Aufgabe 3: Tobis Turnier

Team-ID: 00087

Team-Name: One Man Army

Bearbeiter dieser Aufgabe:
Merlin Moelter

8. September 2020

Inhaltsverzeichnis

Lösungsidee

Im Wesentlichen sollen alle Turnier-Varianten simuliert, und nach jeder Simulation der Gewinner aufgeschrieben werden. Nachdem eine Variante zum Beispiel 100.000 mal simuliert wurde, also 100.000 Gewinner aufgeschrieben wurden, teilt man die Anzahl der Turniere, bei denen der stärkste Spieler gewonnen hat durch 100.000 und erhält somit die Durchschnittliche Anzahl an Turnieren, bei denen der stärkste Spieler gewinnt.

Partie

Um eine Partie des Spieles zwischen zwei Spielern zu simulieren, wird eine Zufallszahl zwischen 0 und 1 generiert. Ist diese Zahl kleiner gleich dem Anteil der Spielstärke des ersten Spielers an der Spielstärke beider Spieler zusammen, so hat der erste Spieler gewonnen, ansonsten der zweite.

Liga

Bei der Simulation der Turniervariante „Liga“ wird zuerst eine Map mit der Spielernummer als Schlüssel und der Anzahl an gewonnenen Partien als Wert erstellt, bei der jeder Spieler den Anfangswert 0 besitzt. Nun wird in zwei verschachtelten Schleifen zweimal über alle Spieler iteriert und für jedes Spieler – Spieler paar, sofern es nicht dieselben sind, eine Partie durchgeführt. Der Gewinner erhält einen Punkt, also wird sein Eintrag in der Spieler – Gewinne Map um eins erhöht. Wurden alle Partien gespielt, wird der Spieler mit dem höchsten Wert in der Spieler – Gewinne Map bestimmt und zurückgegeben.

K.O.

Zuerst wird eine neue Liste erstellt, die alle Spieler in zufälliger Reihenfolge beinhaltet. Die Länge dieser Liste muss eine zweier Potenz sein (2, 4, 8, 16, …), da sich die Spieleranzahl in jeder Runde halbiert und am Ende eins betragen muss. Also iteriert die erste Schleife $\sqrt{Anzahl\ der\ Spieler}$ mal über die zweite Schleife, die über alle verbleibenden Spieler im Turnier mit einer Schrittlänge von zwei iteriert (sie überspringt jeden zweiten Spieler, da immer zwei gegeneinander spielen). In der ersten Schleife wird eine Liste erstellt, in der die Gewinner der Partien gespeichert werden sollen. In der zweiten Schleife wird eine Partie zwischen dem Spieler an dem sich die Schleife gerade befindet und dem nächsten Spieler simuliert, und der Gewinner zur Liste der Gewinner hinzugefügt. Wurden alle Partien einer Iteration gespielt, so wird die Liste der verbleidenden Spieler mit der Liste der Gewinner dieser Iteration überschrieben. Wenn die erste Schleife durchgelaufen ist, also nur noch ein Spieler übrig ist, wird dieser zurückgegeben.

K.O. x 5

Diese Variante funktioniert sehr ähnlich wie die Variante „K.O.“. Anstatt nur eine Partie durchzuführen und den Gewinner dieser Partie in die Liste der Gewinner einzutragen, werden fünf Partien gespielt. Die fünf Partien werden ähnlich zur Variante „Liga“ simuliert, aber anstatt verschiedene Spieler spielen zu lassen, spielen hier fünf Mal dieselben Spieler gegeneinander. Der Gewinner einer Partie ist derjenige Spieler, dessen Wert in der Spieler – Gewinne Map höher ist. Er wird zur Liste der Gewinner hinzugefügt.

Gewinnrate

Die Gewinnrate einer Turnier-Variante ist die relative Anzahl an Partien, in denen der stärkste Spieler gewinnt.

Zuerst wird der Wert der Spiele, die der stärkste Spieler gewonnen hat, auf null gesetzt. Dann iteriert eine Schleife n-mal (wobei n ein möglichst hoher Wert sein sollte) und simuliert bei jeder Iteration einen Ablauf des Turnieres. Wenn der stärkste Spieler bei diesem Durchlauf gewonnen hat, wird der Zähler um eins erhöht. Ist die Schleife beendet, wird das Verhältnis der Anzahl an Spielen, in denen der stärkste Spieler gewonnen hat zu der Gesamtanzahl an Spielen zurückgegeben.

Umsetzung

Für jede Simulation (Turnier-Varianten, Partie) beinhaltet mein Programm eine Funktion, die diese Simulation durchführt. Der Ablauf der Simulationen wurde im Abschnitt „Lösungsidee“ ausreichend beschrieben, weshalb ich darauf nicht weiter eingehen werde.

Zur Generierung der Zufallszahlen verwendet mein Programm die native Javascript-Funktion „Math.random“.

Beispiele

Das Skript nimmt die Nummer der Beispieldatei als Parameter.

Alle Beispiele wurden mit 100.000 (10^5) Iterationen durchgeführt.

$ node index.js 1

Die Gewinnrate für 'Liga' beträgt 34%

Die Gewinnrate für 'KO' beträgt 41%

Die Gewinnrate für 'KOx5' beträgt 60%

$ node index.js 2

Die Gewinnrate für 'Liga' beträgt 21%

Die Gewinnrate für 'KO' beträgt 30%

Die Gewinnrate für 'KOx5' beträgt 36%

$ node index.js 3

Die Gewinnrate für 'Liga' beträgt 32%

Die Gewinnrate für 'KO' beträgt 16%

Die Gewinnrate für 'KOx5' beträgt 28%

$ node index.js 4

Die Gewinnrate für 'Liga' beträgt 12%

Die Gewinnrate für 'KO' beträgt 7%

Die Gewinnrate für 'KOx5' beträgt 7%

Empfehlung

Bei einer ausgeglichenen Verteilung der Spielstärken, wie sie bei dem Beispiel 1 gegeben ist, bietet die Variante „K.O. x 5“ die höchste Gewinnrate. Sind die Spielstärken sehr nah aneinander, wie es bei dem Beispiel 4 der Fall ist, so bietet „Liga“ die höchste Gewinnrate. Da bei einem Turnier in der echten Welt die Spielstärken auch eher um einen Punkt herum liegen als gleichmäßig verteilt zu sein, empfehle ich die Variante „Liga“.

Quellcode

```
class Player {
    constructor(strength, index) {
        this.strength = strength
        this.index = index
    }
}

/**
 * Textdatei einlesen
 */

// Hier habe ich den langweiligen Teil entfernt

players = players.map((strength, i) => new Player(parseInt(strength), i))

/**
 * Lösung
 */

const randomizeArray = require("../lib/randomizeArray.js")

// Den stärksten Spieler bestimmen
const strongestPlayer = players.reduce((strongest, current) =>
current.strength > strongest.strength ? strongest = current : strongest,
players[0])

// Partie
function simluateMatch(firstPlayer, secondPlayer) {
    return Math.random() <= (firstPlayer.strength / (firstPlayer.strength +
secondPlayer.strength)) ? 0 : 1
}
```

```
// Variante: Liga
function simulateLeague() {
    const playerWins = {}

    for (let i = 0; i < players.length; i++) {
        playerWins[i] = 0
    }

    for (let i = 0; i < players.length; i++) {
        for (let j = 0; j < players.length; j++) {
            if (i === j) {
                break
            }

            const winner = simluateMatch(players[i], players[j])

            if (winner === 0) {
                playerWins[i]++
            } else if(winner === 1) {
                playerWins[j]++
            }
        }
    }

    let winner = players[0]
    for (let index in playerWins) {
        // Hier wird bewusst nicht >= verwendet, da mit > gewährleistet
wird, dass bei gleichen Spielständen
        // der Spieler mit der kleinsten Spielernummer gewinnt
        if (playerWins[index] > playerWins[winner.index]) {
            winner = players[index]
        }
    }

    return winner
}

// Variante: KO
function simulateKO() {
    let tournamentPlayers = randomizeArray(players)

    for (let i = 0; i < Math.sqrt(players.length); i++) {
        const winners = []

        for (let j = 0; j < tournamentPlayers.length; j += 2) {
            const winner = simluateMatch(tournamentPlayers[j],
tournamentPlayers[j + 1])

            if (winner === 0) {
                winners.push(tournamentPlayers[j])
            } else if (winner === 1) {
                winners.push(tournamentPlayers[j + 1])
            }
        }

        tournamentPlayers = winners
    }

    return tournamentPlayers[0]
}
```

```
// Variante: KO x 5
function simulateKOx5() {
    let tournamentPlayers = randomizeArray(players)

    for (let i = 0; i < Math.sqrt(players.length); i++) {
        const winners = []

        for (let j = 0; j < tournamentPlayers.length; j += 2) {
            const playerWins = {
                [j]: 0,
                [j + 1]: 0
            }

            for (let k = 0; k < 5; k++) {
                const winner = simluateMatch(tournamentPlayers[j],
tournamentPlayers[j + 1])

                if (winner === 0) {
                    playerWins[j]++
                } else if (winner === 1) {
                    playerWins[j + 1]++
                }
            }

            if (playerWins[j] > playerWins[j + 1]) {
                winners.push(tournamentPlayers[j])
            } else {
                winners.push(tournamentPlayers[j + 1])
            }
        }

        tournamentPlayers = winners
    }

    return tournamentPlayers[0]
}

// Bestimme, wie oft der stärkste Spieler im Durchschnitt gewinnt
function getTournamentWinrate(tournamentFunction, iterations) {
    let strongestPlayerWins = 0

    for (let i = 0; i < iterations; i++) {
        const winner = tournamentFunction()

        if (winner.index === strongestPlayer.index) {
            strongestPlayerWins++
        }
    }

    return strongestPlayerWins / iterations
}

const iterations = 1e5

const ligaPercentage = getTournamentWinrate(simulateLeague, iterations)
const koPercentage = getTournamentWinrate(simulateKO, iterations)
const kox5Percentage = getTournamentWinrate(simulateKOx5, iterations)

console.log("Die Gewinnrate für 'Liga' beträgt", Math.round(ligaPercentage
* 100) + "%")
```

```
console.log("Die Gewinnrate für 'KO' beträgt", Math.round(koPercentage *
100) + "%")
console.log("Die Gewinnrate für 'KOx5' beträgt", Math.round(kox5Percentage
* 100) + "%")
```