Université de Mons
Faculté des sciences
Département d'Informatique

Sokoban

Rapport de projet

Professeur :
Hadrien Melot

Auteurs :
Pignozi Agbenda
Theo Godin
Ugo Proietti

Année académique 2020-2021

Table des matières

1 Introduction

Dans cette section, vous rappelez l'objectif de votre projet et donnez un aperçu de comment vous allez procéder.

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.

2 Mode d'emploi

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

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 - **./gradlewcheckMap –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.

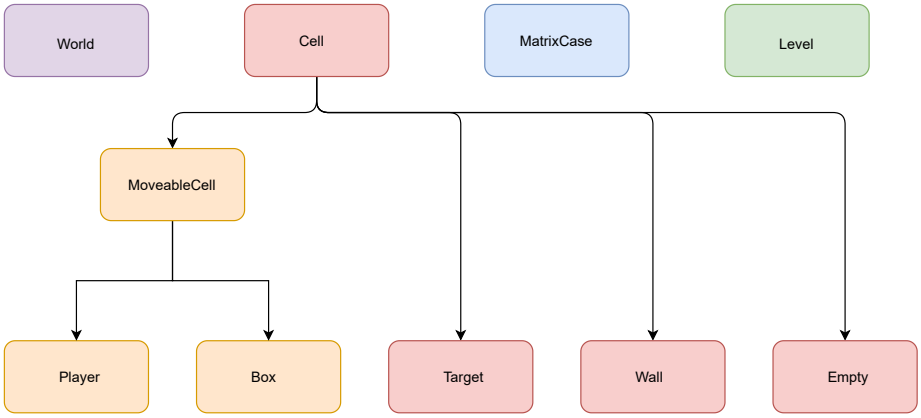
En plus de ces modes, n'importe quel niveau de Sokoban peut également être joué si le joueur possède le fichier d'extension `.xsb`. Il suffit copier celui-ci dans le répertoire `sokoban/app/src/main/resources/levels/save`. Il apparaitra ensuite (après un redémarage du jeu) dans la liste déroulante de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

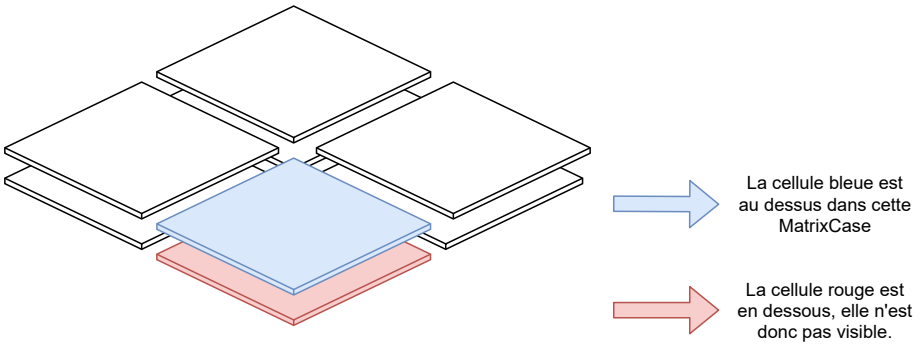
3.1 Programmation orientee objet

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 choisit 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. 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.

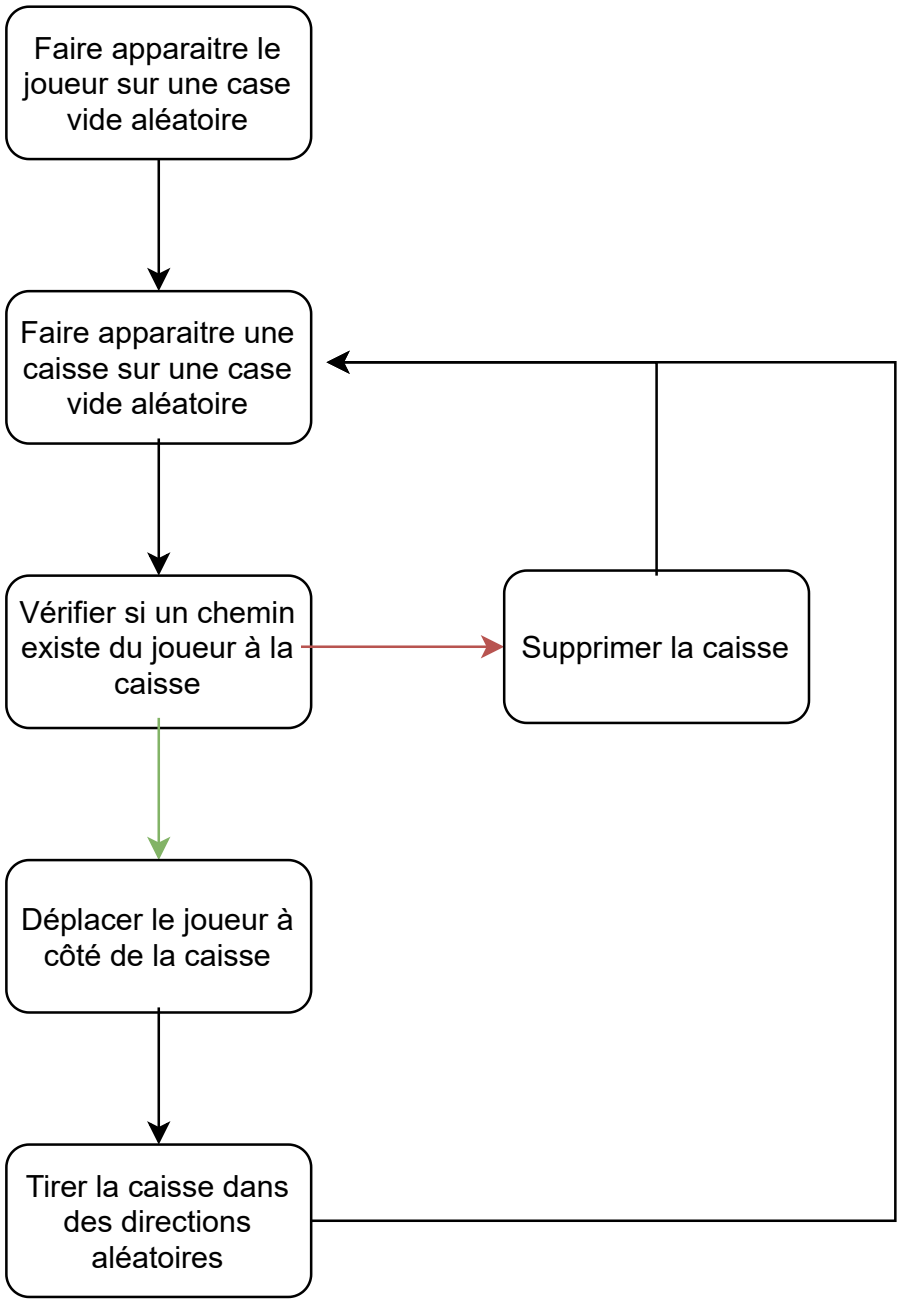
Maintenant que nous avons determiné si le joueur peut se déplacer, on peut déplacer celui-ci dans la matrice représentant la map du niveau et actualiser l'affichage.

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 Pack de textures

3.7 Génération aléatoire de niveau



3.7.1 Placement aléatoire

3.7.2 Backward induction

3.7.3 A* path finding

4 Points forts et points faibles

4.1 Points forts

4.1.1 Graphismes

Textures jolies et interfaces intuitives et soignees

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

5 Erreurs restantes

5.1 Generation

La generation aleatoire de niveaux aleatoires ne fonctionne pas dans tous des cas

5.2 Chargement d'une map

Le jeu a besoin d'etre redemarre pour actualiser la liste de map a charger

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

6.2 Github

6.3 FXML

6.4 Tools

Pourquoi on a fait des tools pour nous faciliter la vie

6.5 Easter eggs

7 Problèmes survenus

7.1 Problèmes logistiques

Nous avons rencontre deux problèmes "logistiques" durant toute la durée du projet. En effet nous sommes deux étudiants sur Linux et un sur Windows. De plus les étudiants sur Linux utilisent Vim et VSCode comme éditeur de texte alors que l'étudiant sur Windows utilise Intellij.
Les problèmes rencontres ont étés lies a la façon dont Intellij gere Git et Gradle, ce n'a pas ete facile de bien le configurer, et encore moins facile de gérer les surprises que Intellij a push sur le repo GitHub. A plusieurs reprises l'étudiant en charge du repo a du gérer des conflits et parfois revenir sur des commits précédents pour annuler ce que Intellij fait en fond sans en avertir l'utilisateur.

7.2 Problèmes algorithmiques

Nous avons du faire des rework

8 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.