université de Caen
Basse-Normandie

TPA
Rapport de Projet

Beauchamp Aymeric 21301016
Chagneux Dimitri 21606807
Mori Baptiste 21602052
Leblond Valentin 21609038

L2-Info-groupe-4A

Table des matières

Objectifs

Description du Sokoban

Le Sokoban est un jeu de réflexion de type puzzle où le joueur doit placer des caisses sur des objectifs placés à l'avance sur la carte. Le joueur gagne si toutes les caisses sont placées sur les objectifs et il ne peut pousser qu'une seule caisse à la fois. Il existe de nombreux niveaux dont la difficulté est variante.

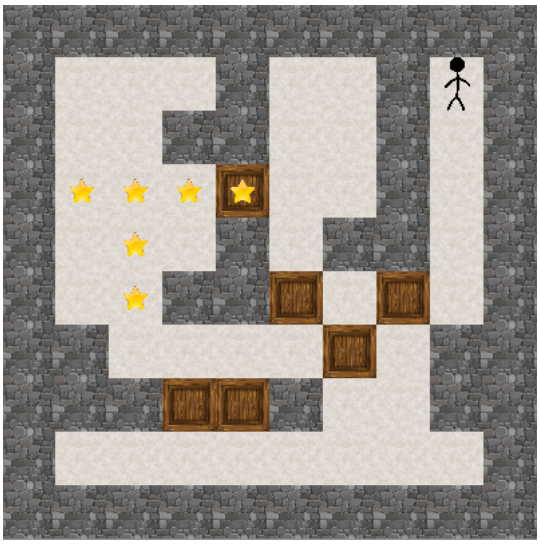

FIGURE 1 – Niveau du Sokoban

Les fonctionnalités attendues

Pour ce projet, nous devions mettre au point une version jouable pour un humain en console, en prenant en compte l'importation de niveaux (au format **.xsb**). Il était également demandé de réaliser une interface graphique et une fonctionnalité permettant une résolution automatique de niveau.
Enfin, permettre de faire jouer en parallèle un humain et un ordinateur, et rendre *anytime* l'algorithme de l'intelligence artificielle. C'est à dire le fait que lorsque le joueur fait un mouvement, l'intelligence artificielle doit en faire un.

1 Fonctionnalités implémentées

1.1 Description des fonctionnalités

Attendues

La version console fonctionne à l'aide de saisies de l'utilisateur qui lui permettent de contrôler le jeu. Une fois un niveau terminé, on demande au joueur si il souhaite passer au niveau suivant.

La carte est chargée à partir d'un fichier **.xsb** contenant des lignes de caractères, que l'on transforment en liste de caractères.
La carte est donc modélisée par des chaînes de caractères :

— # pour un mur
— **$** pour une caisse

— @ pour le joueur
— . pour un objectif
— * pour une caisse sur un objectif
— + pour le joueur sur un objectif
— **espace** pour les cases vides

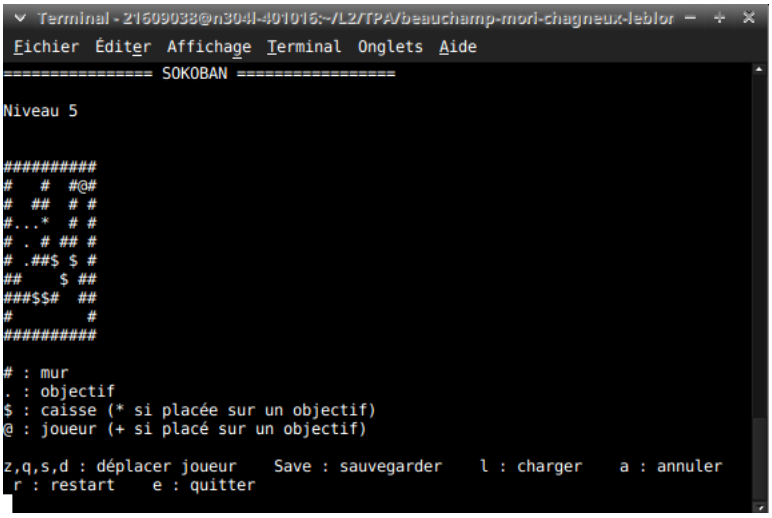

FIGURE 2 – Interface console

Au niveau de l'interface graphique, nous avons une zone de jeu dans laquelle on dessine le niveau et des boutons pour gérer les différentes fonctionnalités. Le personnage est déplaçable avec les flèches directionnelles ou ZQSD, si le joueur a bloqué une caisse ou si il a gagné, il ne peut plus bouger et doit recommencer le niveau ou passer au suivant (seulement si il a gagné). Si le joueur gagne, le personnage effectue le Dab et si il perd, le personnage pleure.

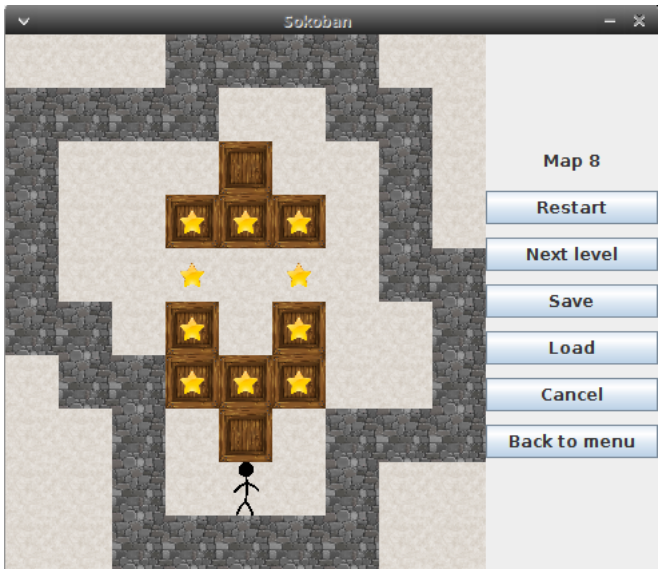

FIGURE 3 – Interface graphique

Ajoutées

Au lancement du programme, l'utilisateur peut choisir un profil ou en créer un nouveau. En fonction de l'avancement du profil donné, le joueur a débloqué un certain nombre de carte, pour débloquer la carte suivante il faut finir le niveau en cours. Lorsqu'un profil est chargé, la

dernière carte non terminée est lancée automatiquement.

Nous proposons également à l'utilisateur de sauvegarder sa partie à un instant du niveau donné, de charger sa sauvegarde (il n'y a pas de conflit entre la sauvegarde de chaque utilisateur), d'annuler le coup précédemment joué (cette fonctionnalité n'annule qu'un seul coup en arrière) et enfin de recommencer le niveau courant.

1.2 Organisation du projet

Pour le début du projet, nous avons tous travaillés ensemble sur la création de la structure de base du jeu sur comment on va représenter chaque éléments qui composent le jeu (joueur, case vide, caisse ou encore la carte qui les contiendra). Ensuite nous nous sommes interrogé sur la manière de déplacer le joueur ainsi que faire pousser les caisse par le joueur et faire en sorte de gérer les collisions avec les murs.
Nous avons rencontrés des difficultés pour la détection d'une caisse qui est bloqué, nous avons d'abord dit qu'une caisse est bloquée si elle se trouve entre deux murs qui se trouve sur deux axes différents (cas de gauche de la **figure 4**). Nous nous sommes rendu vite compte que ce n'était pas le seul cas de caisse bloquée, comme on peut le voir sur la deuxième image de la **figure 4**, on ne peut déplacer aucune des deux caisses car elles se bloquent entre elles, nous avons donc gérer ce cas mais même en fin de projet nous avions oubliés certains cas de caisses bloquées (le fait d'avoir un amas de caisse rendait la détection plus compliquée et les cas étaient particuliers).

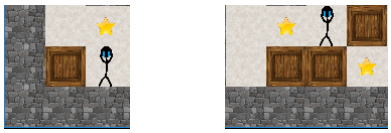

FIGURE 4 – Blocage simple de caisse(s)

Ensuite, nous nous sommes séparés en trois groupes, deux d'entre nous se sont lancés sur l'IA avec l'algorithme A*, puis une autre personne s'est lancée dans la création d'un main et toute la gestions de fichiers (sauvegarde, chargement, traduction entre une carte enregistré dans un fichier et notre représentation d'un niveau de notre code). Enfin la dernière personne s'est occupée de gérer des cas plus complexes de détections de caisses bloquées puis est partie dans la mis en place de l'interface graphique.

L'interface graphique ayant bien avancé, des nouvelles fonctionnalités ont été ajoutées dans celle-ci et des réglages dans la manière de sauvegarder une map ont été changés ce qui a posé un problème au niveau du main qui faisait fonctionner le jeu en version console, celle-ci n'était plus à jour il a fallut donc revoir notre main de la console pour que la fonctionnalité de sélection de profils qui a été implémentée d'abord en version graphique, changer la manière de sauvegarder une map car dans la première version, la méthode ne nous permettait pas de recommencer le niveau courant.

Au niveau de l'IA, nous avions réussi assez tardivement à faire résoudre une grande partie de nos maps en quelques secondes, en effet nous avions rencontré pas mal de problèmes en ce qui concerne les choix que doit effectuer l'IA et de comprendre pourquoi elle ne faisait pas ce que l'on attendait.

2 Éléments techniques

3 Architecture du projet

3.1 Organisation des packages

Notre projet se décompose en trois packages distincts :
— le package **sokoban**, le package principal qui gère toute la version console, le chargement de fichier, et est utilisé dans les deux autres packages ;
— le package **graphique**, qui gère l'interface graphique en utilisant plusieurs classes du package précédent ;
— le package **ia**, qui implémente les algorithmes de résolution automatique.

En créant ces trois packages, nous avons pu travaillé en parallèle sur différents aspects du projet sans pour autant se gêner les uns les autres. Ainsi, une modification du package **ia** ne changeait rien au fonctionnement du sokoban en version console ou graphique.
Cependant, après modification du package **sokoban**, tout le groupe devait récupérer la version la plus récente, car ce package est utilisé par les deux autres.

3.2 Organisation des classes

3.2.1 Package sokoban

Notre package principal comporte treize classes réparties en différentes catégories :
— les classes gérant les entités (personnage, murs, ...) ;
— les classes permettant d'enregistrer/charger une sauvegarde ;
— les classes permettant de lancer le programme.

Toutes les classes de gestion des entités héritent de la classe *Block*, elles permettent de déplacer facilement le personnage et les caisses, de gérer les collisions, les blocages et la fin d'une partie.

Les classes gérant les fichiers permettent de lire et d'écrire dans des fichiers situés dans un dossier "maps" contenant toutes les cartes du jeu, ou dans un fichier "save" stockant les profils et leur sauvegarde/cancel.

Les classes permettant de lancer le programme sont le Board, qui créer la grille de jeu, et le Main.

3.2.2 Package graphique

Ce package permet simplement de gérer la version graphique du Sokoban et de jouer sans les entrées claviers (en appuyant directement sur une touche, on peut déplacer le joueur).

3.2.3 Package ia

Le package **ia** a permit d'effectuer de nombreux tests sans que cela bloque le fonctionnement du jeu. Il est constitué de plusieurs classes implémentant différents algorithmes comme *A**, il permet également de calculer le coût d'un chemin, grâce à la classe *PathCost*.

4 Expérimentations et usages

Conclusion