Département de Génie Informatique Cycle d’ingénieur Logiciels & Systèmes intelligents

METHODOLOGIE DE L’INTELLIGENCE ARTIFICIELLE

Domineering

Réalisé par:
KAISSI Houda
Essalhi Sara

I.But:

Développementd’uneapplicationJava pour lejeu" Domineering" enimplémentant l’algorithmealphaBeta.

II.Introduction

Le jeu de Domineering est un jeu de société abstrait qui se joue à deux joueurs sur une grille rectangulaire. Il appartient à la famille des jeux combinatoires purs, ce qui signifie qu'il se déroule en alternant les tours entre les joueurs, et le résultat dépend uniquement des choix stratégiques des joueurs, sans élément de hasard.

La particularité du jeu de domineering réside dans sa simplicité apparente, mais qui cache des défis tactiques intéressants. La grille de jeu est divisée en cellules rectangulaires, et chaque joueur place alternativement des dominos (pièces de deux cellules adjacentes) de manière à dominer les cases de la grille. L'objectif est de forcer l'adversaire à ne pas pouvoir jouer, soit en bloquant complètement son accès à la grille, soit en contrôlant de manière stratégique les espaces disponibles.

III.Code

```java
boolean drawnPosition(DomineeringPosition p) {
    if (DEBUG) System.out.println("drawnPosition(" + p + ")");
    boolean ret = true;
    for (int i = 0; i < 64; i++) {
        if (p.board[i] == DomineeringPosition.BLANK) {
            ret = false;
            break;
        }
    }
    if (DEBUG) System.out.println("     ret=" + ret);
    return ret;
}
```

La méthode `drawnPosition` en Java vérifie si une position de jeu spécifiée par un objet de la classe `DomineeringPosition` est entièrement remplie, c'est-à-dire si toutes les cases du tableau `board` sont différentes de la constante `DomineeringPosition.BLANK`. Elle retourne `true` si la position est complètement remplie, sinon elle retourne `false`. La méthode affiche des messages de débogage si la constante `DEBUG` est vraie.

```
boolean wonPosition(DomineeringPosition p, boolean player) {
    if (DEBUG) System.out.println("wonPosition(" + p + "," + player + ")")
    boolean ret = false;
    int Hnum = 0;
    int Pnum = 0;
    int Bnum = 0;

    for (int i = 0; i < 64; i++) {
        if (p.board[i] == -1)
            Pnum++;
        else if (p.board[i] == 1)
            Hnum++;
        else
            Bnum++;
    }

    if (player) {
        if (Pnum == 0)
            ret = true;
        else if (Bnum == 0 && Hnum > Pnum)
            ret = true;
    } else {
        if (Hnum == 0)
            ret = true;
```

La méthode `wonPosition` en Java vérifie si le joueur (humain ou programme) spécifié par le paramètre `player` a gagné dans une position de jeu donnée, représentée par un objet de la classe `DomineeringPosition`. Elle retourne `true` si le joueur a gagné, sinon elle retourne `false`. La méthode compte le nombre de cases occupées par le joueur humain (`Hnum`), par le programme (`Pnum`), et les cases vides (`Bnum`). En fonction du joueur spécifié, elle évalue les conditions de victoire. Si le joueur est humain, il gagne s'il n'y a aucune case occupée par le programme (`Pnum == 0`), ou s'il n'y a plus de cases vides et le nombre de cases occupées par le joueur humain est supérieur à celles occupées par le programme (`Bnum == 0 && Hnum > Pnum`). Si le joueur est le programme, la condition est similaire, mais inversée.

```
float positionEvaluation(DomineeringPosition p, boolean player) {
    int count = 0;
    int tour = -1;
    if (player)
        tour = 1;

    for (int i = 0; i < 64; i++) {
        if (p.board[i] == tour)
            count++;
    }

    float eval = 0;
    for (int i = 0; i < 64; i++) {
        if (p.board[i] == 1)
            eval += force[i];
        else if (p.board[i] == -1)
            eval -= force[i];
    }

    count = 65 - count;
    if (wonPosition(p, player)) {
        return eval + (1.0f / count);
    }
    if (wonPosition(p, !player)) {
```

```
void printPosition(DomineeringPosition p) {
    System.out.println(x: "Board position:");
    int count = 0;
    int k = 0;
    int h = 0;
    int pr = 0;

    for (int row = 0; row < 8; row++) {
        System.out.println();
        for (int col = 0; col < 8; col++) {
            switch (p.board[count]) {
                case DomineeringPosition.HUMAN:
                    System.out.print(s: "H");
                    h++;
                    break;
                case DomineeringPosition.PROGRAM:
                    System.out.print(s: "P");
                    pr++;
                    break;
                default:
                    System.out.print(s: "o");
                    break;
            }
            count++;
```

`positionEvaluation :`

- Évalue une position de jeu de domino en fonction du joueur spécifié.
- Attribue des valeurs aux cases occupées par chaque joueur à l'aide du tableau `force`.
- Ajuste la valeur d'évaluation en fonction du nombre total de cases restantes sur le plateau.
- Ajoute ou soustrait une petite quantité à la valeur si la position est gagnante pour le joueur spécifié.
- Renvoie la valeur d'évaluation finale.

2. `PrintPosition :`
3. Imprime la position actuelle du plateau de jeu de domino en utilisant des caractères pour représenter les joueurs humains, le programme et les cases vides.

- Compte le nombre de cases occupées par le joueur humain et le programme.
- Affiche ensuite le nombre de cases occupées par chaque joueur.

```
  List<DomineeringPosition> possibleMoves(DomineeringPosition p,
boolean player) {
        List<DomineeringPosition> ret = new ArrayList<>();
        int tour = -1;
        if (player)
            tour = 1;

        int count = 0, j = 0;

        for (int i = 0; i < 64; i++)
            if (p.board[i] != 0)
                count++;
        int[] occupiedCell = new int[64];
        j = 0;

        for (int i = 0; i < 64; i++) {
            if (p.board[i] != 0) {
                occupiedCell[j] = i;
                j++;
            }
        }

        List<Integer> voisins = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            int[] temp = new int[8];

            temp[0] = occupiedCell[i] - 9;
            temp[1] = occupiedCell[i] - 8;
            temp[7] = occupiedCell[i] + 9;
            temp[6] = occupiedCell[i] + 8;

            if (occupiedCell[i] % 8 != 7) {
                temp[4] = occupiedCell[i] + 1;
                temp[2] = occupiedCell[i] - 7;
            } else {
                temp[4] = -1;
                temp[2] = -1;
            }
            if (occupiedCell[i] % 8 != 0) {
                temp[3] = occupiedCell[i] - 1;
                temp[5] = occupiedCell[i] + 7;
            } else {
                temp[3] = -1;
                temp[5] = -1;
            }

            for (int k = 0; k < 8; k++) {
```

```
            for (int vois = 0; vois < voisins.size(); vois++) {
                if (temp[k] == voisins.get(vois)) {
                    temp[k] = -1;
                }
            }
            if (temp[k] >= 0 && temp[k] < 64)
                if (p.board[temp[k]] != 0)
                    temp[k] = -1;
        }

        for (int k = 0; k < 8; k++) {
            if (temp[k] >= 0 && temp[k] < 64)
                voisins.add(temp[k]);
        }
    }

    List<DomineeringPosition> rett = new ArrayList<>();

    for (int i = 0; i < voisins.size(); i++) {
        int actuel = voisins.get(i);
        int col = (actuel + 1) % 8;
        boolean add = false;

        DomineeringPosition pos2 = new DomineeringPosition();
        for (int l = 0; l < 64; l++) {
            pos2.board[l] = p.board[l];
        }

        for (int l = 0; l < 64; l++) {
            pos2.board[l] = p.board[l];
        }

        // ... (the rest of the code)

        if (add) {
            ret.add(pos2);
        }
    }

    if (rett.size() == 0)
        return ret;

    return
```

explication :

Le code Java implémente une méthode `possibleMoves` qui génère une liste de positions possibles après un mouvement dans le jeu de domino. Il commence par déterminer le joueur actuel à l'aide de la variable `tour` (1 pour le joueur, -1 pour le programme). Ensuite, il collecte les positions des cellules occupées dans le tableau `occupiedCell`. Pour chaque cellule occupée, il calcule les indices des voisins potentiels dans le tableau `temp`. Ensuite, il filtre les voisins déjà visités ou occupés en marquant les indices invalides. Les indices valides sont ajoutés à la liste `voisins`. Le code crée ensuite de nouvelles positions pour chaque voisin valide en copiant la position actuelle et en effectuant le mouvement. Ces nouvelles positions sont ajoutées à la liste `ret` si elles sont valides. Finalement, le code retourne la liste des nouvelles positions.

Pour la page Home :

```
private void initComponents() {

    contentPane = new javax.swing.JPanel();
    boardPane = new javax.swing.JPanel();
    exit = new javax.swing.JButton();
    depthCB = new javax.swing.JComboBox<>();
    deconnexionBtn = new javax.swing.JButton();
    positionUserCB = new javax.swing.JComboBox<>();
    playPositionBtn = new javax.swing.JButton();
    saveBtn = new javax.swing.JButton();
    hommeMachineCB = new javax.swing.JComboBox<>();
    playNewBtn = new javax.swing.JButton();
    noirTxt = new javax.swing.JTextField();
    blancTxt = new javax.swing.JTextField();
    blancLbl = new javax.swing.JLabel();
    noirLbl = new javax.swing.JLabel();
    timerLbl = new javax.swing.JLabel();
    second = new javax.swing.JLabel();
    millisecond = new javax.swing.JLabel();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    resultValueLbl = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    resultValueLbl1 = new javax.swing.JLabel();
```

Domineering position :

```java
public DomineeringPosition() {
    // Initial positions
    board[27] = -1;
    board[28] = 1;
    board[35] = 1;
    board[36] = -1;

    // Fill empty positions with 0
    for (int i = 0; i < 64; i++) {
        if (board[i] != -1 && board[i] != 1)
            board[i] = 0;
    }
}

// Getters and Setters
public int[] getBoard() {
    return this.board;
}

public void setBoard(int[] board) {
    this.board = board;
}

// Check if the game is in a terminal state
```

La classe Java `DomineeringPosition` est conçue pour représenter l'état d'une partie de domineering. Elle comprend un constructeur pour initialiser le plateau de jeu avec des positions spécifiques indiquant l'état initial. Les méthodes getters et setters permettent un accès externe et une modification de l'état du jeu. La classe inclut une méthode, `etat()`, pour vérifier si le jeu est dans un état terminal. La méthode `toString()` fournit une représentation sous forme de chaîne de caractères de l'objet, affichant le contenu du plateau de jeu. Dans l'ensemble, cette classe sert de modèle pour gérer l'état d'une partie de domineering.

L’authentification et l’inscription de l’utilisateur

```
//------------ Constructeur ----------
public Authentification() throws SQLException, ClassNotFoundException {
    initComponents();
    this.errorLbl.setVisible(aFlag: false);

    //--------- Se connecter à la BDD
    connection = ConnectionManager.getConnection();


    //----- Centrer la fenetre
    Dimension screenSize,frameSize;
    int x,y;
    screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frameSize=getSize();
    x=(screenSize.width-frameSize.width)/2;
    y=(screenSize.height-frameSize.height)/2;
    setLocation(x, y);

    //------ mettre la fenetre non Resizable
    this.setResizable(resizable: false);

    //------ Icon de frame
    Image icon = Toolkit.getDefaultToolkit().getImage(url: Authentificatio
    this.setIconImage(image: icon);
```

- Le code établit une connexion à une base de données.

2. **Interface graphique :**

- L'interface graphique comprend des champs pour l'email et le mot de passe, des boutons pour la connexion, l'inscription et la sortie, ainsi que des étiquettes pour afficher des messages.

3. **Fonctionnalités d'authentification :**

- L'utilisateur peut entrer son email et son mot de passe.
- En appuyant sur le bouton "Se connecter", le programme vérifie ces informations dans la base de données.
- Si les informations sont correctes, l'utilisateur est redirigé vers une nouvelle fenêtre (Home).
- En cas d'erreur d'authentification, un message d'erreur est affiché.

4. **Autres fonctionnalités :**

- L'interface graphique est centrée sur l'écran et non redimensionnable.

- Un bouton "S'inscrire" permet de passer à la page d'inscription.
- Un bouton "Quitter" ferme la fenêtre d'authentification.

5. **Gestion des événements :**

- Le code réagit aux actions de l'utilisateur, comme l'appui sur les boutons "Se connecter", "S'inscrire" et "Quitter".

6. **Chiffrement du mot de passe :**

- Le mot de passe entré par l'utilisateur est chiffré avant d'être comparé à celui stocké dans la base de données.

```
ublic class Inscription extends javax.swing.JFrame {

    /**********************************************************************************
     * Les élèment de connection à  la base de données ***********************
     **********************************************************************************

    Connection connection = null;
    ResultSet rs = null;
    ResultSet rs2 = null;
    PreparedStatement ps = null;
    EncryptPassword encryptPwd;


    //------------ Cosnstructeur ----------
    public Inscription() throws SQLException, ClassNotFoundException {
        initComponents();
        this.errorLbl.setVisible(aFlag: false);


        //--------- Se connecter à la BDD
        connection = ConnectionManager.getConnection();


        //      Centrer la fenetre
```

Le code établit une connexion à une base de données.

2. **Interface graphique :**

    - L'interface graphique comprend des champs pour le nom, l'email, et le mot de passe, des boutons pour l'inscription, la connexion et la sortie, ainsi que des étiquettes pour afficher des messages.

3. **Fonctionnalités d'inscription :**

    - L'utilisateur peut entrer son nom, son email et son mot de passe.
    - En appuyant sur le bouton "S'inscrire", le programme vérifie si le compte existe déjà.
    - Si le compte n'existe pas, il est créé dans la base de données.
    - En cas de succès, l'utilisateur est redirigé vers la page d'authentification avec les champs pré-remplis.

4. **Fonctionnalités de connexion :**

    - Le bouton "Se connecter" permet à l'utilisateur de revenir à la page d'authentification.

5. **Autres fonctionnalités :**

    - L'interface graphique est centrée sur l'écran et non redimensionnable.
    - Un bouton "Quitter" ferme la fenêtre d'inscription.

6. **Gestion des événements :**

    - Le code réagit aux actions de l'utilisateur, comme l'appui sur les boutons "S'inscrire", "Se connecter" et "Quitter".

7. **Chiffrement du mot de passe :**

    - Le mot de passe entré par l'utilisateur est chiffré avant d'être stocké dans la base de données.

VII Conclusion

Cette application est basée sur l’algorithme MinMax et la méthode d’élagage AlphaBeta, qui constituent la base du moteur d’intelligence artificielle de tous les jeux de réflexion tel que Domineering. Le principe de recherche d’un coup suivant ces méthodes repose sur le développement d’un arbre de jeu dont les feuilles ont une note obtenue par l’utilisation d’une fonction dévaluation