Aufgabe 4:
Würfelglück

Team-ID: 00370

Team: Linus Schumann

Bearbeiter/-innen dieser Aufgabe:

Linus Schumann

15.11.2021

Inhaltsverzeichnis

1_Lösungsidee 2
1.1_Beachtete Regeln 2
2_Umsetzung 2
2.1_Allgemeines 2
2.2_Klasse „a4“ 3
2.2.1_calculate 3
2.2.2_simulateOneGame 3
2.2.4_printOutResults 3
2.3_Klasse „Game“ 3
2.3.1_move 3
2.3.2_killAndSetPosition 3
2.3.3_Andere Funktionen 4
2.4_Klasse „Player“ 4
2.4.1_set- /getPosition 4
2.4.2_Andere Funktionen 4
3_Beispiele 4
3.1_Würfel 0 4
3.2_Würfel 1 5
3.3_Würfel 2 5

3.4_Würfel 3 6
3.5_eigene Würfel 6
4_Quellcode 7
4.1_Klasse „a4“ 7
4.2_Klasse „Game“ 9
4.3_Klasse „Player“ 11

1_Lösungsidee

Die Grundsätzliche Idee zur Lösung des Problems ist es, ein vollwertiges Mensch-Ärgere-Dich-Nicht Spiel zu programmieren und dann alle Würfel einmal gegen alle anderen antreten zu lassen. Dabei wird besonders Rücksicht darauf genommen, dass nicht Würfel doppelt gegeneinander antreten und hiermit die Laufzeit des Programms enorm verbessert werden kann. Außerdem wird beachtet, dass Würfel ohne eine sechs automatisch verlieren, da es nicht möglich ist, ohne eine sechs auch nur einen Schritt vorwärtszugehen.

1.1_Beachtete Regeln

Folgende Regeln müssen, zusammensetzend aus den BWINF-Regeln und den offiziellen Regeln, beachtet werden:

- Es muss immer mit dem vordersten Spielstein gezogen werden (auch in den Zielfeldern)
- Wenn kein Stein bewegt werden kann, verfällt der Zug
- Die 2 Spieler starten gegenüber
- Bei einer sechs wird ein neuer Spielstein auf das Startfeld gesetzt
- Das Startfeld muss immer frei sein

Daraus ergibt sich außerdem, dass wenn es nicht mehr für die vorderste Figur möglich ist auf das letzte Feld zu gelangen, dass Spiel automatisch nicht mehr gewonnen werden kann. In diesem Fall wird nur der andere Spieler weiter simuliert und überprüft, ob dieser ans Ziel gelangt.

2_Umsetzung

2.1_Allgemeines

Das Programm wurde in Java geschrieben und mit der Java-JDK 17 kompiliert. Im Ordner „executables“ lässt sich die kompilierte .jar-Datei finden und mit der Batchdatei (Windows) oder dem Shell-Script (Linux und MacOS) ausführen. Außerdem lassen sich im Ordner „beispieldaten“ alle in dieser Dokumentation aufgeführten Beispiele wiedergefunden werden.

Im Ordner „source“ lassen sich folgende Java Klassen wiederfinden:

- Main: Die Klasse „Main“ ist die Hauptklasse sie übernimmt die Dateiauswahl und gibt den Dateinamen an die andere Klasse weiter
- a4: Die Klasse „a4“ (Aufgabe 4) übernimmt die Verarbeitung der Daten und die Ausgabe der Ergebnisse

- Game: In der Klasse „Game“ wird das Mensch-Ärgere-Dich-Nicht Spiel mit den dazu gehörigen Regeln umgesetzt.
- Player: Die Klasse „Player“ stellt einen Spieler der das Mensch-Ärgere-Dich-Nicht Spiel aus der Klasse „Game“ spielt da.

2.2_Klasse „a4“

Wie oben erläutert werden in der Klasse „a4“ die Daten verarbeitet, Spiel-Simulationen gestartet und die Ergebnisse ausgegeben. Dabei werden diese Aufgaben auf folgende Funktionen aufgeteilt.

2.2.1_calculate

Zuerst wird in der Funktion „calculate“ die Datei und die damit verbundenen Würfel ausgelesen und in einem zwei-dimensionalen Array gespeichert. Danach wird für jeden Würfel eine gewisse Anzahl an Spielen mit jedem anderen Würfel simuliert. Wie in der Lösungs-Idee beschrieben werden dabei Dopplungen vermieden. Hierbei wird die Anzahl an gewonnen und verlorenen Games gespeichert und das Ergebnis berechnet. Innerhalb eines Toleranzbereichs von 20% wird „unentschieden“ als Ergebnis gespeichert. Am Ende werden dann die Ergebnisse mit Hilfe der Funktion „printOutResults“, die in 2.2.4 näher erläutert wird, ausgegeben.

2.2.2_simulateOneGame

In dieser Funktion wird ein Spiel zwischen zwei Würfeln simuliert. Zuerst wird dabei eine neue Instanz der Klasse „Game“ erstellt und die Würfel übergeben. Danach wird das Spiel gestartet und für jeden Zug eine Zufallszahl aus dem vorhandenen Würfel abgerufen. Dieses Auswählen der Zufallszahl wird von der Funktion „getRandom“, die den Wert eines Würfels an einem zufälligen Index zurückgibt, übernommen.

2.2.4_printOutResults

In dieser Funktion werden die Ergebnisse ausgegeben. Dabei werden zuerst alle Würfel ausgegeben und danach eine Matrix mit den Ergebnissen.

2.3_Klasse „Game“

Wie oben erklärt, wird in dieser Klasse das Mensch-Ärgere-Dich-Nicht Spiel umgesetzt. Zuerst werden dabei im Instruktor zwei Instanzen der Klasse „Player“ erstellt und die Klasse bekommt die zwei Würfel übergeben. Die Hauptfunktion der Klasse ist die Methode „move“, außerdem gibt es noch die Funktion „killAndSetPosition“ und weitere Funktionen.

2.3.1_move

Die Funktion „move“ wird bei jedem Zug einmal aufgerufen. Zuerst wird dabei geprüft, ob das Startfeld frei ist. Wenn dieses nicht frei ist, wird, wenn möglich, das Feld geräumt. Wenn das Startfeld frei ist, wird überprüft, ob eine sechs gewürfelt wurde und wenn ja wird eine noch nicht verwendete Figur auf das Startfeld gesetzt. Wenn keine sechs gewürfelt wurde und das Startfeld frei ist oder das Startfeld belegt ist und nicht geräumt werden kann, wird ein „normaler Zug“ ausgeführt. Dabei wird zuerst die Figur, die am weitesten vorne steht, ermittelt und dann um die Augenzahl des Würfels nach vorne gesetzt.

2.3.2_killAndSetPosition

Die Funktion „killAndSetPosition“ überprüft zuerst, ob die Figur gesetzt werden darf und setzt diese, wenn möglich. Danach wird überprüft, ob eine gegnerische Figur sich auf dem neuen Feld befindet. Wenn ja wird diese Figur zurückgesetzt.

2.3.3_Andere Funktionen

Neben diesen Funktionen gibt es noch 3 weitere. Die erste Funktion hat den Namen „checkNoSolution“ und überprüft, ob ein Spieler nicht mehr ins Ziel kommen kann. Die zweite heißt „isFinished“ und gibt zurück, ob ein Spieler gewonnen hat oder beide Spieler keine Chance mehr haben zu gewinnen. Zuletzt gibt es noch die Funktion „whoHasWon“, die zurückgibt welcher Spieler gewonnen hat.

2.4_Klasse „Player“

Beim Erstellen einer neuen Instanz der Klasse „Game“, wird für jeden Spieler eine neue Instanz dieser Klasse erstellt, die dabei die Aufgabe der Speicherung der aktuellen Positionen übernimmt.

2.4.1_set- /getPosition

Die Hauptfunktion dieser Klasse heißt „setPosition“. In dieser wird eine bestimmte Spielfigur um eine bestimmte Augenzahl des Würfels nach vorne verschoben. Dabei wird überprüft, ob das Feld belegt ist und ob die Figur im Ziel angekommen ist. Mit der Funktion „getPosition“ können die aktuellen Positionen abgerufen werden.

2.4.2_Andere Funktionen

Zusätzlich gibt es noch 3 weitere Funktionen. Die Funktion „isFieldEmpty“ gibt zurück, ob ein bestimmtes Feld frei ist. Die Funktion „kill“ setzt die Figur an einem bestimmten Feld zurück. Zuletzt gibt es noch die Funktion „getFinishedTokens“, die die Anzahl an Figuren im Ziel zurückgibt.

3_Beispiele

3.1_Würfel 0

Ausgabe „wuerfel0.txt“

```
würfel 1 : [1, 2, 3, 4, 5, 6]
würfel 2 : [1, 1, 1, 6, 6, 6]
würfel 3 : [1, 2, 3, 4]
würfel 4 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
würfel 5 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
würfel 6 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20]
             1      2      3      4      5      6
würfel 1 : [-----, lose , lose , win  , win  , win  ]
würfel 2 : [win  , -----, lose , win  , win  , win  ]
würfel 3 : [win  , win  , -----, win  , win  , win  ]
würfel 4 : [lose , lose , lose , -----, equal, win  ]
würfel 5 : [lose , lose , lose , equal, -----, win  ]
würfel 6 : [lose , lose , lose , lose , lose , -----]
```

3.2_Würfel 1

Ausgabe „wuerfel1.txt“

```
Würfel 1 : [1, 2, 3, 4, 5, 6]
Würfel 2 : [2, 3, 4, 5, 6, 7]
Würfel 3 : [3, 4, 5, 6, 7, 8]
Würfel 4 : [4, 5, 6, 7, 8, 9]
Würfel 5 : [5, 6, 7, 8, 9, 10]
Würfel 6 : [6, 7, 8, 9, 10, 11]
             1      2      3      4      5      6
Würfel 1 : [-----, win  , win  , win  , win  , win  ]
Würfel 2 : [lose , -----, win  , win  , win  , win  ]
Würfel 3 : [lose , lose , -----, win  , win  , win  ]
Würfel 4 : [lose , lose , lose , -----, win  , win  ]
Würfel 5 : [lose , lose , lose , lose , -----, win  ]
Würfel 6 : [lose , lose , lose , lose , lose , -----]
```

3.3_Würfel 2

Ausgabe „wuerfel2.txt“

```
Würfel 1 : [1, 1, 1, 1, 1, 6]
Würfel 2 : [1, 1, 1, 1, 6, 6]
Würfel 3 : [1, 1, 1, 6, 6, 6]
Würfel 4 : [1, 1, 6, 6, 6, 6]
Würfel 5 : [1, 6, 6, 6, 6, 6]
             1      2      3      4      5
Würfel 1 : [-----, lose , lose , lose , lose ]
Würfel 2 : [win  , -----, lose , lose , equal]
Würfel 3 : [win  , win  , -----, lose , win  ]
Würfel 4 : [win  , win  , win  , -----, win  ]
Würfel 5 : [win  , equal, lose , lose , -----]
```

3.4_Würfel 3

Ausgabe „wuerfel3.txt“

```
Würfel 1 : [1, 2, 5, 6]
Würfel 2 : [1, 2, 3, 4, 5, 6]
Würfel 3 : [1, 2, 3, 4, 5, 6, 7, 8]
Würfel 4 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Würfel 5 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Würfel 6 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
             1      2      3      4      5      6
Würfel 1 : [-----, win  , win  , win  , win  , win  ]
Würfel 2 : [lose , -----, win  , win  , win  , win  ]
Würfel 3 : [lose , lose , -----, win  , win  , win  ]
Würfel 4 : [lose , lose , lose , -----, equal, win  ]
Würfel 5 : [lose , lose , lose , equal, -----, win  ]
Würfel 6 : [lose , lose , lose , lose , lose , -----]
```

3.5_eigene Würfel

Ausgabe „eigeneWuerfel.txt“

```
Würfel 1 : [1]
Würfel 2 : [1, 2]
Würfel 3 : [1, 2, 3]
Würfel 4 : [1, 2, 3, 4]
Würfel 5 : [1, 2, 3, 4, 5]
Würfel 6 : [1, 2, 3, 4, 5, 6]
Würfel 7 : [1, 2, 3, 4, 5, 6, 7]
Würfel 8 : [1, 2, 3, 4, 5, 6, 7, 8]
             1      2      3      4      5      6      7      8
Würfel 1 : [-----, equal, equal, equal, equal, win  , win  , win  ]
Würfel 2 : [equal, -----, equal, equal, equal, win  , win  , win  ]
Würfel 3 : [equal, equal, -----, equal, equal, win  , win  , win  ]
Würfel 4 : [equal, equal, equal, -----, equal, win  , win  , win  ]
Würfel 5 : [equal, equal, equal, equal, -----, win  , win  , win  ]
Würfel 6 : [lose , lose , lose , lose , lose , -----, equal, win  ]
Würfel 7 : [lose , lose , lose , lose , lose , equal, -----, equal]
Würfel 8 : [lose , lose , lose , lose , lose , lose , equal, -----]
```

An diesem eigenen Beispiel lässt sich noch einmal gut erkennen, dass Würfel ohne 6 automatisch verlieren und daher nicht simuliert werden müssen.

4_Quellcode

4.1_Klasse „a4“

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class a4 {

    //___number of runs___//
    static int numberOfRuns = 10000;
    public static void calculate(String filename) throws
FileNotFoundException {
        //___Read-file___//
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        int numberOfCubes = scanner.nextInt();
        int[][] cubes = new int[numberOfCubes][];
        boolean[] calculatedCubes = new boolean[numberOfCubes];
        int[][] results = new int[numberOfCubes][numberOfCubes];
        for(int i = 0; i < numberOfCubes; i++){
            int numberOfSides = scanner.nextInt();
            cubes[i] = new int[numberOfSides];
            for(int j = 0; j < numberOfSides; j++){
                cubes[i][j] = scanner.nextInt();
            }
        }

        //___Calculate for each Cube___//
        for (int i = 0; i < numberOfCubes; i++){
            for(int j = 0; j < numberOfCubes; j++){
                //___Check if Cube is already calculated___//
                if(calculatedCubes[j]){
                    //___reverse results___//
                    if(results[j][i] != 2){
                        results[i][j] = results[j][i] ^ 1;
                    } else {
                        results[i][j] = 2;
                    }
                    continue;
                //___Check if Cube i and j are the same___//
                } else if(i == j){
                    results[i][j] = -1;
                    continue;
                }
                //___Check if cube contains a six___//
                if(!checkSix(cubes[i])){
                    results[i][j] = 0;
                    if(!checkSix(cubes[j])){
                        results[i][j] = 2;
                    }
```

```
                    continue;
                } else if(!checkSix(cubes[j])){
                    results[i][j] = 1;
                    continue;
                }

                int numberOfWins = 0;
                int numberOfLoses = 0;
                int start = 0;

                //___Calculate x times___//
                for(int k = 0; k < numberOfRuns; k++){
                    int result = simulateOneGame(new int[][]{cubes[i],
cubes[j]},start);
                    //___Check result___//
                    if(result == 0){
                        numberOfWins+=1;
                    } else if(result == 1){
                        numberOfLoses+=1;
                    }
                    //___change player with XOR Operator___//
                    start = start ^ 1;
                }

                //___Calculate relative difference
                double difference =
(double)numberOfWins/(double)numberOfLoses;
                //___set results to array___//
                if(difference > 0.8 && difference < 1.2){
                    results[i][j] = 2;
                } else if(numberOfWins > numberOfLoses){
                    results[i][j] = 0;
                } else if(numberOfWins < numberOfLoses){
                    results[i][j] = 1;
                }

            }
            //___set Cube to calculated___//
            calculatedCubes[i] = true;
        }
        //___print out result___//
        printOutResults(cubes, results);

    }
    private static int simulateOneGame(int[][] cube, int player){
        //___create new game___//
        Game game = new Game(cube);
        //___start game___//
        while(!game.isFinished()){
            //___get Random number from cube___//
            int number = getRandom(cube[player]);
            //___Move token___//
            boolean again = game.move(number,player);
            //___Switch player___//
            if(!again){
                player = player ^ 1;
            }

        }
        //___return winner___//
        return game.whoHasWon();
    }
```

```
    private static int getRandom(int[] cube){
        Random random = new Random();
        //___get Random index between 0 and cube length___//
        int randomInt = random.nextInt(cube.length);
        //___return number from index___//
        return cube[randomInt];
    }
    private static void printOutResults(int[][] cubes, int[][] results){
        //___Print out cubes___//
        for(int i = 0; i < cubes.length; i++){
            System.out.println("Würfel "+(i+1)+" :
"+Arrays.toString(cubes[i]));
        }
        System.out.println();
        System.out.print("             ");
        for(int i = 0; i < cubes.length; i++){
            System.out.print((i+1)+"      ");
        }
        System.out.println();
        //___Print out result-matrix___//
        for(int i = 0; i < cubes.length; i++){
            System.out.println("Würfel "+(i+1)+" :
"+Arrays.toString(results[i])
            .replace("-1","-----")
            .replace("0","win  ")
            .replace("1","lose ")
            .replace("2","equal"));
        }
    }
    private static boolean checkSix(int[] cubes){
        //___Check each number of Cube___//
        for (int cube : cubes) {
            //___Check if number is 6____//
            if (cube == 6) {
                return true;
            }
        }
        return false;
    }
}
```

4.2_Klasse „Game“

```
public class Game {
    //___Create Variables___//
    Player[] player = new Player[2];
    int[][] cubes;
    int numberOfTokens = 4;
    boolean[] noSolution = new boolean[2];
    public Game(int[][] cubes){
        //___Create 2 Objects of player___//
        player[0] = new Player();
        player[1] = new Player();
        //___Save cube___//
        this.cubes = cubes;
    }
```

```
    public boolean move(int n, int p){
        if(!noSolution[p]) {
            //___Check Start-Field___//
            boolean fieldIsFree = player[p].isFieldEmpty(1);
            //___Move token away from start-field
            if (!fieldIsFree) {
                for (int i = 0; i < numberOfTokens; i++) {
                    if (player[p].getPosition(i) == 1) {
                        boolean answer = killAndSetPosition(1 + n, i, p,
n);
                        if (answer) {
                            return n == 6;
                        } else {
                            break;
                        }
                    }
                }
            }
            //___Move token onto start-field___//
            if (n == 6 && fieldIsFree) {
                for (int i = 0; i < numberOfTokens; i++) {
                    int position = player[p].getPosition(i);
                    if (position == 0) {
                        if (killAndSetPosition(1, i, p, 1)) {
                            return true;
                        }
                    }
                }
            }
            //___"normal" movement___//
            int highestPosition = 0;
            int highestPositionIndex = 0;
            //___Find highest position___//
            for (int i = 0; i < numberOfTokens; i++) {
                int position = player[p].getPosition(i);
                if (position > highestPosition && position <= 44 -
player[p].getFinishedTokens ()) {
                    highestPosition = position;
                    highestPositionIndex = i;
                }
            }
            if (highestPosition != 0) {
                checkNoSolution(highestPosition, p);
                if (killAndSetPosition(highestPosition + n,
highestPositionIndex, p, n)) {
                    return n == 6;
                }
            }
        }
        return false;
    }
    private boolean killAndSetPosition(int position, int positionIndex, int
p, int n){
        //___Check if token can be moved___//
        if(player[p].setPosition(positionIndex,n)){
            //___Check if token can be killed__//
            if(position <= 40){
                int killPosition;
                if(position <= 20){
                    killPosition = position + 20;
                } else{
                    killPosition = position - 20;
```

```
                }
                player[p^1].kill(killPosition);
            }
            return true;
        }
        return false;
    }

    private void checkNoSolution(int highestPosition, int p){
        //___Check if there is no possible move___//
        if(highestPosition+cubes[p][0] > 44-player[p].getFinishedTokens()){
            noSolution[p] = true;
        }
    }

    public boolean isFinished(){
        //___Check if one player is finished___//
        return player[0].getFinishedTokens() == 4 ||
player[1].getFinishedTokens() == 4 || (noSolution[0] && noSolution[1]);
    }
    public int whoHasWon() {
        //___Check which player has finished___//
        if (player[0].getFinishedTokens() == 4) {
            return 0;
        } else if(player[1].getFinishedTokens() == 4){
            return 1;
        } else{
            return 2;
        }
    }
}
```

4.3_Klasse „Player“

```
public class Player {
    //___Create Variables___//
    private final int numberOfTokens = 4;
    private final int[] positions = new int[numberOfTokens];
    private int finishedTokens = 0;
    public Player(){
        //___Set default values___//
        for(int i = 0; i < numberOfTokens; i++){
            positions[i] = 0;
        }
    }
    public int getPosition(int index){
        return positions[index];
    }
    public boolean setPosition(int index,int n){
        //___reade old position___//
        int oldPosition = positions[index];
        //___calculate new Position___//
        int newPosition = oldPosition+n;
        //___Check if move is possible___//
        if(isFieldEmpty(newPosition)){
            if(newPosition <= 44){
                positions[index] = newPosition;
                if(newPosition == 44- finishedTokens){
                    finishedTokens +=1;
                }
```

```
                return true;
            }
        }
        return false;
    }
    public boolean isFieldEmpty(int field){
        //___Check if field is empty___//
        for(int i = 0; i < numberOfTokens; i++){
            if(positions[i] == field){
                return false;
            }
        }
        return true;
    }
    public void kill(int field){
        //___Kill token at field___//
        for(int i = 0; i < numberOfTokens; i++){
            if(positions[i] == field){
                positions[i] = 0;
                return;
            }
        }
    }
    public int getFinishedTokens() {
        return finishedTokens;
    }
}
```