Aufgabe 4: Würfelglück

Team-ID: 00921

Team: Ctrl + Intelligence

Bearbeiter dieser Aufgabe:
Simon Horst

20. November 2021

Inhaltsverzeichnis

Lösungsidee....1
Simulation....1
Sonderfälle / Probleme....3
Umsetzung....4
Beispiele....5
Beispiel 1:....5
Beispiel 2:....6
Beispiel 3:....7
Quellcode....7

Lösungsidee

Simulation

Das Programm soll zunächst in der Lage sein, „Mensch ärgere dich nicht“, simulieren zu können. Genauer, muss es ein Spiel zwischen zwei Spielern, wovon jeder Spieler einen unterschiedlichen Würfel hat und sich auf der je gegenüberliegenden Seite, bzw. Farbe, befindet, „nachspielen“. Hierbei muss am Ende ein Gewinner ermittelt werden, wobei auch noch dokumentiert wird ob dieser das Spiel begonnen hat oder nicht. Nach einer Simulation, wird der „zweit-würfelnde“ aus der vorherigen Runde nun zum „erst-würfelnden“, so wird gewährleistet, dass etwa 50% der Spiele, einer anderer angefangen hat.

Um wie in der Aufgabe gefordert, jeden Würfel gegen jeden anderen spielen zu lassen. Muss man zuallererst damit beginnen, so viele Simulationen zu starten, wie es Würfel-Kombination gibt.

- Angenommen man hat n Würfel, so muss jeder Würfel gegen jeden anderen Würfel ausschließlich sich selbst spielen. Es ergeben sich also $n(n-1)$ Kombinationen. (Unter

der Voraussetzung das man doppelte Kombinationsmöglichkeiten mit einbezieht z.B. Würfel1 gegen Würfel2 und Würfel2 gegen Würfel1)

- Da bis zum jetzigen Zeitpunkt für jede Kombination, nur jeweils ein Spiel simuliert wird, sind die erhaltenen Daten nicht Aussagekräftig genug. Um nun bestimmen zu können ob ein Würfel wirklich besser ist, muss die Simulation dementsprechend mehrmals durchgeführt werden. Davon ausgehend werden diese Simulationen k – mal wiederholt.
- Also muss man insgesamt $k(n(n-1))$ Simulationen durchführen.
- Am Ende sollen zu allen Simulationen der Kombinationen, die erhaltenen Ergebnisse zusammengerechnet und ausgewertet werden. Man erhält also bei n Würfeln und k Wiederholungen, für jeden einzelnen Würfel, Daten aus $2(k(n(n-1)))(\frac{1}{n})$ Spielsimulationen (2-mal, Da bei einem Spiel immer 2 Würfel beteiligt sind).

Angenommen man hat 6 verschiedene Würfel und möchte jedes Spiel (einer möglichen Kombination von Würfeln) etwa 100 mal Simulieren. So werden $n(n-1)=6(5)=30$ mal 100 Spiele Simuliert, wobei $30(100)=3000$ Datensätze entstehen. Da bei den 3000 Spielen immer jeweils zwei Würfel beteiligt sind gibt es $2(k(n-1))(\frac{1}{n})=30(100)(2)(\frac{1}{6})=6000(\frac{1}{6})=1000$ Daten zu den einzelnen Würfeln.

Aus diesen Daten kann dann zum Schluss die Wahrscheinlichkeit bestimmt werden: Wie oft ein gegebener Würfel gewinnt und ebenso die Frage beantwortet werden, ob das Anfangen eines Spieles dazu beiträgt zu gewinnen.

Als Beispiel:

Gegeben sind folgende Würfel: W1: D6 (1,2,3,4,5,6) ; W2: D6 (1,2,3,5,6,6)

Bei einer Simulation von 100 Spielen, stellte sich heraus, dass W1 aus $2(k(n(n-1)))(\frac{1}{n})=2(100(2))(\frac{1}{2})=200$ Spielen, 24 gewonnen hatte, wobei 12 davon gewonnen wurden, als W1 anfing. Dementsprechend gewann W1, 12% der Spiele wobei 50% davon gewonnen wurde als W1 den Erstzug hatte

W2 im Gegenzug gewannt 176 Spiele (88%), wobei ebenfalls 50% gewonnen wurde als W2 anfing.

Um die Spiele zu beenden, wurden im durchschnitt 164.68 Runden benötigt.

Man kann also aus als Fazit ziehen, dass es mit W2 deutlich Wahrscheinlicher zu gewinnen ist als mit W1 und dass es egal ist, wer anfängt.

Es kann jedoch bei Gewissen Würfel und Würfelkombinationen zu einem Problem kommen, welches es erschwert den Sieger festzulegen, bzw. das Spiel zu beenden.

Sonderfälle / Probleme

Damit ein Spiel beendet ist und ein Gewinner festgestellt werden kann muss folgende Bedingung erfüllt sein:

Alle Figuren eines Spielers müssen sich in den Zielfeldern befinden.

Daraus ergeben sich zwei Probleme:

1. Der Spieler muss in der Lage sein, seine Figuren überhaupt aus dem Startfeld zu bekommen. Also er muss eine 6 Würfeln zu können.
2. Der Spieler muss in der Lage sein, seine Figuren überhaupt ins das Zielfeld würfeln zu können.

Lösung Problem 1:

Angenommen ein Spieler hat einen D4 als Würfel mit folgenden Seiten: 1,2,3,4

Dieser Spieler würde bis auf seine Startfigur, nie eine andere Figur auf das Spielfeld bringen können, da man dafür eine 6 Würfeln müsste. Dementsprechend gewinnt automatisch der Gegner. (Wenn der Gegner in der Lage ist das Spiel zu beenden, wird dieses fertig gespielt. „In der Lage“ bedeutet in diesem Fall, dass der Gegenspieler nicht Problem 2 oder 1 „zum Opfer“ fällt)
Anmerkung: Die Anzahl der Seiten dieses Würfels ist irrelevant, das Problem wird nur dadurch verursacht, dass der Würfel nicht 6 würfeln kann.

Lösung Problem 2:

Angenommen beide Spieler haben einen Würfel, der nicht 1 würfeln kann (z.B.: D4 (2,3,4,5))

Wenn es sich in Richtung Ende eines Spieles, die Situation ergibt, dass beide Spieler drei ihrer Figuren in ihrem Zielfeld stehen haben und diese b,c,d besetzen und die letzte Figur sich ein Feld vor a befindet, kann kein Sieger festgestellt werden. Da die Letzte Figur sich ein Feld nach vorne Bewegen muss um auf a zu kommen aber die Würfel dies nicht zulassen, könnte man diesen Zustand als „softlock“ bezeichnen. In diesem Fall muss das Spiel nach einer gewissen Zeit terminiert werden, wobei die Spieldaten dann beschreiben müssen, dass es in diesem Spiel keinen Gewinner gab.

Da sich die Spielfiguren innerhalb der Zielfelder fortbewegen können, bzw. aufrücken können, ist jedes Spiel bei denen mindestens einer der Spieler ein Würfel mit einer 1 hat, zu beenden. Da eine 1 immer dem Spieler die Möglichkeit gibt eine Figur zu bewegen. Natürlich kann es in diesem Szenario auch vorkommen, dass 1000mal hintereinander (zum Beispiel) 6 gewürfelt wird und somit

das Spiel solange braucht, dass die Lösung des 2. Problems greift und die Simulation nach einer gewissen Zeit terminiert. Allerdings ist dies so unwahrscheinlich, dass dieses Szenario keine Relevanz zeigt.

Damit ein Spiel zumindest Spielbar ist, muss ein Würfel mindestens eine 6 würfeln können. Wenn ein Spiel immer beendet werden soll, empfiehlt sich ein Würfel zu haben der 1 und 6 würfeln kann.

Daraus kann man auch schon vermuten, dass Würfel die 1 und 6 als Seite besitzen allgemein eine höhere Gewinnchance haben. Wenn man sich das erste Beispiel anschaut (Mit den Würfeln W1 und W2) konnte man auch schon da beobachten, dass W2 deutlich öfter gewann als W1.

Ansonsten gibt es bezogen auf die Würfel und Spielmechanik /den Spielregeln keinerlei Probleme. Die größte Schwierigkeit bezogen auf dieses Problem ist, das Spiel mit allen Regeln zuverlässig zu simulieren.

Umsetzung

Das Programm wurde in Java implementiert, da allerdings die Lösungsidee schon die meisten Probleme beleuchtet hat denen man über den Weg laufen kann, ist diese recht schnell zu erklären. Die „main“-Methode lädt zuerst aus einer Textdatei die nötigen Würfel, die auf ihre Überlegenheit hin getestet werden sollen. Dort wird auch vom Anwender definiert, wie oft ein Spiel simuliert werden soll.

Daraufhin wird ein Spielfeld Objekt erzeugt, dieses Verwaltet alle Funktionen um ein Spiel von „Mensch ärgere dich nicht“ zu simulieren. In Spielfeld wird daraufhin die Methode „simulateNGames“ aufgerufen, die als Parameter die Würfel der 2 Spieler und die Anzahl an Simulationen erwartet und ein Array von „gameresult“ ausgibt. Das Objekt „gameresult“ speichert den Gewinner (Spieler1,Spieler2,Keiner), die Anzahl der Runden, und wer zuerst Würfeln durfte.

Diese Liste wird dann in die Methode „evaluateSimulation“ gegeben, welche die einzelnen Werte des Arrays von „gameresult“ zusammen Addiert und ein „messwerte“ Objekt erstellt und zurückgibt. Das „messwerte“ Objekt speichert wiederum alle Daten bezüglich eines Würfels (Anzahl der gewonnenen Spiele, Anzahl der gewonnenen Spiele als zuerst würfelnder, Anzahl der gespielten Spiele, Anzahl der durchschnittlichen Runden pro Spiel).

Letztlich wird dieses Array der Messwerte (wovon jedes Element zu einem Würfel gehört, also besitzt es n Elemente) ausgegeben.

Anmerkung: Für die Lösung des 2. Problems, wurde eine „Terminierungszeit“ von 20 Millisekunden angesetzt. Bei z.B. „wuerfel1.txt“ kann es deshalb dazu kommen (da die Würfel nicht alle über eine 1 verfügen), dass die Simulation und Auswertung bei über 100 Wiederholungen etwas dauern könnte. Allerdings funktionieren Würfelsätze die immer beendet werden können (also mindestens eine 1 und eine 6 haben), wie z.B.: in „wuerfel{0,2,3}.txt“ auch bei 10.000 Wiederholungen, in unter einer Sekunde.

Beispiele

Um das Programm auszuführen, muss die Batch-Datei „Aufgabe4Starten“ gestartet werden, daraufhin erhält man einen Prompt und muss nun den Dateipfad der zu testenden Datei eingeben (der Name der Datei muss mit Endung eingegeben werden, befindet diese sich im gleichen Ordner wie die Batchdatei, kann man auch nur den Namen eingeben und kann sich den Pfad sparen). Danach erhält man einen Prompt bei dem man die Anzahl der gewünschten Wiederholungen eingeben muss (Einfach eine Zahl eingeben).

Wenn es erfolgreich war, wird für jede Kombination von Spielen zwischen den Würfel ein Ergebnis ausgegeben und am Ende das allgemeine Ergebnis und der beste Würfel der Menge ausgegeben.

Beispiel 1:

```
Simulation fuer Mensch aergere dich nicht
Bitte geben sie den Dateinamen an: wuerfel0.txt
Bitte geben sie die Anzahl der gewuenschten Wiederholungen an: 100

Der Wuerfel, D6 (1,2,3,4,5,6; Erwartungswert:3,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 125.182
Gewinne: 718, das waeren: 71.8%
Gewinne bei denen man angefangen hat: 361, das waeren: 50.27855%


Der Wuerfel, D6 (1,1,1,6,6,6; Erwartungswert:3,50; Varianz:2,50) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 101.92
Gewinne: 973, das waeren: 97.299995%
Gewinne bei denen man angefangen hat: 485, das waeren: 49.84584%


Der Wuerfel, D4 (1,2,3,4; Erwartungswert:2,50; Varianz:1,12) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 64.902
Gewinne: 0, das waeren: 0.0%



Der Wuerfel, D10 (0,1,2,3,4,5,6,7,8,9; Erwartungswert:4,50; Varianz:2,87) hat folgenden Werte erzielen k
oennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 116.559
Gewinne: 529, das waeren: 52.899998%
Gewinne bei denen man angefangen hat: 261, das waeren: 49.338375%


Der Wuerfel, D12 (1,2,3,4,5,6,7,8,9,10,11,12; Erwartungswert:6,50; Varianz:3,45) hat folgenden Werte erz
ielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 110.665
Gewinne: 506, das waeren: 50.6%
Gewinne bei denen man angefangen hat: 264, das waeren: 52.173912%


Der Wuerfel, D20 (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20; Erwartungswert:10,50; Varianz:5,77
) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 112.52
Gewinne: 274, das waeren: 27.4%
Gewinne bei denen man angefangen hat: 129, das waeren: 47.08029%


Es stellt sich also heraus, dass der D6 (1,1,1,6,6,6; Erwartungswert:3,50; Varianz:2,50) mit einer Siege
schance von: 97.299995% am besten ist, um das Spiel zu gewinnen.
Drücken Sie eine beliebige Taste . . .
```

In diesem Beispiel wird das Textdokument „wuerfel0.txt“ eingelesen und die Simulation mit 100 Wiederholungen ausgeführt (Die Spiele bei denen jede Kombination aufgeführt wird wurden hier der Länge halber weggelassen).

Hier ist vor allem Auffällig, dass der Würfel D4 (1,2,3,4) kein einziges mal Gewonnen hat (die Gründe dafür wurden in Problem 1 beleuchtet) und das auch der D20 (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20) mit am schlechtesten abschneidet. Die miserable Performance des D20 lässt sich dadurch erklären, dass es recht unwahrscheinlich ist eine 6 oder 1 zu würfeln, was es schwer macht das Spiel zu beenden

Beispiel 2:

```
Simulation fuer Mensch aergere dich nicht
Bitte geben sie den Dateinamen an: wuerfel1.txt
Bitte geben sie die Anzahl der gewuenschten Wiederholungen an: 100


Der Wuerfel, D6 (1,2,3,4,5,6; Erwartungswert:3,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 159.983
Gewinne: 639, das waeren: 63.9%
Gewinne bei denen man angefangen hat: 323, das waeren: 50.54773%


Der Wuerfel, D6 (2,3,4,5,6,7; Erwartungswert:4,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 127.295
Gewinne: 580, das waeren: 58.0%
Gewinne bei denen man angefangen hat: 293, das waeren: 50.517242%


Der Wuerfel, D6 (3,4,5,6,7,8; Erwartungswert:5,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 109.768
Gewinne: 498, das waeren: 49.8%
Gewinne bei denen man angefangen hat: 242, das waeren: 48.594376%


Der Wuerfel, D6 (4,5,6,7,8,9; Erwartungswert:6,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 99.399
Gewinne: 414, das waeren: 41.4%
Gewinne bei denen man angefangen hat: 205, das waeren: 49.516907%


Der Wuerfel, D6 (5,6,7,8,9,10; Erwartungswert:7,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 87.333
Gewinne: 231, das waeren: 23.1%
Gewinne bei denen man angefangen hat: 113, das waeren: 48.91775%


Der Wuerfel, D6 (6,7,8,9,10,11; Erwartungswert:8,50; Varianz:1,71) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 1000
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 83.076
Gewinne: 98, das waeren: 9.8%
Gewinne bei denen man angefangen hat: 49, das waeren: 50.0%


Es stellt sich also heraus, dass der D6 (1,2,3,4,5,6; Erwartungswert:3,50; Varianz:1,71) mit einer Siegeschance von: 63.
9% am besten ist, um das Spiel zu gewinnen.
```

Hier wird „wuerfel1.txt“ mit 100 Wiederholungen ausgeführt. Da hier (Siehe Problem 2) nicht jedes Spiel fertig gespielt werden kann ist dementsprechend auch die Gewinnchance allgemein niedriger ausgefallen und es wird ein bisschen länger für das Rechnen gebraucht.

Beispiel 3:

```
Simulation fuer Mensch aergere dich nicht
Bitte geben sie den Dateinamen an: wuerfel2.txt
Bitte geben sie die Anzahl der gewuenschten Wiederholungen an: 100
Der Wuerfel, D6 (1,1,1,1,1,6; Erwartungswert:1,83; Varianz:1,86) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 800
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 151.19125
Gewinne: 11, das waeren: 1.375%
Gewinne bei denen man angefangen hat: 6, das waeren: 54.545456%


Der Wuerfel, D6 (1,1,1,1,6,6; Erwartungswert:2,67; Varianz:2,36) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 800
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 163.255
Gewinne: 207, das waeren: 25.875%
Gewinne bei denen man angefangen hat: 98, das waeren: 47.342995%


Der Wuerfel, D6 (1,1,1,6,6,6; Erwartungswert:3,50; Varianz:2,50) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 800
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 128.8175
Gewinne: 424, das waeren: 52.999996%
Gewinne bei denen man angefangen hat: 213, das waeren: 50.23585%


Der Wuerfel, D6 (1,1,6,6,6,6; Erwartungswert:4,33; Varianz:2,36) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 800
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 108.385
Gewinne: 599, das waeren: 74.875%
Gewinne bei denen man angefangen hat: 306, das waeren: 51.085144%


Der Wuerfel, D6 (1,6,6,6,6,6; Erwartungswert:5,17; Varianz:1,86) hat folgenden Werte erzielen koennen:
Anzahl der Spiele: 800
Anzahl der durchschnittlichen Runden um das Spiel zu beenden: 99.73625
Gewinne: 759, das waeren: 94.875%
Gewinne bei denen man angefangen hat: 377, das waeren: 49.67062%


Es stellt sich also heraus, dass der D6 (1,6,6,6,6,6; Erwartungswert:5,17; Varianz:1,86) mit einer Siegeschance von: 94.
875% am besten ist, um das Spiel zu gewinnen.
```

Hier wird „wuerfel2.txt“ mit 100 Wiederholungen ausgeführt. Besonders in diesem Beispiel ist es, dass sich die Vermutung aus dem Anfang bestätigt hat (dass die Anzahl der 6 und 1 sich stark auf die Siegeschance auswirkt). Die Anzahl der Sechsen im Verhältnis zu Einsen auf einem Würfel wirkt sich in diesem Spiel stark Positiv auf die Siegeschance aus.

Quellcode

```
public static void initializeValuesFromFile(String file) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(new File(file)));
        String line;
        line = reader.readLine();
        int n = Integer.valueOf(line); // Anzahl der Wuerfel wird ausgelesen (1. Zeile)
        dices = new wuerfel[n];
        System.out.println("Es gibt: " + n + " Wuerfel \n\n");

        for (int i = 0; i < n; i++) {
            line = reader.readLine();
            String[] elements = line.split(" ");
            int[] faces = new int[elements.length - 1];

            for (int j = 0; j < faces.length; j++) {
                faces[j] = Integer.valueOf(elements[j + 1]); // Die einzelnen Zahlenwerte einer Seite eines Wuerfels wird Festgelegt
            }

            dices[i] = new wuerfel(Integer.valueOf(elements[0]), faces); // Die Anzahl der Seiten des Wuerfels
        }

        reader.close();

        finalResults = new messwerte[dices.length];
        for (int i = 0; i < finalResults.length; i++) {
            finalResults[i] = new messwerte(null, 0, 0, 0, 0); // Initialisierung
        }
    } catch (IOException e) {
        System.out.println("Es gab ein Fehler die Datei zu lesen");
    }
}
```

```
    static wuerfel[] dices; // Speichert alle Würfel
    static messwerte[] finalResults; // Speichert die Finalen Statistiken der Würfel
    static int maxSims; // Die Anzahl an Simulationen

    Run | Debug
    public static void main(String[] args) {
        maxSims = Integer.valueOf(args[1]);
        initializeValuesFromFile(args[0]);
        spielfeld simulations = new spielfeld(); // Die Simulationsebene fuer "Mensch aergere dich nicht"

        // Jeden Wuerfel durchgehen: (n * (n-1)) Kombinationen
        for (int i = 0; i < dices.length; i++) { // Spieler 1
            messwerte[] m = new messwerte[2];
            m[0] = new messwerte(dices[i], 0, 0, 0, 0);
            m[1] = new messwerte(dices[i], 0, 0, 0, 0);
            for (int j = 0; j < dices.length; j++) { // Spieler 2
                if (i != j) {
                    wuerfel[] temp = { dices[i], dices[j] };
                    gameresult[] result = simulations.simulateNGames(temp, maxSims);
                    System.out.println("In " + maxSims + " Simulationen, spielten ein D" + temp[0].getSides() + "("
                            + temp[0].getFacesInString() + "), Spieler 1," + " und ein D" + temp[1].getSides() + "("
                            + temp[1].getFacesInString() + ")" + ", Spieler 2, miteinander.");
                    m = evaluateSimulation(result, temp); // Erechnet von allen Ergebnissen der Simulation einfach die
                                                          // Summen.
                    finalResults[j].setWuerfel(dices[j]);
                    finalResults[j].addToGames(m[1].getGames());
                    finalResults[j].addToRounds(m[1].getRounds());
                    finalResults[j].addToWins(m[1].getWins());
                    finalResults[j].addToWinsAB(m[1].getWinsAB());
                    // Statistiken der einzelnen Spiele den Wuerfeln zuschreiben
                    finalResults[i].setWuerfel(dices[i]);
                    finalResults[i].addToGames(m[0].getGames());
                    finalResults[i].addToRounds(m[0].getRounds());
                    finalResults[i].addToWins(m[0].getWins());
                    finalResults[i].addToWinsAB(m[0].getWinsAB());
                }
            }

        }

        printFinalResults(finalResults); // Gibt einfach die Mitterlwerte der Ergebnisse aus und bestimmt die
                                         // Wahrscheinlichkeiten sowie den Besten wuerfel in die Konsole aus
    }
public class gameresult {
    private int winner;
    private int turns;
    private boolean beginner;
    public gameresult(int pTurns, int pWinner, boolean pBeginner){
        winner = pWinner;
        turns = pTurns;
        beginner = pBeginner;
    }

    public int getTurns(){
        return turns;
    }
    public int getWinner(){
        return winner;
    }

    public boolean getBeginner(){
        return beginner;
    }
}
```

```java
public class messwerte {
    wuerfel dice;
    int wins,winsAsBeginner,games;
    float rounds;
    public messwerte(wuerfel pDice, int pWins, int pWinsAsBeginner, int pGames, float pRounds){
        dice = pDice; // Zu jedem dieser Variablen, gibt es noch getter und setter
        wins = pWins;
        winsAsBeginner = pWinsAsBeginner;
        games = pGames;
        rounds = pRounds;
    }
public class wuerfel {
    Random rand = new Random();
    private int sides;
    private int[] faces;

    public wuerfel(int pSides, int[] pFaces) {
        sides = pSides;
        faces = pFaces;
        //printInformation();
    }

    public void printInformation() {
        System.out.println("Der Würfel hat " + sides + " Seiten:");
        for (int i = 0; i < faces.length; i++) {
            System.out.print(" " + faces[i]);
        }
        System.out.println();
    }

    public int getSides() {
        return sides;
    }

    public int[] getFaces() {
        return faces;
    }

    public int getRandomValue() {
        return faces[rand.nextInt(faces.length)];
    }

    public boolean hasASix(){
        boolean hasOne = false;
        for (int e : this.getFaces()) {
            if(e == 6)
            hasOne = true;
        }
        return hasOne;
    }

    public boolean hasAOne(){
        boolean hasOne = false;
        for (int e : this.getFaces()) {
            if(e == 1)
            hasOne = true;
        }
        return hasOne;
    }
```

```java
public class figur {
    private int position;
    private boolean isPlayerOne;
    public figur(boolean pIsPlayerOne){
        position = -1;
        isPlayerOne = pIsPlayerOne;
    }

    public void setPosInSteps(int step){
        position += step;
    }

    public void setPos(int step){
        position = step;
    }

    public int getPos(){
        return position;
    }

    public boolean isPlayerOne(){
        return isPlayerOne;
    }

   // public void setisPlayerOne(boolean isInThere){
   //     isPlayerOne = isInThere;
   // }
}
```

ab hier nur noch die Klasse Spielfeld in chronologischer Reihenfolge:

```java
public class spielfeld {
    figur[] player1 = new figur[4]; // Spielfiguren
    figur[] player2 = new figur[4];
    boolean player1Turn;
    wuerfel[] dices;

    public spielfeld() {

    }

    public gameresult[] simulateNGames(wuerfel[] pDices, int simulations) {
        dices = pDices;
        gameresult[] result = new gameresult[simulations];
        for (int i = 0; i < result.length; i++) {
            result[i] = playGame(pDices, ((i % 2) == 0) ? true : false); // etwa 50% fängt der eine und 50% der andere
                                                                         // an
        }
        return result;
    }
```

```
public gameresult playGame(wuerfel[] dicesOfEachPlayer, boolean beginner) { // 0: Spieler1, 1: Spieler2
    if (!dicesOfEachPlayer[0].hasASix()) {
        return new gameresult(0, 0, beginner);
    }
    int turn = 0;
    boolean gameIsFinished = false;

    determineBeginner(dicesOfEachPlayer);
    player1Turn = beginner;
    player1[0].setPos(0); // Startposition der Spielerfigur
    player2[0].setPos(0); // Startposition der Spielerfigur

    long start = System.currentTimeMillis();
    while (!gameIsFinished) {
        if (System.currentTimeMillis() - start > 20) // Fall das Spiel nicht nach 20 Millisekunden Fertig ist wird
                                                     // dieses abgebrochen und keiner Gewinnt (zB. 3 Figuren sind
                                                     // schon im Ziel und es gibt keine Zahl die Gewürfelt werden
                                                     // kann um die Letzte Figur ins Ziel zu bringen)
            return new gameresult(0, 2, beginner);
        turn++;
        if (player1Turn) {
            int roll = dicesOfEachPlayer[0].getRandomValue();
            if (roll == 6) { // Falls eine 6 gewürfelt wurde; Alle Bedingungen werden hier ohne EndTurn() abgeschlossen, so kann man wieder Würfeln
                if (AIsFree(player1) != 10) { // Falls Sein A Feld von eigenen Steinen belegt ist
                    if (Move(player1[AIsFree(player1)], roll))
                        continue; // Falls Stein auf A wird dieser um 6 bewegt, wenn nicht möglich dann wird der
                                  // nächstbeste bewegt
                    else if (!makeMove(player1, roll)) {
                        continue;
                    }
                } else { // Wenn A nicht von Eigenen Steinen belegt ist
                         // Versuche Figur die auf a ist um 6 zu bewegen
                    if (getFigureFromB(player1) != 10) { // Wenn auf B mind. ein Spieler ist
                        if (enemyIsNotOnA(player1) == 10) { // Kein Gegner auf A
                            player1[getFigureFromB(player1)].setPos(0);
                            continue;
                        } else // Falls auf A ein Gegner ist, schicke diesen in sein B feld und setze einen aus
                               // eigenem auf A
                        {
                            player2[enemyIsNotOnA(player1)].setPos(-1);
                            player1[getFigureFromB(player1)].setPos(0);
                            continue;
                        }
                    } else {
                        if (!makeMove(player1, roll)) {
                            continue;
                        } // Bewege nächstbesten
                    }
                }
            } else { // Wenn was anderes als eine 6 gewürfelt wird
                if (AIsFree(player1) != 10) { // Falls Sein A Feld von eigenen Steinen belegt ist
                    if (!Move(player1[AIsFree(player1)], roll))
                        if (!makeMove(player1, roll)) {
                            endTurn();
                            continue;
                        }
                } else {
                    if (!makeMove(player1, roll)) {
                        endTurn();
                        continue;
                    }
                }
            }
            if (roll != 6)
                endTurn();
        } else { // Spieler 2
```

```
        } else { // Spieler 2
            int roll = dicesOfEachPlayer[1].getRandomValue();
            if (roll == 6) { // Falls eine 6 gewürfelt wurde; Alle Bedingungen werden hier ohne EndTurn() abgeschlossen, so kann man wieder Würfeln
                if (AIsFree(player2) != 10) { // Falls Sein A Feld von eigenen Steinen belegt ist
                    if (Move(player2[AIsFree(player2)], roll))
                        continue; // Falls Stein auf A wird dieser um 6 bewegt, wenn nicht möglich dann wird der
                                  // nächstbeste bewegt
                    else if (!makeMove(player2, roll)) {
                        continue;
                    }
                } else { // Wenn A nicht von Eigenen Steinen belegt ist
                         // Versuche Figur die auf a ist um 6 zu bewegen
                    if (getFigureFromB(player2) != 10) { // Wenn auf B mind. ein Spieler ist
                        if (enemyIsNotOnA(player2) == 10) { // Kein Gegner auf A
                            player2[getFigureFromB(player2)].setPos(0);
                            continue;
                        } else // Falls auf A ein Gegner ist, schicke diesen in sein B feld und setze einen aus
                               // eigenem auf A
                        {
                            player1[enemyIsNotOnA(player2)].setPos(-1);
                            player2[getFigureFromB(player2)].setPos(0);
                            continue;
                        }
                    } else {
                        if (!makeMove(player2, roll)) {
                            continue;
                        } // Bewege nächstbesten
                    }
                }
            } else { // Wenn was anderes als eine 6 gewürfelt wird
                if (AIsFree(player2) != 10) { // Falls Sein A Feld von eigenen Steinen belegt ist
                    if (Move(player2[AIsFree(player2)], roll))
                        continue; // Falls Stein auf A wird dieser um 6 bewegt, wenn nicht möglich dann wird der
                                  // nächstbeste bewegt
                    else if (!makeMove(player2, roll)) {
                        endTurn();
                        continue;
                    }
                } else {
                    if (!makeMove(player2, roll)) {
                        endTurn();
                        continue;
                    }
                }
            }
            if (roll != 6)
                endTurn();
        }
        if (hasWon() >= 10)
            gameIsFinished = true;
    }
    return new gameresult(turn, ((hasWon() == 10) ? 1 : 0), beginner);
}
```

```
private int hasWon() { // Gibt aus welcher Spieler alle 4 Figuren im Endfeld hat (10 - Spieler1, 20 -
                       // Spieler2, 1 - Niemand)
    boolean winPlayer1 = true;
    for (int i = 0; i < player1.length; i++) {
        if (player1[i].getPos() < 40)
            winPlayer1 = false;
    }
    if (winPlayer1)
        return 10;
    boolean winPlayer2 = true;
    for (int i = 0; i < player2.length; i++) {
        if (player2[i].getPos() < 40)
            winPlayer2 = false;
    }
    if (winPlayer2)
        return 20;
    return 1;
}
```

```
public void endTurn() { // Tauscht einfach die Startbedingung
    player1Turn = (player1Turn) ? false : true;
}

public boolean makeMove(figur[] player, int step) {
    // Versucht zunächst den vordersten zu Bewegen
    figur[] temp = player.clone();
    int front = getfurthestFigure(temp);
    if (Move(player[front], step)) // Sucht den der am weitesten vorne ist.
        return true;
    else {
        figur[] temp2 = removeElement(temp, front); // Falls nicht möglich suche den 2. weitesten
        front = getfurthestFigure(temp2);
        if (Move(player[getIndexFromPos(player, temp2[front].getPos())], step))
            return true;
        else {
            figur[] temp3 = removeElement(temp2, front); // Falls nicht möglich suche den 3. weitesten
            front = getfurthestFigure(temp3);
            if (Move(player[getIndexFromPos(player, temp3[front].getPos())], step))
                return true;
            else {
                figur[] temp4 = removeElement(temp3, front); // Falls nicht möglich suche den 4. weitesten
                front = getfurthestFigure(temp4);
                if (Move(player[getIndexFromPos(player, temp4[front].getPos())], step))
                    return true;
                else {
                    return false; // Mit Rekursion wäre es schöner aber da es immer 4 Figuren sind ist es in
                                  // diesem Fall akzeptabel
                }
            }
        }
    }
}

public int getIndexFromPos(figur[] player, int pos) { // Gibt den Index einer Figur aus, falls diese sich auf der
                                                      // angegebenen Position befindet
    for (int j = 0; j < player.length; j++) {
        if (player[j].getPos() == pos)
            return j;
    }
    return 10; // befindet sich nicht auf dieser Position
}
public figur[] removeElement(figur[] arr, int index) { // Entfernt einfach ein Element aus einem Array anhand des
                                                       // Indexes
    if (arr == null || index < 0 || index >= arr.length)
        return arr;
    figur[] temp = new figur[arr.length - 1];
    for (int i = 0, j = 0; i < arr.length; i++) {
        if (i == index)
            continue;
        temp[j++] = arr[i];
    }
    return temp;
}
```

```
public int enemyIsNotOnA(figur[] player) {
    for (int i = 0; i < player.length; i++) {
        if ((player[i].getPos() + 20) % 40 == 0) // Schaut ob sich ein gegner auf dem eigenen A Feld befindet
            return i;
    }
    return 10; // Falls sich kein Gegner auf dem eigenen A Feld befindet
}

public int getFigureFromB(figur[] player) {
    for (int i = 0; i < player.length; i++) {
        if (player[i].getPos() == -1)
            return i; // Holt eine beliebige Figur die sich im B Feld befindet
    }
    return 10; // Falls B Feld leer
}
public boolean Move(figur a, int step) {
    if (a.getPos() > -1) { // Figur muss im Feld sein
        if (a.getPos() + step > 43) // Falls man über das Ziel hinausgeht -> Zug nicht möglich
            return false;
        if (a.isPlayerOne()) {
            if (isInGoal(player1, a.getPos() + step)) // Falls ein Mitspieler schon im Ziel dieser Stelle ist -> Zug
                                                      // nicht möglich
                return false;
            for (int i = 0; i < player1.length; i++) {
                if (a.getPos() != player1[i].getPos()) {
                    if (willCollide(a, player1[i], step) == 0) { // Falls man mit eigenem Stein kollidiert kann
                                                                 // man nicht ziehen -> Zug nicht möglich
                        return false;
                    }
                }
            }
            for (int i = 0; i < player2.length; i++) {
                if (willCollide(a, player2[i], step) == 1) { // Falls man mit gegnerischen Stein kollidiert kann man
                                                             // ziehen
                    a.setPosInSteps(step); // Falls man auf dem Feld des Gegners landet, wird dieser nach B
                                           // geschickt
                    player2[i].setPos(-1);
                    return true;
                }
            }
            a.setPosInSteps(step); // Falls alle obigen abfragen durchgehen, kann der Stein ohne Umwege nach
                                   // vorne gesetzt werden
            return true;
        }
        if (!a.isPlayerOne()) {
            if (isInGoal(player2, a.getPos() + step)) // Falls ein Mitspieler schon im Ziel dieser Stelle ist -> Zug
                                                      // nicht möglich
                return false;
            for (int i = 0; i < player2.length; i++) {
                if (a.getPos() != player2[i].getPos()) {
                    if (willCollide(a, player2[i], step) == 0) { // Falls man mit eigenem Stein kollidiert kann
                                                                 // man nicht ziehen -> Zug nicht möglich
                        return false;
                    }
                }
            }
            for (int i = 0; i < player1.length; i++) {
                if (willCollide(a, player1[i], step) == 1) { // Falls man mit gegnerischen Stein kollidiert kann man
                                                             // ziehen
                    a.setPosInSteps(step); // Falls man auf dem Feld des Gegners landet, wird dieser nach B
                                           // geschickt
                    player1[i].setPos(-1);
                    return true;
                }
            }
            a.setPosInSteps(step); // Falls alle obigen abfragen durchgehen, kann der Stein ohne Umwege nach
                                   // vorne gesetzt werden
            return true;
        }
    }
    return false;
}
```

```
    public int AIsFree(figur[] player) { // Ist das Feld A Frei
        for (int i = 0; i < player.length; i++) {
            if (player[i].getPos() == 0)
                return i; // Fall nein gib den Index der Figur die auf A steht
        }
        return 10;
    }

    public boolean isInGoal(figur[] player, int pos) { // Schaut ob ein Mitspieler schon im Ziel an dieser Position ist
        for (int i = 0; i < player.length; i++) {
            if (player[i].getPos() == pos)
                return true;
        }
        return false;
    }

    public int getfurthestFigure(figur[] player) { // Holt den Spieler, der am weitesten Vorne steht
        int largest = 0;
        int largestIndex = 0;
        for (int i = 0; i < player.length; i++) {
            if (player[i].getPos() >= largest) {
                largest = player[i].getPos();
                largestIndex = i;
            }
        }
        return largestIndex;
    }

    public void determineBeginner(wuerfel[] dicesOfEachPlayer) { // kann einfach ignoriert werden, da die Regel
                                                                 // überschrieben wurde, dass der Spieler anfängt der
                                                                 // eine höhere Zahl geworfen hat. Stattdessen fängt
                                                                 // einfach jede zweite Runde ein anderer an, somit
                                                                 // fängt 50% der Zeit der Spieler1 an und 50% der Zeit
                                                                 // Spieler 2.
        if (dicesOfEachPlayer[0].getRandomValue() > dicesOfEachPlayer[0].getRandomValue()) {
            player1Turn = true;
        } else if (dicesOfEachPlayer[0].getRandomValue() < dicesOfEachPlayer[0].getRandomValue()) {
            player1Turn = false;
        } else if (dicesOfEachPlayer[0].getRandomValue() == dicesOfEachPlayer[0].getRandomValue())
            determineBeginner(dicesOfEachPlayer);
        for (int i = 0; i < 4; i++) { // Initialisierung der Figuren
            player1[i] = new figur(true);
            player1[i].setPos(-1);
            player2[i] = new figur(!true);
            player2[i].setPos(-1);
        }
    }

    public int willCollide(figur a, figur b, int step) { // Mit; 0: Eigene, 1: Gegner, 10: Niemandem
        if (b.getPos() < 0)
            return 10; // Man kann nicht mir einem Gegner kollidieren, des in seinem B Feld steht
        if ((a.isPlayerOne() && b.isPlayerOne()) || (!a.isPlayerOne() && !b.isPlayerOne()))
            if (a.getPos() + step == b.getPos()) // Kollidieren mit eigenen
                return 0;
        if (a.isPlayerOne() && !b.isPlayerOne())
            if ((a.getPos() + step == (b.getPos() + 20) % 40)) // Kollidieren mit gegner
                return 1;
        if (!a.isPlayerOne() && b.isPlayerOne())
            if ((a.getPos() + step == (b.getPos() + 20) % 40)) // Kollidieren mit gegner
                return 1;
        return 10;
    }
}
```