Université de Mons
Faculté des sciences
Département d'Informatique

Sokoban

Rapport de projet

Professeur :
Hadrien Mélot
Assistants :
Gauvain Devillez
Jérémy Dubrulle
Sébastien Bonte

Auteurs :
Pignozi Agbenda
Theo Godin
Ugo Proietti

Année académique 2020-2021

Table des matières

1 Introduction

1.1 Objectif

Ce projet est réalisé pour le cours de projet d'informatique donné en BAC 1 Sciences Informatiques a l'université de Mons. Il s'agit d'un jeu de sokoban écrit en java sur la version 11. Cette version n'est pas la plus récente (java 15 était disponible lors de la réalisation du projet début 2021) mais c'est la version LTS (long term support) la plus récente. Voulant rendre le jeu open source sur GitHub, il nous semblait important d'utiliser une version LTS afin que d'autres étudiants puissent utiliser et/ou analyser notre code dans les années a venir.

1.2 Procédure

Afin de rendre le projet realisable, nous avons du nous organiser serieusement.
Nous avons commencé par réfléchir a la logique du jeu et faire un premier prototype en Python afin de souligner les points importants avant de se mettre a la programmation du moteur de jeu.
Ensuite differents outils ont été créés, facilitant le developpement du moteur qui était deja assez complexe et permettant de se concentrer sur des fonctionalités plus avancées.
Une base solide etant établie, le developpement de l'interface graphique a pris part au projet. Sans pour autant laisser de côté des améliorations du moteur et des tools.
Nous avons termine par le debugage et l'ecriture du rapport.

2 Mode d'emploi

Le projet utilise Gradle comme système d'automatisation permettant de gérer facilement les dépendances et la compilation du code java et de la javadoc.

2.1 Gradle

Nous avons ajoute deux task a Gradle, trouvable a la fin du fichier *build.gradle*
— checkMap
— movReplay
Ces task sont utilisables de cette maniere :

checkMap
Cette task permet de verifier si des maps au format `.xsb` sont au format attendu.
Premier argument - **f** ou **d** permettant de dire au programme si on veut l'executer sur un **file** ou sur un **directory**
Second argument - **path**
Exemple - **./gradlew checkMap –args="fapp/build/resources/main/levels/map1.xsb"**

movReplay
Cette task permet de rejouer un fichier `.mov` sur un fichier `.xsb`. Cette task va automatiquement chercher dans le repertoire `build/resources/main/levels` et `build/resources/main/appdata/movements`. Et va ecrire un fichier de sortie dans `build/resources/main/levels/save`
Premier argument - **map** La map au format `.xsb`
Second argument - **mov** Le fichier de movements au fornat `.mov`
Exemple - **./gradlew movReplay –args="ma1 mov1"**

2.2 Dépendances

ffmpeg - nécessaire sur les systèmes UNIX afin d'afficher correctement la video de fond de l'écran principal.

2.3 Guide d'utilisation

Notre sokoban comporte 3 modes.

La campagne principale disponible via le menu play. Ce mode permet aux joueurs de jouer 15 niveaux prédéfinis qui se déloquent les uns après les autres. Afin de compléter un niveau, il faut pousser toutes les caisses sur les croix. Le déplacement du personnage se fait avec les touches `z,q,s,d` ou avec les boutons présents sur l'interface graphique. Ces touches peuvent être redéfinies via le menu option.

Les modes Random et Builder accessibles via le menu Arcade. Le mode Random génére une map aléatoire dont on peut choisir les dimensions ainsi que le nombre de boites à pousser.
Le mode Builder permet au joueur de designer ses propres niveaux de jeux.
L'interface de Builder propose un niveau vide, de dimension 15*15, où chaque case peut être modifiée en cliquant dessus après avoir séléctionné un type de case dans la barre de gauche.

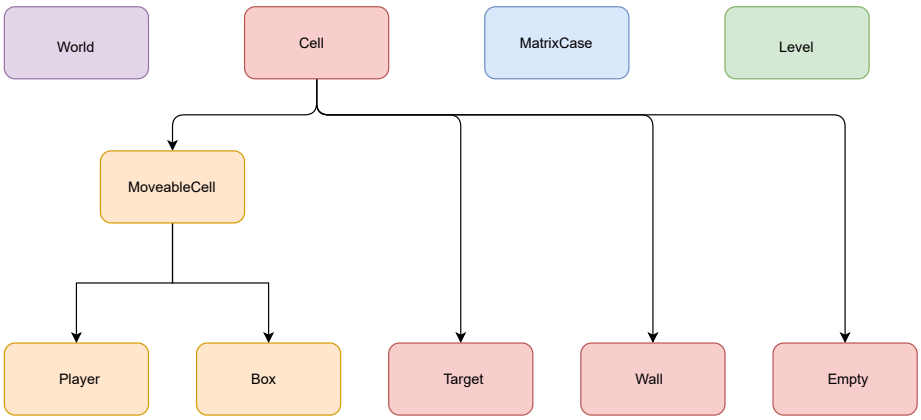
En plus de ces modes, n'importe quel niveau de Sokoban peut également être joué si le joueur possède le fichier d'extension `.xsb`. Il suffit copier celui-ci dans le répertoire `sokoban/app/src/main/resources/levels/save`. Il apparaitra ensuite (après un redémarage du jeu) dans la liste déroulante de la section load dans le menu Arcade avec les niveaux créés par le joueur et les niveau Random enregistrés.

3 Algorithme

Dans cette section, vous allez expliquer les différents algorithmes qui vous paraissent importants pour votre projet. (Pour l'explication : son principe, les grandes lignes de comment il s'exécute, sa complexité,...)

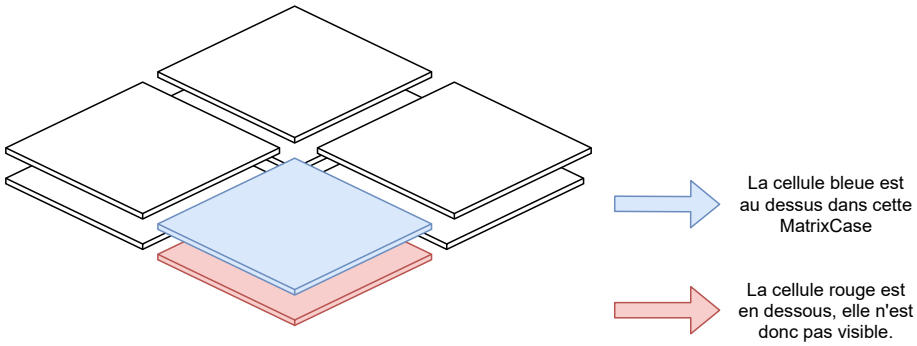
3.1 Programmation orientee objet

Chaque élément de base du jeu est représenté par une classe. Player représente un joueur controllé par l'utilisateur, World représente le monde contenant le joueur et la map dans laquel il évolue, MatrixCase représente une case de la map, Cell représente tous les différents types de cellules et enfin Level représente un niveau de jeu.
Ces différentes classe sont organisées selon la hiérarchie suivante :



3.2 Representation d'une map

Le jeu sokoban nécessite l'implémentation de différentes cellules comme le sol, les murs, les boites, etc..
Une map doit donc pouvoir contenir toutes ces cellules et permettre de les déplacer. Dans ce but, nous avons choisi de représenter une map par une matrice. Chaque "case" de cette matrice est un objet instancié à partir de la classe MatrixCase. Une map est donc une matrice d'objets MatrixCase.
Qu'est-ce qu'une MatrixCase ? C'est un objet qui contient deux cellules. En effet une map de sokoban doit permettre la superposition de deux cellules car le joueur ou une boite peuvent se positionner au dessus du sol ou d'une cellule cible. Schéma d'une MatrixCase :



Grâce à cette représentation d'une map, on peut accéder à n'importe quelle cellule de la matrice grâce à ses coordonnées.

3.3 Chargement d'un niveau

Un objet instancié à partir de la classe Level du package sokoban.Engine.Objects.Level peut contenir toutes les informations d'un niveau de sokoban, comme le joueur, le monde dans lequel il évolue, le nom du niveau et sa taille. La classe Level permet de passer rapidement d'un niveau à l'autre grâce à la méthode setLevel(). Celle-ci prend un nom de map en paramètre et va chercher le fichier xsb correspondant. Nos fichiers xsb respectent les conventions du sokoban i.e. chaque case du jeu est représenté en texte par un charactère spécifique. Ce fichier est ensuite tranformé en String par la méthode load de la classe MapLoader. Ce String est finalement donné en paramètre à la méthode init de la classe Builder qui va pour chaque charactère de ce String créer la case adéquate dans la matrice représentant la map. Ainsi, toutes les informations nécessaires afin d'afficher un niveau sont disponibles via un objet Level.

3.4 Déplacements

Seules les cellules héritant de la classe MoveableCell peuvent être déplacées (voir la hiérarchie des objets). Le joueur est entièrement controllé par les inputs de l'utilisateur. Cependant, il ne pourra se déplacer uniquement sous certaines

conditions. En effet le joueur ne doit pas pouvoir traverser ou pousser n'importe quelle cellule. Voici donc la procédure que nous suivons pour déterminer si oui ou non le joueur peut se déplacer dans la direction donnée par l'utilisateur :
Tout d'abord, il faut analyser la case voisine au joueur dans la direction donnée.

- Si celle-ci n'est ni traversable ni poussable, alors le joueur ne peut pas se déplacer.
- Si elle est traversable et non-poussable, le joueur peut se déplcer.
- Si elle est poussable, il faut alors analyser la cellule suivante (toujours dans la même direction).
    1. Si la cellule suivante est traversable et non-poussable alors, le joueur peut se déplacer et pousser la cellule devant lui.
    2. Sinon, le joueur ne peut pas se déplacer.
- Dans tous les autres cas, le joueur ne pourra pas se déplacer.

Maintenant que nous avons determiné si le joueur peut se déplacer, on peut déplacer celui-ci dans la matrice représentant la map du niveau et actualiser l'affichage.

Cette procédure de déplacement a été implémentée de la façon suivante :
Chaque objet représentant une cellule possède les attributs booléens softCollision et hardCollision. softCollision indique si une cellule est poussable ou non et hardCollision si elle est traversable.
Ainsi, en ayant les coordonnées des cellules intervenants dans le déplacement du joueur dans une direction donnée, nous pouvons effectuer les différents tests cités ci-dessus en vérifiant les collisions d'une cellule d'une case précise de la matrice.

3.5 Options utilisateur

3.6 Génération aléatoire de niveau

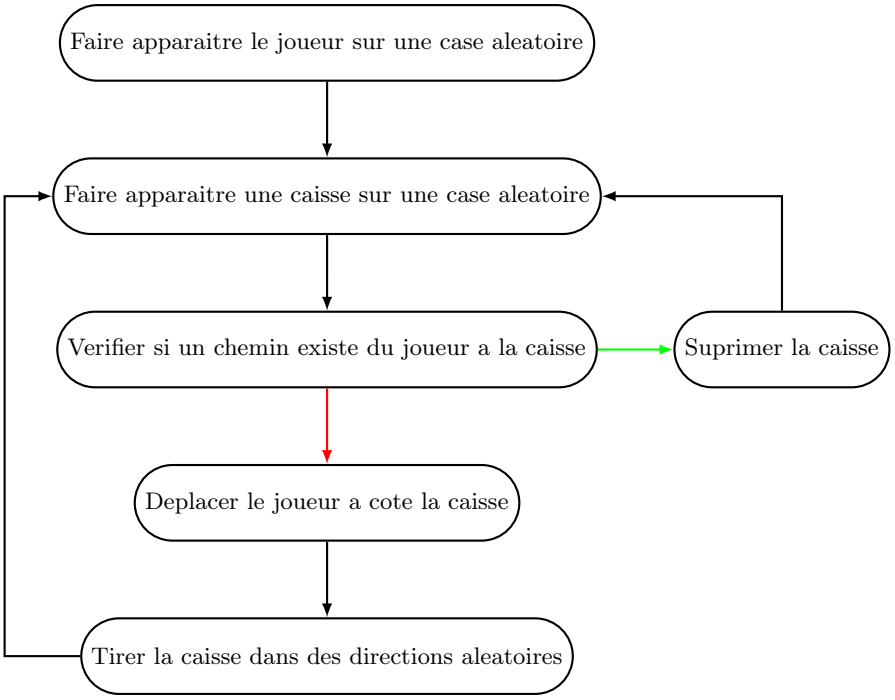
3.6.1 Backward induction

Afin d'implémenter la génération aléatoire de niveaux, nous avons commencé par nous renseigner sur les différentes techniques extistantes, notre première idée fut de nous diriger vers une génération procédurale. Avant de valider ce choix, nous avons questionné notre professeur de mathématique Mr. Brihaye qui nous a conseillé de nous renseigner sur le principe de backward induction. Après quelques recherches, nous avons décidé d'utiliser cette technique qui semblait plus simple à mettre en place. La backward induction consiste à partir d'un problème résolu et faire les étapes en marche arrière jusqu'à retrouver un problème initial. Pour notre sokoban, cela signifie partir d'un niveau résolu et tirer les caisses une à une. Ainsi, on assure que le niveau sera réalisable puisqu'il suffira de faire les mêmes déplacements dans l'autre sens.
Nous avons alors imaginé les étapes nécessaires à la création d'un niveau jouable de sokoban.

1. Créer la map vide initiale

2. Générer les murs aléatoirement
3. Placer le joueur sur une case libre aléatoire
4. Placer aléatoirement et tirer les boites

— Etape 1
La map initiale consiste en une map vide d'une certaine largeur et longueur fermée par des murs.
— Etape 2
Ensuite, les murs sont générés aléatoirements. Pour chaque case de la map on détermine un pourcentage de chance qu'un mur apparaisse.
— Etape 3 et 4
Après avoir placé aléatoirement le joueur, pour chaque boite à placer on va suivre la procédure décrite sur ce schéma :



3.6.2 A* path finding

L'étape 4 de la génération nécessite l'implementation d'un algorithme de pathfinding.
Nous avons choisi le très connu A* algorithm pour son adaptabilité et le très grand nombre de ressources disponibles pour son implémentation.
cet algorithme consiste à se déplacer d'un point A vers un point B de case en case, en choisissant comme prochaine case celle qui a la plus courte distance vers le point B et le point A.

4 Points forts et points faibles

4.1 Points forts

4.1.1 Graphismes

La plupart des éléments graphiques ont été réalisées par un membre du groupe, ce qui nous a permi d'être totalement libre par rapport à la direction artistique de notre jeu. Ainsi, le style retro wave a été choisie pour les graphismes et la musique de notre jeu. C'est un style peu utilisé pour des jeux de sokoban et qui plait à tous les membres. Au niveau des interfaces graphiques, ils ont été rendus épurés et le plus intuitifs possible.

4.1.2 Performances

Jeu fluide partout sauf dans le builder

4.1.3 Editeur de niveau

4.1.4 Enregistreur de niveau aleatoires

4.1.5 Menu d'options

Le menu permettant de modifier les options est relativement complet

4.1.6 Pack de textures

Il est tres facile de changer les textures et d'importer son propre pack

4.2 Points faibles

4.2.1 Generation aleatoire

4.2.2 Interface non responsive

4.2.3 Taille des niveaux

La taille des niveaux est limitee a un carre de 15*15 blocs

4.2.4 Parametres permanents

La classe Settings.java permet de stocker des valeurs et de les recuperer dans un fichier. Nous voulions l'utiliser pour conserver les parametres utilisateur entre chaques redemarrages mais nous n'avons pas eu le temps de l'integrer. Cette classe utilise java.util.Properties afin de recuperer les variables sauvegardees, modifier les valeurs et stocker les modifications.

Nous avons decide de laisser cette classe malgre le fait qu'elle n'est pas utilisee. Elle sert actuellement de "preuve de concept" et pourra eventuellement etre utilisee plus tard si nous decidons de continuer de travailler sur le jeu.

5 Erreurs restantes

5.1 Generation

Si il est impossible de placer une boîte accessible par le joueur et qui peut être déplacée d'au moins une case, le jeu plante.

Cela arrive sur des petites maps avec trop de caisses. Le jeu essaye de placer des caisses mais n'y arrive pas et répète cette opération sans cesse, ce qui fige le jeu. Il faut absolument forcer son arrêt (par exemple ctrl + c dans le terminal).

Ce bug peut être réglé mais nous n'avons pas eu le temps de nous en occuper. Il est assez dérangeant car il faut absolument relancer le jeu mais il peut être évité en choisissant de tailles plus grandes et avec moins de caisses.

5.2 Chargement d'une map

Quand l'utilisateur ajoute une map dans le dossier qui y est consacré, ou s'il sauvegarde un map fraîchement terminée, il ne pourra pas la charger. En effet, le jeu a besoin d'un redémarrage pour recharger la liste des maps. Nous n'avons pas trouvé comment rafraîchir cette liste pendant que le jeu est lancé.

Ce bug est assez dérangeant car il force l'utilisateur à redémarrer le jeu ce qui n'est pas du tout pratique.

5.3 Builder

Si l'utilisateur crée une map non fermée dans le menu Arcade -> Builder, il pourra l'enregistrer mais il ne pourra pas la charger dans Arcade -> Load.
La méthode mapTrimmer située dans sokoban.Engine.Tools.MapLoader rend les maps non rectangulaires en map rectangulaires afin de travailler avec une hauteur et une largeur fixe peut importe la position sur la map. Cela simplifie le fonctionnement du moteur du jeu mais a pour conséquence de générer une erreur si une map n'est pas fermée.

Dans notre cas l'erreur est simplement indiquée dans le terminal et nous n'avons pas pris la peine de l'implémenter dans l'interface. Ce bug n'est pas très gênant car il ne fait pas planter le jeu, si l'utilisateur charge une map non fermée il ne se passe tout simplement rien à ses yeux.

6 Choix personnels

Dans cette section, vous allez expliquer et justifier les choix que vous aurez fait (par exemple pourquoi utiliser un tri à la place d'un autre).

6.1 Open source

Nous avons décide de rendre le projet entièrement open-source et de l'héberger sur GitHub. En effet, nous tenons énormement à ce mouvement et développer un programme fermé n'est pas une possibilité pour nous.

L'open-source représente l'avenir de l'informatique et de l'enseignement, tout le monde n'a pas le luxe de se payer différentes resources. L'open-source représente aussi la garantie d'un code plus sur, plus performant et plus flexible. Nous espérons que notre projet pourra servir de support pour d'autres étudiants et nous les encourageons a faire de même afin de contribuer au mouvement FOSS.

6.2 FXML

6.3 Tools

Pourquoi on a fait des tools pour nous faciliter la vie

6.4 Easter eggs

7 Conclusion

Enfin dans cette section, vous ferez un bref rappel du sujet de votre projet, de comment vous avez fait pour résoudre le problème, des résultats s'il y en a(s'il y a beaucoup de résultats préférer une section à part entière) et enfin ce que le projet vous a apporté.