Université de Mons
Faculté des sciences
Département d'Informatique

# Jeu du Sokoban

# Rapport de projet

*Professeur :*
Hadrien Mélot

*Auteur :*
François Vion
Godwill Louhou


Année académique 2020-2021

# 1 Introduction

Le Sokoban est un jeu de réflexion inventé au Japon en 1982 dans lequel  le joueur contrôle un personnage devant déplacer des boites sur des cases cibles. Il peut se déplacer dans les quatre directions et pousser une seule caisse à la fois. Une fois toutes les caisses rangées, le joueur peut passer au niveau suivant. (Cf. Wikipédia)

Pour le projet d'informatique de juin 2021, il a été demandé de réaliser un Sokoban en Java. Ce rapport présentera dans un premier temps la répartition du travail au sein du groupe. Dans un second temps, il détaillera la description de nos différentes parties ainsi que les choix effectués dans chacune d'entre elles. Ensuite, il listera les points positifs et négatifs du programme. Par après, il expliquera les erreurs connues que nous n'avons pas su régler. Par la suite, il parlera des aspects positifs et négatifs du projet. Par ailleurs, un guide du jeu reprendra toutes les consignes d'utilisation. Enfin, la conclusion reprendra les éléments clés de ce rapport.

# 2 Sommaire

# 3 Répartition du travail

Nous avons divisé la charge de travail en 3 parties : la logique, la génération de niveau et l'aspect graphique. François a commencé avec la partie logique pendant que Godwill s'occupait de la génération de niveau. Malheureusement notre trinôme s'est retiré du projet en mars, nous avons donc entamé à deux la partie graphique.

# 4 Description et choix

Nous avons fait le choix de mettre sur thème le jeu afin qu'il ne soit pas ennuyeux. Les habituelles boites ont été remplacées par des "**orcs**" et le personnage est un "**aventurier**". L'aventurier doit pousser tous les orcs sur les **flaques de lave** (les objectifs) pour gagner le niveau. Néanmoins, dans le rapport et dans le code nous utilisons des termes "techniques" comme boite, personnage, point, sol ou mur pour désigner les différents éléments du jeu.

## 4.1 Génération de niveau

Pour cette partie de notre jeu, il fallait implémenter un générateur de niveaux aléatoires réalisable, c'est-à-dire que le joueur puisse finir le niveau. Après avoir épluché de nombreux documents à ce sujet, notamment les travaux de I.Parberry et B.Kartal, il en est ressorti que l'utilisation d'algorithmes de recherches tels que le MCTS (le Monte Carlo Tree Search) pouvaient mener à un temps de génération assez élevé (pouvant aller jusqu'à 26 heures). Il fut donc décidé de générer les niveaux de la manière suivante :

- Le joueur renseigne la taille de la carte (qui est au minimum de 7 x7). Cela permettra de créer une map vide qui servira de base pour la génération.
- Le joueur renseigne le nombre de boites qu'il veut dans le niveau.
- Une carte est donc créée, et le personnage ainsi que les boites sont placées aléatoirement sur celle-ci.
- Le personnage que l'on appelle le « robot » va se déplacer de façon aléatoire avec un nombre prédéfini de mouvements sur la carte (selon la taille) et ainsi déplacer les caisses par la même occasion.
- Une fois tous les mouvements effectués, les positions finales des boites sont stockées en converties en cases objectives.
- Les boites et le personnage sont renvoyés à leur position initiale, et les cases non visitées sont changées en murs. Cela permet de garantir la faisabilité du niveau car les cases visitées par le robot restent disponibles pour le joueur, lui permettant dans le pire des cas de reproduire le chemin du robot pour finir le niveau.
- Enfin, la carte est exportée en fichier .xsb et stockée dans le fichier « custom levels », permettant ainsi au joueur d'aller récupérer le niveau ainsi généré.

Ainsi, en plus de la garantie de la faisabilité du niveau, cette méthode apporte un avantage autre : la rapidité de génération. Le temps de génération prend en moyenne moins d'une seconde.

**POINTS NÉGATIFS** : La qualité du niveau n'est pas garantie, de plus les cartes générées peuvent être assez triviales. La qualité des cartes baisse plus la taille de la carte augmente.

**POINTS POSITIFS** : Rapidité de génération, permet de choisir la taille et le nombre de boites du niveau.

## 4.2 Partie logique

On retrouve dans la partie logique :

- La création de la carte grâce à un fichier.
- Les déplacements du personnage et des boites.
- L'utilisation des fichiers .mov.
- La gestion des fichiers de cartes afin de savoir s'ils ont déjà été faits.

Pour réaliser tout cela, nous avons eu une approche orienté objet de sorte que la carte est représenté comme un objet Map qui possède :

- Un tableau 2 dimensions d'objets Case, ces Case ont comme attribut un caractère qui définit leur nature.
- Un objet Character qui représente le personnage.
- Une ArrayList d'objets Box.

On peut visualiser la carte comme un décor (le tableau d'objets Case) sur lequel se déplacent le personnage et les boites "fictivement". Ils ne font pas partie du tableau mais on peut savoir leur position grâce à leurs attributs communs : line et col (pour ligne et colonne) qui sont héritées de leur classe parent Mobile.

Pour les fichiers .mov, nous avons décidé de les encoder sous forme d'une suite de caractères "n", "s", "e" ou "w" pour north, south, east, west. Pour écrire un fichier .mov, un objet personnage ajoute chaque direction qu'il prend lorsqu'il se déplace dans une arraylist qui est simplement concaténée et mise dans un fichier. Pour la lecture, le personnage effectue tous les mouvements correspondants aux caractères du fichier.

Afin de bloquer et de débloquer les boutons dans le menu du choix de niveau, le jeu écrit sur la dernière ligne du fichier .xsb "Done". Il suffit donc de vérifier si la dernière ligne du niveau précédent est "Done" pour débloquer le bouton.

## 4.3 Interfaces graphique

### Interface du menu

L'interface du menu se présente sous la forme d'un menu avec plusieurs options :

- Jouer
- Partie Perso
- Importer
- Générateur de niveaux
- Options
- Quitter

Les fonctions de ces boutons sont détaillées dans la partie **GUIDE DU JEU** du rapport.

### Interface graphique du jeu

La partie "en jeu" est implémentée sous la forme d'un BorderPane. Dans le centre, on retrouve la carte, sur la droite, on a les boutons et sur le dessus il est affiché le nom du niveau.

La carte est un GridPane composé d'images correspondant à une case de la carte. Sur la droite, on retrouve une VBox qui contient les différents boutons de déplacement et ceux pour recommencer, sauvegarder et quitter. Lorsque l'on clique sur un des boutons de direction, cela effectue plusieurs actions :

- Le personnage se déplace (si possible) et peut engendrer le déplacement d'une boîte
- Un nouveau GridPane est généré en fonction du potentiel changement
- Le GridPane est inséré au centre d'un BorderPane en plus des boutons à droite et du nom de la carte en haut
- Le BorderPane est ajouté dans un Pane qui sert à mettre le fond d'écran
- Le Pane est défini comme nouvelle scène

Cette méthode provoque un souci de fluidité car à chaque mouvement, toutes les images doivent être à nouveau chargées et redimensionnées en fonction de la taille de la carte. Ce problème s'amplifie avec le nombre de cases à afficher. La solution serait de précharger toutes les images et d'ensuite les ajouter au GridPane a chaque déplacement mais nous n'avons pas eu le temps de l'implémenter.

Ensuite pour les 3 boutons, **recommencer**, **sauvegarder** et **quitter**:
- le bouton **recommencer** permet de recharger le jeu à partir de la carte de base. Si jamais le joueur recommence en ayant sauvegardé avant, la carte rechargée est celle de base et non celle sauvegardée.
- le bouton **sauvegarder** permet d'une part, de créer une sauvegarde des mouvements du joueur en créant un fichier .mov dans le dossier "moves" du jeu. D'autre part, une copie de l'état actuel du jeu sera créée, et prendra pour nom le nom du niveau suivi de "-save". Si le niveau avait déjà été sauvegardé, la sauvegarde sera écrasée par la nouvelle. Un des inconvénients de cette méthode est qu'il faut "recharger" la liste des niveaux une fois la sauvegarde faite pour accéder au niveau

sauvegardé (sauf si le jeu est relancé avant).
- le bouton **quitter** permet simplement de revenir dans la sélection des niveaux en changeant la “Scene” du jeu par celle de la sélection.

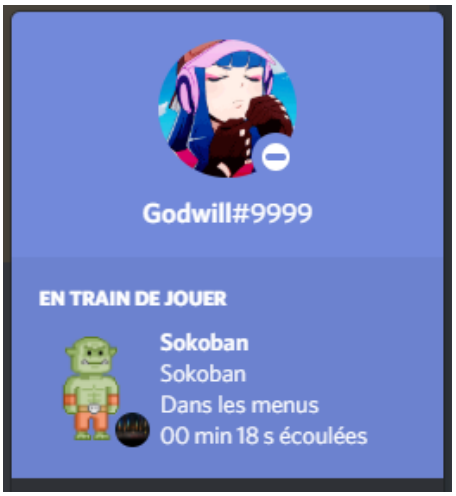
## Fonctionnalités supplémentaires

Les fonds d’écran

Les fonds d’écrans varient dans le jeu. Ils sont choisis aléatoirement à partir du dossier “ingamewallpapers” et sont donc sélectionnés aléatoirement au lancement de l’application, selon un thème « pixel art ». Ainsi, cela permet de rafraîchir l’environnement de jeu et éviter si possible une certaine redondance. (Le joueur pourra ainsi choisir d’importer ses propres fonds d’écran de taille 1280*720 si possible pour le jeu dans les OPTIONS).

Discord Rich Presence

Grâce à la documentation fournie par Discord, le Rich Présence est implémenté dans le jeu, permettant d’afficher le jeu et son état en tant que statut sur le réseau social Discord. Bien que ce ne soit pas une option principale, il a été jugé intéressant de l’ajouter à notre jeu étant donné que la majeure partie des joueurs de jeux vidéo utilisent ce réseau social au quotidien.




Musique

De la musique en boucle a été ajoutée au jeu, permettant de ne pas laisser le joueur jouer dans le silence total. Le volume peut être contrôlé à partir du menu Options du jeu ou bien directement depuis le mélangeur de volume du système d’exploitation du joueur. Il y a également un bouton permettant de changer de musique aléatoirement ainsi qu’une option pour en importer.

# 5 Points forts et points faibles du programme

**Points forts :**

- Tout d'abord, l'interface est très intuitive et assez fluide. La navigation ne pose aucun souci.
- Un effort a été apporté sur l'esthétique, notamment au niveau des animations.
- La génération des niveaux est simple d'utilisation, et les contrôles de jeu sont simples.
- Plusieurs options pratiques comme le fait de pouvoir recommencer ou sauvegarder le niveau sont disponibles pour l'utilisateur.
- L'ambiance du jeu est poussée, grâce au thème (aventurier, temple, jungle, etc), les visuels dans le thème "pixel art" et la musique de fond. Ce qui n'en fait pas un énième Sokoban.

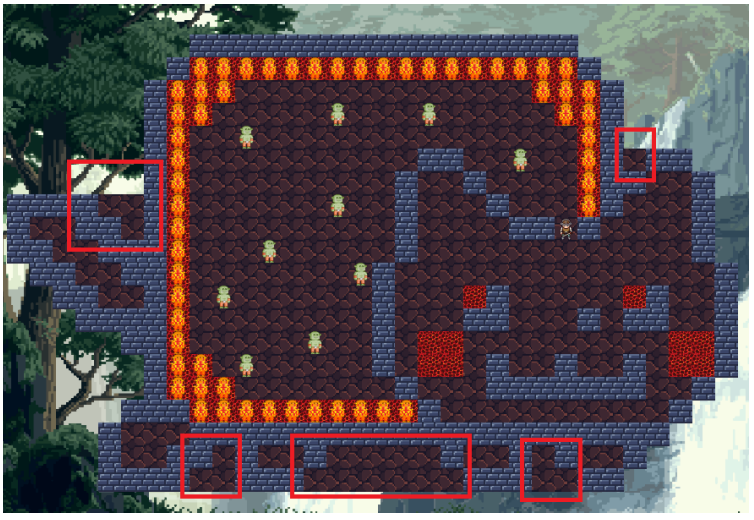
**Points faibles :**

- La génération de la carte ne garantit pas de niveaux intéressants ou difficiles en fonction de la taille ou du nombre de boites.
- Comme cité plus tôt, le jeu peut devenir lent si l'on charge une carte très grande
- Malgré nos efforts, le code ne respecte pas toutes les normes (encapsulations pas entière, variables en anglais ou en français)

# 6 Erreurs connues

Sur certaines cartes, on peut remarquer un léger problème d'affichage (voir photo). Cela est dû à la manière dont les " " d'un fichier .xsb sont transformées en case vide ou en case de sol. En effet, la création de la carte se fait en divisant la chaque ligne en 2, puis en allant de gauche à droite jusqu'au milieu, au premier mur rencontré, les " " deviennent du sol au lieu d'être du vide. Pareil pour la 2eme moitié qui commence à droite et va vers la gauche jusqu'au milieu.

Une solution serait de ne pas faire de différence entre sol et vide mais cela rendrait le jeu moins beau, surtout que ce problème est rare.



Pour la génération de niveau, il est possible, si le nombre de boites est trop élevé et/ou que la taille de la carte est trop petite, que cela mène à un crash de l'application. Une solution serait de trouver un ratio entre le nombre de boites et la taille du niveau et de restreindre automatiquement le choix du nombre de boites pour l'utilisateur.

Lors des tests sur les machines Linux, il été constaté que les niveaux du menu **jouer** peuvent ne pas être dans l'ordre attendu. Pour pallier ce problème, créer une méthode de tri pourrait permettre de réagencer les éléments dans la liste.

La musique semble ne pas fonctionner avec toutes les versions de Linux et peut poser problème lors de l'exécution du code sur certaines machines, ce qui est un souci de compatibilité connu. Cependant, sur les machines utilisées pour développer (windows), le jeu ainsi que sur les machines de la ESCHER de l'UMONS, la musique ne pose aucun problème. Il serait intéressant de pouvoir en faire la démonstration lors de la défense du projet sur nos machines personnelles afin de montrer le plein potentiel de notre application.

Lorsque le joueur sauvegarde le jeu et recommence, puis quitte le niveau, la carte revient à l'état sauvegardé. Il faudrait supprimer la sauvegarde si on détecte que le joueur a voulu recommencer le niveau.

# 7 Aspects Positif et Négatif du projet

Ce projet nous permet de développer dans un premier temps notre vision d'ensemble et de mettre de l'ordre dans nos pensées et nos idées. En effet, structurer et ordonner correctement les différentes étapes et fonctions à implémenter est primordial pour le développeur et le développement de ses applications.

De plus, nos connaissances se trouvent élargies grâce à la démarche de recherche (pour utiliser les différents modules) et s'approfondissent grâce aux multiples débogages et essais que nous avons dû faire.

Enfin, le travail d'équipe a lui aussi été renforcé, compétence essentielle pour le travail en entreprise dans le domaine de l'informatique. La distribution des tâches et la répartition du temps selon ces mêmes tâches nous a permis de mieux comprendre comment gérer son temps efficacement.

Néanmoins, la perte d'un collègue nous a posé pas mal de soucis, notamment pour ce qui est de devoir se réorganiser et se redistribuer les tâches. Celles-ci avaient été définies dès le début, et la perte de ce membre nous a handicapé pour la réalisation du projet. De plus, le fait de devoir suivre les cours en parallèle du projet n'est pas chose aisée et pousse à une gestion du temps assez stricte.

# 8 Guide du jeu

Après l'ouverture du jeu, l'utilisateur arrive sur le menu principal. Il peut ensuite choisir entre 6 options différentes.

- Tout d'abord, le bouton **JOUER** permet de lancer les 10 niveaux de base inclus dans le jeu. Pour y jouer, il suffit de cliquer sur le nom du niveau. Dans le jeu en lui-même, il y a les flèches directionnelles cliquables à la souris, ainsi que les boutons **Recommencer** et **Sauvegarder**.
Le bouton **Recommencer** va simplement remettre le joueur au début du niveau. Le bouton **Sauvegarder** permet de créer une sauvegarde de la partie en cours. Pour accéder à la sauvegarde, il suffit d'utiliser le bouton **Refresh** dans le menu de sélection des niveaux et ainsi jouer au niveau sauvegardé. La sauvegarde se supprime automatiquement lorsque le niveau est terminé. Pour accéder aux autres niveaux à partir du niveau 2, il faut finir le niveau précédent pour accéder aux suivants.
- Le bouton **PARTIE PERSONNALISÉE** permet de jouer aux niveaux importés et générés par l'utilisateur. Le joueur peut ajouter au maximum 30 niveaux supplémentaires et devra en supprimer pour en rajouter d'autres. Cliquer sur le nom du niveau permet d'y jouer.
- Le bouton **IMPORTER DES NIVEAUX** permet d'importer des niveaux personnalisés et les ajoute au jeu dans la section PARTIE PERSONNALISÉE.
- Le bouton **GÉNÉRATEUR DE NIVEAUX** permet d'accéder au menu de la génération de niveaux. Son menu comporte 4 sections personnalisables : la hauteur, la longueur, le nombre de boites et enfin le nom du niveau. La hauteur et la longueur doivent être comprises entre 7 et 20, et le nombre de boites entre 1 et 15. La taille recommandée est de 11x11 et 3 boites. Un nombre de boites trop élevé pour la taille du niveau peut mener à un crash du programme. Si le nom du niveau n'est pas spécifié, un nom sera généré automatiquement par le programme. Pour jouer aux niveaux générés, il suffit de rafraîchir la liste des niveaux personnalisés dans le menu éponyme.
- Le bouton **APPLIQUER UN .MOV** donne la possibilité à l'utilisateur d'appliquer un fichier .mov à un fichier .xsb de son choix. Pour cela, il faut sélectionner un fichier en entrée et sélectionner un fichier .mov . Il suffit ensuite de cliquer sur le bouton **APPLIQUER** pour récupérer le niveau dans le dossier "output levels".
- Le bouton **OPTIONS** permet d'accéder au réglages du volume et de la musique. En effet, le slider permet de contrôler le niveau de volume, et un bouton **CHANGE DE MUSIQUE** permet de changer la musique aléatoirement. Il est également possible d'importer de la musique ou des fonds d'écrans pour le jeu (pour une question d'optimisation, il est préconisé de ne pas utiliser de gifs ni d'images trop grandes, la taille optimale est de 1280x720).
- Des **tests gradle** sont disponibles. En lançant la commande **gradle test**, plusieurs actions seront effectuées :
    - le premier test permet de vérifier que le personnage ne passe pas à travers les murs. Pour cela, celui-ci effectue plusieurs fois à la suite un déplacement contre un mur à droite au niveau de la carte.
    - le deuxième test permet de vérifier que les boites ne passent pas à travers les murs non plus. Le personnage pousse une boîte plusieurs fois de suite contre un mur.
    - le troisième test permet de vérifier que les boites ne se passent pas au travers les unes des autres. Pour cela, le personnage pousse une boite contre l'autre plusieurs fois de suite dans la même direction.

- Il est possible d'utiliser une commande pour appliquer un .mov à un fichier .xsb directement depuis le terminal. Pour cela, il faut entrer la commande suivante :
**gradle applyMoves --args="fichier_entree.xsb fichier_mouvements.mov fichier_sortie.xsb"**
Il faut impérativement que le fichier en entrée soit présent dans le dossier "inputlevels" et que le fichier des mouvements soit dans le dossier "moves". Le fichier résultant est en sortie dans le fichier "outputlevels".
- une **task** gradle supplémentaire à été ajoutée permettant de jouer au jeu en version terminal. Pour se faire, il faut entrer la commande suivante :
**gradle loadFileXsb --args="nom_du_niveau.xsb"**
Le niveau doit se trouver dans le dossier "inputlevels" pour que cela fonctionne.

# 9 Conclusion

Pour conclure, le projet se veut simple et efficace tout en étant agréable visuellement. Malgré ses quelques défauts et erreurs, nous espérons que la qualité de celui-ci satisfera vos attentes. Selon nous, le projet nous semble complet et satisfait toutes les consignes données. Nous vous remercions pour le temps accordé à la lecture de ce rapport.

# 10 Sources

Images :

https://www.clipartmax.com/middle/m2i8K9K9A0H7b1N4_arrow-clip-art-at-clker-com-vector-online-royalty-free-left-arrow/

https://wallpaperaccess.com/pixel-art-desktop#related

http://gaurav.munjal.us/Universal-LPC-Spritesheet-Character-Generator/

https://www.shutterstock.com/fr/image-illustration/pixel-orc-8-bit-video-game-298554893

https://www.shutterstock.com/fr/image-vector/pixel-art-lava-seamless-texture-vector-1674474640

https://www.shutterstock.com/fr/image-vector/textures-tile-seamless-pattern-mega-set-1510465106

Retouches et modifications des images faites sur Gimp

Code :

http://motion.cs.umn.edu/pub/SokobanMCTS/DataDrivenSokobanMCTS.pdf

http://ianparberry.com/techreports/LARC-2011-01.pdf

http://www.puzzlegen.tysonsorensen.net/

https://youtu.be/7hZVRDxpbCE

https://github.com/patrickTingen/Sokoban

https://sokoban.pagesperso-orange.fr/webpages/sokoban.htm

Cours de Programmation et Algorithmique 2 de Monsieur Quoitin

Javadoc de JavaFX

Musique :

YouTube Audio Library