Eric Wolf, 19.11.2020

BWINF Aufgabe 3

Lösungsidee

Wie in den Materialien beschreiben stelle ich die Turnierpläne als Binärbäume dar, wobei die inneren Knoten Begegnungen zwischen Spielern und die Blätter Spieler darstellen.

Die Spieler-Knoten speichern nur die Nummer ihres jeweiligen Spielers, während die Begegnungs-Knoten die zwei beteiligten Parteien und eine Rundenanzahl speichern.

Dies sorgt dafür, das jede Form von KO-Turnier unabhängig der Rundenzahlen im gleichen Format (JSON?) dargestellt werden kann, den ein einfaches KO-Turnier ist das gleiche wie ein KO-Turnier mit jeweils einer Runde.

Die Ligen könnte ich mit Hilfe eines Iterators implementieren, welcher die einzelnen Begegnungen liefert und wessen Ergebnisse gespeichert und ausgewertet werden.

Umsetzung

Die Spieler werden durch die Klasse „Player“ mit ihrer Stärke als Attribut dargestellt.

Diese Klasse besitzt die Methode „match“, welche ein Spiel wie in den Materialien beschrieben gegen einen Gegner simuliert und den Gewinner zurück gibt.

Um den Spielern Nummern zuzuweisen wird die Menge aller Spieler eines Turniers in der Klasse „Players“ gespeichert, welche Zugriff auf die einzelnen Spieler über eine Spielernummer gewährt und im Attribut „strongest“ die Spielernummer des stärksten Spielers speichert.

Umgesetzt habe ich die Turnierpläne mit dem „Kompositum“ Entwurfsmuster, welches aus eine abstrakten Klasse „TournamentNode“ und den zwei davon abgeleiteten Klassen „PlayerNode“ und „MatchNode“ besteht.

Jeder Knoten implementiert eine Methode zum simulieren eines Turniers, welche die Menge aller Spieler entgegennimmt und die Nummer des Gewinners zurückliefert.

Bei den Spieler-Knoten wird dafür einfach die Nummer des repräsentierten Spielers zurückgegeben, während Begegnungs-Knoten zuerst die Gewinner ihrer beiden Parteien simulieren, die daraus resultierenden Spieler gegeneinander antreten lassen und am Ende den Spieler mit den meisten Siegen zurückgeben.

Im Falle eines Gleichstandes wird wie bei der Liga der Spieler mit der kleinsten Spielernummer zurückgegeben, für was ich mich mangels Anforderungen entschieden habe.

Das Lesen eines Turnierplans wird über eine Funktion realisiert, welche den geparsten Inhalt einer JSON Datei entgegennimmt rekursiv einen Baum aufbaut bis ein Spieler-Knoten erreicht ist.

Ligen habe ich wie in der Lösungsidee umgesetzt, jedoch sorgte die erste und simple Version des

Iterators für doppelte Begegnungen zwischen Spielern, weil bei einer Begegnung das Vertauschen der beiden Partien egal ist (A vs. B ist das gleiche wie B vs. A) und für jeden Spieler Begegnungen mit allen anderen Spielern generiert wurden.

Das habe ich gelöst, indem der Iterator nur noch Paare für jeden Spieler bildet, bei welchen der Gegner eine höhere Spielernummer besitzt.

Dies sorgt dafür, das Begegnungen immer nur in eine Richtung (kleine zu großer Nummer) gebildet werden und die Duplikate (große zu kleiner Nummer) entfallen.

Beispiel (Kreis = Spieler, Pfeil = Begegnung):

Beispiele

Das Programm trägt den Namen rng.py und lässt sich mit dem Python 3 Interpreter ausführen (getestet Version 3.7.3 und 3.9.0).

Es nimmt als Kommandozeilenparameter einen Pfad zur Spielerliste, wie oft jedes Turnier wiederholt werden soll und eine beliebige Anzahl an Pfaden zu Turnierplänen entgegen.

Die Spielerlisten befinden sich im Verzeichnis „spieler“ während sich die Beispielpläne aus dem Material im JSON Format im Verzeichnis „plaene“ befinden.

Eine Liga lässt sich nur sehr umständlich durch JSON darstellen, weswegen sie durch ihre interne Implementation jedes mal ausgeführt wird.

Testen der Turnierpläne 1 bis 3 (mit x5) mit Spielerliste 1 bis 2 und 9000 Wiederholungen:

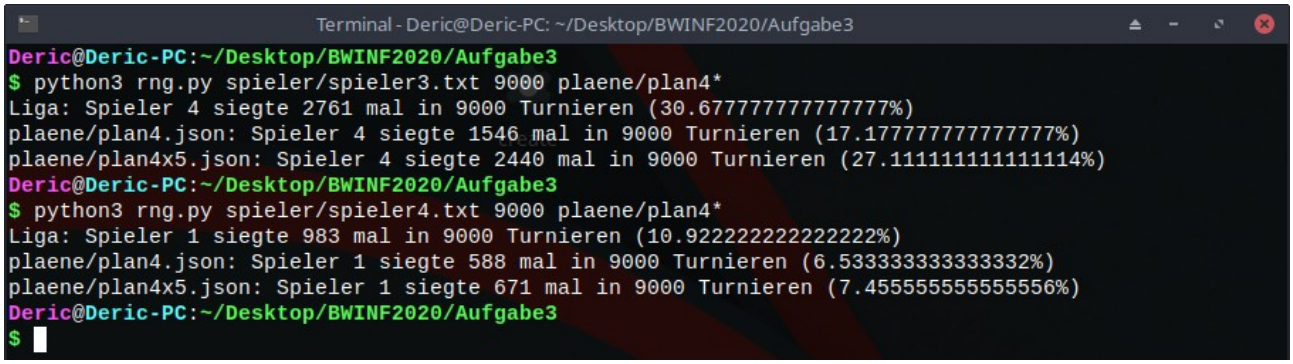
(Wildcard wird durch die Shell ersetzt, Notfalls alle Pfade angeben)

```
Terminal - Deric@Deric-PC: ~/Desktop/BWINF2020/Aufgabe3
Deric@Deric-PC:~/Desktop/BWINF2020/Aufgabe3
$ python3 rng.py spieler/spieler1.txt 9000 plaene/plan1* plaene/plan2* plaene/plan3*
Liga: Spieler 8 siegte 3122 mal in 9000 Turnieren (34.68888888888889%)
plaene/plan1.json: Spieler 8 siegte 4271 mal in 9000 Turnieren (47.455555555555556%)
plaene/plan1x5.json: Spieler 8 siegte 5433 mal in 9000 Turnieren (60.36666666666667%)
plaene/plan2.json: Spieler 8 siegte 3233 mal in 9000 Turnieren (35.922222222222224%)
plaene/plan2x5.json: Spieler 8 siegte 4981 mal in 9000 Turnieren (55.34444444444444%)
plaene/plan3.json: Spieler 8 siegte 3908 mal in 9000 Turnieren (43.422222222222224%)
plaene/plan3x5.json: Spieler 8 siegte 6244 mal in 9000 Turnieren (69.37777777777778%)
Deric@Deric-PC:~/Desktop/BWINF2020/Aufgabe3
$ python3 rng.py spieler/spieler2.txt 9000 plaene/plan1* plaene/plan2* plaene/plan3*
Liga: Spieler 8 siegte 1904 mal in 9000 Turnieren (21.155555555555555%)
plaene/plan1.json: Spieler 8 siegte 2758 mal in 9000 Turnieren (30.644444444444446%)
plaene/plan1x5.json: Spieler 8 siegte 3237 mal in 9000 Turnieren (35.96666666666667%)
plaene/plan2.json: Spieler 8 siegte 2498 mal in 9000 Turnieren (27.755555555555556%)
plaene/plan2x5.json: Spieler 8 siegte 3221 mal in 9000 Turnieren (35.788888888888884%)
plaene/plan3.json: Spieler 8 siegte 4070 mal in 9000 Turnieren (45.22222222222222%)
plaene/plan3x5.json: Spieler 8 siegte 5283 mal in 9000 Turnieren (58.699999999999996%)
Deric@Deric-PC:~/Desktop/BWINF2020/Aufgabe3
$
```

Eric Wolf, 19.11.2020

Testen des Turnierplans 4 und 5 mit Spielerliste 3 bis 4 und 9000 Wiederholungen:

(Plan 4 ist eine auf 16 Spieler erhöhte Variante von Plan 2, Plan 5 eine von Plan 3)



Aus den Ergebnissen geht hervor, das meistens die Variante KOx5 den stärksten Spieler am meisten gewinnen lässt, wobei die Variante von Plan 3 die höchste Wahrscheinlichkeit bietet.

Jedoch zeigt sich bei den Versuchen mit 16 Spielern, dass bei höheren Spielerzahlen die Liga für bessere Ergebnisse sorgt.