Romain Heurtaux,Matheo Jean-Marie-Lebon,Johann Paimboeuf / groupe Némée But 1 Informatique.

// Implementation

Ech3c_Java

I Lancer une partie

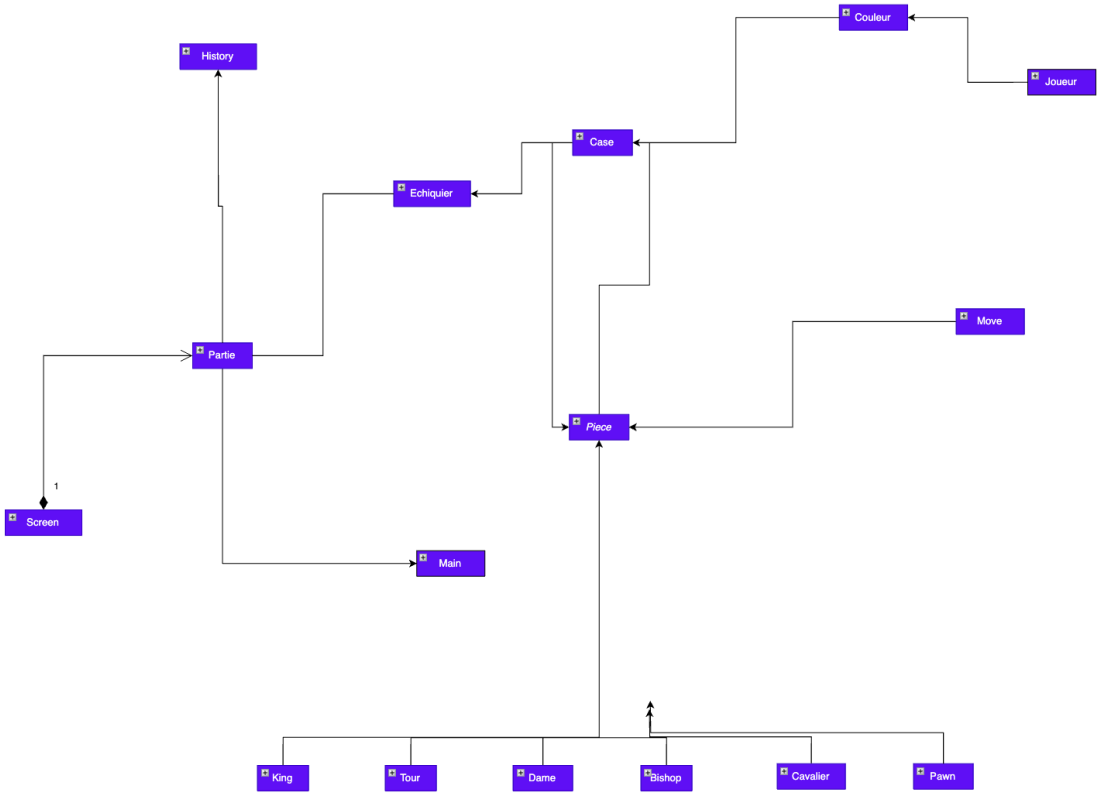
Avant tout, il s'agit d'un travail en équipe. Nous avons chacun contribué à différentes parties de ce projet, simultanément. Ainsi, nous avons utilisé GIT pour gérer le projet. De ce fait, le code source ainsi qu'une version compilée peuvent également être télécharger depuis le lien suivant : https://github.com/hrtxr/Ech3c_Java/. Pour lancer une partie d'échecs, nous avons mis en place un système simple et accessible via le terminal. Tout d'abord, il est nécessaire de compiler l'ensemble du projet. Pour cela, nous avons créé un script nommé compileAll.sh, qui automatise la compilation de toutes les classes réparties dans les différentes branches. Il suffit de se placer à la racine du projet à l'aide de la commande cd (si ce n'est pas déjà le cas), puis d'exécuter le script avec la commande suivante :

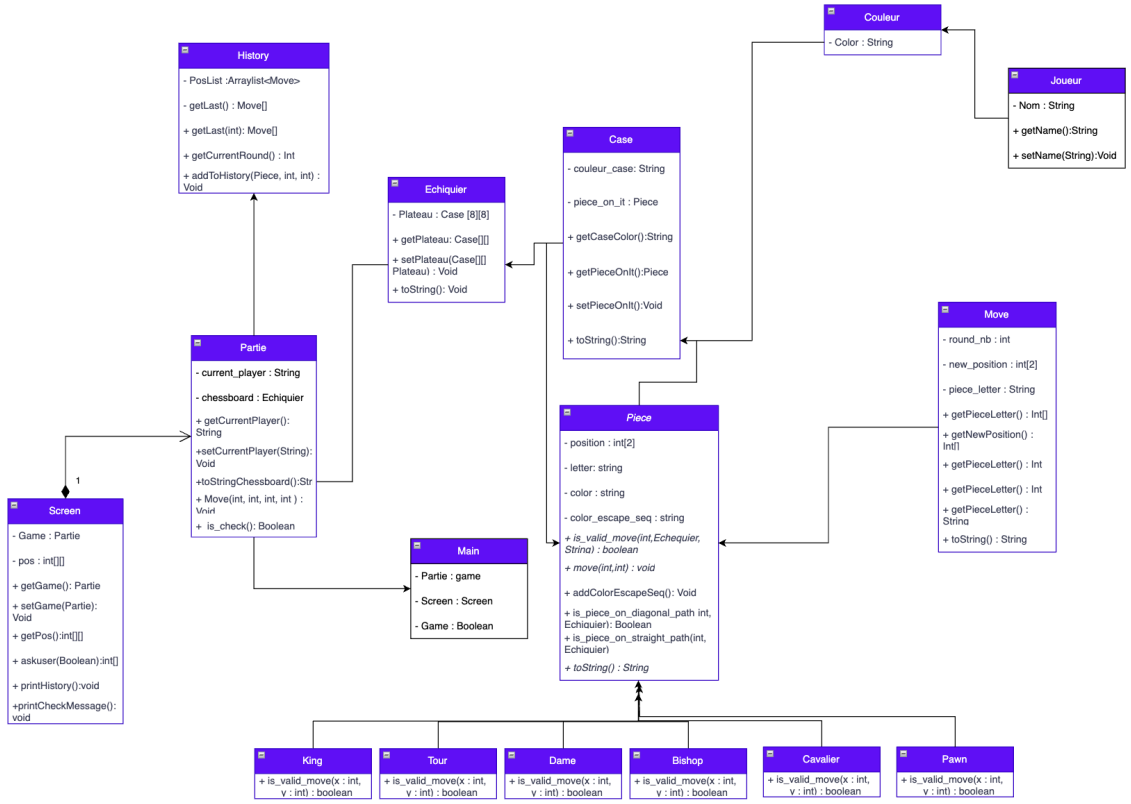
```
javac -cp bin/ -d bin/ Ech3c_Java/*.java
```

Ce script va compiler l'intégralité du code Java en s'appuyant sur la hiérarchie des packages et des dépendances entre classes. Une fois la compilation terminée sans erreur, on peut alors lancer le jeu en exécutant la classe principale du projet. Pour cela, il faut utiliser la commande java suivie du nom complet de la classe principale (avec son package si nécessaire). Par exemple :

```
java -cp bin Main
```

II Diagramme des classes





III Compte rendu de ce qui a été fait

Ce qui a été fait :

Modélisation des pièces avec une hiérarchie de classes :
Une classe abstraite Piece (supposée) sert de base aux différentes pièces (King, Queen, Rook, Bishop, Knight, Pawn). Chaque pièce implémente une méthode is_validMove propre à sa logique de déplacement, offrant ainsi une architecture claire et facilitant la maintenance du code.

Gestion basique des règles de déplacement :
Chaque pièce valide ses mouvements en fonction des règles spécifiques à son type (par exemple, le roi peut se déplacer d'une case dans toutes les directions, le cavalier selon un mouvement en L). Ces règles tiennent compte de la couleur du joueur, de la présence éventuelle d'une pièce amie sur la case de destination, et de la position actuelle de la pièce.

Gestion du plateau avec la classe Echiquier :
Le plateau conserve l'état des pièces sur les cases et propose une méthode getPiece permettant de récupérer la pièce présente sur une case donnée, ce qui facilite la vérification des mouvements et la gestion des captures.

Gestion de la partie dans la classe Partie :
Cette classe orchestre l'ordre des joueurs, traite les déplacements via la méthode Move, et assure le changement de joueur à chaque tour. La boucle principale de jeu est mise en œuvre dans la classe Main, avec une interaction utilisateur simplifiée via la classe Screen.

Affichage et interaction utilisateur :
La classe Screen se charge de l'affichage de l'échiquier et collecte les entrées utilisateur pour la sélection des pièces et des cases de destination, assurant ainsi un premier prototype fonctionnel et interactif.

Ce qui n'a pas été fait :
Plusieurs pistes d'amélioration peuvent être envisagées afin d'enrichir et de consolider ce projet d'échecs. L'intégration des règles avancées du jeu, telles que le roque, la promotion des pions et la prise en passant, permettrait de rendre les parties plus complètes et conformes aux règles officielles. La gestion précise de

l’état de la partie, incluant la détection des situations d’échec, d’échec et mat, ainsi que de pat, apporterait une dimension stratégique essentielle. Sur le plan de l’interface utilisateur, une évolution vers une interface graphique améliorerait significativement l’ergonomie et l’expérience de jeu. L’ajout de fonctionnalités interactives, telles qu’un historique complet des coups joués, la possibilité d’annuler des mouvements, ou encore un système de sauvegarde des parties, renforcerait l’aspect pratique et convivial de l’application. Du côté de la jouabilité, l’intégration d’une intelligence artificielle permettrait de proposer un mode solo avec différents niveaux de difficulté, tandis qu’un mode multijoueur en ligne élargirait les possibilités de confrontation. Par ailleurs, une architecture plus modulaire, assurant une séparation claire entre la logique du jeu et la présentation, faciliterait les évolutions futures, notamment l’ajout de tests automatisés et l’implémentation de design patterns pour améliorer la maintenabilité et la qualité du code. Ces améliorations contribuent à transformer ce projet, déjà bien structuré, en une application d’échecs complète, fonctionnelle et agréable à utiliser.

IV Répartition des Classes

Dans le cadre du projet de développement d'un jeu d'échecs, l'équipe s'est divisée le travail en trois branches principales, chacune centrée sur un aspect fondamental du fonctionnement du jeu. Chaque membre s'est ainsi vu confier une responsabilité bien définie, avec des classes spécifiques à concevoir, coder et tester. Voici une présentation détaillée des contributions de chacun :

🎨 Romain (branch **pieces**)

Romain s'est chargé de toute la modélisation des pièces d'échecs ainsi que de l'affichage graphique. Il a conçu une architecture orientée objet où chaque pièce hérite d'une classe de base commune, avec ses propres règles de déplacement.

- **Piece** : classe abstraite mère de toutes les pièces. Elle contient les attributs communs (position, couleur, etc.) et des méthodes génériques (comme **estDeplacementValide()** à surcharger).

- **Roi**, **Dame**, **Tour**, **Fou**, **Cavalier**, **Pion** : ces classes héritent de **Piece** et redéfinissent les règles spécifiques de déplacement. Chacune respecte les règles officielles du jeu d'échecs.

- **Screen** : gère l'affichage visuel du plateau et des pièces. Il est responsable de dessiner l'échiquier, de mettre à jour les mouvements à l'écran et d'assurer l'interaction graphique avec l'utilisateur.

Romain a donc apporté l'ossature visuelle et comportementale des pièces, en veillant à la cohérence et à l'intégration future avec la logique du jeu.

🧠 Mathéo (branch **players**)

Mathéo s'est occupé de tout ce qui concerne les joueurs, les mouvements et la logique associée aux couleurs et aux cases.

- **Move** : représente un mouvement effectué. Il enregistre les coordonnées de départ et d'arrivée, ainsi que la pièce déplacée. Essentiel pour le suivi des parties.

- **Joueur** : représente un joueur humain ou IA, avec ses informations (nom, couleur, etc.) et sa capacité à effectuer un coup.

- **Couleur** : classe ou enum définissant les deux couleurs possibles dans le jeu (blanc et noir), utilisée pour distinguer les camps.

- **Case** : représente une case de l'échiquier, avec sa position et la pièce qu'elle contient (ou non). C'est une brique centrale dans la gestion du plateau.

- **History** : garde en mémoire tous les coups joués. Permet de revenir en arrière, de vérifier l'historique et de détecter certaines conditions de fin (comme les répétitions).

Mathéo a donc géré toute la logique fonctionnelle liée aux joueurs et à leurs interactions avec le plateau, en structurant un système robuste pour suivre l'état de la partie.

♟️ Johann (branch **game**)

- Johann a conçu la structure générale du jeu, en particulier l'organisation du plateau et la gestion globale d'une partie d'échecs.
- **Echiquier** : classe représentant le plateau de jeu, initialisé avec les 64 cases et les pièces à leur position de départ. Elle orchestre les interactions entre les pièces, les mouvements, et les règles.

- **Partie** : classe maîtresse du jeu. Elle gère le déroulement d'une partie complète, l'alternance des joueurs, la détection des conditions de victoire (échec, échec et mat, pat, etc.) et l'appel aux autres classes (affichage, logique, etc.).

Grâce à Johann, le système dispose d'une gestion centralisée et fluide d'une partie d'échecs complète, rendant l'ensemble du projet cohérent, modulaire et fonctionnel.

Malgré la répartition des tâches, nous nous sommes régulièrement entraidés sur les différentes classes afin de garantir la cohérence du projet et de surmonter ensemble les éventuelles difficultés techniques.

Conclusion

Ce projet universitaire a permis de développer un moteur d'échecs structuré autour d'une architecture orientée objet claire et cohérente. La répartition des tâches entre les membres du groupe a favorisé une organisation efficace et une collaboration fructueuse, chaque contributeur prenant en charge un aspect fondamental du système : la modélisation des pièces, la gestion des joueurs et la coordination générale du déroulement de la partie.

Les fonctionnalités mises en œuvre permettent aujourd'hui de lancer une partie d'échecs avec un système basique de gestion des déplacements, une représentation du plateau et une interaction utilisateur simple via le terminal. Cette première version constitue ainsi un prototype fonctionnel qui pose des bases solides pour la poursuite du développement.

Plusieurs axes d'amélioration ont été identifiés afin d'enrichir le projet, notamment l'intégration des règles avancées du jeu, le perfectionnement de l'interface utilisateur, la gestion complète de l'état de la partie, ainsi que l'implémentation de fonctionnalités supplémentaires telles qu'un historique détaillé des coups, des options de sauvegarde, ou encore un module d'intelligence artificielle.

Ces évolutions permettront d'augmenter la robustesse, l'ergonomie et la complexité stratégique du jeu tout en conservant une architecture modulaire facilitant la maintenance et les développements futurs. Ce travail représente ainsi une étape importante dans la réalisation d'une application d'échecs complète et utilisable, qui pourra être enrichie dans le cadre d'études ou de projets ultérieurs.