Relazione progetto programmazione avanzata

4Game

Obbiettivi

All'interno del progetto dovranno essere sviluppate le classi/interfacce che consentano:

1. La gestione della partita;
2. La definizione di giocatori personalizzati (per definire avversari “automatici”);
3. La gestione dell'interazione dell'utente.

L'architettura delle classi, così come il ruolo delle stesse, dovranno essere descritte nella relazione.

Il mio progetto per l’esame di programmazione avanzata si basa sullo sviluppo del gioco Forza-4 presente tra le possibile scelte di sviluppo specificate

Classi

Le classi che ho sviluppato sono rispettivamente Player,Matrix,Game

Le ereditarietà sono le seguenti: Matrix estende la classe Player

```java
import java.io.*;

public class Player {

    private String namePlayer;
    private boolean turn;
    private int value;
    private int countWin = 0;

    public void setName(String NamePlayer) {
        this.setNamePlayer(NamePlayer);
    }

    public void setTurn(boolean Turn) {
        this.turn = Turn;
    }

    public boolean getTurn() {
        return turn;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }

    public void incCountWin() {
        this.countWin++;
    }

    public int getCountWin() {
        return countWin;
    }
```

```
public int getCountWin() {
    return countWin;
}

public String getNamePlayer() {
    return namePlayer;
}

public void setNamePlayer(String namePlayer) {
    this.namePlayer = namePlayer;
}
```

La classe Player serve per la definizione dei giocatori, all'interno ogni Player possiede

- Un nome definito come String
- Un turno definito come boolean
- Un valore definito come integer
- Un countWin definito come integer

All'interno troviamo i metodi di accesso, utili per impostare e leggere i valori che andremo ad implementare all'interno del main.

I metodi set sono tutti di tipo void, in quanto non presentano valore di ritorno e servono appunto per settare il nome, il turno e il valore associato al Player.

I metodi get invece servono per ottenere il risultato che andremo ad associare nel main quando bisognerà ottenere un valore dalle variabili.

Il metodo incCountWin serve per incrementare il valore della variabile che sarà utilizzata poi per determinare il vincitore della partita.

Classe Matrix

```java
public class Matrix extends Player {

    private int rows;
    private int cols;
    public int[][] Mat;

    Matrix(int r, int c) {
        rows = r;
        cols = c;
        Mat = new int[rows][cols];
    }

    public boolean InputMatrix(int Position, Player p) {

        int possibleRow = rows - 1;
        int selectedColumn = Position - 1;
        // if all column is full
        if (Mat[0][selectedColumn] != 0) {
            return false;
        } else {
            while (true) {
                if (Mat[possibleRow][selectedColumn] == 0) {
                    Mat[possibleRow][selectedColumn] = p.getValue();
                    break;
                } else {
                    possibleRow--;
                }
            }
        }

        print_Matrix();
        return true;
    }



   public int checkWin() {
        final int HEIGHT = rows;
        final int WIDTH = cols;
        final int EMPTY_SLOT = 0;
        for (int r = 0; r < HEIGHT; r++) { // iterate rows, bottom to top
            for (int c = 0; c < WIDTH; c++) { // iterate columns, left to right
                int player = Mat[r][c];
                if (player == EMPTY_SLOT)
                    continue; // don't check empty slots

                if (c + 3 < WIDTH && player == Mat[r][c + 1] && // look right
                        player == Mat[r][c + 2] && player == Mat[r][c + 3])
                    return player;
                if (r + 3 < HEIGHT) {
                    if (player == Mat[r + 1][c] && // look up
                            player == Mat[r + 2][c] && player == Mat[r + 3][c])
                        return player;
                    if (c + 3 < WIDTH && player == Mat[r + 1][c + 1] && // look up & right
                            player == Mat[r + 2][c + 2] && player == Mat[r + 3][c + 3])
                        return player;
                    if (c - 3 >= 0 && player == Mat[r + 1][c - 1] && // look up & left
                            player == Mat[r + 2][c - 2] && player == Mat[r + 3][c - 3])
                        return player;
                }
            }
        }
        return EMPTY_SLOT; // no winner
    }

    public void print_Matrix() {
        System.out.println();
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 7; j++) {
                System.out.flush();
                System.out.print(Mat[i][j] + " ");
            }
            System.out.println();
```

La classe Matrix estende la classe Player e possiede:

- Un int rows (le righe della matrice) privato
- Un int cols (le colonne della matrice) privato
- Un int Mat[][] (la struttura della matrice) pubblica

Matrix accetta due integer come input ovvero r e c e dichiara la matrice come Mat[rows][cols]

All'interno del main sarà possibile definire la dimensione della matrice.

La classe possiede tre metodi quali

- InputMatrix
- checkWin
- print_Matrix

Il metodo inputMatrix è dichiarato come boolean e ha valore di ritorno true nel momento in cui l'inserimento all'interno della nostra matrice viene eseguito correttamente e non presenta errori di qualche tipo.

I parametri di ingresso sono un int Position per la posizione e un Player(il metodo costruttore si trova all'interno della classe Main)

La posizione deve essere inizializzata poiché il giocatore quando digita una della colonne dove vuole posizionare la pedina non si aspetta che la matrice si conta da 0 a 6 e non da 1,7 quindi inizialmente bisogna decrementare la scelta del giocatore di 1.

Dato che è un metodo ricorsivo il primo controllo che andremo a fare è sulla colonna, ovvero se la colonna selezionata dal giocatore è già piena e non è possibile aggiungerci pedine.

In caso contrario significa che è possibile aggiungere pedine così la matrice alla colonna selezionata e alla riga più bassa possibile sarà associato p.getValue() per capire se l'inserimento lo sta facendo il giocatore oppure il computer.

In caso in cui il valore non fosse 0 l'inserimento verrà effettuato sempre sulla stessa colonna ma con possibleRow decrementata ovvero salendo verso l'alto, segue poi il metodo print per stampare a video la matrice e il valore di ritorno true perché l'inserimento è andato a buon fine.

Il metodo printMatrix è molto semplice in quanto tramite un doppio ciclo for controlla tutte le posizioni della matrice e la stampa in modo ordinato, quel (+" ") serve appunto per rendere l'idea della matrice originale del gioco poiché senza di questo la matrice sarebbe stata scalata senza la possibilità di capire bene dove si sta posizionando e dove sono andate a finire le pedine.

Il metodo checkWin è di tipo integer e serve per effettuare un controllo ricorsivo sulle posizioni della matrice.

Il giocatore o il computer vincono la partita quando posizionano quattro pedine orizzontali verticali o addirittura oblique.

checkWin è definito come:

- final int HEIGHT=rows (altezza)
- final int WIDTH=cols (larghezza)
- final int EMPY_SLOT

la nostra altezza corrisponde alle righe in quanto le pedine vengono poste dalla casella più in basso fino a quelle più alte

Allo stesso tempo la nostra altezza corrisponde alle colonne, semplicemente secondo la logica dell'inserimento sono invertite.

Il doppio ciclo for serve per iterare le righe dal basso verso l'alto.

Per facilitare il controllo questo avviene a step, questi sono suddivisi in quattro parti ovvero:

- Un controllo verso l'alto
- Un controllo verso destra
- Un controllo dall'alto verso destra
- Un controllo dall'alto verso sinistra

Così facendo siamo sicuri che a ogni inserimento corrisponderà un controllo in tutte le direzioni e restituirà l'integer player in caso di compiuta osservanza.

Nel caso in cui la partita fosse patta il valore di ritorno sarà EMPY_SLOT

Il metodo isFull serve solo per il controllo servirà per l'inserimento all'interno della matrice delle pedine fino al suo completamento.

La classe Game

```
public class Game {

    public static void main(String[] args) throws IOException {
        Player a = new Player();
        Player b = new Player();
        a.setValue(1);
        b.setValue(2);
        a.setTurn(true);
        a.setTurn(false);
        int min = 1;
        int max = 7;
        int position;
        String input = null;
        int random;
        Scanner in = new Scanner(System.in);
        System.out.println("Write your name");
        a.setNamePlayer(in.next());
        System.out.println("Write your enemy's name");
        b.setNamePlayer(in.next());
```

La classe Game possiede all'interno il metodo main dal quali avvengono tutte le iterazioni con le variabili e i metodi che abbiamo già definito

Comprende il metodo costruttore per la definizione dei Player ovvero un Player a e un Player b che corrispondono rispettivamente al giocatore e al computer

Vengono settati i valori di inserimento (1,2) ovvero quei valori che saranno visibili a ogni inserimento nella matrice

Vengono poi settati i turni true per il giocatore e false per il computer, così facendo sarà sempre il giocatore a iniziare per primo, una possibile nuova implementazione si potrebbe fare tramite il Random settando così in maniera casuale il Player che inizia la partita.

Seguono poi due integer min e max settati rispettivamente a 1 e 7 i quali saranno utili al computer tramite un metodo random per settare le pedine all'interno della matrice in modo corretto evitando di superare il buffer della matrice.

L'int position servirà tramitenextInt() per leggere i numeri corrispondenti alle colonne dove si vuole inserire.

Un integer random che servirà tramite il metodo random e cambierà di valore ogni volta che sarà il computer ad inserire la pedina.

Inizialmente verrà chiesto tramite lo Scanner di digitare il nome del giocatore e il nome del computer contro cui si vuole giocare, questo ci sarà utile per rilegger poi all'interno dello Score.txt lo storico delle partite che si è giocato.

```
    do {
        Matrix x = new Matrix(6, 7);
        System.out.println("Enjoy this game\n");
        x.print_Matrix();
        do {
            do {
                System.out.println("Insert coil position 1 to 7");
                position = in.nextInt();
            } while (x.InputMatrix(position, a) == false);
            if (x.checkWin() == 1) {
                System.out.println("Ha vinto:" + a.getNamePlayer());
                a.incCountWin();
                break;
            }
            do {
                random = (int) (Math.random() * (max - min) + min);
            } while (x.InputMatrix(random, b) == false);
            if (x.checkWin() == 2) {
                System.out.println("Ha vinto: " + b.getNamePlayer());
                b.incCountWin();
                break;
            }
        } while (!x.isFull());

        updateScore(a, b);
    } while (playAgain());

}
```

Questo pezzo di codice rappresenta il core del nostro progetto.

Inizialmente viene definito il metodo costruttore per la matrice e viene inizializzata a 6 e 7, viene poi stampata a video la matrice.

Questo do-while si ripete fino a che il giocatore utilizza il metodo playAgain() ovvero decide di giocare nuove partite contro la macchina.

Segue poi un do-while che continuerà fino a che la matrice non sarà piena tramite il metodo isFull()

Il codice che si trova all'interno richiede al giocatore di inserire la pedina in una posizione va da 1 a 7 ovvero le colonne della matrice

Quando l'inserimento del giocatore va a buon termine segue il do-while riferito al computer, random viene settata con un numero che va da 1 a 7 in maniera casuale e viene inserito all'interno della matrice

Entrami i do-while sia per il giocatore che per il computer comprendono il metodo checkWin()

Se il metodo checkWin() riferito a x ovvero alla nostra matrice è uguale a 1 significa che il giocatore ha posizionato 4 pedine in maniera corretta per vincere e quindi viene stampato la fine della partita con il nome del giocatore che si è selezionato

Nel caso in cui invece checkWin() abbia valore 2 significa che questo è stato compiuto dalla macchina e quindi sarà stampato a video il nome della macchina che abbiamo associato.

Seguono poi gli incrementi di incCountWin per incrementare il numero delle vincite che saranno visibili all'interno dello storico Score.txt

```
private static boolean playAgain() {
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Play again? (Y/N): ");
    String replay = keyboard.nextLine();
    return replay.equalsIgnoreCase("Y");
}
```

```
private static void updateScore(Player a, Player b) throws IOException {
    BufferedWriter buffer = new BufferedWriter(new FileWriter("Score.txt", true));
    String output = "Human: " + a.getNamePlayer() + " vs " + "Computer: " + b.getNamePlayer() + " "
            + a.getCountWin() + "-" + b.getCountWin() + "\n";
    buffer.write(output);
    buffer.close();
}
```

playAgain è il metodo che consente di scegliere di rigiocare la partita.

Tramite uno scanner di stringa presenta come valore di ritorno replay.equalsIgnoreCase(Y), tramite questa funzionalità si può digitare una ipsilon minuscola o maiuscola in maniera indifferente.

Il metodo updateScore() ha come parametri i due Player e serve per eseguire l'upgrade dello storico delle partite.

Viene creato(se non è presente) un file Score.txt e vengono scritti nel file i nomi dei giocatori e tramite il metodo getCountWin() il relativo risultato delle partite.