PG4 : Akropolis

Nidhal Moussa & Mathéo Piget & Benzerdjeb Reyene & Gbaguidi Nerval & Chetouani Bilal

May 13, 2024

Contents

1 Introduction

1.1 Présentation du Projet

Le projet consiste en la réalisation d'un jeu de société nommé Akropolis, qui est un jeu tour par tour dont le but est de construire une cité antique. Le jeu se déroule sur un plateau de jeu composé de cases hexagonales. Chaque joueur possède un certain nombre de ressources qu'il peut utiliser pour construire des bâtiments sur les cases du plateau. Chaque bâtiment rapporte des points de victoire à son propriétaire en remplissant certaines conditions. Il est aussi possible de les faire supperposer pour obtenir d'avantage de points. Il faut donc gérer le placement des bâtiments pour maximiser les points de victoire. La pioche de tuiles est limitée. Le jeu se termine lorsqu'il n'en reste qu'une. Le joueur qui a le plus de points de victoire à la fin de la partie est déclaré vainqueur.

Réalisé en java et en utilisant la bibliothèque graphique Swing, le jeu est jouable en local sur un ordinateur.

1.2 Objectifs

L'objectif principal du projet est de réaliser un jeu de société complet et fonctionnel, avec une interface graphique permettant de jouer en local sur un ordinateur. Les objectifs secondaires ont été de réaliser un jeu agréable à jouer, avec une interface graphique intuitive et des graphismes de qualité.

2 Gestion du projet

2.1 Méthode

Pour la gestion du projet, nous avons opté pour une méthode de développement agile, basée sur des itérations courtes et des réunions régulières pour faire le point sur l'avancement du projet et discuter des problèmes rencontrés. Nous avons également utilisé un dépôt Git pour gérer les différentes versions, ce qui nous a permis de travailler en parallèle sur les différentes parties du jeu et de fusionner nos modifications facilement. Nous avons priviligé en priorité la réalisation du modèle de jeu en implémentant les règles du jeu. Nous avons ensuite travaillé sur la vue (ce qui fût la partie la plus longue) et relier le modèle à la vue en implémentant le contrôleur.

2.2 Répartition des taches

Pour la réalisation du projet, nous avons réparti les tâches en fonction des compétences de chacun. La polyvalence de chacun nous a permis de travailler sur plusieurs aspects du jeu. Nous avons également organisé des réunions régulières (plus ou moins obligatoire grace à ce cours) pour discuter de l'avancement du projet et des problèmes rencontrés (voir section 4).

2.3 Architecture du Projet

Pour ce projet, nous avons opté pour une architecture orientée objet, avec une séparation claire entre les différentes parties du jeu, basé sur un modèle MVC (Modèle-Vue-Contrôleur) car beaucoup plus modulable et facile à maintenir, pour faire des modifications ou ajouter des fonctionnalités. Nous avons également utilisé des classes utilitaires pour gérer le jeu. La partie graphique du jeu est gérée par la bibliothèque Swing, qui permet de créer des interfaces graphiques en Java. Aucune autre bibliothèque n'a été utilisée pour la réalisation du jeu (à part JUnit pour les tests unitaires). Avec une grande partie assez abstraite, le jeu est facilement modifiable et extensible, principalement pour les jeux de société.

3 Développement et Fonctionnalités

Le développement du jeu a été réalisé en plusieurs mois, en plusieurs étapes, en commençant par la réalisation des mécaniques de jeu de base, puis en ajoutant des fonctionnalités supplémentaires et en améliorant l'interface graphique. Dans un premier temps, étant donné que le modèle de conception du jeu est basé sur le modèle MVC, nous avons commencé par la réalisation du modèle, la vue et le

contrôleur plus ou moins en même temps. Parallèlement à cela, nous avons travaillé sur les tâches annexes comme la gestion des images, des sons, mais également sur des scripts bash et powershell. Mais également sur des tests unitaires pour vérifier le bon fonctionnement de notre code.

3.1 Organisation du Code

Le code du jeu est organisé en plusieurs packages, chacun regroupant des classes ayant un rôle adéquat. Le modèle du jeu est regroupé dans le package `model`, la vue dans le package `view` et le contrôleur dans le package `controller`, eux même contenant des sous-packages pour organiser les classes. Les classes utilitaires sont regroupées dans le package `util`. Les classes de tests sont regroupées dans le package `test`. Les ressources du jeu (images, sons, etc.) sont stockées dans le dossier `res`. Pour rentrer dans les détails, le modèle est composé de plusieurs classes, chacune représentant un élément du jeu (joueur, plateau, bâtiment, etc.). La vue est composée de plusieurs classes, chacune représentant un élément graphique du jeu (fenêtre principale, plateau de jeu, etc.). Pour plus de lisibilité et de clairety, nous avons décidé de faire un schéma UML de notre projet, que vous pouvez retrouver en annexe, voir section 6.

3.2 Fonctionnalités Principales

Le jeu dispose d'un menu principal, qui permet de lancer une nouvelle partie, de voir les règles, de voir les crédits, de modifier certains paramètres ou de quitter le jeu. Une fois une partie lancée, le jeu se déroule en tours par tours. Le jeu suit les règles du jeu Akropolis, avec des bâtiments à construire, des ressources à gérer, et des points de victoire à accumuler. Le jeu se termine lorsqu'il n'y a plus de tuiles dans la pioche, et le joueur avec le plus de points de victoire est alors déclaré vainqueur.

Etant donné que le jeu est un jeu de société, les fonctionnalités principales sont les suivantes :

- **Plateau de Jeu :** Le plateau de jeu est composé de cases hexagonales, sur lesquelles les joueurs peuvent placer de tuiles.
- **Ressources :** Chaque joueur possède un certain nombre de pierres qu'il peut utiliser pour construire des tuiles.

Images et Textures

- **Chargement des Images :** Les images et textures du jeu sont stockées dans des fichiers séparés.
- Le jeu les charge dynamiquement lors de son exécution.
- **Format des Images :** Les images sont au format PNG pour assurer une qualité graphique et compatibilité optimale et pour gérer la transparence avec Swing.

Fichiers Sonores

- **Musique de Fond :** La musique de fond du jeu est également gérée en tant que ressource sonore, contribuant à l'ambiance globale.

3.3 Fichiers de Configuration

3.4 Logique de Jeu

Le jeu est un jeu tour par tour, où chaque joueur peut choisir une tuile à placer sur le plateau de jeu. Le coût de la tuile est définis par la position de cette dernière dans le site. On utilise les rochers pour payer les tuiles. Le site est modélisé par la classe Site qui contient un tableau de tuiles de taille précalculée en fonction du nombre de joueurs. Grâce à la position de la tuile dans le tableau, on peut déterminer son coût. Pour que cela marche, il faut que les tuiles soient placées dans l'ordre. Il faut donc réaranger le site à chaque fois qu'une tuile est placée pour ne pas avoir de valeurs null entre les tuiles. La pioche est modélisée grâce à la classe StackTile qui extend de Stack<Tile> et qui contient les tuiles restantes à piocher. Elle implémente une méthode pour générer ces tuiles de manière aléatoire lors de la création de l'objet. Nous avons veiller à générer un nombre minimum de certains types de tuiles pour éviter les

parties trop déséquilibrées. La classe Player contient les informations sur le joueur, comme son nom, son score, ses ressources et sa grille de jeu. La grille de jeu est représentée par une table de hashage qui associe une tuile à une position. Elle a aussi en référence le joueur pour pouvoir lui ajouter des ressources lorsqu'il supperpose une mine. Elle gére si un ajout à une position est possible ou non dans la méthode addTile. La classe Tile contient simplement une liste d'hexagones qui la compose. Chaque hexagone contient une position représenté par un Vecteur3D. La classe Hexagone est abstraite. Nous définissons donc nos différents types concrets d'hexagones :

- **Place** : Les places du jeu, elles permettent d'avoir un multiplicateur de score.
- **Quarries** : Les carrières du jeu, elles permettent de gagner des ressources lorsqu'elles sont supperposées.
- **District** : Les quartiers du jeu, ils permettent de gagner des points de victoire.

La classe Hexagon et Tile implémentent toutes les deux serializable. Cela est dû au fait que nous voulious ajouter en extension un mode multijoueur en ligne. Cette fonctionnalité n'a pas été implémentée mais nous avons prévu le coup en rendant nos classes serializable (plus de détails dans la section 3.6).

3.5 Interface Graphique

Comme mentionné précédemment, l'interface graphique du jeu est réalisée en utilisant la bibliothèque Swing, qui permet de créer des interfaces graphiques en Java. L'avantage de son utilisation est qu'elle est incluse dans la bibliothèque standard de Java, ce qui signifie qu'elle est disponible sur toutes les plateformes Java. Pour rappel, nous allons brièvement expliquer le fonctionnement de Swing.
Swing utilise un modèle de composants légers, ce qui signifie que les composants Swing sont indépendants de la plateforme et ne dépendent pas des composants natifs de la plateforme. Cela signifie que les composants Swing ont un aspect et un comportement cohérents sur toutes les plateformes (sur le papier en tout cas mais en pratique c'est pas toujours le cas). C'est cela qui le différencie de AWT (Abstract Window Toolkit) qui utilise des composants lourds qui dépendent des composants natifs de la plateforme. Cela ne signifie pas que Swing ne peut pas utiliser les composants natifs de la plateforme, on peut toujours utiliser les composants AWT dans Swing. Par contre on ne peut pas utiliser les composants Swing dans une application AWT. Le JFrame est la fenêtre principale de l'application Swing. C'est la fenêtre qui contient tous les autres composants Swing. Les composants Swing sont des objets qui héritent de la classe JComponent. On peut ajouter des composants Swing à un JFrame en utilisant la méthode add() de la classe Container. Il y a pas mal de composants qui sont disponibles dans Swing, comme le JPanel pour regrouper nos composants, le JButton pour créer un bouton programmable ou encore le JLabel pour afficher du texte.
Tout le rendu graphique est géré par Swing dans un thread séparé appelé Event Dispatch Thread (EDT). Il est important de ne pas bloquer l'EDT avec des opérations longues, sinon l'interface graphique deviendra non réactive. Il est aussi impossible de modifier les composants Swing en dehors de l'EDT, Swing n'est pas thread-safe. Cela ne veut pas dire qu'on ne peut pas utiliser plusieurs threads dans une application Swing, mais il faut faire attention à ne pas modifier les composants Swing en dehors de l'EDT. Fort heureusement, Swing fournit des méthodes pour exécuter du code dans l'EDT, comme la méthode invokeLater() de la classe EventQueue. Il est également possible de créer des threads SwingWorker pour exécuter des tâches longues en arrière-plan sans bloquer l'EDT.
Aussi il est tout à fait possible de créer ses propres composants Swing en étendant la classe JComponent ou une de ses sous-classes (comme JPanel). On peut alors redéfinir les méthodes paintComponent() et/ou paint() pour personnaliser le rendu du composant. C'est ce que nous avons fait pour la plupart des composants de notre jeu puisque la plupart des composants Swing manquent de fonctionnalités. Par exemple, le Jlabel ne permet pas d'avoir une bordure autour de notre texte, il a fallu créer notre propre composant pour cela.
Swing utilise la programmation événementielle pour gérer les interactions de l'utilisateur. Les composants Swing possèdent des écouteurs appelés EventListeners qui sont notifiés lorsqu'un événement que l'on souhaite écouter se produit. Les EventListeners sont des interfaces qui définissent des méthodes qui sont appelées lorsqu'un événement se produit. Les classes très utiles de Listeners sont les MouseListener qui permettent de gérer les événements de la souris (les clics, les déplacements, etc.) et les KeyListener qui permettent de gérer les événements du clavier (les touches pressées, relâchées, etc.). Il est également

possible de créer ses propres écouteurs en implémentant les différentes interfaces de Listener.
Enfin, il existe des LayoutManagers qui permettent de gérer la disposition des composants dans une fenêtre. On peut sinon hardcoder la position des composants mais il faudra alors gérer les redimensionnements de la fenêtre. Si nous n'avons pas de LayoutManager, il faudra donc définir la taille et la position de chaque composant à la main sinon ils ne s'afficheront pas. Cela peut être très fastidieux et compliqué, c'est pourquoi il est recommandé d'utiliser un LayoutManager lorsque c'est possible. Le problème est que les LayoutManagers de Swing ne sont pas toujours très flexibles. Ils sont souvent soit trop rigides et simplistes, soit trop complexes et difficiles à utiliser. C'est pourquoi il est souvent nécessaire de combiner plusieurs LayoutManagers pour obtenir le résultat souhaité.
Le framework Swing est assez ancien : il date de 1997 et a été inclu dans la JDK 1.2. Les fonctionnalités sont limités : il n'y a qu'une classe de Timer pour gérer les animations, pas de support pour les shaders, pas de support direct pour les effets sonores, la 3D et pas de décodeur vidéo intégré. Ceci n'est pas aidé par l'existence d'autres frameworks plus modernes et plus flexibles comme JavaFX qui inclue un support pour les animations, les effets sonores, la 3D et les vidéos (pour les shaders il faudra passer par OpenGL par exemple). Cela dit, il est tout à fait possible de combiner Swing avec JavaFX pour profiter des avantages des deux frameworks. On peut aussi utiliser des bibliothèques tierces pour ajouter des fonctionnalités manquantes à Swing, ou même créer ses propres composants personnalisés. Notre interface reste assez simple à cause de ces limitations et de la non utilisation de frameworks tiers.

3.6 Extensions

4 Difficultés Rencontrées

Implémentation des tuiles et des coordonnées choix -¿ on savait pas trop comment aborder le problème Implémentation de la vue assez difficile -¿ car contrainte hexagones + gestion des coordonnées Implémentation des règles du jeu -¿ pas forcément évident de les implémenter correctement Gestion de Swing qui n'est pas forcément très évident

5 Conclusion

6 Annexes

Contents