# Lösungsweg

Die Idee war es, für jede Turnierart eine Funktion zu schreiben, die das Turnier einmal durchspielt und dann den Gewinner zurückgibt. Diese Funktionen können dann jeweils mehrmals aufgerufen werden und die Wahrscheinlichkeit für den besten Spieler zu gewinnen kann berechnet werden. Anhand dieser Wahrscheinlichkeit kann herausgefunden werden, welche Methode sich am besten eignet.

# Umsetzung

## Allgemeine Methoden

public static bool makeGame(int[] spielstärken, Tuple<int, int> spieler, Random rnd) {

int rn = rnd.Next(0, spielstärken[spieler.Item1] + spielstärken[spieler.Item2]);

bool ret = rn < spielstärken[spieler.Item1];

return ret;

}

Die Funktion makeGame würfelt eine zufälligr Zahl zwischen 0 und der addierten Spielstärke der entsprechenden Spieler aus und gibt true zurück, wenn der erste, und false zurück, wenn der zweite Spieler gewonnen hat. In einer Variante der Methode wird dieses Prozedere 5 mal ausgeführt und der Spieler als Sieger ausgegeben, der am öftesten gewonnen hat.

## Liga

Beim Ligaspiel spielt jeder Spieler mit jedem anderen Spieler. Der Spieler mit den meisten Gewinnen ist der Sieger. Dazu iteriert das Programm über alle Spieler, prüft, dass die beiden Kontrahenten nur einmal gegeneinander antreten (j<i) und berechnet dann ein zufälliges Spielergebnis mit der makeGame Funktion. Der Eintrag des jeweiligen Siegers in dem gewinner array wird hochgezählt. Der Spieler mit den meisten Siegen wird zurückgegeben.

public static int liga(int[] spielstärken) {

Random rnd = new Random();

int[] gewinne = new int[spielstärken.Length];

for (int i = 0; i < spielstärken.Length; i++) {

for (int j = 0; j < spielstärken.Length; j++) {

if (i != j && j < i) {

if (makeGame(spielstärken, new Tuple<int, int>(i, j), rnd)) {

gewinne[i]++;

}

else {

gewinne[j]++;

}

}

}

}

Console.WriteLine(getHighest(gewinne));

return getHighest(gewinne);

}

## kO Spiel

List<Tuple<int, int>> ausstehendeSpiele = new List<Tuple<int, int>>();

if (kombimodus == 0) {

for (int i = 0; i < spielstärken.Length; i += 2) {

ausstehendeSpiele.Add(new Tuple<int, int>(i, i + 1));

}

}

else {

for (int i = 0; i < spielstärken.Length / 2; i++) {

ausstehendeSpiele.Add(new Tuple<int, int>(i, spielstärken.Length - i - 1));

}

}

Zuerst wird die Kombination der Gegner errechnet. Das haben wir auf zwei Weisen implementiert: kombimodus 0 ist eine Möglichkeit, wo 1 gegen 2, 3 gegen 4, usw. spielt; kombimodus 1 wäre bei 8 Spielern 0 gegen 7, 1 gegen 6, usw.

Nun betrachten wir die Vorfinalspiele und das Finale gesondert.

List<Tuple<int, int>> ausstehendeSpieleNeu = new List<Tuple<int, int>>();

for (int i = 0; i < Math.Log(spielstärken.Length, 2) - 1; i++) {

for (int j = 0; j < ausstehendeSpiele.Count; j += 2) {

int spieler1neu; int spieler2neu;

if (times5) {

spieler1neu = makeGamex5(spielstärken, ausstehendeSpiele[j], rnd) ? ausstehendeSpiele[j].Item1 : ausstehendeSpiele[j].Item2;

spieler2neu = makeGamex5(spielstärken, ausstehendeSpiele[j + 1], rnd) ? ausstehendeSpiele[j + 1].Item1 : ausstehendeSpiele[j + 1].Item2;

}

else {

spieler1neu = makeGame(spielstärken, ausstehendeSpiele[j], rnd) ? ausstehendeSpiele[j].Item1 : ausstehendeSpiele[j].Item2;

spieler2neu = makeGame(spielstärken, ausstehendeSpiele[j + 1], rnd) ? ausstehendeSpiele[j + 1].Item1 : ausstehendeSpiele[j + 1].Item2;

}

ausstehendeSpieleNeu.Add(new Tuple<int, int>(spieler1neu, spieler2neu));

}

printAusstehende(ausstehendeSpiele);

ausstehendeSpiele.Clear();

ausstehendeSpiele = new List<Tuple<int, int>>(ausstehendeSpieleNeu);

ausstehendeSpieleNeu.Clear();

}

In den Vorfinalspielen werden für jede Wettbewerbsebene (i-Schleife) nacheinander jeweils zwei Spielergebnisse berechnet. Zuerst betrachten wir also die Pärchen (1 2) und (3 4), die in ausstehendeSpiele stehen. Die jeweiligen Gewinner werden dann als neues Pärchen in die ausstehendeSpieleNeu Liste eingetragen. Das wird solange widerholt, bis die aktuelle Wettbewerbsebene abgearbeitet ist. Dann wird ausstehendeSpieleNeu in ausstehendeSpiele umgefüllt und die nächste Ebene abgearbeitet. Um auch die kOx5 Variante zu ermöglichen kann man über die boolsche times5 Variable steuern, ob jedes Spiel ein- oder fünfmal gespielt werden soll.

Im Finale wird der Sieger zurückgegeben.

//finale:

printAusstehende(ausstehendeSpiele);

if (times5) {

return makeGamex5(spielstärken, ausstehendeSpiele[0], rnd) ? ausstehendeSpiele[0].Item1 : ausstehendeSpiele[0].Item2;

}

else {

return makeGame(spielstärken, ausstehendeSpiele[0], rnd) ? ausstehendeSpiele[0].Item1 : ausstehendeSpiele[0].Item2;

}

## runMult

public static float runMult(int[] spielstärken, int maxNum, int modus) {

int victories = 0;

int bestManInd = getHighest(spielstärken);

for (int i = 0; i < maxNum; i++) {

int erg = modus==0 ? liga(spielstärken) : kO(spielstärken, (modus==2));

if (erg == bestManInd) {

victories++;

}

}

Console.WriteLine("{0} : {1}", victories, maxNum);

return ((victories) / maxNum);

}

in runMult wird ein Turnierdurchlauf in einem der Modi (liga, kO, …) so oft wie angegeben wiederholt und das Verhältnis von Siegen des besten Spielers zu den Gesamtspielen zurückgegeben.

## Ergebnisse

Hier die Ausgabe des Programms. Es berechnet nacheinander eine Lösung für Liga, kO und kOx5 jeweils mit verschiedenen Durchlaufgrößen. kO und kOx5 werden jeweils im kombimodus 0 und 1 berechnet. Ausgegeben werden immer die Anzahl der Siege des Besten Spieler, Gesamtanzahl der Spiele, sowie den Anteil der Siege an den Gesamtdurchläufen in Prozent. (Ergebnisse siehe nächste Seite)

## Fazit/Empfehlung

Je weniger Teilnehmer das Turnier hat und je stärker sich die Spielstärke des besten Spieler von den anderen abhebt, desto zuverlässiger ist die kOx5 Methode. Wenn das Turnier viele und oder ähnlich gute Teilnehmer hat, ist die liga Variante die Beste. Welche Spielerkombination bei einem kO Turnier am besten ist, ist schwer zu bestimmen und bedarf weiterer Untersuchung; grade in Zusammenhang mit Abhängigkeiten von anderen Einflüssen, wie zum Beispiel der Sortierung der Spielstärke Liste.

Dieses Fazit bezieht sich auf die auf das resultierende Sieg-Spiel-Verhältnis bei den jeweils meisten Durchläufen (107082), da wir diese Zahl, nach dem Gesetz der großen Zahlen, für am Aussagekräftigsten halten.

## Spielstaerken4

100

95

95

95

95

95

95

95

95

95

95

95

95

95

95

95

Modus: **liga**

1 von 10 : 10,00 %

2 von 31 : 6,45 %

10 von 172 : 5,81 %

263 von 2255 : 11,66 %

12132 von 107082 : **11,33 %**

Modus: **kO**

Kombimodus: 0

0 von 10 : 0,00 %

4 von 31 : 12,90 %

9 von 172 : 5,23 %

141 von 2255 : 6,25 %

7539 von 107082 : **7,04 %**

Kombimodus: 1

0 von 10 : 0,00 %

3 von 31 : 9,68 %

12 von 172 : 6,98 %

168 von 2255 : 7,45 %

7581 von 107082 : **7,08 %**

Modus: **kOx5**

Kombimodus: 0

0 von 10 : 0,00 %

1 von 31 : 3,23 %

16 von 172 : 9,30 %

154 von 2255 : 6,83 %

8142 von 107082 : **7,60 %**

Kombimodus: 1

1 von 10 : 10,00 %

7 von 31 : 22,58 %

13 von 172 : 7,56 %

169 von 2255 : 7,49 %

8078 von 107082 : **7,54 %**

## Spielstaerken3

22

38

66

93

51

51

58

67

51

57

57

60

73

13

41

42

Modus: **liga**

1 von 10 : 10,00 %

12 von 31 : 38,71 %

50 von 172 : 29,07 %

679 von 2255 : 30,11 %

33685 von 107082 : **31,46 %**

Modus: **kO**

Kombimodus: 0

2 von 10 : 20,00 %

5 von 31 : 16,13 %

31 von 172 : 18,02 %

410 von 2255 : 18,18 %

17794 von 107082 : **16,62 %**

Kombimodus: 1

0 von 10 : 0,00 %

4 von 31 : 12,90 %

19 von 172 : 11,05 %

304 von 2255 : 13,48 %

16778 von 107082 : **15,67 %**

Modus: **kOx5**

Kombimodus: 0

5 von 10 : 50,00 %

9 von 31 : 29,03 %

54 von 172 : 31,40 %

611 von 2255 : 27,10 %

29768 von 107082 : **27,80 %**

Kombimodus: 1

2 von 10 : 20,00 %

5 von 31 : 16,13 %

44 von 172 : 25,58 %

560 von 2255 : 24,83 %

26008 von 107082 : **24,29 %**

## Spielstaerken2

10

10

10

10

80

80

80

100

Modus: **liga**

3 von 10 : 30,00 %

8 von 31 : 25,81 %

44 von 172 : 25,58 %

475 von 2255 : 21,06 %

22528 von 107082 : **21,04 %**

Modus: **kO**

Kombimodus: 0

2 von 10 : 20,00 %

6 von 31 : 19,35 %

52 von 172 : 30,23 %

643 von 2255 : 28,51 %

30189 von 107082 : **28,19 %**

Kombimodus: 1

1 von 10 : 10,00 %

9 von 31 : 29,03 %

57 von 172 : 33,14 %

678 von 2255 : 30,07 %

32834 von 107082 : **30,66 %**

Modus: **kOx5**

Kombimodus: 0

3 von 10 : 30,00 %

13 von 31 : 41,94 %

63 von 172 : 36,63 %

828 von 2255 : 36,72 %

38352 von 107082 : **35,82 %**

Kombimodus: 1

6 von 10 : 60,00 %

11 von 31 : 35,48 %

51 von 172 : 29,65 %

800 von 2255 : 35,48 %

38919 von 107082 : **36,35 %**

## Spielstaerken1

0

10

20

30

40

50

60

100

Modus: **liga**

6 von 10 : 60,00 %

14 von 31 : 45,16 %

54 von 172 : 31,40 %

816 von 2255 : 36,19 %

37247 von 107082 : **34,78 %**

Modus: **kO**

Kombimodus: 0

3 von 10 : 30,00 %

12 von 31 : 38,71 %

65 von 172 : 37,79 %

770 von 2255 : 34,15 %

38093 von 107082 : **35,57 %**

Kombimodus: 1

4 von 10 : 40,00 %

13 von 31 : 41,94 %

74 von 172 : 43,02 %

1062 von 2255 : 47,10 %

51064 von 107082 : **47,69 %**

Modus: **kOx5**

Kombimodus: 0

6 von 10 : 60,00 %

20 von 31 : 64,52 %

92 von 172 : 53,49 %

1209 von 2255 : 53,61 %

59538 von 107082 : **55,60 %**

Kombimodus: 1

6 von 10 : 60,00 %

19 von 31 : 61,29 %

97 von 172 : 56,40 %

1370 von 2255 : 60,75 %

65141 von 107082 : **60,83 %**