Aufgabe 3

Lösungsidee:

Bei dieser Aufgabe geht es eher um die Erkenntnisse, die in Bezug auf die verschiedenen Turniersysteme gewonnen werden können. Man muss die Siege ausgeben, die verschiedene Spieler mit verschieden Spielstärken über eine bestimmte Anzahl an Turniersimulationen erringen. Dann kann man einen Bezug zwischen Turniersystem, Wiederholungen und Spielstärke herstellen.

Umsetzung:

Zur Lösung dieses Problems habe ich Python gewählt. Die Spielstärken, Wiederholungen und Turniersystem werden als Kommandozeilenargumente eingelesen. Daraufhin wird eine Wiederholung initiiert, die so oft das gewünschte Turniersystem simuliert, wie eingegeben. Dann wird eine Liste angelegt, die so viele Elemente hat, wie Nutzer simuliert werden sollen. Diese Elemente sind Integer, die um 1 erhöht werden, wenn der Spieler, zu dem dieser Integer gehört, ein simuliertes Turnier gewinnt. In der Liga „spielt“ jeder Spieler bzw. seine Spielstärke ein Mal gegen jede andere, außer sich selbst. Dort kann es pro Turnier mehr als einen Sieger geben, anders als in den K.O.-Systemen. Bei K.O wird die Startaufstellung „ausgelost“, wie in einem echten Turnier im K.O.-System. Dann wird das Spiel jeder Paarung ausgeführt und der Sieger kommt in die nächste Runde, woraufhin er gegen den Sieger der benachbarten Paarung spielt, bis nur noch ein Spieler übrig bleibt. Das Selbe gilt für K.O. * 5, jedoch mit 5 Partien pro Paarung, bevor ein Sieger ausgerufen wird.

Beispiele (variieren aus offensichtlichen Gründen zwischen Ausführungen):

Als Beispiele habe ich die gegebenen Beispiele der BWInf-Materialien gewählt:

$ python3 aufgabe3.py spielstaerken1.txt 10 liga liefert*:*

[0, 10, 20, 30, 40, 50, 60, 100] # Dies sind die beteiligten Spieler

[0, 0, 2, 0, 3, 2, 4, 6] # Dies sind die Anzahlen der Siege je Spieler mit gleichem Index

[100] # Dies ist/sind der bzw. die Spieler mit den meisten Siegen

Das waren also 10 Wiederholungen im Liga-System.

$ python3 aufgabe3.py spielstaerken1.txt 100 liga liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 16, 21, 20, 24, 17, 19, 22]

[40]

$ python3 aufgabe3.py spielstaerken1.txt 100000 liga

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 19799, 19682, 19762, 19647, 19686, 19918, 19784]

[60]

Im Liga-System bleiben die Siege bei diesen Spielstärken (abgesehen von der 0) sehr ausgeglichen.

$ python3 aufgabe3.py spielstaerken1.txt 10 ko liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 0, 0, 0, 1, 2, 2, 5]

[100]

$ python3 aufgabe3.py spielstaerken1.txt 100 ko liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 1, 2, 4, 9, 19, 25, 40]

[100]

$ python3 aufgabe3.py spielstaerken1.txt 100000 ko liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 786, 3438, 7064, 11355, 16172, 20722, 40463]

[100]

Im K.O.-System ist die Anzahl der Siege offenbar abhängig von der Spielstärke, mit stärkeren Spielern als häufigeren Siegern.

$ python3 aufgabe3.py spielstaerken1.txt 10 ko5 liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 0, 0, 0, 0, 1, 1, 8]

[100]

$ python3 aufgabe3.py spielstaerken1.txt 100 ko5 liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 0, 0, 3, 6, 17, 18, 56]

[100]

$ python3 aufgabe3.py spielstaerken1.txt 100000 ko5 liefert:

[0, 10, 20, 30, 40, 50, 60, 100]

[0, 14, 472, 2360, 6127, 11970, 18976, 60081]

[100]

Im K.O. * 5 – System lehnt sich die Verteilung der Siege offenbar noch stärker in Richtung der stärkeren Spieler.

$ python3 aufgabe3.py spielstaerken3.txt 10 liga liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[0, 0, 0, 3, 1, 0, 0, 1, 0, 1, 2, 2, 1, 0, 2, 1]

[93]

$ python3 aufgabe3.py spielstaerken3.txt 100 liga liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[9, 7, 6, 19, 4, 13, 6, 8, 4, 4, 13, 11, 5, 6, 7, 2]

[93]

$ python3 aufgabe3.py spielstaerken3.txt 100000 liga

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[8895, 8942, 8728, 8975, 9001, 9013, 9080, 8804, 8866, 8856, 8968, 8792, 52, 8926, 8994, 9000]

[58]

$ python3 aufgabe3.py spielstaerken3.txt 100000 liga liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[8864, 9092, 8844, 9052, 8958, 8918, 8763, 8953, 8750, 8716, 8892, 8859, 8989, 8897, 174, 8768]

[38]

Auch mit diesem Satz an Spielstärken bleibt die Verteilung der Siege in erster Linie sehr ausgeglichen.

$ python3 aufgabe3.py spielstaerken3.txt 10 ko liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[0, 0, 1, 0, 1, 1, 1, 2, 0, 1, 0, 0, 2, 0, 1, 0]

[67, 73]

$ python3 aufgabe3.py spielstaerken3.txt 100 ko liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[3, 1, 8, 11, 3, 4, 13, 11, 3, 6, 7, 9, 15, 0, 4, 2]

[73]

$ python3 aufgabe3.py spielstaerken3.txt 100000 ko liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[648, 2742, 9030, 16756, 5219, 5368, 6931, 9267, 5372, 6695, 6695, 7459, 10965, 125, 3276, 3452]

[93]

K.O. liefert Ergebnisse, die anhand der Spielstärke so auch ungefähr zu erwarten wären, hat jedoch aufgrund der größeren Spielergruppe eine starke Inkonsistenz bei weniger gespielten Runden.

$ python3 aufgabe3.py spielstaerken3.txt 10 ko5 liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 1, 2, 2, 0, 0, 0]

[66, 93, 60, 73]

$ python3 aufgabe3.py spielstaerken3.txt 100 ko5 liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[0, 1, 7, 32, 5, 2, 6, 11, 3, 2, 9, 7, 14, 0, 0, 1]

[93]

$ python3 aufgabe3.py spielstaerken3.txt 100000 ko5 liefert:

[22, 38, 66, 93, 51, 51, 58, 67, 51, 57, 57, 60, 73, 13, 41, 42]

[39, 914, 9623, 27900, 3685, 3580, 6165, 10079, 3553, 5759, 5707, 6730, 13396, 0, 1365, 1505]

[93]

Auch bei K.O. * 5 werden die Beobachtungen der ersten Spielstärken bestätigt.

Alle diese Erkenntnisse lassen sich auch logisch aus dem Gesetz der großen Zahlen bzw. Grundlagen der Wahrscheinlichkeit schlussfolgern.

Wenn Tobi ein möglichst ausgewogenes Turnier haben möchte, sollte er ein Turnier im Ligamodus veranstalten. Sollen die Siege von der Spielstärke abhängig sein, sollte er für maximale Konsistenz K.O. * 5 wählen. Dieser Modus ist zwar konstanter, was die Ergebnisse betrifft, benötigt jedoch auch mehr Zeit in der Ausführung, als sein Pendant, das klassische K.O.-System. Für eine Mischung aus Spielgeschwindigkeit und Sieg des Stärkeren, sollte Tobi dieses auswählen.

Quellcode

```
import sys

from random import random, shuffle

from math import log



def einlesen(anzahl):

    """Liest die Spielstärken ein."""

    # Initialisierung der Liste der Spielstärken ´spieler´

    spieler = []

    # Lies jede Spielstärke ein und speichere jede Spielstärke als ´int´ in ´spieler´

    for i in range(0, anzahl):
```

```
        spieler.append(int(f.readline().strip("\n")))
    return spieler



def liga(spieler):
    """Simuliert ein Turnier im Liga-System"""
    # Initialisierung der Liste der Sieger, da in einer Liga mehrere Spieler gewinnen können
    sieger = []
    for mitlauf in range(0, len(spieler)):
        # Erstell für jeden Spieler eine Liste bestehend aus Spielstärke und Siegen
        spieler[mitlauf] = [spieler[mitlauf], 0]
    # Iteriere jedes Element gegen jedes andere Element
    for i in range(0, len(spieler)):
        for j in range(0, len(spieler)):
            # Der Spieler spielt nicht gegen sich selbst
            if i != j:
                # Ist die Spielstärke eines Spielers ´0´, dann kann dies automatisch als Sieg für
                # den Gegner gewertet werden
                if spieler[j][0] != 0.0 and spieler[j][0] != 0.0:
                    # Erzeuge eine pseudozufällige Zahl
                    zahl = random()
                    # Ist diese kleiner als der Quotient aus der ersten Spielstärke und der Summe
                    # der Spielstärken, werte dies als einen Sieg für den ersten Spieler
                    if zahl < spieler[i][0]/(spieler[i][0]+spieler[j][0]):
                        spieler[i][1] += 1
                    # Sonst, werte dies als Sieg für den zweiten Spieler
                    else:
                        spieler[j][1] += 1
                elif spieler[j][0] == 0.0:
                    spieler[i][1] += 1
                else:
                    spieler[j][1] += 1
    # Ab hier wird nur noch die Anzahl der Siege benötigt, also wird an jedem Index in ´spieler´
    # nur noch diese gespeichert
    for i in range(0, len(spieler)):
        spieler[i] = spieler[i][1]
```

```
    # In ´sieger´ werden alle Indizes gespeichert, an denen die Siege am höchsten sind
    for i, j in enumerate(spieler):
        if j == max(spieler):
            sieger.append(i)
    return sieger


def ko(spieler):
    """Simuliert ein Turnier im K.O.-System"""
    # Initialisierung der Liste der für die nächsten Runde qualifizierten Indizes
    qualifiziert = []
    # Speichere so viele Indizes, wie es Spieler gibt
    for i in range(0, len(spieler)):
        qualifiziert.append(i)
    # Ändere die Reihenfolge der Indizes zu einer zufälligen Abfolge
    shuffle(qualifiziert)
    # Die Anzahl der Runden beträgt in einem K.O.-System immer den Logarithmus
    # zur Basis 2 der Spielerzahl, daher werden so viele Runden simuliert
    for runden in range(0, int(log(len(spieler), 2))):
        # Die Anzahl der Paarungen ist immer halb so hoch wie die Anzahl der Spieler
        for mitlauf in range(0, int(len(qualifiziert)/2)):
            # Die Paarung besteht aus dem Spieler an der Stelle der Mitlaufvariable
            # ´mitlauf´ und dem an der nächsten Stelle
            spielende = [spieler[qualifiziert[mitlauf]],
                         spieler[qualifiziert[mitlauf+1]]]
            # Ist die Spielstärke eines Spielers ´0´, dann kann dies automatisch als Sieg für
            # den Gegner gewertet werden
            if spielende[1] != 0.0 and spielende[0] != 0:
                # Erzeuge eine pseudozufällige Zahl
                zahl = random()
                # Ist diese kleiner als der Quotient aus der ersten Spielstärke und der Summe
                # der Spielstärken, werte dies als einen Sieg für den ersten Spieler
                if zahl < spielende[0]/(spielende[0]+spielende[1]):
                    # Lösche das Element des Verlierers, wodurch die for-Schleife an die richtige Stelle springt
                    del qualifiziert[mitlauf+1]
                # Sonst, werte dies als Sieg für den zweiten Spieler
```

```
            else:
                del qualifiziert[mitlauf]
        elif spielende[0] == 0:
            del qualifiziert[mitlauf]
        else:
            del qualifiziert[mitlauf+1]
    return([qualifiziert[0]])



def ko5(spieler):
    """Genau wie ´ko´, bloß mit 5 Spielen pro Paarung"""
    qualifiziert = []
    for i in range(0, len(spieler)):
        qualifiziert.append(i)
    shuffle(qualifiziert)
    for runden in range(0, int(log(len(spieler), 2))):
        for mitlauf in range(0, int(len(qualifiziert)/2)):
            spielende = [spieler[qualifiziert[mitlauf]],
                         spieler[qualifiziert[mitlauf+1]]]
            # Initialisierung der Siege der Paarung, der Index entspricht
            # dem Index des Spielers in ´spielende´
            siege = [0, 0]
            if spielende[1] != 0.0 and spielende[0] != 0:
                # Die Spielstärken bestreiten 5 'Spiele' gegeneinander, bevor ein
                # Sieger ernannt wird
                for spiele in range(0, 5):
                    if spielende[1] != 0.0 and spielende[0] != 0:
                        zahl = random()
                        if zahl < spielende[0]/(spielende[0]+spielende[1]):
                            siege[0] += 1
                        else:
                            siege[1] += 1
            elif spielende[0] == 0:
                siege[1] = 1
            else:
                siege[0] = 1
```

```
            # Nun wird die Liste umgekehrt, da im Falle eines Sieges von Index 1
            # nichts zu ´mitlauf´ beim Entfernen addiert werden soll, umgekehrt
            # bei einem Sieg von Index 0
            siege.reverse()
            del qualifiziert[mitlauf+siege.index(max(siege))]
    return([qualifiziert[0]])


def wiederholung(wiederholungen, func):
    """Wiederholt ´func´ ´wiederholungen´ mal."""
    # Die Datei ´f´ wird global gebraucht
    global f
    # Öffne die Datei der Spielstärken
    f = open(sys.argv[1], "r")
    # Die Anzahl der Spieler ist in der ersten Zeile der Datei definiert
    spieler = int(f.readline().strip("\n"))
    # Kopiere die verfügbaren Funktionen in ´funktionen´
    funktionen = globals().copy()
    funktionen.update(locals())
    # Die aufzurufende Funktion ´func´ befindet sich in ´funktionen´
    func = funktionen.get(func)
    # Initialisierung der Liste der Anzahl der Siege
    siege = []
    # Lies die Spielstärken durch Aufrufen von ´einlesen´ auf
    spielende = einlesen(spieler)
    # IO-Operationen sind beendet
    f.close()
    # Kopiere ´spielende´ für einen späteren Vergleich
    spielende_kopie = spielende[:]
    print(spielende)
    # Initialisierung der Siege der Paarung, der Index entspricht
    # dem Index des Spielers in ´spielende´
    for i in range(0, spieler):
        siege.append(0)
    # Rufe ´func´ ´wiederholungen´ mal auf
    for i in range(0, wiederholungen):
```

```
        # Speichere den Sieger dieser Iteration in ´sieger´
        sieger = func(spielende)
        # Erhöhe die Anzahl der Siege dieser Sieger in ´siege´
        for j in range(0, len(sieger)):
            siege[sieger[j]] += 1
    # Initialisierung der Liste der auszugebenden Sieger
    ausgabe = []
    # In ´ausgabe´ werden alle Indizes gespeichert, an denen die Siege am höchsten sind
    for i, j in enumerate(siege):
        if j == max(siege):
            ausgabe.append(spielende_kopie[i])
    print(siege)
    return ausgabe


if len(sys.argv) == 4:
    print(wiederholung(int(sys.argv[2]), sys.argv[3]))
else:
    print("Korrekte Nutzung: aufgabe3.py <Datei>.txt <Wiederholungen> <liga|ko|ko5>")
```