Rapport de projet de fin d'année

Maroan AKKA et Benjamin RONDO

17 Mai 2023

Contents

List of Figures

1 Introduction

Pour ce semestre , le projet attribué est la création d'un jeu de blocage avec des polyominos . Chaque joueur dispose d'un ensemble de pièces, qui sont des polyominos, et qu'il place tour à tour sur un plateau (une grille rectangulaire). Il s'agit d'un jeu de blocage : le perdant est le premier joueur qui ne peut plus placer de pièce. Les polyominos mis à disposition des joueurs sont au nombre de 3 :
Les dominos de taille 2 ayant 2 dispositions soit horizontale ou verticale

Les triominos de taille 3 ayant 6 positions , on a deux types de triominos :
- La forme allongée qui a donc 2 dispositions (soit il pointe à gauche soit à droite)
- La forme en " L " qui a également 2 dispositions avec une rotation à 90°
Pour les triominos on a 3 pièces qui ont chacunes 2 dispositions en jouant avec son orientation .
Les tetrominos de taille 4 ayant 19 dispositions :

2 Développement

Pour ce qui est des classes développées dans notre projet , elles sont au nombre de 7 :
- 1] Grille.java
- 2] Jeu.java
- 3] Joueur.java
- 4] JoueurIA.java
- 5] Piece.java
- 6] Position.java
- 7] Main.java

Pour développer ce projet , nous avons procédé par étapes :
- Etape 1 : Avoir un tableau de jeu fonctionnel
- Etape 2 : Pouvoir initialiser un dominos sur le plateau
- Etape 3 : Initialiser un triominos sur le plateau
- Etape 4 : Initialiser un tetrominos sur le plateau
- Etape 5 : Mettre en place une IA contre qui jouer

Entre chacune de ces étapes , il y a eu des sous étapes pour continuer . Par exemple entre l'étape 2 et 3 on a dû coder une fonction pour eviter que deux pièces se superposent les unes sur les autres .

2.1 La classe Grille

La première étape du projet est donc de commencer par coder la classe **Grille** qui sera le fondement de notre programme :
On a donc commencé par créer un plateau de jeu . Pour se faire on a choisi de la faire sous forme de matrice d'objet .
On a définit chaque chaque case du plateau comme un objet Position préalablement déclarer dans une classe Position .
On définit deux boucles "for" , une première qui va nous servir à créer les lignes et la seconde les colonnes de la grille . On a ensuite codé une simple fonction nous permettant d'afficher la grille
(Voir Annexe Figure 1)

2.1.1 Placer une pièce :

Par la suite , on a travaillé sur la fonction permettant de placer une pièce .
La fonction fonctionne comme ceci :
On récupère la position (définit par x et y) de notre pièce et on impose un booléen " impossible " à faux , ce qui équivaut à dire que placer une pièce est possible .
La fonction placerPiece() vérifie si une pièce peut être placée sur une grille à une position donnée. Il itère sur les cases de la matrice de la pièce et vérifie si chaque case est valide (à l'intérieur de la grille et vide) pour placer la pièce. Si le placement est impossible, un message est affiché, sinon, les cases correspondantes dans la grille sont mises à jour avec la valeur de la pièce. Finalement, la méthode renvoie un booléen indiquant si le placement était possible ou non.
(Voir Annexe Figure 2)

2.1.2 Méthode Rotation:

Un autre type de fonction importante dans cette classe est de savoir s'il est possible de placer un dominos , un triominos ect... avec toutes orientations possibles .
Cette méthode parcourt une grille de 10 lignes et 12 colonnes et vérifie s'il est possible de placer des dominos. Elle renvoie true si une paire de cases adjacentes (à droite ou en dessous) est libre, indiquant que des dominos peuvent être placés. Sinon, elle renvoie false pour indiquer qu'il n'est pas possible de placer des dominos.
On a ensuite refait la même methode pour chaque pièce et chaque rotation en adaptant le code
(Voir Annexe Figure 3)

Maintenant que la classe Grille est définie et que l'on peut générer un plateau de jeu où l'on peut poser différentes pièces .

2.2 La classe Jeu

On passe maintenant à la classe Jeu :

Concernant les choix faits pour cette classe , on a décidé de garder les dimensions en " dur " dans le code soit 10 lignes / 12 colonnes . On aurait très bien pu rendre les dimensions plus flexibles en demandant à l'utilisateur de les saisir mais cela peut entraîner un décalage au niveau de l'affichage du plateau et dans certains cas un problème au niveau du placement des pièces .

2.2.1 Méthode GameBasic:

Une des fonctions les plus importantes concerne l'initialisation d'une partie :
On instancie une grille , un joueur , un joueur IA et une variable answer qui correspond à la réponse de l'utilisateur pour la position de la pièce .
On créer ensuite chaque type de pièce avec le nombre de pièces demandé . Par exemple 1 pièce domino , 2 pièces triomino et 7 pièces tétrominos .
(Voir Annexe Figure 4)

2.2.2 Méthode Partie Gagnée:

L'autre fonction importante de cette classe est celle qui nous aide à déterminer si une partie est terminée ou non .
Pour savoir si le joueur peut continuer à jouer , on vérifie si un domino peut être placé sur la grille , si c'est le cas , on peut donc jouer au moins 1 tour .
Si la partie pour les dominos renvoie "true" on passe à la prochaine condition qui fonctionne de la même manière , puis de même pour les tétrominos .
Si aucune de ces conditions n'est vraie cela signifie que le joueur adverse peut encore placé des pièces donc que notre joueur n'a pas encore gagné .

En résumé, cette méthode vérifie si le joueur a gagné en vérifiant si le joueur adverse ne peut plus placer aucune pièce sur la grille. Elle examine les possibilités de placement pour les dominos, les triominos et les tetrominos. Si aucune de ces pièces ne peut être placée, le joueur actuel est déclaré vainqueur. Sinon, le jeu continue.
(Voir Annexe Figure 6)

2.3 La classe Joueur :

2.3.1 Méthode Joueur:

On passe ensuite à la classe Joueur :
Une classe très simple de fonctionnement , un joueur est défini par le nombre de pièce au total a sa disposition et du nombre de pièces de chaque type .
On a decidé d'attribuer par défaut le nombre de pièce comme demandé dans l'énonce :
Une piece domino , 2 pièces triomino et 7 pièces tétrominos . La suite de la classe concerne l'affichage à l'écran où le joueur choisi la pièce qu'il veut placer , sa rotation et sa position avec la possibilité de revenir en arrière avec des conditions fixées pour éviter de placer une pièce en dehors des limites du plateau .
(Voir Annexe Figure 5)

2.3.2 Méthode JoueurIA:

Nous avons également une classe JoueurIA pour gérer l'adversaire , avec la même structure que la classe Joueur (au niveau des getters et setters) . L'idée de base était de développer un adversaire capable de réflechir au meilleur positionnement pour gagner la partie le plus rapidement possible .

On a rapidement vu que la mise en place d'une telle fonction était hors de notre portée , nous avons donc opté pour que l'IA choisi aléatoirement une pièce parmi celles disponible et la place également au hasard.

2.4 La classe Pièce

Passons à la classe Pièce , une classe assez simple sur son développement mais est une classe importante du projet . Dans cette classe , on crée une pièce avec les attributs passés en paramètres accompagnés de tous les getters et setters pour récupérés ces attributs (taille , position x , position y ect...)
(Voir Annexe Figure 5)

2.4.1 Méthode Rotation:

Avec cela , nous avons codé une fonction qui s'occupe de gérer la rotation de notre pièce .
Cette méthode réalise une rotation de 90 degrés sur la matrice d'une pièce en utilisant une nouvelle matrice de même taille pour stocker la matrice résultante. La rotation est effectuée en parcourant la matrice d'origine et en assignant les éléments correspondants de la matrice résultante selon la formule de rotation. Finalement, la matrice de la pièce est mise à jour avec la nouvelle matrice résultante de la rotation.

(Voir Annexe Figure 7)

2.4.2 Méthode Création de pièce avec rotation:

Pour terminer dans cette classe , nous avons créer pour chaque pièce et chaque rotation une pièce qui est représentée par une matrice de la taille de la pièce remplie de 1 .
La matrice est ensuite stockée dans l'attribut Matrice de l'objet Piece .
(Voir Annexe Figure 8)

3 Images annexes

```java
private Position[][] grille;

/**
 * Constructeur de la classe Grille
 * @param
 * @return
 */
Grille () {
    grille = new Position[10][12]; // 10 lignes et 12 colonnes
    for (int i = 0; i < 10; i++) { // 10 lignes
        for (int j = 0; j < 12; j++) { // 12 colonnes
            grille[i][j] = new Position(); // chaque case de la grille est un objet de type Position
        }
    }
}

/**
 * @return
 */
public void afficher() {
    for (int i = 0; i < 10; i++) { // 10 lignes
        System.out.print("| "); // debut de ligne
        for (int j = 0; j < 12; j++) { // 12 colonnes
            System.out.print(grille[i][j].getValeur() + " "); // affichage de la valeur de chaque case
            if (j == 11){ // fin de ligne
                System.out.print("|"); // fin de ligne
            }
        }
        System.out.println();
    }
    System.out.println();
}
```

Figure 1: Classe Grille

```java
public boolean placerPiece(Piece piece) { // placer une piece sur la grille
    int x = piece.getPos_x();
    int y = piece.getPos_y();
    boolean impossible = false;
    int[][] matrice = piece.getMatrice();
    for (int i = 0; i < piece.getTaille(); i++) {
        for (int j = 0; j < piece.getTaille(); j++) {
            if ((Est_pas_dans_la_grille(y+i,x+j)) || (Est_Occupe(y+i,x+j))) {
                impossible = true;
            }
        }
    }
    if (impossible) {
        System.out.println("La case est déjà occupée impossible de placer la pièce");
    }
    else {
        for (int i = 0; i < piece.getTaille(); i++) {
            for (int j = 0; j < piece.getTaille(); j++) {
                if ((matrice[i][j] == 1) && (Est_dans_la_grille(y+i,x+j)) && (Est_vide(y+i,x+j))) {
                    grille[y+i][x+j].setValeur(piece.getValeur());
                }
            }
        }
    }
    return impossible;
```

Figure 2: Fonction Placer Piece

```java
    public boolean Dominos_possible() {
        for (int i = 0; i < 10; i++) { // 10 lignes
            for (int j = 0; j < 12; j++) { // 12 colonnes
                if (Est_libre(i, j)) {
                    if (Est_libre(i, j+1)) {
                        return true;
                    }
                    if (Est_libre(i+1, j)) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
    /**

    Vérifie s'il est possible de placer un Triomino en forme de barre dans la grille.

    @return true si un Triomino en forme de barre peut être placé, false sinon.
    */
    public boolean Triominos_barre_possible() {
        // Parcours toutes les cases de la grille pour vérifier s'il est possible de placer un Triomino en forme de barre.
        for (int i = 0; i < 10; i++) { // 10 lignes
            for (int j = 0; j < 12; j++) { // 12 colonnes
                if (Est_libre(i, j)) {
                    if (Est_libre(i, j+1)) {
                        if (Est_libre(i, j+2)) {
                            return true;
                        }
                    }
                    if (Est_libre(i+1, j)) {
                        if (Est_libre(i+2, j)) {
                            return true;
                        }
                    }
                }
            }
```

Figure 3: Fonction Domino Possible

```java
public void Game_Basic() {
    Grille grille = new Grille();
    Joueur joueur = new Joueur();
    Scanner sc = new Scanner(System.in);
    JoueurIA ordinateur = new JoueurIA();
    int answer;
    Piece[] dominos = new Piece[1];
    Piece[] triominos = new Piece[2];
    Piece[] tetrominos = new Piece[7];
    //dominos
    dominos[0] = new Piece(2, "dominos", "O", 0, 0);
    dominos[0].dominos();
    dominos[0].setValeur("0");
    //triominos
    triominos[0] = new Piece(3, "triominos", "O", 0, 0);
    triominos[0].triominos_barre();
    triominos[0].setValeur("0");
    triominos[1] = new Piece(2, "triominos", "O", 0, 0);
    triominos[1].triominos_L();
    triominos[1].setValeur("0");
    //tetrominos
    tetrominos[0] = new Piece(3, "tetrominos", "O", 0, 0);
    tetrominos[0].Tetrominos_T();
    tetrominos[0].setValeur("0");
    tetrominos[1] = new Piece(3, "tetrominos", "O", 0, 0);
    tetrominos[1].Tetrominos_S();
    tetrominos[1].setValeur("0");
    tetrominos[2] = new Piece(3, "tetrominos", "O", 0, 0);
    tetrominos[2].Tetrominos_L();
    tetrominos[2].setValeur("0");
    tetrominos[3] = new Piece(2, "tetrominos", "O", 0, 0);
    tetrominos[3].Tetrominos_Carre();
    tetrominos[3].setValeur("0");
    tetrominos[4] = new Piece(3, "tetrominos", "O", 0, 0);
    tetrominos[4].Tetrominos_L_inverse();
    tetrominos[4].setValeur("0");
    tetrominos[5] = new Piece(3, "tetrominos", "O", 0, 0);
    tetrominos[5].Tetrominos_S_inverse();
    tetrominos[5].setValeur("0");
    tetrominos[6] = new Piece(4, "tetrominos", "O", 0, 0);
```

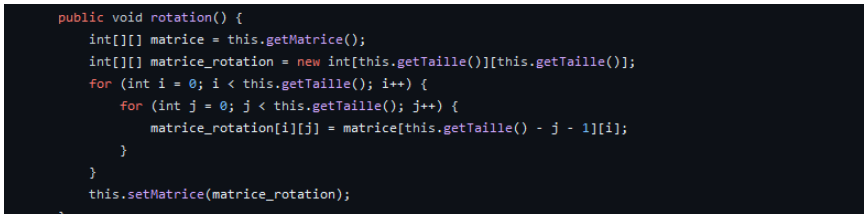
Figure 4: Fonction Initialisation d'une partie

```java
public int Saisir_Piece(Grille grille, Piece [] pieces){
    int answer;
    Scanner sc = new Scanner(System.in);
    do {
        System.out.println("Veuillez saisir la piece que vous voulez placer : ");
        System.out.println("1 : Domino");
        System.out.println("2 : Triomino barre");
        System.out.println("3 : Triomino L");
        System.out.println("4 : Tetromino T");
        System.out.println("5 : Tetromino S");
        System.out.println("6 : Tetromino L");
        System.out.println("7 : Tetromino Carre");
        System.out.println("8 : Tetromino L inverse");
        System.out.println("9 : Tetromino S inverse");
        System.out.println("10 : tetromino barre");
        System.out.println("11 : quittez le jeu");
        answer = sc.nextInt();
    }while(answer < 1 || answer > 11);
    return answer;
}

public int Saisir_Rotation(){
    int answer;
    Scanner sc = new Scanner(System.in);
    do {
        System.out.println("Veuillez saisir la rotation que vous voulez faire : ");
        System.out.println("0 : 0°");
        System.out.println("1 : 90°");
        System.out.println("2 : 180°");
        System.out.println("3 : 270°");
        System.out.println("4 : quittez le jeu");
        answer = sc.nextInt();
    }while(answer < 0 || answer > 4);
    return answer;
}
```

Figure 5: Classe Joueur

```
    public boolean Win_Joueur(Joueur joueur,Grille grille) {
        //pour detecter si il y a un gangant il faut que le joueur adverse ne puisse plus placer aucune pieces
        //on va donc regarder si les deux joueurs ne peuvent plus placer de pieces
        // Vérification si les dominos peuvent être placés
        if (grille.Dominos_possible() == false) {
            return true;
        }
        else {
            // Vérification si les triominos peuvent être placés
            if (grille.Triominos_barre_possible() == false && grille.Triominos_L_possible() == false) {
                return true;
            }
            else {
            // Vérification si les tetrominos peuvent être placés
                if (grille.Tetrominos_Carre_possible() == false && grille.Tetrominos_L_possible() == false
                        && grille.Tetrominos_S_possible() == false && grille.Tetrominos_S_inverse_possible() == false
                        && grille.Tetrominos_T_possible() == false && grille.Tetrominos_L_inverse_possible() == false
                        && grille.Tetrominos_barre_possible() == false) {
                    return true;
                }
                else {
                    return false;
                }
            }

        }
    }
}
```

Figure 6: Fonction pour savoir si une partie est gagnée



```
    public void rotation() {
        int[][] matrice = this.getMatrice();
        int[][] matrice_rotation = new int[this.getTaille()][this.getTaille()];
        for (int i = 0; i < this.getTaille(); i++) {
            for (int j = 0; j < this.getTaille(); j++) {
                matrice_rotation[i][j] = matrice[this.getTaille() - j - 1][i];
            }
        }
        this.setMatrice(matrice_rotation);
```

Figure 7: Fonction pour faire une rotation miroir

```java
public void dominos() {
    this.setTaille(2);
    int[][] matrice = new int[this.getTaille()][this.getTaille()];
    // 0 1 0
    // 0 1 0
    matrice[0][0] = 1;
    matrice[1][0] = 1;
    this.setMatrice(matrice);
}
/**
Cette méthode permet de créer une pièce en forme de "L" qui est représentée par une matrice de taille 4x4 contenant des 1.
La matrice est ensuite stockée dans l'attribut "matrice" de l'objet TetrisPiece courant et la taille est définie à 4.
*/
public void Tetrominos_L() {
    this.setTaille(4);
    int[][] matrice = new int[this.getTaille()][this.getTaille()];
    // 0 0 1 0
    // 0 0 1 0
    // 0 0 1 1
    matrice[0][0] = 1;
    matrice[1][0] = 1;
    matrice[2][0] = 1;
    matrice[2][1] = 1;
    this.setMatrice(matrice);
}
/**
Cette méthode permet de créer une pièce en forme de "L" inversé qui est représentée par une matrice de taille 4x4 contenant des 1.
La matrice est ensuite stockée dans l'attribut "matrice" de l'objet TetrisPiece courant et la taille est définie à 4.
*/
public void Tetrominos_L_inverse() {
    this.setTaille(4);
    int[][] matrice = new int[this.getTaille()][this.getTaille()];
    // 0 0 1 0
    // 0 0 1 0
    // 0 1 1 0
    matrice[0][2] = 1;
```

Figure 8: Fonction pour créer une pièce et gérer sa rotation