Aufgabe 3: Tobis Turnier

Team-ID: 00129

Team: PIE_Team

Bearbeiter dieser Aufgabe:
Jonathan Busch

5. November 2020

Inhaltsverzeichnis

Lösungsidee

Die verschiedenen Turniervarianten werden mit Hilfe eines Zufallszahlengenerators simuliert. Es werden so lange Simulationen durchgeführt, bis das Programm abgebrochen wird. Dieses Ereignis wird dem Programm über ein Fenster mit Button übermittelt.

Es wird außerdem eine weitere, nicht vorgeschlagene, Turniervariante verwendet: Liga-x5. Dabei wird (wie bei K. O. x5) jedes Spiel 5 mal ausgetragen, um den Einzelsieger zu bestimmen.

Umsetzung

Zur Erzeugung von ganzen Zufallszahlen wird die Klasse `java.util.Random` verwendet. Da dieser Pseudozufallszahlengenerator exakte Gleichverteilung der Wahrscheinlichkeiten nur bei 2er-Potenzen erreicht, werden mit diesem Generator direkt nur Zufallszahlen mit der nächstgrößeren 2er-Potenz über der angeforderten Größenbegrenzung generiert. Als generierte Zufallszahl im angeforderten Bereich wird die erste Zahl zurückgeliefert, die im Bereich ist. Da jede Zahl im 2er-Potenzbereich die gleiche Wahrscheinlichkeit hat, hat auch jede Zahl im angeforderten Bereich die gleiche Wahrscheinlichkeit, zurückgegeben zu werden.

Um einen Liga-Durchgang zu simulieren, wird in einem Array der Punktestand von jedem Spieler festgehalten. Während die Spiele durchgeführt werden, wird in einer anderen Variablen der nach Zwischenergebnissen beste Spieler gespeichert.

Um einen K. O. Durchgang zu simulieren, wird ein binärer Baum verwendet. Dieser besteht im Speicher aus einem Array mit doppelt so vielen Einträgen wie es Spieler gibt. Die Startverteilung der Spieler auf Paare steht in der hinteren Hälfte, wobei immer zwei Spieler hintereinander stehen,

die gegeneinander spielen. Den nächsten Platz in einer Runde, an den ein Spieler kommt, wenn er gewinnt, errechnet man einfach, indem man seine bisherige Position im Baumarray durch 2 teilt und abrundet. Im folgenden Beispiel ist in dem Baum an jedem Platz der Arrayindex eingetragen, an dem der Spieler gespeichert wird.

<table>
<tr><td colspan="16">1</td></tr>
<tr><td colspan="8">2</td><td colspan="8">3</td></tr>
<tr><td colspan="4">4</td><td colspan="4">5</td><td colspan="4">6</td><td colspan="4">7</td></tr>
<tr><td colspan="2">8</td><td colspan="2">9</td><td colspan="2">10</td><td colspan="2">11</td><td colspan="2">12</td><td colspan="2">13</td><td colspan="2">14</td><td colspan="2">15</td></tr>
<tr><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
</table>

Der Gewinner steht am Ende an Position 1 im Array.

Beispiele

```
$ java A3Main
8
0
10
20
30
40
50
60
100
Anzahl:  3407872 Simulationen
Liga:    0.3460071270282452
Liga x5: 0.5296017573429987
K.O.:    0.4035800053523137
K.O. x5: 0.6017215435321515

$ java A3Main
8
10
10
10
10
80
80
80
100
Anzahl:  3465216 Simulationen
Liga:    0.20961521590573287
Liga x5: 0.22549820848108748
K.O.:    0.30188882886377066
K.O. x5: 0.35888441009160754

$ java A3Main
16
22
```

```
38
66
93
51
51
58
67
51
57
57
60
73
13
41
42
Anzahl:  3440640 Simulationen
Liga:    0.3158839634486607
Liga x5: 0.5193336123511905
K.O.:    0.1668736049107143
K.O. x5: 0.27844092959449407

$ java A3Main
16
100
95
95
95
95
95
95
95
95
95
95
95
95
95
95
95
Anzahl:  3457024 Simulationen
Liga:    0.1145563351599526
Liga x5: 0.13108413479339456
K.O.:    0.06902815832345971
K.O. x5: 0.07539143494520142
```

Man erkennt: Wenn die Spieler sehr unterschiedliche Spielstärken haben, ist das K. O. System gut geeignet, um den besten Spieler zu ermitteln. Je ähnlicher die Spielstärken der Spieler aber sind, desto besser funktioniert das Liga-System im Gegensatz zum K. O. System.

Generell erhöht es die Gewinnwahrscheinlichkeit des stärksten Spielers, Einzelspiele jeweils zu wiederholen. Das ist aber in der Praxis mit viel Aufwand verbunden, da proportional mehr Spiele gespielt werden müssen.

Quellcode

```
...
public final class A3Main {
  private static volatile boolean running = true;

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // Spielstärken einlesen
    ...
    sc.close();
    Random rnd = new Random();
    // Zufallsfunktionen für das Match
    IntBinaryOperator singleMatch = (a, b) -> nextBoolean(rnd, a, b) ? 1 : 0;
    IntBinaryOperator x5Match = (a, b) -> {
      int w = 0;
      for (int i = 0; i < 5; ++i) if (nextBoolean(rnd, a, b)) ++w;
      return w > 2 ? 1 : 0;
    };
    // Simulationen und Gewinne zählen
    int num;
    int divCount = 0;
    int divX5Count = 0;
    int koCount = 0;
    int koX5Count = 0;
    int[] copy = new int[players.length];
    final int buffer = 1 << 13;
    // Fenster erstellen
    ...
    // Simulationen
    for (int i = 0; i < copy.length; ++i) copy[i] = i;
    for (num = 0; running; num += buffer) {
      for (int i = 0; i < buffer; ++i) {
        // Liga simulieren
        if (simulateDivision(singleMatch, players) == strongest) ++divCount;
        if (simulateDivision(x5Match, players) == strongest) ++divX5Count;
        // zufällige Startverteilung für K. O. erstellen
        for (int j = 0; j < copy.length; ++j) {
          int k = nextInt(rnd, copy.length);
          int tmp = copy[j];
          copy[j] = copy[k];
          copy[k] = tmp;
        }
        // K. O. simulieren
        if (simulateKO(singleMatch, players, copy) == strongest) ++koCount;
        if (simulateKO(x5Match, players, copy) == strongest) ++koX5Count;
      }
      btn.setText(Integer.toString(num));
    }
    // Ergebnis ausgeben
    double numD = num;
    System.out.println("Anzahl:  " + num + " Simulationen");
    System.out.println("Liga:    " + divCount / numD);
    System.out.println("Liga x5: " + divX5Count / numD);
    System.out.println("K.O.:    " + koCount / numD);
    System.out.println("K.O. x5: " + koX5Count / numD);
  }

  // Liga simulieren
  public static int simulateDivision(IntBinaryOperator rnd, int[] players) {
    int[] score = new int[players.length]; // Punktzahlen
    int max = 0;
    int index = 0; // Gewinner laufend ermitteln
```

```
    for (int a = 0; a < players.length; ++a) for (int b = a + 1; b < players.length; ++b) { //
                              für alle Paare
      int winner = rnd.applyAsInt(players[a], players[b]) != 0 ? a : b;
      ++score[winner]; // Gewinner bekommt 1 Punkt
      if (score[winner] > max || (score[winner] == max && winner < index)) {
        max = score[winner]; // Gewinner mit höchster Punktzahl aktualisieren
        index = winner;
      }
    }
    return index;
  }

  // KO simulieren, rnd: „Zufallsoperator“ mit Wahrscheinlichkeiten, der bestimmt, ob 1 oder 5
                              Matches gespielt werden
  public static int simulateKO(IntBinaryOperator rnd, int[] players, int[] startDistribution) {
    int numPlayers = Integer.highestOneBit(players.length); // 2er-Potenz erzwingen
    // binärer Baum
    int[] koRounds = new int[numPlayers * 2];
    for (int i = 0; i < numPlayers; ++i) {
      koRounds[i + numPlayers] = startDistribution[i]; // initialisieren mit Startverteilung
    }
    for (int nP = numPlayers; nP > 1; nP /= 2) { // für alle Runden
      for (int i = nP / 2; i < nP; ++i) { // für alle Einzelmatches
        int a = koRounds[i * 2]; // Spieler 1
        int b = koRounds[i * 2 + 1]; // Spieler 2
        koRounds[i] = rnd.applyAsInt(players[a], players[b]) != 0 ? a : b; // Gewinner kommt in
                              die nächste Runde
      }
    }
    return koRounds[1];
  }

  // nextBoolean() für nicht-gleiche Wahrscheinlichkeitsverteilung
  private static boolean nextBoolean(Random rnd, int probTrue, int probFalse) {
    return nextInt(rnd, probTrue + probFalse) < probTrue;
  }

  // für exakte Wahrscheinlichkeitsverteilung:
  // Random kann nur 1:1 Wahrscheinlichkeiten generieren, also exakt gleiche Verteilungen nur bei
                              Bounds der Form 2^n.
  private static int nextInt(Random rnd, int bound) {
    int eb = Integer.highestOneBit(bound);
    if (eb < bound) eb <<= 1;
    // 2^n Bound-Wert für gleiche Verteilung
    int rs;
    do {
      rs = rnd.nextInt(eb);
      // nur Werte < bound werden akzeptiert
    } while (rs >= bound);
    return rs;
  }
}
```