

Projet Théorie des graphes

Cas du SOKOBAN

Sommaire

Présentation	2
Règles du jeu	3
Cahier des charges.....	4
Jeu de base	4
Niveau 1 - Débutant	5
Niveau 2 – Intermédiaire.....	8
Niveau 3 – Avancé	9
Versionning de votre projet : GIT + Bitbucket.....	10
Travail demandé, contraintes et consignes.....	11

Présentation

Sokoban est un jeu vidéo de puzzle inventé au Japon. Ce nom que l'on écrit 倉庫番 en japonais, ou *sōkoban* transcrit avec la méthode Kunrei, désigne un garde d'entrepôt.

Le jeu original a été écrit par Hiroyuki Imabayashi et comportait 50 niveaux. Il remporte en 1980 un concours de jeu vidéo pour ordinateur. Plus tard Hiroyuki Imabayashi est devenu président de la compagnie japonaise Thinking Rabbit Inc. qui détient aujourd'hui les droits sur le jeu depuis 1982.

Aujourd'hui, il existe de multiples jeux dérivés de ce jeu, par exemple *Boxworld*, une variante fonctionnant sous Windows et incluant 100 niveaux. Microsoft propose ce jeu sous le nom de « Pousse Bloc ». Comme les règles sont simples, le jeu est facile à programmer. Plusieurs versions ont été écrites en JavaScript ; il est ainsi possible de jouer en ligne avec un navigateur web. Il existe des logiciels proposant un affichage 3D (le principe du jeu reste en 2D, comme certains jeux d'échecs 3D).

Source : Wikipedia

Sokoban est un jeu très intéressant à étudier car il fait partie des problèmes de décision les plus complexes à résoudre par une machine (Intelligence Artificielle). En effet, il a été démontré que Sokoban appartient à la catégorie des problèmes de type NP-Complet comme le jeu de Go, c'est-à-dire :


- L'arbre de recherche croît exponentiellement avec un facteur de branchement assez important (contrainte mémoire forte)
- Le temps de résolution est très long (contrainte processeur forte)

A titre d'exemple, la taille de l'arbre de recherche pour une grille de 20x20 (taille modérée) est de l'ordre de 10^{98} (le jeu de GO pour une grille de 19x19 est de l'ordre de 10^{177}).

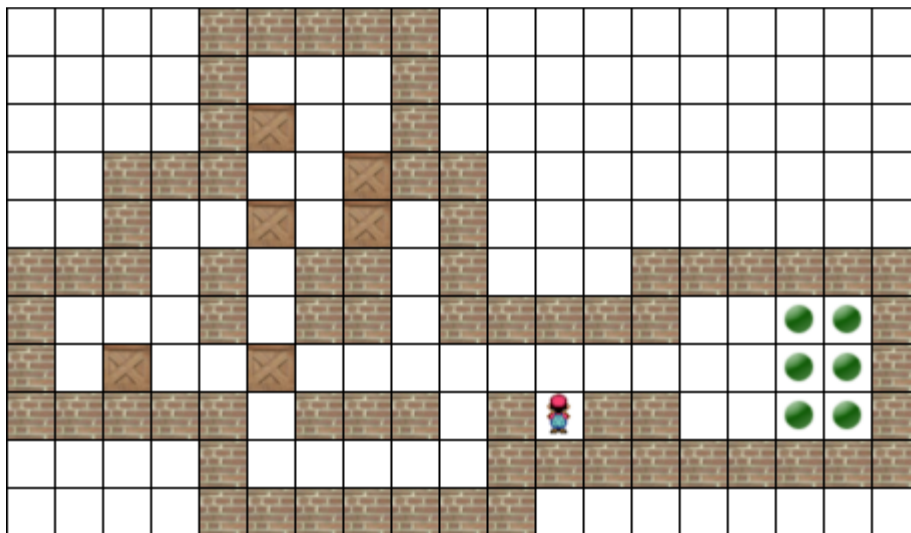
Règles du jeu

Sokoban est peut être un jeu difficile à résoudre pour une machine, pourtant les règles du jeu sont d'une simplicité enfantine.

Un niveau se présente sous la forme d'un labyrinthe contenant des éléments. Le labyrinthe sera modélisé par une matrice 2D d'éléments. Voici la liste des différents éléments :

Élément	Sprite (Image)
MUR	
SOL	Aucun
CAISSE	
CAISSE PLACÉE	
OBJECTIF (Goal)	
Personnage (une image par direction)	

Le joueur peut se déplacer sur le sol et les goals mais ne peut pas traverser les murs. De même les caisses se bloquent au niveau des murs.

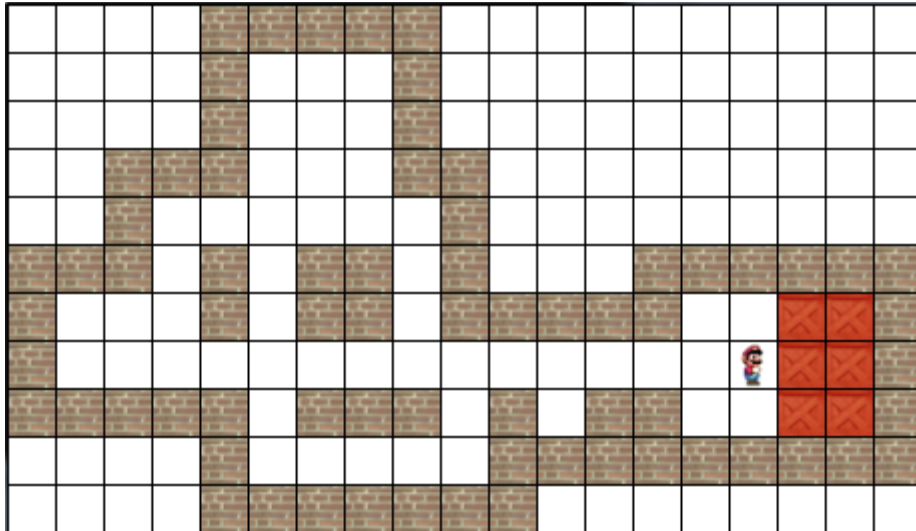


Niveau 1 du package « XSokoban »

Maintenant, voici les règles du jeu que vous devrez respecter à la lettre :

1. Le joueur ne peut que **pousser** les caisses (il est donc INTERDIT de tirer une caisse)
2. Le joueur ne peut pousser **qu'une seule caisse** à la fois
3. Le joueur ne peut se déplacer que dans les quatre directions (HAUT/BAS/GAUCHE/DROITE)
4. Pour gagner, toutes les caisses doivent être poussées sur un goal

Le niveau est résolu lorsque toutes les caisses sont situées sur un goal (peu importe l'ordre).



Niveau 1 terminé après 97 coups

Vous voyez que les règles sont très simples et le jeu est relativement facile à jouer pour un humain. Pourtant, c'est une autre histoire pour l'ordinateur...






Cahier des charges

Cette année, il y a une nouveauté : le cahier des charges est rédigé par niveau. Vous devez suivre l'ordre exacte des questions afin de progresser efficacement. Lisez plusieurs fois l'énoncé, au premier abord ce n'est pas facile mais en fait on se rend compte que les algorithmes demandés ne sont pas si difficiles quand on comprend ce que l'on fait :)

Jeu de base

Implémenter le jeu de base en suivant les instructions suivantes :

- Chaque niveau sera représenté par une matrice 2D en mémoire (utiliser le *vector* !)
- Les niveaux seront stockés dans des fichiers texte qui seront fournis. Voici un tableau montrant la corrélation entre les chiffres et les éléments du décor :

Identifiant	Élément du décor
0	SOL
1	
2	
3	
4	
5	

A noter que l'identifiant 6 sera utilisé pour symboliser le personnage sur un goal

- Le déplacement du personnage se fera case par case (gérer les collisions)
- Quand on termine un niveau, c'est à dire que toutes les caisses sont placées sur un goal, on charge le niveau suivant.

Pour vous aider, voici un exemple possible de classe pour gérer le jeu de base en entier : Soit une classe *Maze* contenant :

- un attribut pour gérer le terrain (matrice 2D)
- un attribut pour savoir si le niveau est résolu ou non
- un attribut qui gère la position du personnage
- un attribut qui gère la position de toutes les caisses
- un attribut qui gère la position de tous les goals (à noter que les goals sont fixes)
- une méthode qui charge un niveau à partir d'un fichier texte
- une méthode qui vérifie si le niveau est résolu
- une méthode qui déplace le joueur et pousse les caisses
- une méthode qui dessine à l'écran le niveau

Et c'est tout !

N'oubliez pas de tester que vos niveaux sont jouables à l'aide du clavier.

Ne passer pas trop de temps sur cette partie (qui n'a rien à voir avec la Théorie des Graphes !). Une seule classe suffit par exemple pour gérer le jeu en entier. Inutile aussi de faire de l'héritage et du polymorphisme sauf si vous savez ce que vous faites :)

Niveau 1 - Débutant

Voici le premier niveau de difficulté proposé. Ce niveau est accessible pour tout le monde.

Un peu de vocabulaire qui sera valable pour l'ensemble du projet :

- **Noeud** : Correspond à un état donné du jeu à un instant t . Dans le cas du Sokoban, il s'agit de la position des caisses et du joueur.
- **Noeud solution** : Noeud où toutes les caisses sont placées sur un goal.
- **Transition** : Action permettant de passer d'un état à un autre. Dans le cas du Sokoban, il s'agit d'une poussée d'une caisse. Le déplacement du personnage ne sera pas pris en compte. Cela permettra d'améliorer l'efficacité des algorithmes par la suite.
- **Graphe** : Ensemble de noeuds reliés par des arcs. Un arc représente une transition d'un état à un autre.
- **Arbre de recherche** : Arbre qui comprend l'ensemble des noeuds possibles pour un niveau. Son ordre de grandeur varie mais est de l'ordre de 10^{98} pour le Sokoban !!

- **Coût** : Quantité permettant de mesurer la valeur d'un noeud. Dans le cas du Sokoban, l'unité est le déplacement d'une caisse. (le déplacement du joueur aura un coût nul puisqu'il n'est pas pris en compte dans les transitions). Un noeud ayant un coût faible sera considéré comme plus prometteur qu'un noeud ayant un coût fort. Le coût représente en fait le nombre de poussées restantes (estimation) pour résoudre un niveau.
- **Solution optimale** : Chemin minimum parcouru dans le graphe à partir du noeud initial vers un noeud solution. En d'autres termes, il s'agit du nombre de poussées minimal qu'il faut pour amener toutes les caisses sur un goal et donc résoudre le niveau.

Attention : L'ordre de visite des successeurs se fera exclusivement dans cet ordre : HAUT/BAS/GAUCHE puis DROITE

Travail à réaliser :

1. Implémenter un premier algorithme naïf permettant de résoudre un niveau par brute force. Pour cela, on testera tous les déplacements du personnage (un déplacement peut entraîner la poussée d'une caisse) jusqu'à trouver une séquence qui résout le niveau.
2. Implémenter un sous programme (ou plusieurs) permettant de visualiser votre solution (le personnage se déplace et bouge les caisses tout seul). Cela doit fonctionner pour tous les algorithmes demandés (brute force, BFS, DFS...). Un exemple sera fourni en vidéo.
3. Implémenter un algorithme permettant de résoudre un niveau avec un BFS (parcourt en largeur) puis un DFS (parcourt en profondeur)
4. Implémenter un algorithme permettant de détecter les cases mortes (deadlock en anglais). Une caisse poussée sur l'une de ces cases est forcément bloquée et donc le niveau devient insolvable. Pour la suite du projet, ces cases mortes seront affichées en ROUGE lorsqu'on charge le niveau.

Comparer les 3 algorithmes précédents, avec et sans la détection des deadlocks, selon les critères suivants :

- Vitesse de résolution
- Nombre de nœuds explorés
- Charge mémoire (quantité de mémoire nécessaire pour résoudre le niveau)
- ...

Quel(s) algorithme(s) donne(nt) une solution optimale (plus court chemin) ? Expliquez !

Quel sont les impacts sur l'arbre de recherche quand on active l'algorithme de détection des deadlocks ?

N'hésiter pas à faire un tableau récapitulatif avec des tests et captures d'écran à l'appui. Faites attention à faire tous vos tests sur la même machine afin qu'ils soient représentatifs.

Niveau 2 – Intermédiaire

Vous verrez que les algorithmes précédents sont loin d'être performants pour résoudre les niveaux. En effet, l'arbre de recherche étant gigantesque, il faut trouver un moyen de juger si le prochain nœud à explorer est bon ou pas. Le fait de dire que tel ou tel nœud est « mieux qu'un autre » s'appelle une **heuristique**. Une heuristique est une fonction mathématiques qui calcul un coût pour un nœud donné.

Soit $h(n)$ le coût du nœud n et $h'(n)$ le coût exact du nœud n . Si $h(n) = h'(n)$ on dit que l'heuristique est parfaite et dans ce cas, on converge directement vers un noeud solution sans faire de détour dans l'arbre de recherche. Evidemment, trouver une heuristique parfaite est impossible. Le but sera de trouver la meilleure approximation possible afin de converger le plus vite sans « se perdre » dans l'arbre de recherche.

1. *Proposer une heuristique (naïve ou non) dans le cas du Sokoban pour évaluer un noeud.*
2. *Implémenter un algorithme basé sur le « Best First Search ». Cet algorithme est presque identique au BFS (Breadth First Search vu en cours) sauf qu'il utilise une heuristique pour choisir le prochain noeud à explorer. Faites une recherche Google si vous souhaitez en savoir plus. Utilisez donc l'heuristique de la question précédente.*
3. *Implémenter un algorithme basé sur « l'Astar ». Cet algorithme utilise l'heuristique suivante $f(n) = h(n) + g(n)$ avec $h(n)$ l'heuristique de la question 1 et $g(n)$ qui retourne la profondeur du noeud dans l'arbre.*

Comparer à nouveau ces algorithmes et conclure.

Est ce que ces deux nouveaux algorithmes sont optimaux (plus court chemin) ? Justifiez !

Niveau 3 – Avancé

Ce niveau doit être abordé uniquement pour les meilleurs et ceux qui veulent prétendre au 19 ou 20. Vous devez au préalable avoir fait les niveaux 1 et 2 avant de vous attaquer au niveau 3.

Vous verrez que les algorithmes précédents ne sont toujours pas suffisants. Le problème principal est qu'on parcourt un trop grand nombre de nœuds inutiles et du coup, on perd du temps et de la mémoire. Il faut donc trouver un moyen d'éviter de « perdre son temps » dans l'arbre de recherche.

Prenez un niveau au hasard et essayer de détecter de nouvelles positions dites mortes (*deadlock*) en jouant au clavier. Pour rappel, ces positions ne peuvent pas mener à une solution...

1. *Proposer un algorithme dynamique de détection des deadlocks. Un algorithme dynamique est un algorithme qui s'exécute pendant la résolution d'un problème. Exemple: 4 caisses qui forment un carré est un deadlock excepté si toutes ces caisses sont sur un goal...*
2. *Avec votre algorithme « Best First Search », en ajoutant votre algorithme de la question précédente, comparer le temps de résolution des niveaux à l'aide d'un tableau*
3. *Vous avez peut être remarqué que certains niveaux possède des tunnels. En utilisant cette remarque, essayez de créer des macro-mouvements permettant de déplacer une caisse de l'entrée du tunnel à la sortie du tunnel en un seul coup ! Appliquez ce nouveau concept à votre algorithme « Best First Search ».*
4. *Vous avez peut être remarqué que les goals sont le plus souvent situés dans une zone fermée comportant une seule entrée. En utilisant cette remarque, essayez d'améliorer votre algorithme « Best First Search ».*
5. *En utilisant le principe suivant : « si une caisse peut être poussée directement vers un goal alors elle doit être poussée immédiatement vers ce goal », essayez d'améliorer encore votre algorithme « Best First Search ».*

En combinant les algorithmes 4 et 5, vous réduirez énormément la taille de l'arbre de recherche permettant ainsi de résoudre de très nombreux niveaux...

Vous pouvez aussi modifier votre heuristique de la partie précédente afin d'en trouver une plus puissante...

Allez vous réussir enfin à résoudre tous les niveaux ? Mais le jeu en vaut la chandelle... Bon courage !

Versionning de votre projet : GIT + Bitbucket

Cette année, vous devez obligatoirement utiliser l'outil de versionning qui sera mis à votre disposition sur Campus.

Vous utiliserez GIT avec Bitbucket pour gérer votre projet. Une formation vous sera dispensée pour apprendre à utiliser ces outils.

Lors de la soutenance, vous devrez montrer les différentes phases de développement de votre projet (les "révisions" du projet). Vous devez aussi être en mesure de montrer "qui a fait quoi" dans le projet.

Grâce au versionning, vous n'aurez plus de problèmes du type :

- c'est mon camarade qui a tout le projet, je n'ai pas pu corriger les erreurs...
- mon disque dur est mort la veille de la soutenance...
- on m'a volé mon ordinateur le jour de la soutenance...
- j'ai renversé du liquide sur mon clavier...
- je comprends pas, mon code a été écrasé ? et je n'ai pas fait de backups...
- ...

Et donc plus d'excuses le jour de la soutenance :)

Travail demandé, contraintes et consignes

Vous devez réaliser l'ensemble du cahier des charges de base (expliqué dans la paragraphe précédent) ainsi que les niveaux 1 et 2. Le langage de programmation est imposé : **C++ en POO** obligatoire... (il est interdit de code en "C" dans un projet C++) et pour la partie graphique vous utiliserez **Allegro 4.2 ou 4.4**. Votre code devra être modulaire, respecter les interfaces et bien commenté !

Cette partie sera notée sur 18 points.

L'utilisation correcte du versionning sera notée sur 2 points.

Un code qui ne compile pas ou qui plante au démarrage ne vaut pas plus de 10/20. Tester donc votre programme avant de le déposer sur Campus...

Pour vous aider, le Webservice CodeAnalysis sera disponible sur le lien suivant :

<http://codeanalysis.fr/webService/>

Tutoriel d'utilisation de CodeAnalysis :

<http://campus.ece.fr/mod/resource/view.php?id=17363>

Votre travail sera jugé sur les critères suivants :

- Le respect rigoureux des règles du jeu énoncées précédemment (CDC)
- La modularité de votre conception et donc de votre code
- La bonne répartition des tâches entre les membres de l'équipe
- ...

Le rendu se fera exclusivement via Campus avant la date limite (la deadline sera écrite sur la page Campus). Tout retard sera pénalisé par **2 points** par jour de retard.

Un rapport sera aussi demandé (maximum 20 pages). Il sera noté sur 20 et devra comprendre :

- présentation rapide du projet (pas de copier/coller du sujet !)
- les choix de programmation (structure de données, conteneurs de la STL...)
- un diagramme de classe général du solveur de votre programme
- les algorithmes de théorie des graphes et scénarii illustrés de leurs résultats
- les réponses aux questions du sujet (avec tableaux comparatifs)
- la répartition des tâches au sein de l'équipe
- un graphe de Pert avec les dates au plus tôt, dates au plus tard, marges et chemin critique
- les difficultés rencontrées
- un bilan collectif
- un bilan individuel
- les citations de vos sources !