Rapport de Projet – JavaChess

1. Présentation

JavaChess est un jeu d'échecs développé en Java avec une interface graphique basée sur Swing. Le projet respecte une architecture MVC et implémente les règles classiques d'un jeu d'échecs (déplacements, roque, prise en passant, promotion, échec et mat, pat). Plusieurs extensions ont également été ajoutées.

Le projet a été structuré avec Maven, permettant notamment de faciliter la création d'un exécutable (*.jar*).

2. Fonctionnalités de base

Fonctionnalité	Note	Temps de travail
Déplacement des pièces	Calcul des cases accessibles avec des Décorateurs	10h
Application des coups	Application d'un coup joué	6h
Calcul des fins de partie	Détection par échec et mat ou pat	3h
Vue graphique Swing	Affichage du plateau et des pièces	6h

3. Extensions

Extension	Note	Temps de travail
Vue console	Une vue permettant de jouer dans la console	5h
Intelligence artificielle	Possibilité de jouer contre une IA avec niveau de difficulté ajustable	2h
Sauvegarde / chargement PGN	Exportation & importation des parties au format PGN (historique des coups)	7h
Sauvegarde / chargement FEN	Exportation & importation des parties au format FEN (position statique des pièces)	3h
Fin de partie par répétition	Détection d'une répétition de position à l'aide d'une HashMap	2h
Échecs 960 (Échecs aléatoires Fischer)	Variante avec position initiale aléatoire	4h
Menu de sélection du mode de jeu	Interface de sélection du mode de jeu : IA, 2 joueurs, Chess960 et chargement d'une partie	5h

4. Architecture

Le projet suit une architecture MVC (Modèle Vue Contrôleur). Les sources (fichiers java et ressources) sont placées dans le dossier ***src*** et la documentation dans le dossier ***doc***.

L'application a été séparé en plusieurs packages et classes :

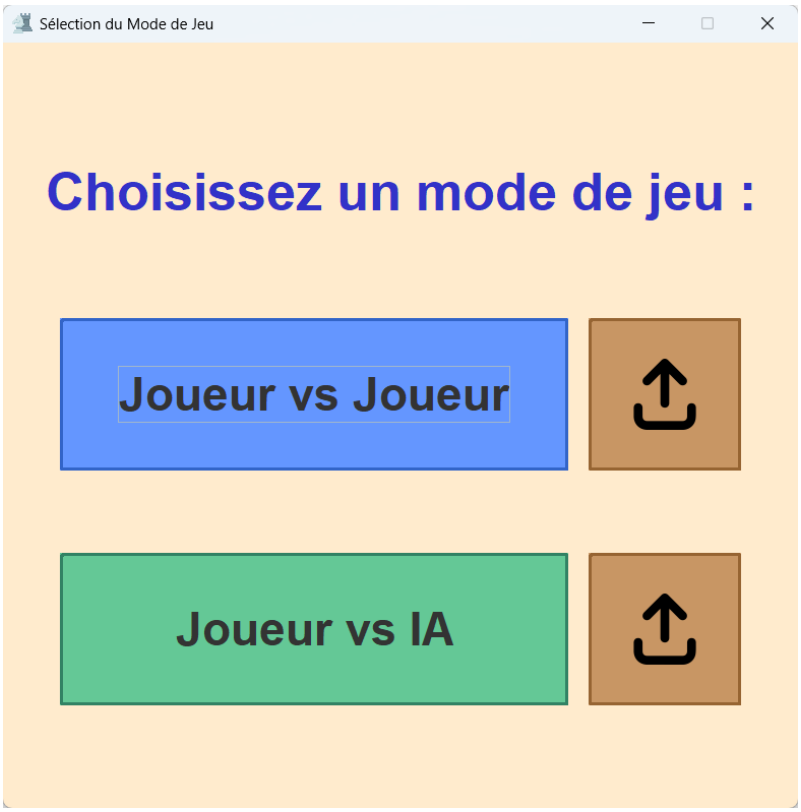
- **modele**
  - **jeu** : Gère la logique principale du jeu.
    - *Jeu* : Classe centrale du modèle, elle contient la boucle principale du jeu.
    - *Joueur* & *JoueurIA* : Représente un joueur humain et une IA respectivement.
    - *Coup* : Modélise un coup avec une position de départ et d'arrivée.
    - *Couleur* : Enum définissant les couleurs des joueurs et des pièces (BLANC, NOIR).
    - *GameEvent* : Enum représentant les événements de jeu.
  - **pieces** : Contient les pièces d'un jeu d'échec.
    - *Piece* : Classe abstraite définissant les caractéristiques des pièces.
    - *Pion*, *Tour*, *Cavalier*, *Fou*, *Dame*, *Roi* : Sous classe représentant chaque pièce avec ses mouvements associés.
  - **mouvement** : Implémente les règles de déplacement via des décorateurs.
    - *DecoratorCasesAccessibles* : Classe abstraite définissant une interface commune pour les décorateurs.
    - *DecoratorLigne*, *DecoratorDiag*, *DecoratorCavalier*, *DecoratorPion*, *DecoratorRoi* : gèrent les mouvements en ligne, en diagonale, du cavalier, du pion et du roi.
  - **plateau** : Modélise le plateau et ses cases
    - *Plateau* : Représente le plateau et permet l'accès et la modification des cases.
    - *Case* : Représente une case.

- **vue**
  - *MenuSelection* : Choix du mode de jeu (2 joueurs, IA, Chess960, chargement de partie).
  - *VueControleur* : Interface graphique Swing (affichage du plateau, gestion des clics).
  - *MenuSelectionConsole* & *VueControleurConsole* : Version console.
  - *SoundPlayer* : Classe outil permettant de jouer un son.
- *Main* : point d'entrée de l'application

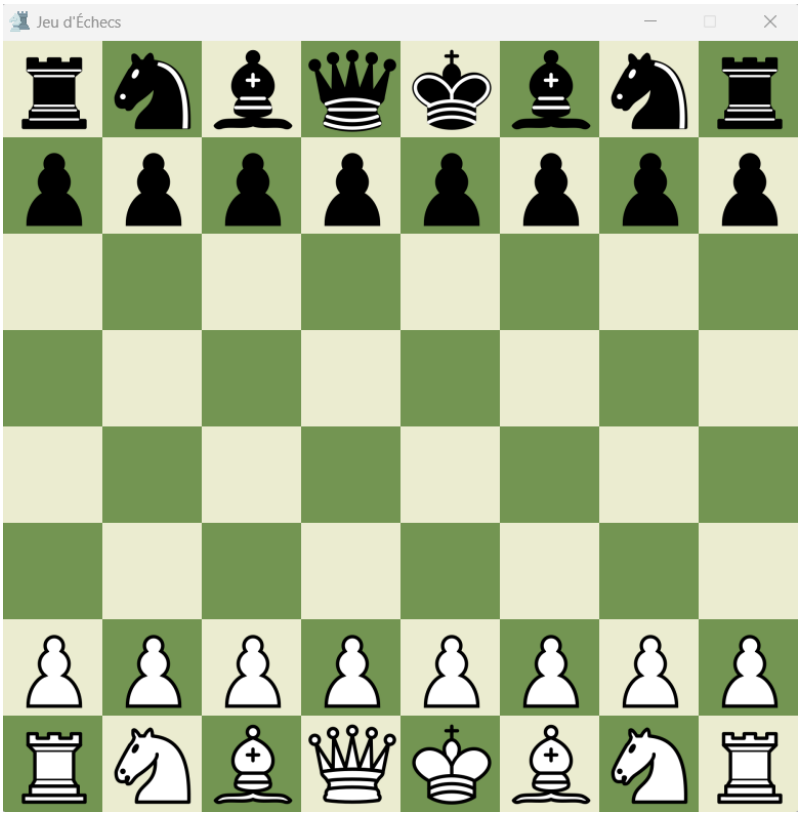
> *Voir diagramme des classes en annexe ou dans le dossier **doc**.*

5. Captures d'écran

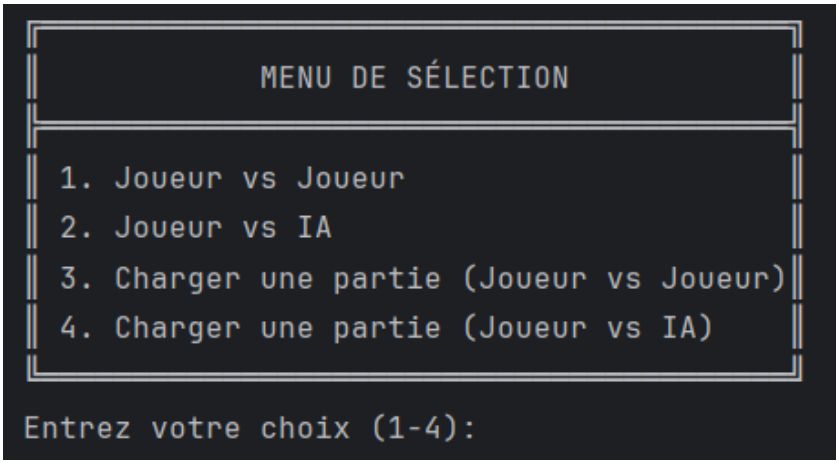
Menu de sélection du mode de jeu :



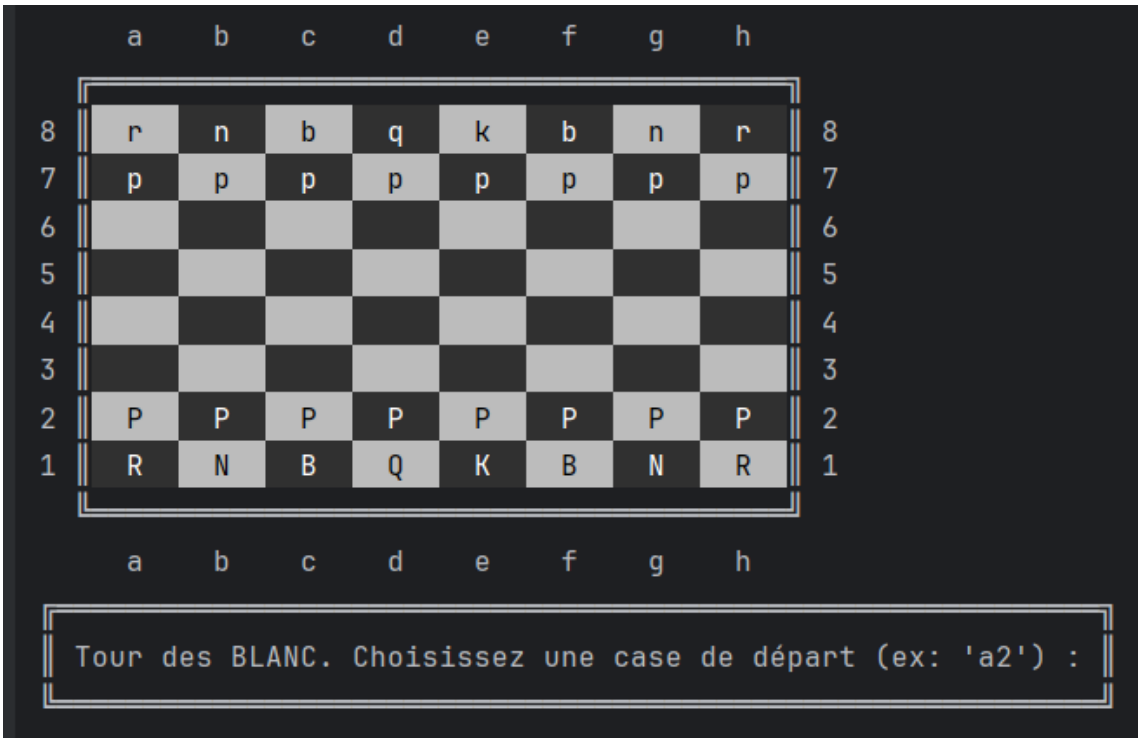
Plateau version graphique :



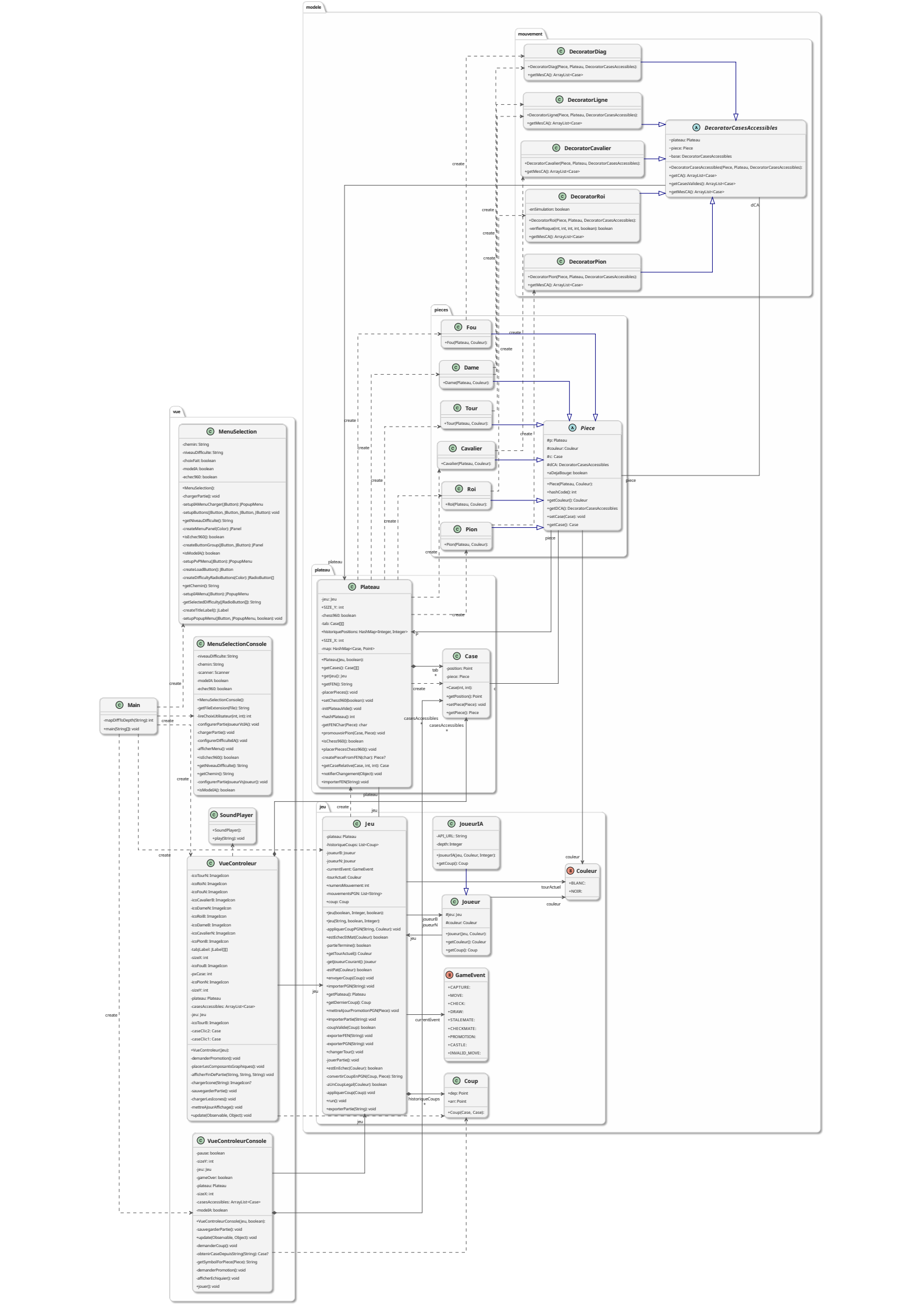
Menu de sélection version console :



Plateau version console :



6. Annexes


modele
mouvement
DecoratorDiag
+DecoratorDiag(Piece, Plateau, DecoratorCasesAccessibles):
+getMesCA(): ArrayList<Case>
DecoratorLigne
+DecoratorLigne(Piece, Plateau, DecoratorCasesAccessibles):
+getMesCA(): ArrayList<Case>
DecoratorCasesAccessibles
-plateau: Plateau
-piece: Piece
-base: DecoratorCasesAccessibles
+DecoratorCasesAccessibles(Piece, Plateau, DecoratorCasesAccessibles):
+getCA(): ArrayList<Case>
+getCasesValides(): ArrayList<Case>
+getMesCA(): ArrayList<Case>
DecoratorCavalier
+DecoratorCavalier(Piece, Plateau, DecoratorCasesAccessibles):
+getMesCA(): ArrayList<Case>
DecoratorRoi
-enSimulation: boolean
+DecoratorRoi(Piece, Plateau, DecoratorCasesAccessibles):
-verifierRoque(int, int, int, int, boolean): boolean
+getMesCA(): ArrayList<Case>
DecoratorPion
+DecoratorPion(Piece, Plateau, DecoratorCasesAccessibles):
+getMesCA(): ArrayList<Case>
pieces
Fou
+Fou(Plateau, Couleur):
Dame
+Dame(Plateau, Couleur):
Tour
+Tour(Plateau, Couleur):
Cavalier
+Cavalier(Plateau, Couleur):
Roi
+Roi(Plateau, Couleur):
Pion
+Pion(Plateau, Couleur):
Piece
#p: Plateau
#couleur: Couleur
#c: Case
#dCA: DecoratorCasesAccessibles
+aDejaBouge: boolean
+Piece(Plateau, Couleur):
+hashCode(): int
+getCouleur(): Couleur
+getDCA(): DecoratorCasesAccessibles
+setCase(Case): void
+getCase(): Case
plateau
Plateau
-jeu: Jeu
+SIZE_Y: int
-chess960: boolean
-tab: Case[][]
+historiquePositions: HashMap<Integer, Integer>
+SIZE_X: int
-map: HashMap<Case, Point>
+Plateau(Jeu, boolean):
+getCases(): Case[][]
+getJeu(): Jeu
+getFEN(): String
-placerPieces(): void
+setChess960(boolean): void
-initPlateauVide(): void
+hashPlateau(): int
-getFENChar(Piece): char
+promouvoirPion(Case, Piece): void
+isChess960(): boolean
+placerPiecesChess960(): void
-createPieceFromFEN(char): Piece?
+getCaseRelative(Case, int, int): Case
+notifierChangement(Object): void
+importerFEN(String): void
Case
-position: Point
-piece: Piece
+Case(int, int):
+getPosition(): Point
+setPiece(Piece): void
+getPiece(): Piece
vue
MenuSelection
-chemin: String
-niveauDifficulte: String
-choixFait: boolean
-modeIA: boolean
-echec960: boolean
+MenuSelection():
-chargerPartie(): void
-setupIAMenuCharger(JButton): JPopupMenu
-setupButtons(JButton, JButton, JButton, JButton): void
+getNiveauDifficulte(): String
-createMenuPanel(Color): JPanel
+isEchec960(): boolean
-createButtonGroup(JButton, JButton): JPanel
+isModeIA(): boolean
-setupPvPMenu(JButton): JPopupMenu
-createLoadButton(): JButton
-createDifficultyRadioButtons(Color): JRadioButton[]
+getChemin(): String
-setupIAMenu(JButton): JPopupMenu
-getSelectedDifficulty(JRadioButton[]): String
-createTitleLabel(): JLabel
-setupPopupMenu(JButton, JPopupMenu, boolean): void
MenuSelectionConsole
-niveauDifficulte: String
-chemin: String
-scanner: Scanner
-modeIA: boolean
-echec960: boolean
+MenuSelectionConsole():
-getFileExtension(File): String
-lireChoixUtilisateur(int, int): int
-configurerPartieJoueurVsIA(): void
-chargerPartie(): void
-configurerDifficulteIA(): void
-afficherMenu(): void
+isEchec960(): boolean
+getNiveauDifficulte(): String
+getChemin(): String
-configurerPartieJoueurVsJoueur(): void
+isModeIA(): boolean
Main
-mapDiffToDepth(String): int
+main(String[]): void
SoundPlayer
+SoundPlayer():
+play(String): void
jeu
Jeu
-plateau: Plateau
-historiqueCoups: List<Coup>
-joueurB: Joueur
-joueurN: Joueur
-currentEvent: GameEvent
-tourActuel: Couleur
+numeroMouvement: int
-mouvementsPGN: List<String>
+coup: Coup
+Jeu(boolean, Integer, boolean):
+Jeu(String, boolean, Integer):
-appliquerCoupPGN(String, Couleur): void
+estEchecEtMat(Couleur): boolean
-partieTermine(): boolean
+getTourActuel(): Couleur
-getJoueurCourant(): Joueur
-estPat(Couleur): boolean
+envoyerCoup(Coup): void
+importerPGN(String): void
+getPlateau(): Plateau
+getDernierCoup(): Coup
+mettreAJourPromotionPGN(Piece): void
+importerPartie(String): void
-coupValide(Coup): boolean
-exporterFEN(String): void
-exporterPGN(String): void
+changerTour(): void
-jouerPartie(): void
+estEnEchec(Couleur): boolean
-convertirCoupEnPGN(Coup, Piece): String
-aUnCoupLegal(Couleur): boolean
-appliquerCoup(Coup): void
+run(): void
+exporterPartie(String): void
JoueurIA
-API_URL: String
-depth: Integer
+JoueurIA(Jeu, Couleur, Integer):
+getCoup(): Coup
Joueur
#jeu: Jeu
#couleur: Couleur
+Joueur(Jeu, Couleur):
+getCouleur(): Couleur
+getCoup(): Coup
Couleur
+BLANC:
+NOIR:
GameEvent
+CAPTURE:
+MOVE:
+CHECK:
+DRAW:
+STALEMATE:
+CHECKMATE:
+PROMOTION:
+CASTLE:
+INVALID_MOVE:
Coup
+dep: Point
+arr: Point
+Coup(Case, Case):
VueControleur
-icoTourN: ImageIcon
-icoRoiN: ImageIcon
-icoFouN: ImageIcon
-icoCavalierB: ImageIcon
-icoDameN: ImageIcon
-icoRoiB: ImageIcon
-icoDameB: ImageIcon
-icoCavalierN: ImageIcon
-icoPionB: ImageIcon
-tabJLabel: JLabel[][]
-sizeX: int
-icoFouB: ImageIcon
-pxCase: int
-icoPionN: ImageIcon
-sizeY: int
-plateau: Plateau
-casesAccessibles: ArrayList<Case>
-jeu: Jeu
-icoTourB: ImageIcon
-caseClic2: Case
-caseClic1: Case
+VueControleur(jeu):
-demanderPromotion(): void
-placerLesComposantsGraphiques(): void
-afficherFinDePartie(String, String, String): void
-chargerIcone(String): ImageIcon?
-sauvegarderPartie(): void
-chargerLesIcones(): void
-mettreAJourAffichage(): void
+update(Observable, Object): void
VueControleurConsole
-pause: boolean
-sizeY: int
-jeu: Jeu
-gameOver: boolean
-plateau: Plateau
-sizeX: int
-casesAccessibles: ArrayList<Case>
-modeIA: boolean
+VueControleurConsole(jeu, boolean):
-sauvegarderPartie(): void
+update(Observable, Object): void
-demanderCoup(): void
-obtenirCaseDepuisString(String): Case?
-getSymbolForPiece(Piece): String
-demanderPromotion(): void
-afficherEchiquier(): void
+jouer(): void
create
piece
dCA
tab
p
c
casesAccessibles
tourActuel
couleur
joueurB
joueurN
currentEvent
historiqueCoups