Python Aufgabe Pilzsammler:innen (Spieltheorie)

Torben Friedrich Görner

Dezember 2022

1 Das Szenario

Täglich gehen Plizsammler:innen als Gruppe in den Wald, um Pilze zu sammeln. Am Abend wollen alle eine Pilzpfanne kochen. Wenn die Pilzsammler:innen am Abend ihre gesammelten Sorten teilen, muss jeder nur noch 1€ für zusätzliche Pilze ausgeben. Wenn beide nicht teilen, müssen beide 3€ für weitere Pilze ausgeben. Wenn A teilt, B aber nicht, muss A 5€ ausgeben und B 0€. Die Kosten können der Matrix entnommen werden.

Die Plizsammler:innen gehen jeden Tag los um Pilze zu sammeln und ggf. zu teilen. Welche Strategie zur Entscheidungsfindung ob geteilt werden soll oder nicht ist am erfolgreichsten ? Um diese Frage zu untersuchen entwickeln wir eine Simulation.

Matrix

Kostentabelle	**A teilt**	**A teilt nicht**
B teilt	1/1	5/0
B teilt nicht	0/5	3/3

Table 1: Kosten für Plizsammler:innen

Bitte schreit eure Ergebnise auf. Nutzt hierfür Word, Google Docs oder ähnliches. Ihr dürft die einzelnen Aufgaben gerne mit eigenen Ideen erweitern.

2 Aufgabe 1 - Erstellen von Strategien

Gegeben seien die folgenden Klassen. *Picker*, die Basis Klasse für Pilzsammler:innen von welcher *Coop* erbt. *Coop* ist ein/e Pilzsammler:inn welche immer teilt (immer kooperiert).

```
class Picker:                # Basis Klasse (Oberklasse) für Pilzsammler:innen
  def __init__(self, name):
    self.name = name
    self.last = {}      # Letzte durchgeführte Handlungen für meine Gegenüber (dic mit name
    self.opp_last = {} # Letzte durchgeführte Handlungen meiner Gegenüber (dic mit name und
    self.costs = 0      # Kosten für Pilzsammler:inn
    self.days = 0       # Anzahl der Tage an denen gesammelt wurde

  def inc(self, n):
    """Erhöhe Kosten um n"""
    self.costs += n
    self.days += 1

  def init(self, opp_list):
    """Initalisierung des dictionaries opp_last mit
    namen als Key und jeweils 0 als Value. """
    for person in opp_list:
      if not(person.name is self.name):
        self.opp_last[person.name] = 0
        self.last[person.name] = 0


class Coop(Picker):          # Teilt die Pilze immer
  def share(self, opp):
    self.last[opp.name] = 0
    self.opp_last[opp.name] = opp.last[self.name]
    return self.last[opp.name]
```

Entwickle folgende weitere Strategien (Pilzsammler:innen Klassen) nach dem Vorbild von *Coop*.

- Pilzsammler:innen Strategie welche nie teilt (Betray).
- Pilzsammler:innen Strategie welche immer abwechselnd teilt oder nicht teilt (Altern).
- Pilzsammler:innen Strategie welche immer genau das tut, was ihr Gegenüber in der letzten Runde getan hat (TitForTat).
- Pilzsammler:innen Strategie welche immer das Gegenteil tut, was ihr Gegenüber in der letzten Runde getan hat (Opposite).

3 Aufgabe 2 - Anlegen von Pilzsammler:innen für Simulationen

Erstelle verschiedene Gruppen von Plizsammler:innen. Bilde dazu Gruppen mit jeweils 6 Personen. Verwende hierfür eine Liste in welcher Plizsammler:innen verwaltet werden. Unten ist ein Beispiel abgebildet.

```
ni1  = Coop('Nice1' )
ni2 = Coop('Nice2' )
persons = [ni1, ni2]
```

4 Aufgabe 3 - Erstellen einer Simulation

Teste verschiedene Konfigurationen von Gruppen und analysiere deine Ergebnisse. Nutze hierfür den unten abgebildeten Code. Versuche die Struktur nachzuvollziehen und die Simulation zu verstehen. (Den Code findest du auch in meinem GitHub Repo)

```
## Pilzsammler:innen sammeln verschiedene Pilzsorten. Sie wollen eine Pilzpfanne
## kochen. Dabei können sie am Ende eines Tages ihre Pilze teilen oder nicht.
## Teilen Sie ihre Pilze, so müssen sie nur 1€ für weitere Pilze ausgeben.
## Teilen Sie nicht, so müssen beide 3€ für weitere Pilze ausgeben.
## Teilt Person A, Person B aber nicht, so muss Person B nichts ausgeben und Person A 5€
from math import *
from random import randrange, seed, shuffle


## Strategien für Pilzsammler:innen

class Picker:                # Basis Klasse (Oberklasse) für Pilzsammler:innen
  def __init__(self, name):
    self.name = name
    self.last = {}     # Letzte durchgeführte Handlungen für meine Gegenüber (dic mit name
    self.opp_last = {} # Letzte durchgeführte Handlungen meiner Gegenüber (dic mit name und
    self.costs = 0     # Kosten für Pilzsammler:inn
    self.days = 0      # Anzahl der Tage an denen gesammelt wurde

  def inc(self, n):
    """Erhöhe Kosten um n"""
    self.costs += n
    self.days += 1

  def init(self, opp_list):
    """Initalisierung des dictionaries opp_last mit
    namen als Key und jeweils 0 als Value. """
```

```
    for person in opp_list:
      if not(person.name is self.name):
        self.opp_last[person.name] = 0
        self.last[person.name] = 0


class Coop(Picker):           # Teilt die Pilze immer
  def share(self, opp):
    self.last[opp.name] = 0
    self.opp_last[opp.name] = opp.last[self.name]
    return self.last[opp.name]

## Erweitere hier den Code mit deinen Strategien


def simulateDay(a,b,n):
  """Führt die Gegenüberstellung (Teilen oder nicht teilen) aus."""
  seed()
  ascore = bscore = 0
  for i in range(0,n):
    x = a.share(b), b.share(a)
    if x == (0,0):                    # Beide teilen (Kosten 1)
      a.inc(1); b.inc(1)
    elif x == (0,1):                  # B teilt nicht (Kosten A = 5, B = 0)
      a.inc(5); b.inc(0)
    elif x == (1,0):                  # A teilt nicht (Kosten B = 5, A = 0)
      a.inc(0); b.inc(5)
    elif x == (1,1):                  # Beide teilen nicht (Kosten = 3)
      a.inc(3); b.inc(3)


## Erstellen von Pilzsammler:innen

ni1 = Coop('Nice1')
ni2 = Coop('Nice2')
ni3 = Coop('Nice3')
ni4 = Coop('Nice4')
ni5 = Coop('Nice5')
ni6 = Coop('Nice6')
ni7 = Coop('Nice7')

persons = [ni1,ni2,ni3,ni4,ni5,ni6,ni7]     # Liste der Pilzsammler:innen

## Erstelle hier deine Pilzsammler:innen und
## Baue eine Liste 'persons' aus ihnen wie in Aufgabe 2.
## Ersetze dafür die oben angelegte Demo Liste aus nur Coop Pilzsammler:innen
```

```
def simulateDays(days):
  """Simulation von teilenden oder nicht teilenden Pilzsammler:innen über mehrere Tage"""
  seed()
  totalpen = 0

  for person in persons:
    person.init(persons)

  for day in range(0, days):                          # Ausführen der Gegenüberstellung (Tei
    shuffle(persons)
    simulateDay(persons[0], persons[1], 6)

  results = {}                                        # Dictionary der Personen (Namen) und

  for person in persons:
    results[person.name] = person.costs / person.days
    totalpen += person.costs
    #print(person.name, '\t', 'mit durchschnittlichen Kosten von', '\t' , person.costs/pers


  for result in results:
    print(result, '\t', results[result])

  print('\n\nTotal Penalties Suffered: ',totalpen)


simulateDays(100000) # Simulieren von 100.000 Tagen
```

5 Aufgabe 4 - Stretegien im Verleich

Welche Strategien sind wann besonders erfolgreich ?

- Welche Strategie ist in einer gemischten Gruppe am erfolgreichsten ?
- In welchen Gruppen ist die Strategie Betray (niemal teilen) am erfolgreichsten ?
- Gibt es Gruppenkonfigurationen, sodass TitForTat Sammler:innen schlechter abschneiden als andere Strategien ? Wenn ja, welche Strategie kann besser abschneiden und wie sieht die Gruppenkonfiguration aus ?

6 Aufgabe5

Entwickle eine Strategie für Pilzsammler:innen wobei immer zufällig geteilt wird oder nicht (Crazy). Unter folgender Quelle findest du ein Tutorial für den Umgang mit Zufall in Python. `https://www.w3schools.com/python/ref_random_randint.asp`

7 Aufgabe 6 - Visualisierung der Daten

Plotte deine Ergbnise der Simulationen als Diagramm. Unter folgender Quelle findest du eine Einführung in Histogramme, sowie ein Schritt-für-Schritt Beispiel.

`https://www.humaneer.org/blog/how-to-plot-a-histogram-using-python-matplotlib/`

Optional : Überlege dir welche Daten noch relevant sein könnten um sie zu plotten. Recherchiere hierzu bei Bedarf im Netz.