RAPPORT

ENGINEERING SCHOOL
Creating the future together

JEU DU CHATEAUX

ROMAIN CUCHE-MAKSIMILIEN DEUTSCH

Sommaire

I. Introduction

Dans le cadre de notre module de Conception et Programmation Objet, nous avons entrepris le développement du *"Jeu de Château"*, un jeu de stratégie codé en Java mettant en scène deux joueurs s'affrontent sur un plateau de 7x13 cases. L'objectif principal ? Capturer le château adverse ou éliminer toutes les pièces de son adversaire pour remporter la victoire.

Ce projet ambitieux nous a permis de plonger concrètement dans l'univers de la programmation orientée objet et des interfaces graphiques Java. Inspiré par les jeux de stratégie traditionnels comme les échecs, mais avec ses propres règles adaptées, le "Jeu de Château" représente une synthèse des compétences acquises tout au long de notre formation.

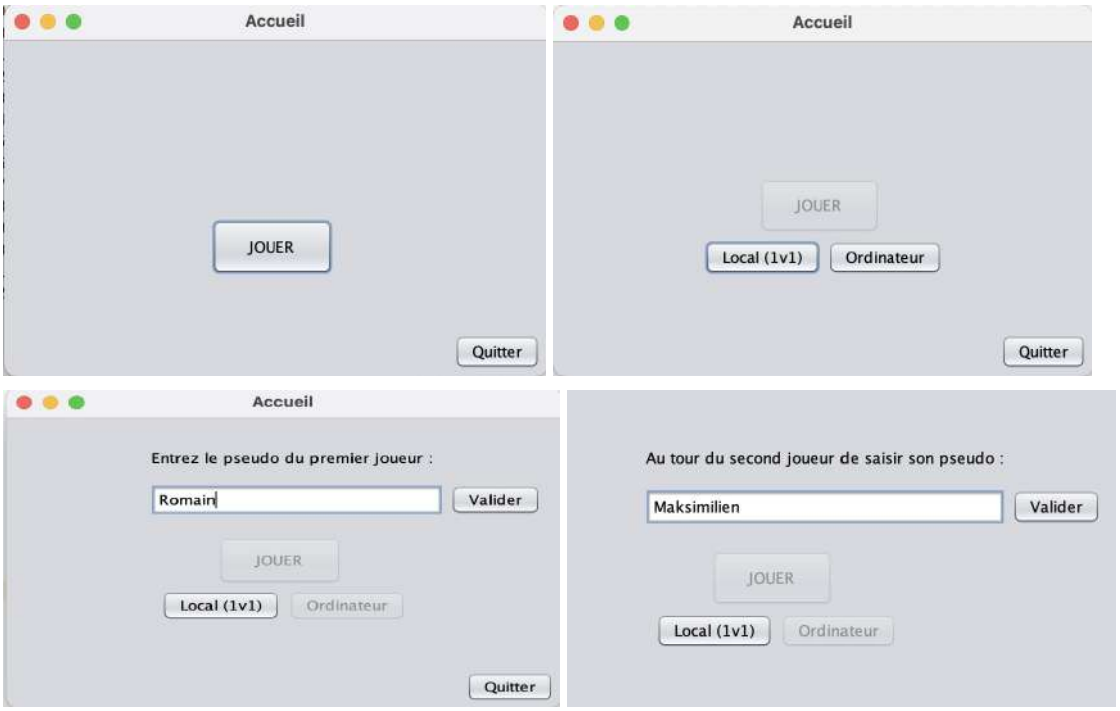
Les premiers mois ont été consacrés à la conception de la logique métier : implémentation des déplacements spécifiques des pions et cavaliers, gestion des captures par saut, et vérification des conditions de victoire. Puis nous avons relevé le défi de l'interface graphique, découvrant les composants Swing pour créer une expérience utilisateur interactive avec plateau visuel, sélection de pièces et contrôles de jeu intuitifs.

Ce rapport détaillera les aspects techniques de notre réalisation, les fonctionnalités clés implémentées (dont le système de sauvegarde/chargement), les défis surmontés et les pistes d'amélioration. Il témoigne de notre progression dans la maîtrise de Java, depuis les concepts objet fondamentaux jusqu'à la création d'une application graphique complète et fonctionnelle.

II. Partie Graphique

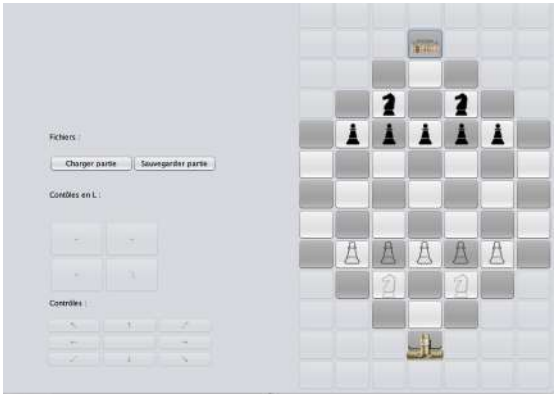
Le projet "Jeu de Château" comprend deux interfaces principales qui jouent des rôles complémentaires dans l'expérience de jeu : la classe FAccueil pour la configuration initiale et la classe PlateauJeu pour le déroulement de la partie.

A) FAcceuil



La classe FAccueil sert d'interface d'accueil et de configuration du jeu. Elle permet aux joueurs de démarrer une nouvelle partie en proposant deux modes de jeu : un affrontement local entre deux joueurs humains ou une partie contre l'ordinateur (bien que cette dernière option ne soit pas encore totalement fonctionnelle). Lorsqu'un joueur clique sur le bouton "JOUER", l'interface lui demande de saisir les pseudos des joueurs selon le mode choisi. Un tirage aléatoire détermine ensuite quelle couleur (blanc ou noir) est attribuée à chaque joueur et qui commencera la partie. Une fois ces paramètres établis, FAccueil transmet ces informations et lance l'interface principale du jeu via la classe PlateauJeu.

B) PlateauJeu

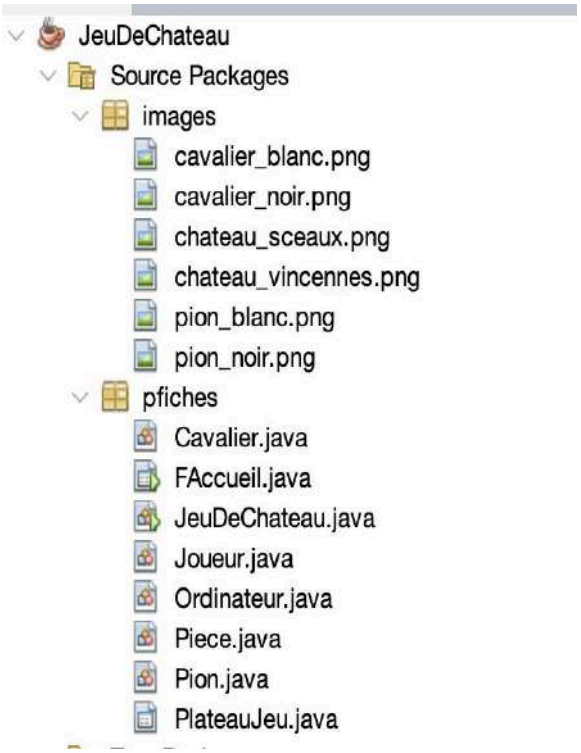


La classe PlateauJeu constitue le cœur interactif du jeu. Elle affiche un plateau de 7x13 cases sous forme de grille cliquable, où chaque case est représentée par un bouton. Les différentes pièces (pions et cavaliers) ainsi que les châteaux sont visibles grâce à des icônes spécifiques, avec des couleurs distinctes pour les pièces blanches et noires. Les joueurs interagissent avec le plateau en sélectionnant d'abord une de leurs pièces (qui se met alors en surbrillance en jaune), puis en choisissant un mouvement parmi les boutons directionnels disponibles. Les cavaliers bénéficient de boutons supplémentaires pour leurs déplacements en L caractéristiques. Le système vérifie en permanence la validité des mouvements, gère les captures de pièces adverses (qui impliquent un saut par-dessus la pièce capturée), et détecte les conditions de victoire (soit par capture du château adverse, soit par élimination de toutes les pièces de l'adversaire). Entre les tours, l'interface alterne automatiquement entre les joueurs et propose des fonctionnalités supplémentaires comme la sauvegarde et le chargement de parties.

Ces deux interfaces travaillent en étroite collaboration : FAccueil initialise et configure les paramètres de base des joueurs, puis passe le relais à PlateauJeu qui gère toute la logique et l'interactivité du jeu proprement dit. Alors que FAccueil se concentre sur la phase de préparation (choix des pseudos, des couleurs et du mode de jeu), PlateauJeu prend en charge tout le déroulement de la partie, depuis les déplacements des pièces jusqu'à la détection des victoires, en passant par les options de sauvegarde.

III. Structure Code

A.Code



Récapitulatif des méthodes:

Classe Piece (Classe mère)

public void deplacer(Piece[][] plateau, String deplacement)

- **Description** : Gère le déplacement générique d'une pièce (pions).
- **Paramètres** :
    1. plateau : Matrice représentant l'état actuel du jeu.
    2. deplacement : Direction ("up", "down", "left", "right", etc.).
- **Fonctionnement** :
    1. Calcule la nouvelle position.
    2. Vérifie les limites du plateau.
    3. Gère la capture si une pièce adverse est présente (saut par-dessus).
    4. Met à jour la position de la pièce.
    5.

public Piece[][] plateau()

- **Description** : Initialise un plateau vide (7x13).
- **Retourne** : Une matrice vide de Piece.

Classe Cavalier (Hérite de Piece)

public void deplacer(Piece[][] plateau, String deplacement)

- **Description** : Surcharge la méthode pour les déplacements en L.

- **Spécificités** :
    - Gère 4 types de mouvements en L ("lul", "lur", "ldl", "ldr").
    - Implémente la capture avec saut similaire aux pions.

Classe Joueur

public boolean verifGagne(Piece[][] plateau, String couleur)

- **Description** : Vérifie les conditions de victoire.
- **Conditions** :
    1. Capture du château adverse (position (3,1) pour blanc, (3,11) pour noir).
    2. Élimination de toutes les pièces adverses.
- **Retourne** : true si le joueur a gagné.

Classe JeuDeChateau

public void sauvegarder(String nomFichier)

- **Description** : Sauvegarde l'état du jeu dans un fichier.
- **Format** : *Joueur1:pseudo,couleur*
  ,Joueur2:pseudo,couleur
  Piece:type,couleur,x,y

public void charger(String nomFichier)

- **Description** : Restaure une partie depuis un fichier.
- **Reconstruit** :
    - Les joueurs.
    - Les pièces avec leurs positions.

Classe PlateauJeu (Interface Graphique)

public void initialiserPlateau(Joueur joueur1, Joueur joueur2)

- **Description** : Initialise le plateau avec les pièces en position de départ.
- **Actions** :
    - Place les pions et les cavaliers.
    - Affiche les châteaux (icônes).

public void actualiserPlateau()

- **Description** : Met à jour l'affichage graphique.
- **Fonctionnement** :
    - Parcourt la matrice du jeu.
    - Assigne les icônes correspondantes aux boutons.

public void deplacerSiPossible()

- **Description** : Tente d'exécuter un déplacement après vérification.
- **Vérifications** :
    1. Pièce sélectionnée valide.
    2. Tour du joueur actuel.

3. Mouvement autorisé.

Classe PlateauJeu

public void bFinDeTourActionPerformed(ActionEvent evt)

Description : Gère le tour de l'IA lorsqu'on clique sur "Fin de tour".
Paramètres :

- evt : Événement de clic sur le bouton.

Fonctionnement :

1. Calcul des distances :
    - Pour chaque pièce de l'IA, calcule la distance euclidienne avec les pièces adverses.
    - Stocke les distances dans une matrice distance[7][13].
2. Sélection de la pièce :
    - Utilise getXMin() et getYMin() pour identifier la pièce IA la plus proche d'un adversaire.
    - Vérifie que les coordonnées sont valides (xMin, yMin).
3. Détermination de la direction :
    - Compare les écarts horizontaux/verticaux (deltaX, deltaY).
    - Choisit la direction ("up", "down", "left", "right") pour se rapprocher.
4. Exécution du déplacement :
    - Appelle pieceOrdinateur.deplacer() avec la direction calculée.
    - Passe le tour au joueur humain via changJoueurActuel().

private void getXMin(double[][] distances)

Description : Trouve la coordonnée X de la pièce IA la plus proche d'un adversaire.
Paramètres :

- distances : Liste 2D des distances euclidiennes.

Retourne :

- xMin : Coordonnée X de la pièce à déplacer (ou -1 si aucune).

private void getYMin(double[][] distances)

Description : Trouve la coordonnée Y de la pièce IA la plus proche.
Mêmes paramètres/sortie que getXMin, mais pour l'axe Y.

1.

B. Changement dans les méthodes

Premièrement, nous avons supprimé la classe Ordinateur, pour quelles raisons ?
On va traiter les déplacements de l'ordinateur directement dans la classe PlateauJeu, grâce à la méthode bFinDeTourActionPerformed.
L'ordinateur est une instance de la classe joueur.

Le changement majeur est la suppression des Scanner, car dans la partie graphique l'output ne nous intéresse plus, a part pour le développeur de déboguer son programme.
On a modifié la méthode déplacer dans Pièce et donc dans Cavalier :

```
public void deplacer(Piece[][] plateau, String deplacement) {
    mouvementValide = "true";  // comme on peut pas appliquer de getter à un boolean.
    captureValide = "true";
    int ancienX = x;
    int ancienY = y;
    int newX = x;
    int newY = y;
    System.out.println("deplacement en cours");
        switch (deplacement) {
            case "down":    newY = y + 1; mouvementValide = "true"; break;
            case "up":      newY = y - 1; mouvementValide = "true"; break;
            case "left":    newX = x - 1; mouvementValide = "true"; break;
            case "right":   newX = x + 1; mouvementValide = "true"; break;
            case "diagul":  newX = x - 1; newY = y - 1; mouvementValide = "true"; break;
            case "diagur":  newX = x + 1; newY = y - 1; mouvementValide = "true"; break;
            case "diagdl":  newX = x - 1; newY = y + 1; mouvementValide = "true"; break;
            case "diagdr":  newX = x + 1; newY = y + 1; mouvementValide = "true"; break;
            default:
                System.out.println("Mouvement invalide.");
                return; // On sort si le mouvement est invalide
        }
        if (newX < 0 || newX >= plateau.length || newY < 0 || newY >= plateau[0].length) {
            System.out.println("Mouvement hors des limites du plateau !");
            mouvementValide = "false";
            return;  //pour la partie graphique c'est bien comme ça on reclique juste pour rlancer la méthode;
            }
        if (plateau[newX][newY] != null) {
            if (plateau[newX][newY].getCouleur().equals(this.getCouleur())) {
                System.out.println("Vous ne pouvez pas capturer vos propres pieces !");
            } else {
                // Capture d'une pièce adverse
```

```
                // Capture d'une pièce adverse
                System.out.println("Capture réussie !");
                plateau[ancienX][ancienY] = null;
                plateau[newX][newY] = null;

                int sautX = newX + (newX - ancienX);
                int sautY = newY + (newY - ancienY);

                if (sautX >= 0 && sautX < plateau.length && sautY >= 0 && sautY < plateau[0].length) {
                    plateau[sautX][sautY] = this;
                    x = sautX;
                    y = sautY;
                } else {
                    System.out.println("Saut hors des limites du plateau !");
                    plateau[ancienX][ancienY] = null;  //modifié le 06/05 car on ne doit pas faire le saut
                    plateau[newX][newY] = this;     // maintenant on supprime la pièce et on la remplace par la nc
                    x = newX;
                    y = newY;
                }
            }
        } else {
            // Déplacement simple
            plateau[ancienX][ancienY] = null;
            plateau[newX][newY] = this;
            x = newX;
            y = newY;
            System.out.println("Nouvelle position de " + getIdPiece() + " : (" + x + ", " + y + ")");

        }
}
```

```java
public Cavalier(String idPiece, String couleur, int x, int y){    // creation du constructeur du cavalier.
        super(idPiece, couleur, x, y);   //on appelle le constructeur de la classe Piece.
}

//Creation des méthodes "getter" pour pouvoir réutiliser le nom et la couleur des pièces :

public String getIdPiece(){
    return idPiece;
}

public String getCouleur(){
    return couleur;
}

public int getX(){
    return x;
}

public int getY(){
    return y;
}

public void deplacer(Piece[][] plateau, String deplacement) {
    int ancienX = x;
    int ancienY = y;
    int sautX = 0;
    int sautY = 0;

        System.out.println("Quel déplacement souhaitez-vous faire ? (lul/lur/ldl/ldr)");
        int newX = x; // Nouvelle position X
        int newY = y; // Nouvelle position Y
        switch (deplacement) {
            case "lul":
                newX = x - 1;
                newY = y - 2;
                sautX = newX;
                sautY = newY - 1;
                break;
            case "lur":
                newX = x + 1;
                newY = y - 2;
                sautX = newX;
                sautY = newY - 1;
                break;
            case "ldl":
                newX = x - 1;
                newY = y + 2;
                sautX = newX;
                sautY = newY + 1;
                break;
            case "ldr":
                newX = x + 1;
                newY = y + 2;
                sautX = newX;
                sautY = newY + 1;
                break;
            default:
                super.deplacer(plateau, deplacement);
                return; // Sort de la méthode si le mouvement est invalide
        }
    if (newX < 0 || newX >= plateau.length || newY < 0 || newY >= plateau[0].length) {
```

```
            System.out.println("Mouvement hors des limites du plateau !");
            mouvementValide = "false";
            return;  //pour la partie graphique c'est bien comme ça on reclique juste pour rlancer la méthode;
            }
        if (plateau[newX][newY] != null) {
            if (plateau[newX][newY].getCouleur().equals(this.getCouleur())) {
                System.out.println("Vous ne pouvez pas capturer vos propres pieces !");
            } else {
                // Capture d'une pièce adverse
                System.out.println("Capture réussie !");
                plateau[ancienX][ancienY] = null;
                plateau[newX][newY] = null;


                if (sautX >= 0 && sautX < plateau.length && sautY >= 0 && sautY < plateau[0].length) {
                    plateau[sautX][sautY] = this;
                    x = sautX;
                    y = sautY;
                } else {
                    System.out.println("Saut hors des limites du plateau !");
                    plateau[ancienX][ancienY] = null;  //modifié le 06/05 car on ne doit pas faire le saut
                    plateau[newX][newY] = this;     // maintenant on supprime la pièce et on la remplace par la r
                    x = newX;
                    y = newY;
                }
            }
        } else {
            // Déplacement simple
            plateau[ancienX][ancienY] = null;
            plateau[newX][newY] = this;
```


```
            System.out.println("Mouvement hors des limites du plateau !");
            mouvementValide = "false";
            return;  //pour la partie graphique c'est bien comme ça on reclique juste pour rlancer la méthode;
            }
        if (plateau[newX][newY] != null) {
            if (plateau[newX][newY].getCouleur().equals(this.getCouleur())) {
                System.out.println("Vous ne pouvez pas capturer vos propres pieces !");
            } else {
                // Capture d'une pièce adverse
                System.out.println("Capture réussie !");
                plateau[ancienX][ancienY] = null;
                plateau[newX][newY] = null;


                if (sautX >= 0 && sautX < plateau.length && sautY >= 0 && sautY < plateau[0].length) {
                    plateau[sautX][sautY] = this;
                    x = sautX;
                    y = sautY;
                } else {
                    System.out.println("Saut hors des limites du plateau !");
                    plateau[ancienX][ancienY] = null;  //modifié le 06/05 car on ne doit pas faire le saut
                    plateau[newX][newY] = this;     // maintenant on supprime la pièce et on la remplace par la r
                    x = newX;
                    y = newY;
                }
            }
        } else {
            // Déplacement simple
            plateau[ancienX][ancienY] = null;
            plateau[newX][newY] = this;
```

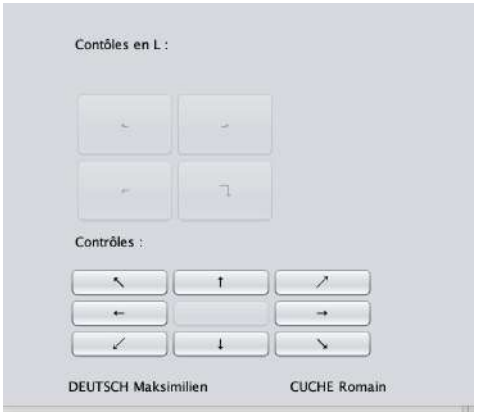
IV. Jeu D’essaie

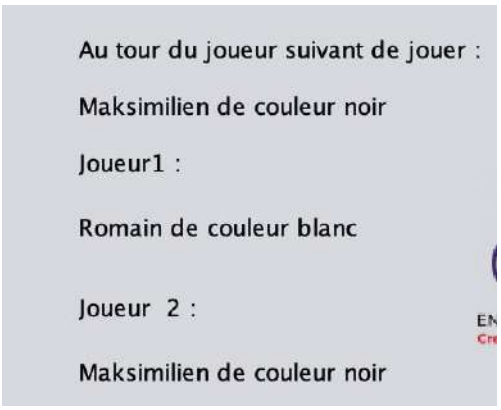
A. En 1vs1

Lorsqu’on lance le jeu on apparaît sur cette page, en cliquant sur “Jouer” on peut choisir de jouer contre l’ordinateur ou contre un adversaire en 1 contre 1.

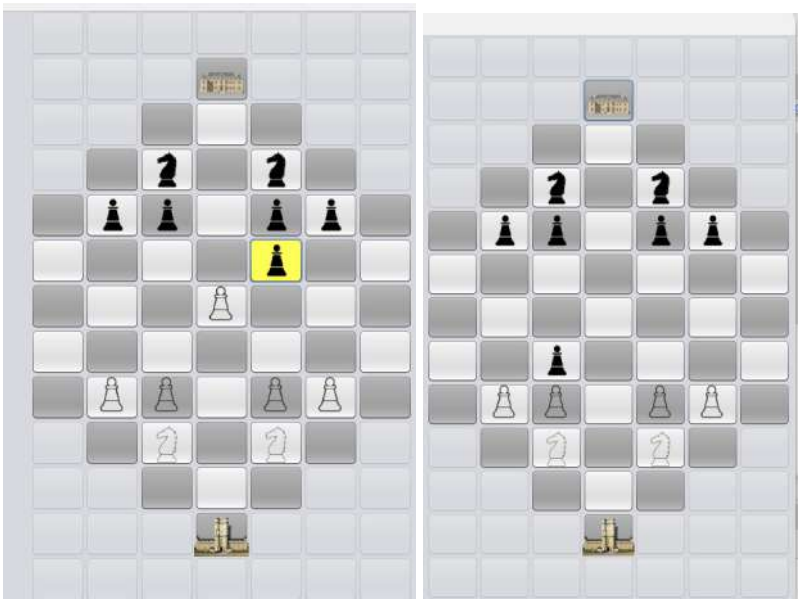
Une fois avoir cliquer sur 1vs1 on doit entrer nos pseudo un part un afin de choisir de manière aléatoire t grâce à la méthode tirage dans FAcceuil qui va commencer

La partie commence dans cet exemple par les blancs. Le joueur va pouvoir deplacer ses pions en cliquant au préalable dessus (il apparait en jaune) puis en appuyant sur les boutons de commande ci dessous. Les commandes en L sont utilisées pour les cavaliers

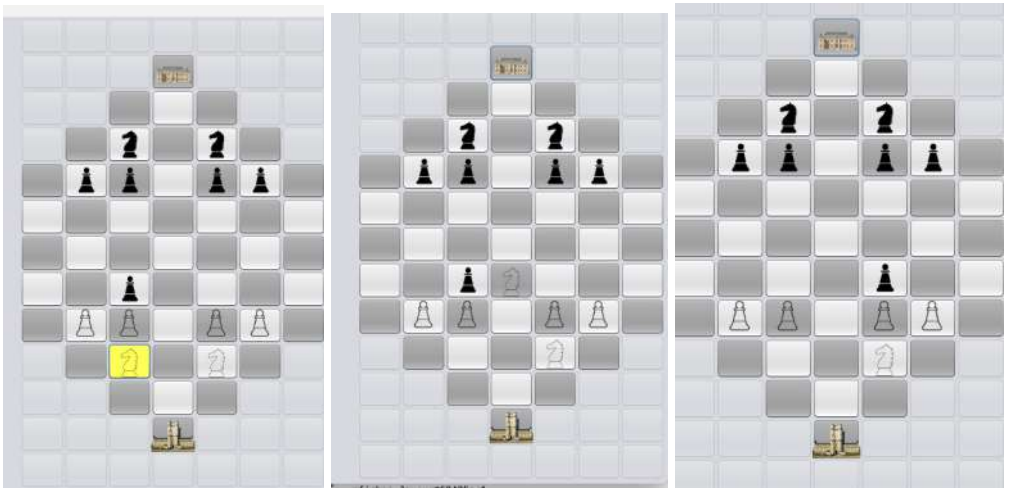




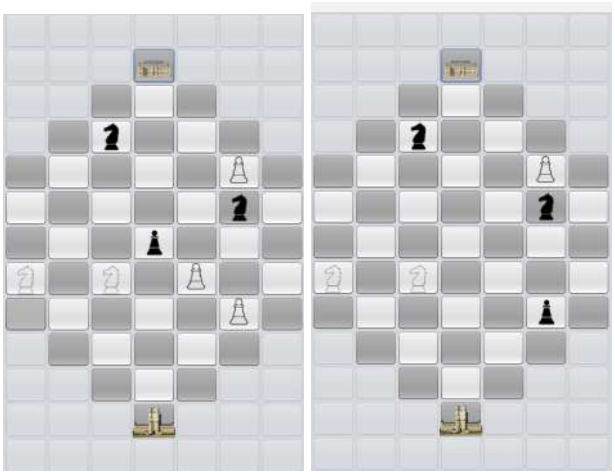
Un rappel est indiqué en haut de page pour dire quelle couleurs est attribués aux joueurs.



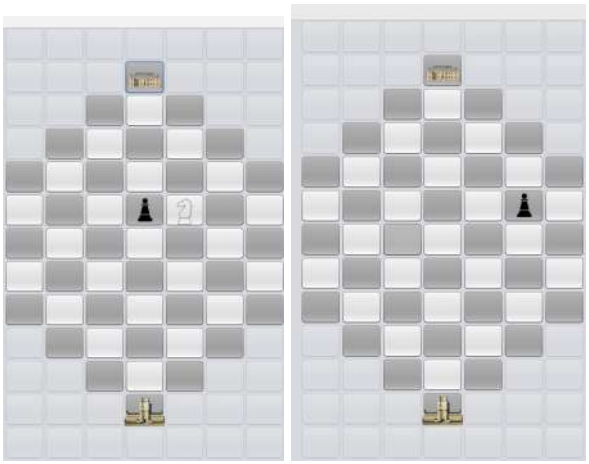
Le pion peut manger un joueurs sur le côté ou de face



Comme ci dessus on peut voir que le cavalier peut se déplacer en L et que le pion peut manger sur les côtés également



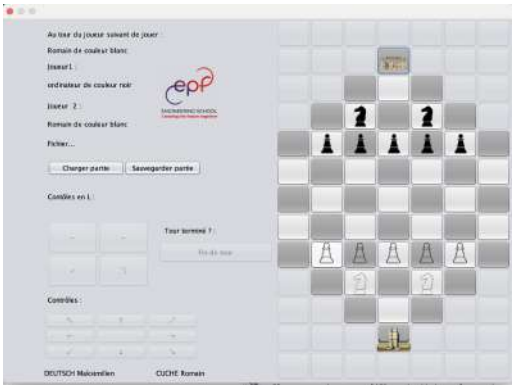
En diagonal on peut voir que le pion peut manger deux joueurs.



Lorsqu'un joueur réussit à manger tous les autres pions de l'autre, alors il gagne la partie et un message s'affiche.

```
deplacement en cours
Capture réussie !
Victoire ! L'adversaire n'a plus de pieces.
pfiches.Joueur@53a54a44
```

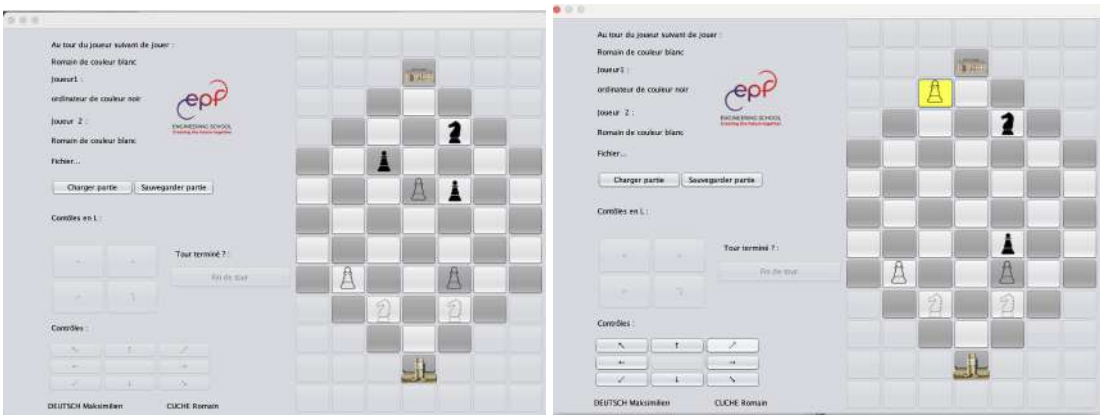
B. Contre Ordinateur



Une fois lancé en mode ordinateur encore une fois le premier à jouer est tiré au sort grâce à la méthode tirage dans FAcceuil


Dans cette exemple l'odinateur est de couleur Noir, on peut voir qu'il cherche directement a attaquer les joueurs blanc.


Ainsi l'ordinateur est en capacité de prendre des pions adverses.



```
Nouvelle position de pion39 : (3, 1)
Victoire ! Une piece blanca capture le chateau adverse.le joueurblanc a gagne
pfiches.Joueur@481e5c96
```

On peut voir à présent une nouvelle manière de gagner en prenant directement le château adverse.

C. Sauvegarde

Grâce au bouton “Sauvegarder partie” le joueur va pouvoir Enregistrer une partie et si il quitte la partie qu’il revient et qu’il appuie sur charger la partie, sa partie qui aura été préalablement sauvegardée pourra continuer.
Même si on a changé de pseudo, lorsqu'on charge une partie sauvegardé les pseudos s’actualise avec les nouveaux.

IV. Bilan/Repartition Travail

Fonctionnalités implémentées	Réalisé par	État	Détails
Initialisation du plateau et des pièces	Maksimilien & R...	✅	Création dynamique du plateau 7x13 et placement initial des pions/cavaliers
Système de déplacement des pièces	Maksimilien & R...	✅	Implémentation des règles de déplacement (pions + cavaliers en L)
Mécanique de capture par saut	Maksimilien	✅	Gestion des captures avec saut par-dessus la pièce adverse
Détection des conditions de victoire	Maksimilien & R...	✅	Vérification de la capture du château ou élimination de toutes les pièces
Interface graphique (Swing)	Maksimilien & R...	✅	Création des boutons interactifs, affichage des pièces et des châteaux
Sauvegarde/Chargement des parties	Maksimilien	✅	Système de serialisation dans un fichier texte
Alternance des tours entre joueurs	Romain	✅	Gestion dynamique du joueur actuel avec affichage
Mode contre l'IA (basique)	Maksimilien	✅	IA aléatoire implémentée (niveaux difficiles en développement)
Gestion des erreurs (mouvements invalides)	Maksimilien & R...	✅	Messages d'erreur pour les clics hors plateau ou sur pièces adverses

V. Conclusion

Ce projet de Jeu de Château a été une expérience extrêmement enrichissante qui nous a permis d'explorer en profondeur les possibilités de Java, et particulièrement sa dimension graphique. Travailler sur une interface visuelle interactive a donné tout son sens à notre apprentissage, transformant des lignes de code abstraites en un véritable jeu fonctionnel et attrayant.
Ce qui rend cette expérience encore plus gratifiante, c'est la formidable dynamique d'équipe que nous avons développée. Depuis septembre, nous avons constaté une belle évolution dans notre façon de collaborer : là où nos premiers projets étaient parfois réalisés dans l'urgence, nous avons cette fois su nous organiser efficacement, anticiper les étapes et répartir intelligemment les tâches. Cette maturité acquise nous a permis non seulement de respecter parfaitement le cahier des charges, mais aussi d'aller plus loin en ajoutant des fonctionnalités supplémentaires et en soignant particulièrement l'aspect visuel pour un rendu final à la fois technique et esthétique.
Au-delà du résultat concret, c'est cette progression dans notre maîtrise de Java et dans notre capacité à travailler en binôme qui nous rend particulièrement fiers. Ce projet ludique nous a offert l'opportunité d'exprimer notre créativité tout en consolidant nos compétences techniques - une combinaison parfaite pour clore cette année en beauté !

Notre méthode pour gérer l'IA est assez primitive, l'ordinateur ne peut se déplacer que orthogonalement, mais celle-ci peut être améliorée assez facilement.
Nous allons dans le futur ajouter des popups d'informations, pour savoir quand un mouvement est impossible ou non.
De plus, nous n'avons pas paramétré la charge du cavalier, ce qui peut être fait dans le futur.
Finalement nous pourrions améliorer l'esthétique de notre jeu, car ici elle est assez rudimentaire.