Einführung in die Programmierung

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Prof. Dr. Detlef Krömker
Prof. Dr. Rüdiger Brause
Dipl. Inf. Carsten Heep

Übungsblatt Nr. 4[1]

Ausgabe: 08.11.2013
Abgabe: 25.11.2013

Hinweis: *Dieses Aufgabenblatt ist, wie alle weiteren EPR Aufgabenblätter, mit einer Bearbeitungszeit von zwei Wochen dafür ausgelegt in einem Zweierteam gelöst zu werden.*
Neben der reinen Implementierung wird bei der Bearbeitung Wert auf die folgenden Punkte gelegt: *Dokumentation*[2], *Strukturierung* und Einhalten des *Style-Guides*[3].
Die genaue Aufschlüsselung der Bepunktung ergibt sich wie folgt:

- 50% entfallen auf die reine Funktionalität der Implementierung,
- 15% fließen in die Struktur ein. (Ist der Code logisch unterteilt?),
- 15% für die Einhaltung des Style-Guides und
- 20% entfallen schließlich auf die angegebenen Testfälle und die Dokumentation.

Achtung: Achten Sie darauf die Variable `__author__` in **allen** Quellcode Dateien **korrekt** zu setzen. Abgaben die nicht dieser Vorgabe entsprechen werden **nicht** Bewertet!

Quartett

Σ __ / **20**

Ziel dieses Übungsblattes ist es, ein Quartett Kartenspiel auf Konsolenebene zu programmieren. Das Spiel besteht aus 32 Karten á 8 Quartetten; mit je 4 Karten. Es treten zwei, oder mehr Spieler gegeneinander an. Bei zwei Spielern erhält jeder Spieler 10 Karten. Die restlichen 12 Karten bleiben als Stapel verdeckt liegen. Wenn mehr als zwei Spieler gegeneinander antreten werden die Karten so gleichmäßig (wie möglich) auf alle Mitspieler verteilt und der Stapel entfällt. Jeder Spieler überprüft seine Karten und legt alle Quartette ab, die sich auf der Hand befinden. Ein beliebiger Mitspieler beginnt das Spiel. Dieser fragt einen anderen Spieler gezielt nach einer Karte, die ihm zum Bilden eines Quartetts fehlt. Ist der gefragte Mitspieler im Besitz dieser Karte so gibt er diese Karte dem Fragenden. Dieser Vorgang setzt sich solange fort, bis der Mitspieler nicht im Besitz der gesuchten Karte ist. In diesem Fall zieht der fragende eine Karte vom Stapel (falls vorhanden) und der nächste Spieler ist an reihum an der Reihe zu fragen. Wann immer einer der beiden Spieler ein Quartett gebildet hat wird dieses abgelegt. Hat einer der Spieler keine Karten mehr auf der Hand ist das Spiel beendet. Gewonnen hat, wer die meisten Quartette bilden konnte.

[1] Es dürfen keine Lösungen aus dem Skript, dem Internet oder anderen Quellen abgeschrieben werden. Diese Quellen dürfen nur mit Quellenangaben verwendet werden und es muss ein hinreichend großer Eigenanteil in den Lösungen deutlich zu erkennen sein.
[2] `http://www.python.org/dev/peps/pep-0257`
[3] `http://www.python.org/dev/peps/pep-0008`

Aufgabe 4.1: **User Interface** Punkte: ____ / 8

Entwickeln Sie eine einfache Benutzungsschnittstelle auf Konsolenebene, die sich durch Tastatureingaben bedienen lässt. Das Interface soll dem Benutzer folgende Möglichkeiten bieten:

1. Die Anzahl der Mitspieler (mindestens 2, maximal 8) festlegen und deren Namen erfragen.
2. Zwischen *Realen-* und *Computergegnern* unterscheiden.
3. Im Falle eines realen Spielers diesen dazu auffordert, einen Mitspieler nach einer Karte zu fragen.
4. Dem Benutzer die Möglichkeit bieten **jederzeit** eine neu Runde zu starten und die bestehende abzubrechen.
5. Dem Benutzer die Möglichkeit bieten, das Spiel **jederzeit** zu *beenden.*

Ihr *User Interface* soll dabei robust angelegt sein, so dass falsche Eingaben des Benutzers **nicht** zu einem Absturz führen.
Hinweise:

- Entwickeln Sie eine Funktion für typsichere Benutzereingaben.
- Stellen Sie sicher, dass zu Beginn des Spiels jeder Spieler mindestens 4 Karten auf seiner Hand hält.

Aufgabe 4.2: **Implementierung** Punkte: ____ / 9

Implementieren Sie das Spiel unter Einhaltung der oben genannten Spielregeln. Essentielle Schritte sind:

- Anlegen eines Kartenstapels mit 8 Quartetten.
- Mischen und verteilen der Karten auf mindestens 2 Mitspieler.
- Ablegen von Quartetten.
- Das Fragen nach einer Karte eines der Mitspielern und ggf. das Übergeben dieser Karte an den Fragenden.
- Verdecktes Ziehen einer Karte vom Stapel, falls nur 2 Spieler antreten.
- Überprüfen, ob das Spiel beendet ist.
- Ermitteln und ausgeben des Siegers.

Aufgabe 4.3: **Computergegner** Punkte: ____ / 3

Erweitern Sie Ihr Programm um die Funktionalität eines Computergegneres. Die Festlegung, bei welchem Mitspieler es sich um einen *„realen“* oder *„virtuellen“* Gegner handelt, erfolgt wie in Aufgabenteil 1 beschrieben, beim Starten eines neuen Spieles.

Hinweise:

- Der Befehl `randint` aus dem Modul `random` generiert eine Zufallszahl.
- Mit dem Befehl `os.system("cls")` aus dem Modul `os` lässt sich der Inhalt des Bildschirm - in Windows Systemen - löschen, bzw. mit `os.system("clear")` in Unix/Mac OS.

```
1 import os
2 if os.name == "posix":
3     os.system('clear')
4 elif os.name in ("nt", "dos", "ce"):
5     os.system('cls')
6 else:
7     print(80 * "\n")
```

- Mit dem Befehl `getch` aus dem Modul `getch` bzw. `msvcrt` lassen sich einzelne Buchstaben einlesen, die nicht mit Enter bestätigt werden müssen.

```
1 try:
2     import os
3     if os.name in ("nt", "dos", "ce"):
4         from msvcrt import getch as getch
5     else:
6         from getch import getch as getch
7 except:
8     getch = input
```

- Die Symbole ♡, ♢, ♣ und ♠ lassen sich z.B. durch Unicode-Zeichen darstellen.

```
1 import sys
2 if sys.stdin.encoding.lower() == "cp850": ## Windows console
3     HEARTS, DIAMONDS, CLUBS, SPADES = '\x03', '\x04', '\x05', '\x06'
4 elif sys.stdin.encoding.lower() in ("utf-8", "cp1252"): ## Unix || IDLE
5     HEARTS, DIAMONDS, CLUBS, SPADES = '\u2665', '\u2666', '\u2663', '\u2660'
6 else:
7     HEARTS, DIAMONDS, CLUBS, SPADES = 'H', 'D', 'C', 'S'
```