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 **3**
- 1.1 Objectif . . . 3
- 1.2 Procédure . . . 3
- 1.3 Organisation . . . 3

2 Mode d'emploi **5**
- 2.1 Gradle . . . 5
- 2.2 Dépendances . . . 5
- 2.3 Guide d'utilisation . . . 5

3 Implémentation **7**
- 3.1 Programmation orientee objet . . . 7
- 3.2 Representation d'une map . . . 7
- 3.3 Chargement d'un niveau . . . 8
- 3.4 Déplacements . . . 8
- 3.5 Options utilisateur . . . 9
- 3.6 Génération aléatoire de niveau . . . 9
  - 3.6.1 Backward induction . . . 9
  - 3.6.2 A* path finding . . . 10

4 Interface **11**
- 4.1 FXML . . . 11
- 4.2 JavaFX . . . 11
- 4.3 Ecran de démarrage . . . 12
- 4.4 Menu d'option . . . 12
- 4.5 Scene de jeu . . . 13
- 4.6 Erreur . . . 14

5 Points forts et points faibles **15**
- 5.1 Points forts . . . 15
  - 5.1.1 Graphismes . . . 15
  - 5.1.2 Performances . . . 15
  - 5.1.3 Editeur de niveau . . . 15
  - 5.1.4 Enregistreur de niveau aleatoires . . . 15
  - 5.1.5 Menu d'options . . . 15
  - 5.1.6 Pack de textures . . . 15
- 5.2 Points faibles . . . 15
  - 5.2.1 Generation aleatoire . . . 15
  - 5.2.2 Interface non responsive . . . 15
  - 5.2.3 Taille des niveaux . . . 15
  - 5.2.4 Parametres permanents . . . 16

6 Erreurs restantes **16**
6.1 Generation 16
6.2 Chargement d'une map 16
6.3 Builder 16

7 Choix personnels **17**
7.1 Open source 17
7.2 FXML 17
7.3 Tools 17
7.4 Easter eggs 17

8 Conclusion **17**

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 étant 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 terminé par le débugage et l'écriture du rapport.

1.3 Organisation

Pour rester bien organisé, il était important d'établir qui fait quoi. Nous nous sommes réparti les tâches de cette manière :

Pignozi Agbenda s'est occupé en majorité de l'interface graphique en JavaFX. Il s'est aussi renseigné sur FXML et le CSS mais nous n'avons pas utilisé ces technologies dans notre projet.

Théo Godin a pensé et construit la logique du jeu et l'aspect programmation orientée objet. Il a aussi développé la plus grosse partie du générateur de niveau et a aidé P.Agbenda pour la partie interface.

Ugo Proietti a développé les tools et a pensé la logique de la génération. C'est également lui qui s'est occupé de la logistique, c'est-à-dire la configuration et l'enseignement de Gradle et de git au groupe.

Nous avons collaboré en utilisant git et hébergé notre repo sur Github. Une branch était créée par fonctionnalité. Une fois terminée, nous faisons un pull re-

quest sur la branch main. Nous avons également utilisé la fonctionnalité Issues permettant de rester à jour sur les bugs détectés. L'utilisation de Github nous a énormément aidé dans l'organisation du projet.
Voici le lien du repo (qui restera en privé avant d'avoir eu l'autorisation du professeur et des assistants de le passer en publique) : https://github.com/SIERRA-880/sokoban

Discord a été le canal de communication privilégié durant le développement.

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.

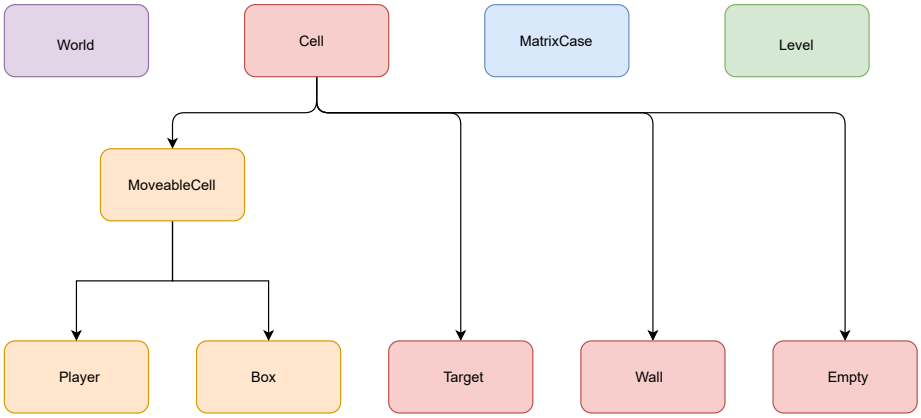
Les modes Random et Builder accessibles via le menu Arcade. Le mode Random génére une map aléatoire dont on peut choisir les dimensions ainsi que le nombre de boites à pousser.
Le mode Builder permet au joueur de designer ses propres niveaux de jeux.
L'interface de Builder propose un niveau vide, de 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 Implémentation

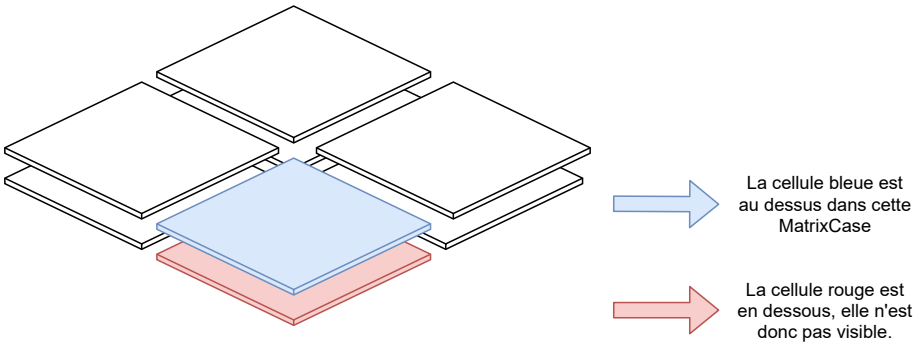
3.1 Programmation orientee objet

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.

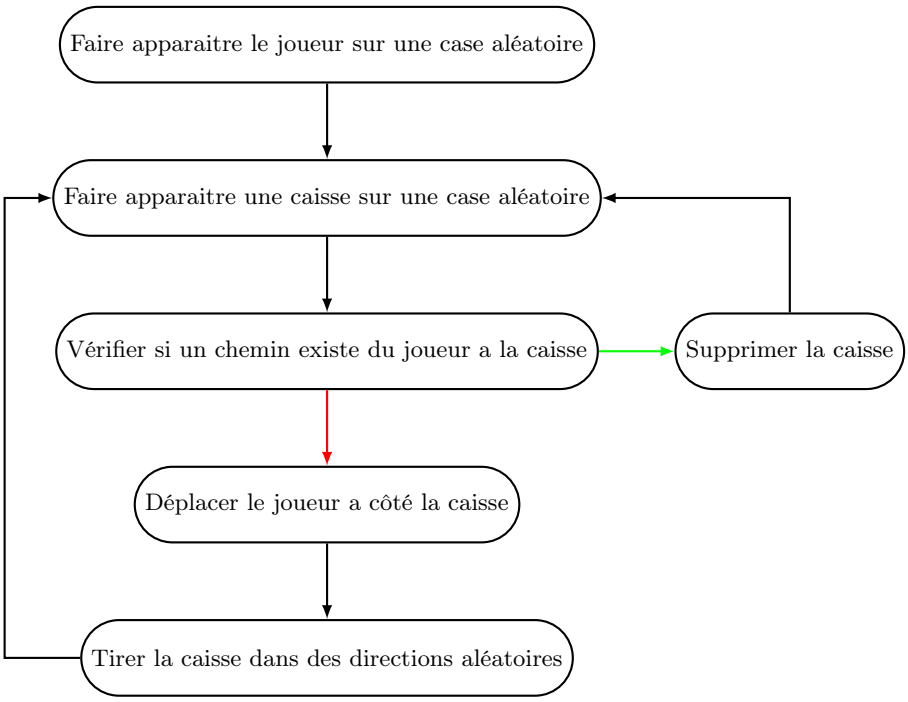
3.5 Génération aléatoire de niveau

3.5.1 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 :



3.5.2 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 Interface

Concernant le design de l'interface graphique on s'est fié aux instructions d'un membre de l'équipe qui avait déjà un peu d'expérience car il avait comme projet personnel de créer un jeu.
Son idée était la suivante : diviser toute la partie interface graphique en 3 sections. Une section Scene qui comme son nom l'indique sera utile pour la création des différentes scènes contenues dans notre jeu et une section pour des objets réutilisable dans plusieurs scènes appelé Widgets. On a aussi la classe Game qui sert à lancer l'application, la classe Controller qui contient des méthodes utilisées par tous les scènes et deux enum différent contenant pour l'un les scènes et l'autre les types de cells. Une fois la marche à suivre décidée nous nous sommes attelés au travail, bien sûr le plus compliqué n'était pas d'avoir un plan mais de s'y tenir. En effet on s'est vite rendu compte que les objets que l'on mettait dans la section Widgets n'étaient pas que des widgets mais aussi différents Pane et d'autre formes de layout. Nous n'avons cependant pas changé les noms des sections de un par habitude et de deux car il y avait toujours une séparations claire entre les deux grandes sections. Et à partir de ce moment là, Scenes resterait la section qui ne contiendrait que des scènes, et Widget deviendrai la section contenant tout ce qu'on pouvait trouver dans une scène de notre jeu, que ce soit notre classe ImageButton.java qui est réutilisé une multitude de fois, ou la classe OptionPane.java qui n'est utilisé qu'une fois.

Mais avoir un plan n'était pas tout, il nous fallait aussi savoir vers quelle style graphique se diriger. Après un brainstorm ou plusieurs idées ont été évoquées (Portal, Pokémon,Bomberman,...), ce fût le thème "Cyber-retrowave" qui a fait l'unanimité et une fois le thème décidé il ne nous restait plus qu'à commencer le travail.

4.1 FXML

Un des membres de notre équipe avait lancé comme idée d'utiliser SceneBuilder ce qui permettrait de rapidement et facilement créer et disposer les éléments d'une scène et d'en voir le résultat en direct. Après quelques recherches nous avons découvert qu'en général, la partie purement graphique se faisait en FXML et javaFx n'était utilisé que pour créer des interactions entre les différents éléments. Mais on a abandonné de peur de mal gérer les relations entre javaFx et FXML en particulier le changement de scènes.

4.2 JavaFX

Une fois le langage choisi et approuvé par tous, nous avons commencé à coder. Contrairement à nos attentes, JavaFx était assez facile à comprendre de par sa très forte similitude avec Java. Nous avons quand même eu un petit soucis à comprendre les méthodes d'actions et les Handler.

Notre façon de faire est simple : on crée une classe Scene portant le nom de la scène qu'elle représente et on lui associe une classe Pane. Cette dernière servira à contenir la majorité des éléments devant être présents dans la scène même si ce n'est pas toujours le cas.

4.3 Ecran de démarrage

Initialement, nous avions comme idée d'avoir un écran d'accueil animé. On a d'abord pensé faire défiler une même image en boucle. Le principe était simple, prendre deux images identiques, en faire défiler une sur l'écran et quand celle-ci sort de l'écran la deuxième image commencera à défiler de sorte à avoir une boucle perpétuelle. Mais on a pas réussi à l'implémenter. On rencontrait des problèmes avec l'effet de transition utilisé pour faire défiler l'image et on arrivait pas non plus à synchroniser la transition entre les deux images sans compter le fait de trouver une image qui ferait une belle boucle. On a donc abandonné cette idée et on a, à la place adopté la stratégie consistant à jouer une vidéo en fond. Les avantages étaient :

— On avait plus de transitions à gérer
— Des vidéos libres d'accès qui boucle avec elles-même sont facilement trouvables.

Après quelque temps de recherche nous avons trouvé la vidéo qui fera office de fond d'écran lors du démarrage du jeu, et en plus elle correspondait parfaitement avec notre thème cyber-retrowave. On a réussi à faire exactement ce qu'on voulait et on a même pu ajouter des fonctionnalité qu'on retrouve dans beaucoup de jeu arcade, comme par exemple le menu qui apparaît d'un côté une fois qu'on a cliqué sur l'écran, le petit message qui clignote quand on clique...

4.4 Menu d'option

On voulait absolument laisser le choix à l'utilisateur de changer les touches utilisées pour déplacer le joueur. Pareillement pour le contrôle des différents sons présents car on sait d'expérience que le son d'un jeu de ce type peut vite devenir frustrant quand on y joue pendant longtemps. Le contrôle de la luminosité est un plus apporté pour tous les joueurs nocturnes qui sont éblouis par leurs écrans de jeu. On y donne aussi l'accès à un des easter eggs qu'on a implémenté. On est à nouveau très content du résultat mais on a rencontré beaucoup de problèmes pour cette scène.

1. Les différents layout utilisés se superposent mal, on avait donc des boutons qui n'étaient plus visible ou certains qui étaient encore visible mais ne réagissaient plus aux clics. Ce fut l'un des plus gros obstacles rencontré durant la création de l'interface graphique. Et comme c'était la première fois qu'on avait ce problème, on a pris beaucoup de temps pour trouver une solution pour y remédier. La solution bien que longue était simple, redimensionner les layouts pour qu'ils ne se superposent plus. Une fois implémenté, on a avait plus de problèmes.

2. Le slider contrôlant la luminosité. On pensait au début que ça allait être facile et qu'il ne faudrait que changer la luminosité du Stage ce qui affecterait la luminosité de tout les scènes, mais malheureusement c'était impossible. La classe Stage ne possède pas de telles méthodes. On s'est ensuit dit qu'on pouvait le faire sur un Pane qu'on et qu'on mettait le même Pane dans chaque scène et quand on change de scène de la scène A vers la scène B on efface toute chose du pane concernant la scène A et on y met les choses nécessaire pour la scène B. Mais le problème, on ne peut pas passer le même Pane en paramètre pour plusieurs scene. La solution a ça fut d'appliquer le même effet de luminosité à chaque scene et indépendamment. On fait ça dans la classe Controller.java.
3. L'easter-egg qui consistait à faire apparaître une image et un son d'explosion dès qu'on clic sur une scène. Le premier souci était de faire apparaître une image à l'endroit du clic, on a tout essayé mais tout ce qu'on arrivait à faire était de faire apparaître l'image en haut à gauche de l'écran, mais jamais à l'endroit du clic. L'un des membres a donc proposé de changer d'optique et de juste changer le skin du curseur et le transformer en image d'explosion, et ça a marché ! Le deuxième souci fut de trouver un son libre d'explosion. Maintenant qu'on avait l'image et le son, le plus gros restait à faire : implémenter la chose. Et c'est là qu'on a eu notre troisième et avant dernier souci.
4. L'effet ne marchait que sur la scène d'options et pas sur les autres scènes. La solution fut d'implémenter une fonction qui prend en paramètre la scène et qui y applique l'effet souhaité, en l'occurrence le fait de changer l'image du curseur en image d'explosion et d'en jouer le son.
5. Le cinquieme et dernier souci fut de désactiver l'effet quand la case y étant associée présente dans les options était décoché. On n'y arrivait pas, on a donc décidé de remplacer l'effet explosion par un autre effet mais un qui ne fait rien de sorte à "override" le dernier.

4.5 Scene de jeu

Pour ce qui est de la représentation de la map ou évolue le personnage jouable, on a pas eu trop de soucis et cela pour 2 raisons :

1. Un membre de l'équipe a fait un prototype sur python ce qui nous a beaucoup aidé car il fallait juste traduire le python en java.
2. On était tous familier avec la représentation d'une carte de jeu sous forme de grille et donc de matrice.

Le problème majeur qu'on a eu avec la scène où se déroule une partie était le fait de bouger le joueur. Comme on a commencé par cette scène là, on était pas encore familiarisé avec les méthodes de setOnAction() et on comprenais pas comment les implémenter ni ou les implémenter. Après des recherches sur StackOverflow et GeeksForGeeks, on a trouvé comment faire et on a réussi l'implémentation. Le problème était qu'au début du projet on avait créé cette

méthode dans la classe Game.java or elle doit uniquement servir au lancement du jeu et rien d'autre. Quand on a changé l'endroit de définition de la méthode ça fonctionnait à nouveau mais on a un dernier souci. En effet, comme cette scène était la première qu'on a implémenté, quand on a voulu rajouter une scène, celle contenant les levels sélectionnable, la méthode ne marchait plus sur la scène affichant la partie en cours après le passage de la scène des level vers celle de la partie. La seule solution qu'on a trouvé c'est de la définir dans la classe de la scène affichant la partie nommée LevelScene.

4.6 Erreur

Un autre gros problème qu'on a eu c'était avec notre gestion des erreurs dans les "try-catch". On pensait initialement qu'un simple printStackTrace() dans le catch serait suffisant. C'est seulement après un entretien avec l'un des assistants qu'on a appris qu'il était préférable d'avoir un indicateur visuel pour les erreurs, car comme l'assistant nous a dit, chaque utilisateur n'est un programmeur qui ira chercher l' erreur dans le terminal. On a d'abord débattu sur le fait de créer notre propre type d'erreur et d'y mettre une méthode qui ferait apparaître à l'écran un message mais au final on a décidé d'utiliser la classe Alert() qui était idéale dans notre situation. On a implémenté une méthode dans la classe Controlleur.java qui ferait apparaître un message d'erreur à l'écran avec la classe Alert() tout ça avec un bouton permettant de quitter l'application après avoir lu l'erreur présentée.

5 Points forts et points faibles

5.1 Points forts

5.1.1 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.

5.1.2 Performances

Jeu fluide partout sauf dans le builder

5.1.3 Editeur de niveau

L'éditeur de niveau est un mode supplémentaire permettant d'imaginer et créer facilement ses propres niveaux de sokoban. Ce mode permet la création d'une map de maximum 15 cases de largeur et hauteur. L'éditeur est composé d'un menu contenant tous les types de cellules que l'utilisateur pourra placer sur sa map et d'une grille sur laquelle il pourra cliquer pour changer la cellule ciblée par la cellule choisie.

5.1.4 Enregistreur de niveau aleatoires

5.1.5 Menu d'options

Le menu permettant de modifier les options est relativement complet

5.1.6 Pack de textures

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

5.2 Points faibles

5.2.1 Generation aleatoire

5.2.2 Interface non responsive

5.2.3 Taille des niveaux

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

5.2.4 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.

6 Erreurs restantes

6.1 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.

6.2 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.

6.3 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.

7 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).

7.1 Open source

Nous avons décidé 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 sûr, 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.

7.2 FXML

7.3 Tools

sokoban.Engine.Tools contient des classes statiques. Elles sont utilisées à différents endroits du code, permettant de le simplifier en les appelant.

On peut par exemple lire un fichier, écrire un fichier, formater un fichier .xsb. Choses qui prennent de la place et qui auraient pollué le code principal. En les regroupant dans un package, le code est beaucoup plus structuré.

7.4 Easter eggs

8 Conclusion

Le projet nous a appris énormément de choses, gérer notre temps, travailler en groupe, la programmation, le crunch...
Ce travail de groupe était notre première expérience tous les trois et nous sommes satisfait de ce résultat imparfait. Faire face à des problèmes nous a montré à quel point l'organisation est importante et nous a forcé à penser les choses a l'avance. Cela nous a aussi permis de développer notre créativité et de

pouvoir la mettre en œuvre à notre image.

Ce projet nous a pris énormément de temps et nous avons mis de côté d'autres cours. Nous avons aussi fait une lourde erreur, après avoir participé au CSCBE notre rythme était cassé. Il nous a fallu plusieurs semaines pour nous remettre sérieusement au travail ce qui impacte la qualité de ce que nous avons pu développer.