Connect Four Game

Présenté par : ADBIB Ilham

Encadré par :M.EL HAJJAMY Oussama

Plan

1

Présentation de jeu

2

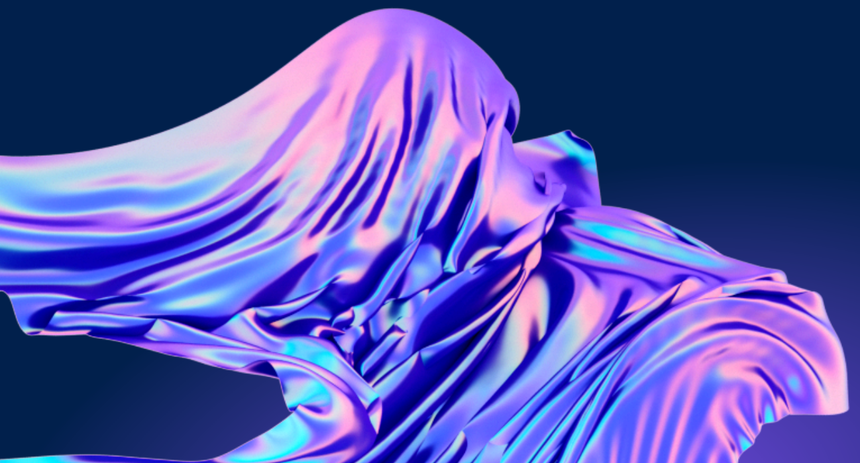
Présentation des design patterns utilisés

3

Capture d’application

Présentation de jeu

Connect Four est un jeu de stratégie joué à deux. L'objectif est de former une ligne continue de quatre jetons de sa propre couleur, horizontalement, verticalement ou en diagonale.



Architecture MVC

Interagit avec la vue

View

Envoi d'interaction

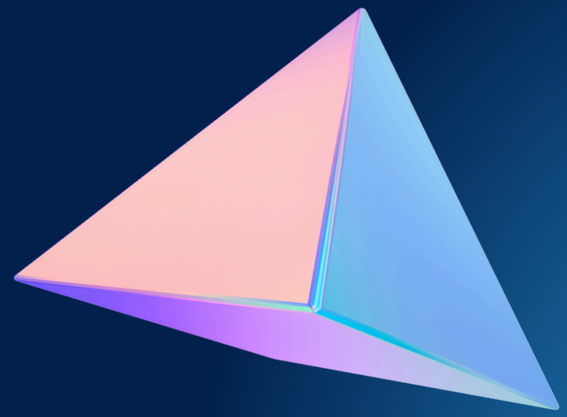
Mettre à jour

Controller

Mettre à jour

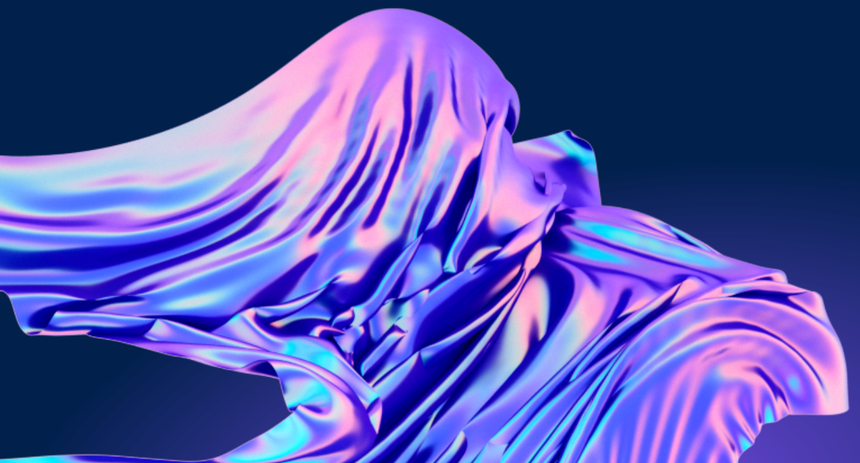
Notification

Modèle

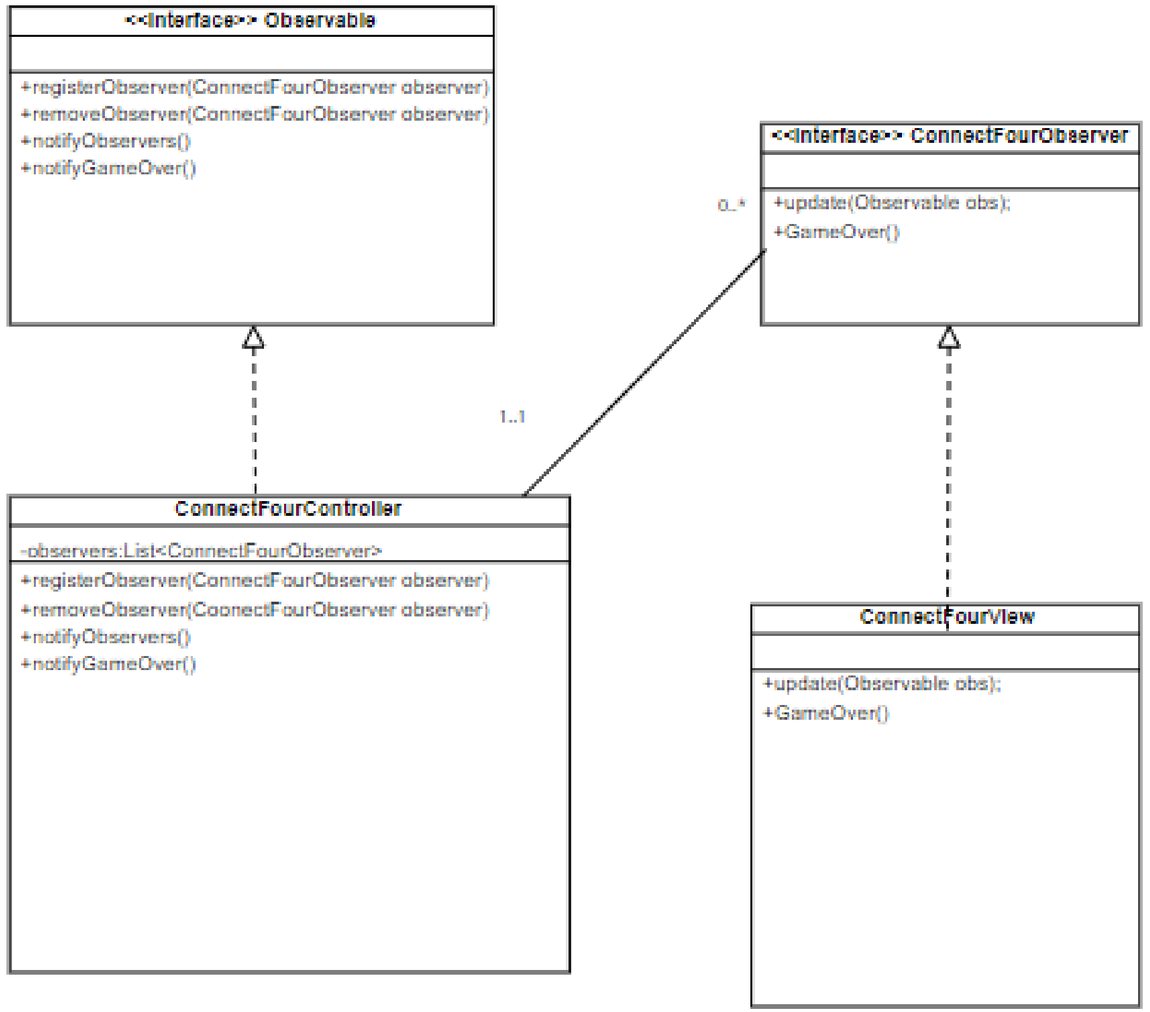


2.1 Observer pattern

Observer permet de synchroniser l'interface utilisateur avec les changements d'état du jeu en temps réel. Lorsqu'un événement important se produit, comme une mise à jour de Board ou la fin du jeu, les observateurs sont notifiés pour mettre à jour la vue,



2.1 Observer pattern



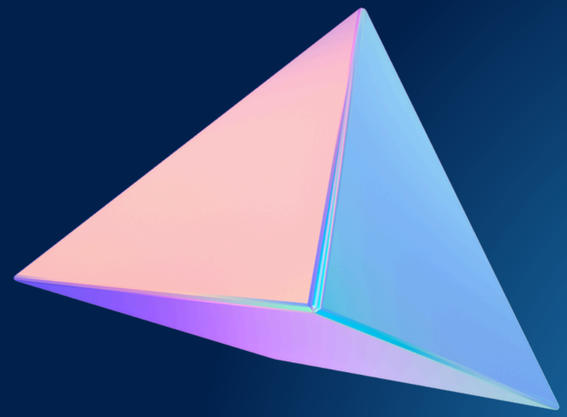
2.1 Observer pattern

```java
package ilham.adbib.connectfour.model;

import ilham.adbib.connectfour.view.ConnectFourObserver;

public interface Observable {  4 usages  1 implementation
    void registerObserver(ConnectFourObserver observer);
    void removeObserver(ConnectFourObserver observer);//
    void notifyObservers(); // Notify all observers  2 usag
    void notifyGameOver(); // Notify observers that the g
}
```

```java
package ilham.adbib.connectfour.view;

import ilham.adbib.connectfour.model.Observable;

public interface ConnectFourObserver {  13 usages  1 implementation
    void update(Observable obs);  1 usage  1 implementation
    void gameOver();  1 usage  1 implementation
}
```

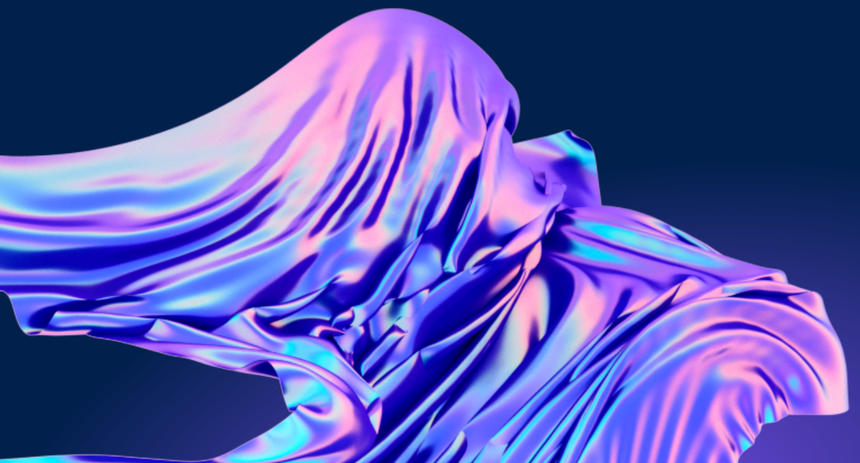
```java
public class ConnectFourController implements Observable {  17 usages

    private List<ConnectFourObserver> observers;// list to store all observers that are monit
    private ConnectFourView view;// reference to the view component that provides visual repr
    private ConnectFourModel model;// state and logic of the game.  19 usages
    public void run(ConnectFourModel model, ConnectFourView view) {  1 usage
        this.model = model;
        this.view = view;
        this.updateModel(this.view.getBoard(), this.view.getPlayer1(), this.view.getPlayer2());
        this.switchPlayer();//switch to the first player
    }
```

```java
public class ConnectFourView implements ConnectFourObserver {

    private Board board;  12 usages
    private Player player1;  6 usages
    private Player player2;  7 usages
    private ConnectFourController controller;//interaction w ith
    private static final String defaultPlayer = "human";  2 usages
    private JFrame frameMain;  14 usages
    private JRadioButton player1_op1;  9 usages
    private JRadioButton player1_op2;  8 usages
    private JRadioButton player2_op1;  9 usages
    private JRadioButton player2_op2;  8 usages
    private JButton start;  6 usages
```

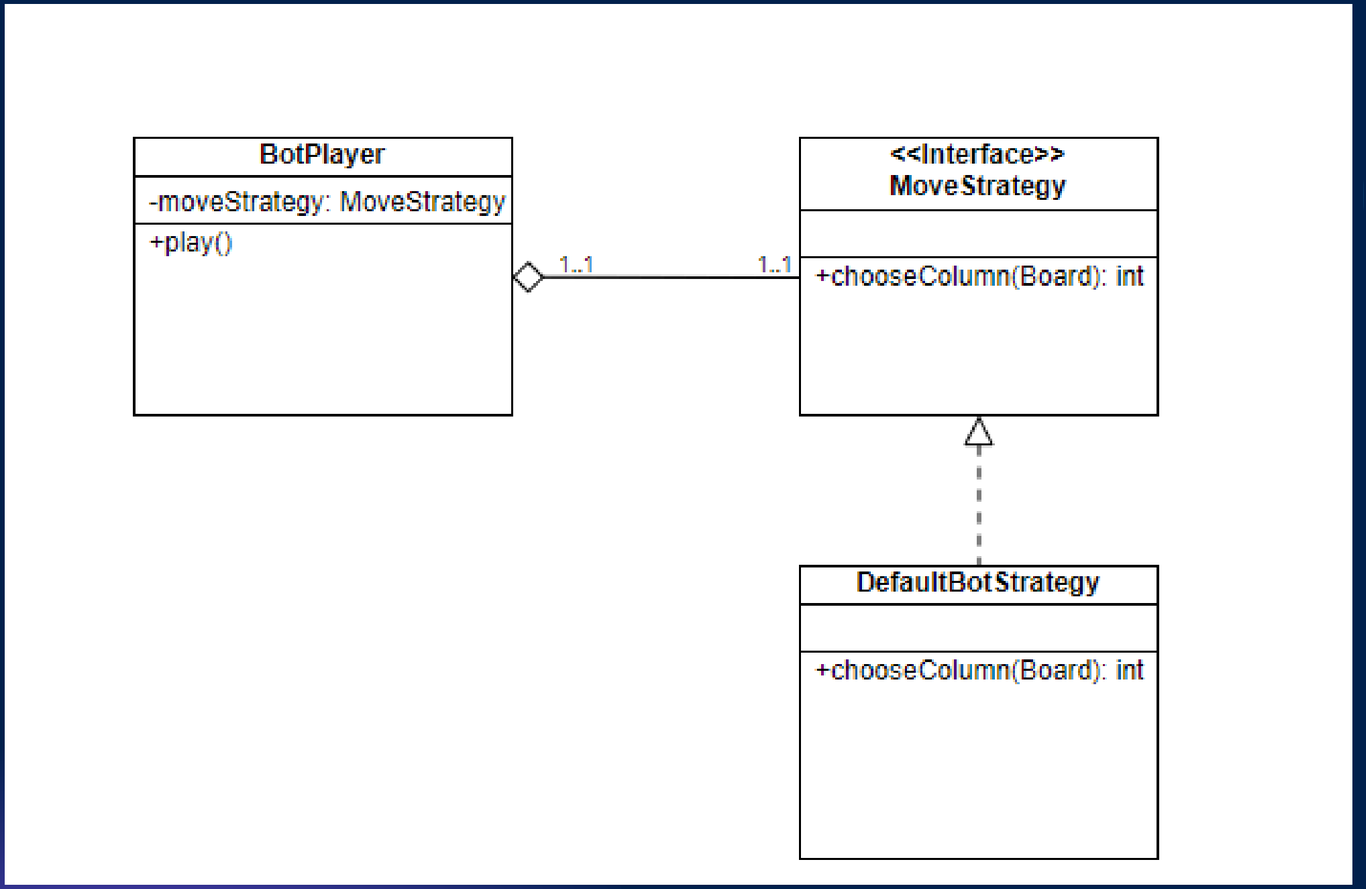


2.2 Strategy pattern

Le pattern Strategy permet au joueur Bot de sélectionner ses coups de manière flexible pour décider de ses mouvements, en s'adaptant au plateau de jeu



2.2 Strategy pattern



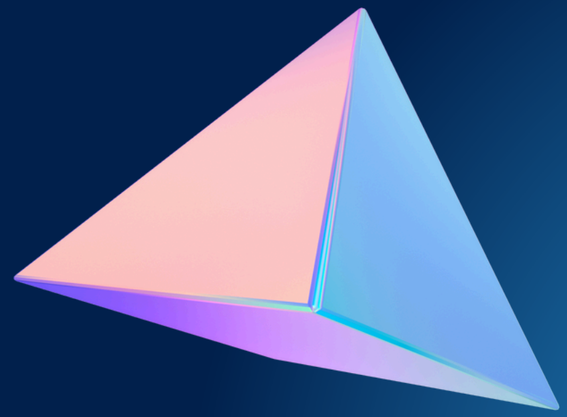
2.2 Strategy pattern

```java
public class BotPlayer implements Player {  9 usages
  private Board board;  3 usages
  private MoveStrategy moveStrategy;  2 usages
  public BotPlayer(Board board, MoveStrategy moveStrategy) {
    this.board = board;
    this.moveStrategy = moveStrategy;
  }
  @Override  3 usages
  public void play() {
    int column = moveStrategy.chooseColumn(board);
    board.move(column);
  }
```

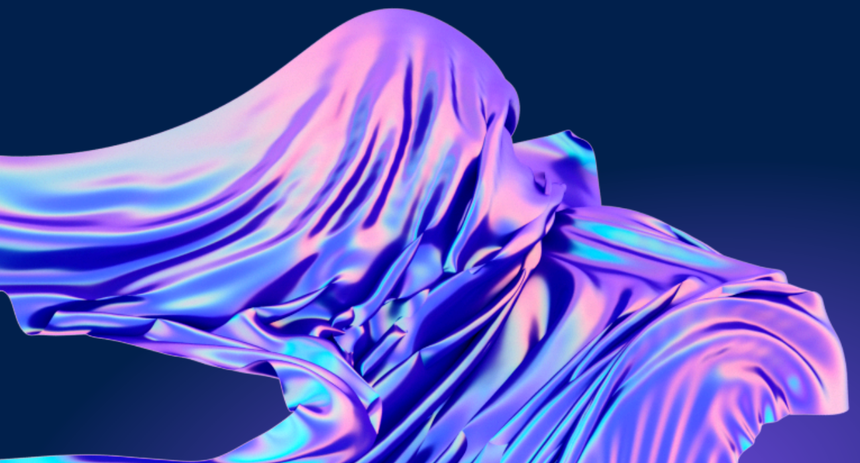
```java
package ilham.adbib.connectfour.model;

public interface MoveStrategy {  4 usages  1
    int chooseColumn(Board board);  1 usage
}
```

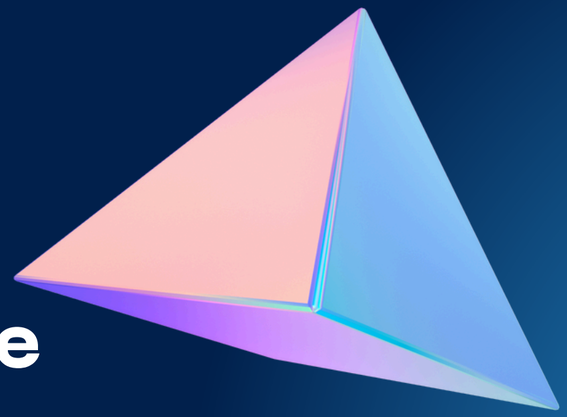
```java
public class DefaultBotStrategy implements MoveStrategy {
    @Override  1 usage
    public int chooseColumn(Board board) {
        int possibleNextCol = checkSelf(board);
        if (possibleNextCol != -1) return possibleNextCol;

        possibleNextCol = checkOpponent(board);
        if (possibleNextCol != -1) return possibleNextCol;

        Random randomGenerator = new Random();
```

2.3 Builder pattern



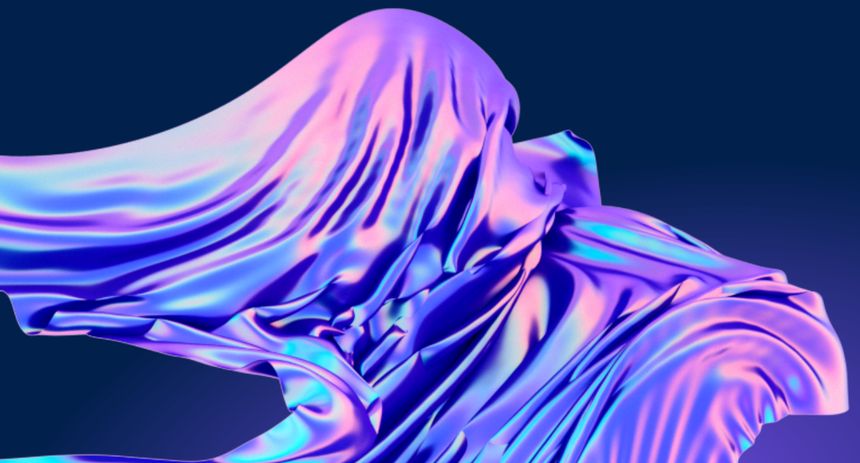
Builder est utilisé pour construire des objets complexes étape par étape. Il permet de séparer la construction d'un objet complexe de sa représentation, de sorte que le même processus de construction puisse créer différentes représentations



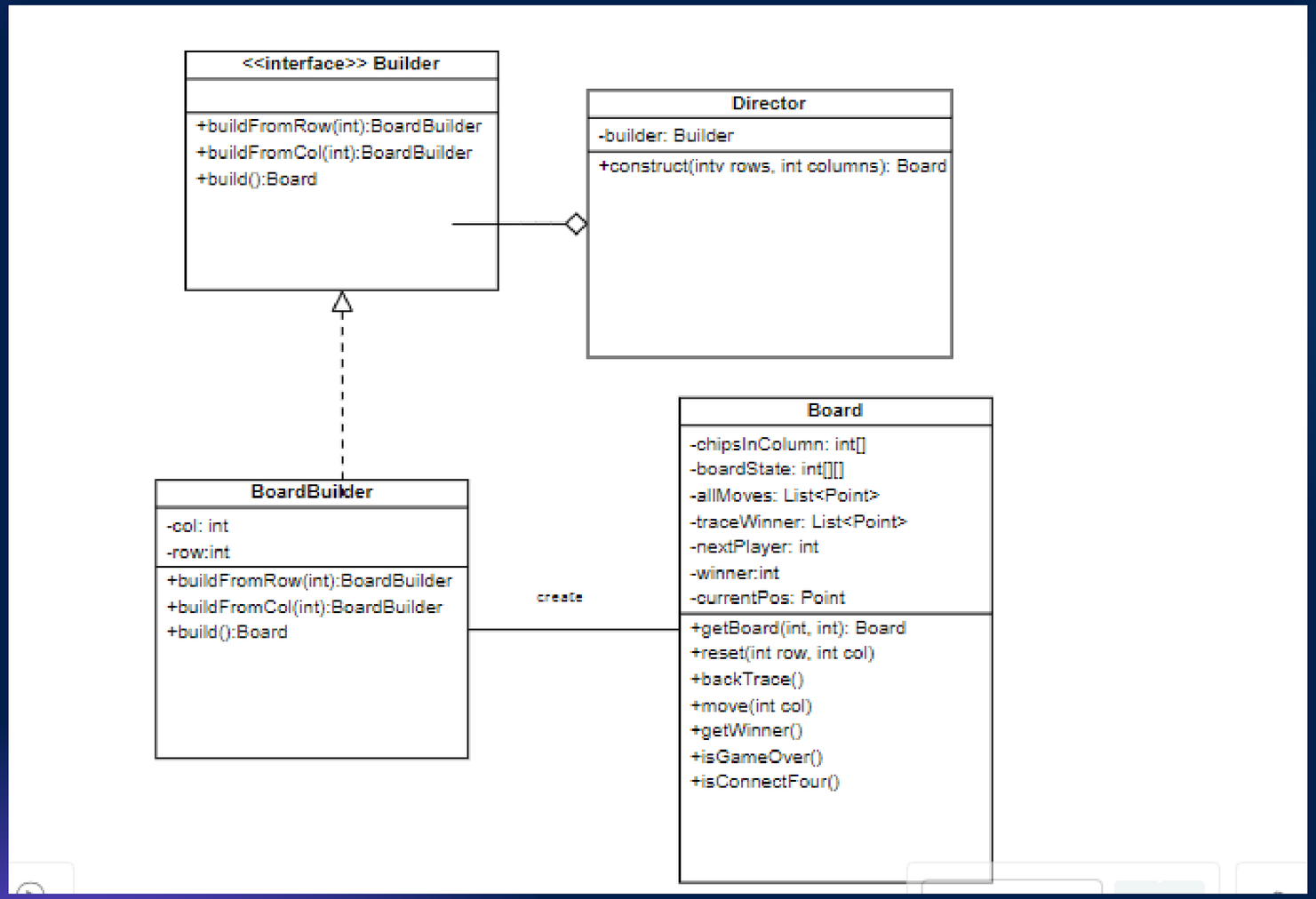


2.3 Builder pattern dans le jeu

Builder crée concrètement un Board configuré selon des spécifications définies, facilitant la gestion des différentes configurations(rangées et colonnes) et assurant que le plateau est correctement initialisé avant son utilisation dans le jeu.



2.3 Builder pattern



2.3 Builder pattern

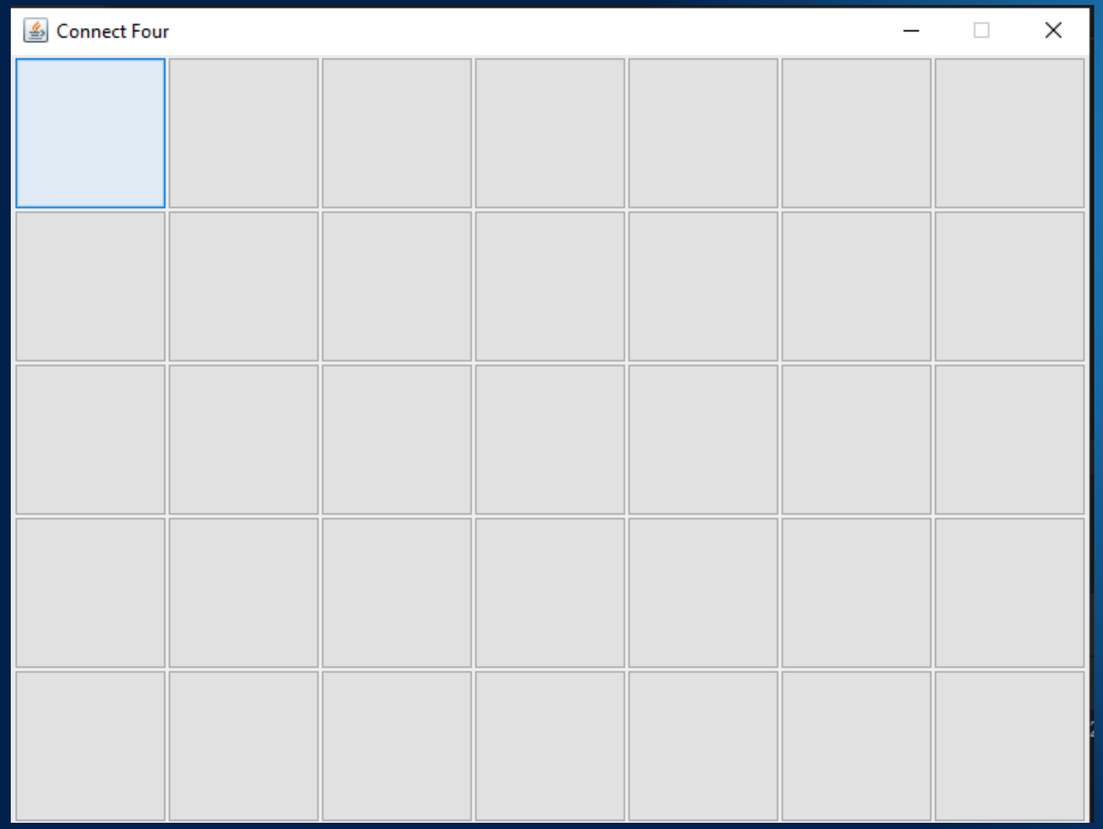
```java
package ilham.adbib.connectfour.model;

public interface Builder {  1 usage  1 implementa
    BoardBuilder buildFromRow(int rows);  3 u
    BoardBuilder buildFromCol(int columns);
    Board build();  3 usages  1 implementation
}
```

```java
public class Director {  3 usages

        private BoardBuilder builder;  2 usages

        public Director(BoardBuilder builder) { this.builder = builder; }

        public Board construct(int rows, int columns) {  1 usage
            return builder.buildFromRow(rows).buildFromCol(columns).build();
        }
    }
```

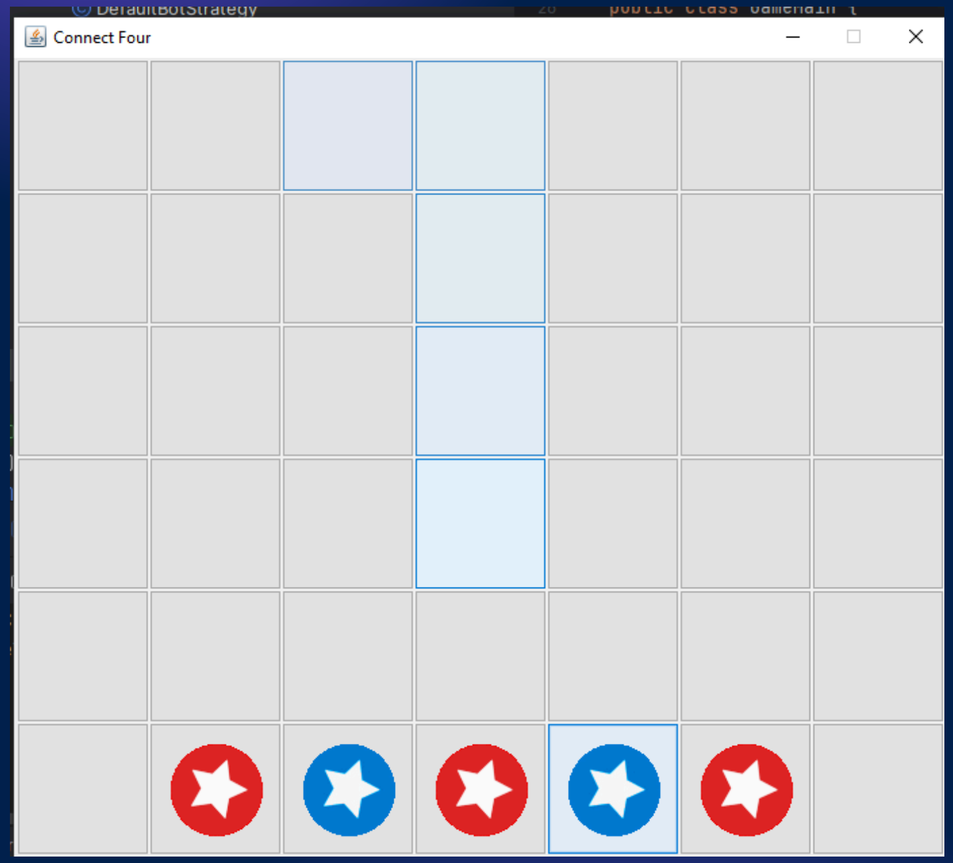
```java
public class BoardBuilder implements Builder{  11 usages
  private int row;  2 usages
  private int col;  2 usages
  @Override  3 usages
  /*public BoardBuilder buildFromRow(int row): This method set
  public BoardBuilder buildFromRow(int row) {
    this.row = row;
    return this;
  }
```

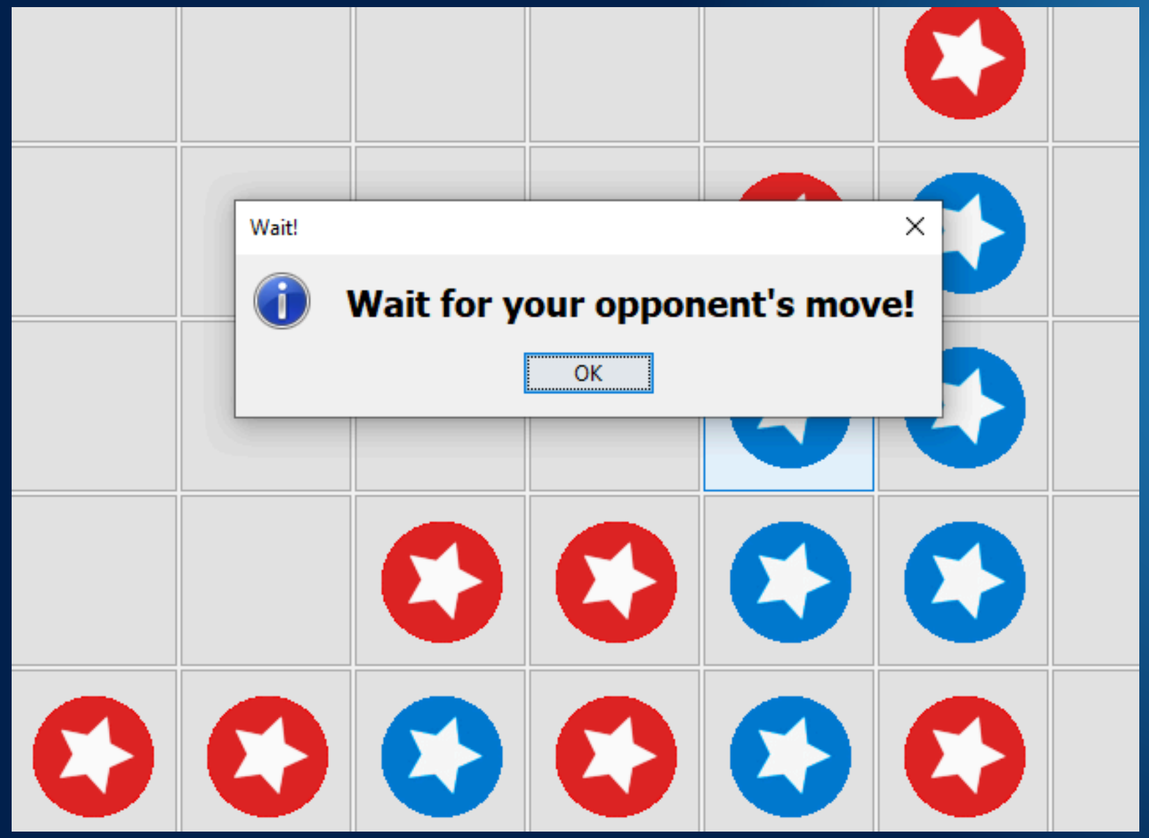
```java
public class Board{

  private int[] chipsInColumn;  6 usages
  private int[][] boardState;  7 usages
  private List<Point> allMoves;  9 usages
  private List<Point> traceWinner;  14 usages
  private int nextPlayer;  5 usages
  private int winner;  4 usages
  private static Board board = new Board();  5 usages
```

Captures d'écran d'application

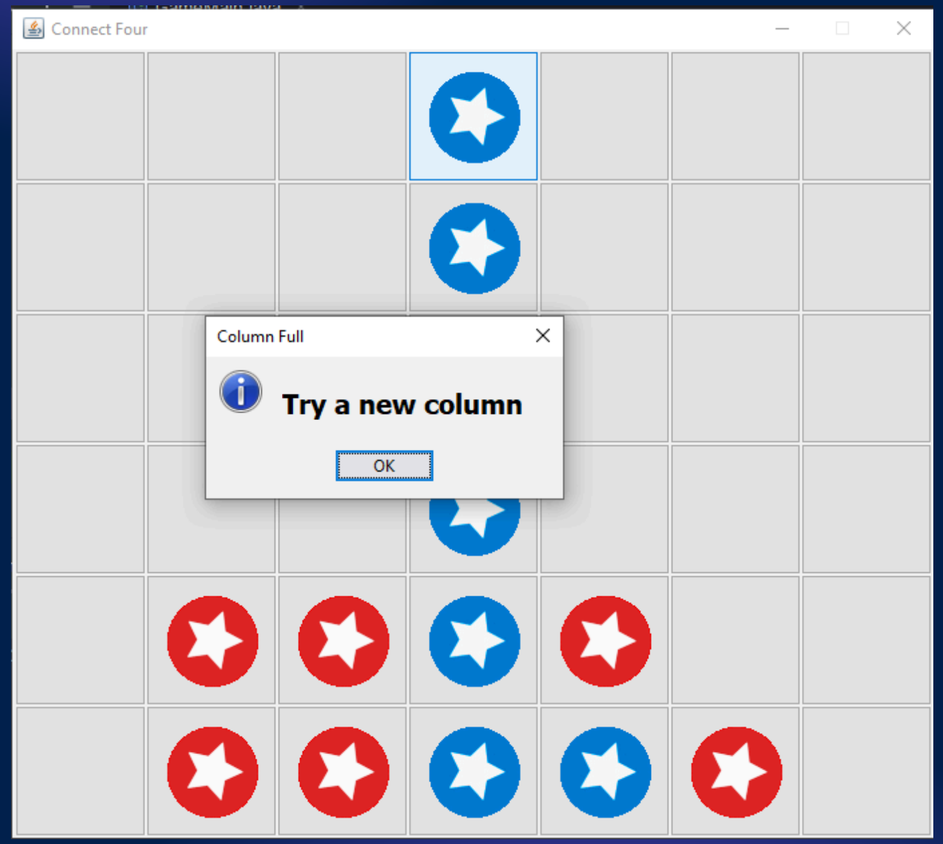


Captures d'écran d'application





Captures d'écran d'application



Merci Pour Votre Attention