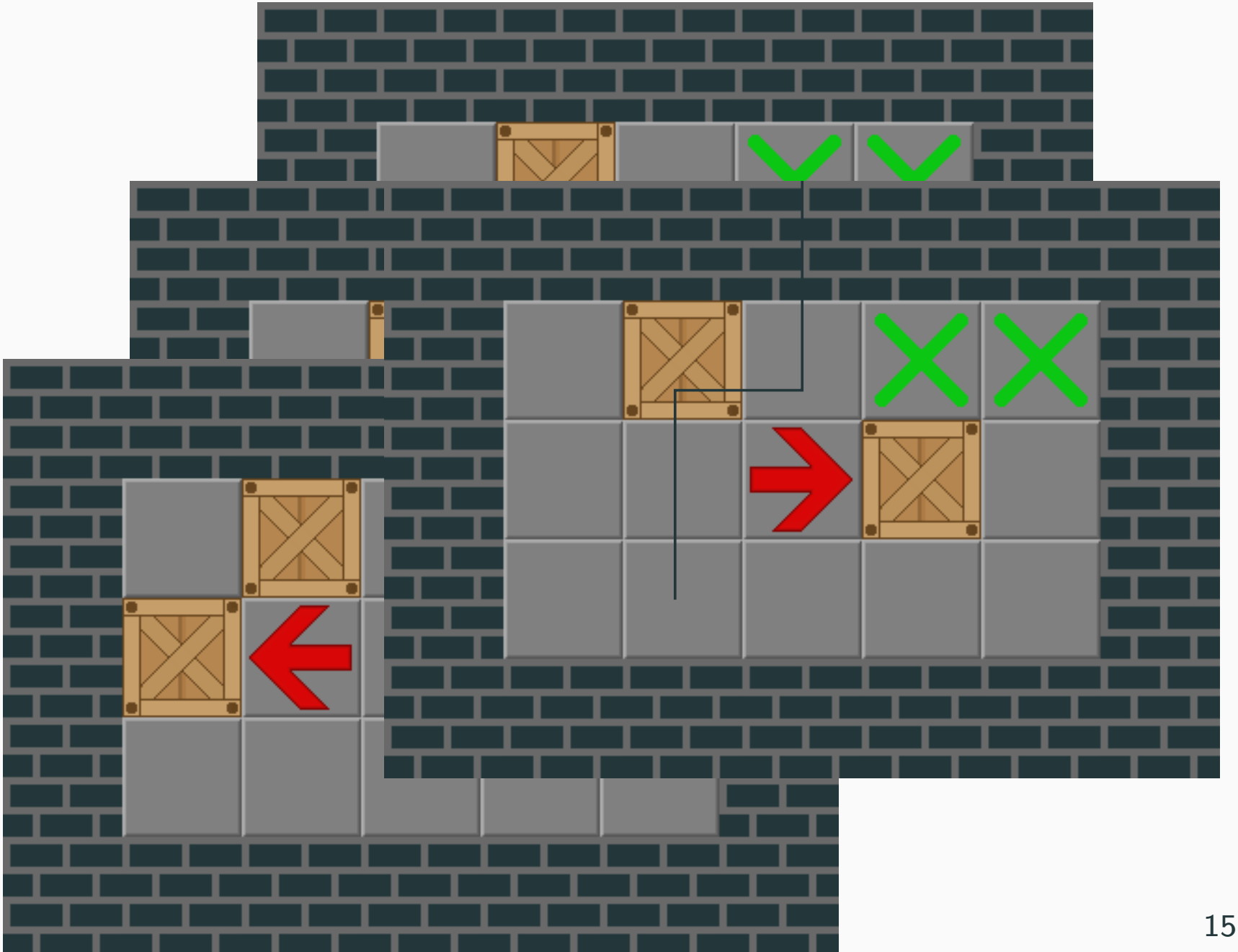
Arbre des états



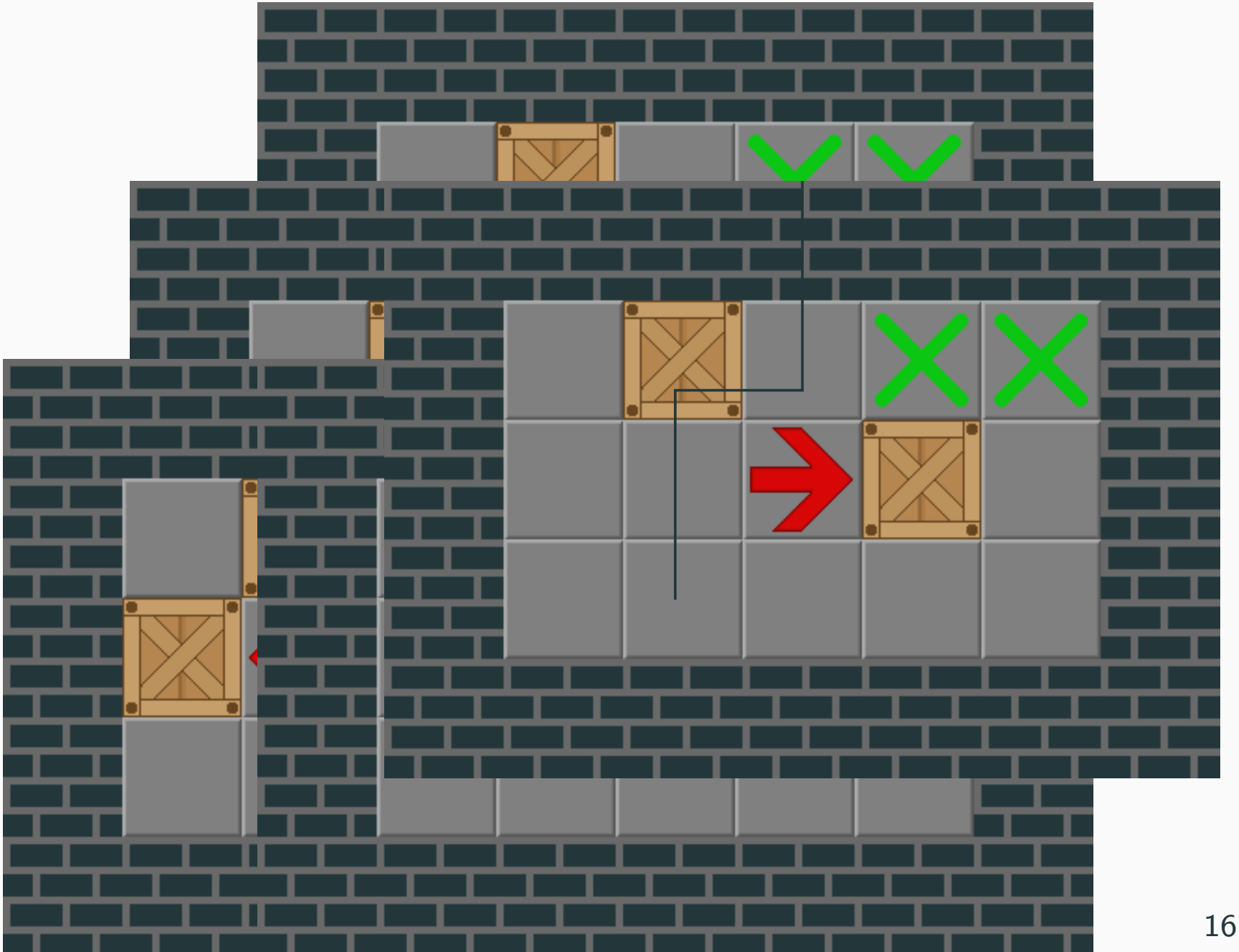
Arbre des états



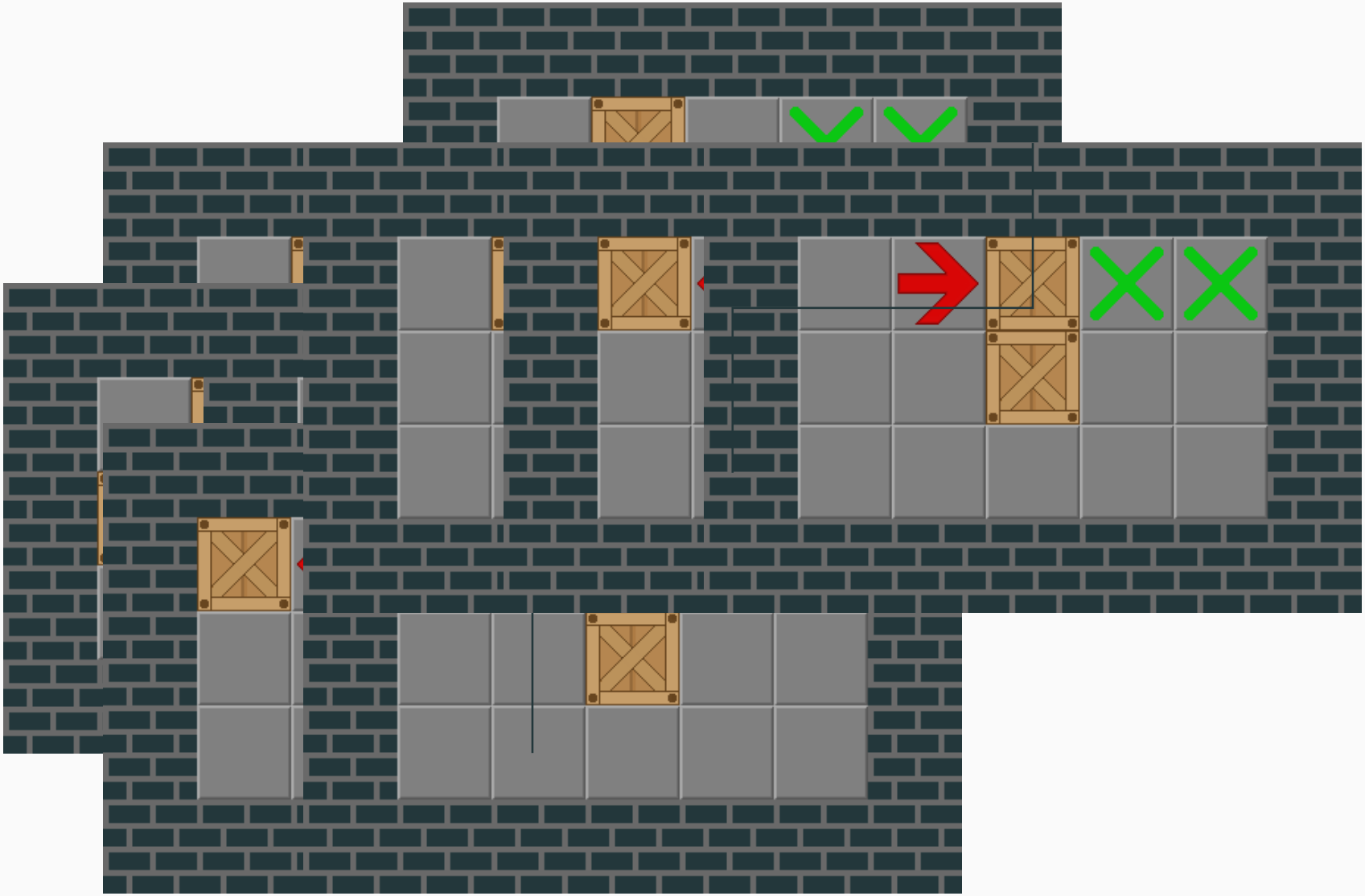
Arbre des états



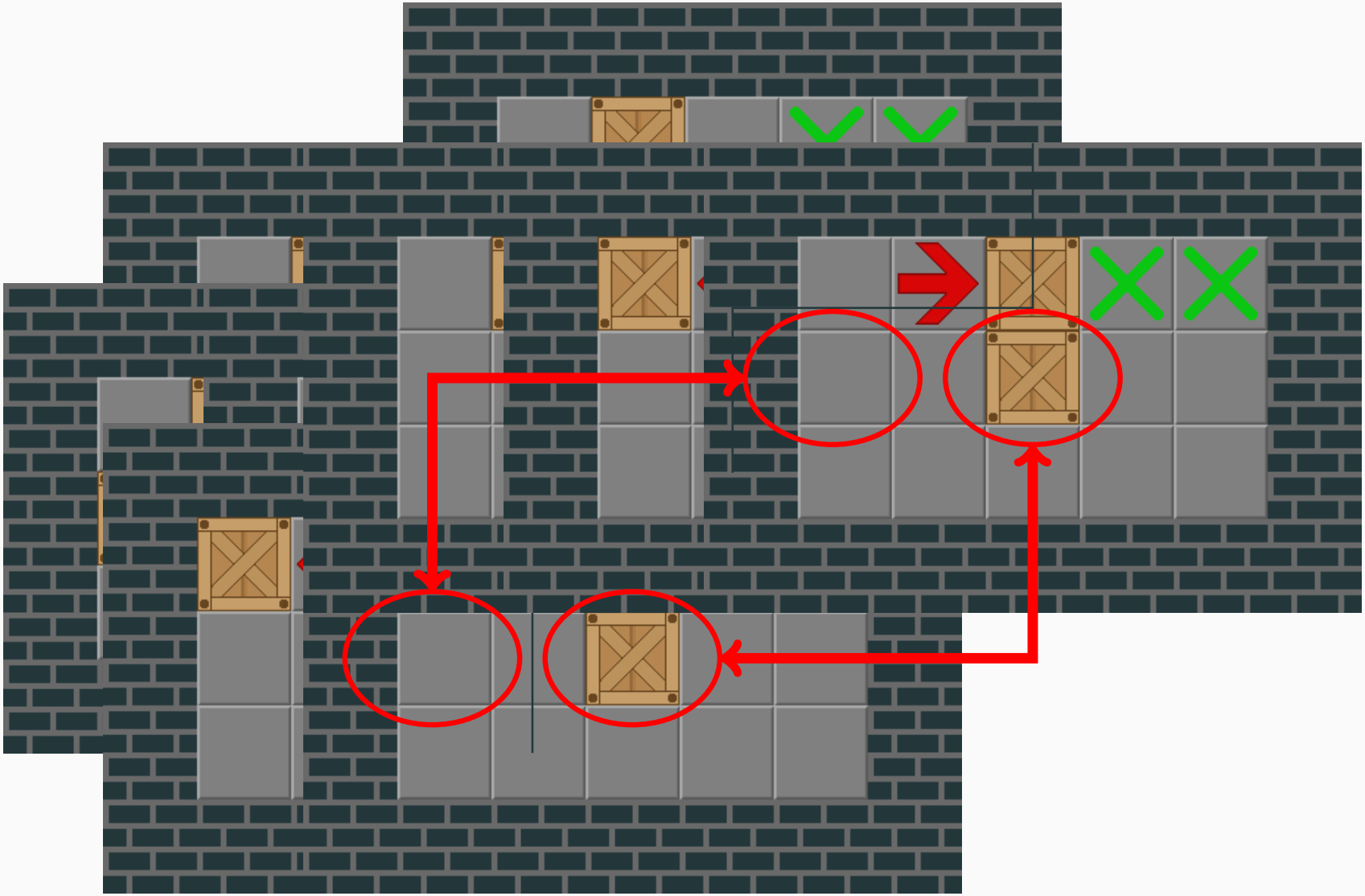
Arbre des états



Exemple développé



Un graphe vu comme un arbre



Hash de Zobrist

Initialisation :

$$T = \begin{array}{c} \begin{array}{cc} \text{caisse} & \text{joueur} \end{array} \\ \begin{pmatrix} 6357 & 5742 \\ -1378 & 42 \\ \vdots & \vdots \\ 93268 & -278 \end{pmatrix} \end{array} \begin{array}{c} \text{case} \\ 0 \\ 1 \\ \vdots \\ wh-1 \end{array}$$

Usage : $(c_1, ..., c_n)$ n caisses et p position du joueur :

$h = \overset{n}{\underset{i=0}{\mathbf{XOR}}} T[c_i][0] \,\mathbf{XOR}\, T[p][1]$

Passer d'une configuration à une autre : $c_i \to c_i', p \to p'$

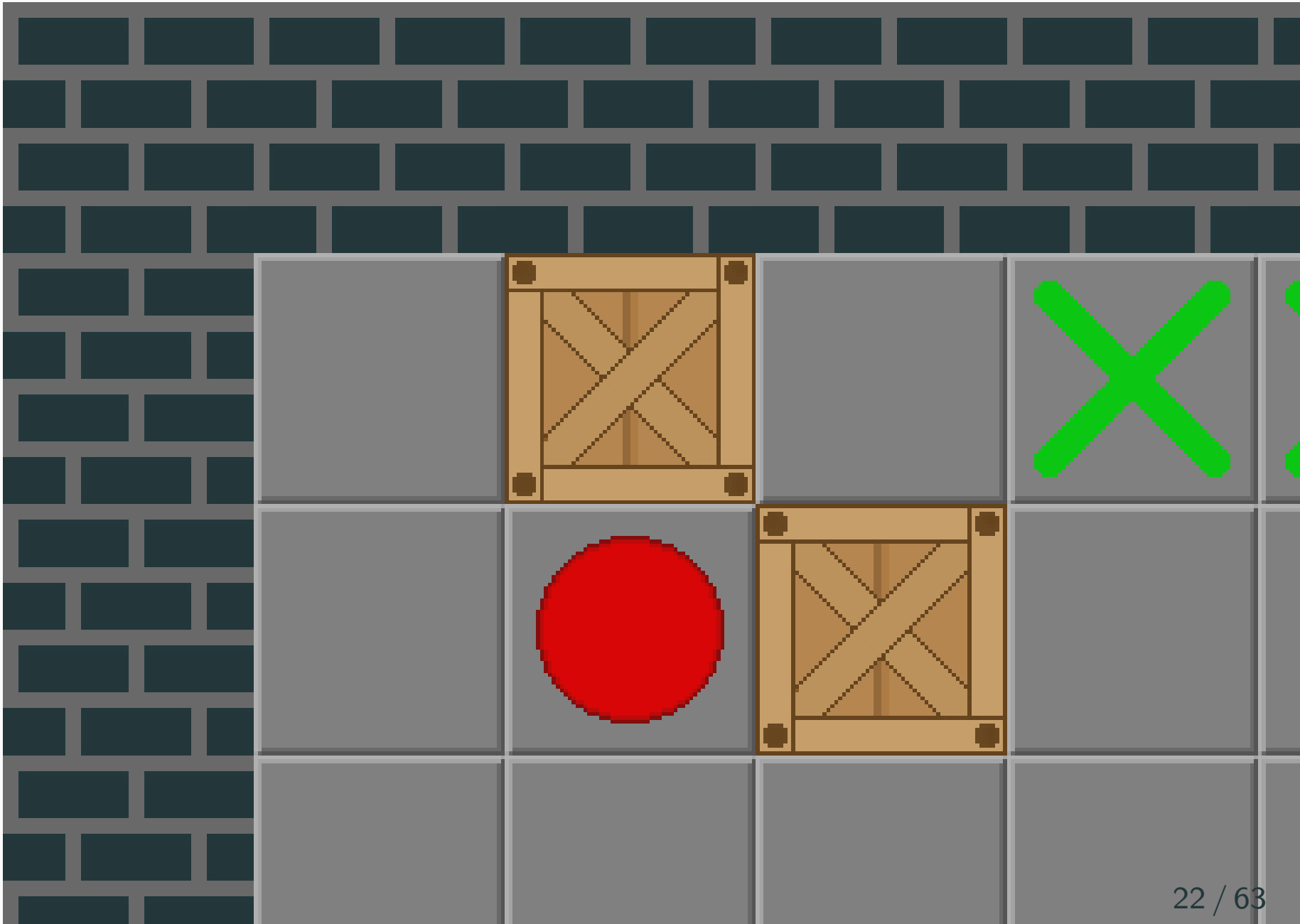
$h = h \,\mathbf{XOR}\, T[c_i][0] \,\mathbf{XOR}\, T[c_i'][0] \,\mathbf{XOR}\, T[p][1] \,\mathbf{XOR}\, T[p'][1]$

Réduction de l'espace de recherche

Réduction de l'espace de recherche

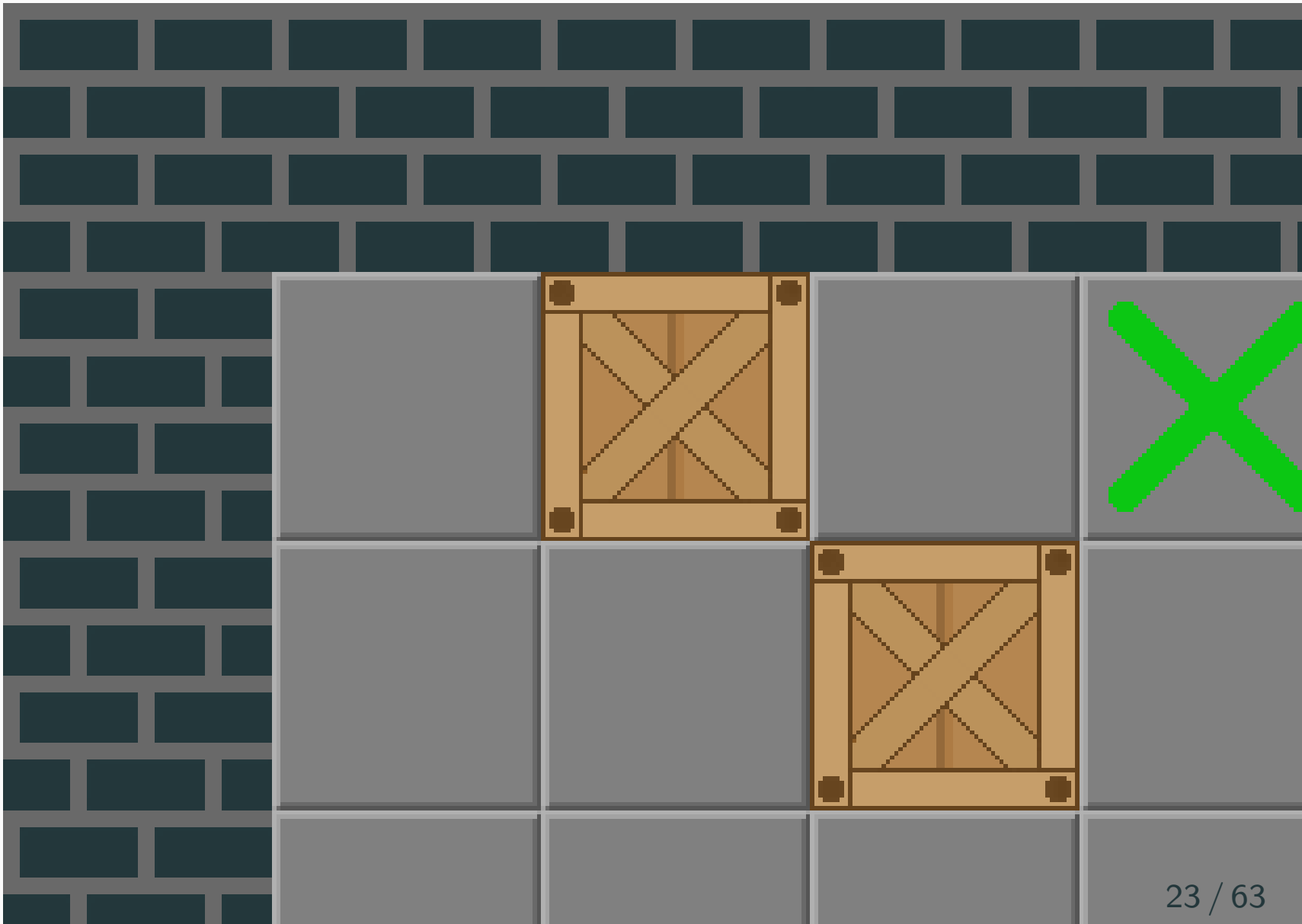
Analyse statique

Détection des positions mortes *(dead positions)*



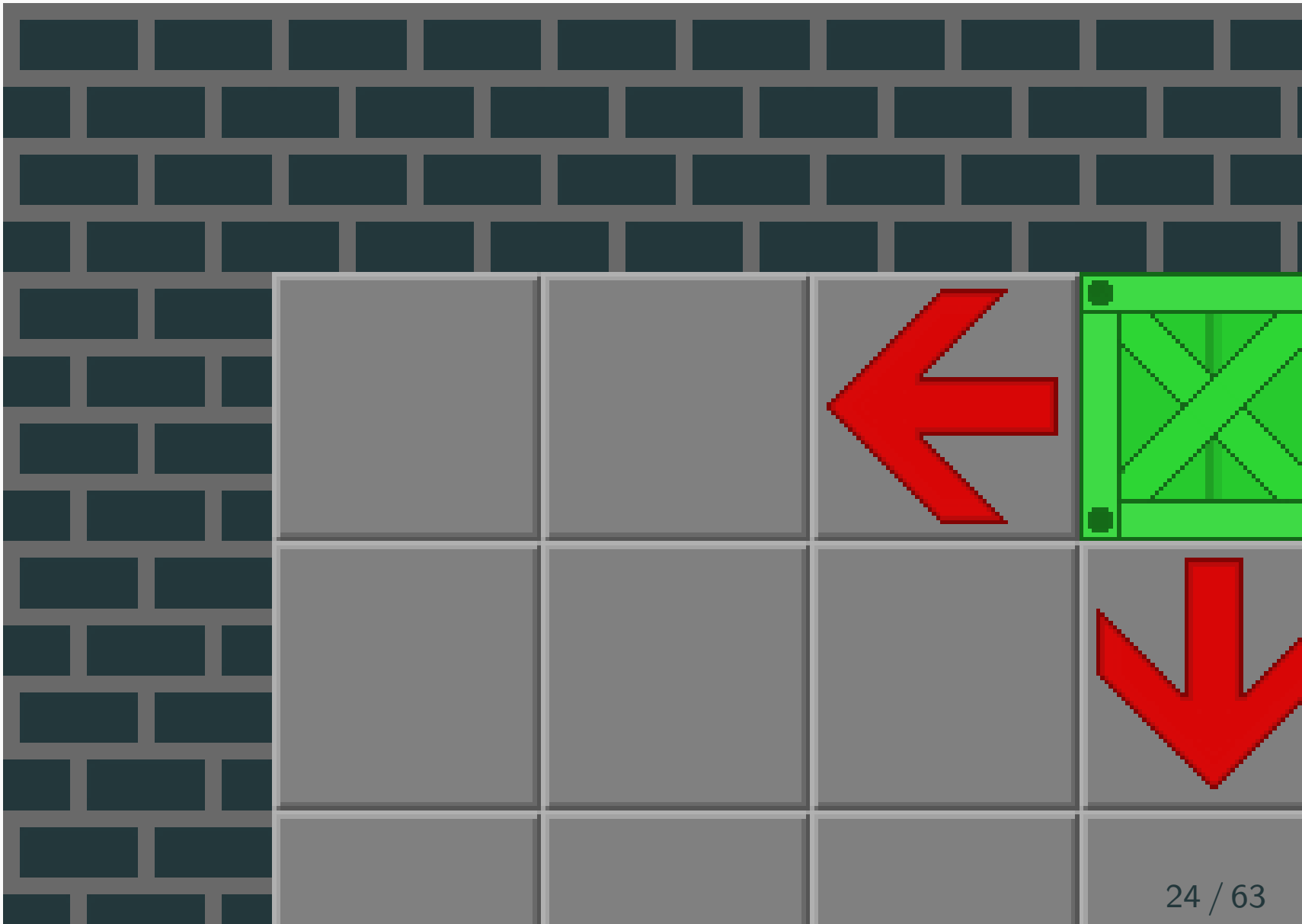
Détection des positions mortes *(dead positions)*

1.



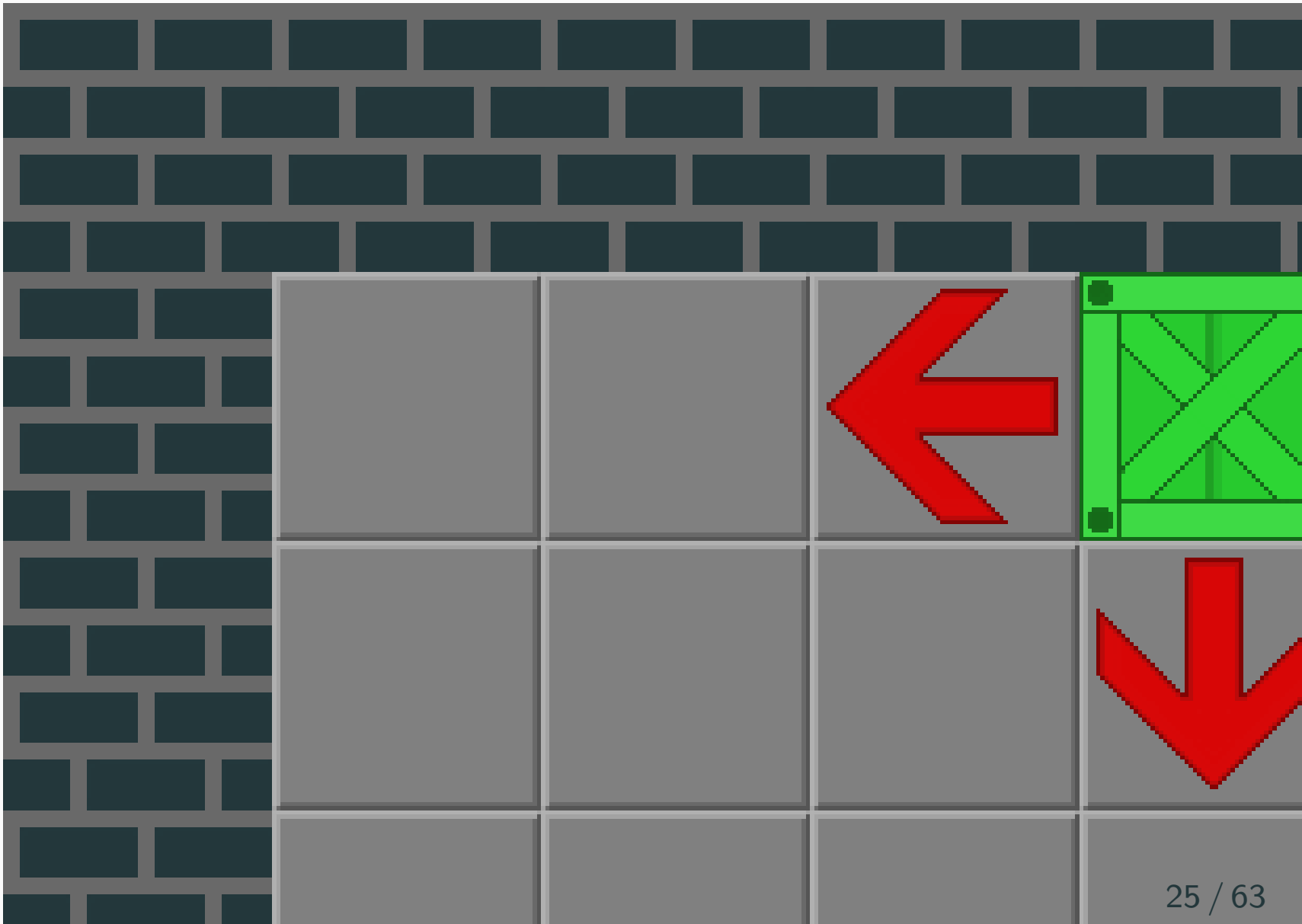
Détection des positions mortes *(dead positions)*

2.

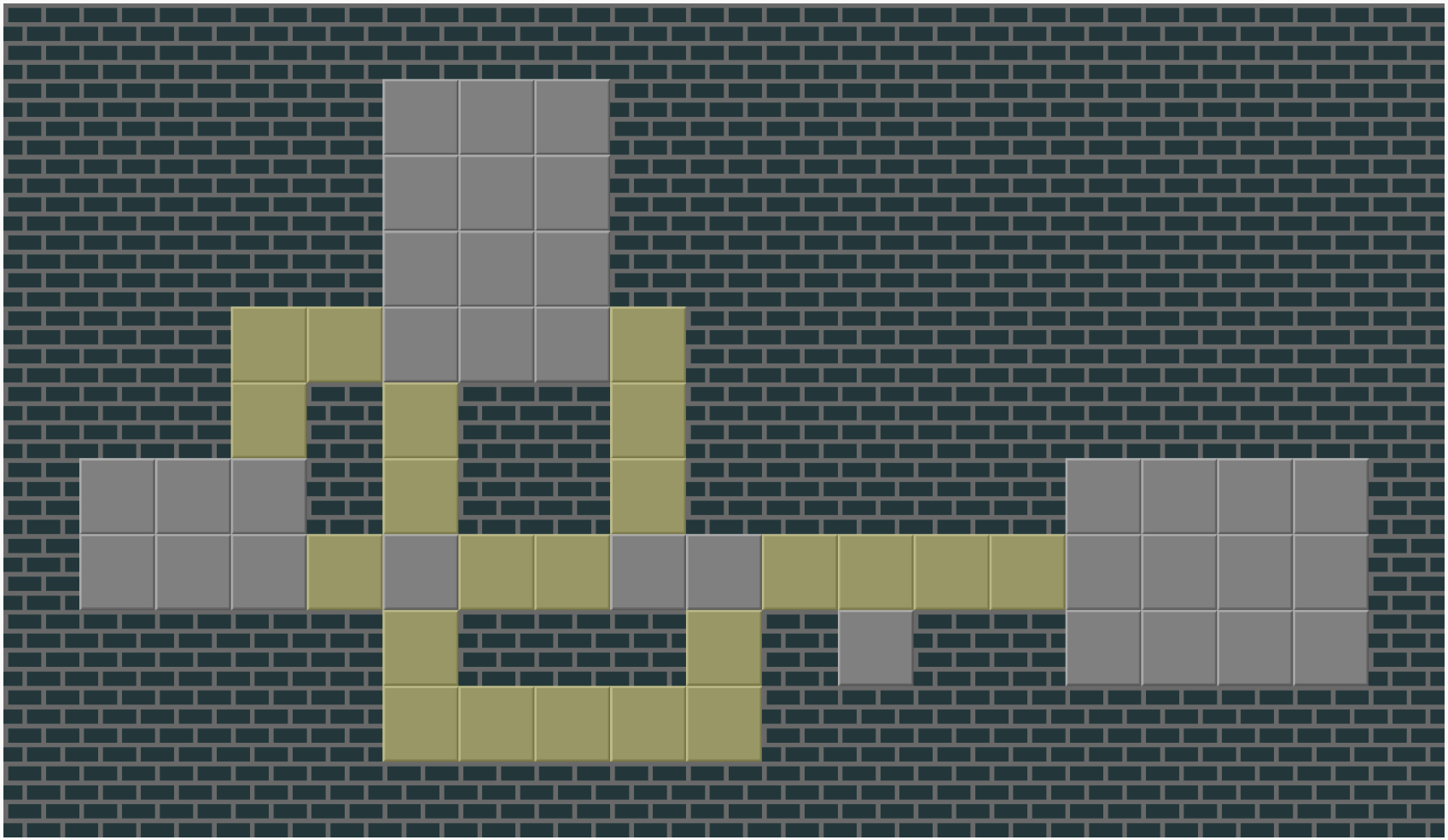


Détection des positions mortes *(dead positions)*

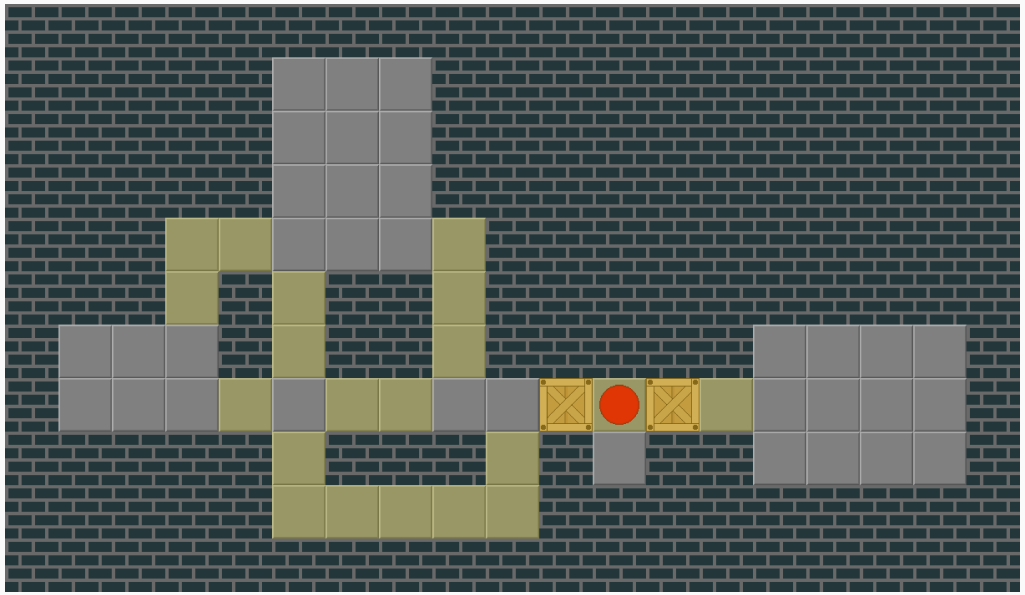
2.

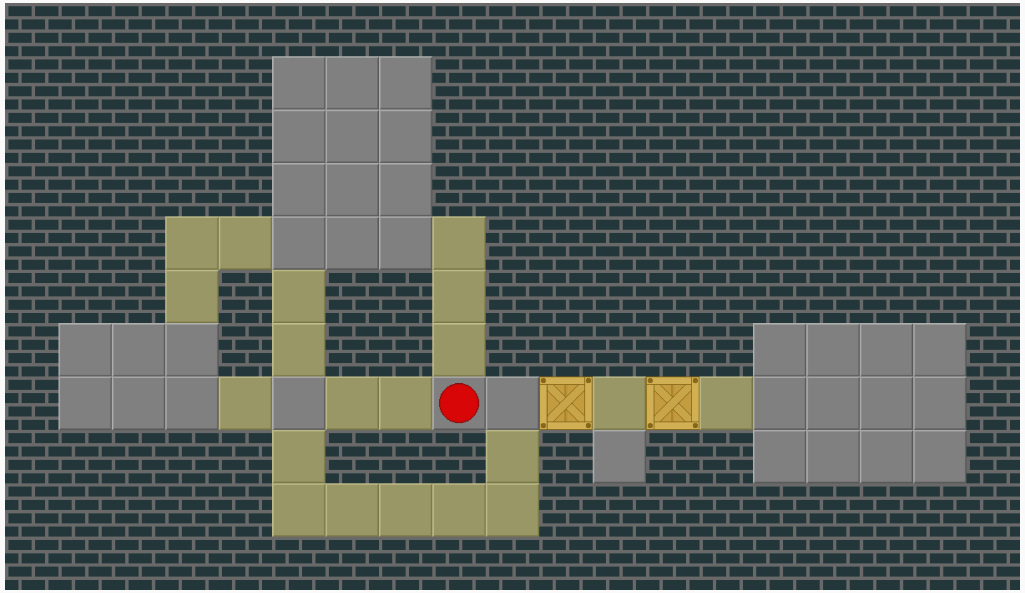


Détection de tunnels

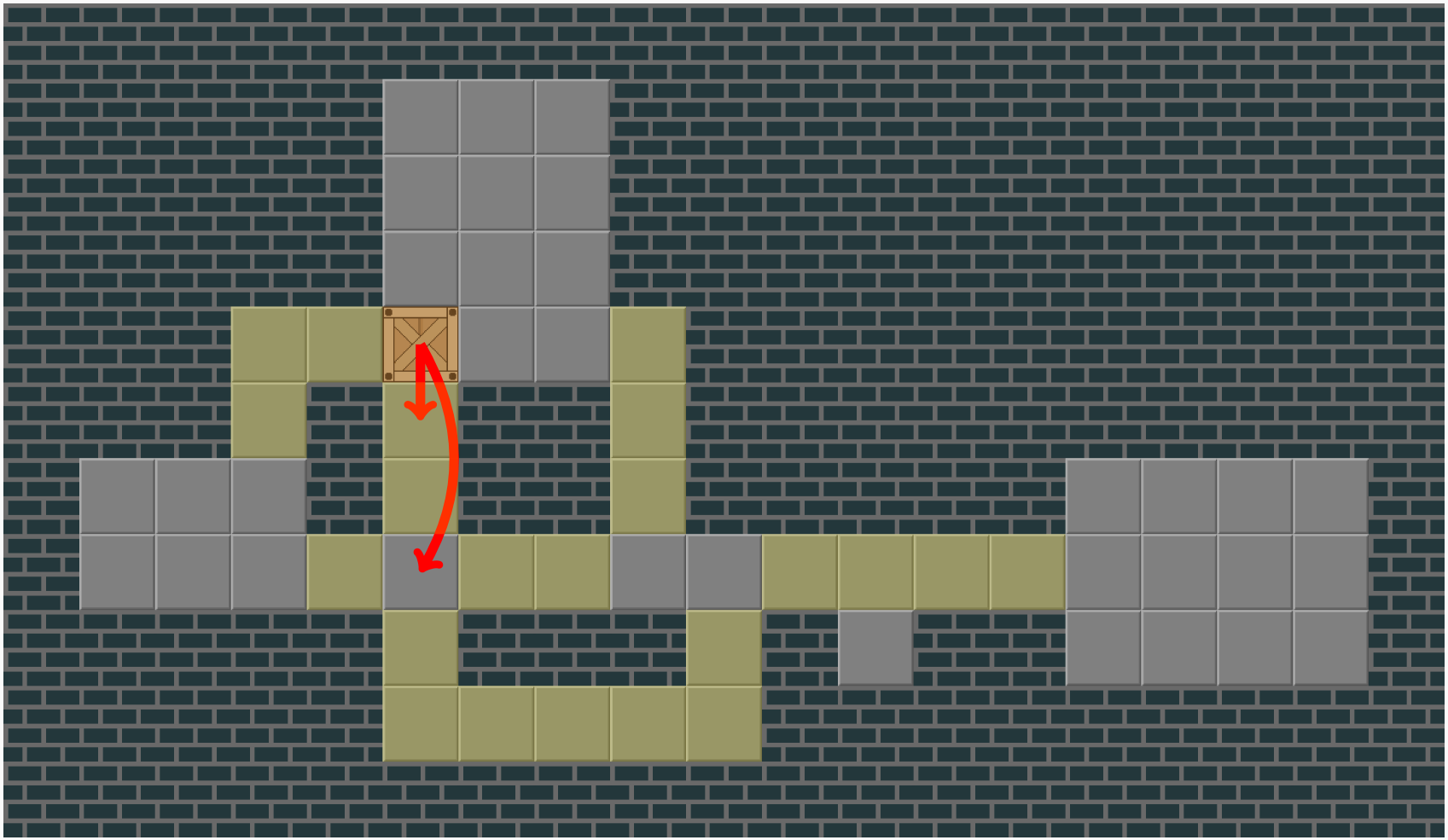


Détection de tunnels

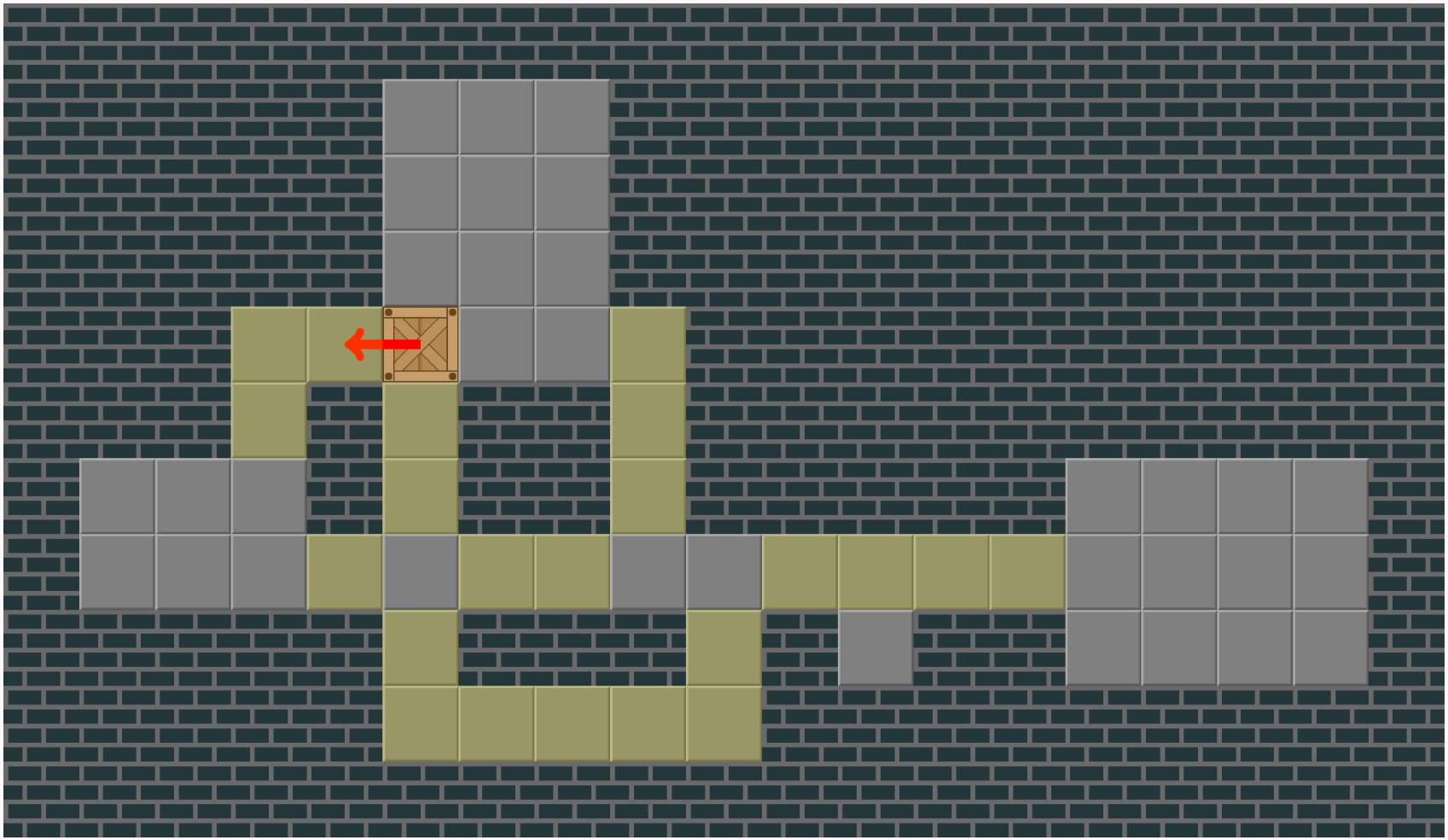




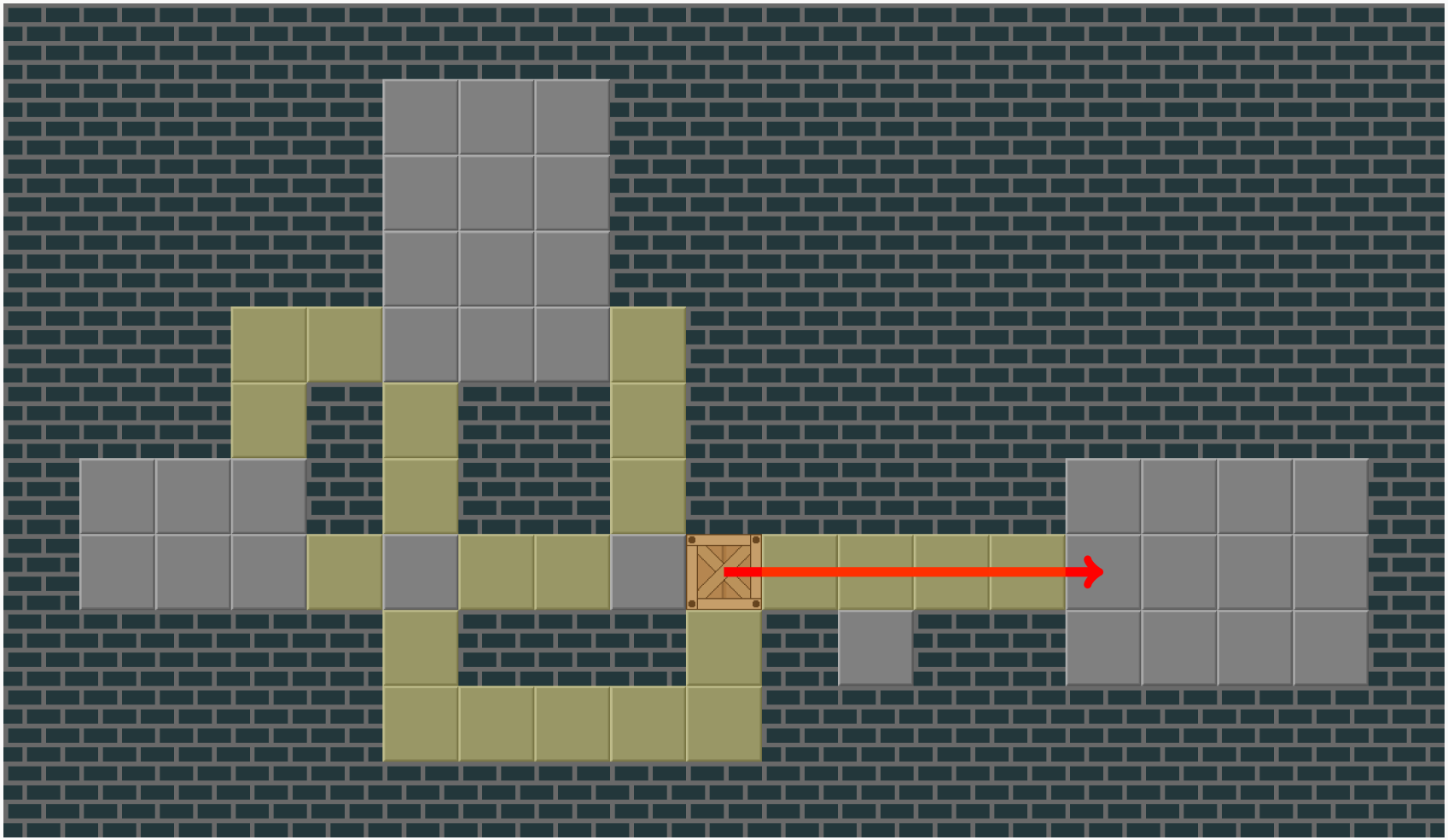
Détection de tunnels



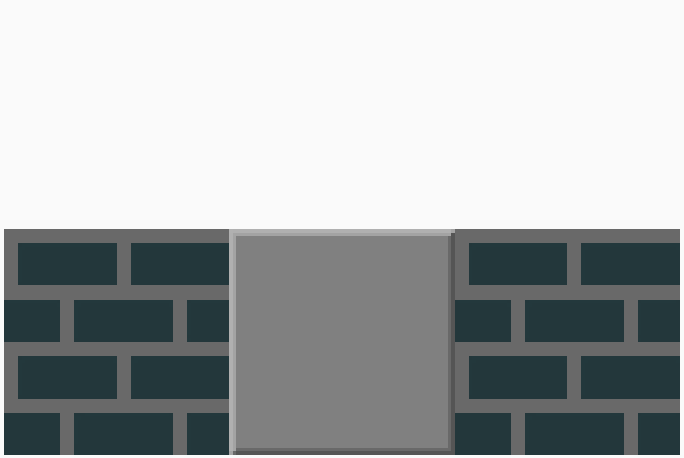
Détection de tunnels

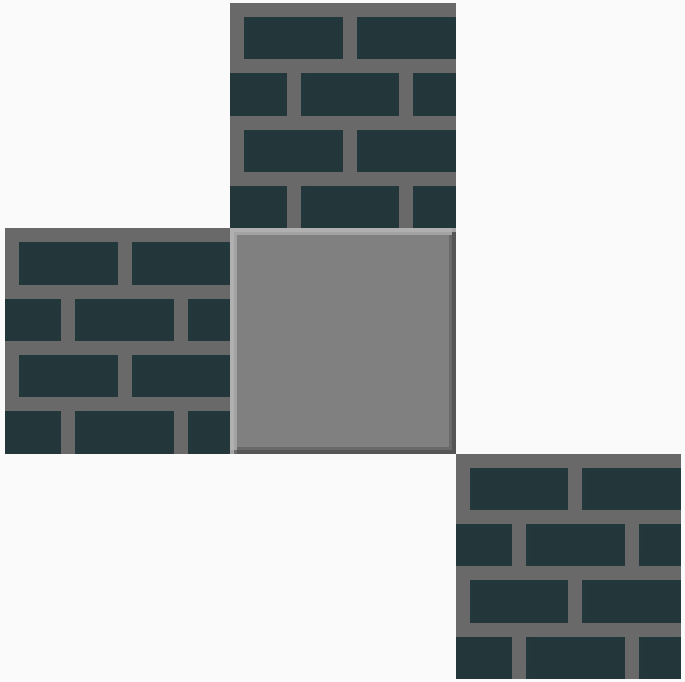


Détection de tunnels

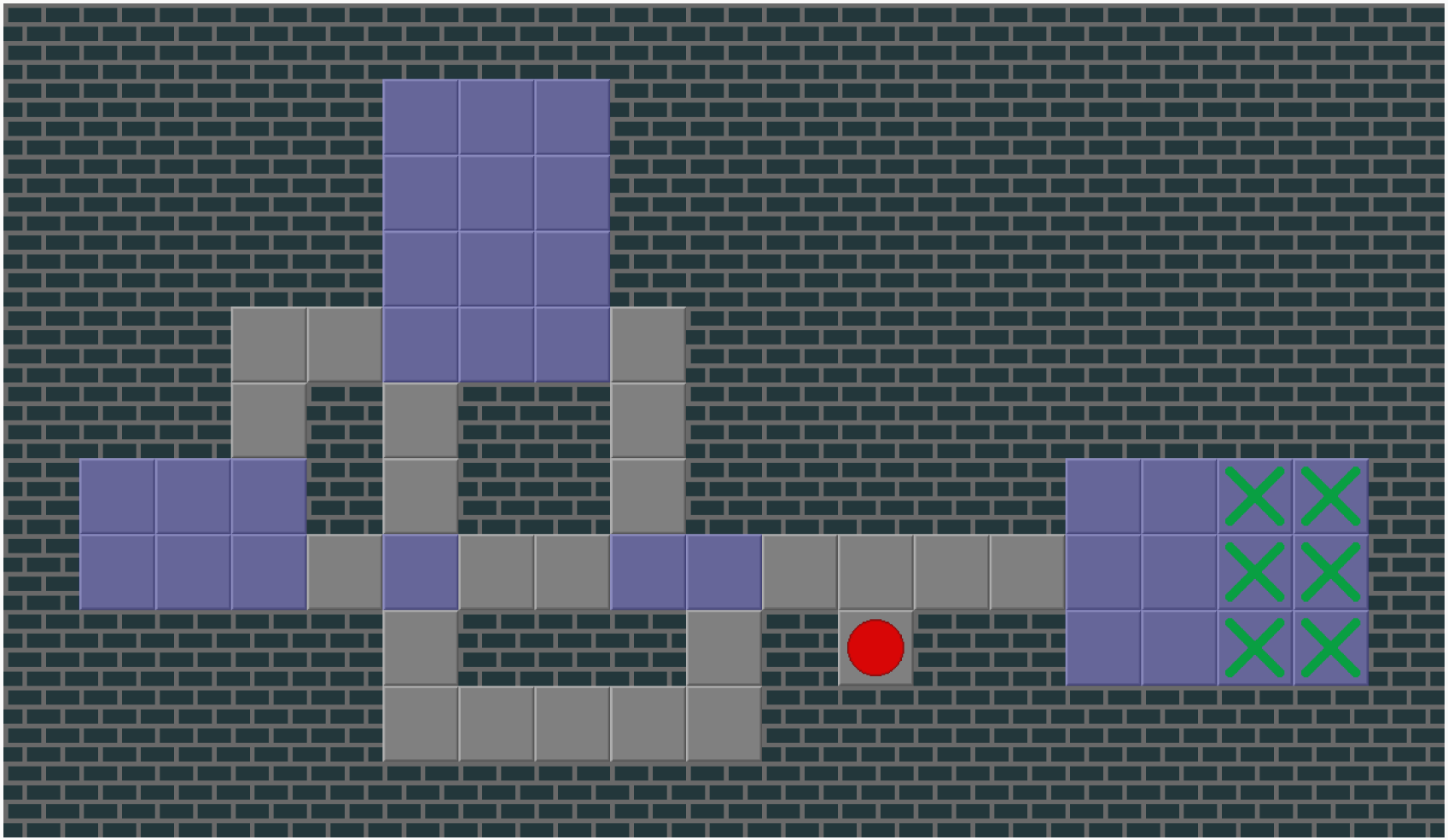


Détection de tunnels

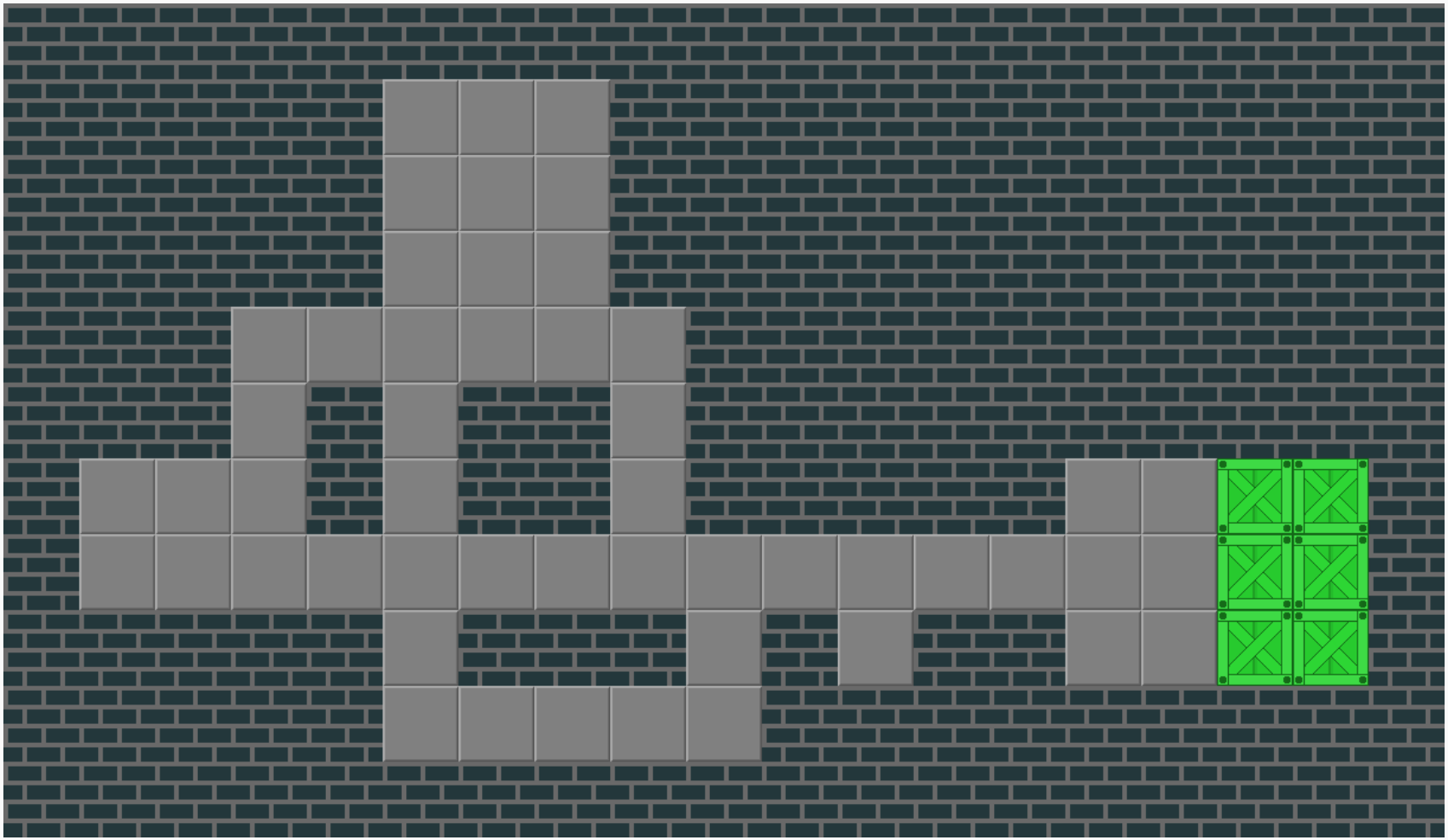




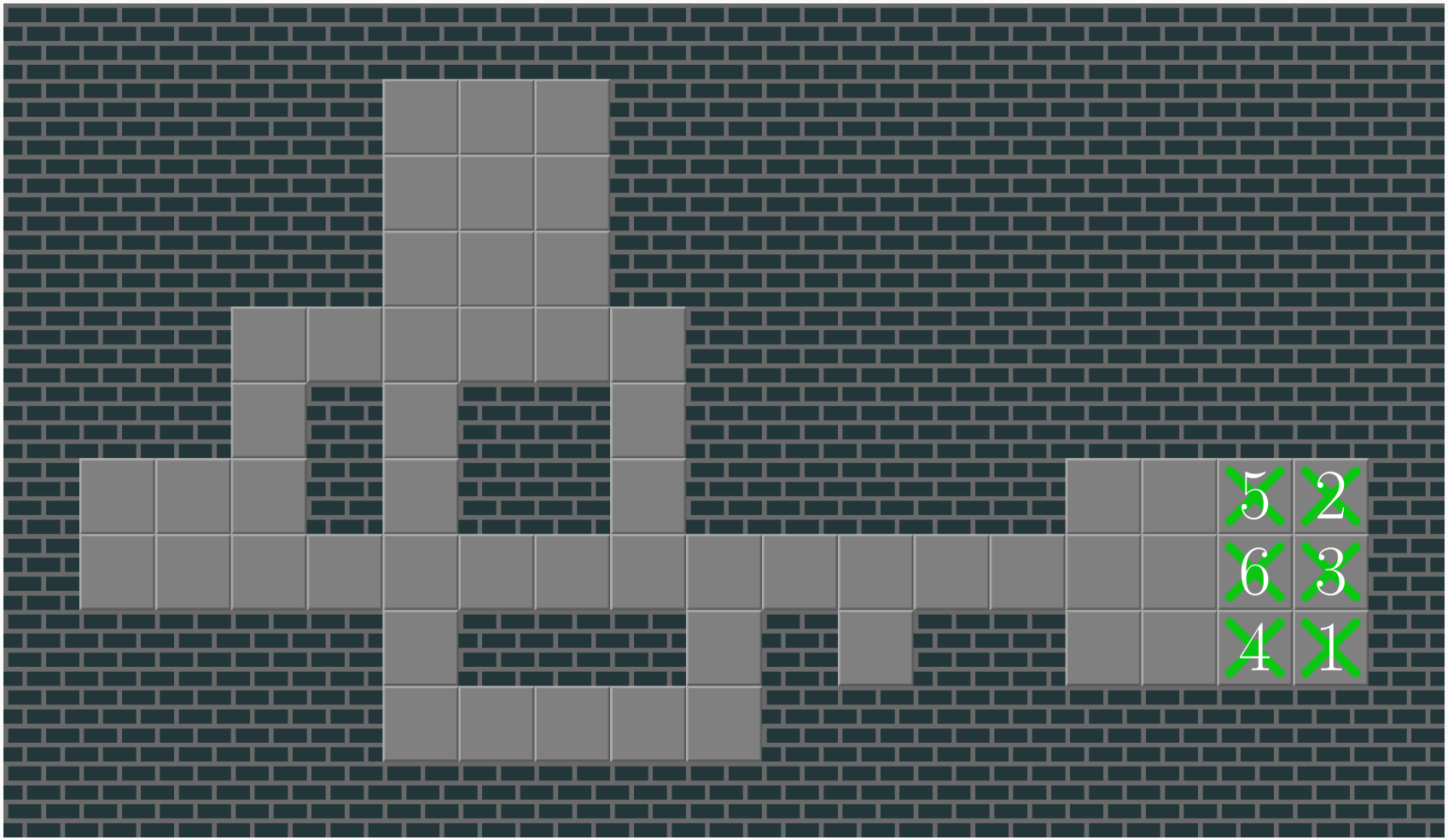
Salles et ordre de rangement *(packing order)*



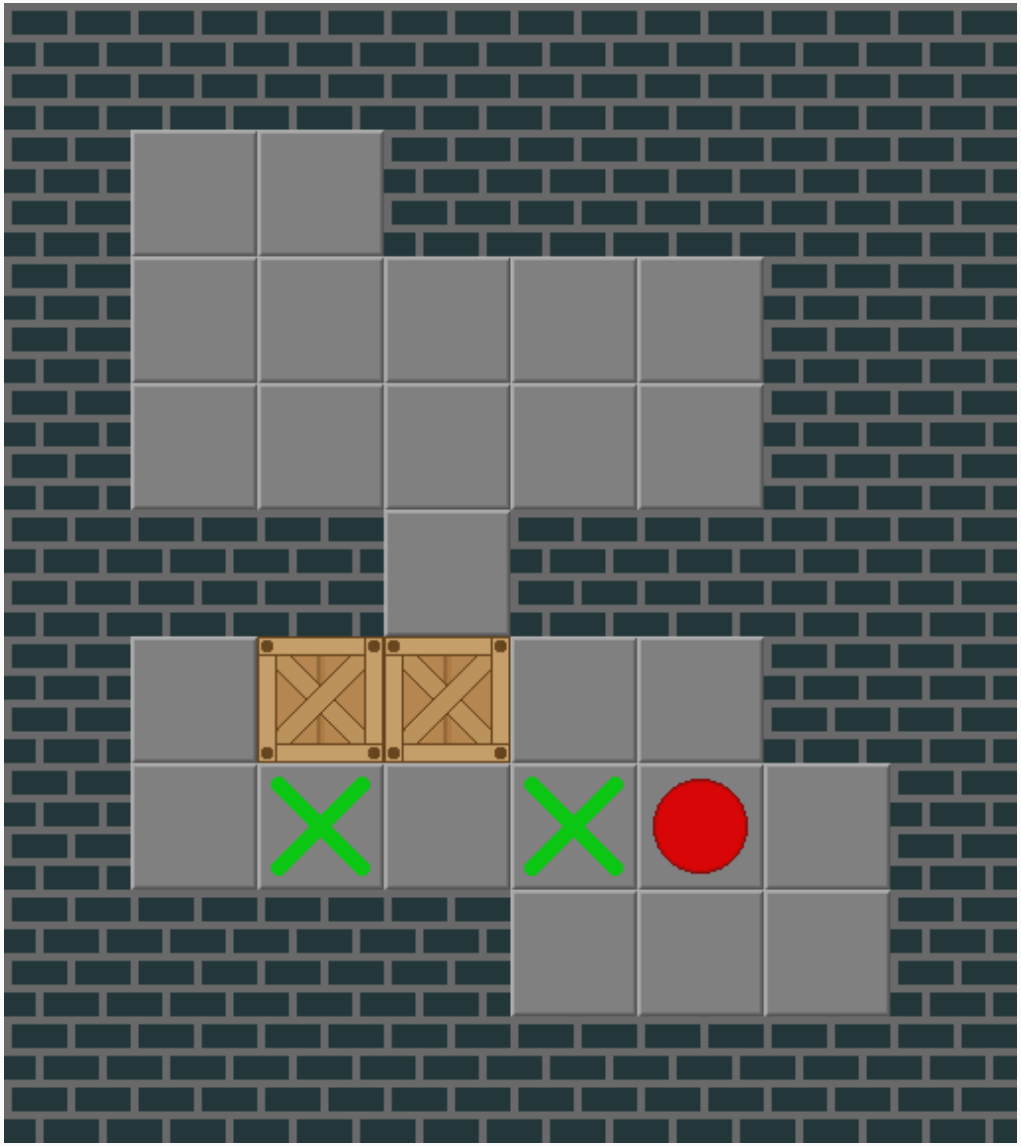
Salles et ordre de rangement *(packing order)*



Salles et ordre de rangement *(packing order)*



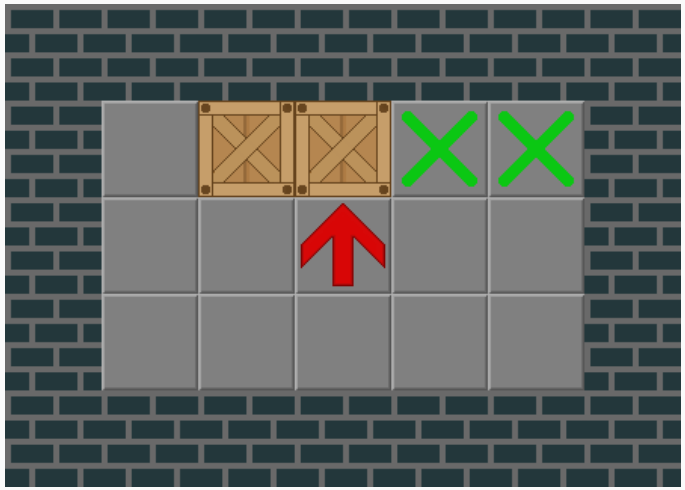
Salles et ordre de rangement *(packing order)*



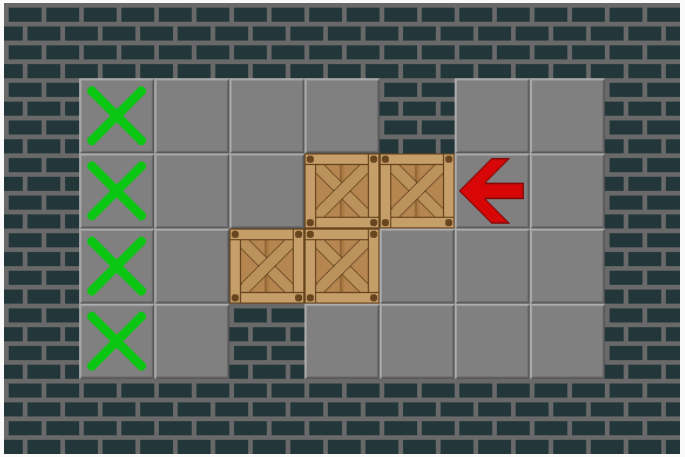
Réduction de l'espace de recherche

Analyse dynamique

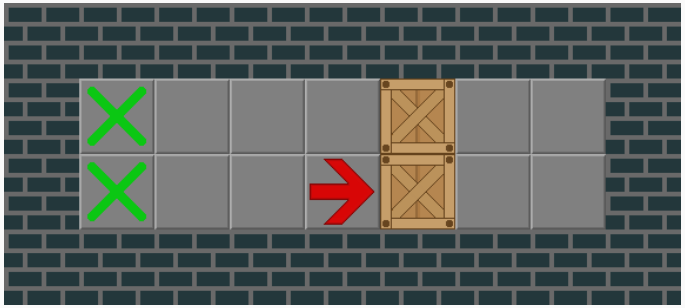
Détection d'impasses *(deadlocks)*



(a) *Freeze deadlock n°1*



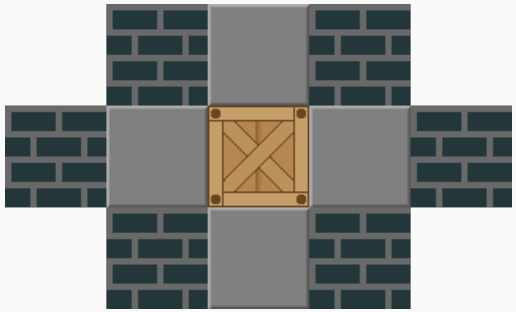
(b) *Freeze deadlock n°2*



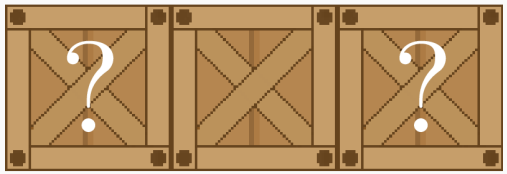
(c) *PI Corral deadlock*

Détection de *freeze deadlocks*

(a) *Règle n°1*

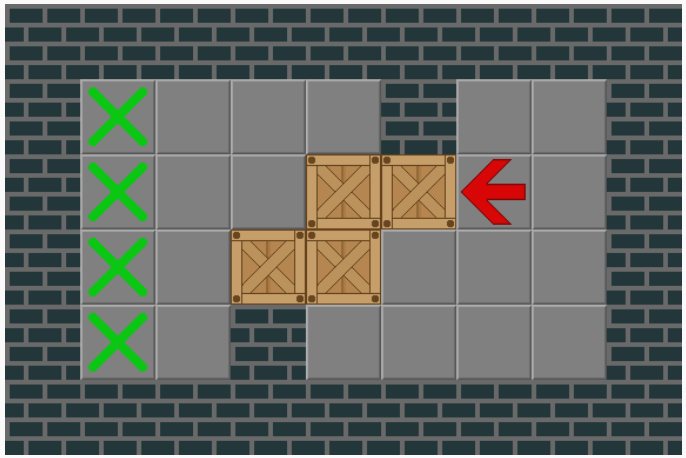


(b) *Règle n°2*

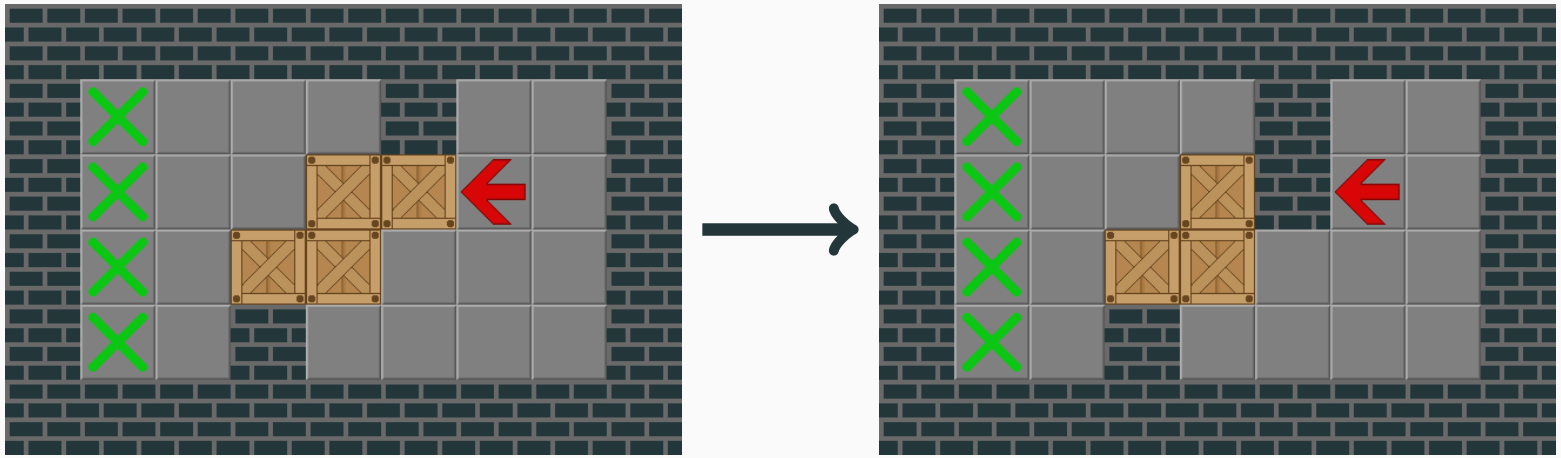


(c) *Règle n°3*

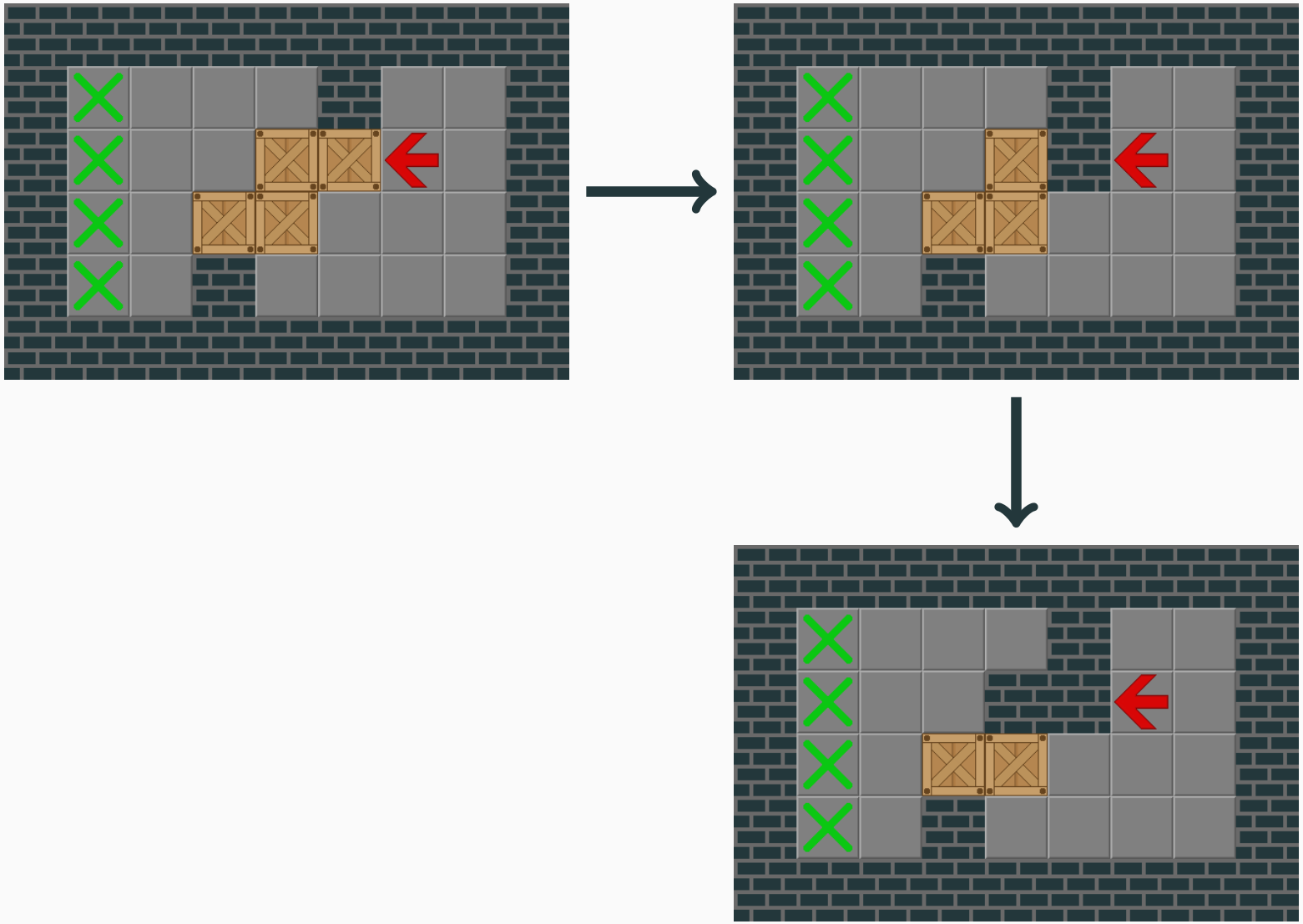
Détection de *freeze deadlocks*



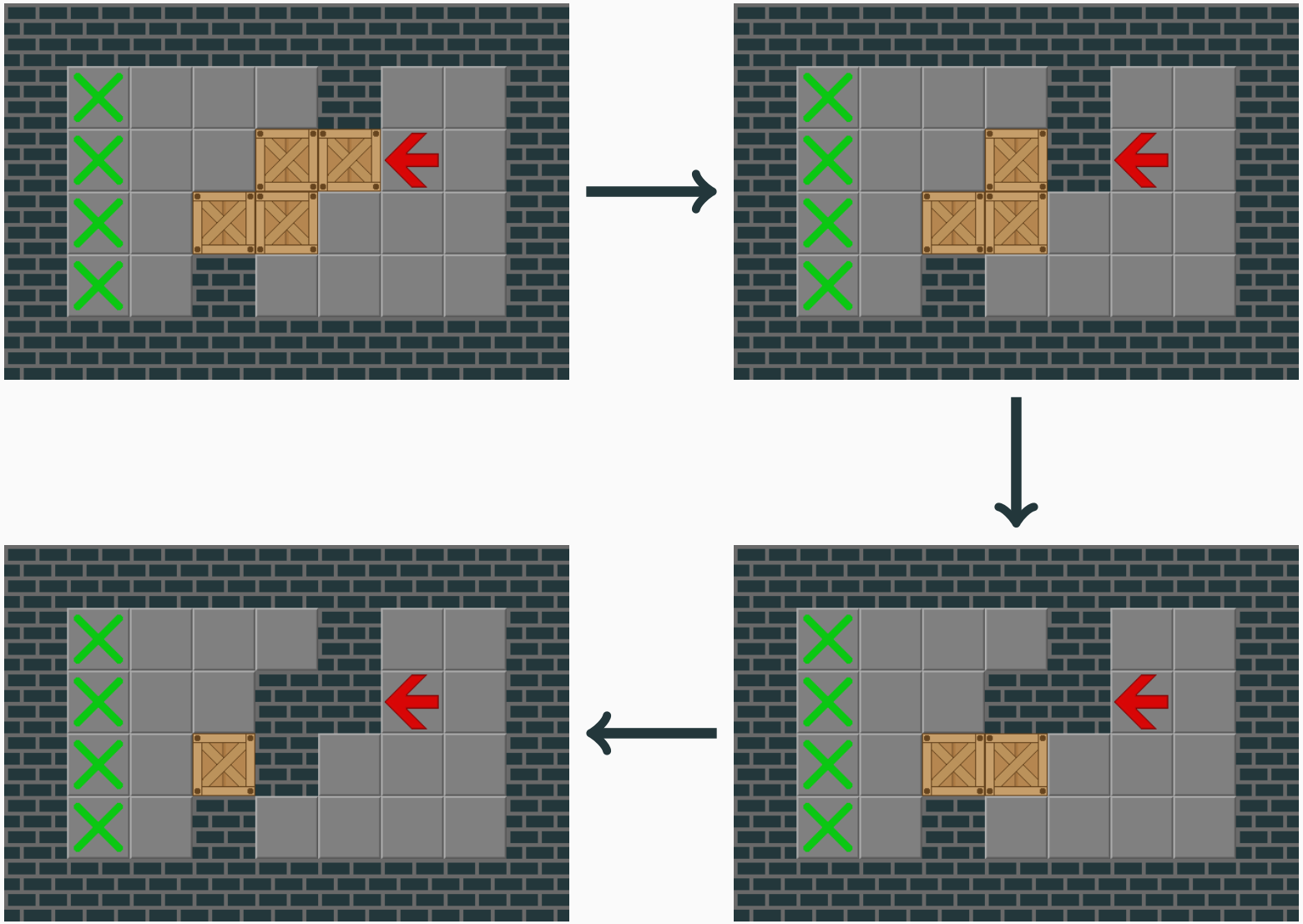
Détection de *freeze deadlocks*



Détection de *freeze deadlocks*

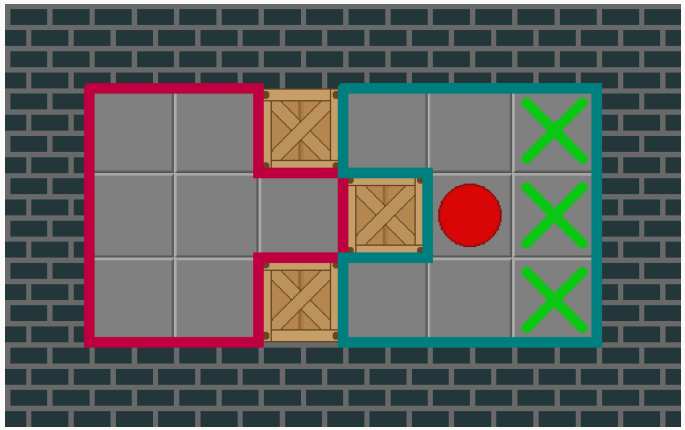


Détection de *freeze deadlocks*

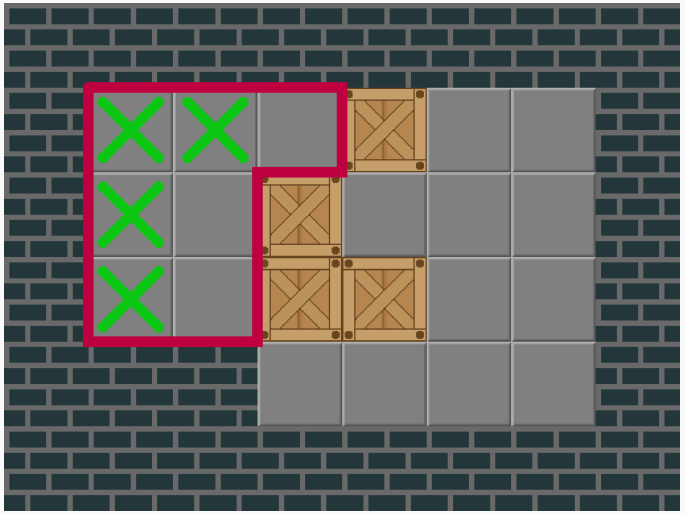


Gelée!

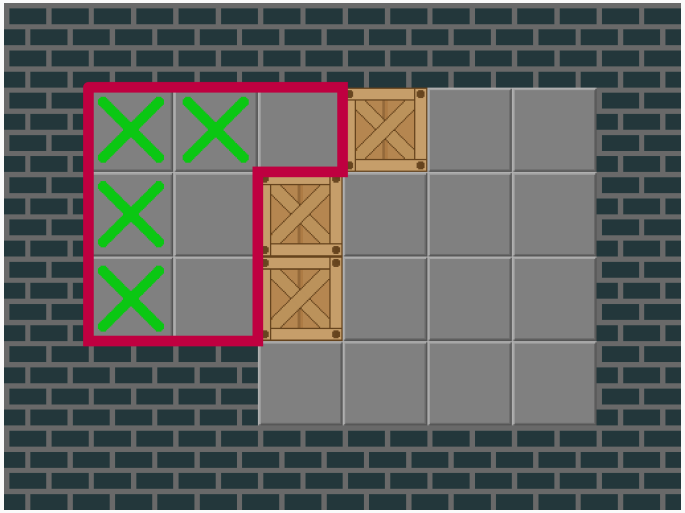
Détection de *PI Corral deadlocks*



(a) *Corral*

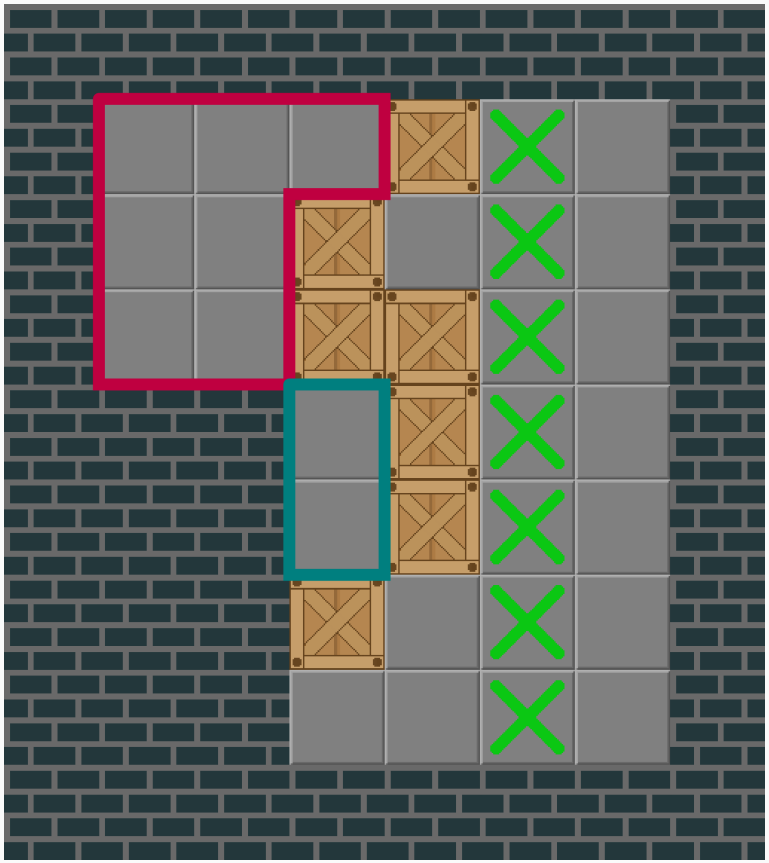


(b) *I Corral*



(c) *PI Corral*

Détection de *PI Corral deadlocks*



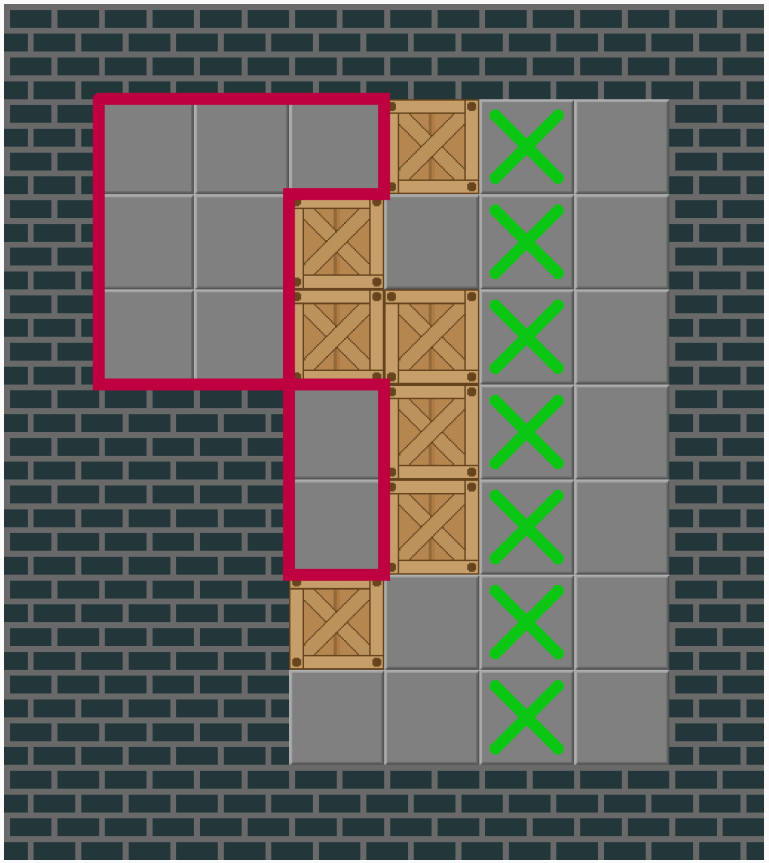
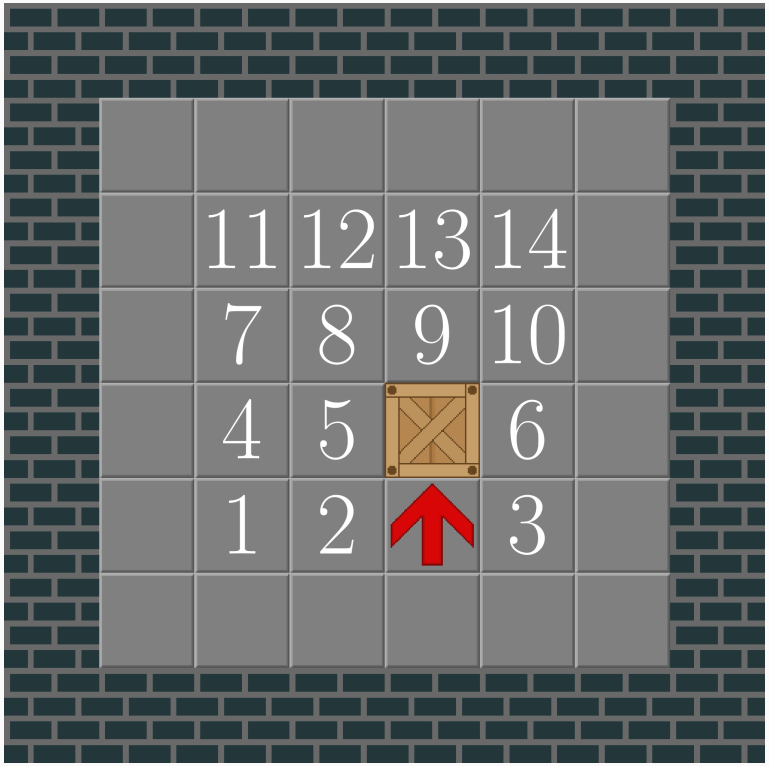


Table de *deadlocks*



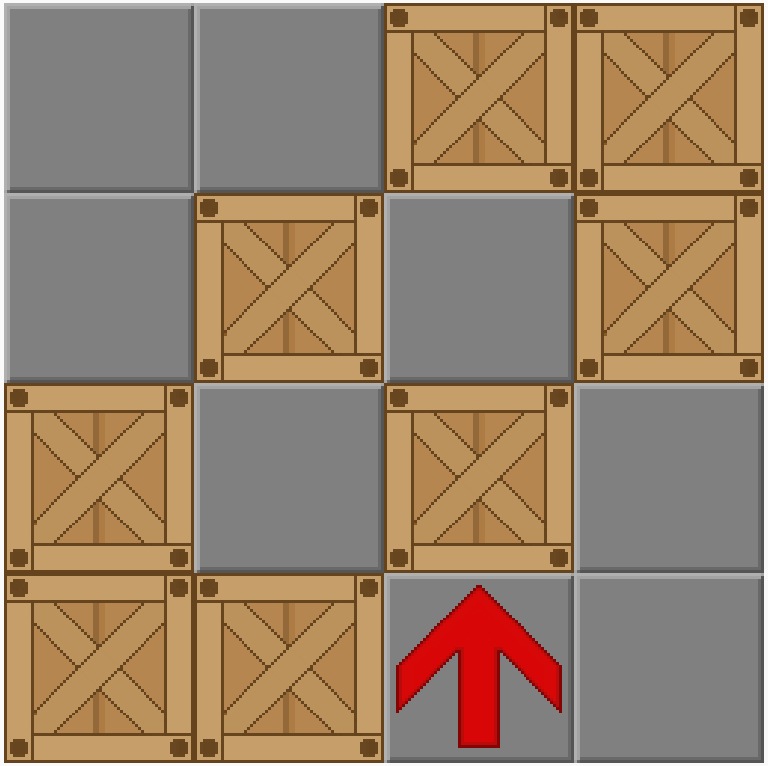
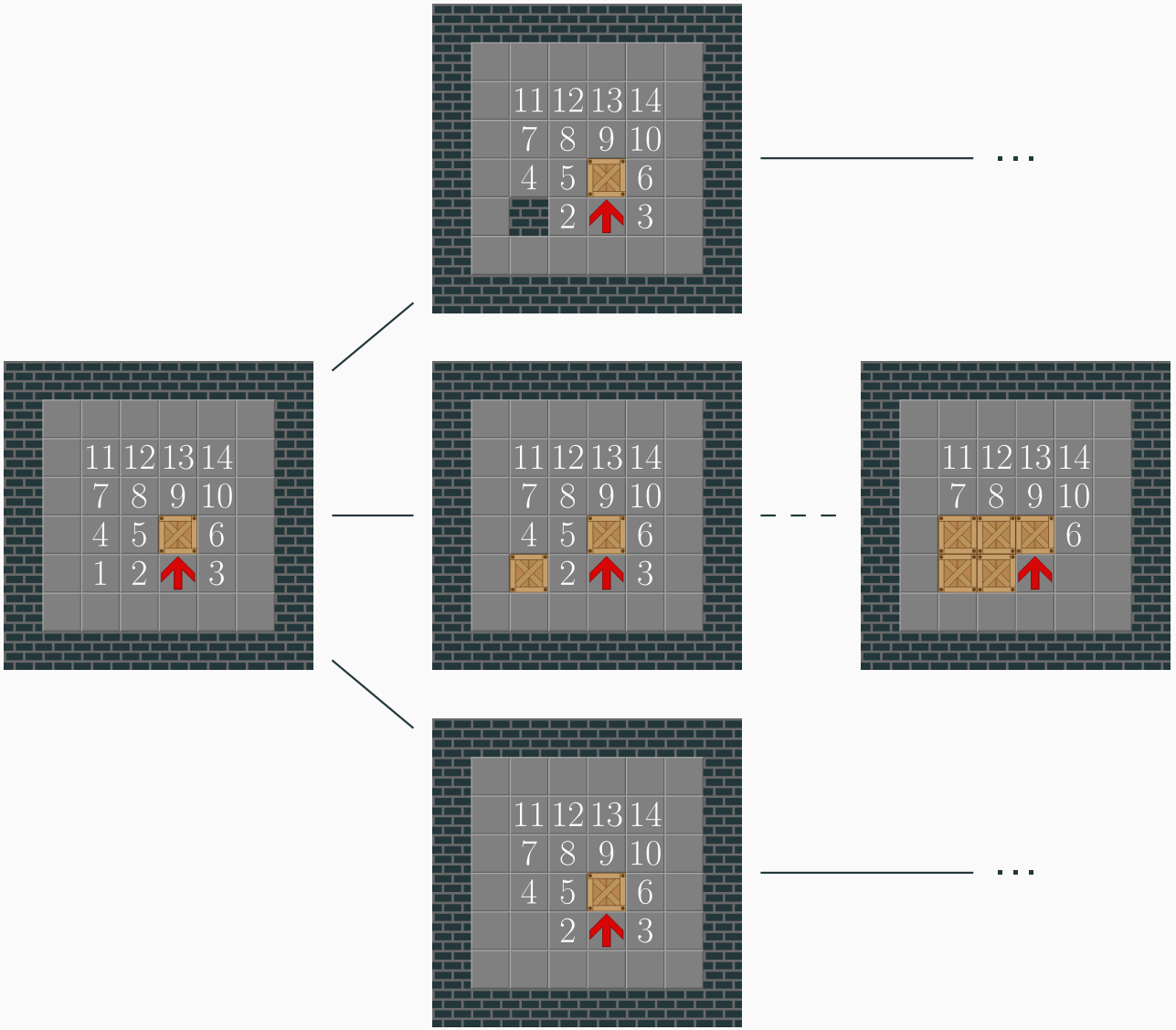


Table de *deadlocks*



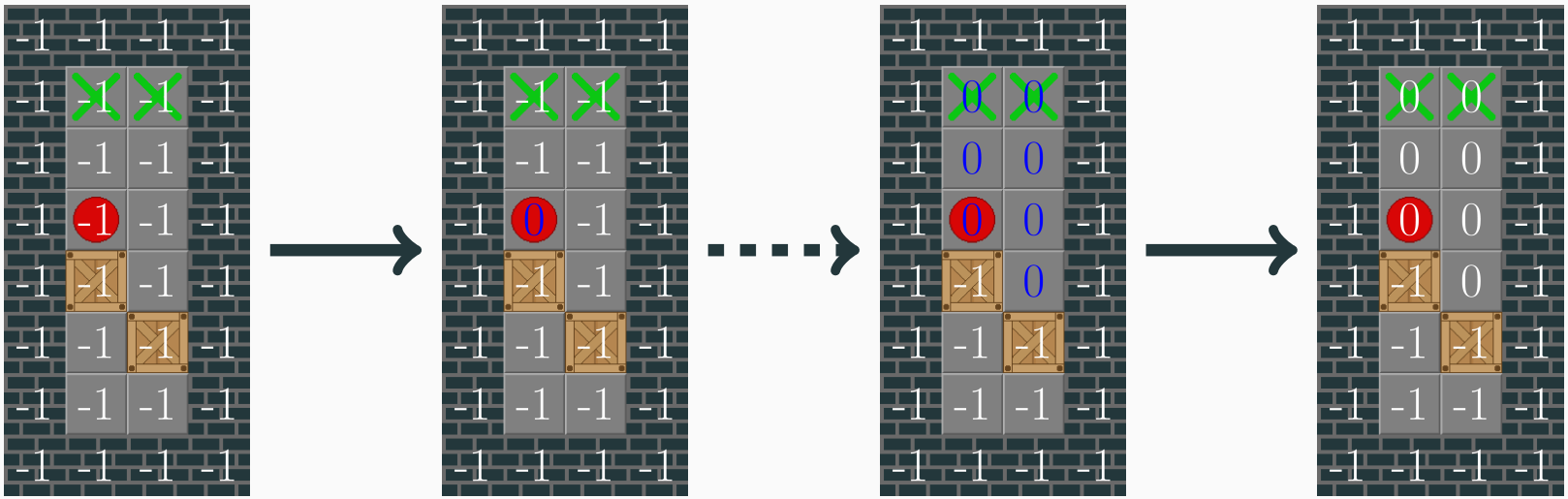
Recherche dirigée par une heuristique

Heuristique simple *(Simple Lower Bound)*

Heuristique gloutonne *(Greedy Lower Bound)*

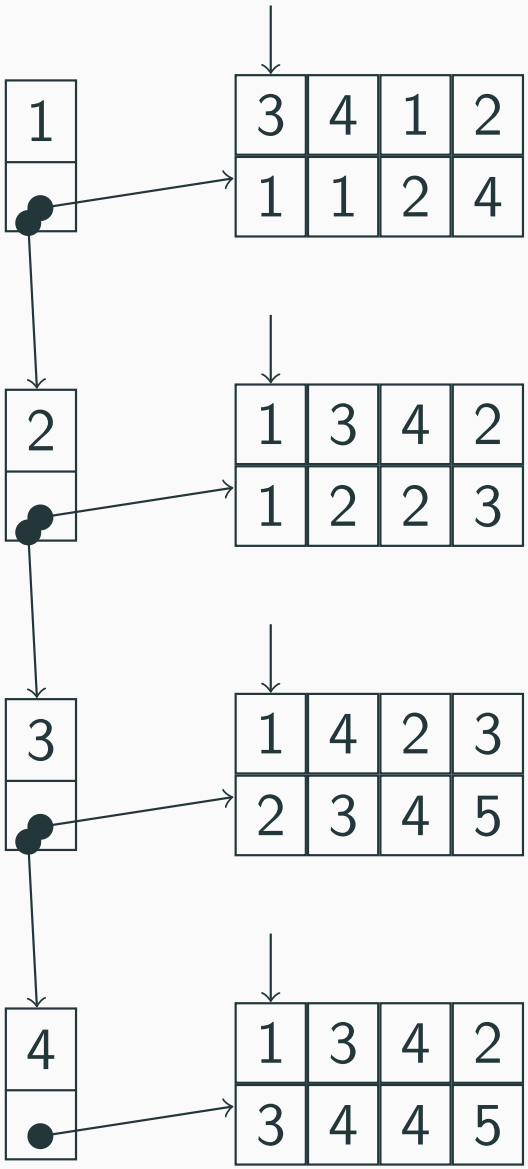
Optimisations

Parcours de graphes : démarquer tous les noeuds en $\mathcal{O}(1)$



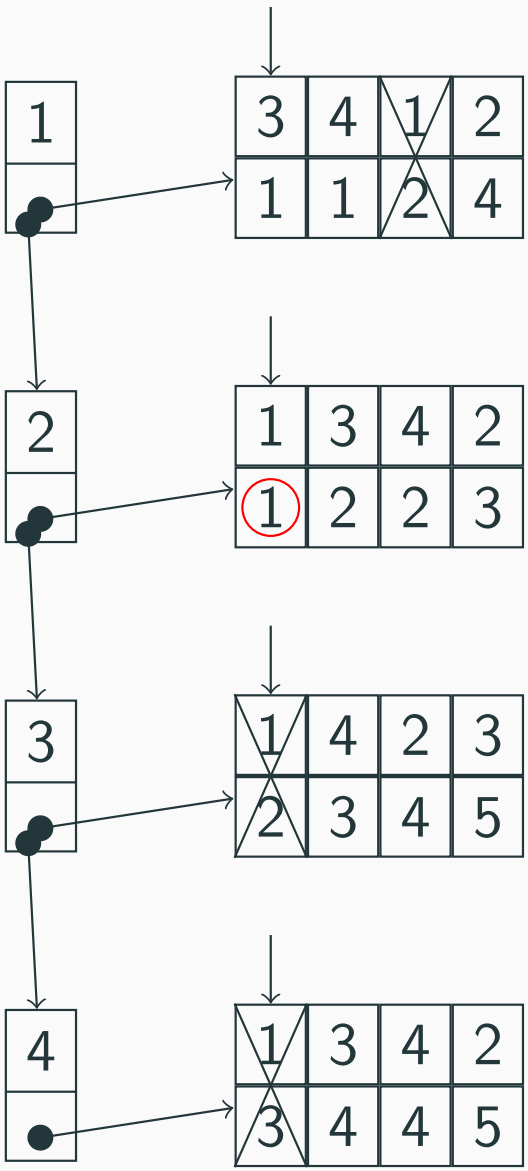
$m = 0$ $m = 0$ $m = 0$ $m = 1$

Greedy Lower Bound en $\mathcal{O}(n^2)$



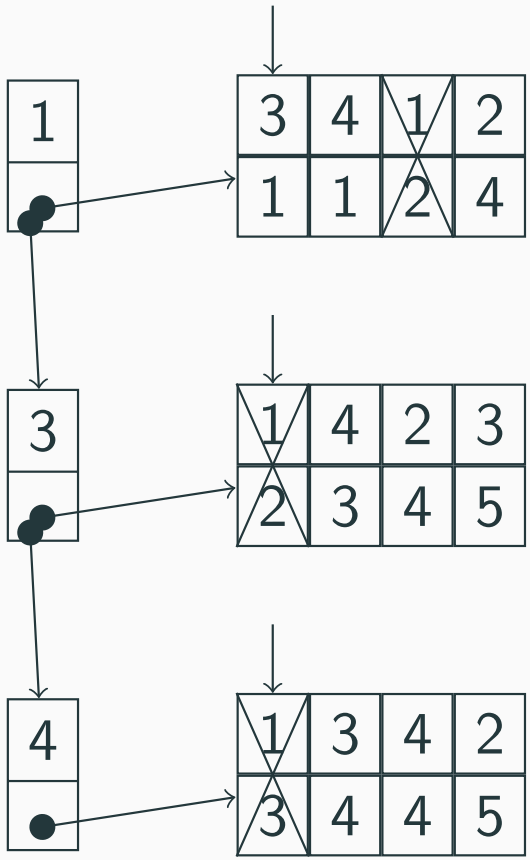
$h =$

Greedy Lower Bound en $\mathcal{O}(n^2)$



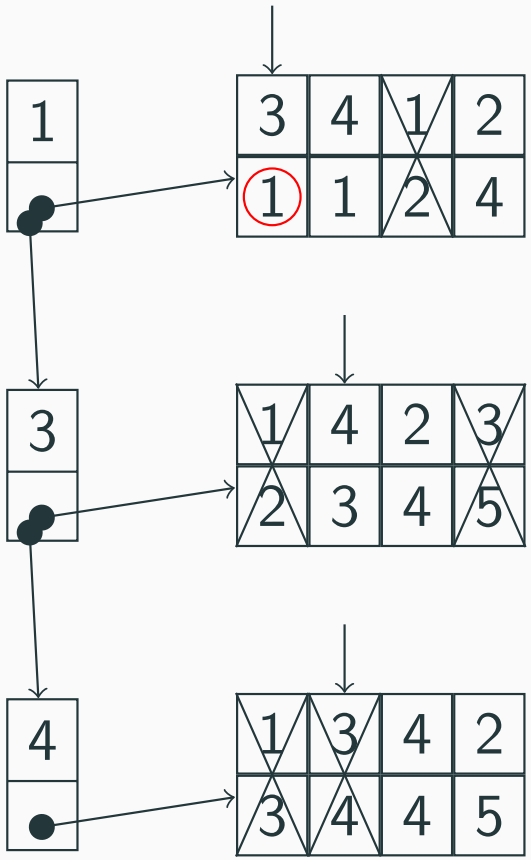
$h = 1+$

Greedy Lower Bound en $\mathcal{O}(n^2)$



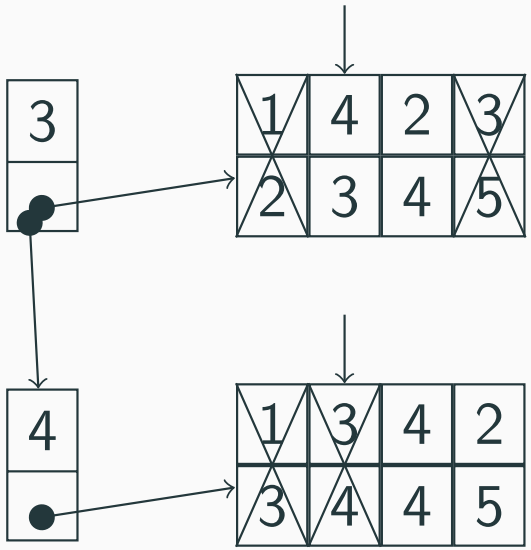
$h = 1+$

Greedy Lower Bound en $\mathcal{O}(n^2)$



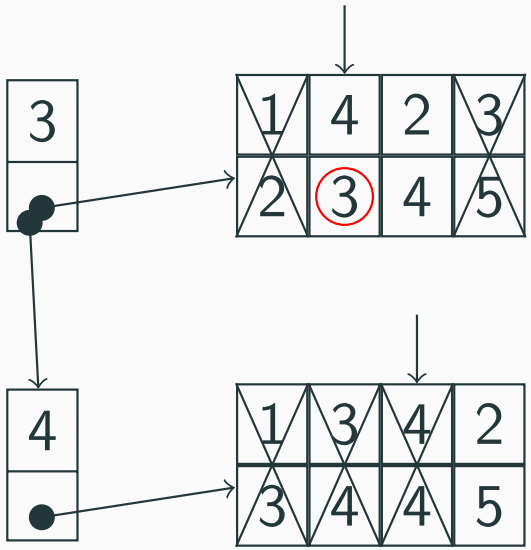
$$h = 1 + 1 +$$

Greedy Lower Bound en $\mathcal{O}(n^2)$



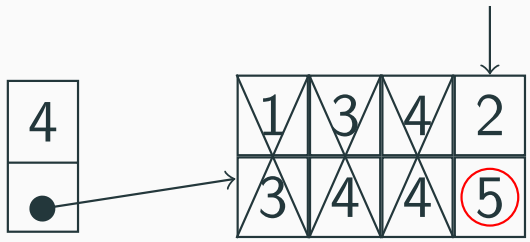
$$h = 1 + 1 +$$

Greedy Lower Bound en $\mathcal{O}(n^2)$



$$h = 1 + 1 + 3 +$$

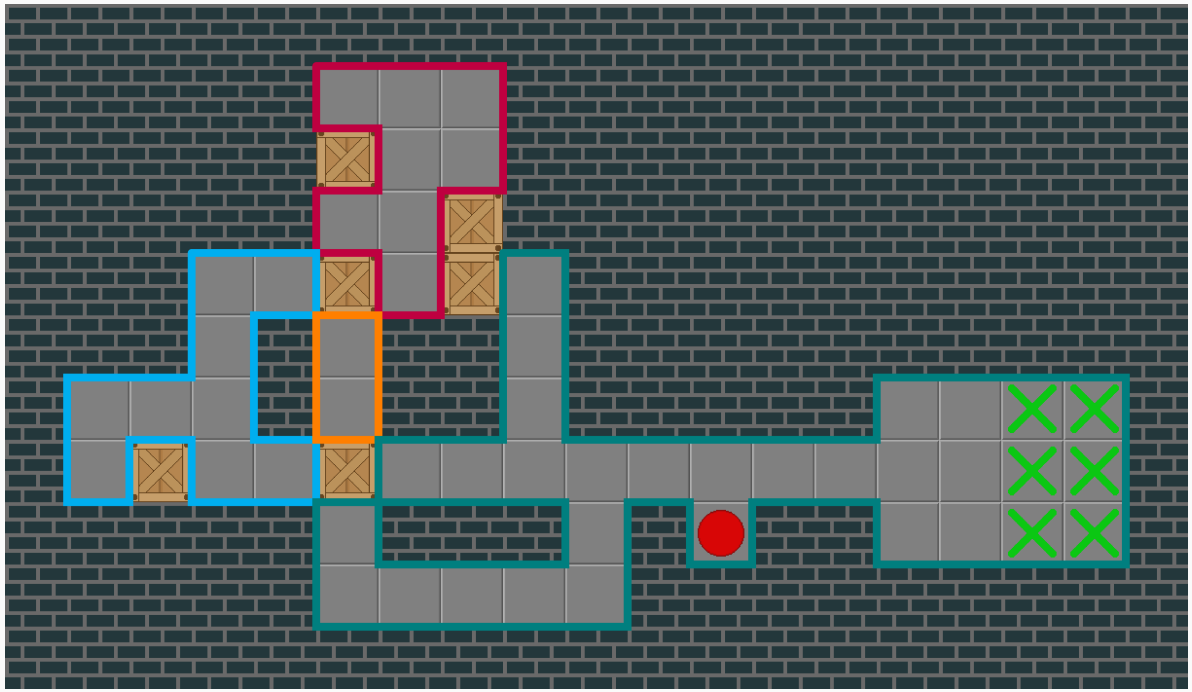
Greedy Lower Bound en $\mathcal{O}(n^2)$



$$h = 1 + 1 + 3 + 5 = 10$$

Calcul des *corrals* en $\mathcal{O}(wh)$

Utilisation de *Union-Find* : partition de $[\![0; wh - 1]\!]$.



Calcul des *corrals* en $\mathcal{O}(wh)$

```
1:  procedure CORRAL(x, y)
2:      if not solid(x,y) then
3:          createSingleton(x, y)
4:      else
5:          if solid(x-1, y) and solid(x,y-1) then
6:              createSingleton(x, y)
7:          else if not solid(x-1, y) and solid(x,y-1) then
8:              addToCorral(x-1,y, x,y)
9:          else if solid(x-1, y) and not solid(x,y-1) then
10:             addToCorral(x,y-1, x,y)
11:         else
12:             addToCorral(x-1,y, x,y)
13:             union(x,y-1, x,y)
14:         end if
15:     end if
```

Résultats

Annexe

Tableau des compléxités