Aufgabe 4: Würfelglück

20. November 2021

Team-ID: 00821
Team: IaC 0.9524
Bearbeiter dieser Aufgabe: Lorenz Wildberg

Lösungsidee:

Um herauszufinden, welcher Würfel die höchsten Gewinnchancen bietet, muss man möglichst viele Mensch-ärgere-dich-nicht Spiele simulieren, wobei immer jeweils zwei virtuelle Spieler mit zwei unterschiedlichen Würfeln antreten. Zum Schluss wertet man die Ergebnisse aus und der Würfel, der am häufigsten gewonnen hat ist der beste.

Umsetzung:

Als erstes habe ich die benötigten Bibliotheken inportitert.

```
import sys, random
```

Dann werden die Kommandozeilenargumente eingelesen. Als erstes wird die Eingabedatei mit den Würfelinformationen eingelesen und in der Variable „input“ gespeichert. Falls etwas falsch eingegeben wurde, wird eine Fehlermeldung ausgegeben und abgebrochen.

```
input: list = None
try:
    input = open(sys.argv[1]).readlines()
except (IndexError, FileNotFoundError):
    print("Fehler: Argument 1 muss die Datei mit den Würfeln sein")
    exit(1)
```

Und in der Variable „wiederholungen“ wird die Anzahl der durchzuführenden simulationen gespeichert. Wenn ein fehler aufgetreten ist, wird wieder das programm nach einer Meldung beendet.

```
wiederholungen: int = None
try:
    wiederholungen = int(sys.argv[2])
except (IndexError, ValueError):
      print("Fehler: Argument 2 muss die Anzahl der Wiederholungen
sein")
    exit(1)
```

Danach wird eine Liste „wuerfel_liste“ mit allen Würfeln erstellt. Jeder Würfel in dieser Liste wird jeweils wieder durch eine Liste mit den Augenzahlen auf den Seiten des Würfels repräsentiert.
Die Variable „anzahl_wuerfel“ beinhaltet die Anzahl der zu vergleichenden Würfel.

```
wuerfel_liste: list = []
```

```
for i in range(int(input[0])):
    ein_wuerfel: list = []
    first: bool = True
    for j in input[i+1].split(" "):
        if first:
            first = False
            continue
        ein_wuerfel += [int(j)]
    wuerfel_liste += [ein_wuerfel]
anzahl_wuerfel: int = len(wuerfel_liste)
```

Dann werden mehrere Klassen erstellt, die das Spiel darstellen sollen.
Die Klasse „Figur“ repräsentiert eine von vier normalen Spielfiguren pro Spieler.

```
class Figur:
    def __init__(self):
        # 0 - Figur ist im Startfeld.
        # 1-40 - Figur ist positionauf den Speilbrett.
        # 41-44 - Figur ist in den Zielfeldern.
        self.feld: int = 0

    def vorruecken(self, felder: int):
        self.feld += felder

    def wird_geschlagen(self):
        if not self.feld > 40:
            self.feld = 0

    def auf_a_feld(self) -> bool:
        return self.auf_feld_nummer(1)

    def in_b_bereich(self) -> bool:
        return self.auf_feld_nummer(0)

    def im_ziel(self) -> bool:
        return self.feld > 40

    def auf_feld_nummer(self, feld_nummer: int) -> bool:
        return self.feld == feld_nummer
```

Die Klasse „Spieler“ repräsentiert einen von zwei Spielern pro Partie. Jeder Spieler besitzt vier Figuren. Diese sind in einer liste „figuren“ gespeichert. Außerdem hat jeder Spieler auch einen eigenen Würfel „wuerfel“, der aus der eingelesenen Wüfelbeschreibung, mithilfe einer Nummer des Würfels „wuerfel_nummer“ in der Liste „wuerfel_liste“ vom Anfang erstellt wird.

```
class Spieler:
    def __init__(self, wuerfel_nummer: int):
```

```
        self.wuerfel_nummer: int = wuerfel_nummer
        self.wuerfel: list = wuerfel_liste[wuerfel_nummer]

        self.figuren: list = [Figur() for i in range(4)]
        self.figuren[0].feld = 1 # Die erste figur startet auf dem A-
Feld

    def wuerfele(self) -> int:
        return random.choice(self.wuerfel)

    def figur_in_b_bereich(self) -> bool:
        for figur in self.figuren:
            if figur.in_b_bereich():
                return True
        return False

    def alle_im_ziel(self) -> bool:
        for figur in self.figuren:
            if not figur.im_ziel():
                return False
        return True

    def feld_a_ist_frei(self) -> bool:
        return self.feld_nummer_ist_frei(1)

    def feld_nummer_ist_frei(self, feld_nummer: int) -> bool:
        for figur in self.figuren:
            if figur.auf_feld_nummer(feld_nummer):
                return False
        return True

    def get_figur_auf_feld_a(self) -> Figur:
        return self.get_figur_auf_feld_nummer(1)

    def get_figur_auf_feld_nummer(self, nummer: int) -> Figur:
        for figur in self.figuren:
            if figur.auf_feld_nummer(nummer):
                return figur
        return None

    def darf_vorruecken(self, vorrueck_figur: Figur, felder: int) ->
bool:
        if (vorrueck_figur.feld + felder) > 44:
            return False
        for figur in self.figuren:
            if figur.feld == (vorrueck_figur.feld + felder):
                return False
```

```
        return True

    def hohle_aus_b_bereich(self):
        for figur in self.figuren:
            if figur.in_b_bereich():
                figur.auf_a_feld()
                return
        return

    def get_weiteste_ziehbare_figur(self, felder: int) -> Figur:
        weiteste_figur: Figur = None
        for figur in self.figuren:
                    if (weiteste_figur == None or figur.feld >
weiteste_figur.feld) and self.darf_vorruecken(figur, felder):
                weiteste_figur = figur
        return weiteste_figur
```

Die Klasse „Spielfeld“ regelt dagegen den Ablauf des Spiels. Sie beinhaltet zwei Spieler, die gegen einander antreten. „nicht_am_zug“ ist, wie der Name schon sagt, der Spieler, der gerade nicht am Zug ist, und „am_zug“ der, der gerade am Zug ist. „am_zug“ ist am Anfang auch der, der das Spiel beginnt.

Die Funktion „ziehe“ führt einen Zug des Spielers „am_zug“ aus. In einer Schleife wird als erstes immer gewürfelt. Wenn eine sechs gewürfelt wurde, wird später das ganze wiederholt, bis keine sechs mehr gewürfelt wird und danach die Schleife abgebrochen wird.
Als erstes wird geprüft, ob eine Figur auf dem „A-Feld steht, und diese wenn mölich bewegt. Wenn das nicht funktioniert, Wird versucht, eine Figur aus dem „B-Bereich“ auf das „A-Feld“ zu holen. Das geht aber nur, wenn das „A-Feld“ frei ist, und noch mindestens eine Figur im „B-Bereich“ steht. Wenn das aber auch nicht funktioniert, muss der Spieler ganz normal eine Seiner figuren ziehen. Wenn das auch nicht klappt, kann er gar nichts machen.
Das gesamte Spiel wird kjedoch von der Funktion „simuliere“ gesteuert. Als erstes wwerden zwei Würfel, mit denen gespielt werden soll, übergeben. Dann werden beide Spieler jeweils mit diesen Würfeln erstellt. Dann wird eine Schleife solange ausgeführt, bis das Spiel zuende ist.
Erst einmal zieht der Spieler, der gerade am Zug ist. Wenn das Spiel schon zuende ist, weil alle Figuren eines Spielers im Ziel stehen, wird die Schleife abgebrochen, und die Nummer des Würfels des Gewinners zurückgegeben. Wenn aber nicht, werden die Spieler in den variablen „am_zug“ und „nicht_am_zug“ ausgetauscht, sodass im nächsten Schleifendurchlauf der nächste am Zug ist.

```
class Spielfeld:
    def __init__(self):
        self.nicht_am_zug: Spieler = None
        self.am_zug: Spieler = None

    def ziehe(self):

        # solange der Würfel eine 6 würfelt
        while True:
            augenzahl: int = self.am_zug.wuerfele()
```

```
            # bewege die Figur auf dem A-Feld wenn möglich
                        if  not  self.am_zug.feld_a_ist_frei()  and
self.am_zug.figur_in_b_bereich():
                figur: Figur = self.am_zug.get_figur_auf_feld_a()
                if self.am_zug.darf_vorruecken(figur, augenzahl):
                    self.vorruecken_mit_schlagen(figur, augenzahl)

            # Hole eine Figur aus dem B-Bereich wenn möglich
             elif self.am_zug.figur_in_b_bereich() and augenzahl == 6
and self.am_zug.feld_a_ist_frei():
                self.am_zug.hohle_aus_b_bereich()

            # ziehe ganz normal
            else:
                                                    figur:   Figur   =
self.am_zug.get_weiteste_ziehbare_figur(augenzahl)
                if figur != None:
                    self.vorruecken_mit_schlagen(figur, augenzahl)

            if augenzahl != 6:
                break

    def vorruecken_mit_schlagen(self, figur: Figur, augenzahl: int):
        end_feld: int = figur.feld + augenzahl
        feld_fuer_gegner: int = None
        if end_feld > 20:
            feld_fuer_gegner = end_feld - 20
        else:
            feld_fuer_gegner = end_feld + 20
        if not self.nicht_am_zug.feld_nummer_ist_frei(end_feld):
            self.nicht_am_zug.get_figur_auf_feld_nummer(end_feld).wir
d_geschlagen()
        figur.vorruecken(augenzahl)

    # gibt die Nummer des Wüfels des Gewinners zurück
    def simuliere(self, wuerfel1: int, wuerfel2: int) -> int:
        self.am_zug = Spieler(wuerfel1)
        self.nicht_am_zug = Spieler(wuerfel2)

        while not self.spiel_zuende():
            self.ziehe()
            if self.spiel_zuende():
                return self.am_zug.wuerfel_nummer

            # wechsle den Spieler ab, der am Zug ist
            temp: Spieler = self.am_zug
```

```
            self.am_zug = self.nicht_am_zug
            self.nicht_am_zug = temp

    def spiel_zuende(self) -> bool:
        return self.am_zug.alle_im_ziel()
```

Um wieder zum Anfang zu kommen, nachdem die Eingabedatei in die Würfelliste umgewandelt wurde, werden nun für jede Kombination aus zwei verschiedenen Würfeln zwei Partien simuliert. Jeweils beginnt der andere, um gerechtigkeit zu bewahren. Das ganze wird dann auch noch so oft wie in „anzahl_wiederholungen“ gespeichert wiederholt. Je mehr Wiederholungen, desto besser die Werte, da das ganze ja immer noch vom Zufall abhängt. Die Gewinnerwürfel aller Spiele werden dann einfach in einer Liste namens „alle_gewinner_wuerfel“ gespeichert.

```
alle_gewinner_wuerfel: list = []
for i in range(wiederholungen):
    for wuerfel1 in range(anzahl_wuerfel):
        for wuerfel2 in range(anzahl_wuerfel):

            # jede Kombination soll nur einmal simuliert werden
            if not wuerfel2 <= wuerfel1:
                spielfeld: Spielfeld = Spielfeld()
                gewinner_wuerfel: int = spielfeld.simuliere(wuerfel1,
wuerfel2)
                alle_gewinner_wuerfel.append(gewinner_wuerfel)
                # nochmal, aber diesmal fängt der andere an
                spielfeld: Spielfeld = Spielfeld()
                gewinner_wuerfel: int = spielfeld.simuliere(wuerfel2,
wuerfel1)
                alle_gewinner_wuerfel.append(gewinner_wuerfel)
```

Danach werden die Gewinne gezählt. Dabei wird für jeden Würfel die Anzahl des Vorkommens in der Liste „alle_gewinner_wuerfel“ berechnet und dann geprüft, ob es davor schon einen Würfel mit höherem Vorkommen gab. Wenn nicht, wird er in „oefteste_gewinner“ abgespeichert. Nachdem alle Würfel durchlaufen wurden, hat man den oder die Gewinner, nämlich dann, wenn mehrere Würfel gleich of vorkamen.

```
# zähle die Gewinne
oefteste_gewinner: list = []
gewinner_hat_so_oft_gewonnen: int = 0
for i in range(anzahl_wuerfel):
    so_oft_gewonnen: int = alle_gewinner_wuerfel.count(i)
    if gewinner_hat_so_oft_gewonnen < so_oft_gewonnen:
        oefteste_gewinner = [i]
        gewinner_hat_so_oft_gewonnen = so_oft_gewonnen
    elif gewinner_hat_so_oft_gewonnen == so_oft_gewonnen:
        oefteste_gewinner.append(i)
```

Zum Schluss werden noch die Inhalte der Liste „oefteste_gewinner“ mit einer schönen Nachricht ausgeben.

```
for wuerfel in oefteste_gewinner:
      print("Mit Würfel Nummer {0} haben sie die besten Chancen zu gewinnen.".format(wuerfel))
```

Beispiele:

Mit der Beispieldatei "wuerfel0.txt" mit 100 durchläufen bekommt man beispielsweise diese Ausgabe:

```
$ ./main.py wuerfel0.txt 100
Mit Würfel Nummer 4 haben sie die besten Chancen zu gewinnen.
```