Analysieren und Visualisieren mit Python

dritte Übungsaufgabe

Aufgrund der Wahl Ihres Studienfaches ist Wahrscheinlichkeitsrechnung kein Problem für Sie. Über den Satz von Bayes können Sie nur herzhaft lachen. Umso mehr sind Sie genervt, dass es heutzutage immer noch Menschen gibt, die die Lösung des Monty-Hall-Problems (auch Ziegenproblem genannt) ungeniert in aller Öffentlichkeit hinterfragen. Solche Menschen begegnen Ihnen in den zahlreichen Kneipen der Regensburger Altstadt, beim täglichen Einkauf und selbst in den Cafeterien der Universität. Dem soll durch eine Simulation, die Sie von nun an jederzeit präsentieren können, ein für alle Mal der Garaus gemacht werden.
Rekapitulieren Sie zunächst, was es mit dem Monty-Hall-Problem auf sich hat. Wenn Ihnen das Lesen des Artikels auf Wikipedia zu eintönig erscheint, können Sie sich beispielsweise auch das Video auf https://www.youtube.com/watch?v=mhlc7peGlGg ansehen.
Sollten Sie nun auch daran zweifeln, ob es wirklich eine lohnenswerte Strategie ist, die Türe zu wechseln, nachdem der Moderator eine Ziege befreit hat, müssen Sie sich nicht grämen. Vielleicht überzeugen Sie ja die Ergebnisse der Simulation, die sich ganz ohne Kenntnisse in Stochastik programmieren lässt.
Sie brauchen für die Lösung der folgenden Aufgaben keine Ausnahmebehandlungen zu implementieren. Gehen Sie davon aus, dass alle Funktionen richtig verwendet werden und der Spieler in Aufgabenteil b) nur korrekte Eingaben macht. Als kleine Hilfestellung können Sie das mitgelieferte Script nutzen, um die Funktionalität der in Aufgabenteil a) zu schreibenden Klassen zu überprüfen. Dass alle Tests erfolgreich sind, gibt Ihnen aber keine Garantie, dass alles korrekt gelöst wurde. Sie dürfen die zu implementierenden Klassen nach Belieben um weitere Methoden oder Attribute erweitern. Sorgen Sie mittels der Abfrage *if __name__ == "__main__":* (siehe Vorlesungsfolien 1.6) dafür, dass Code außerhalb von Klassen und Funktionen, den Sie zur Funktionsüberprüfung gerne erstellen dürfen, nicht ausgeführt wird, wenn Ihr Script importiert wird.

3a) (6 Punkte)
Beginnen Sie damit, die **Klasse *Door*** (bitte beachten Sie die exakte Schreibweiße) zu implementieren (siehe Vorlesungsfolien 2.1). Diese soll über die drei **Instanzattribute *car*, *chosen* und *id*** verfügen. Das Attribut *car* soll speichern, ob hinter der jeweiligen Türen-Instanz ein Auto versteckt ist (*True*) oder nur eine Ziege (*False*). **Bei der Instanziierung** eines Objektes soll ***car*** auf den Wert ***False*** gesetzt werden. Auch das Attribut ***chosen*** soll zunächst mit dem Wert ***False*** belegt werden. Falls der Spieler die jeweilige Türe bei seiner ersten Entscheidung auswählt, wird der Wert dann umgestellt. Das **Attribut *id*** dient zur Nummerierung der Türen. Der darin zu hinterlegende **Wert soll der *__init__*-Methode bei Erzeugung eines Objektes übergeben werden**.
Implementieren Sie nun die **Klasse *MontyHall*, die mehrere *Door*-Instanzen verwaltet**. Sie sollen sich in dieser Implementierung nicht auf 3 Türen beschränken. So soll es zum Beispiel auch möglich sein, 20 (bzw. n) Türen zu erzeugen, wobei nur hinter einer der Türen ein Auto versteckt ist. Nach der Entscheidung des Spielers für eine der Türen wird der Moderator von den verbleibenden 19 (n-1) Türen dann 18 (n-2) öffnen, hinter denen jeweils eine Ziege versteckt ist. Der Spieler kann dann entscheiden, ob er zur (einzig) verbleibenden Türe wechselt oder bei seiner anfänglichen Wahl bleibt. In Abbildung 1 sehen Sie einen beispielhaften Spielablauf mit vier Türen.
Bei der Initialisierung eines Objektes soll der ***__init__*-Methode** deshalb die **Anzahl der zu erzeugenden Türen als Parameter** übergeben werden. Überlegen Sie sich dabei selbst, wie Sie die **erzeugten Door Instanzen verwalten**. Die **Türen** sollen mittels ihres **Attributs *id*** angefangen bei *1* **durchnummeriert** werden. Die Id soll hierfür **als Zeichenkette** gespeichert werden. Im *id*-Attribut der ersten Türe soll also nicht der Integer *1* sondern der String *"1"* abgelegt werden. Wählen Sie dann

zufällig eine dieser *Door*-Instanzen aus und setzen Sie deren Attribut ***car* auf *True***. Hinter dieser Türe ist somit das Auto versteckt. Zur zufälligen Auswahl einer Türe können Sie die Funktion *random.choice(<Liste>)* des *random*-Moduls verwenden (*import random* nicht vergessen, siehe Vorlesungsfolien 1.4).
Realisieren Sie als nächstes zwei Methoden, die die Auswahl einer Türe durch den Spieler ermöglichen. Fangen Sie mit der **Methode *choose*** an, die die **Id einer Türe als Parameter** übergeben bekommt. Die Türe, deren *id*-Attribut dem übergebenen Parameter entspricht, soll dann als ausgewählt markiert werden, indem **ihr Attribut *chosen* auf *True*** gesetzt wird.
Implementieren Sie dann die **Methode *choose_random.*** Diese soll das Attribut ***chosen* einer zufällig gewählten Türe** auf ***True*** setzen.
Nachdem der Spieler sich für eine Tür entschieden hat, brauchen Sie eine **Methode**, die den Moderator **alle bis auf zwei Türen öffnen** lässt. Im Fall von insgesamt drei Türen wird vom Moderator also beispielsweise eine Tür geöffnet. Nennen Sie diese Methode ***open_empty_doors***. Wieder können Sie die Funktion *random.choice* verwenden, um Türen zufällig auszuwählen. Stellen Sie dabei sicher, dass **weder** die vom Nutzer **ausgewählte** Türe (***chosen = True***) **noch** die Türe mit dem **Auto** dahinter (***car = True***) geöffnet wird. **Geben Sie** dann ein ***set* zurück** (siehe Vorlesungsfolien 1.1), das die ***id*-Attribute** (Typ *string*) **aller Türen** enthält, **die geöffnet wurden**.
Der Nutzer hat nun die Möglichkeit, entweder bei seiner ersten Wahl zu bleiben und somit die Tür mit *chosen = True* zu öffnen oder sich umzuentscheiden und die andere verbleibende Tür zu öffnen, welche nicht bereits vom Moderator geöffnet wurde. Implementieren Sie hierfür die **Methode *open_door***. Diese soll einen ***swap*-Parameter** entgegennehmen, der festlegt, ob der Nutzer bei seiner anfänglichen Wahl bleibt (dann ist der Parameter *False*) oder ob er sich umentscheidet (dann ist der Parameter *True*). Die Methode soll dann ***True* zurückgeben**, **falls** sich hinter der zu öffnenden Türe das **Auto** versteckt (***car = True***) oder ***False***, **falls** der Spieler nur eine **Ziege** gewinnt.

3b) (2 Punkte)
Sie sollen die Klasse *MontyHall* nun dazu nutzen, ein Spiel zu schreiben. Implementieren Sie hierfür die Funktion *play* (außerhalb der Klasse). Diese nimmt einen Parameter entgegen, der die Anzahl der Türen festlegt. Der Default-Value dieses Parameters soll 3 sein (siehe Vorlesungsfolien 1.3). Nutzen Sie die Funktion *raw_input*, um Eingaben des Nutzers zu lesen. Bei Aufruf von *play(5)* soll nun beispielsweise folgender Ablauf realisiert werden:

Auf dem Bildschirm wird angezeigt
Which door to choose (1-5):
Der Nutzer gibt nun eine Id, beispielsweise 4 ein.
Nun wird angezeigt, welche Türen vom Moderator geöffnet werden
Host opens door 1
Host opens door 2
Host opens door 5
Der Nutzer kann sich nun entscheiden, ob er bei seiner anfänglichen Wahl bleibt oder ob er wechseln möchte. Er kann dafür den Buchstaben s oder k eingeben.
Swap (s) or keep (k):
Nun wird angezeigt, ob sich hinter der gewählten Tür ein Auto (Car!) oder eine Ziege (Goat!) befindet.
Car!

3c) (2 Punkte)
Um die Gewinnwahrscheinlichkeit für eine bestimmte Strategie abzuschätzen, soll die Funktion *simulate* (wieder außerhalb der Klasse) implementiert werden. Die Funktion soll die Klasse *MontyHall* nutzen, um mehrere Spieldurchläufe zu simulieren und abschließend zu berechnen, bei welchem Teil der Spiele ein Auto gewonnen wurde. Die Funktion soll drei Parameter entgegennehmen: Der erste Parameter gibt die Anzahl der existierenden Türen pro Spiel an

(Default-Value = 3). Der zweite Parameter gibt an, wie viele Spiele pro Aufruf der Funktion simuliert werden sollen (Default-Value = 10000). Der dritte Parameter gibt an, ob der simulierte Spieler die Strategie verfolgt, seine erste Entscheidung zu verwerfen und die Tür zu wechseln (*True*) oder ob er die Strategie verfolgt, bei seiner ursprünglichen Entscheidung zu bleiben (*False*). Default-Value soll hier *True* sein. Der Rückgabewert der Funktion soll die ermittelte Gewinnwahrscheinlichkeit sein. Zur Berechnung soll also die Anzahl der Spiele, bei denen ein Auto gewonnen wurde, durch die Gesamtzahl der gespielten Spiele geteilt werden. Wenn Sie die Funktion nun mit *print simulate()* aufrufen und somit das Ergebnis anzeigen, sollte bei korrekter Funktionsweise ca. 0.66... angezeigt werden.

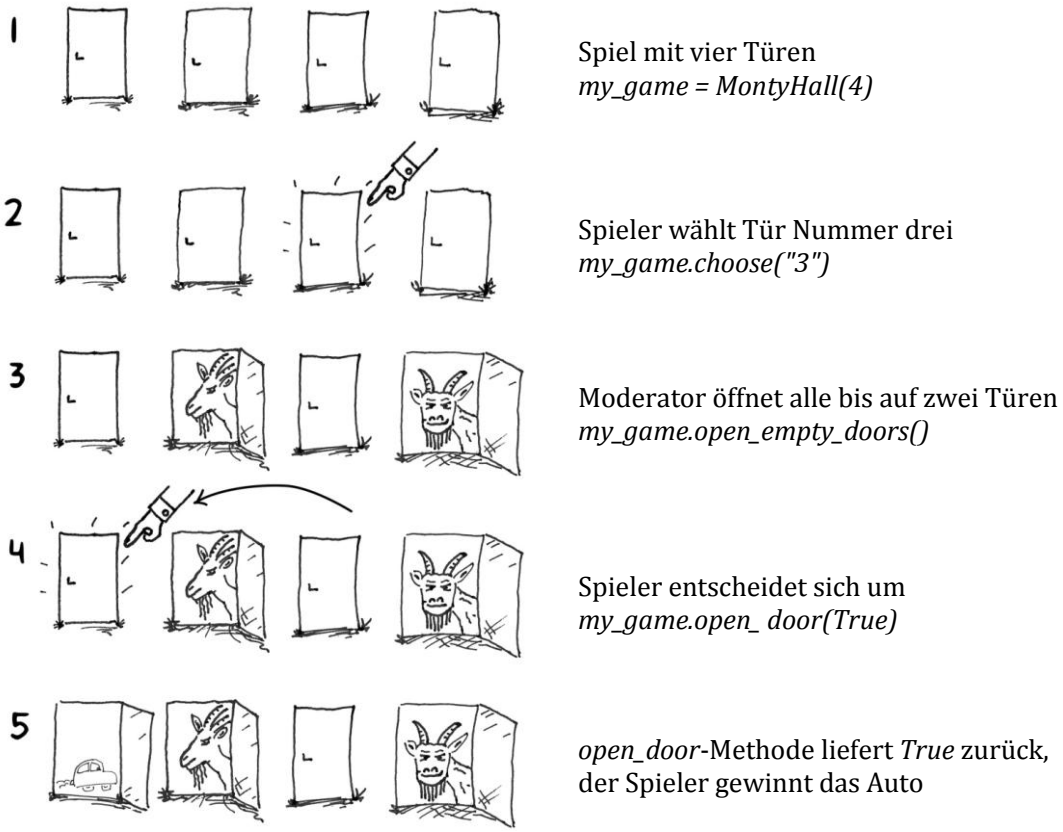

Abbildung 1: Beispielhafter Spielablauf mit 4 Türen

Spiel mit vier Türen
my_game = MontyHall(4)

Spieler wählt Tür Nummer drei
my_game.choose("3")

Moderator öffnet alle bis auf zwei Türen
my_game.open_empty_doors()

Spieler entscheidet sich um
my_game.open_ door(True)

open_door-Methode liefert *True* zurück, der Spieler gewinnt das Auto

Implementieren Sie zur Lösung dieser Aufgabe die Datei *ziegen.py* und reichen Sie sie über GRIPS bis zum 19.12.2018 um 23:55 Uhr ein.

Bitte geben Sie ausschließlich die (unverpackte) Datei ziegen.py ab.

Ihre Abgaben werden automatisiert mit den Abgaben der anderen Teilnehmer und gegebenenfalls mit den Abgaben der letzten Semester verglichen. Auch bei abgeänderten Variablenamen werden gleiche Lösungsansätze von der verwendeten Software erkannt. Im Falle eines Plagiats bekommen alle Gruppenmitglieder die Aufgabe als Plagiat bewertet, unabhängig davon, ob die Lösung weitergegeben oder übernommen wurde. Arbeiten Sie deshalb (nur) gemeinsam mit Ihren Übungspartnern an den Aufgaben. Falls Sie Ihre Übungspartner nicht erreichen, schreiben Sie mir eine E-Mail, sodass wir die Gruppeneinteilung anpassen können. Sollten Sie Code-Blöcke aus dem Internet übernehmen, kennzeichnen Sie die Quellen nachvollziehbar in den Kommentaren Ihrer Abgabe. Falls Sie bei der Bearbeitung der Aufgabe auf Probleme stoßen, können Sie die Übungstermine nutzen und dort Ihre Fragen stellen.

Das Auto einfach mal stehen zu lassen und auf der Ziege zur Uni zu reiten, hält die Ziege fit, schont die Nerven und verbessert Ihren CO_2-Fußabdruck um 7,3%

Viel Erfolg!