Algorithmique - INFOB237

Projet - Treasures & Monsters™

Simon LEJOLY - simon.lejoly@unamur.be

Alix DECROP - alix.decrop@unamur.be

Année académique 2023-2024

1 Introduction

Le présent document décrit le projet que vous êtes amenés à réaliser dans le cadre du cours d'algorithmique. Ce projet consiste en l'implémentation d'un petit jeu, sobrement appelé "Treasures & Monsters™". Cette implémentation nécessitera l'utilisation des différentes techniques algorithmiques étudiées dans le cadre du cours.

Bien sûr, l'implémentation d'un jeu complet nécessite de travailler sur des aspects qui ne sont pas purement liés à de l'algorithmique, comme l'interface. Bien que cela fournisse un résultat plus satisfaisant, cela augmente également la charge de travail. C'est pourquoi une partie du projet est facultative. **Les parties obligatoires et optionnelles sont clairement indiquées dans la suite de cet énoncé.**

Le délivrable final du projet dépendra de vos ambitions. Le minimum consiste en l'implémentation des différentes méthodes algorithmique, accompagnée d'un rapport. Si vous décidez d'aller plus loin dans la réalisation du projet, vous pourrez y joindre une vidéo de votre jeu fonctionnel et un rapport plus étoffé.

Cet énoncé décrit en détails des différents aspects du projet. La présentation du jeu à implémenter est abordée en premier lieu. S'en suit la partie obligatoire du projet, décrivant les algorithmes à implémenter. Ensuite, la partie optionnelle décrit comment utiliser les algorithmes codés au sein d'un jeu fonctionnel et complet. Pour terminer, des détails propres à l'implémentation sont fournis, avant d'expliciter les délivrables du projet et sa cotation.

2 Présentation du jeu

Treasures & Monsters™ est un rogue-light[1] minimaliste consistant en une succession de niveaux. Chaque niveau place le héros du joueur en haut d'une grille, représentant un donjon rempli de trésors... et de monstres. Héros, trésors et monstres ont chacun une certaine valeur associée, représentant respectivement les points de vie du héros (limitée à 100), la valeur du trésor (limitée à 99) et la force du monstre (limitée à 50). L'objectif du héros est de descendre de plus en plus bas dans le donjon en récoltant un maximum de trésors tout en préservant sa vie.

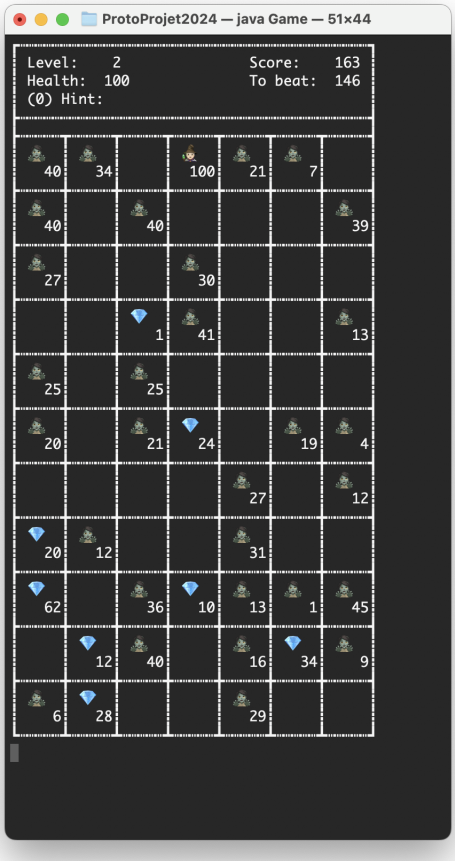

Figure 1: Un exemple de niveau, avec une interface textuelle.

Le challenge du jeu vient du fait que le héros ne peut pas se déplacer librement dans le donjon. Premièrement, il lui est impossible de revenir vers le haut. Deuxièmement, il ne peut pas revenir sur ses pas en parcourant un étage : une fois qu'il a choisi d'aller vers la gauche ou vers la droite, il ne peut que continuer dans cette direction ou descendre un étage plus bas.

Lors de chaque niveau, le jeu indique au joueur un score à battre. Ce score est calculé sur

[1] L'erreur d'écriture est purement volontaire.

base du niveau et est volontairement imparfait, afin que le joueur puisse le dépasser en jouant intelligemment. Si le joueur termine le niveau avec un score supérieur au score à battre, il gagne un indice, qui pourra être utilisé par le joueur dans un niveau suivant. Utiliser un indice permet d'obtenir la solution parfaite du niveau actuel, et ainsi le battre sans soucis.

Une fois que le héros a atteint l'étage le plus bas d'un niveau, se déplacer à nouveau vers le bas chargera le prochain niveau. Le joueur est replacé tout en haut du niveau, au milieu de l'étage, et il regagne 50 points de vie. Un nouveau score à battre est alors calculé, et la partie continue. La partie s'arrête lorsque le héros n'a plus de point de vie, c'est à dire lorsque sa quête infinie prend fin...

3 Partie obligatoire : algorithmes

Cette partie reprend la description des algorithmes à implémenter pour réaliser le projet, ainsi que des questions s'y rapportant. Il est important de noter que si le jeu vous a été expliqué avec un exemple de plateau de dimensions 7x11, les implémentations que vous fournirez doivent être fonctionnelles pour n'importe quelle dimension de plateau. Veillez à ne pas déplacer les méthodes et les interfaces ainsi qu'à conserver leur signature. Cela nous permettra de faire tourner une batterie de tests sur vos implémentations et en conséquence de les évaluer. Nous vous invitons donc à faire vos propres tests également, pour vous assurer que vos programmes sont robustes ; La méthode `main` reprise dans le squelette d'implémentation peut vous permettre de faire ces tests. N'hésitez pas à la compléter au moment de la remise pour qu'elle présente les résultats de vos algorithmes à l'exécution du programme.
Pour récapituler, vos implémentations et vos réponses aux questions relatives aux algorithmes sont toutes deux cruciales dans la réussite de ce projet. L'implémentation permet d'évaluer votre compréhension pratique du cours, tandis que les questions permettent d'évaluer votre compréhension plutôt théorique du cours.

3.1 Générer & Tester

Un des aspects principaux du jeu est la génération aléatoire de niveaux. Cet aspect est crucial et délicat : une génération trop aléatoire risque de mettre le joueur dans des situations impossibles mais une génération trop déterministe rendra le jeu ennuyeux. Vous décidez qu'une génération intelligente devrait garantir un bon équilibre. Par défaut, une case du plateau a 1 chance sur 6 de contenir un trésor, 1 chance sur 3 de contenir un monstre et 1 chance sur 2 d'être vide. La valeur des trésors est déterminée aléatoirement et uniformément entre 1 et 99, celle d'un monstre entre 1 et 50. La case d'apparition du joueur est toujours vide[2]. Pour vous assurer que le niveau reste faisable, vous testez deux contraintes sur toutes les *rangées* de cases générées. La première est que chaque rangée doit contenir au minimum 2 monstres et au maximum 2 trésors. La seconde est que la valeur totale des trésors présents dans la rangée ne peut pas dépasser la valeur des monstres de cette même rangée.

1. Quels sont les avantages et les inconvénients de la méthode Générer & Tester pour ce problème ?
2. Déterminez les fonctions qui seront nécessaires pour résoudre ce problème et spécifiez les en utilisant OpenJML.
3. Si votre algorithme contient un appel récursif, quels sont les cas de base et les cas récursifs ? Si il contient une boucle, pouvez-vous en spécifier l'invariant et le variant/la condition de terminaison ?
4. Implémentez l'algorithme. Quel est l'ordre de grandeur de sa complexité en temps d'exécution et en mémoire ?
5. Votre implémentation est-elle efficace ? Comment pourrait-elle être optimisée ?

3.2 Diviser pour Régner

L'aspect "diviser pour régner" du jeu réside dans l'agencement de chaque niveau. Pour donner un peu de structure à ces niveaux aléatoires, vous choisissez de trier les rangées : celles avec le

[2]Ceci peut être réglé en dehors de l'algorithme.

plus de monstres costauds devraient être en haut, et celles avec les plus gros trésors en bas. La valeur d'une rangée dépend donc de sa valeur totale en trésors, soustraite de sa valeur totale en monstres.

$$RowValue = \Sigma_{row}(TreasureValues) - \Sigma_{row}(MonsterStrengths)$$

1. Quels sont les avantages et les inconvénients de la méthode Diviser pour Régner dans ce problème ?
2. Quel algorithme de tri (nom donné en anglais) allez vous utiliser précisément ? Quels sont ses avantages/inconvénients ? Et sa complexité algorithmique habituelle ?
3. Déterminez les fonctions qui seront nécessaires pour résoudre ce problème et spécifiez les en utilisant OpenJML.
4. Si votre algorithme contient un appel récursif, quels sont les cas de base et les cas récursifs (ou bien le cas s'il n'y en a qu'un seul) ? S'il contient une boucle, pouvez-vous en spécifier l'invariant et le variant/la condition de terminaison ?
5. Implémentez l'algorithme. Quel est l'ordre de grandeur de sa complexité en temps d'exécution et en mémoire ?
6. Votre implémentation est-elle efficace ? Comment pourrait-elle être optimisée ?

3.3 Programmation dynamique

Maintenant que vous savez générer des niveaux, il est temps de se soucier de l'indice à donner au joueur. Quand il est utilisé, cet indice permet au joueur d'obtenir la série de déplacements, de la position actuelle du héros à la fin du niveau, qui fourniront le meilleur score tout en préservant la vie du héros. Plus concrètement, ce chemin maximise la métrique suivante :

$$HeroHealth + TotalTreasuresCollected$$

1. Quels sont les avantages et les inconvénients de la programmation dynamique pour ce problème ?
2. Quelle approche spécifique (ascendante/descendante, itérative/récursive) allez-vous utiliser pour résoudre ce problème ?
3. Quels sont vos cas de base/solutions minimales et vos cas récursifs/solutions intermédiaires et finale ?
4. Quelles valeurs allez vous devoir transmettre/garder à jour pour construire la solution des cas intermédiaires sur base des solutions des cas plus simples ?
5. Déterminez les fonctions qui seront nécessaires pour résoudre ce problème et spécifiez les en utilisant OpenJML
6. Implémentez l'algorithme. Quel est l'ordre de grandeur de sa complexité en temps d'exécution et en mémoire ?

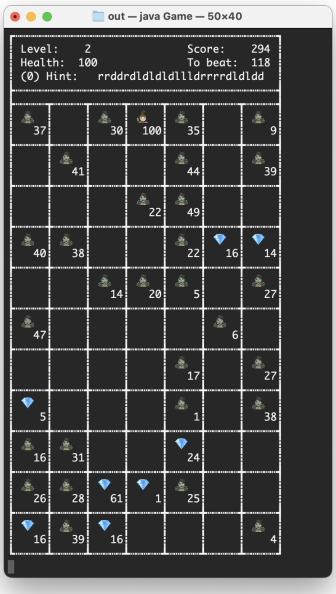

Figure 2: Dans ce niveau, le joueur a utilisé un indice. En suivant les déplacements indiqués, son score sera de 137 pour ce niveau et il terminera avec 59 points de vie.

7. Si votre code contient des boucles, specifiez à l'aide d'OpenJML leurs variants/condition de terminaison et invariants.

8. Votre implémentation est-elle efficace ? Comment pourrait-elle être optimisée ?

3.4 Recherche gloutonne

Afin d'ajouter un peu de challenge, le jeu doit donner au joueur un objectif de score à battre pour chaque niveau. Cet objectif doit correspondre à un score atteignable mais pas parfait pour autant. Pour ce faire, vous choisissez de mettre au point un algorithme capable d'évaluer la valeur d'une case en explorant les cases environnantes pour connaître leur valeur puis en choisissant la meilleure. La valeur d'une case correspond à la même métrique que celle présentée plus haut pour la solution parfaite. Vous êtes conscients que cet algorithme, présenté comme ici, donnerait juste la solution parfaite. Pour le limiter, vous choisissez de lui ajouter une contrainte de profondeur : la valeur d'une case est calculée sur base du *meilleur chemin de 5 déplacements à partir de cette case.*

Pour clarifier les choses, prenont l'exemple concret présenté dans la figure 2. Le héros est sur sa case de départ et peut choisir de descendre, d'aller à gauche ou d'aller à droite. Il va évaluer la valeur de chacune de ses cases en explorant tous les chemins qu'elles proposent. Un exemple de chemin possible en choisissant d'aller sur la case du bas est "dlddr". Ce chemin évite les monstres mais ne récolte aucun trésors, il est donc évalué à 100 (la santé restante du joueur). Un autre chemin serait, en choisissant d'aller sur la case de droite, "rrddd". Ce chemin fait atterrir le joueur sur un trésor de valeur 16, mais lui fait affronter un monstre de valeur 35

sur le chemin, ce qui mène à une valeur finale évaluée à 81. Après avoir évalué tous les chemins pour les 3 cases possibles, l'algorithme choisira d'aller d'abord vers le bas. De là, il reprendra une nouvelle recherche pour savoir quelle direction prendre, en explorant les 3 nouvelles cases suivantes possibles.

1. Quels sont les avantages et les inconvénients de la méthode gloutonne pour ce problème ?
2. Donnez un exemple concret où cet algorithme passera à côté de la solution optimale. Vous pouvez utiliser le plateau ci-dessus ou en créer un vous-mêmes.
3. Déterminez les fonctions qui seront nécessaires pour résoudre ce problème et spécifiez les en utilisant OpenJML.
4. Implémentez l'algorithme. Quel est l'ordre de grandeur de sa complexité en temps d'exécution et en mémoire ?
5. Si votre algorithme contient un appel récursif, quels sont les cas de base et les cas récursifs ? Si il contient une boucle, pouvez-vous en spécifier l'invariant et le variant/la condition de terminaison ?
6. Votre implémentation est-elle efficace ? Comment pourrait-elle être optimisée ?

4 Partie optionnelle : jeu complet et extensions

La première étape pour rendre le jeu réellement jouable sera de lui fournir une interface. Une simple interface textuelle pourra déjà amplement faire l'affaire vu la simplicité du jeu. Le joueur pourra indiquer la direction en écrivant la direction dans laquelle il veut déplacer le héros (ex: "d" pour "down", "l" pour "left", "r" pour "right"). Vous pouvez également utiliser des librairies graphiques disponibles en Java (Processing, Swing, ...).

Pour rendre le jeu plus intéressant, certains changements de règles pourraient être utiles. Les indices semblent trop puissants : donner l'entièreté de la solution parfaite d'un niveau permettra quasi systématiquement au joueur de dépasser le score à battre et donc de gagner un nouvel indice pour le niveau suivant. De même, un joueur qui évite simplement d'affronter des monstres sans se soucier des trésors serait capables d'avancer infiniment dans le jeu. Vous êtes libres d'équilibrer la logique de succession des niveaux pour contrer ces problèmes. On pourrait ainsi imaginer un magasin entre deux niveaux où le joueur pourrait échanger ses trésors contre de la santé, une armure, un indice, ...

Vous pouvez également inclure des fonctionnalités nouvelles et plus profondes. Plein d'idées pourraient améliorer le jeu : ajouter un monstre légendaire très résistant mais qui doublerait la valeur de tous les trésors du niveau si il est abattu, proposer un multiplicateur de score quand le joueur tue plusieurs monstres d'affilée, introduire un "boss final" à un certain niveau, corser les niveaux au fur et à mesure, ...[3]

Attention : ces modifications ne doivent pas vous empêcher de fournir dans `Algorithms.java` les algorithmes demandés, qui doivent fonctionner pour les règles par défaut présentées dans cet énoncé. Si vos modifications impliquent de changer ces algorithmes, vous pouvez copier la structure de `Algorithms.java` dans une autre interface (e.g. : `AdvancedAlgorithms.java`) et y apporter les changements nécessaires à votre jeu.

[3] *"La créativité, c'est l'intelligence qui s'amuse."* -Albert Einstein

5 Détails d'implémentation

Un squelette d'implémentation des quatre algorithmes à implémenter vous est fourni sur Webcampus. Il contient une interface (`Algorithms.java`) et une méthode pour chaque algorithme. Il vous est demandé de **respecter la signature de ces méthodes** (types des paramètres et de la valeur de retour). Vous pouvez ajouter des méthodes aux interfaces pour découper la tâche.

Chacune des méthodes présentes dans l'interface devra être spécifiée à l'aide d'OpenJML. Bien qu'il ne vous soit pas demandé que ces spécifications puissent être prouvées par un solver, un minimum de rigueur est demandé dans leur rédaction. Les spécifications devront également indiquer les auteurs de la specification et de l'implémentation de la méthode concernée.

```
/*@ [First, OpenJML doc]
  @ requires param > 0;
  @ ensures \return > 0;
  @ pure
  --- [Second, authors info]
  * Specification: Alex           | Last update: 25/03/24
  * Implementation: Marty, Melman | Last update : 27/03/24
  * Test/Debug: Gloria            | Last update : 01/04/24
***/
int maFonction(int param) {...}
```

Il vous est demandé d'utiliser le repository GitHub mis à votre disposition par la faculté. Les commits réalisés sur ce repository serviront à l'évaluation de votre implication dans le projet : **il est attendu que chaque étudiant réalise les commits des portions de code qu'il aura écrites et puisse expliquer leurs implémentations**. Il serait donc incohérent de voir un étudiant crédité dans la spécification d'une fonction sans qu'il n'ait de commit sur cette partie du code.

Nous tenons à rappeler que les objectifs pédagogiques du cours d'algorithmique forment un pré-requis, à la fois pour la suite de votre cursus universitaire mais également pour vos futurs postes en informatique. Nous vous invitons donc à vous entraider, mais en veillant à ce que chacun développe sa compréhension des sujets traités. Coder la partie d'un collègue à sa place n'est pas lui rendre service. Soyez également prudents si vous partagez des portions de code complètes avec d'autres groupes, car en cas de plagiat le groupe ayant fourni le code sera autant tenu responsable que celui l'ayant copié. De même, l'utilisation d'outils de génération de code (ChatGPT, GitHub Copilot, ...) doit se faire avec beaucoup d'esprit critique. Si leur utilisation pour générer du code qui serait rébarbatif à écrire manuellement pourra être tolérée (par exemple : coder une méthode qui affiche de façon lisible le contenu d'un ArrayList d'Arrays ou construire une classe représentant un Tuple), **leur utilisation pour résoudre tout ou une partie d'un *problème algorithmique* à votre place sera sanctionnée par un 0 pour l'entièreté du projet, et risque d'être étendu à toute le groupe**. Si toutefois vous utilisez ces outils, nous vous demandons de les créditer comme auteurs dans la spécification[4].

Pour faciliter l'exécution de votre programme, nous vous invitons à rassembler votre code dans un dossier "src" et les éventuelles versions compilées dans un dossier "out". De cette façon, les commandes suivantes devraient permettre de lancer votre implémentation :

```
#!/bin/bash

javac -classpath "src/" -d "out/" src/Game.java
```

[4]Qui sait, peut-être cela vous épargnera de la torture lorsque les IAs prendront le contrôle du monde...

```
java -cp "out/" Game
```

Si les modifications que vous apportez dans les aspects optionnels du projet impliquent d'autres étapes pour le lancer, vous les indiquerez dans le fichier `README.md`. Ce genre de modification n'est cependant pas encouragé.

6 Délivrables et cotation

6.1 Délivrables

La complétion du projet implique la remise d'au moins **deux délivrables** : le code de votre projet ainsi qu'un rapport.

6.1.1 Code

Le code de votre projet devra être présent sur le GitHub de votre groupe ainsi que **dans une archive .zip** remise sur webcampus avant la deadline. **Vous serez évalués sur le contenu du code de l'archive**, afin de pouvoir éventuellement continuer à modifier le code sur votre repository après le projet si vous en avez l'envie. Veillez à ce que cette archive contienne tout votre code ainsi que les informations nécessaires à l'exécution du programme.

Voici quelques exemples de critères qui seront appliqués dans l'évaluation du code :

- Le programme se lance (compile et exécute) et tourne sans erreurs/crash
- Les méthodes renvoient les résultats attendus
- Les signatures des méthodes fournies dans le squelette d'implémentation n'ont pas été modifiées
- Les spécifications sont correctement écrites (OpenJML, nom des auteurs, dernière date de modification)
- Le code est lisible, écrit proprement
- Le code propose des commentaires pour expliciter les portions plus difficiles à comprendre
- Les méthodes et les boucles présentes dans `Algorithms.java` sont correctement spécifiées à l'aide d'OpenJML
- Les implémentations des algorithmes suivent clairement la structure attendue par rapport à leur "type" d'algorithme (diviser pour régner, gloutons, ...)
- Le code est relativement optimisé pour en réduire la complexité

6.1.2 Rapport

En plus du code, vous remettrez un rapport expliquant les divers aspects de la réalisation de ce projet. Ce rapport contiendra nécessairement une section dédiées aux réponses des questions posées dans la section "Algorithmes" de l'énoncé. Si vous avez implémenté le jeu complet, apporté vos modifications aux règles, conçu une interface, vous l'indiquerez dans une autre section. Vous pouvez également mentionner votre fonctionnement de groupe, les difficultés rencontrées dans la réalisation du projet, ou tout autre aspect que vous jugez pertinent. Pour la rédaction du rapport, nous vous encourageons à utiliser le template LaTeXfourni sur Webcampus, en utilisant un éditeur comme Overleaf.

Votre rapport sera évalué sur base des critères suivants :

- Il reprend le n° du groupe ainsi que les noms de ses membres
- Le contenu du rapport est complet (partie obligatoire, optionnelle, ...)
- Le rapport est bien écrit (orthographe, logique du texte, ...)

- Le rapport suit une mise en page propre
- Le rapport est complet (*en particulier pour ce qui concerne les questions algorithmiques*) mais concis
- **Le rapport ne dépasse pas 6 pages**, sans compter la page de garde.

Si d'autres documents peuvent être utiles à la correction (e.g. : vidéo d'une partie jouée sur votre moteur de jeu), vous pouvez les inclure dans votre repository et les mentionner dans votre rapport.

Gardez également en tête que votre activité sur le repository sera également utile à l'évaluation. Même si c'est secondaire, une activité régulière sur le repository de la part de chaque membre de groupe sera plus appréciée qu'un "rush final" la veille de la deadline.

La deadline de fin du projet est fixée au **vendredi 10 mai à 23h00**. Au vu de la dimension réduite du projet et du temps disponible avant la deadline, **un retard dans la remise se soldera systématiquement d'un 0 pour le projet**, pour chaque membre de l'équipe. Nous vous encourageons à prendre de l'avance pour vous assurer que la remise se déroule sans encombre.

6.2 Cotation

Le projet compte pour 30% de la note finale du cours. La note du projet dépendra d'une grille de cotation basée sur les critères cités plus haut. Des points bonus pourront être donnés sur base du travail optionnel fourni. Ainsi, un projet minimal mais parfait aura une note parfaite. De même, un projet moyen sur la partie obligatoire mais ayant mis des efforts sur la partie optionnelle sera tout de même apprécié. Notez cependant qu'un projet qui ne témoignerait pas d'une maîtrise minimale des concepts d'algorithmique vus durant les cours et les TPs ne peut pas espérer être suffisant, peu importe le niveau de travail optionnel réalisé.

La correction de votre rapport et de votre code donneront une note provisoire qui servira de base lors d'un examen oral en groupe. Cet examen oral vise à évaluer la participation de chaque membre dans le travail du groupe. Une note différente d'un membre à l'autre de l'équipe est donc possible selon notre impression de votre participation active durant le projet. L'examen oral pourra également permettre aux groupes qui auront codé un jeu complet de nous le présenter plus en détails.

Il est important de noter qu'au vu de la charge pratique que représente l'organisation de ce projet (constitution de groupes, énoncé original, examens oraux, ...), **il n'est pas possible de réaliser le projet en seconde session. En cas d'échec durant la session de juin, votre note de projet sera reconduite pour la session d'août.** En cas d'échec pour cette année académique, le projet sera à refaire pour l'année académique suivante.

Nous sommes disponibles pour vous aider dans la réalisation de ce projet si des aspects vous semblent peu clairs. N'hésitez pas à nous contacter par email pour tout problème (e.g. : ambiguïté dans l'énoncé, bloqué dans votre code, problème de groupe, ...).

Nous vous souhaitons un excellent travail !