École Centrale Lyon

UE d'Approfondissement
Rapport

Jeu de l'araignée

Élèves :
Valentin Verdon
Nathan Bauerlé

Enseignant :
Stéphane Derode

6 octobre 2022

Table des matières

1 Introduction

Ce rapport présente notre Jeu de l'araignée réalisé dans le cadre du cours Applications concurrentes, mobiles et réparties en Java. Vous retrouverez dans le dossier le fichier exécutable de notre projet ainsi que nos codes et un guide d'utilisation du jeu.

1.1 Présentation du jeu

Le jeu de l'araignée se joue à deux joueurs sur une grille de 3 cases par 3 cases dans laquelle il faut aligner ses 3 pions. Ces pions seront positionner dans une première phase à tour de rôle (3 pions par personne), puis dans une seconde phase, ces pions pourront être déplacés sur une case libre dans un rayon d'une case afin de les aligner.

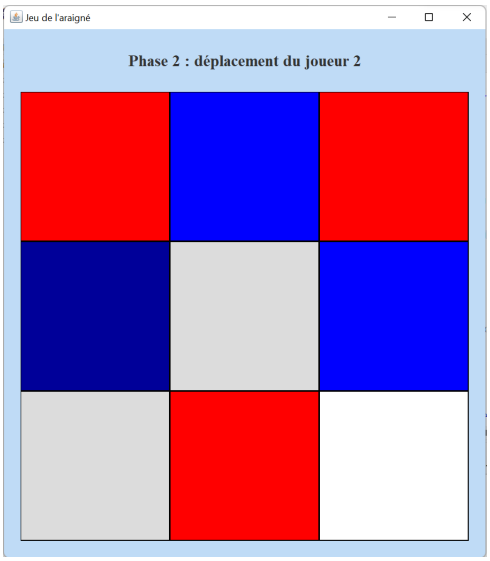

FIGURE 1 – Exemple de partie

Légende de la figure 1 :
- case rouge : joueur 1
- case bleu : joueur 2
- case blanche : case non occupée
- case grise : case non occupée qui peut recevoir le pion rouge en haut à gauche (ce pion a été sélectionné au préalable).

2 Organisation du programme

Notre programme est constitué d'une classe principale nommée Main_jeu qui contient une fonction main et dans laquelle l'interface est programmée. Cette classe hérite de la classe JFrame. La fonction main appelle une instance de la classe Main_jeu.

L'interface se compose de trois fenêtres : une fenêtre d'accueil, une fenêtre de jeu et une fenêtre de résultat. Ces fenêtres sont des JPanel que l'on affiche et cache en fonction de l'action de l'utilisateur.

Le JPanel d'accueil s'affiche au lancement du jeu.
Le JPanel de jeu s'affiche si le bouton "Commencer" ou "Recommencer" sont activés.
Le JPanel de résultat s'affiche lorsqu'un joueur gagne la partie.
Le JPanel de jeu se compose d'une grille permettant aux joueurs de placer leurs pions.

La grille de jeu est composée de Pions sur lesquelles nous allons modifier les couleurs en fonction des différentes actions réalisées. La classe Pions qui hérite de la classe JButton sera présentée plus tard dans le rapport.

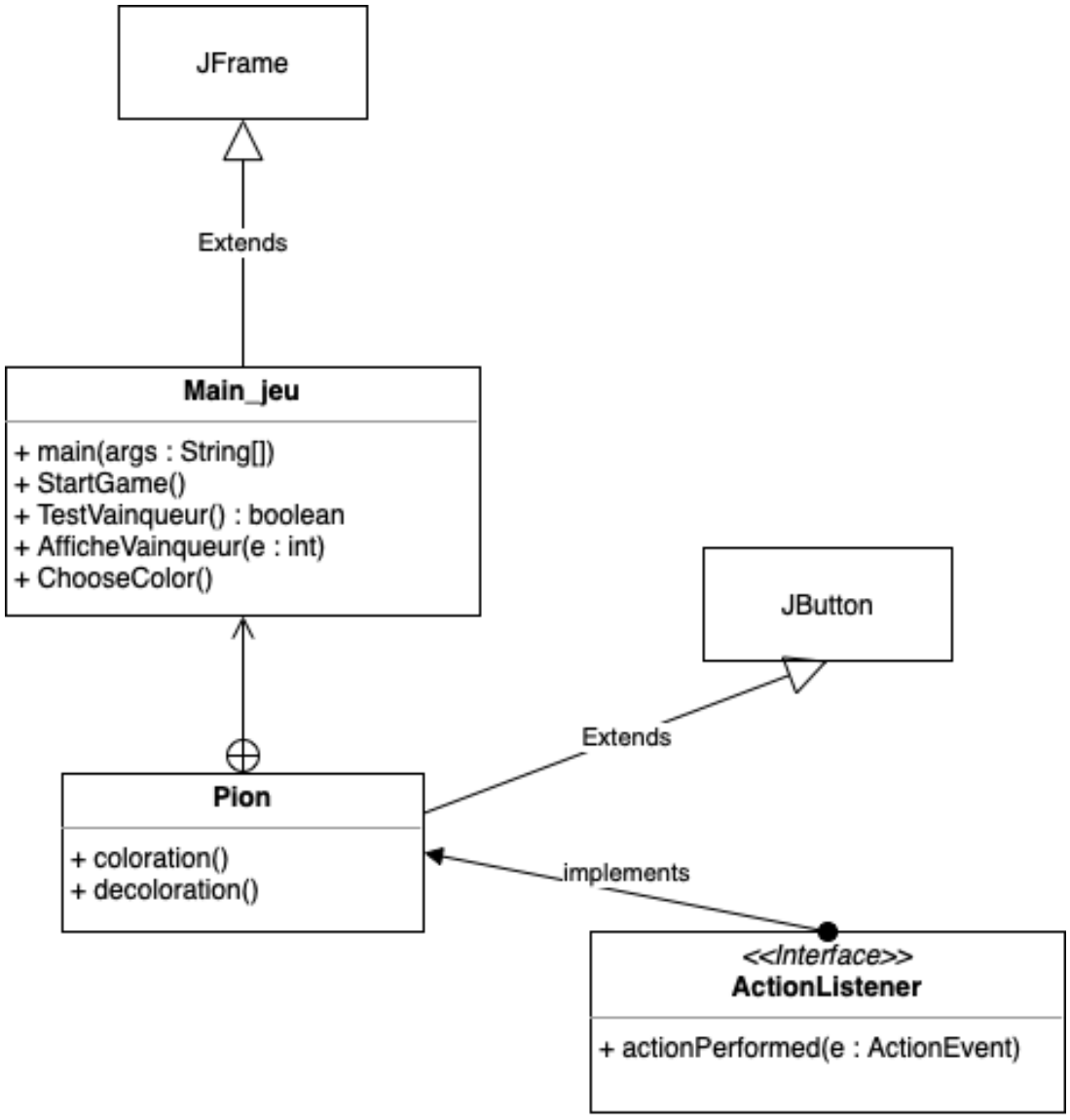

FIGURE 2 – placement des pions

3 Présentation de la classe Pion

Nous avons décidé, afin de grandement simplifié le programme, d'utiliser pour représenter les pions des boutons, qui vont simplement changer de couleur pour indiquer à quel

joueur il appartient. Nous avons donc implémenté une classe Pion qui hérite de la classe JButton.

3.1 Placement des pions

Pour placer un pion, il est nécessaire de respecter certaines conditions. Tout d'abord, la case doit être inoccupée, ensuite il faut que le pions soit au plus le 6ème à être posé car il y a au total 6 pions. Enfin, il faut que le pion poser corresponde bien au joueur et qu'au prochain tour cela soit à l'autre joueur de jouer.

Voici le code de cette action :

```
@Override
public void actionPerformed(ActionEvent e) {


    // choix des pions (phase 1)
    if ((Total_activated < 6) && (etat == 0) ) {
        coloration();
        Total_activated ++;
        annonce.setText("Phase 1 : Choix des cases du joueur "+String.valueOf(Joueur));
        if (TestVainqueur() == true) {
            AfficheVainqueur(etat);
        }
    }
```

FIGURE 3 – placement des pions

Ici, le test permet de savoir si le nombre de pions poser est bien inférieur à 6 et si la case n'est pas occupée. En effet, la case possède 3 états : 0 correspond à libre et 1 ou 2 qui correspondent à une occupation par le joueur 1 ou 2.
La fonction *coloration* permet de mettre la bonne couleur au bouton. Le texte présent en haut de la zone de jeux est également modifié pour indiquer que c'est au joueur suivant de jouer. Finalement, nous avons un dernier test qui consiste à vérifier si le joueur à gagner en plaçant son pion (ce test sera étudier dans la partie suivante).

3.2 Déplacement des pions

Pour déplacer le pion, deux étapes sont nécessaires. Tout d'abord, il faut sélectionner le pions à déplacer puis dans un second temps il faut choisir la case qui va recevoir le pion.

3.2.1 Choix du pion

Lorsque l'on choisis le pion, il faut que celui-ci appartienne bien au joueur qui est entrain de jouer. Si c'est la cas, on modifie légèrement la couleur du pion sélectionné pour que celui-ci soit visible. Dans le cas où on change de pion que l'on souhaite déplacer, on remet la couleur d'origine du pion (ses coordonnée sont stockées dans les variables mem_i et mem_j. La variable *wantmove* permet de savoir si le joueur à sélectionné sont pion pour passer à l'étape suivante : le choix de la case d'arrivée.

```
// selection de la case à deplacer
if(etat == Joueur) {
    if(etat == 1) {
        setBackground(new Color((int) (red1*0.6),(int) (green1*0.6),(int) (blue1*0.6)));
        if((wantMove ==true)&&(mem_i!=indice(this)[0])&&(mem_j!=indice(this)[1])) Pions[mem_i][mem_j].setBackground(couleur_j1);
    }
    else {
        setBackground(new Color((int) (red2*0.6),(int) (green2*0.6),(int) (blue2*0.6)));
        if((wantMove ==true)&&(mem_i!=indice(this)[0])&&(mem_j!=indice(this)[1])) Pions[mem_i][mem_j].setBackground(couleur_j2);
    }

    mem_i = indice(this)[0];
    mem_j = indice(this)[1];
    mem_etat = etat;
    wantMove = true;

    for (int i=0; i<3;i++) {
        for (int j=0; j<3;j++) {
            if(Pions[i][j].etat == 0) {
                if ((Math.abs(i-indice(this)[0])<2)&&(Math.abs(j-indice(this)[1])<2)) {
                    if(etat == 1) {
                        Pions[i][j].setBackground(new Color(220,220,220)); //on indique au joueur les possibilté de mouvement
                    }
                    else {

                        Pions[i][j].setBackground(new Color(220,220,220));
                    }
                }
                else {
                    Pions[i][j].setBackground(Color.white);
```

FIGURE 4 – Choix du pion à déplacer

La seconde partie du code qui contient les boucles for permet de tester si les cases sont voisines du pion sélectionné et si celles-ci ne sont pas occupées. Dans ce cas, elles sont surlignées en gris pour les rendre visibles.

Voici le rendu lorsque le pion est sélectionné :

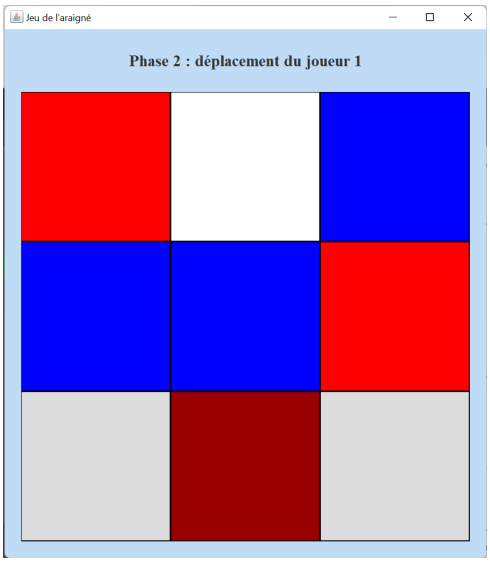

FIGURE 5 – Pion sélectionné

3.2.2 Choix de la case qui reçoit le pion

Dans cette phase, le joueur sélectionne une case qui doit recevoir le pion en cours de déplacement. Il faut vérifier que celle-ci soit voisines de celle qui est en cours de déplacement et qu'elle ne soit pas occupée. Il faut également vérifier qu'une case a été sélectionné préalablement. Si la case choisie est valide, il faut modifier sa couleur en celle du joueur,

remettre la couleur d'origine (blanc) à la case anciennement occupée et remettre cette même couleur aux cases qui avaient été surlignée.

```
// selection de la destination et deplacement
if((wantMove == true) && (etat == 0)) {
    if ((Math.abs(mem_i-indice(this)[0])<2)&&(Math.abs(mem_j-indice(this)[1])<2)) {
        coloration();
        Pions[mem_i][mem_j].decoloration();
        wantMove = false;

        for (int i=0; i<3;i++) {
            for (int j=0; j<3;j++) {
                if(Pions[i][j].etat == 0) {
                    Pions[i][j].setBackground(Color.white);
                }
            }
        }

        if (TestVainqueur() == true) {
            AfficheVainqueur(etat);
```

FIGURE 6 – Choix de la case qui reçoit le pion

Une dernière ligne de code est présente (non visible sur 6) qui permet de mettre à jour le texte d'indication pour informer que c'est à l'autre joueur de jouer.

4 Victoire d'un joueur

Un joueur gagne la partie lorsque ses 3 pions sont alignés sur la grille. Ainsi, on vérifie cette condition à chaque changement sur la grille, c'est à dire à chaque placement ou déplacement de pion.

Pour réaliser ce test, on parcours les différents pions dans les quatre directions possibles en vérifiant si ils sont dans le même état (état 1 ou 2). Si c'est le cas, cela signifie que l'un des joueurs a 3 pions alignées et qu'il a gagné. On affiche alors la fenêtre de victoire avec le numéro du joueur.

```
//Test si le mouvement fait gagner la partie
public boolean TestVainqueur() {

    for (int i=0; i<3; i++) {
        if ((Pions[i][0].etat == Pions[i][1].etat) && (Pions[i][0].etat == Pions[i][2].etat) && (Pions[i][0].etat != 0)) {
            return true;
        }
    }

    for (int j=0; j<3; j++) {
        if ((Pions[0][j].etat == Pions[1][j].etat) && (Pions[0][j].etat == Pions[2][j].etat) && (Pions[0][j].etat != 0)) {
            return true;
        }

    }

    if ((Pions[0][0].etat == Pions[1][1].etat) && (Pions[0][0].etat == Pions[2][2].etat) && (Pions[0][0].etat != 0)) {
        return true;
    }


    if ((Pions[0][2].etat == Pions[1][1].etat) && (Pions[0][2].etat == Pions[2][0].etat) && (Pions[0][2].etat != 0)) {
        return true;
    }

    return false;

}
```

FIGURE 7 – Test Victoire

Et voici le rendu de la fenêtre de victoire :

FIGURE 8 – Fenêtre victoire

5 Conclusion

Le rendu final nous donne un jeu à l'aspect minimaliste qui permet une expérience utilisateur simple et intuitive. Le choix de mettre peu d'image et de label est volontaire afin d'avoir un visuel simple et de nous concentrer sur les fonctionnalités du jeu. Ce rapport ne présente que les fonctionnalités de bases mais on peut retrouver d'autres fonctionnalités auxiliaires comme le changement de couleur des pions ou la gestion des différents panel.