**Speedy Roadie**

Louis Dhanis, Corentin Dachy

Projet d’informatique, mai 2017

Table des matières

[*1.* *Introduction* 1](#_Toc482439726)

[2. Répartition du travail 1](#_Toc482439727)

[3. Divers choix personnels 1](#_Toc482439728)

[*a.* *Thème du jeu et mode histoire* 1](#_Toc482439729)

[4. Points forts 2](#_Toc482439730)

[5. Faiblesses 2](#_Toc482439731)

[6. Erreurs connues 2](#_Toc482439732)

[7. Apports positifs de ce projet 2](#_Toc482439733)

[8. Guide utilisateur de Speedy Roadie 2](#_Toc482439734)

# *Introduction*

1. Répartition du travail

Afin de se répartir les tâches équitablement et selon les préférences de chacun, Corentin Dachy s’est occupé de la partie algorithmique back-end du travail tandis que je me suis attelé à la partie graphique du Sokoban.

1. Divers choix personnels

Divers choix ont ponctué le développement de notre jeu. Tout du long, nous avons discuté les différents aspects de la mise en œuvre de chaque élément. Dans cette section nous allons aborder les choix les plus importants que nous avons effectués pour ce projet.

* Thème du jeu et mode histoire

En tant qu’amoureux de l’univers de la scène metal, nous avons vu en ce projet un moyen de raconter une histoire. En nous concertant nous nous sommes dit que le sokoban pouvait représenter une scène où un roadie (personne qui travaille pour un groupe et qui aide à l’organisation de la scène) nommé Speedy, devrait déplacer des caisses contenant des instruments de musique pour les ranger afin rétablir l’ordre des choses dans le metal, ce dernier ayant été corrompu par un circle pit maléfique.

L’interface graphique devait suivre avec le thème. Corentin a donc demandé à <INSERER NOM DE TON AMI>, un graphiste doué dans le pixel art, de dessiner les différents sprites (éléments graphiques) du jeu.

1. Points forts

* Sauvegardes en continu

Une fois que le joueur a commencé une partie, on en enregistre une copie dans le dossier PermanSave au format .xsb (format de fichier des plateaux de jeu Sokoban). Dans ce même répertoire, on enregistre le fichier .mov (format de jeu de l'historique des mouvement du joueur) qui se met à jour automatiquement avec chaque mouvement du joueur.

On peut donc garder une sauvegarde en continu de l'état d'avancement du joueur dans le niveau courant. Si le joueur quitte inopinément la partie et relance le jeu, une petite fenêtre lui demandera s'il souhaite continuer sa partie à l'endroit où il était.

* Lecture de .mov en mode cinématique

Si le joueur charge un fichier .xsb et un fichier .mov, le jeu "lira" le fichier de mouvements comme une série d'instructions déplaçant progressivement le joueur sur le plateau avec un intervalle de 0,25 secondes entre chaque déplacement.

Pour ce faire, on a implémenté une classe étendant ActionListener nommée ClockListener qui s'exécute périodiquement grâce à un Timer.

On ôte le focus de l'interface graphique afin d'éviter que le joueur fausse les déplacements en déplaçant le personnage pendant la lecture du .mov

Une fois le déplacement terminé, on rend le focus à l'élément graphique représentant la partie courante et le joueur peut reprendre la partie.

* Mode histoire

Comme expliqué dans le point 3 de ce document, nous avons mis en place un mode histoire où le personnage principal du jeu, Speedy, passe de scènes en scènes.

Pour ce mode, nous avons également mis en place une sauvegarde de l'état d'avancement du personnage dans un fichier sauvegarde.xml se trouvant dans le dossier "ClassicMode".

Dans ce fichier sont enregistrés les textes s'affichant avant chaque niveau, l'état d'avancement (si on peut faire un niveau, l'attribut "doable" du fichier de sauvegarde est à la valeur "true" et donc le bouton dans la liste des niveaux du menu ClassicMode est cliquable, sinon il est grisé) et le nombre de pas nécessaires au joueur pour terminer le niveau.

Afin de réinitialiser la sauvegarde sans trop de difficultés (pour éviter que le joueur ait accès aux niveaux suivants avant d'avoir les niveaux précédents).

Au lancement du jeu, on initialise une liste simplement chaînée où chaque élément est un niveau. La classe des maillons de la chaîne est LevelNode et celle de la chaîne en elle-même est StoryMode.

Les LevelNode permettent d'instancier les boutons du menu et ont donc la faisabilité du niveau (boolean doable). Sont également stockées en variables de classe le niveau (de type Game) le texte à afficher avant le niveau et son identifiant dans le XSB (int id). Les maillons contiennent aussi le maillon de la chaîne les suivant.

1. Faiblesses

* Mode histoire

A chaque fin de niveau, on affiche un bouton permettant de passer au niveau suivant.

Or, pour savoir vers quel niveau rediriger, on utilise une classe étendant JButton dans laquelle on passe en paramètre l'id du niveau. Pour récupérer le niveau correspondant à cet id, on doit parcourir toute la chaîne. La complexité de cette opération est en O(n).

1. Erreurs connues

* Lecture de .mov en mode cinématique

En essayant notre SpeedyRoadie sur différentes plateformes et sur des ordinateurs ayant différentes configurations, nous avons remarqué que la lecture du .mov en mode cinématique pouvait poser problème.

Il arrive que parfois des déplacements ne soient pas pris en compte, faussant la cinématique. Nous avons recherché la raison de cette erreur mais nous ne sommes pas parvenus à en trouver l'origine ou la solution.

* Exécution sans Apache ant et ressources graphiques

Notre SpeedyRoadie a été développé dans l'optique d'être exécuté via les commandes ant build, ant run, ant clean, ant test et ant reset (voir point 4, Mode histoire).

Si le jeu est exécuté via les commandes java directement (et donc depuis un autre dossier que le dossier où se trouve le fichier build.xml) le chemin vers les images de fond (logo de bienvenue, texture de boutons, etc.) ne fonctionne pas car c'est un chemin relatif au dossier racine.

1. Apports positifs de ce projet
2. Guide utilisateur de Speedy Roadie