

# Résolution de niveaux du Sokoban

---

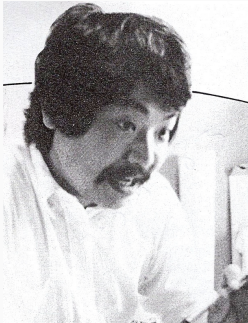
Nom Prénom

Candidat n°01234

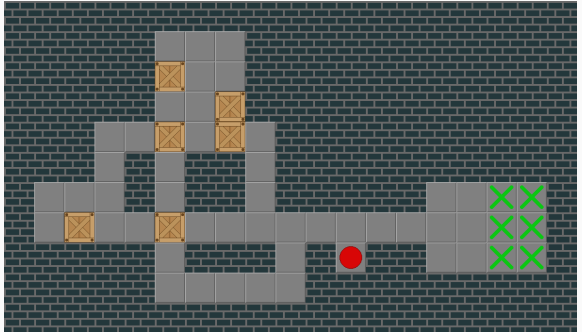
# Introduction

---

# Le jeu du Sokoban



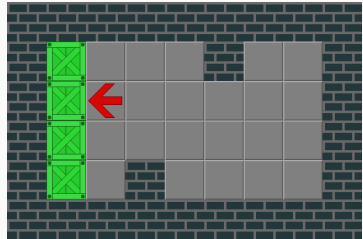
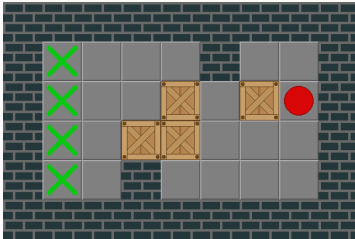
Hiroyuki Imabayashi

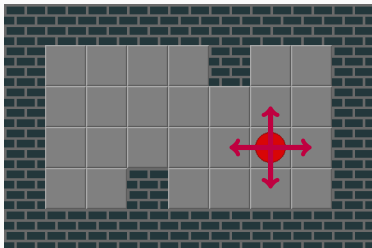


*X Sokoban*

# But du jeu

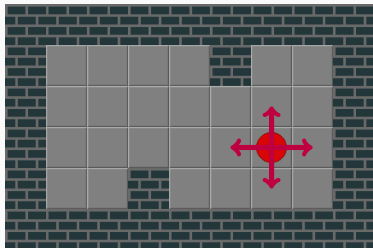
Déplacements



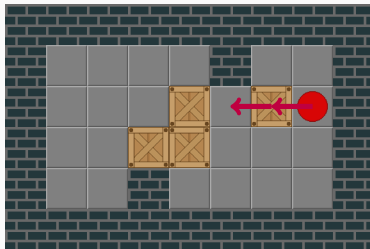


Déplacements autorisés

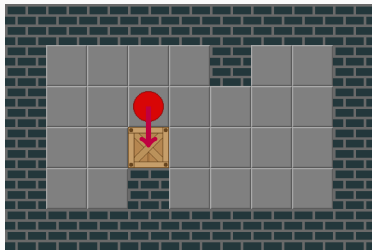
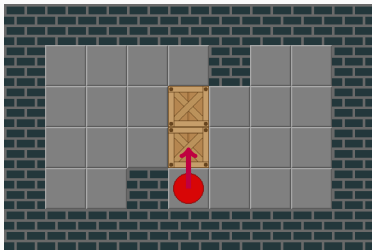
# Règles



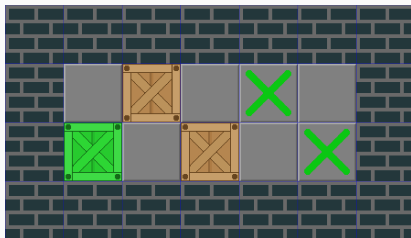
Déplacements autorisés



# Règles



# Tuiles



Mur



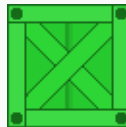
Sol



Caisse



Cible



Caisse sur une cible



**Quelles stratégies adopter pour trouver une solution le plus rapidement possible à un niveau de Sokoban ?**

```
Welcome to sokoshell - Version 1.0  
Type 'help' to show help. More help for a command with 'help command'  
sokoshell> █
```

Introduction

Principe de résolution

Réduction de l'espace de recherche

- Analyse statique

- Analyse dynamique

Recherche dirigée par une heuristique

Optimisations

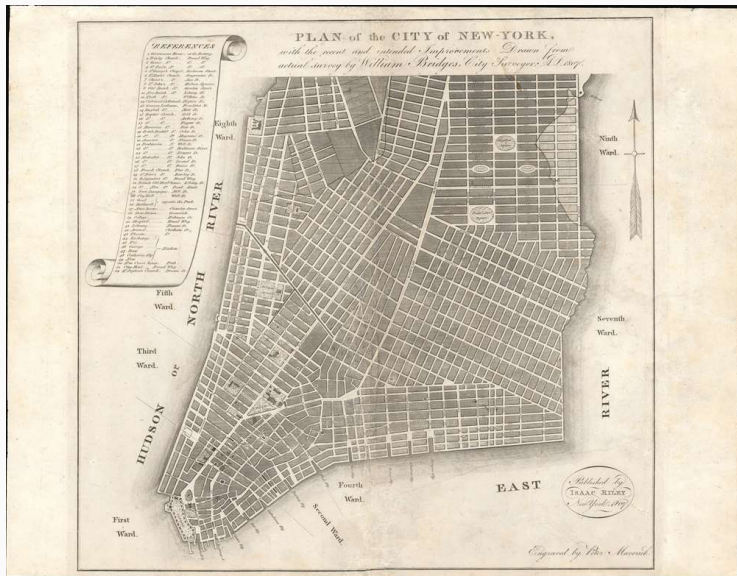
Résultats

Annexe

## Lien avec le thème de l'année



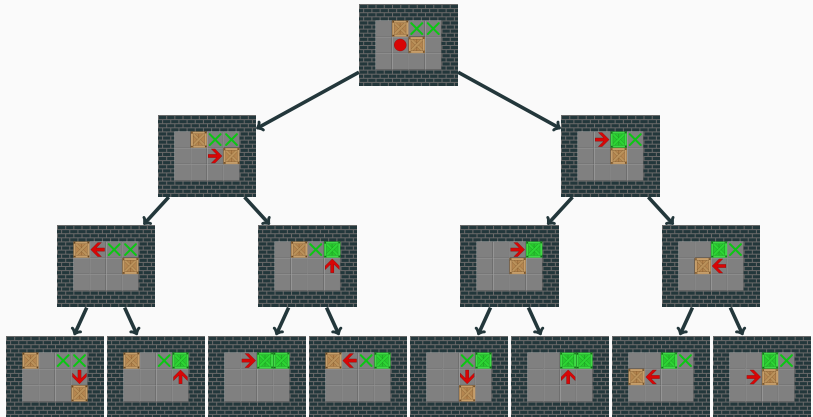
# Lien avec le thème de l'année



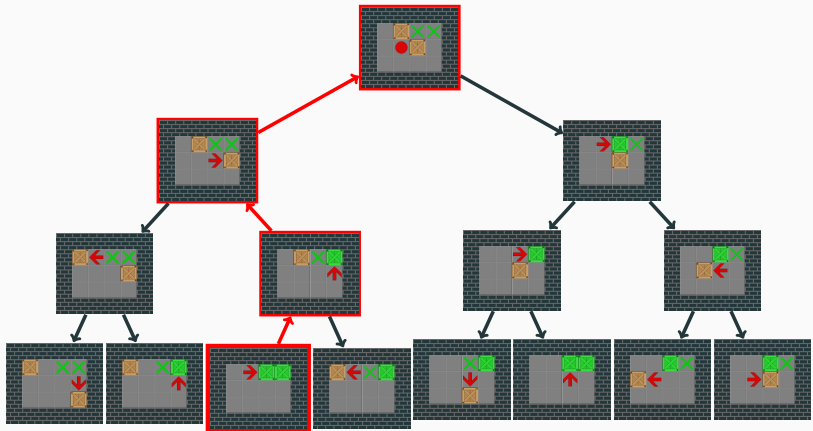
# Principe de résolution

---

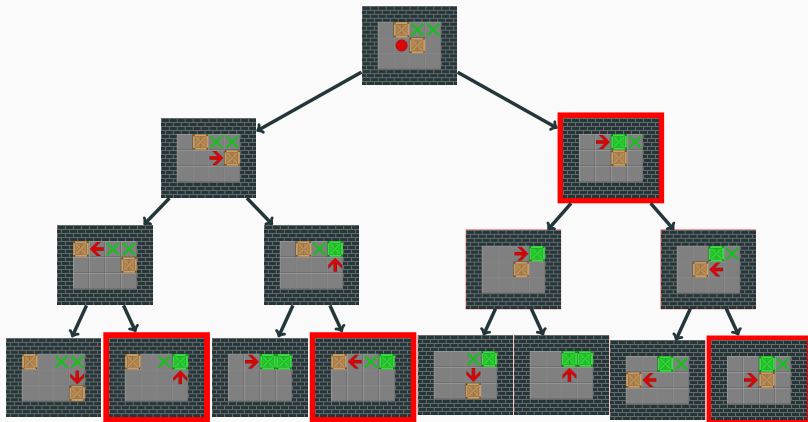
# Arbre des états



# Arbre des états



# Arbre des états





## Calcul du *hash* d'un état - Hash de Zobrist

Propriétés du **XOR** :

1.  $a \mathbf{XOR} a = 0$
2. **XOR** commutatif, associatif
3. **XOR** préserve l'aléatoire

Initialisation :

$$T = \begin{matrix} & \begin{matrix} \text{caisse} & \text{joueur} & \text{case} \end{matrix} \\ \begin{pmatrix} 6357 & 01234 \\ -1378 & 42 \\ \vdots & \vdots \\ 93268 & -278 \end{pmatrix} & \begin{matrix} 0 \\ 1 \\ \vdots \\ wh - 1 \end{matrix} \end{matrix}$$

## Calcul du *hash* d'un état - Hash de Zobrist

- $(c_1, \dots, c_n)$   $n$  caisses et  $p$  position du joueur :

$$h = \mathbf{XOR}_{i=0}^n T[c_i][0] \mathbf{XOR} T[p][1]$$

en  $\mathcal{O}(n)$

- **Connaissant le hash de l'état parent** :  $c_i \rightarrow c'_i, p \rightarrow p'$

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

en  $\mathcal{O}(1)$

# Réduction de l'espace de recherche

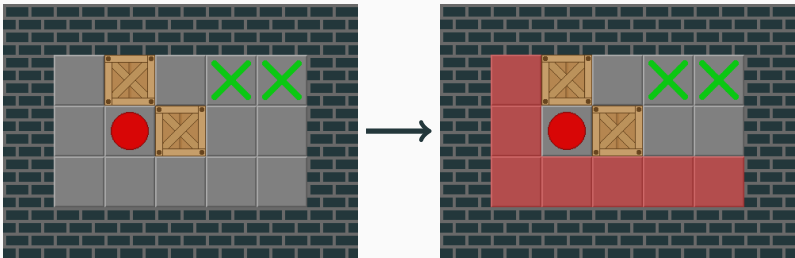
---

# Réduction de l'espace de recherche

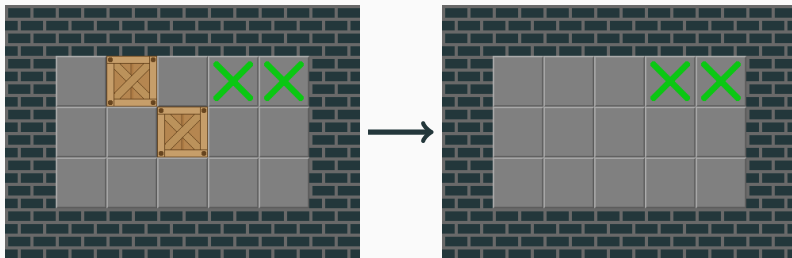


Analyse statique

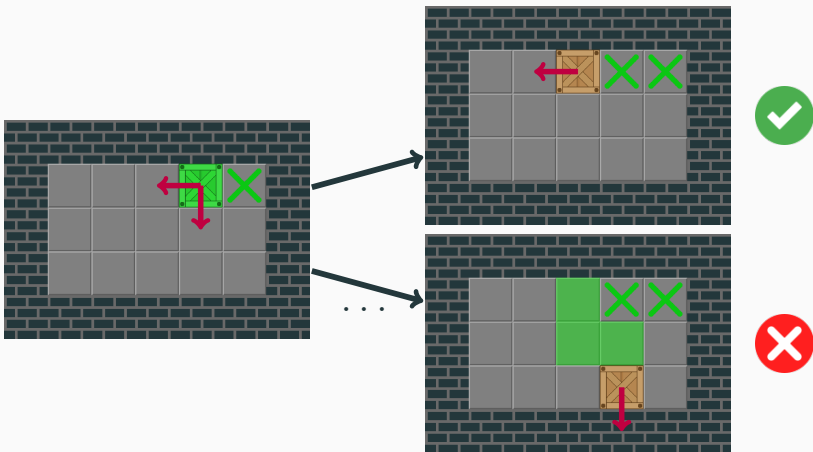
## Détection des positions mortes (*dead positions*)



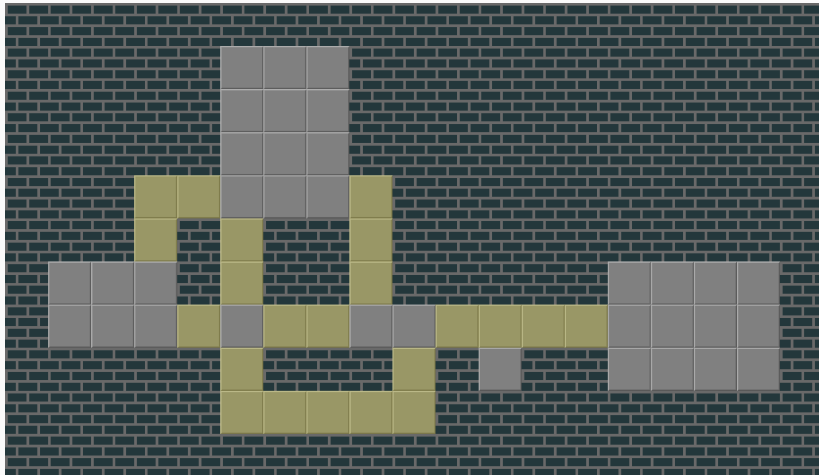
## Détection des positions mortes (*dead positions*)



## Détection des positions mortes (dead positions)

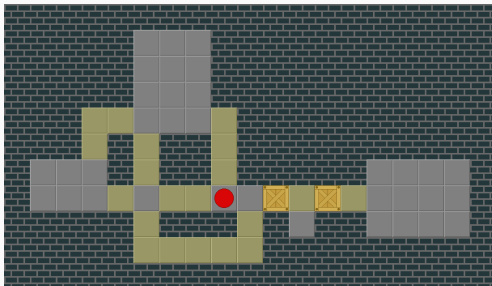
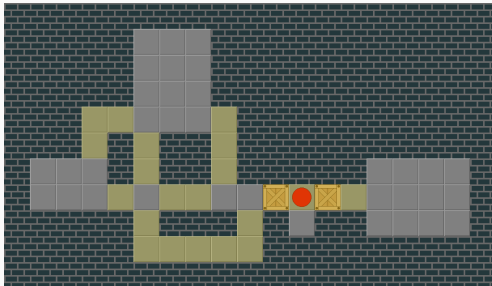


# Détection de tunnels

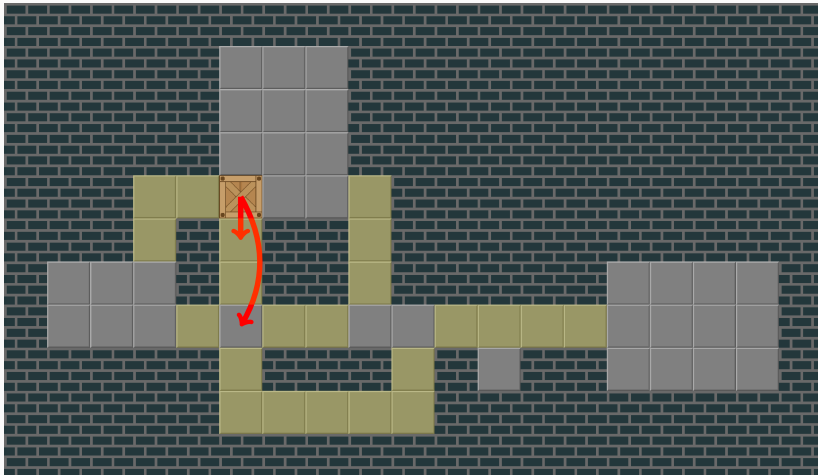




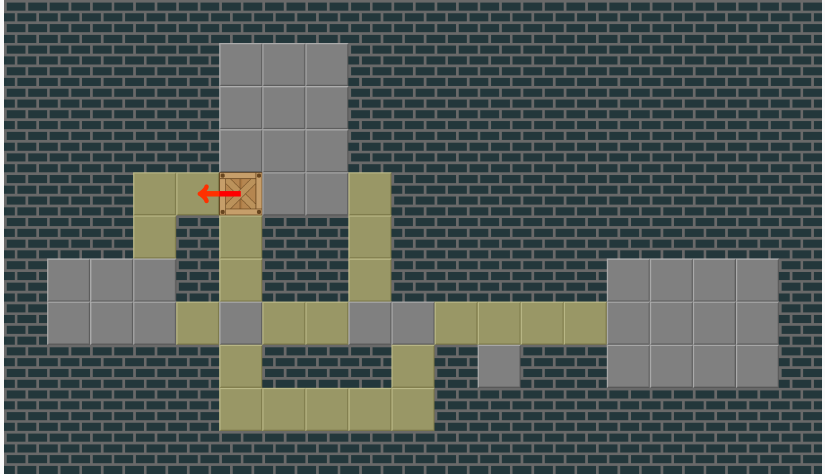
# Détection de tunnels



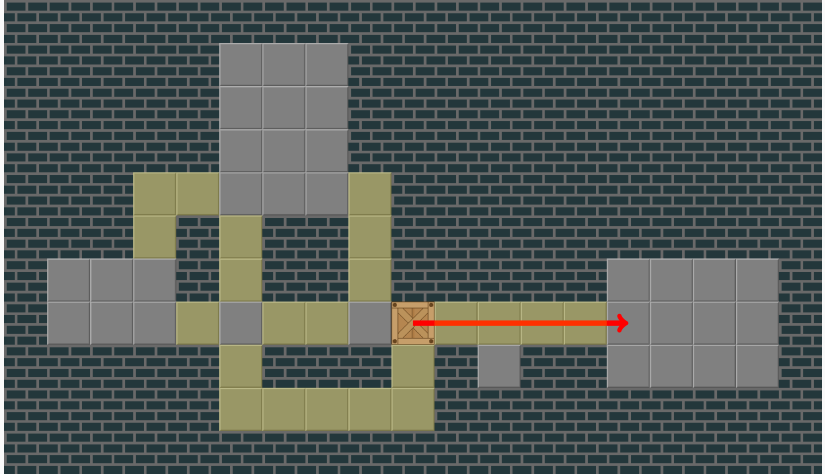
# Détection de tunnels



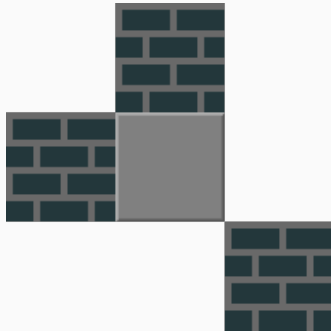
# Détection de tunnels



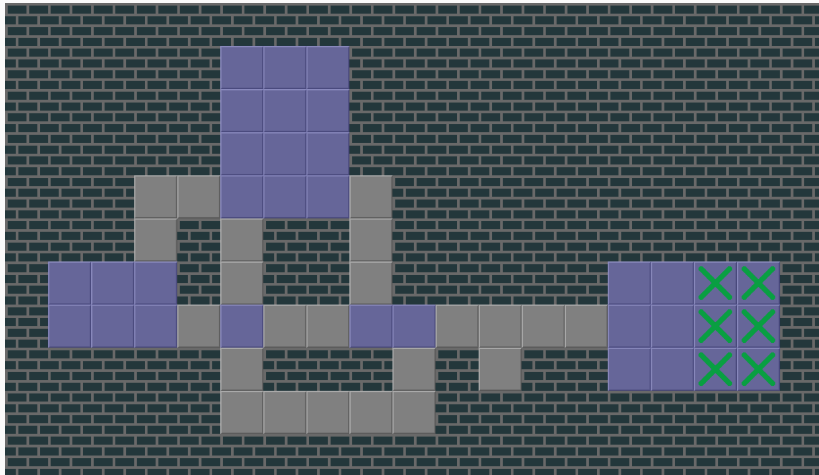
# Détection de tunnels



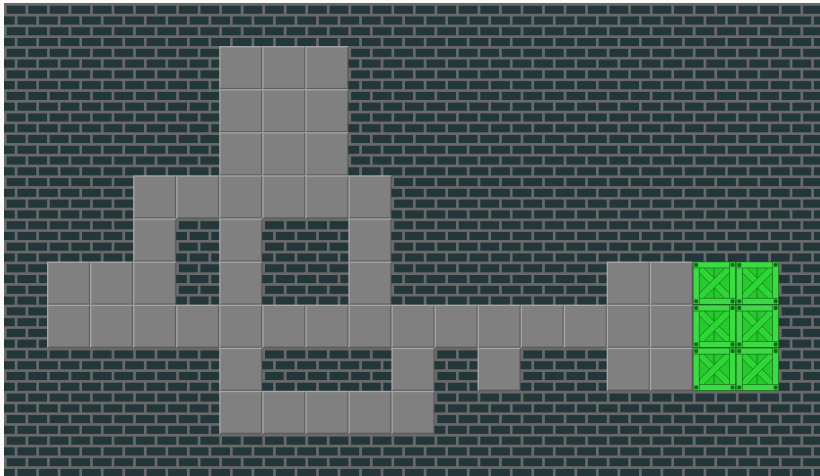
# Détection de tunnels



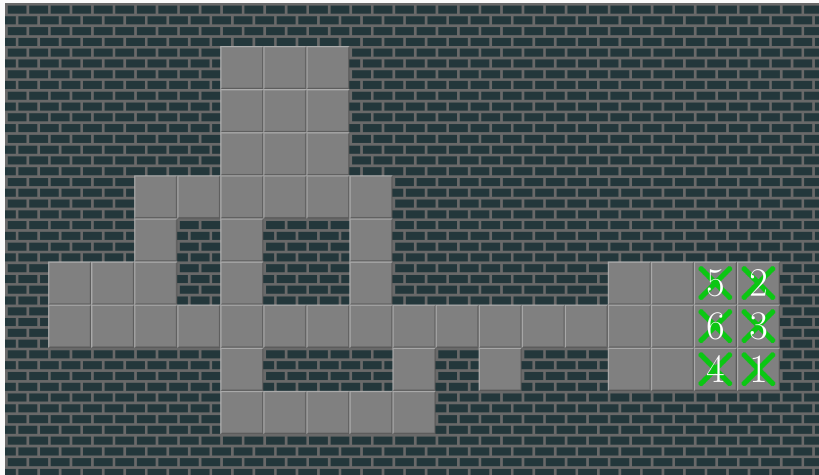
## Salles et ordre de rangement (*packing order*)



## Salles et ordre de rangement (*packing order*)

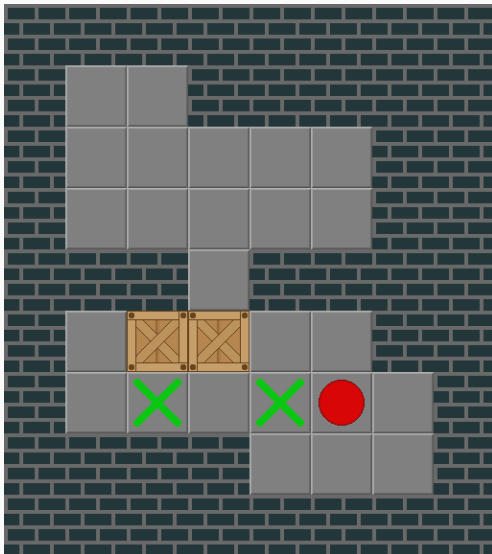


## Salles et ordre de rangement (*packing order*)





## Salles et ordre de rangement (*packing order*)

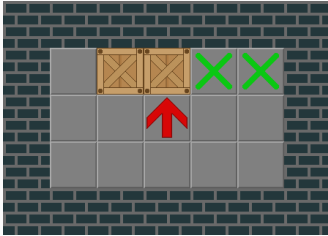


# Réduction de l'espace de recherche

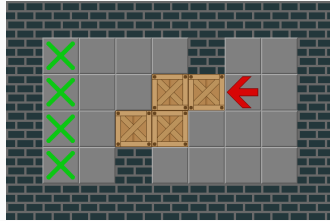


Analyse dynamique

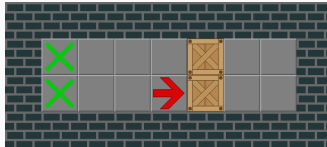
# Détection d'impasses (*deadlocks*)



(a) *Freeze deadlock n°1*



(b) *Freeze deadlock n°2*

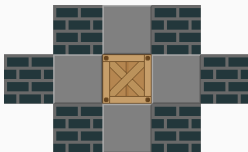


(c) *PI Corral deadlock*

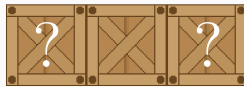
# Détection de *freeze deadlocks*



(a) Règle n°1

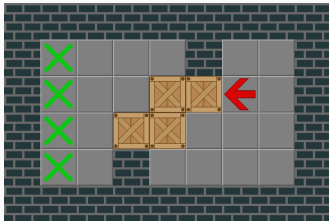


(b) Règle n°2

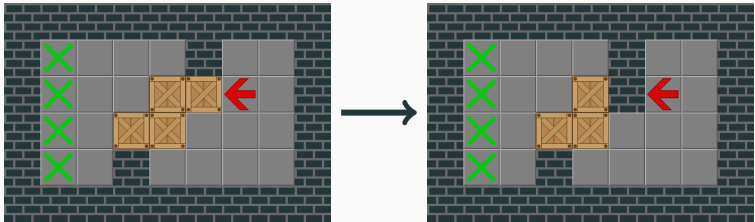


(c) Règle n°3

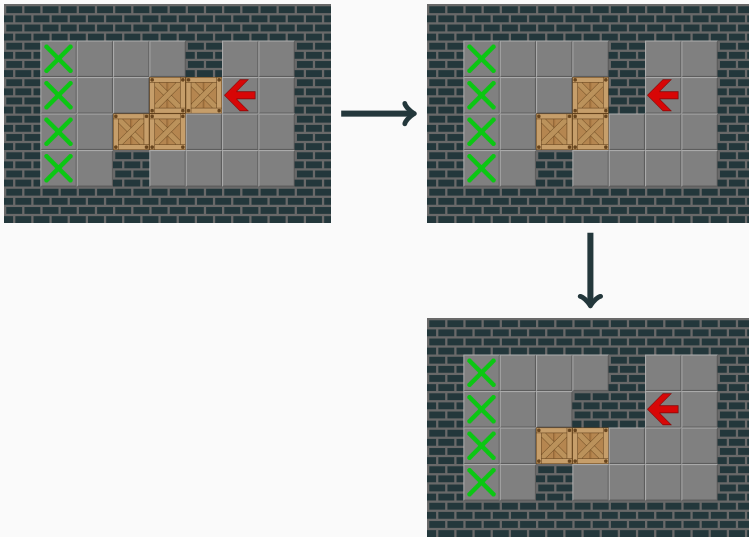
## Détection de *freeze deadlocks*



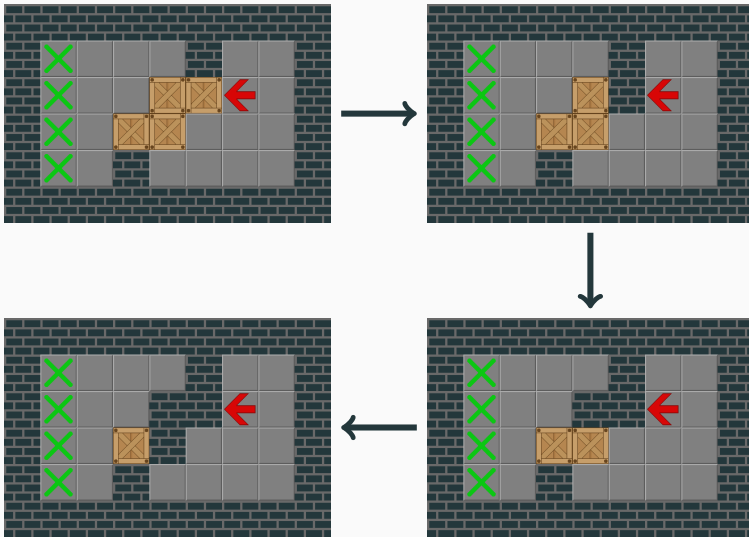
## Détection de *freeze deadlocks*



## Détection de *freeze deadlocks*



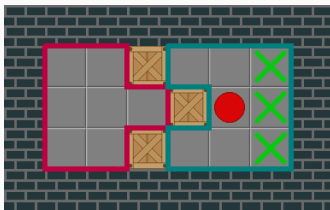
## Détection de *freeze deadlocks*



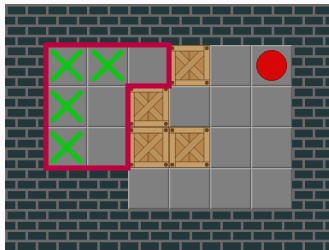
Gelée!



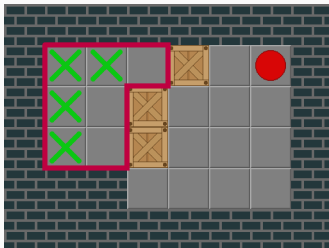
# Détection de *PI Corral* deadlocks



(a) *Corral*

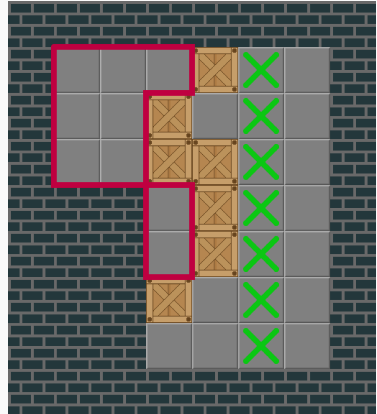
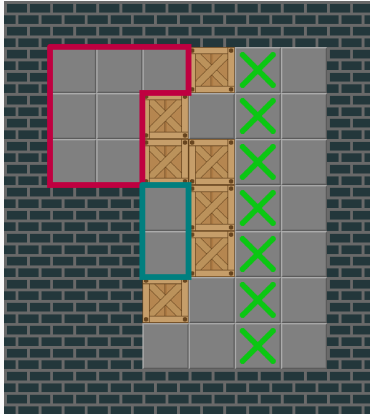


(b) *I Corral*

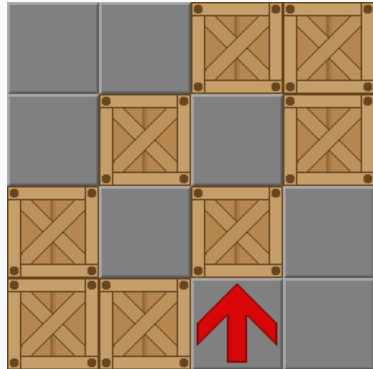
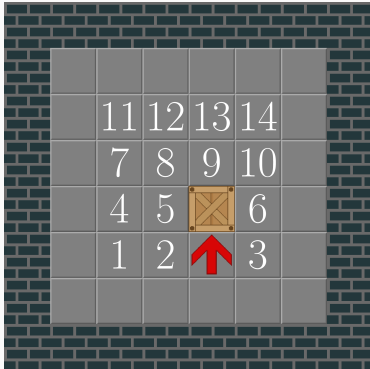


(c) *PI Corral*

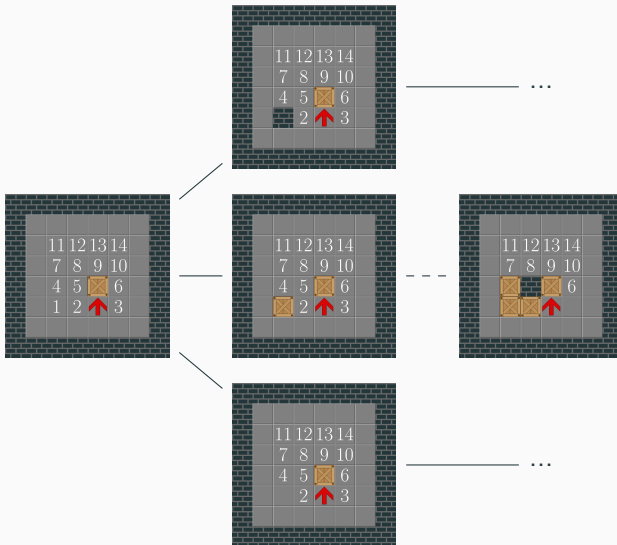
## Détection de *PI Corral* deadlocks



## Table de *deadlocks*



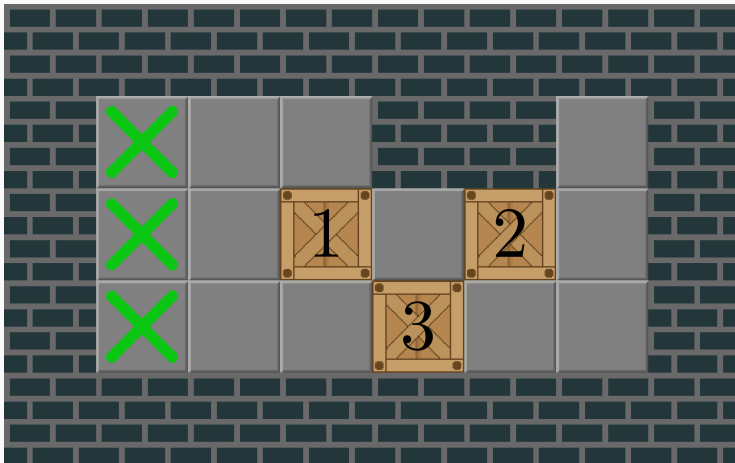
# Table de *deadlocks*



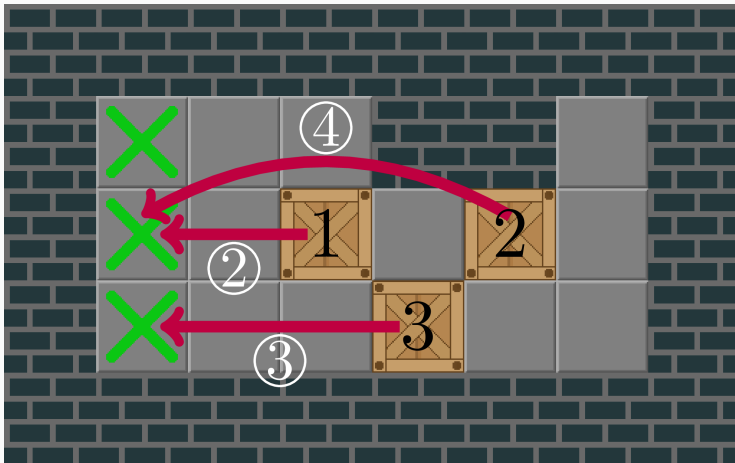
## Recherche dirigée par une heuristique



## Heuristique simple (*Simple Lower Bound*)

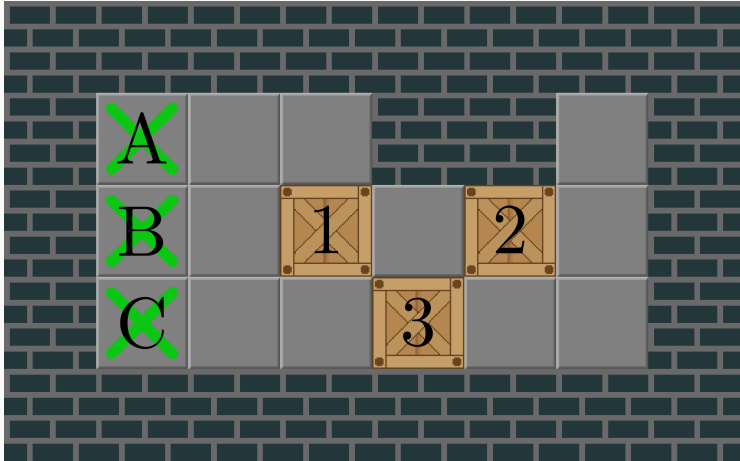


## Heuristique simple (*Simple Lower Bound*)



$$2 + 4 + 3 = 9$$

## Heuristique gloutonne (*Greedy Lower Bound*)





# Heuristique gloutonne (*Greedy Lower Bound*)

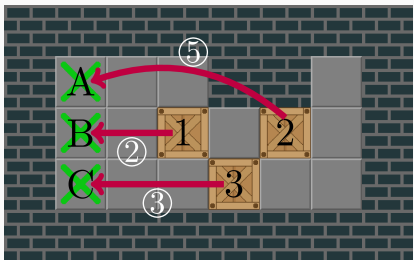


Caisse → Cible	Distance
1 → A	3
1 → B	2
1 → C	3
2 → A	4
2 → B	4
2 → C	5
3 → A	5
3 → B	4
3 → C	3



Caisse → Cible	Distance
<b>1 → B</b>	<b>2</b>
1 → A	3
1 → C	3
<b>3 → C</b>	<b>3</b>
2 → B	4
3 → B	4
2 → A	5
2 → C	5
<b>3 → A</b>	<b>5</b>

# Heuristique gloutonne (*Greedy Lower Bound*)



$$2 + 3 + 5 = 10$$

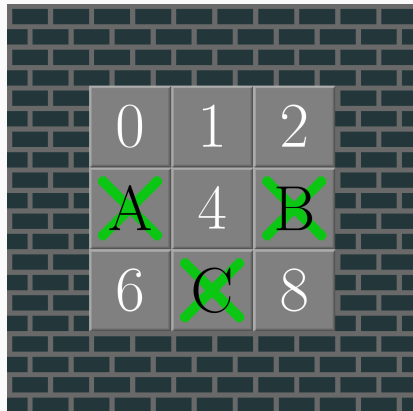
Caisse → Cible	Distance
<b>1 → B</b>	<b>2</b>
1 → A	3
1 → C	3
<b>3 → C</b>	<b>3</b>
2 → B	4
3 → B	4
2 → A	5
2 → C	5
<b>3 → A</b>	<b>5</b>

# Optimisations



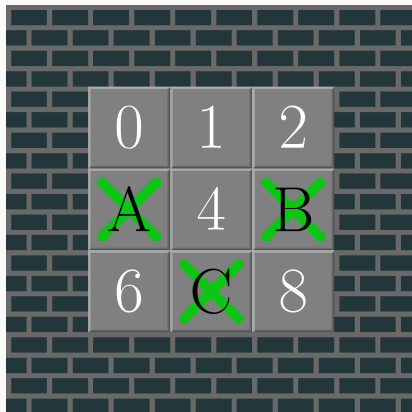
# Précalcul des distances caisses-cibles

Case	Distances		
	A	B	C
0	1	3	3
1	2	2	2
2	3	1	3
3	0	2	2
4	1	1	1
5	2	0	2
6	1	3	1
7	2	2	0
8	3	1	1

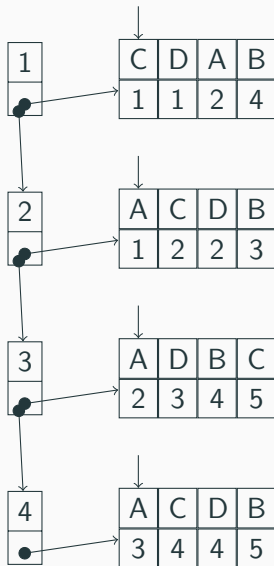


# Précalcul des distances caisses-cibles

Case	Distances triées		
0	A : 1	B : 3	C : 3
1	A : 2	B : 2	C : 2
2	B : 1	A : 3	C : 3
3	A : 0	B : 2	C : 2
4	A : 1	B : 1	C : 1
5	B : 0	A : 2	C : 2
6	A : 1	C : 1	B : 3
7	C : 0	A : 2	B : 2
8	B : 1	C : 1	A : 3

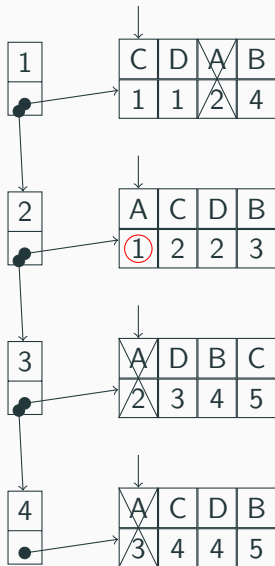


## Greedy Lower Bound en $\mathcal{O}(n^2)$



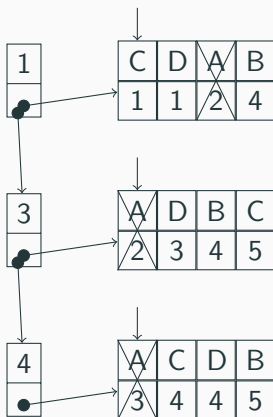
$h =$

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



$$h = 1 +$$

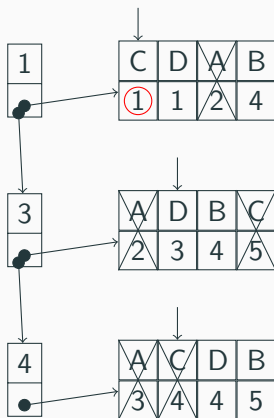
## Greedy Lower Bound en $\mathcal{O}(n^2)$



$$h = 1 +$$

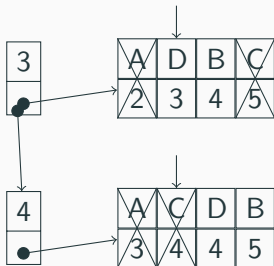


## Greedy Lower Bound en $\mathcal{O}(n^2)$



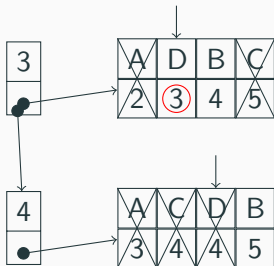
$$h = 1 + 1 +$$

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



$$h = 1 + 1 +$$

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



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

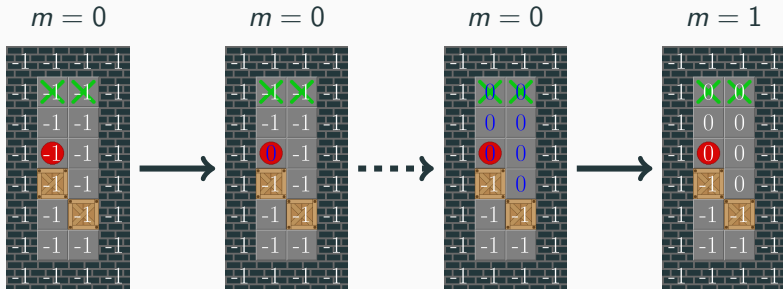
## Greedy Lower Bound en $\mathcal{O}(n^2)$



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

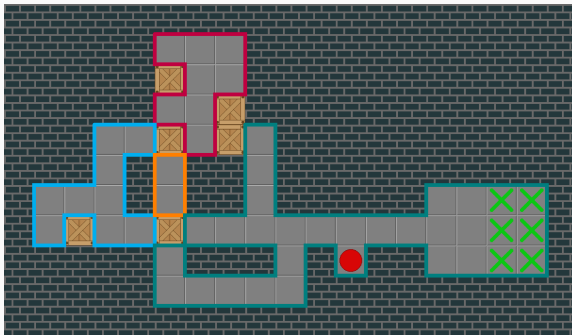
# Parcours de graphes : démarquer tous les nœuds en $\mathcal{O}(1)$

nœud marqué ssi valeur =  $m$



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

Utilisation de *Union-Find* : partition de  $\llbracket 0; wh - 1 \rrbracket$ .



## 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
16: end procedure
```

## Résultats



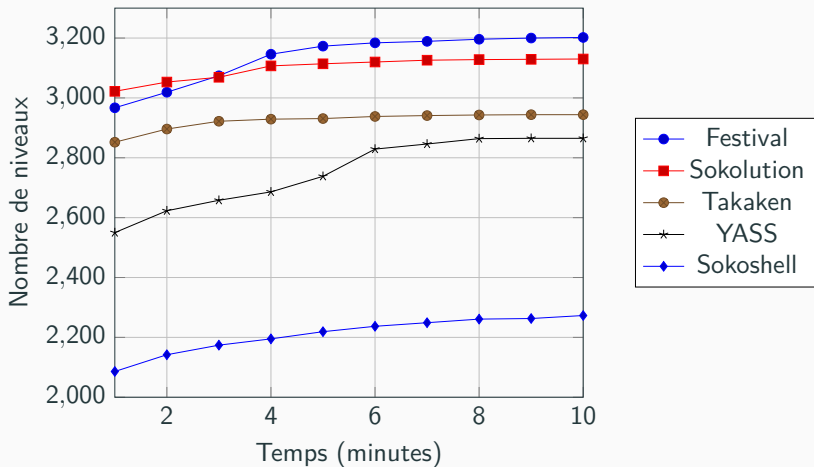


## Nombre de niveaux résolus

Limite de temps : 10 min. Limite de RAM : 32 Gb.

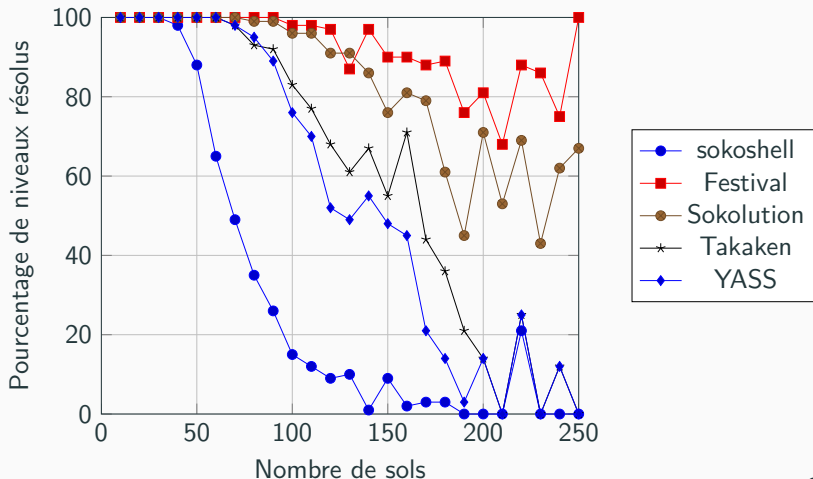
Ensemble de niveaux	XSokoban	<i>Large test suite</i>
Nombre de niveaux	90	3272
A*	11	2204
fess0	15	2273
Festival (Yaron Shoham)	90	3202
Sokolution (Florent Diedler)	90	3130
Takaken (Ken'ichiro Takahashi)	90	2944
YASS (Brian Damgaard)	89	2865

Nombre de niveaux résolus (cumulés) en fonction du temps



## Temps moyen passé par niveaux

Solveur	A*	fess0	Festival	Sokolution	Takaken	YASS
Temps moyen	3min 28s	3min 16s	3s	2s	7s	24s



## Annexe



# Tableau des complexités