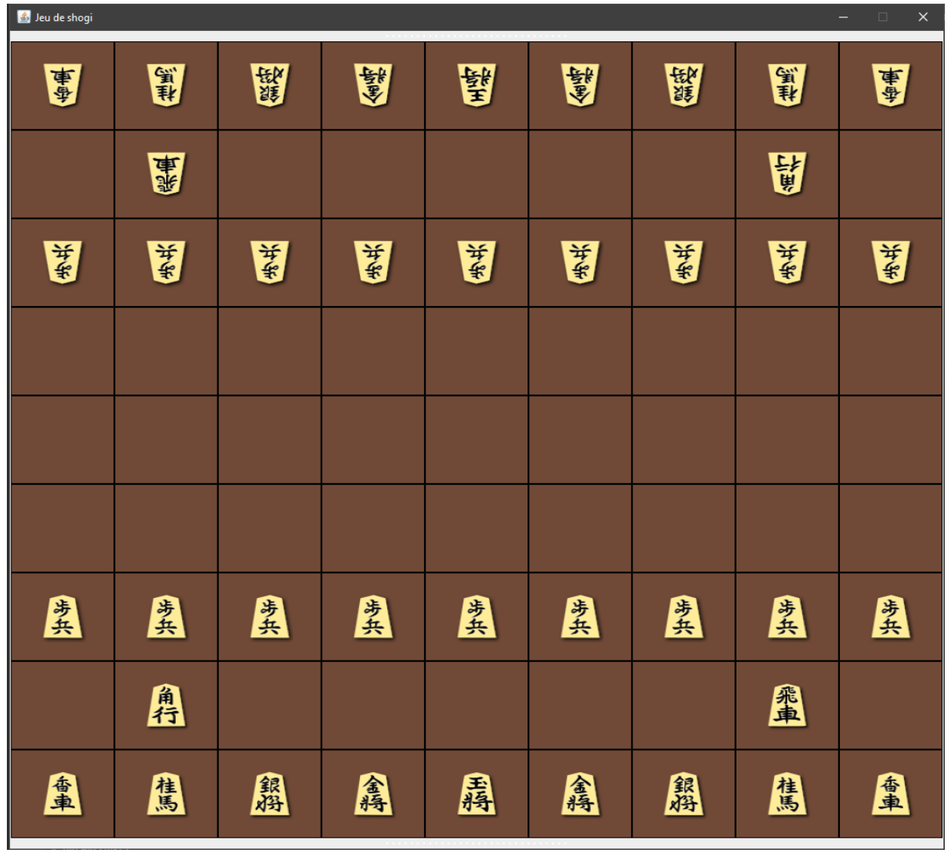

PROJET JAVA : SHOGI

AS/FC 2020-2021
Arthur GRÉGOIRE
Kalil KEFI
Jérôme KRAFT

- ÉTAT DE L'ART DU SHOGI

- Historique du jeu

- Le marché

Google Play

- Offres

- Application de la référence en ligne (site web conseillé par la fédération française de Shogi).

- Le plus populaire.

playok

- Un service lambda

- Le plateau

```java
public class Plateau {

    private Case[][] plateau = new Case[9][9];
    private Reserve[] reserveJoueur = {null, new Reserve(1), new Reserve(2)};

    /**
     * Definit les attributs de la classe "Plateau" :
     *
     * @param plateau plateau de jeu constitue comme un tableau a 2 dimensions de 9x9 objets de type "Case"
     *                (soit 100 cases : 10 cases (0-9) en abscisse, 10 cases en ordonnee)
     * @param reserveJoueur objet de type "Reserve" qui constitue la reserve de chaque joueur
     */

    // Constructeur
    public Plateau() {
        for(int i = 0; i < plateau.length; i++) {
            for(int j = 0; j < plateau[i].length; j++) {
                plateau[i][j] = new Case(i, j);
```

- Les cases

```java
public class Case {
    private int x;
    private int y;
    private Piece p;

    /**
     * Definit les attributs de la classe "Case" :
     *
     * @param x abscisse de l'objet
     * @param y ordonnee de l'objet
     * @param p objet de type "Piece" qui va etre affecte a cette case
     */

    // Constructeur
    public Case(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

- Les pièces

```java
public class Piece {

    private String nom = "";
    protected int joueur;
    ImageIcon i;
    ImageIcon icon;
    protected boolean promu = false;

    /**
     * Definit les attributs de la classe "Piece" :
     *
     * @param nom attribue une chaine de carateres comme nom de la piece
     * @param joueur determine si la piece appartient au joueur 1 ou 2
     * @param i associe une icone à la piece (attribut temporaire permettant la conversion vers Image)
     * @param icon associe une icone à la piece
     * @param promu attribut booleen indiquant si la piece est promue ou non
     */

    // Constructeur
    public Piece(int j) {
        this.joueur = j;
    }
```

- Exemple du Pion

```java
public class Pion extends Piece {

    // Constructeur
    public Pion(int joueur) {
        super(joueur);
        setNom("Pion");
        setIcon();
    }
```

Exemple du Roi

```java
public boolean peutSeDeplacer(Case posDepart, Case posArrivee, Plateau p) {

    // Deplacement possible si les conditions ci-dessous sont verifiees :

    // Deplacement interdit si la case d'arrivee est deja occupee par une piece appartenant au joueur actif
    if(posArrivee.getP() != null) {
        if(posDepart.getP().getJoueur() == posArrivee.getP().getJoueur()) {
            return false;

            /* La 1ere condition verifie s'il y a deja une piece presente sur la case ou le joueur veut deplacer
             * sa piece ("!= null").
             * La 2nde condition verifie si la piece sur cette case appartient au joueur actif (".getP().getJoueur()"),
             * ce qui interdit son deplacement (ne peut pas manger ses propres pieces) */
        }
    }

    // Limite le deplacement a 1 case maximum en ordonnee et en abscisse
    if((Math.abs(posDepart.getY() - posArrivee.getY()) <= 1) && (Math.abs(posDepart.getX() - posArrivee.getX()) <= 1)) {
        return true;
    }

    // Deplacement interdit si les conditions ci-dessus ne sont pas verifiees
    return false;
}
```

Sur le plateau

```java
// Effectue le deplacement d'une piece
public void deplacementPiece(Case posDepart, Case posArrivee, int tour) throws Exception {
    Piece pieceDepart = posDepart.getP();

    // Deplacement d'une piece presente sur le plateau de jeu :

    // Verifie que la piece a deplacer appartient bien au joueur actif et qu'un deplacement a bien lieu
    if(pieceDepart.getJoueur() == tour && (posDepart.getY() != posArrivee.getY() || posDepart.getX() != posArrivee.getX())) {
        if(pieceDepart.peutSeDeplacer(posDepart, posArrivee, this)) {
            try {

                // Verifie si le joueur 1 deplace une de ses pieces dans sa zone de promotion (cases 6, 7, 8)
                if(posArrivee.getX() >= 6 && posDepart.getP().getJoueur() == 1) {
                    posDepart.getP().estPromue();
                }

                // Verifie si le joueur 2 deplace une de ses pieces dans sa zone de promotion (cases 0, 1, 2)
                if(posArrivee.getX() <= 2 && posDepart.getP().getJoueur() == 2) {
                    posDepart.getP().estPromue();
                }
            } catch(Exception e) {}

            // Verifie, si la case d'arrivee n'est pas vide, que la piece sur cette case n'appartient pas au joueur actif
            // et l'ajoute ensuite a sa reserve
            if(posArrivee.getP() != null) {
                if(posDepart.getP().getJoueur() != posArrivee.getP().getJoueur()) {
                    reserveJoueur[posDepart.getP().getJoueur()].ajouterPiece(posArrivee.getP());
                }

                // Verifie si la case sur laquelle va se deplacer le joueur contient le roi adverse
                if(posArrivee.getP().getNom().equals("Roi")) {
                    JOptionPane.showMessageDialog(null, "Le joueur " + posDepart.getP().getJoueur() + " gagne !", "Fin de partie",
                    JOptionPane.INFORMATION_MESSAGE);
                    System.exit(0);
                }
            }

        // Reinitialise les cases de depart et d'arrivee, avant de placer la piece deplacee sur la case d'arrivee
        posDepart.setP(null);
        posArrivee.setP(null);
        posArrivee.setP(pieceDepart);
        }
```

La réserve

```java
public class Reserve {

    private ArrayList<Piece> pieces = new ArrayList<Piece>();
    private int joueur;

    /**
     * Definit les attributs de la classe "Reserve" :
     *
     * @param pieces cree une ArrayList pour stocker les pieces presentes dans la reserve de chaque joueur
     * @param joueur determine si la reserve appartient au joueur 1 ou 2
     */

    // Constructeur
    public Reserve (int j) {
        this.joueur = j;
        for (int i = 0; i < 38; i++) {
            pieces.add(null);

            /*  Initialise une ArrayList nommee "pieces" pour chaque joueur
             * avec 38 cases vides (chaque joueur peut avoir au maximum 38
             * pieces dans sa reserve) */
        }
    }
```

Le parachutage

```java
// Parachutage d'une piece depuis la reserve du joueur actif
else if(posDepart.getY() == 100 && posDepart.getX() == 100) {

    // Verifie que la case sur laquelle va etre parachutee la piece est vide
    if(posArrivee.getP() == null) {

        // Verifie que, si la piece a parachuter est un pion, il n'y ait pas d'autre pion du meme joueur dans la colonne de parachutage
        if(posDepart.getP().getNom().equals("Pion")) {
            for(int i = 0; i < 9; i++) {
                if(plateau[i][posArrivee.getY()].getP() != null) {
                    if(plateau[i][posArrivee.getY()].getP().getNom().equals("Pion")
                            && plateau[i][posArrivee.getY()].getP().getJoueur() == posDepart.getP().getJoueur()) {
                        Piece p = posDepart.getP();
                        int j = p.getJoueur();
                        if(pieceDepart.getJoueur() == 1) {
                            p.setJoueur(2);
                        }
                        else {
                            p.setJoueur(1);
                        }
                        reserveJoueur[j].ajouterPiece(p);
                        throw new Exception();

                        /* Verifie la presence d'une autre piece "Pion" dans la colonne de la case d'arrivee :
                         *  - "if(posDepart.getP().getNom().equals("Pion"))" : verifie si la piece a parachuter est un "Pion", auquel cas =>
                         *  - "for(int i = 0; i < 9; i++)" : parcourt la colonne
                         *  - "if(plateau[i][posArrivee.getY()].getP() != null)" : et verifie si elle contient d'autres pieces, auquel cas =>
                         *  - "if(plateau[i][posArrivee.getY()] ... == posDepart.getP().getJoueur())" : si un pion du joueur actif est
                         *          present dans la colonne de la case d'arrivee
                         *  - renvoie la piece a deplacer dans la reserve du joueur actif ET lui change son appartenance (pour eviter que les
                         *          pieces du joueur adverse capturees par le joueur actif ne soient renvoyees dans la reserve du joueur
                         *          adverse)
                         *  - lance une exception pour empecher le deplacement */
                    }
                }
            }
        }
        pieceDepart.retrograde();
        posDepart.setP(null);
        posArrivee.setP(null);
        posArrivee.setP(pieceDepart);
```

- La promotion / La rétrogradation
- Sur le plateau

```java
// Methode pour promouvoir une piece
public void estPromue(){
    promu = true;
    setIcon(this.getNom());

    /* Passe le booleen a "true" (piece promue maintenant) et lui affecte egalement une
     * nouvelle icone */
}

// Methode pour retrograder une piece
public void retrograde(){
    promu = false;
    setIcon();

    /* Passe le booleen a "false" (piece retrogradee maintenant) et lui reaffecte son
     * icone de base */
}
```

```java
try {

    // Verifie si le joueur 1 deplace une de ses pieces dans sa zone de promotion (cases 6, 7, 8)
    if(posArrivee.getX() >= 6 && posDepart.getP().getJoueur() == 1) {
        posDepart.getP().estPromue();
    }

    // Verifie si le joueur 2 deplace une de ses pieces dans sa zone de promotion (cases 0, 1, 2)
    if(posArrivee.getX() <= 2 && posDepart.getP().getJoueur() == 2) {
        posDepart.getP().estPromue();
    }
} catch(Exception e) {}
```

- L'interface

```
public class Interface extends JFrame {

    private static final long serialVersionUID = 1L;
    static JFrame fenetre = new JFrame();
    static JPanel UIplateau = new JPanel(new GridLayout(9,9));
    static JPanel[] UIreserve = {null, new JPanel(), new JPanel()};
    static JButton[][] cases = new JButton[9][9];
    public static JButton[] boutonReserve[] = {null, new JButton[38], new JButton[38]};
    static Plateau p = new Plateau();
    static Case caseSelectionnee = null;
    public static int tour = 1;
```

- Les boutons du plateau

```
// Ajoute un bouton dans chaque ligne du plateau
for(int x = 0; x < 9; x++) {

    // Ajoute un bouton pour chaque colonne de cette ligne
    for(int y = 0; y < 9; y++) {
        cases[x][y] = new JButton();
        cases[x][y].setOpaque(true);
        cases[x][y].setSize(108, 108);
        cases[x][y].setBorder(new LineBorder(Color.black));
        cases[x][y].setBackground(Color.decode("#704a37"));
        cases[x][y].addActionListener(new ActionListener() {
```

- Les boutons de la réserve

```
// Ajoute des JButtons pour les pieces presentes dans les reserves
for(int joueur = 1; joueur <= 2; joueur++) {
    for(int i = 0; i < 38; i++) {
        boutonReserve[joueur][i] = new JButton("");
        boutonReserve[joueur][i].setOpaque(true);
        boutonReserve[joueur][i].setSize(108, 108);
        boutonReserve[joueur][i].setBorder(new LineBorder(Color.white));
        boutonReserve[joueur][i].setBackground(Color.white);
        final int finalJoueur = joueur;

        /* Definit les caracteristiques de chaque JButton composant la reserve :
         *      - 38 JButtons a instancier, car chaque joueur peut avoir (au maximum) toutes les pieces dans sa reserve,
         *      - sa visibilite,
         *      - sa taille,
         *      - ses couleurs de delimitation et d'arriere-plan */

        // Parachutage d'une piece sur le plateau depuis la reserve du joueur actif
        boutonReserve[joueur][i].addActionListener(new ActionListener() {
```

- Le placement des fenêtres

```
// Constructeur
public Interface() {

    // Creation de la fenetre du jeu
    fenetre.setLayout(new BorderLayout());
    fenetre.setSize(1080, 972);
    fenetre.setTitle("Jeu de shogi");
    fenetre.add(UIplateau, BorderLayout.CENTER);
    fenetre.setResizable(false);
    fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    /* Cree une fenetre (objet de type JFrame) a qui l'on attribue une taille (".setSize") et un titre (".setTitre").
     * Ajoute egalement le plateau, que l'on centre au milieu du Layout. La taille de la fenetre contenant le jeu est fixe
     * et non modifiable par les joueurs (".setResizable"). La derniere ligne definit le comportement adoptee par l'interface
     * graphique lorsque l'on clique sur la croix de fermeture */

    // Creation des reserves pour les 2 joueurs
    fenetre.add(UIreserve[1], BorderLayout.PAGE_START);
    fenetre.add(UIreserve[2], BorderLayout.PAGE_END);

    /* Ajoute 2 nouveaux Layouts dans la fenetre de jeu, 1 pour le joueur 1 en haut de la fenetre et 1 pour le joueur 2
     * en bas de la fenetre */

    // Taille des differents elements de la fenetre de jeu
    UIplateau.setSize(972, 972);
    UIreserve[1].setSize(1080, 54);
    UIreserve[2].setSize(1080, 54);
```

- Les actions listeners

```
public void actionPerformed(ActionEvent e) {
    for(int n = 0; n < 38; n++) {
        if(e.getSource() == boutonReserve[finalJoueur][n]) {
            if(p.getReserve(finalJoueur).getPiece(n).getJoueur() != tour) {
                Case c = new Case(100, 100);
                Piece pi = p.getReserve(finalJoueur).getPiece(n);
                pi.setJoueur(p.getReserve(finalJoueur).getJoueur());
                c.setP(pi);
                caseSelectionnee = c;
                boutonReserve[finalJoueur][n].setVisible(false);
                p.getReserve(finalJoueur).setPiece(n, null);
```

```
public void actionPerformed(ActionEvent e) {
    for(int x = 0; x < 9; x++) {
        for(int y = 0; y < 9; y++) {
            if(e.getSource() == cases[x][y]) {
                int k = 0;

                // Verifie si la case contient une piece
                try {
                    k = p.getCase(x, y).getP().getJoueur();
                } catch(Exception e1) {}   // e1 car ActionEvent = e

                // Verifie si la piece a deplacer appartient bien au joueur actif et si la case temporaire est vide
                if(caseSelectionnee == null && k == tour) {
                    if(p.getCase(x, y).getP() != null) {
                        caseSelectionnee = p.getCase(x, y);

                        // Indique les deplacements possibles pour la piece selectionnee avec un "." et un changement
                        // de couleur de la case
                        for(int a = 0 ; a < 9; a++) {
                            for(int o = 0; o < 9; o++) {
                                if(p.getCase(x, y).getP().peutSeDeplacer(p.getCase(x, y), p.getCase(a, o), p)) {
                                    cases[a][o].setText(cases[a][o].getText() + ".");
                                    cases[a][o].setBackground(Color.decode("#633a25"));
                                }
                            }
                        }
                    }
                } else {
                    try {
                        // Effectue le deplacement et incremente le compteur de tours
                        p.deplacementPiece(caseSelectionnee, p.getCase(x, y), tour);
                        cases[x][y].setForeground(Color.black);
                        caseSelectionnee = null;
                        if(tour == 1) {
                            tour = 2;
                        } else {
                            tour = 1;
                        }
                        miseAJourPlateau();
                    } catch(Exception e2) {

                        // Deplacement impossible
                        e2.printStackTrace(System.out);
                        System.out.println("Mouvement invalide");
                        caseSelectionnee = null;
                        miseAJourPlateau();
```

- Actualisation de l'interface

```
public static void miseAJourPlateau() {

    // Actualisation du plateau apres chaque deplacement de piece
    for (int x = 0; x < 9; x++) {
        for (int y = 0; y < 9; y++) {

            // Actualisation de la case si une piece est presente dessus
            if (p.getCase(x, y).getP() != null) {
                cases[x][y].setIcon(p.getCase(x, y).getP().getIcon());
                System.out.println(p.getCase(x, y).getP().getIcon());
                cases[x][y].setText("");
                cases[x][y].setBackground(Color.decode("#704a37"));

            } else {
                // Actualisation de la case s'il n'y a pas de piece presente dessus
                cases[x][y].setText("");
                cases[x][y].setIcon(null);
                cases[x][y].setForeground(Color.black);
                cases[x][y].setBackground(Color.decode("#704a37"));
            }
        }
    }
```

```
// Actualisation des reserves des joueurs
for (int j = 1; j <= 2; j++) {
    for (int i = 0; i < 8; i++) {

        // Actualisation de la case si une piece est presente dessus
        if (p.getReserve(j).getPiece(i) != null) {
            boutonReserve[j][i].setText(p.getReserve(j).getPiece(i).getNom());
            boutonReserve[j][i].setVisible(true);

        } else {
            // Actualisation de la case s'il n'y a pas de piece presente dessus
            boutonReserve[j][i].setVisible(false);
        }
```

• État de l'art du Shogi

- Historique du jeu
  - Apparition vers la fin du VI^{ème} siècle dans l'est de l'Asie.
  - Le Shogi prend en popularité au Japon à partir de 794.
  - Il s'est rapidement exporté vers le reste du monde au VII^{ème} siècle.
- Le marché
  - Compte une cinquantaine d'appli sur le PlayStore.
  - Jeux web, seul ou en ligne.
- Offres

playok

- Peu plaisant graphiquement.
- Interface peu intuitive.
- Peu de joueurs.
- Ne propose aucun mode off-line.
- Pas de mode alternatifs au match traditionnel.

SHOGI

Shogi Free - Japanese Chess

Cross Field Inc. Jeux de société ★★★★☆ 42 793

PEGI 3

Contient des annonces · Achats via l'application proposés

Vous ne disposez d'aucun appareil

Ajouter à la liste de souhaits

Installer

- Le plus populaire.
- Beginner friendly.
- Fonctionnalités on/off-line.
- Ranking en ligne.
- Problèmes à résoudre (Tsume Shogi).
- Sauvegarde des logs pour relecture.
- Plusieurs niveaux de difficulté de l'IA.

81Dojo (World Online Shogi)

81Dojo Jeux de société ★★★★☆ 1 307

PEGI 3

Vous ne disposez d'aucun appareil

Ajouter à la liste de souhaits

Installer

- Application de la référence en ligne (site web conseillé par la fédération française de Shogi).
- Moins beginner friendly.
- Possibilité d'analyser en détail ses parties.
- Variantes graphiques.
- Nombreux handicaps disponible.