Projet d'informatique

UMONS – Année académique 2020 – 2021

Le contenu de ce document est présenté lors de la première séance du projet, le mardi 9 février 2021. Il vous est conseillé de relire attentivement ce document et d'y revenir au besoin.

1 Objectifs du projet

Les cinq principaux objectifs de ce projet sont les suivants :

1. Participer au développement d'un logiciel de taille conséquente (représentant environ 150 heures de travail, à répartir entre 3 étudiants).
2. Appliquer les notions d'*algorithmique* vues au cours « Programmation et Algorithmique 1 ».
3. Appliquer les notions de programmation *orientée objet* vues au cours « Programmation et Algorithmique 2 ».
4. S'initier aux bonnes méthodes / habitudes utiles au développement d'un projet logiciel (design modulaire des classes, éviter la redondance du code, documentation correcte, création et utilisation de tests, création de packages, etc.).
5. Apprendre à gérer son temps, à travailler en groupe et à s'organiser.

Gardez ces objectifs à l'esprit : ils nous servent également de critères pour vous évaluer (voir ci-dessous).

Contenu

2 Déroulement général du projet

Le but final de ce projet est de réaliser une application graphique en Java permettant de jouer au jeu de Sokoban. Le jeu de Sokoban est un jeu de type « puzzle » dans lequel le joueur doit déplacer des caisses dans un entrepôt de telle sorte à placer les caisses sur des cases cibles. L'application devra permettre de sauvegarder et charger l'historique d'une partie; de charger des niveaux créés par l'utilisateur; et de générer automatiquement des niveaux. Une fois que les fonctionnalités de base exigées seront correctement mises en place, vous aurez le loisir de personnaliser et d'étendre votre projet (ce qui est encouragé, notamment via l'évaluation comme expliqué ci-dessous).

Le déroulement du projet est particulier. Vous allez devoir le réaliser petit à petit, mais sans perdre de temps et en vous répartissant correctement les tâches. Vous devrez faire des choix consciencieux lors de la conception. Pour ce faire, vous utiliserez au mieux les notions de la programmation *orientée objet* vues au cours de « Programmation et Algorithmique 2 ». Ce cours étant donné en parallèle au projet, nous vous proposerons un calendrier adapté et nous organiserons des séances particulières pour vous aider sur un thème précis (voir ci-dessous).

2.1 Séances à l'horaire et travail en dehors de celles-ci

Parmi les heures prévues à l'horaire, il y a deux types de séances :

— **Séances spéciales** : séances (obligatoires) durant lesquelles un assistant présente un sujet directement utile pour la réalisation du projet (par ex., une séance sur la conception d'interfaces graphiques). La première séance de ce type aura lieu le mercredi 17 février 2021.

— **Séances questions** : séances (facultatives) où un assistant est présent pour répondre aux questions. Vos questions doivent être préparées à l'avance car l'assistant passera une et une seule fois parmi chaque groupe pour y répondre !

Un calendrier reprenant le type des séances est disponible sur moodle.

Le nombre d'heures nécessaires pour réaliser le projet est important (environ 60h par personne) et les heures indiquées dans l'horaire ne sont donc pas suffisantes : ce sont des heures destinées à vous donner des consignes ou des conseils, et permettant de répondre à vos questions. Pour mener à bien votre projet, vous devrez les compléter par des heures de travail en dehors des séances ! Essayez également de vous répartir les tâches de manière équilibrée et de vous organiser au mieux au sein de votre groupe. La gestion de votre temps est une des difficultés de ce projet : ne laissez pas le retard s'accumuler.

2.2 Aide et réponses aux questions : séances et forum

Nous ne répondons pas aux questions par email. Par contre, outre les séances « questions » décrites ci-dessus, un **forum** est disponible sur moodle pour y poser vos questions et pour vous entraider entre étudiants. L'utilisation du forum doit se faire de manière constructive et courtoise :

— Soignez la manière de rédiger vos questions et vos réponses pour qu'elles soient claires et compréhensibles.

— Vous êtes autorisés à répondre aux autres étudiants mais dans le but de les faire réfléchir ou en donnant des pointeurs vers des références utiles. Ne donnez donc pas de solution

toute faite. Ne postez pas non plus du code qui sera utilisé dans votre projet.
— L'équipe enseignante ne donnera a priori pas de réponse, sauf si la question (et sa réponse) s'avère constituer un éclaircissement utile à l'ensemble des étudiants. Même dans ce cas, n'attendez pas de réponse immédiate : il peut être nécessaire de discuter des éclaircissements à donner avant de les publier.

L'équipe enseignante peut modifier (corriger) ou supprimer vos messages si cela s'avère nécessaire. Une utilisation abusive du forum entraînera la fermeture de celui-ci.

Pour éviter que les choses importantes soient réalisées en dernière minute, nous ne répondrons plus à vos questions lors des derniers jours avant la remise du projet. En pratique, la dernière séance pour poser vos questions aura lieu le lundi 10 mai 2021. Vous êtes prévenus plusieurs mois à l'avance : prenez donc vos dispositions !

2.3 Soumission et défense du projet final

En fin d'année, vous rendrez un rapport écrit, votre code source et défendrez oralement votre projet (la défense orale a lieu pendant la session d'examen de juin). L'évaluation se base sur tous ces points (voir ci-dessous).

En ce qui concerne la rédaction du rapport, sachez que la présentation et l'orthographe seront considérées lors de l'évaluation. Pour vous aider, le document « Eléments de rédaction scientifique en informatique » est mis à votre disposition sur e-learning et rassemble quelques conseils [1].

La défense orale se fera par groupe lors de la session de juin. Elle commencera par une démonstration du programme (environ 10 minutes) et se terminera pas une série de questions posées de manière individuelle ou collective. Même si une partie du programme a été réalisée par une personne du groupe, les deux membres doivent être capables de répondre aux questions. La date et le lieu vous seront communiqués ultérieurement (via les horaires d'examens de la première session). Notez que les deux étudiants d'un groupe seront interrogés de manière équilibrée lors de la défense et n'obtiendront pas forcément la même note.

3 Aspects administratifs et consignes

3.1 Composition des groupes

Le projet est réalisé par **groupes de 3 étudiants**. Vous nous communiquerez la composition de votre groupe pour le jeudi 25 février 2021 au plus tard via Moodle. Nous nous attendons à ce que le projet soit réalisé à trois et à vous voir tous les trois présents lors de la défense. **Vous devez gérer vous même les soucis éventuels (abandon, manque d'implication de l'un, etc.).** La gestion correcte du groupe fait partie des objectifs à atteindre et vous serez amenés à faire plusieurs travaux de groupes tout au long de vos études. Pour arriver au terme de votre projet nous vous encourageons à vous répartir les tâches et à communiquer entre vous de façon efficace. Il vous sera demandé de préciser comment vous vous êtes réparti les tâches dans le rapport.

1. Tout au long de vos études vous serez amenés à rédiger différents rapports, jusqu'à la rédaction de votre mémoire.

3.2 Rapport

Le rapport final doit contenir :
— la répartition des tâches au sein du groupe ;
— une description argumentée des choix personnels effectués ;
— les points forts de votre projet (fonctionnalités supplémentaires, optimisation, complexité,...) ;
— les points faibles de votre projet (vitesse d'exécution lente, faiblesses de certains algorithmes,...) ;
— les différentes erreurs connues du programme. Les erreurs rencontrées lors de nos tests et qui ne sont pas répertoriées dans le rapport seront considérées plus sévèrement ;
— les apports positifs et/ou négatifs de ce projet ; et
— un mini guide utilisateur précisant la manière dont l'application peut être utilisée (par exemple pour sauver ou charger une partie).

En plus des éléments repris ci-dessus, vous pouvez ajouter tout élément qui vous semble utile comme par exemple un *diagramme de classes UML* (cf. cours de « Programmation et Algorithmique 2 »).

Le rapport devra être remis en PDF. **Attention à votre orthographe, elle sera considérée lors de l'évaluation.** Pour rappel, une note contenant des conseils de rédaction est disponible sur la page moodle du projet.

Pour vous aider à réaliser votre rapport, nous vous conseillons de maintenir un « journal de bord » tout au long du projet (concrètement, cela peut être un petit carnet ou un simple fichier de texte). Celui-ci vous permettra d'y noter progressivement vos idées, d'y indiquer la répartition des tâches (qui réalise quelle tâche au sein de votre groupe). Egalement, si quelque chose dans l'énoncé vous paraît ambigu ou trop peu défini, vous devrez faire un choix (en le justifiant) : notez ce choix et vos arguments dans votre journal puis dans votre rapport.

3.3 Code source

Voici les consignes à respecter concernant le code source.
— Les noms des interfaces, classes et méthodes que vous créerez seront en **anglais**.
— Le code doit obligatoirement être **documenté** en utilisant la javadoc de base (`@return`, `@param` et `@throws`).
— Les commentaires et la documentation peuvent être en français ou en anglais (mais restez cohérents une fois la langue choisie).
— Vous devrez inclure des tests unités pour **au moins une partie précise de votre projet** (voir section **??**).
— Pour l'évaluation et la défense orale, votre code sera exécuté sur une machine de l'université et seule la version soumise sera prise en compte. La description technique de la machine utilisée correspond à une machine des salles de TPs (Salles Russell ou Vaughan, De Vinci, sous Linux).
— Vous devez utiliser *Gradle* (voir Section 4) pour simplifier et automatiser le processus de compilation et d'exécution de votre code source : en ouvrant votre archive les 4 commandes suivantes doivent impérativement avoir le comportement attendu :
  1. `gradle clean` : supprime tous les fichiers `.class`, s'il y en a ;
  2. `gradle build` : compile votre programme ;
  3. `gradle run` : exécute votre programme ;

4. `gradle test` : lance les tests unités.

3.4 Défense du projet

Lors de la défense orale du projet (pendant la session de juin), votre application sera exécutée sur une machine des salles Russell ou Vaughan et non pas sur un ordinateur personnel.

3.5 Plagiat et triche

Le *plagiat* consiste à s'approprier comme personnel du texte, une image ou du code réalisé par une autre personne (en ce compris un texte traduit), sans le préciser de manière explicite. Vous pouvez baser vos arguments sur des éléments que vous avez lu, mais vous devez citer vos sources. S'appuyer sur des résultats connus pour en développer de nouveaux est d'ailleurs à la base de la démarche scientifique. Il faut simplement être honnête et très clair sur ces points, mais le plagiat est donc simple à éviter.

D'autre part, le plagiat est **strictement interdit** à l'université et peut entraîner des sanctions graves allant beaucoup plus loin que le simple échec au projet (cf. règlement des études et page dédiée à ce sujet sur le site de l'université [2]). Vous devez soumettre vos fichiers via moodle (et donc pas par email) où ils subiront une détection automatique du plagiat.

Le projet étant individuel, le fait que deux groupes différents ont manifestement échangé du code sera donc considéré comme de la triche et sera sanctionné en conséquence (pour les deux groupes).

3.6 Notes de présence et absences

En ce qui concerne le projet en première session, vous obtiendrez
- une **note entre 0 et 20** si le projet est soumis, recevable *et* défendu oralement pendant la session ;
- une **note de 0 sur 20** si le projet n'est pas recevable pour au moins un des critères énoncés en Section 5 ;
- une **note de présence** si le projet n'est pas déposé mais que vous signalez votre intention de ne pas le soumettre en envoyant un email à `hadrien.melot@umons.ac.be` avant le vendredi 14 mai 2021 à midi (vous recevrez un accusé de réception).

Dans *tous les autres cas* (soumis mais pas défendu, pas soumis sans nous prévenir, etc.), vous serez noté **absent** pour le projet d'informatique.

4 Gradle

Il vous est demandé de fournir un fichier build.gradle afin de permettre l'utilisation de *Gradle* [3] pour compiler, exécuter et tester votre projet.

2. `https://sharepoint1.umons.ac.be/EN2/universite/admin/aff_academiques/Pedagogie_Qualite/Plagiat/plagiat.html`
3. `https://gradle.org/`

En ouvrant votre archive dans une console, les 4 commandes suivantes doivent impérativement avoir le comportement attendu :

1. `gradle clean` : supprime tous les fichiers `.class`, s'il y en a ;
2. `gradle build` : compile votre programme ;
3. `gradle run` : exécute votre programme ;
4. `gradle test` : lance les tests unités.

Pour cela, vous devrez respecter la structure imposée par *Gradle* :

```
<votre projet> ..... dossier de votre projet
├── build.gradle
└── src
    ├── main
    │   ├── java ..... contient les fichiers .java
    │   └── resources ..... contient les resources (images, ...)
    └── test
        └── java ..... contient les fichiers .java pour les tests
```

5 Checklist des éléments indispensables pour que votre projet soit corrigé !

Vérifiez scrupuleusement les éléments repris ci-dessous qui décrivent comment nous déterminerons si votre projet est recevable. **Un projet non recevable n'est pas corrigé et entraîne une note de 0 / 20.** Notez que ce sont des choses toutes simples à vérifier et qu'aucune exception ne sera faite, quelles que soient les circonstances. Ce serait vraiment dommage de ne pas pouvoir présenter votre projet à cause d'un des points ci-dessous !

1. *Deadline.* La date limite de remise est le **vendredi 14 mai 2021 à 12h**. **La plateforme n'acceptera pas de retard.** Aucun ajout ni aucune modification du projet ne sera acceptée au-delà de cette date.
2. *Compilation.* Pour tester si votre projet est recevable nous procéderons à l'exécution des 3 commandes suivantes, dans cet ordre : `gradle clean`, `gradle build`, `gradle run`. Votre code doit pouvoir être compilé et exécuté sur une machine de l'université et seule la version soumise sera prise en compte. La description technique de la machine correspond à une machine des salles Russell et Vaughan (sous **Linux (Ubuntu)**). La compilation **ne peut pas produire d'erreur** empêchant la création des fichiers `.class` (warnings autorisés). En d'autres mots, nous devons être capable d'exécuter le programme sans modifier le code.
3. *Archive.* Le travail doit être livré sous la forme d'une archive (`.zip` ou un format libre) portant, en majuscules uniquement, votre nom. L'archive **doit** contenir un répertoire portant ce même nom. Ce répertoire comporte :
   a) le rapport au format **PDF** ;
   b) le code de votre application (en respectant la structure imposée par *Gradle* comme expliqué dans la Section 4) ;
   c) un fichier `build.gradle` (*Gradle*) pour permettre d'utiliser les quatre commandes `gradle` décrites ci-dessus.

Ne sont requis que le code source (fichiers `.java`), les tests unités, ainsi que les fichiers nécessaires à la compilation et l'exécution (par ex. images). Votre ne devez **pas** inclure les fichiers compilés `.class`. Si vous souhaitez fournir d'autres éléments, placez-les dans un sous-répertoire nommé `misc` et indiquez dans un fichier `README.txt` les détails de ces différents éléments.

6 Conseils

Comment éviter le stress et m'assurer à temps que mon projet sera recevable ?

1. *Mettez la priorité là où il le faut.* Assurez-vous d'avoir en priorité une version simple mais qui répond à nos exigences et qui implémente les fonctionnalités demandées. Ensuite, s'il vous reste du temps, vous pourrez peaufiner le côté esthétique ou améliorer d'autres points de détails.
2. *Rédaction.* Rédiger prend du temps ! Le rapport ne doit pas être plus long que nécessaire mais doit contenir ce qui est attendu (voir section 3.2), et le présenter de manière adéquate. Commencez votre rapport au plus vite (pourquoi pas dès maintenant ?). Soignez la forme autant que le contenu et surtout : citez vos sources et évitez le plagiat (voir ci-dessus).
3. *Test réel.* Faites un test dans la salle Escher en utilisant exactement ce que nous recevrons :
   — ouvrez sur une machine du Escher l'archive telle que vous comptez la soumettre ;
   — testez la compilation et l'exécution.
4. *Soyez prévoyants.* Prévoyez de soumettre une version préliminaire au moins 24 heures à l'avance pour éviter toute mauvaise surprise (problèmes de connexion, etc.). C'est toujours mieux d'être évalué sur cette version sans doute imparfaite [4], plutôt que de ne pas être jugé du tout !
5. *Réfléchissez bien à votre design.* Ce projet comporte des éléments (comme les différentes pièces du jeu de Stratego) qui permettent une exploitation intense de certains concepts de la programmation orientée objet (comme la notion d'héritage ou les interfaces). Etant donné que ceci est votre premier projet de la sorte, trouver la bonne utilisation de ces notions ne sera pas facile du premier coup. N'hésitez pas à penser à plusieurs solutions et à les comparer avant d'aller trop loin dans votre développement. N'hésitez pas non plus à recommencer votre design si vous vous rendez compte qu'il n'est pas bien adapté. Par exemple, si vous constatez que vous avez des méthodes qui contiennent des instructions conditionnelles avec un grand nombre de branches ("if-else if-else if-...-else"), vous n'avez sans doute pas exploité suffisamment le polymorphisme. Nous sommes là pour vous aider et discuter de vos idées.
6. *Petit à petit, l'oiseau fait son nid.* Au début du projet, vous n'aurez pas encore à votre disposition toutes les connaissances en Java pour implémenter complètement votre projet. Ce n'est pas grave : faites une première version simplifiée avec des choses que vous savez faire. Par exemple, vous pourriez commencer par une première version d'un Stratego très simplifié en mode "texte" (dans la console) où il n'y a qu'une seule pièce qui peut se déplacer sur le plateau...

4. Si vous êtes trop perfectionniste, quelle version sera vraiment parfaite ?

7 Evaluation de votre projet

Sur quels critères vais-je être évalué ?

Nous nous basons sur le rapport, le code et la défense orale. Pour chacun de ces éléments, une série de critères et de questions sont utilisés pour évaluer la qualité d'un projet.

1. *Rapport.*
   (a) Forme : orthographe, style adapté, structure, clarté.
   (b) Contenu : le contenu est-il complet et informatif ? Pour rappel, votre rapport doit contenir :
   — Une description argumentée des choix personnels effectués ; en particulier vous expliquerez les idées exploitées pour décrire vos IA ;
   — Les points forts de votre projet (fonctionnalités supplémentaires, optimisation, complexité,...) ;
   — Les points faibles de votre projet (vitesse d'exécution lente, faiblesses de certains algorithmes,...) ;
   — Les différentes erreurs connues du programme. Les erreurs rencontrées lors de nos tests et qui ne sont pas répertoriées dans le rapport seront considérées plus sévèrement ; et
   — Les apports positifs et/ou négatifs du projet.
2. *Code.*
   (a) Les fonctionnalités de base sont-elles respectées ?
   (b) Des éléments supplémentaires aux fonctionnalités de base ont-ils été ajoutés ?
   (c) Les tests unités (pour au moins une partie du projet) sont-ils complets et pertinents ?
   (d) Les algorithmes ont-ils été écrits avec un souci d'efficacité ?
   (e) Les notions de la programmation Orientée Objet (héritage, interface, exceptions, etc.) sont-elles utilisées de manière correcte, élégante et pertinente ?
   (f) La documentation (javadoc de base : `@return`, `@param` et `@throws`) est-elle présente ?
   (g) La documentation (javadoc) est-elle pertinente et claire ?
   (h) Le code est-il de bonne qualité et bien organisé ?
   — lisibilité ;
   — non redondance ;
   — design modulaire ;
   — organisation en packages ;
   — respect des conventions, etc.
3. *Défense orale.*
   (a) Réponses aux questions.
   (b) Intérêt de la démonstration.
4. *Respect des consignes.* Présence aux séances obligatoires, etc.

Rappel : les étudiants au sein d'un groupe n'obtiendront pas systématiquement la même note. Cela dépendra des réponses aux questions ou d'autres critères objectifs.

8 Prototype en Python

Avant de se lancer dans un gros projet, il est utile de réaliser un prototype. Un prototype est une version rapide d'un projet, souvent implémentée sans interface graphique et dans un langage syntaxiquement simple (comme le Python). Il sert à dégrossir les choses pour mieux comprendre le sujet, commencer à réfléchir au design des classes, se rendre compte de ce qui pourrait marcher ou pas, etc. Il n'est pas nécessaire que le prototype soit écrit dans le langage final, puisque son but est de mieux appréhender le projet et de concrétiser les choses. De plus, étant donné que le cours de « Programmation et Algorithmique 2 » qui vous apprendra le Java est donné en parallèle au projet lors de ce quadrimestre, cela vous permettra de commencer rapidement puisque vous connaissez déjà le Python.

Pour ces raisons, **nous vous conseillons fortement d'écrire un prototype en Python avant de commencer votre projet en Java**. Il serait intéressant que vous indiquiez dans votre rapport si celui a été utile et s'il vous a permis par exemple de mieux concevoir vos classes.

9 Description de votre application

Il est attendu que chaque groupe d'étudiants conçoive une application logicielle en essayant au maximum d'appliquer les concepts et principes vus au cours de l'année, principalement les concepts orienté objet (héritage, interfaces, exceptions, ...). A ce sujet, la section 7 donne la liste des critères qui nous serviront à évaluer votre projet.

Votre application doit permettre, entre autres, de jouer à Sokoban via une interface graphique et en utilisant la souris (et pourquoi pas les flèches directionnelles). La page wikipedia `https://fr.wikipedia.org/wiki/Sokoban` présente ce jeu et en explique le principe. Il existe plusieurs sites web vous permettant de jouer en ligne et de vous familiariser avec le jeu.

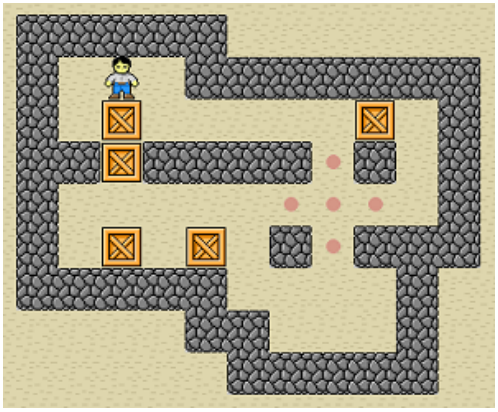

FIGURE 1 – Un jeu de Sokoban

Les différentes options de jeu (voir ci-dessous) pourront être sélectionnées via des menus.

L'intérêt d'un jeu tel que le Sokoban est d'avoir la possibilité de jouer à différents niveaux de difficulté progressive et qui soient réalisables. Dans la suite de ce texte, nous dirons qu'un niveau du jeu qui est possible ou réalisable est un niveau *jouable*.

9.1 Niveaux du jeu de Sokoban et fichiers `.xsb`

Vous mettrez en place un système permettant de charger des niveaux à partir d'un fichier de texte au format `.xsb`. Ce format est défini sur la page wikipedia donnée ci-dessus et permet de décrire simplement un niveau du jeu. De plus, vous offrirez trois manières d'accéder à ou de construire des niveaux jouables.

1. Le premier ensemble de niveaux jouables sera directement accessible en lançant le jeu et contiendra au moins 10 niveaux jouables de difficulté progressive. Le niveau le plus simple sera accessible dès le départ et les niveaux suivants seront accessibles petit à petit, à condition de réussir le niveau précédant. Vos niveaux seront définis via des fichiers `.xsb` stockés dans un répertoire `levels` et pour lesquels vous établirez une convention de nommage permettant à votre programme de savoir quel est l'ordre de difficulté des niveaux (par exemple, `level_`x`.xsb` où x est un entier représentant le niveau de difficulté). Il existe des
   listes de niveaux disponibles en ligne que vous pouvez utiliser (notez néanmoins leur origine dans votre rapport).
2. Vous offrirez également la possibilité à l'utilisateur d'ajouter ses propres niveaux dans le jeu via des fichiers `.xsb`. Vous êtes libres de procéder comme bon vous semble mais vous devez expliquez dans votre rapport comment l'utilisateur peut ajouter ses niveaux.
3. Vous mettrez en place un générateur automatique de niveaux jouables le plus souple possible pour l'utilisateur : par exemple, via un menu où l'utilisateur peut choisir une taille pour la grille du jeu, un niveau de difficulté, etc. Attention : vos niveaux doivent être jouables ! Cette étape constitue donc un problème algorithmique intéressant. Vous expliciterez votre solution dans le rapport : comment procédez-vous ? Qu'est-ce qui assure que les niveaux générés soient jouables ? etc.

9.2 Historique d'une partie, fichiers `.mov` et mode console

Il doit être possible, à partir d'une situation initiale définie via un fichier `.xsb`, d'enregistrer l'historique d'une partie en cours ou terminée. Pour ce faire votre programme créera un fichier `.mov` (pour les mouvements) qui reprendra la liste des mouvements joués. Vous êtes libre de définir la manière dont ces fichiers `.mov` sont créés (sérialisation, fichier texte, etc.).

L'utilisateur pourra demander à votre programme d'appliquer un fichier `.mov` sur un fichier `.xsb`, ce qui permettrait par exemple de sauver les étapes de résolution d'un niveau afin de le rejouer automatiquement dans le logiciel. Vous pouvez permettre cela via l'interface graphique mais il est impératif que cette application d'un fichier `.mov` soit possible en mode console. Concrètement, la commande à lancer dans la console aura la forme suivante :

```
java sokoban input.xsb test.mov output.xsb
```

Celle-ci aura pour effet d'appliquer le fichier `test.mov` sur le fichier `input.xsb` et doit produire en sortie un fichier `output.xsb` qui correspond au résultat. N'oubliez pas de préciser dans votre rapport quelle est le nom de la commande à lancer pour le mode console et dans quel répertoire il faut lancer celle-ci.

9.3 Tests unités

Vous devez **obligatoirement** fournir quelques tests unités. Chacun de ces tests prendra la forme d'un scénario qui fonctionne selon la procédure suivante :

— charger un fichier `.xsb` ;
— charger un fichier au format `.mov` contenant quelques mouvements ;
— appliquer les mouvement du fichier `.mov` au fichier `.xsb` ;
— vérifier que le jeu obtenu correspond bien à la situation attendue (ce résultat attendu peut être stocké également sous la forme d'un fichier `.xsb`).

Voici quelques idées de tests unités que les scénarios tels que décrits ci-dessus pourraient vérifier :

— déplacer une caisse vers un emplacement libre ;
— se déplacer sur une case libre ;
— se déplacer "contre un mur" (ce qui doit revenir à ne pas se déplacer) ;
— bouger une caisse contre un mur (ce qui doit revenir à ne pas la bouger) ;
— bouger une caisse sur une case de chargement.

Vous pouvez tester de manière facultative tout autre comportement de votre programme.

9.4 Eléments supplémentaires

Les éléments décrits ci-dessus constituent la version de base de votre projet. Une version de base qui serait *parfaitement* conçue et qui attesterait du souci de bien faire (bonne utilisation du paradigme orienté objet, clarté du code, jouabilité, pas de bug détecté, documentation complète, respect des consignes, rapport clair et complet, etc. : voir Section 7) vous permettrait d'avoir une note globale de maximum 16/20.

Pour augmenter la note, diverses choses supplémentaires peuvent être ajoutées au projet. Par exemple :

— Chronomètre et scores basés sur le temps de jeu et le nombre de coups.
— Tableau des meilleurs scores.
— Ajout de statistiques.
— Système de profils permettant à chaque joueur d'avoir ses propres statistiques et de reprendre son dernier niveau.
— Système de trophées pour certaines réalisations du joueur.
— Efforts particuliers pour les graphismes.
— Format `.xsb` enrichi permettant l'usage d'un numéro au lieu de $ pour une caisse. Le numéro indique le nombre maximum de poussées ($0 < x < 10$) que l'on peut réaliser sur la caisse avant qu'elle ne soit figée.
— Possibilité de pouvoir "tirer" une caisse au lieu de la pousser (par ex., en appuyant sur "espace" avant de faire un mouvement opposé à une caisse adjacente).
— Ajout d'un "adversaire" qui bouge d'une case tous les 2 tours, toujours en direction du joueur, et qui fait perdre le joueur si l'adversaire se trouve sur la même case (une sorte de "fantôme" dans pacman). Il est possible de régler la difficulté de l'adversaire (par ex, un "facile" ne bouge qu'une fois tous les 4 mouvements du joueur, un moyen tous les 3, un difficile tous les 2).
— Détection d'une partie perdue : quand le joueur, quelque soit son mouvement, ne peut faire bouger aucune caisse. Plus difficile : quand le joueur, quelque soit son mouvement, ne peut atteindre qu'un point fixe qui ne lui permet pas de remporter la partie (par ex, une caisse qui oscille sur x positions mais qu'on ne pourra jamais amener sur la

bonne zone). Cela pourrait être un test non-exhaustif : peut-être que des situations "non terminables" ne seront pas détectées, mais qu'au moins, on puisse détecter des situations typiques.

— Ajout de téléporteurs.
— Mode labyrinthe (au niveau de la génération du niveau).
— Nombre de mouvements limités (vérifier si le niveau reste jouable cependant).
— Caisses plus grandes (ce qui nécessite de plus grands passages).
— Editeur graphique de niveaux.
— Gestion de thèmes pour les graphismes.
— Bruitages sonores.

Cette liste n'est pas exhaustive et les initiatives personnelles de qualité sont encouragées (il est cependant conseillé d'avoir d'abord une version de base solide avant de vous lancer dans les éléments optionnels).

Bon travail !