Projet de Programmation

Haimoud Sarah

Zidani Rababe

Dans ce projet nous devions coder en java deux jeux dérivés du domino. Pour le premier nous avons trois parties Tuile.java la base, TuileInterface.java et Plateau.java que vous retrouverez sur Moodle et en Annexe.
Pour le deuxième jeu, "Trax", nous nous sommes inspirés du principe du premier jeu. Nous avons utilisé un peu prêt le même principe de l'interface plateau pour disposer les tuiles. Nous avons réussi à mettre en place la classe "Tuile.java", qui représente une tuile de jeu avec deux images distinctes pour ses faces.

Le Premier Jeu

La classe Tuile

Tout d'abord, nous avons commencé avec Tuile.java la base pour définir les tuiles et leurs méthodes utiles pour la rotation et la compatibilité.

Nous avons choisit un tableau d'entier de 4 lignes (côtés) et 3 colonnes (chifrres) pour représenter nos tuiles :

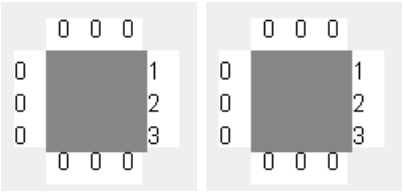
```
private int[][] chiffres = new int[4][3]; // 4 côtés, 3 chiffres par côté

public Tuile(int[] haut, int[] droit, int[] bas, int[] gauche) {
    chiffres[0] = haut;
    chiffres[1] = droit;
    chiffres[2] = bas;
    chiffres[3] = gauche;
}
```

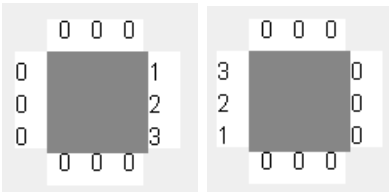
La fonction rotation (cf Annexe) de la classe Tuile permet de faire pivoter les côtés d'une tuile dans le sens horaire, en déplaçant chaque côté à la position du suivant. On change l'orientation de la tuile sans altérer l'ordre ou la disposition des chiffres sur chaque côté.

Ensuite, nous avons la fonction fait la fonction estCompatible afin de vérifier ou non que deux tuiles puissent être placés à côté. Cette fonction a subit plusieurs transformation au cours du projet car au début nous ous étions dit que 2 tuiles étaient compatibles seulement si elles avaient un côté en commun. Seulement en faisant l'interface, nous nous sommes rendu compte que c'était plus compliqué que ça.

En effet, si tuile 1 et 2 sont similaires telles que :



Après rotation de la tuile 2:



Exception, si les 3 chiffres sont les mêmes, la compatibilité est vraie dans tout les cas. Ainsi, nous avons compris que les chiffres doivent êtres inversés. Ainsi sont définis nos conditions de compatibilité.

La classe TuileInterface

Le but de cette classe était de construire une première interface pour modéliser deux tuiles, les faire pivoter et vérifier leur compatibilité.

L'interface graphique étend JFrame et utilise des composants Swing pour interagir avec l'utilisateur, nous nous sommes servis du cours mais aussi de documentation sur internet tels que KooR.fr - Description de la classe javax.swing.JComponent , Graphiques en SWING (irif.fr) et aussi des vidéos tutoriels.

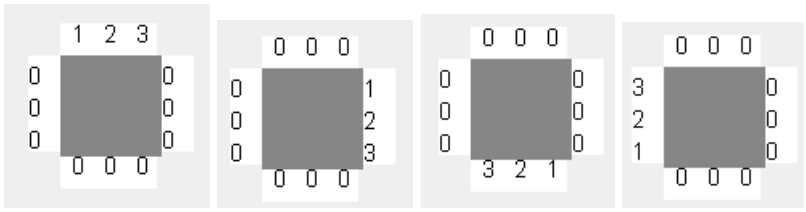
- TuilePanel: Sous-classe de JPanel conçue pour afficher une tuile. Chaque TuilePanel est associé à une instance de Tuile et redéfinit la méthode paintComponent(Graphics g) dans laquelle nous appellons drawTuile notre méthode personnalisée pour dessiner la tuile.

- Boutons : Des instances de JButton sont utilisées pour les actions de rotation des tuiles et de vérification de compatibilité. Ces boutons sont équipés de ActionListener qui invoquent les méthodes rotation et estCompatible sur les tuiles et rafraîchissent les affichages.

- Pour la méthode drawTuile(Graphics g, Tuile tuile) dans TuilePanel:

- Calcul de Position : Calcule le centre du panel pour positionner le carré central de la tuile et ajuste les positions pour dessiner les chiffres sur chaque côté.

- Dessin des côtés : Utilise une boucle pour parcourir les chiffres de chaque côté et les dessine en ajustant leur positionnement basé sur le côté qu'ils représentent, inversant l'ordre pour les côtés bas et gauche pour maintenir une lecture comme une tuile qui pivote dans la vraie vie.



- Gestion des Événements:

- Les ActionListeners attachés aux boutons capturent les clics et déclenchent des rotations ou des vérifications de compatibilité.

Précisons que nous avons trouvé les fonctions suivantes en nous documentant pour avoir l'interface la plus claire possible.

g.fillRect(xPos, yPos, width > height ? numWidth : width, height > width ? numHeight : height); //rectangle plein

g.drawString(String.valueOf(numToDraw), textX, textY); //convertie nos int en string pour l'affichage des chiffres

FontMetrics fm = g.getFontMetrics(); // encapsule les dimenssions et la police des caractères

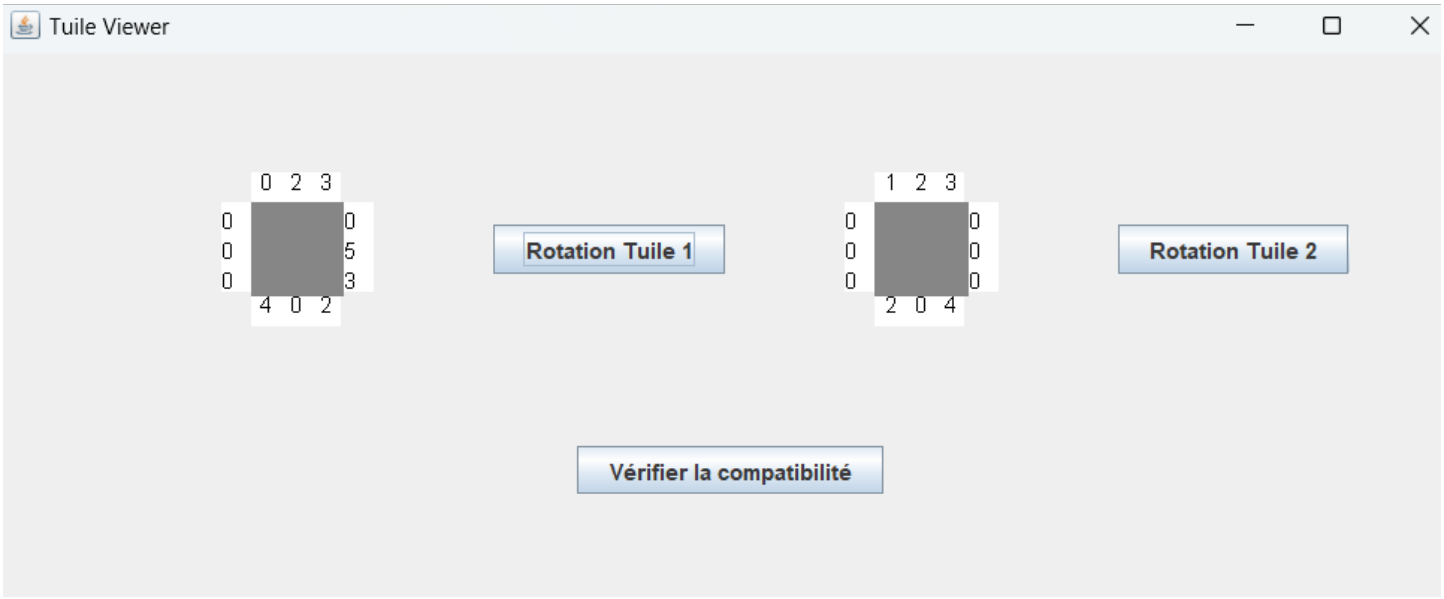
int adjust = fm.getAscent() / 2; // utilisé pour centrer verticalement le texte dans son cadre de dessin

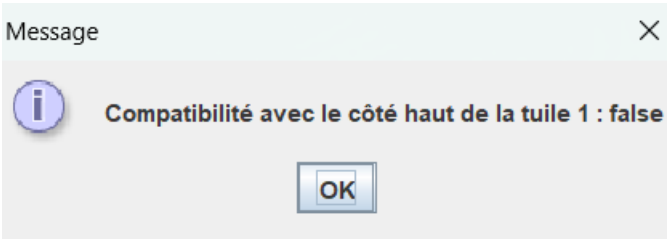
JOptionPane.showMessageDialog(null, "Compatibilité avec le côté " + directions[i] + " de la tuile 1 : " + compatible); //affiche une boite de dialogue

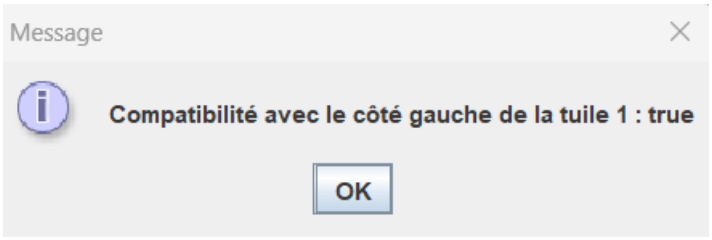
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //application doit se terminer quand on ferme la fenetre

SwingUtilities.invokeLater() // evite les bug, manipulation sure de l'interface utilisateur

Finalement, on obtient une interface assez claire et une boîte de dialogue pour la compatibilité :



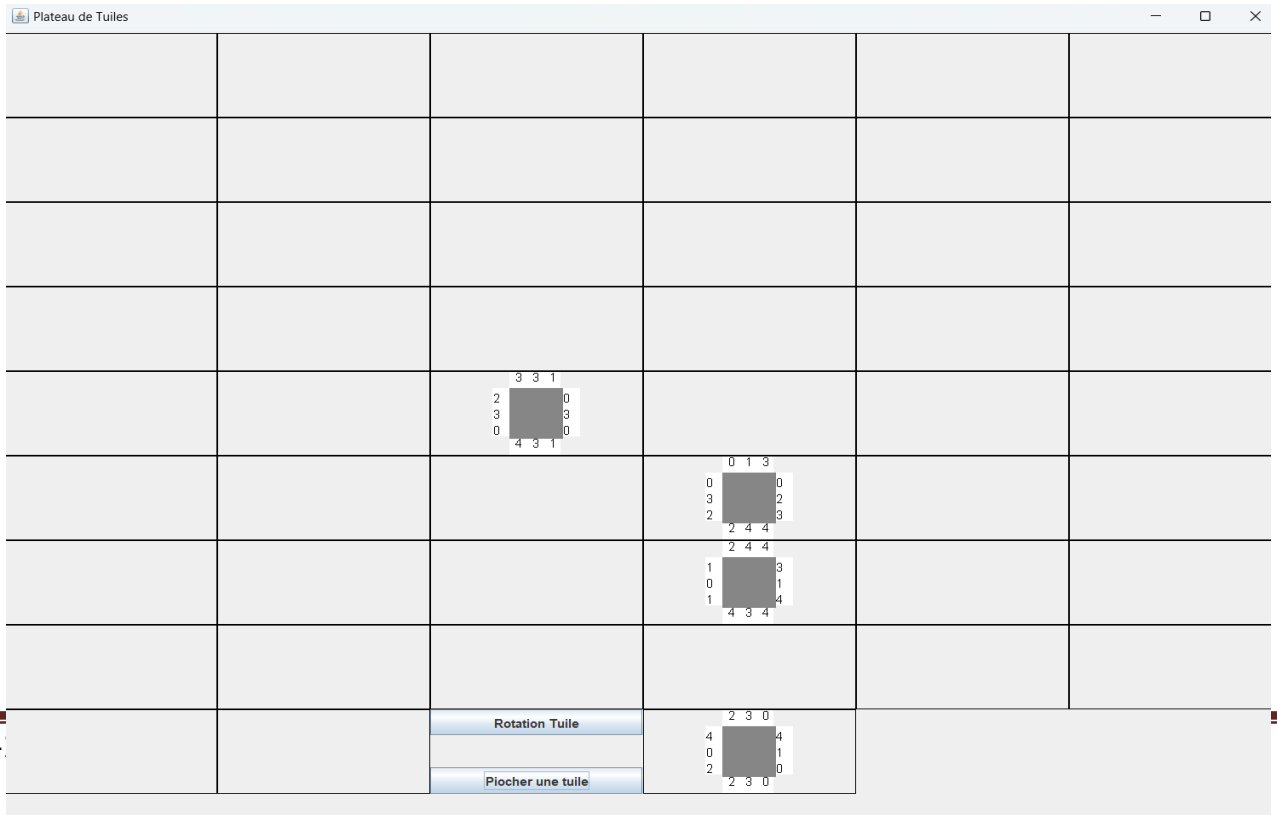




La classe Plateau

L'idée ici est de créer un plateau de jeu, il faut alors une pioche, la possiblité de faire pivoter la tuile piochée et des cases où placer ces tuiles.

Voici à quoi ressemble notre interface :



Nous avons 2 boutons, « Piocher une tuile »on pioche une tuile et elle s'affiche à droite, « Rotation Tuile » on fait pivoter la tuile piochée dans la case à droite.

On pose une tuile dans le plateau en cliquant sur une case du plateau, une fois posée dans le plateau la tuile ne peut plus pivoter. C'est au prochain joueur de piocher sur la même interface.

Décrivons notre programme (cf Annexe):

On a reutilisé TuilePanel décrit précédemment.

- private Tuile[] tuilesSurPlateau; : un tableau stockant jusqu'à 50 tuiles.

- private Tuile tuileActive;: la tuile actuellement sélectionnée pour manipulation.

- private Random random; : pour générer des chiffres aléatoires sur les tuiles.

- private TuilePanel tuilePreviewPanel; : un panneau pour afficher la tuile piochée.

- private JButton rotateButton, piocherButton; : boutons pour la rotation et la pioche de nouvelles tuiles.

- Méthode constructeur Plateau() :

- Initie le tableau de tuiles et définit la disposition du panneau à un grille de 10x5.

- Appelle initializePlateau() pour configurer le plateau avec des écouteurs d'événements et initializePreviewAndRotation() pour préparer les interactions.

InitializePlateau(): Remplit le plateau avec des instances de `TuilePanel`, chacune capable de recevoir une tuile si sélectionnée. Chaque panneau a un MouseListener qui place une tuileActive dans le panneau sélectionné si celui-ci est vide, illustrant ainsi le placement des tuiles sur le plateau.

initializePreviewAndRotation(): Configure le panneau de prévisualisation et les boutons de contrôle. Le bouton de rotation appelle la méthode rotation() de tuileActive lorsqu'il est cliqué, et le bouton de pioche permet de tirer une nouvelle tuile avec des chiffres aléatoires sur ses côtés.

Les actions de ces boutons déclenchent des redessins du tuilePreviewPanel pour refléter les changements.

- piocherTuile() : Génère une nouvelle tuile avec des côtés aléatoires, utilisée pour simuler la pioche de tuiles dans un jeu réel. Cette fonction est essentielle pour maintenir l'interactivité et la variabilité du jeu.

Le Deuxième Jeu

La classe Tuile

Cette classe permet de choisir l'image en fonction du type de tuile spécifié. Elle inclut une fonctionnalité de rotation, permettant à la tuile de pivoter de 90 degrés à chaque invocation.

La classe ajuste la taille des images pour qu'elles correspondent aux dimensions du panneau (100x100 pixels) et gère le rendu graphique de la tuile avec sa rotation actuelle.

```
public Tuile(boolean isCrossTile) {
    this.isCrossTile = isCrossTile;
    setPreferredSize(new Dimension(100, 100));
    this.rotationState = 0;

    try {

        if (isCrossTile) {
            image = ImageIO.read(getClass().getResourceAsStream("Tuile1.jpeg"));
        } else {
            image = ImageIO.read(getClass().getResourceAsStream("Tuile2.jpeg"));
        }

        Image scaledImage = image.getScaledInstance(100, 100, Image.SCALE_SMOOTH);
        image = new BufferedImage(100, 100, BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2d = image.createGraphics();
        g2d.drawImage(scaledImage, 0, 0, null);
        g2d.dispose();
    } catch (IOException e) {
        e.printStackTrace();

    }
}
```

Initialise les variables et définit les préférences de taille de la tuile à 100x100 pixels.

Charge l'image appropriée (Tuile1.jpeg ou Tuile2.jpeg) en fonction du type de tuile.

Redimensionne l'image chargée pour qu'elle corresponde à la taille du composant, garantissant ainsi que l'image s'affichera proprement adaptée.

- Méthode rotate():

```
public void rotate() {
    this.rotationState = (this.rotationState + 1) % 4;
    repaint();
}
```

Incrémente l'état de rotation de la tuile et boucle autour de 0 à 3 (0, 90, 180, 270 degrés).

Redemande le rafraîchissement du rendu de la tuile, ce qui entraîne un appel à paintComponent.

- Méthode paintComponent(Graphics g):

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g.create();

    int x = (getWidth() - image.getWidth()) / 2;
    int y = (getHeight() - image.getHeight()) / 2;


    g2d.rotate(Math.toRadians(rotationState * 90), getWidth() / 2.0, getHeight() / 2.0);
    g2d.drawImage(image, x, y, this);
    g2d.dispose();
}
```

Méthode surchargée de JPanel pour dessiner la tuile sur le composant.

Crée un contexte graphique (Graphics2D) pour le dessin avancé.

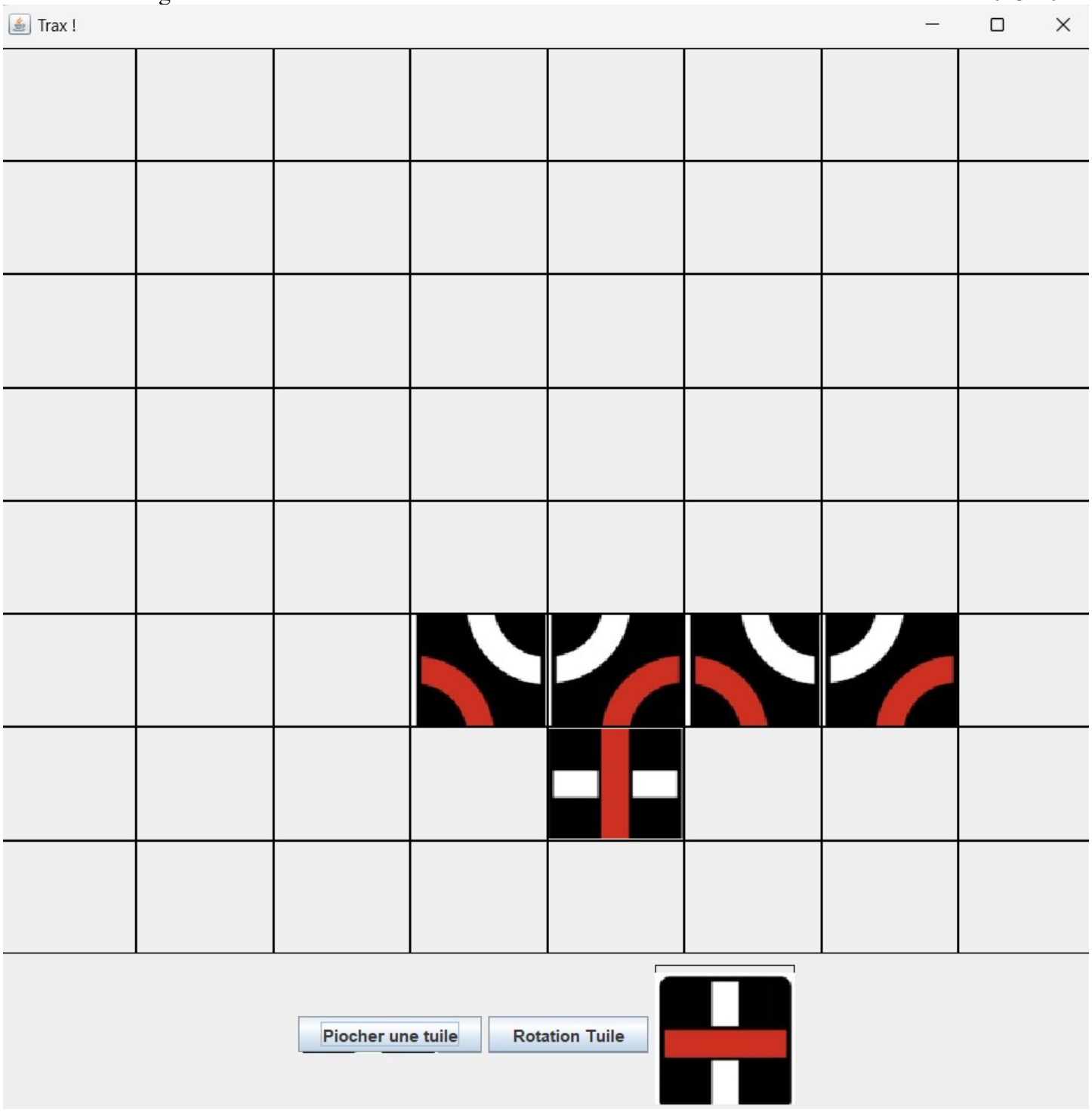
Applique la rotation à l'image en fonction de rotationState.

Centre l'image dans le composant, ajustant sa position pour que l'axe de rotation soit le centre du composant.

Affiche l'image et libère les ressources graphiques après l'affichage.

La classe Plateau

Voici à quoi ressemble notre interface :



La classe Plateau est conçue pour gérer l'interface et les interactions de ce jeu . Elle fournit une grille de jeu de 8x8 sur laquelle les joueurs peuvent placer des tuiles, ainsi que des fonctionnalités pour piocher et faire pivoter ces tuiles.

Voici les fonctions clés :

1. Construction et initialisation :

Constructeur: Configure la grille de jeu et les panneaux de contrôle, initialise la grille avec des panneaux interactifs pour placer les tuiles.

Gestion des tuiles :

- piocherTuile(): Permet de tirer une tuile au hasard entre deux types possibles et de l'afficher dans un espace de visualisation dédié.

```
private Random random = new Random();

  public void piocherTuile() {
      boolean isCrossTile = random.nextBoolean();

      selectedTile = new Tuile(isCrossTile);
      holdingArea.add(selectedTile);
      holdingArea.revalidate();
      holdingArea.repaint();
  }
```

- rotateTile(): Permet de faire pivoter la tuile sélectionnée afin de modifier son orientation.

```
public void rotateTile() {
    if (selectedTile != null) {
        selectedTile.rotate();
        holdingArea.repaint();
    }
}
```

- placeTile(int x, int y, JPanel panel): Place la tuile sélectionnée dans la grille à l'emplacement spécifié par l'utilisateur.

```
  public void placeTile(int x, int y, JPanel panel) {
      if (grid[x][y] == null && selectedTile != null) {
          grid[x][y] = selectedTile;
          panel.add(selectedTile);
          selectedTile = null;
          holdingArea.removeAll();
          holdingArea.revalidate();
          holdingArea.repaint();
          panel.revalidate();
          panel.repaint();
      }
  }
```

2. Interface utilisateur :

initializeGrid(JPanel gridPanel): Peuple le panneau de la grille avec des panneaux cliquables qui permettent de placer les tuiles.

En somme, Plateau facilite l'interaction avec un jeu de tuiles via une interface graphique, offrant des options simples pour la manipulation des tuiles et la visualisation de l'état du jeu.

Pour ce jeu, nous n'avons pas réussi à implémenter toutes les fonctionnalités prévues. Cependant, nous avons réussi à créer des tuiles recto-verso et à configurer un plateau de jeu de 8x8 cases pour leur placement. De plus, nous avons intégré un système de sélection aléatoire des tuiles et ajouté une fonctionnalité permettant de faire pivoter les tuiles.