Projet d'algo 2022

DAZY Margaux
HUGUET Cyril
Radjabou Mounir
DONGMO KENFACK Gausse

Jeu d'échecs

Cahier des charges

Le but de ce projet est de reproduire à l'aide d'une IHM Java principalement un jeu d'échecs, et ce, le plus fidèlement possible.

- Les deux joueurs qui s'affrontent, doivent pouvoir s'identifier sur la plateforme de jeu plusieurs fois, en conservant leurs informations dans une base de données
- Un temps limite est imparti pour chaque partie, qui doit pouvoir être décidé en amont par les joueurs, ainsi que la personne qui commence
- L'algorithme doit compter les points pour obtenir un score et annoncer le vainqueur
- Une intelligence artificielle doit être créée et capable de jouer contre un joueur si cette option est demandée

Problèmes principaux envisagés et rencontrés

L'un des plus gros défis de ce projet pour nous est de rendre possible le jeu contre l'IA. On a décidé de s'en occuper en dernier lieu, lorsque tout le reste aura été correctement fait. Une autre difficulté sera la gestion d'une base de données et les liens SQL-java, alors que nous n'avons que très peu de connaissances en la matière. Les spécificités du jeu d'échecs et ce qu'on appellera les "technicités du jeu", telles que le Roque, la Promotion du pion en reine ou encore l'échec et mat, sont autant de mini-challenges que nous souhaitions ajouter à la structure de base du jeu. Du point de vue de l'IHM, la liaison entre les différentes fenêtres, qui sont au nombre de 4, a demandé une attention particulière notamment parce que certaines variables créées dans l'une devaient pouvoir être réutilisées dans d'autres.

Description succincte d'une partie

Lors de l'exécution, une première fenêtre s'ouvre : c'est la fenêtre *Menu 1*, qui permet aux utilisateurs de choisir chacun leur tour un pseudo et un mot de passe. Ils peuvent choisir de se connecter. Dans ce cas, ils entrent un pseudo et un mot de passe déjà existants (vérification de leur exactitude avec la BD), ou de s'inscrire, et dans ce cas le programme vérifie bien qu'un utilisateur ne s'est pas déjà inscrit avec le même pseudo. Dans les deux cas, les pseudos sont mis dans des variables nom1 et nom2, pour être réutilisées plus tard.
Une fois les deux joueurs inscrits (ou un seul joueur s'il décide de jouer contre l'IA), une deuxième fenêtre s'ouvre ; c'est le *Menu 2.* Ici, on choisit le temps de jeu souhaité et quel

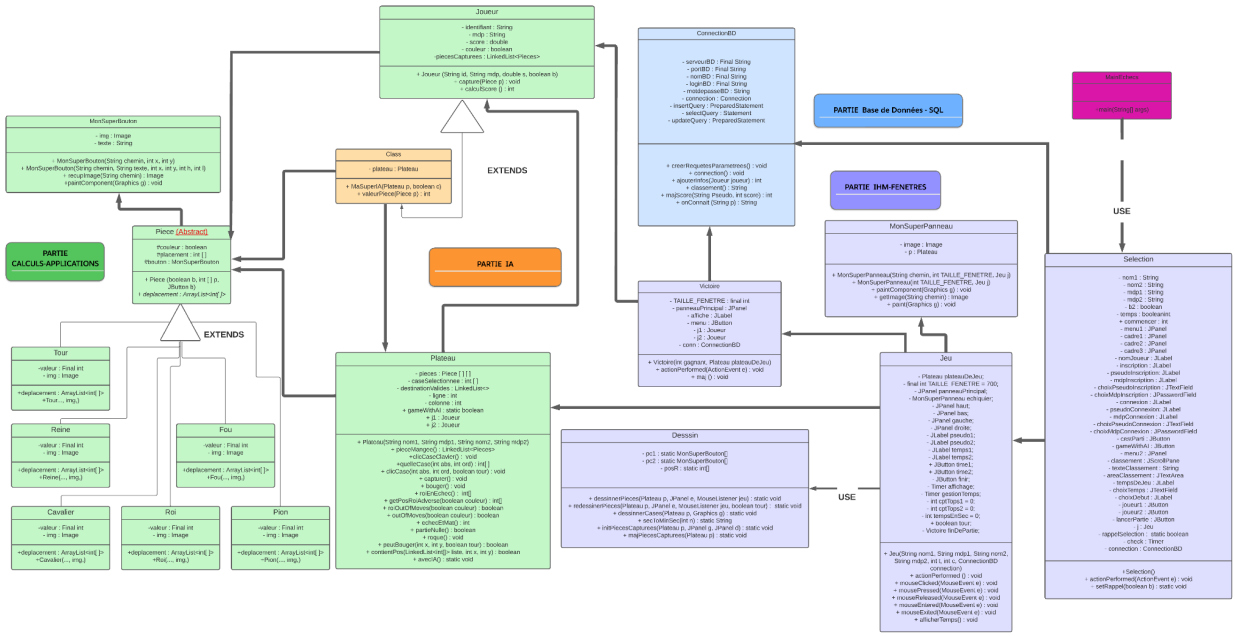
joueur commence la partie. A gauche de la fenêtre, le classement s'affiche, avec les joueurs précédemment inscrits et leur score final, pour motiver les joueurs.

La configuration de la partie maintenant terminée, c'est l'heure du Jeu : la fenêtre de jeu s'ouvre. Les deux joueurs ont chacun le même temps imparti pour réfléchir à leurs coups. Les pièces peuvent être déplacées sur l'échiquier selon bien sûr les déplacements autorisés pour chacune d'elles. Lorsqu'une pièce est "capturée" par le joueur adverse, celle-ci est comptabilisée sur le côté de l'échiquier, sous le dessin correspondant. En parallèle, la pièce est ajoutée à la liste de pièces capturées du joueur, ce qui permet ensuite de calculer son score (chaque pièce rapportant un certain nombre de points) ceci est aussi important lors de l'écriture de la fonction d'évaluation de l'IA.

Lorsque la partie prend fin (victoire de l'un ou match nul), une dernière fenêtre s'ouvre indiquant le joueur vainqueur ou l'égalité. Le score des joueurs est mis à jour dans notre BD ainsi que leur temps de jeu en minutes, afin d'actualiser le classement. A la fin, l'utilisateur peut s'il le souhaite revenir au *Menu 2*, pour recommencer une partie.

Diagramme UML

Notre code se sépare en 5 parties distinctes mais interdépendantes : le Main, la partie Calculs Application, l'IHM, l'IA et les Bases de données. Au total, 17 classes définissent ce projet. (Nous choisirons de ne pas préciser sur le diagramme UML les getters et setters.)



Vous pouvez consulter ce diagramme UML avec le lien :
https://lucid.app/lucidchart/0f5bf65f-670e-461a-94e7-6c18cfd32e20/edit?invitationId=inv_b35c17ca-5e20-4fc1-a4d5-35a512237435

1. Calcul Application

Cette partie est celle qui définit toutes les instances principales utilisées dans l'IHM. Voici quelques éclaircissements sur les classes qu'elle englobe :

1.1. Pièce

Classe abstraite, mère des classes suivantes : Roi, Reine, Cavalier, Tour, Fou et Pion. Elle possède les attributs communs à ces dernières : couleur, bouton, img et placement. Nous avons choisi d'en faire une classe abstraite car la méthode deplacement, qui renvoie la liste des déplacements possibles pour une pièce n'est pas implémentée de la même manière selon le type de pièce.

1.2. Roi, Reine, Cavalier, Tour, Fou, Pion

Sous-classes de Piece, ces classes représentent chacune un type de pièces posées sur l'échiquier. Elles contiennent chaque déplacement possible associé à la pièce, son image, ainsi que sa valeur, c'est-à-dire le nombre de points qu'elle rapporte.

1.3. Joueur

La couleur du joueur est définie par un booléen : noir -> false et blanc -> true (il est important de preciser que dans tout le reste du jeu la couleur blanche est assignée à true et au chiffre 2 et la couleur noire est assignée à false et 1).
L'attribut piecesCapturees est un tableau d'entiers : chaque indice (de 0 à 4) correspond à un type de pièce et lorsqu'une pièce dont le type est associé à l'indice i est capturée dans le jeu, on a piecesCapturees[i]++ grâce à la méthode *capture*. Enfin le score est calculé en faisant le produit de la valeur de la pièce par le nombre de pièces de ce type comptabilisées.

1.4. MonSuperBouton

Ici, le but est de trouver, avec les bons paramètres, l'image associée à une pièce et de créer une nouvelle instance : un *SuperBouton*. Deux constructeurs ont été établis pour les deux types de pièces existantes dans la fenêtre de jeu : les pièces sur le plateau et celles mangées. Trois méthodes sont présentes en plus : *recupImage*, *paintComponent*, et *setTextPerso.* La 1ere a pour rôle de récupérer l'image à partir de son chemin qui est sous la forme "images/pieceCouleur". Les 2 autres sont utilisées pour changer l'aspect visuel du bouton (on a "**Override**" la méthode paintComponent de la classe JButton de java) et *setTextePerso* permet de modifier l'attribut texte (utilisé pour afficher le nombre de pièces capturées)

1.5. Plateau

Cette classe crée le plateau et place les pièces dessus en fonction de leur couleur. Elle initialise aussi, dans le constructeur, les deux joueurs qui vont s'affronter, en fonction de leurs données personnelles.

- *quelleCase* permet d'avoir accès à la position d'une case cliquée.
- *clicCase* utilise la précédente pour différencier trois types de déplacements : l'impossibilité de bouger sur la case sélectionnée, un déplacement simple ou la capture

d'une pièce. Pour cela, il est nécessaire de vérifier les destinations possibles de la pièce et de tester la présence d'une autre pièce sur son trajet. Elle s'occupe aussi de la "dé-sélection" d'une case. Après un déplacement, la pièce change de position grâce à *bouger()* ou *capturer()* et l'ancienne est mise à *null*.

- *capturer* et *bouger* sont assez similaires : toutes deux testent la nature de la pièce et créent une nouvelle pièce à la position choisie. L'ancienne position voit son image changée en *null*. La différence entre les deux se situe dans *capturer*. En effet, elle vérifie la couleur de la pièce puis change le nombre de pièces mangées de la bonne couleur à l'aide de *capture* (une méthode de la classe Joueur).
- Les dernières méthodes sont spécifiques aux mouvements particuliers du roi : l'échec, l'égalité, le roque et l'échec et mat. Pour que ces 4 méthodes fonctionnent, une cinquième a été faite, c'est *getPosRoiAdverse* qui renvoie la position du roi. Elle utilise un break pour limiter les calculs non nécessaires de l'algorithme (tout comme *roiOutOfMove*, *roiEnEchec* et *roque*). En plus de celle-ci d'autres méthodes naissent dans cette classe au fur et à mesure que l'on progresse.

2. IHM

La partie IHM compte 5 classes permettant l'affichage de 4 fenêtres à la suite.
Le fonctionnement global et les interactions avec ces fenêtres ont été détaillés dans le **3)** de ce document. Quelques éclaircissements :

2.1. Selection

Beaucoup d'attributs, en partie choisis pour un aspect plus soigné de nos fenêtres. Plusieurs d'entre eux ont été placés à la main au lieu d'utiliser les layout dans un souci de précision et prédiction de futures améliorations.
A noter : l'attribut *rappelSelection* (choix de le créer en static pour l'utiliser dans la fenêtre Victoire) qui permet de revenir au Menu 2 après une partie et le timer *check* qui lui est lié.

2.2. Jeu

Cette fenêtre est consacrée uniquement au jeu avec ses 5 panneaux échiquier : haut, bas, gauche et droite qui sont utilisés pour contenir tous les JComponent. Nous soulignons l'importance des boutons *time1 et time2 ("finaliser")* qui permettent d'indiquer un changement de tour : lorsque le bouton *time1* est appuyé, le décompte du joueur 1 s'arrête et il n'a plus la main sur ses pièces. Le bouton *finir*, en bas à droite, permet de mettre fin à la partie.

2.3. Victoire

Cette fenêtre est créée dans Jeu, à la fin de la partie. Elle affiche une image différente selon le gagnant. Elle offre la possibilité de revenir au *Menu 2* pour recommencer une partie.

2.4. MonSuperPanneau

A la base la création de cette classe vient de notre souci de dessiner dans un panneau et non dans un JFrame, faire donc un classe qui hérite de JPanel était l'évidence. On en a profité pour redéfinir PaintComponent et on lui a donné des propriétés spécifiques..

2.5. Dessin

Le but principal de cette classe est d'alléger les classes Plateau, Dessin et Jeu en initialisant et en mettant à jour le dessin d'un échiquier. Vous remarquerez que toutes ses méthodes permettent la gestion du visuel et sont en **static.** C'est là que toutes les pièces sont dessinées et redessinées après chaque coup. C'est aussi là que les pièces sont calquées sur le côté de l'échiquier pour comptabiliser le nombre de pièces capturées par chaque joueur.

3. Base de données

La classe ConnectionBD gère toutes les interactions avec notre base de données. Créée sur le serveur de l'INSA, on peut y accéder avec un login et mot de passe. Nous avons trois différentes requêtes : *insertQuery (*ajouter un joueur à la base), *selectQuery* (récupérer puis afficher les informations des joueurs dans le classement), et *updateQuery* (mettre à jour le score de chacun à la fin). Nous utilisons des PreparedStatement car il est aisé de remplacer ensuite les points d'interrogation de la requête par la bonne donnée.

4. Intelligence Artificielle

La classe MaSuperIA permet la création de l'IA qui en fait est un joueur d'où l'héritage. Elle contiendra les méthodes permettant à l'IA d'évaluer les coups grâce à un algorithme min-max pour pouvoir jouer le plus intelligemment possible.

Améliorations possibles et bugs

Nous envisageons les améliorations suivantes :

- proposer un choix de thème général à l'utilisateur (assez facile grâce au constructeur de MonSuperPanneau)
- Inclure plus de propositions de modes différents dans le jeu (sur le temps total, le temps pour chaque coup…)
- mieux mettre à profit notre base de données en enregistrant le déroulement des parties, créer une table de fin de partie…
- finaliser la programmation de l'IA

Le principal bug du jeu se trouve au niveau de la fenêtre de Jeu : il s'agit de la difficulté que l'on a parfois à cliquer sur une case pour la sélectionner. Il faut parfois réessayer plusieurs fois pour que la case soit réellement cliquée. On émet l'hypothèse que ceci est dû à la mauvaise synchronisation entre le mouseListener() de l'échiquier et des pièces.
D'autres bugs sont à signaler : lors d'une deuxième partie (après être revenu au Menu), l'affichage du nombre de pièces capturées n'est pas stable.

Semainier

Disponible au lien ci-dessous :
https://docs.google.com/spreadsheets/d/1mKoD0lUTnCEOaB0VolUuSQId5njvHXfpKPAILOOumkQ/edit?usp=sharing

Forces et faiblesses

Forces :

- un code bien indenté avec des commentaires
- les noms des variables et méthodes sont assez parlants
- Le choix des structures de données permet une bonne utilisation de la mémoire (on privilégie l'utilisation des tableaux et aussi on a utilisé des LinkedList car on avait un fort taux d'insertion/suppression et moins d'accès aléatoires).
- Les parcours en FOR-EACH ont été privilégiés devant les boucles for classiques.
- Aucune librairies tierces n'a été utilisée dans la gestion de l'IHM

Faiblesses :

- la plupart des parcours du plateau sont des algorithme en O(8*8) sachant qu'on en fait au moins 4 toutes les 10 millisecondes c'est assez coûteux.
- les méthodes similaires auraient pu être associée (capturer() et bouger() ou encore roitOutOfMoves() et OutOfMoves())
- Possibilité de se passer de MouseListener
- Nous n'avons pas eu le temps de finaliser tout ce qui était lié à l'IA (mise à jour du score, du temps de jeu, il n'est pas possible de prendre en compte une victoire contre l'IA et celle-ci ne joue pas vraiment de manière intelligente).

Pourcentage d'implication

Gausse	Mounir	Margaux	Cyril
40%	30%	20%	10%

Bibliographie

Pour coder notre jeu d'échecs, nous avons utilisé :

- **Base de données et git:** les cours d'openclassroom.com
- **IHM et Grabcad :** site WayToLearnX.com, diapos de cours et TD du S3
- **SQL :** site mariadb.org, diapos de cours et openclassroom
- **Redéfinition des méthodes et la gestion des JComponent :** documentation java (principalement docs.oracle.com)
- **UML** : lucid.app
- **Compréhension et interprétation des erreurs SQL et Java :** askcodez.com, javawithus.com