

Université de Mons
Faculté des sciences
Département d'Informatique

Sokoban

Rapport de projet

Professeur :
Hadrien MELOT

Auteurs :
Pignozzi AGBENDA
Theo GODIN
Ugo PROIETTI



Année académique 2020-2021

Table des matières

1	Introduction	3
1.1	Objectif	3
1.2	Procédure	3
2	Mode d'emploi	3
2.1	Gradle	3
2.2	Dépendances	4
2.3	Guide d'utilisation	4
3	Algorithme	5
3.1	Programmation orientee objet	5
3.2	Representation d'une map	5
3.3	Chargement d'un niveau	7
3.4	Déplacements	7
3.5	Options utilisateur	7
3.6	Pack de textures	7
3.7	Génération aléatoire de niveau	7
3.7.1	Placement aléatoire	8
3.7.2	Backward induction	8
3.7.3	A* path finding	8
4	Points forts et points faibles	8
4.1	Points forts	8
4.1.1	Graphismes	8
4.1.2	Performances	8
4.1.3	Editeur de niveau	8
4.1.4	Enregistreur de niveau aleatoires	8
4.1.5	Menu d'options	8
4.1.6	Pack de textures	8
4.2	Points faibles	8
4.2.1	Generation aleatoire	8
4.2.2	Interface non responsive	8
4.2.3	Taille des niveaux	8
5	Erreurs restantes	8
5.1	Generation	8
5.2	Chargement d'une map	8
6	Choix personnels	9
6.1	Open source	9
6.2	Github	9
6.3	FXML	9
6.4	Tools	9
6.5	Easter eggs	9

7	Problèmes survenus	9
7.1	Problèmes logistiques	9
7.2	Problèmes algorithmiques	9
8	Conclusion	9

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 à 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 à venir.

1.2 Procédure

Afin de rendre le projet réalisable, nous avons dû nous organiser sérieusement.

Nous avons commencé par réfléchir à la logique du jeu et faire un premier prototype en Python afin de souligner les points importants avant de se mettre à la programmation du moteur de jeu.

Ensuite différents outils ont été créés, facilitant le développement du moteur qui était déjà assez complexe et permettant de se concentrer sur des fonctionnalités plus avancées.

Une base solide étant établie, le développement de l'interface graphique a pris part au projet. Sans pour autant laisser de côté des améliorations du moteur et des outils.

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 ajouté deux tâches à Gradle, trouvables à la fin du fichier *build.gradle*

- `checkMap`
- `movReplay`

Ces tâches sont utilisables de cette manière :

checkMap

Cette tâche permet de vérifier si des maps au format `.xsb` sont au format attendu.

Premier argument - **f** ou **d** permettant de dire au programme si on veut l'exécuter 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 mouvements au format `.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 redéfinie via le menu option.

Les modes Random et Builder accessibles via le menu Arcade. Le mode Random génère 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.

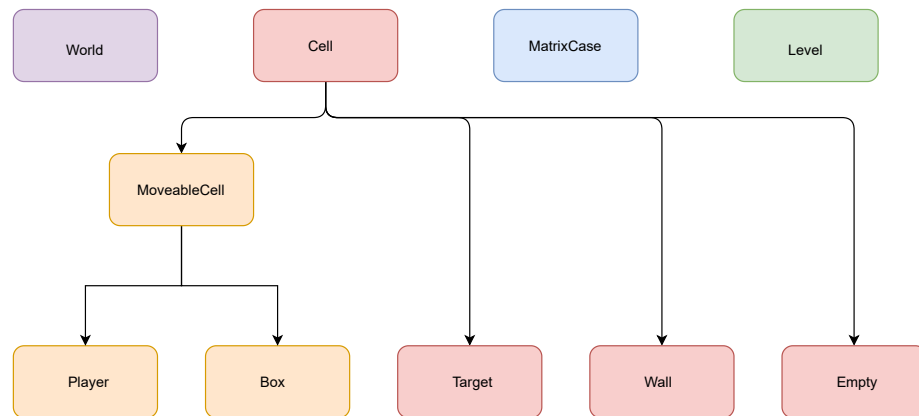
L'interface de Builder propose un niveau vide, de dimension 15*15, où chaque case peut être modifiée en cliquant dessus après avoir sélectionné un type de case dans la barre de gauche.

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 repertoire `sokoban/app/src/main/resources/levels/save`. Il apparaîtra ensuite dans la liste déroulante de la section load dans le menu Arcade avec les niveaux créés par le joueur et les niveau Random enregistrés.

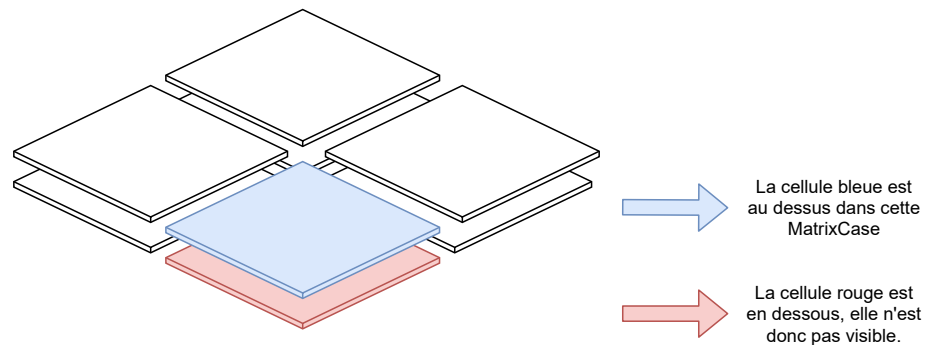
3 Algorithme

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



3.2 Representation d'une map



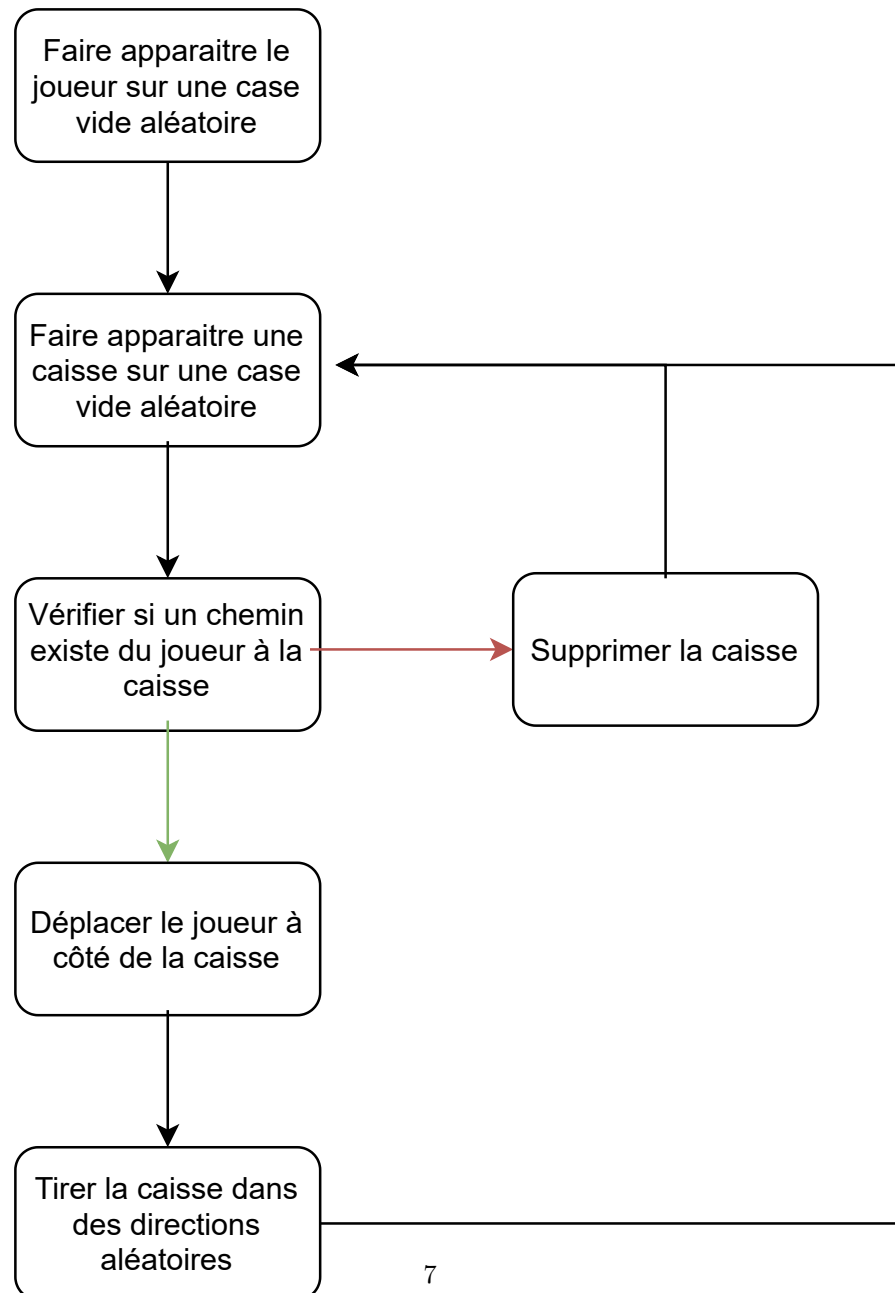
3.3 Chargement d'un niveau

3.4 Déplacements

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 soignées

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 rencontré 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 rencontrés ont été liés à la façon dont IntelliJ gère Git et Gradle, ce n'a pas été facile de bien le configurer, et encore moins facile de gérer les surprises que IntelliJ a push sur le repo GitHub. À plusieurs reprises l'étudiant en charge du repo a dû 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 dû 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.