Chasse au trésor

Programmation Object Avancée

François Poguet
Enzo Costantini
TP1A

Licence Informatique
Semestre 4

Table des matières

Table des figures

Introduction

Dans le cadre du module de *Programmation Objet Avancée*, le projet de fin de semestre est le développement d'une application de chasse au trésor. Le projet était divisé en deux échéances, d'abord un rendu de la version console de l'application, puis de la version graphique. La deuxième partie étant le prolongement de la première, ce rapport portera sur la version finale de du programme. Nous vous présenterons dans un premier temps l'application, ce qui était attendu, et ce que nous avons fait. Ensuite, nous verrons la conception et le développement de notre programme. Et enfin nous finirons sur un bilan à plusieurs niveaux vis avis de ce projet.

1 Présentation de l'application

1.1 Le sujet du projet

L'application demandée permet de simuler un jeu de chasse au trésor, c'est-à-dire de suivre l'évolution de personnages sur un damier constitué de différents types de cases. Les personnages sont guidés à chaque tour par les cases du damier, chaque type de case a un comportement différent. Ainsi, les personnages se déplacent automatiquement, sans choix de l'utilisateur.
Le sujet demandait au minimum de représenter graphiquement une partie de chasse au trésor, en demandant à l'utilisateur de cliquer sur un bouton pour exécuter un tour. Cependant il était aussi ouvert à de nombreuses extensions et améliorations.
L'objectif du projet était dans un premier temps de mettre en oeuvre les concepts de la programmation objet, à savoir l'héritage, le polymorphisme ainsi que les notions de classes abstraites et d'interfaces. Puis dans un second temps, le but était de représenter l'application graphiquement avec la bibliothèque *Swing* de *Java*, en implémentant une architecture *Modèle-Vue-Controleur*.

1.2 Les extensions implémentées

Nous avons profité de la liberté offerte par le sujet et des nombreuses possibilités d'améliorations pour aller plus loin que le cahier des charges minimales. Nous avons implémenté des extensions permettant de répondre à des besoins d'une telle application, et aussi de découvrir d'autres aspects de la programmation-objet en Java.

La génération aléatoire du damier
Cette première amélioration est arrivée tôt dans le projet, puisqu'elle nous paraissait vraiment nécessaire, nous n'imaginions pas développer un jeu de chasse au trésor avec seulement quelques damiers préfabriqués.

Les déplacements automatiques
Implémenté dès le début du projet, nous avons mis en place un système de

d'enchainement pour que les tours s'enchainent sans avoir besoin de cliquer pour jouer, comme la première, cette option nous paraissait indispensable. L'utilisateur a aussi la possibilité de mettre la partie en pause à tout moment.

La configuration totale
Nous avons décidé, pour élargir les possibilités de l'utilisateur, de rendre configurables toutes les données du jeu, c'est-à-dire la taille du damier, le nombre de joueurs, la densité des murs ou encore l'intervalle de temps entre chaque tour.

Le mode édition
Le mode édition nous semblait être une extension importante de ce projet, cela donne à l'utilisateur un nouveau mode de jeu, lui offrant encore plus de liberté. Nous avions pour objectif personnel de rendre l'interface du mode édition la plus ergonomique possible.

La sauvegarde du damier dans des fichiers
En développant le mode édition, nous nous sommes rendu compte que la création du damier pourrait être bien plus utile si l'utilisateur avait la possibilité de sauvegarder ses créations, nous avons donc pensé à ce moment-là à intégrer un système de fichiers pour pouvoir sauvegarder et réutiliser n'importe quel damier, qu'il soit créé à la main ou généré aléatoirement.

La barre de menu
Cette extension minime nous aura permis d'approfondir nos connaissances en *Swing*, et par la même occasion définir des raccourcis clavier pour l'application.

2 Conception et développement

2.1 Diagramme et choix d'implémentation

Le modèle

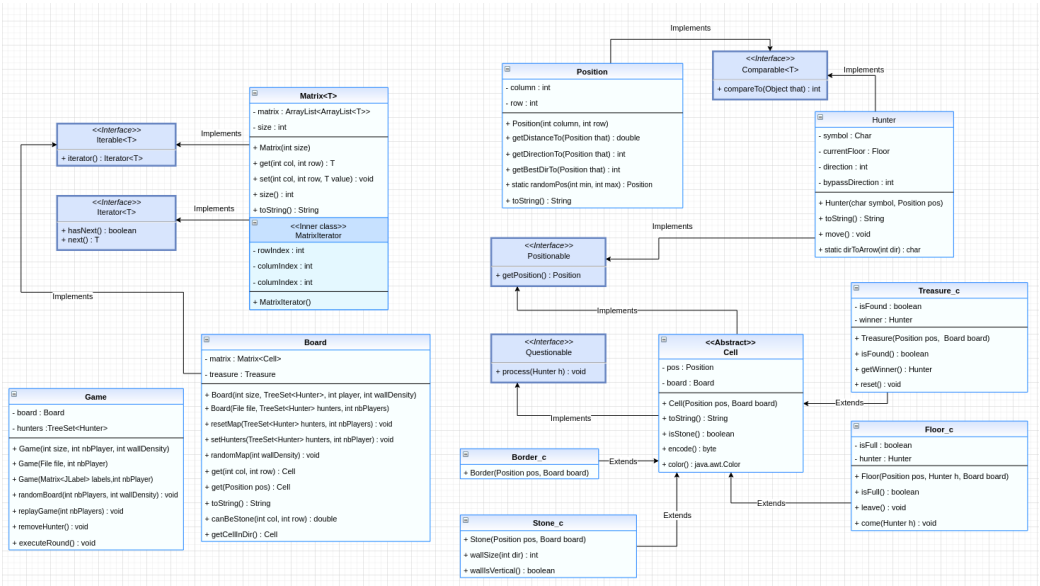


FIGURE 1 – Diagramme de classes du modèle

La position d'un hunter
Bien que la classe *Hunter* implémente l'interface *Positionable*, le hunter ne stocke pas une position en tant que telle. Il possède une case courante, qui elle, est caractérisée par une position.

Les murs et leur contournement
Pour l'implémentation des murs, nous avons réfléchi à une éventuelle classe *Mur* qui regrouperait plusieurs cases de type *Stone*, mais nous ne pensons pas que cela soit vraiment utile.
Pour contourner les murs, les hunters possèdent une "direction de déviation" qui leur permettra de les contourner au mieux.

La collection Matrix<T>

Nous avons choisi d'implémenter nous-mêmes une collection pour représenter un tableau à deux dimensions. L'avantage de cette collection générique est qu'elle sera adaptée au mieux aux besoins de l'application. Pour cela, des ArrayList seront utilisées, car une fois la matrice créée, les opérations principales seront des recherches et non des ajouts/suppressions.

Le contrôleur et les vues

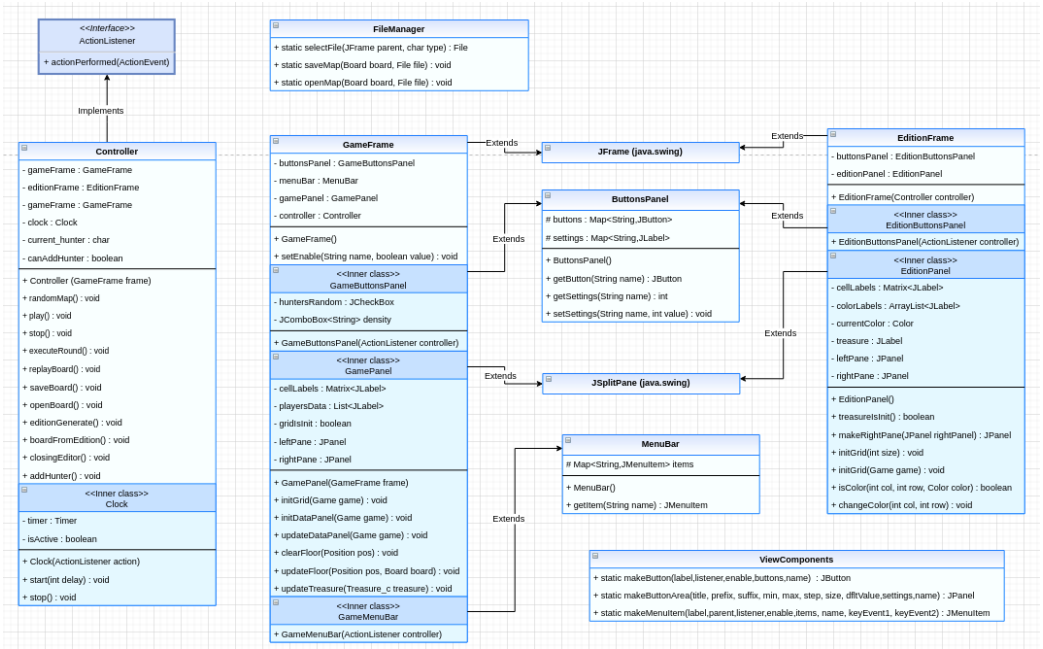


FIGURE 2 – Diagramme de classes du contrôleur et des vues

La classe FileManager

Cette classe regroupe des méthodes statiques relatives à la gestion des fichiers, comme l'ouverture ou la sauvegarde d'un damier.

L'organisation des vues

Pour factoriser au maximum le code source et ainsi faciliter l'ajout de po-

tentielles futures vues, nous avons implémenté une classe *ButtonsPanel* et une classe *MenuBar*. Ces deux éléments sont souvent utilisés dans les différentes fenêtres d'une application. Ces classes présentent les comportements communs à ces éléments et sont destinées à être hérité pour ajouter des spécificités. Ainsi nous utilisons, dans les vues, des classes internes, pour bien séparer les différents éléments et simplifier leur modification.

La classe ViewComponents
Cette classe regroupe des méthodes statiques relatives à la construction de composants graphiques des fenêtres comme les boutons. Cela permet d'alléger significativement le code source des fenêtres graphiques.

2.2 Algorithmes intéressants

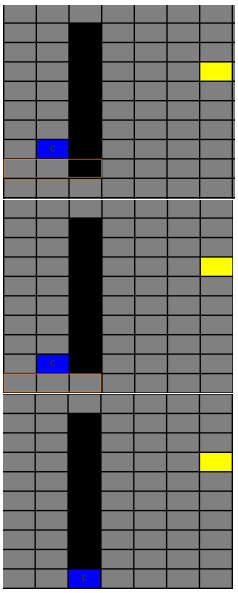
Nous allons maintenant vous présenter et expliquer quelques algorithmes qui nous ont semblé importants dans le développement de cette application. Les codes sources de ces algorithmes sont disponibles dans les annexes.

La génération aléatoire
La génération aléatoire du damier se base sur des probabilités. C'est-à-dire que le damier est initialisé ligne par ligne et pour chaque case, en fonction de la configuration, la case à une probabilité **p** de devenir un mur. Les différents cas possibles et les probabilités associées sont :

1. Si la case est collée à un bord : 0.
2. Si une des cases en haut gauche ou haute droite est un mur : 0.
3. Si la case en haut n'est pas un mur :
   (a) Si c'est le début d'un mur : **p1**.
   (b) Si c'est la seconde pierre d'un mur horizontal : **p2**.
   (c) Si c'est la n iéme (n>2) pierre d'un mur horizontal : **p3**.
4. Si la case en haut est un mur :
   (a) Si la case à gauche est un mur : 0.
   (b) Si c'est la deuxième pierre du mur vertical : 1.
   (c) Si c'est la n iéme (n>2) pierre d'un mur vertical : **p3**.

Les trois variables **p1**, **p2** et **p3** permettent de configurer la génération aléatoire, elles varient en fonction de la densité des murs choisis par l'utilisateur, et ont été testées pour avoir un résultat satisfaisant, longueur des murs variable et équilibre entre le nombre de murs horizontaux/verticaux.

Le contournement des murs
La gestion des murs a été la seule difficulté dans l'implémentation du déplacent des joueurs jusqu'au trésor. Pour gérer cela, nous avons utilisé attribut supplémentaire pour le joueur, la direction de déviation. Cette direction est décidée dès le premier contact avec le mur, c'est à ce moment-là que la direction idéale est calculée en fonction de la position du joueur par rapport au mur, lors des tours suivants et jusqu'à ce que le mur soit contourné, le joueur se contente de suivre sa direction de déviation. Exemple :



Ici le hunter heurte le mur en allant au trésor, il est décidé maintenant que la direction de déviation sera vers le bas.

Avec cette direction de déviation, le joueur a la possibilité d'aller sur les trois cases du bas, il choisit la meilleure parmi elles.

Il effectue la même opération, jusqu'à arriver à un bord du mur, une fois le mur contourné, le joueur reprend son chemin.

FIGURE 3 – Illustration du contournement des murs

La direction de déviation est réinitialisée et n'est plus utilisée une fois le mur contourné.
Le code relatif au contournement des murs est assez long, car nous avons

voulu traiter au mieux tous les cas possibles, nous avons pensé à le factoriser, mais la compréhension serait beaucoup réduite.

La compression des fichiers
Lors de la sauvegarde d'un damier dans un fichier, il est compressé pour réduire au maximum la taille du fichier. Le fichier binaire est d'abord composé d'un entier indiquant la taille du damier, ensuite chaque type de case est représenté par un byte. L'algorithme de compression est un algorithme standard basé sur la répétition de cases de même type successives. Si plusieurs cases de même type se suivent horizontalement, le nombre de cases sera écrit et sera suivis du numéro de type de la case.

2.3 Environnement et méthodes de travail

Eclipse pour le développement
Le projet aura été entièrement réalisé sous Eclipse, nous avons pu découvrir et utiliser les nombreux outils que cet environnement de développement propose. Nous avons souvent eu recours à l'outil de debuggage, nous avons mis un peu de temps à comprendre comment bien l'utiliser, mais nous en avons gagné beaucoup grâce à cet outil par la suite. Nous avons aussi exploré d'autres fonctionnalités comme le "coverage".

Git & GitLab pour la gestion de projet
Pour ce qui est de la gestion du développement à plusieurs, nous avons utilisé Git pour gérer la sauvegarde est le versioning du projet, et le service en ligne GitLab, qui nous a permis de collaborer facilement. L'environnement Eclipse cité plus haut intègre une fonctionnalité pour gérer un dépot Git, nous l'avons largement utilisée et cela aura été très pratique.

LaTeXet draw.io pour le rapport
Pour rédiger un rapport de qualité, nous avons choisi d'utiliser LaTeX. Pour le diagramme de classes présenté plus haut, nous avons utilisé le logiciel libre draw.io.

3 Bilan du projet

3.1 La version finale

Une interface ergonomique
Nous avons essayé de rendre l'application la plus intuitive possible, cependant nous aurions aimé ajouter un manuel d'utilisation pour clarifier certains points (voir *améliorations possibles* page 13).

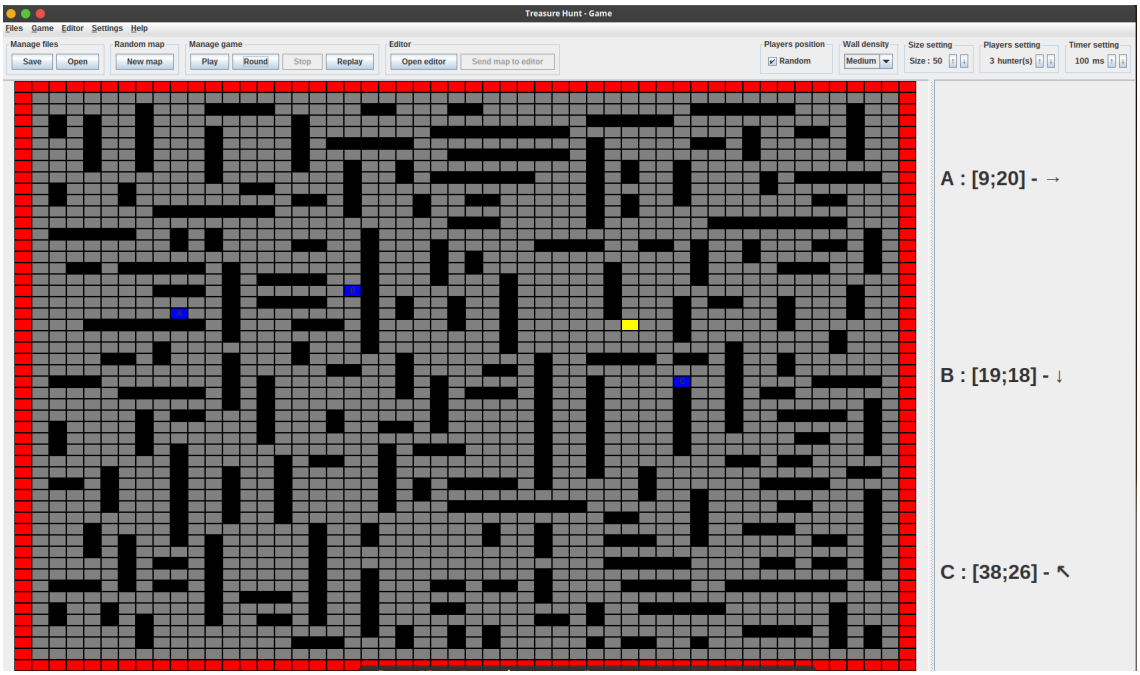


FIGURE 4 – Fenêtre de jeu de l'application

De nombreux boutons
Par le biais d'un panel de contrôle (voir Figure 4), l'utilisateur peut facilement sauvegarder ou ouvrir un damier, ou générer un nouveau damier en le configurant grâce aux différentes options proposées. (*Attention : les options désirées doivent être sélectionnées avant de cliquer sur le bouton pour générer le damier ou rejour la partie.*)
Il peut contrôler le déroulement d'une partie, c'est-à-dire la lancer en mode

automatique, l'arrêter à n'importe quel moment, ou encore jouer au tour par tour.
Il peut également rejouer une partie, c'est-à-dire replacer les joueurs sur le même damier.
Le placement des joueurs peut être aléatoire ou manuel, auquel cas l'utilisateur n'a qu'à cliquer sur une case pour y positionner un joueur.
Toutes ses fonctionnalités sont également accessibles depuis la barre de menu et des raccourcis clavier (voir Figure 5).

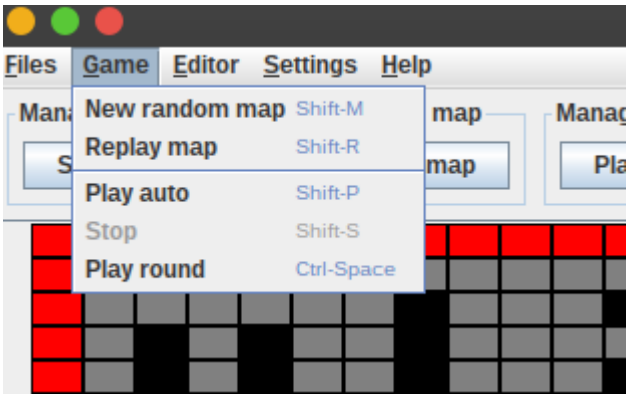


FIGURE 5 – Barre de menu et raccourcis clavier

Le mode édition
Nous voulions implémenter un mode édition le plus simple d'utilisation possible, pour éditer un damier il suffit de sélectionner la couleur voulue sur le panel de droite, puis cliquer directement sur les cases pour ajouter ou supprimer les différents types de cases (voir Figure 6).
Le joueur peut créer un damier à partir d'un damier vide, ou alors importer le damier de la fenêtre de jeu, il est donc possible de créer, sauvegarder et modifier un damier à l'infini.
Le joueur n'a qu'a cliquer sur un bouton pour envoyer le damier à la fenêtre de jeu.

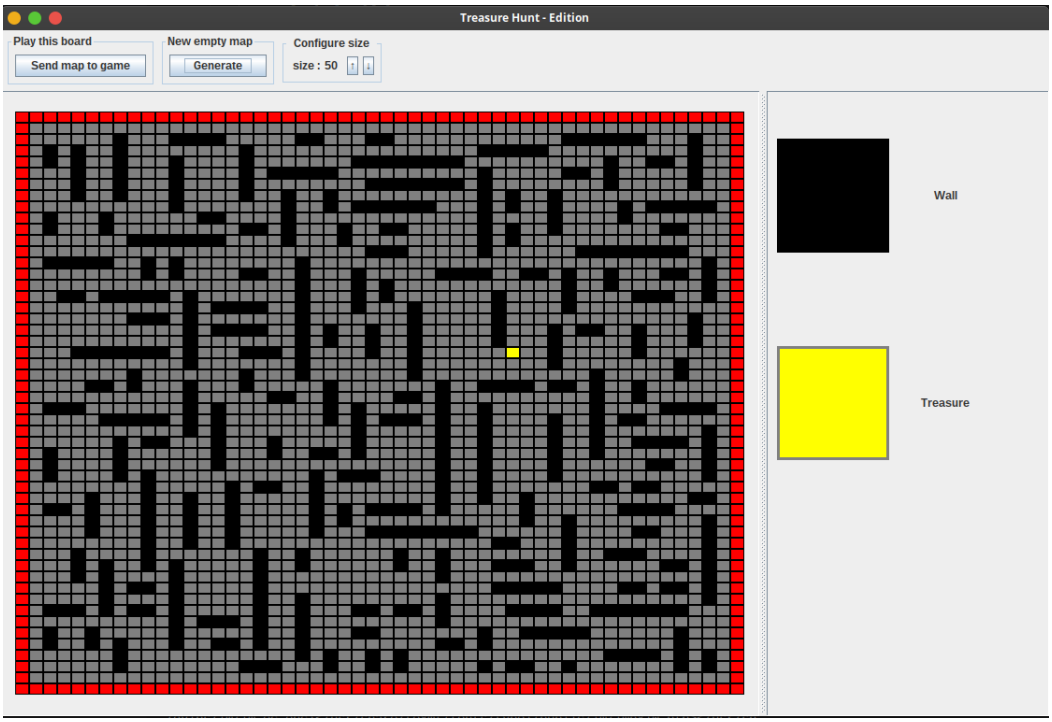


FIGURE 6 – Fenêtre d'édition d'un damier

3.2 Bilan technique et pédagogique

Un projet riche techniquement
La version finale de l'application répond à toutes les attentes du sujet, nous sommes satisfaits de ce que nous avons produit et de la manière dont s'est déroulé le projet.
Nous avons abordé de nombreux aspects de la programmation en JAVA, comme l'architecture MVC utilisée pour la première fois en projet.
Nous avons largement approfondi le paradigme de programmation événementielle avec les interfaces graphiques de la bibliothèque Swing.
Nous voulions implémenter la possibilité de sauvegarde pour en apprendre davantage sur la manipulation de fichiers, cela nous aura demandé du temps et de la recherche, mais ce ne sera que bénéfique pour la suite de notre formation.

Une gestion de projet efficace

Il faut savoir que deux autres projets étaient développés en parallèle de celui-ci, dans les modules de *Langages du web* et *Programmation système*, nous avons donc dû organiser notre temps pour fournir un travail de qualité sur ces trois projets.

Nous avons passé du temps sur la partie analyse avant de commencer à coder pour que le développement se fasse de manière cohérente du début à la fin, nous ne sommes à aucun moment revenus sur nos idées de départ.

De plus, ce semestre aura été en partie dispensé à distance en raison du confinement dû à l'épidémie de COVID-19, un autre facteur important qui nous a demandé une adaptation et nous avons significativement gagné en autonomie.

Améliorations possibles

De nombreuses améliorations nous sont venues à l'esprit, nous avons dû choisir lesquelles nous voulions implémenter. Le sujet suggérait la possibilité d'ajouter d'autres types de cases, nous avons fait le choix de ne pas suivre cette idée pour avoir plus de temps sur d'autres extensions qui nous paraissaient plus importantes et plus riches d'apprentissages, l'implémentation ne serait pas compliquée et nous le ferions bien évidemment si l'application venait à être réellement utilisée.

Nous avons pensé à d'autres options plus minimes, comme un manuel d'utilisation ou la possibilité de joindre les développeurs (à savoir nous) depuis l'application pour reporter un bug ou donner son avis.

Le temps aura malheureusement limité les possibilités, mais nous sommes vraiment satisfaits du résultat obtenu.