Jeu d'échec

Java et Objet

L'objectif de ce projet est de réaliser un mini jeu d'échecs avec une interface graphique. On rappelle que ce jeu se déroule sur un échiquier de 8 cases par 8 cases, alternant cases blanches et cases noires. Deux joueurs humains (on ne s'intéressera pas ici à la réalisation d'une éventuelle intelligence artificielle), chacun d'entre eux doté de 8 pièces et 8 pions, s'affrontent sur cet échiquier. Chacun dispose d'un roi, et l'objectif pour son adversaire est de le mettre en *échec et mat*. La section suivante détaille plus précisément les règles qui seront considérées dans ce projet, car certaines règles seront ignorées.

Règles du jeu d'échecs

Dans un jeu d'échecs, le joueur blanc et le joueur noir disposent chacun d'un roi, d'une dame, de deux fous, de deux cavaliers, de deux tours (ce sont les 8 pièces), ainsi que de 8 pions. Chacune des pièces et pions d'un joueur a la même couleur que lui. Le joueur blanc commence la partie en jouant un *coup*, c'est-à-dire en effectuant un unique déplacement d'une de ses pièces ou d'un de ses pions. Puis le joueur noir joue un coup à son tour, et la partie se déroule ainsi en alternant les coups blancs et noirs, jusqu'à ce qu'elle se termine. Un pion ou une pièce ne peut jamais se déplacer sur une case occupée par un pion ou une pièce de la même couleur. Lorsqu'un pion ou une pièce se déplace sur une case occupée par une pièce ou un pion de l'autre couleur, on dit qu'elle *prend* cette pièce, qui est retirée de l'échiquier jusqu'à la fin de la partie (la prise, lorsqu'elle est possible, n'est jamais obligatoire sauf en cas d'échec). Les règles de déplacement des cinq types de pièces sont les suivantes :

- Le **roi** peut se déplacer d'une seule case dans n'importe quelle direction (diagonale, horizontale ou verticale),
- La **dame** peut se déplacer d'un nombre de cases quelconque dans toutes les directions (diagonale, horizontale ou verticale),
- La **tour** peut se déplacer d'un nombre de cases quelconque en ligne droite (horizontalement ou verticalement),
- Le **fou** peut se déplacer d'un nombre de cases quelconque en diagonale,
- Le **cavalier** peut se déplacer de trois cases en tout : deux cases horizontalement et une case verticalement, ou bien une case horizontalement et deux cases verticalement.

Une pièce ne peut jamais se déplacer *à travers* une case déjà occupée, sauf le cavalier, qui a la particularité de sauter par-dessus les autres pièces et pions. Les règles de déplacement des **pions** dépendent du contexte :

- Un pion ne peut prendre une pièce adverse qu'en avançant en diagonale d'une case (il ne peut jamais prendre en reculant),
- Lorsqu'un pion ne s'est pas encore déplacé depuis le début de la partie, il peut avancer d'une ou deux cases verticalement (il ne peut pas reculer), si aucune pièce ou aucun pion (quelle que soit sa couleur) ne se trouve sur son passage,
- Lorsqu'un pion s'est déjà déplacé dans la partie, il ne peut avancer que d'une case verticalement (il ne peut pas reculer),
- Lorsqu'un pion a traversé l'échiquier, il se transforme en pièce (fou, cavalier, tour ou dame) du choix du joueur.

Un roi est dit en **échec** lorsqu'une pièce ou un pion adverse le menace, c'est-à-dire est en mesure de le prendre à son prochain tour si le joueur du roi menacé ne fait rien pour l'éviter. Dans un tel cas, le joueur du roi doit tout faire, à son tour de jeu, pour que son roi ne soit plus en échec (soit en prenant la pièce qui le menace, soit en déplaçant son roi, soit en mettant une de ses pièces entre

son roi et la pièce qui le menace). De même, un joueur ne peut jamais jouer un coup s'il conduit à mettre son propre roi en échec. Si le roi d'un joueur est en échec et qu'il ne peut rien faire pour mettre fin à cette situation, on dit que ce joueur est **échec et mat** : il perd donc la partie, et son adversaire est déclaré vainqueur.

Si un joueur ne peut, à son tour de jeu, effectuer un seul déplacement sans mettre son propre roi en échec, mais que ce dernier n'est pas déjà en échec, alors on dit qu'il y a **pat**, et la partie est déclarée nulle.

Dans un premier temps, nous ne considérerons, dans le cadre de ce projet, que l'**échec et mat** et le **pat** comme fins possibles pour une partie. Voici les règles qui ne seront pas considérées dans ce projet :

- Le **roque** (coup impliquant un roi et une tour) ne sera pas possible,
- Lorsqu'un pion traversera l'échiquier, il se transformera en dame (il ne pourra se transformer ni en fou, ni en cavalier, ni en tour),
- Un pion ne pourra jamais effectuer de **prise en passant** (une règle de prise spéciale, qui peut normalement s'appliquer quand un pion adverse avance de deux cases),
- La **règle des 50 coups**, qui stipule que la partie peut être déclarée nulle par un des joueurs si cinquante déplacements consécutifs (noirs et blancs confondus) ont été effectués sans qu'aucune pièce ne soit prise et sans qu'aucun pion n'ait bougé, ne sera pas appliquée,
- La règle de la **répétition de situation**, qui stipule que la partie peut être déclarée nulle si une même disposition de pièces se reproduit trois fois lors d'une partie, ne sera pas appliquée non plus.

Architecture proposée du projet

En plus des fonctionnalités relatives aux règles du jeu des échecs qui permettront de vérifier les déplacements des pièces par les joueurs, l'idée est de développer une interface graphique pour rendre l'utilisation du jeu plus facile. Pour cela, la méthode principale du jeu `main` consistera au lancement de la fenêtre principale du jeu. Ce seront ensuite les clics souris des joueurs qui détermineront les actions à effectuer. La fenêtre du jeu est illustrée en Figure 1.

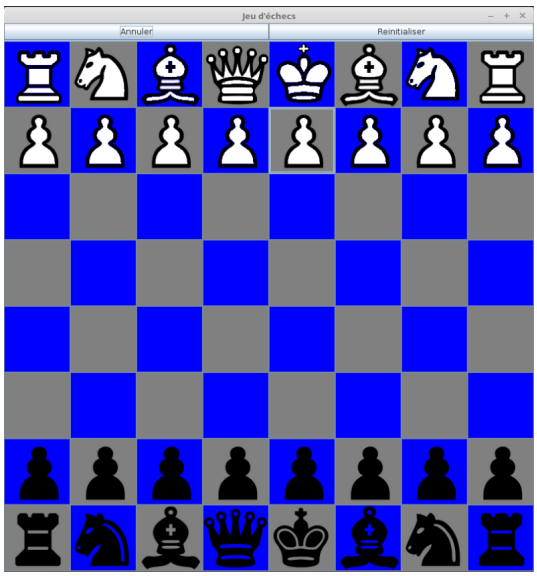

FIGURE 1 – Fenêtre principale du jeu d'échecs

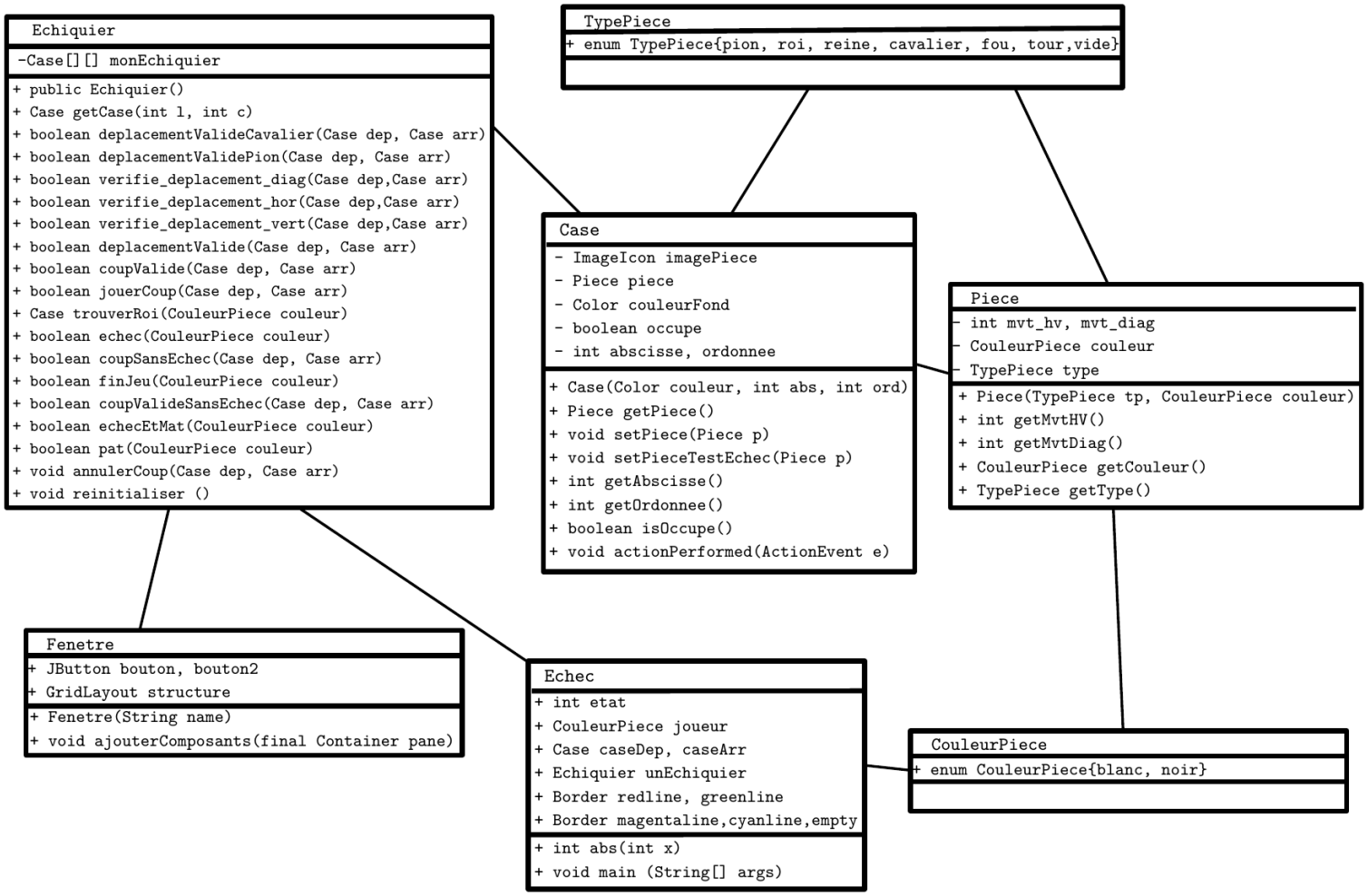

FIGURE 2 – Architecture proposée pour le jeu d'échecs

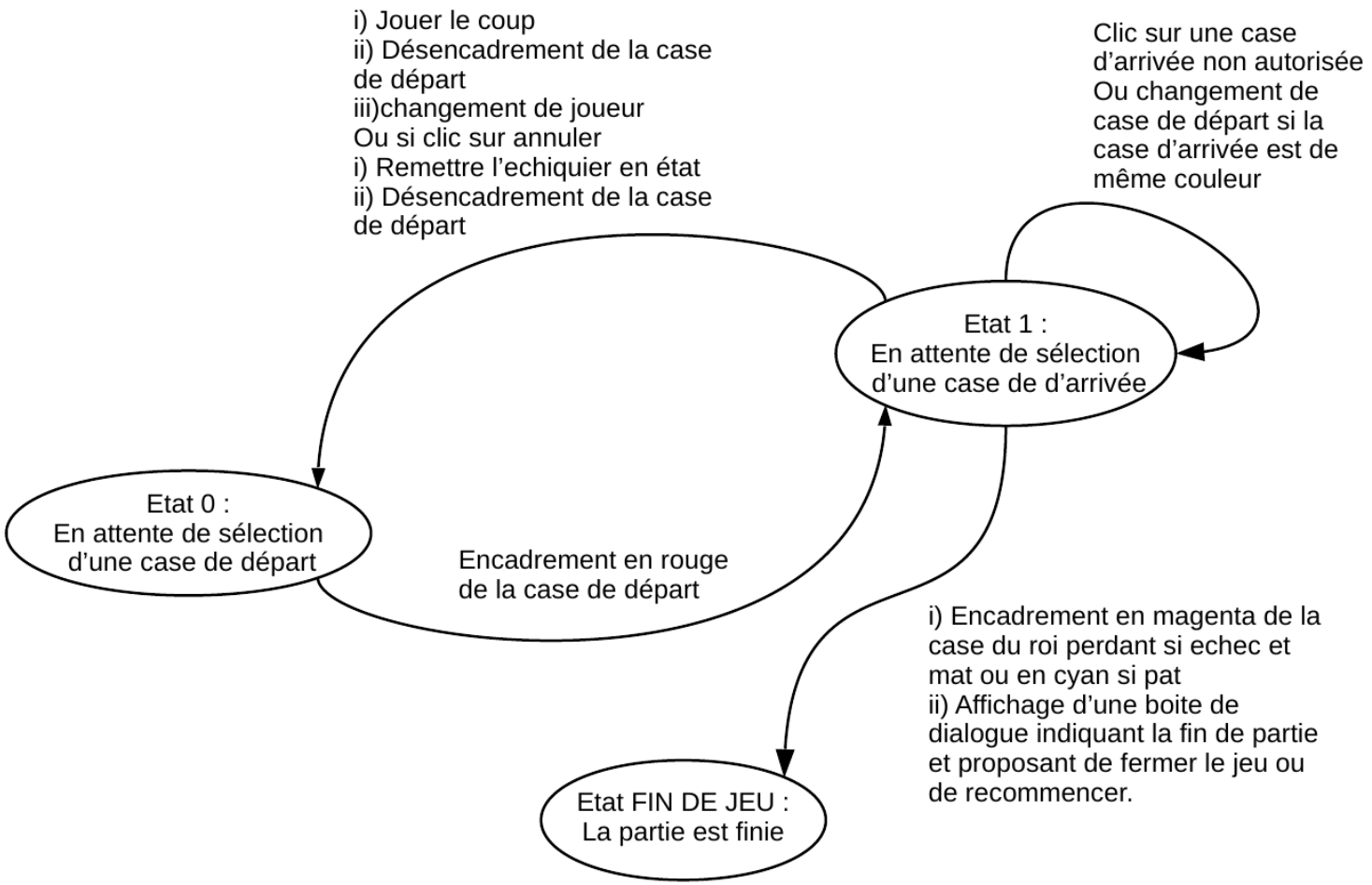

FIGURE 3 – Graphe d'états du jeu d'échecs

L'architecture du projet est présentées en Figure 2. Elle sera composée de 7 classes :

— Les classes `TypePiece` et `CouleurPiece` qui seront des enumerations des types et couleurs possibles des pièces.

— La classe `Piece` qui representera les différentes pièces du jeu avec leurs caractéristiques, et en particulier, leur `couleur`, `type`, et amplitudes de déplacements horizontaux, verticaux ou diagonaux autorisés (`mvt_hv`,`mvt_diag`). Cette classe fournira un constructeur et des accesseurs.

— La classe `Case` qui heritera de la classe `JButton`. Il sera donc possible d'effectuer une action en fonction d'un clic sur une `Case`. Elle sera composée de plusieurs attributs. Deux attributs propres aux boutons : sa couleur de fond de type `Color` et une image si elle est occupée par une pièce de type `ImageIcone`. Elle aura également comme attributs un `boolean` qui indique si elle est occupée, une pièce, et de ses coordonnées dans l'échiquier. Cette classe fournira un constructeur et des accesseur et des modificateurs. Elle fournira également l'implémentation d'une méthode particulière `ActionPerformed` qui indiquera les actions à effectuer en fonctions des clics sur la fenêtre et de l'état du jeu. Le graphe d'état de la Figure 3 décrit les différents états du jeu que nous allons considérer.

— La classe `Echiquier` qui héritera de la classe `JPanel`. Un échiquier sera uniquement composé d'un tableau de `8` $\times$ `8` `Case`. L'échiquier sera ensuite ajouté à la fenêtre principale du jeu. C'est dans cette classe que les règles du jeu des échecs seront implémentées. Le constructeur initialisera la place des pièces : les 8 pièces blanches seront placées sur la ligne 0, dans cet ordre : tour, cavalier, fou, dame, roi, fou, cavalier, tour. Puis, les 8 pions blancs seront placés sur la ligne 1. Ensuite, les 8 pions noirs seront placés sur la ligne 6. Enfin, 8 pièces noires seront placées sur la ligne 7, dans cet ordre : tour, cavalier, fou, dame, roi, fou, cavalier, tour. Un accesseur retournera la case de l'échiquier dont les coordonnées seront passées en paramètre. En plus, cette classe implémentera les méthodes suivantes :

  — `boolean deplacementValideCavalier(Case dep, Case arr)` qui vérifie que le déplacement d'une pièce de type `cavalier` entre les cases `dep` et `arr` est valide. Plus précisément, il faut vérifier que la distance entre la case d'arrivée et la case de départ est de deux cases horizontalement et une case verticalement, ou bien deux cases verticalement et une case horizontalement.

  — `boolean deplacementValidePion(Case dep, Case arr)` qui vérifie que le déplacement d'une pièce de type `pion` entre les cases `dep` et `arr` est valide. Plus précisément, il faut prendre en compte les trois cas : si la case de destination est occupée par une pièce ou un pion d'une autre couleur, alors il faut vérifier que l'on *avance* d'une case en diagonale ; sinon il faut vérifier que l'on *avance* d'une ou deux cases si le pion ne s'est pas déjà déplacé (c'est-à-dire s'il est toujours sur sa ligne de départ) et d'une seule case dans le cas contraire.

  — `boolean verifieDeplacementDiag(Case dep,Case arr)` qui vérifie que le déplacement diagonal d'une pièce entre les cases `dep` et `arr` est possible : qu'il n'y a pas de pièce sur le chemin.

  — `boolean verifieDeplacementHor(Case dep,Case arr)` qui vérifie que le déplacement horizontal d'une pièce entre les cases `dep` et `arr` est possible : qu'il n'y a pas de pièce sur le chemin.

  — `boolean verifieDeplacementVert(Case dep,Case arr)` qui vérifie que le déplacement vertical d'une pièce entre les cases `dep` et `arr` est possible : qu'il n'y a pas de pièce sur le chemin.

  — `boolean deplacementValide(Case dep, Case arr)` qui vérifie que le déplacement d'une pièce de n'importe quel type entre les cases `dep` et `arr` est valide. Plus précisément, si on déplace un roi, une reine, une tour ou un fou, il faut d'abord vérifier que le déplacement est de l'un des 3 types suivant : horizontal, vertical, ou diagonal. Ensuite, il faut vérifier que la pièce considérée est autorisée à se déplacer d'autant de cases horizontalement, verticalement ou en diagonale (selon le cas). Par exemple, le roi est autorisé à se déplacer d'une case horizontalement, mais un fou ne le serait pas (par contre, il serait autorisé à se déplacer de deux cases en diagonale). Enfin, il faut vérifier que toutes les cases situées sur la ligne horizontale, verticale ou diagonale (selon le cas) reliant la case de départ à la case de destination sont inoccupées.

- `boolean coupValide(Case dep, Case arr)` qui renvoie `true` si le coup est valide. Il faudra d'abord vérifier que la case d'arrivée est vide ou que si elle est occupée, c'est bien par une pièce de la couleur adverse, puis que le déplacement souhaité est un déplacement valide.
- `boolean jouerCoup(Case dep, Case arr)` qui consistera à déplacer la pièce souhaitée si le coup est valide, la méthode renverra `true` dans ce cas. Attention, si la pièce sélectionnée est un pion et que la case d'arrivée est la ligne 0 pour les noirs ou 7 pour les blancs, il faudra en plus transformer la pièce en reine.
- `Case trouverRoi(CouleurPiece couleur)` qui retourne la case où se trouve le roi de la couleur passée en paramètre.
- `boolean echec(CouleurPiece couleur)` qui renvoie `true` si le joueur de la couleur passée en paramètre est en échec.
- `boolean coupSansEchec(Case dep, Case arr)` qui renvoie `true` si le déplacement de la pièce de la case `dep` vers la case `arr` ne met pas le roi du joueur courant en echec.
- `boolean finJeu(CouleurPiece couleur)` qui renvoie `true` si le joueur de la couleur passée en paramètre peut encore déplacer au moins une de ces pièces sans se mettre en echec.
- `boolean coupValideSansEchec(Case dep, Case arr)` qui renvoie `true` si le déplacement de la pièce de la case `dep` vers la case `arr` ne met pas le roi du joueur courant en echec et que c'est un coup valide.
- `boolean echecEtMat(CouleurPiece couleur)` qui retourne `true` si le joueur de couleur la passée en paramètre est en echec et mat.
- `boolean pat(CouleurPiece couleur)` qui retourne `true` si il y a pat pour le joueur de la couleur passée en paramètre.
- `void annulerCoup(Case dep, Case arr)` qui annulera le coup précédent. On supposera ici qu'il n'est possible d'annuler qu'un seul coup.
- `void reinitialiser()` qui reinitialisera la partie.

— La classe `Fenetre` qui étend la classe `JFrame`. Elle sera composée de deux boutons (`annuler` et `reinitialiser`) et d'une structure de type `Gridlayout` où l'échiquier sera affiché. Elle possède un constructeur et la méthode `ajouterComposants` qui ajoute ses attributs à la fenêtre affichée.

— La classe principale `Echec` qui contient des variables globales et la méthode principale `main`. Les variables globales sont l'`etat` du jeu (0 ou 1), le `joueur` courant, les cases de départ et d'arrivée du prochain coup, l'échiquier et des variables de type `Border` pour encadrer les cases de couleurs différentes en fonction des situations du jeu : `redline`, `greenline`, `magentaline` ,`cyanline` et `empty`. La methode `main` consistera à créer la fenêtre principale et à la lancer.

Travail à rendre

Le rendu du projet se décompose en deux parties : le code et un rapport, chacun étant rendu en binôme :

— Votre projet contiendra dans une même archive (`.zip`), les sources java et un rapport (`.pdf`). Ce rapport précisera le périmètre de votre programme, ses limites et si nécessaire les choix effectués. Il contiendra en outre l'architecture des classes de votre application.

— Le développement de ce programme pourra être réalisé en binôme. Chacun ayant la responsabilité d'une partie du développement. Les deux noms devront apparaître dans la description du projet. Chaque fichier source contiendra le nom du responsable de son codage.

La date ultime de remise est le 9 juin 2019 avant minuit sur la plateforme `claroline`. Attention, au delà de cette date, il vous sera impossible de rendre votre travail. Aucune dérogation ne sera faite. Pour plus de sécurité, ne pas attendre la dernière limite. On peut rendre son travail autant de fois que désiré. Chaque nouvelle version écrase la précédente.

Le plagiat n'est pas autorisé. Une comparaison automatique des sources sera effectuée pour vérifier la facture personnelle du travail rendu.