Hochschule Darmstadt
Prof. Dr. Benjamin Meyer

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

22. November 2020

Programmieren, Algorithmen und Datenstrukturen I, WS 2020/21
Aufgabenblatt 2

Wir schreiben unser eigenes Roulette-Spiel!

2.1 Input und Output

Speichern Sie sich zuerst das aktuelle Vermögen des Spielers ab. Bei Programmstart beträgt dies 10.000€. Solange noch Geld übrig ist, soll in einer Schleife der nächste Spielzug abgefragt werden. Entscheiden Sie selber, welche Art von Schleife Sie hierfür nutzen wollen.
Für einen Spielzug muss der Spieler drei Werte eingeben:

- seinen Einsatz als positive ganze Zahl
- die Zahl, auf die er wettet
- fürs Erste: die Zahl, die im Roulette-Rad gefallen ist (Roulette läuft zwischen 0 und 36).

Hat der Spieler die richtige Zahl getroffen, gewinnt er das 35-fache seines Einsatzes (und bekommt auch seinen ursprünglichen Einsatz zurück)! Wurde auf die falsche Zahl gesetzt, ist das Geld weg.
Geben Sie aus, ob gewonnen oder verloren wurde, und wie viel Geld noch übrig ist.

2.2 Zufallszahlen

Die 'richtige' Roulette-Zahl selber einzugeben ist natürlich keine schöne Lösung.
Mithilfe der `rand()`-Funktion lässt sich eine Zufallszahl generieren, die ab jetzt das Roulette-Rad simulieren soll. `rand()` generiert hierfür Werte zwischen 0 und der Konstanten `RAND_MAX`.
Überlegen Sie sich, wie Sie hiermit eine Integer-Variable in dem Wertebereich [0...36] erzeugen können. Sie können auch die Funktion `round(double x)` zu Hilfe nehmen, welche eine Gleitkommazahl mathematisch korrekt rundet.
Ersetzen Sie mit diesem Ausdruck die manuelle Eingabe der Roulette-Zahl aus Aufgabe 1.

Leider beginnt der Generator immer mit der gleichen Zahl. Um das zu verhindern, können Sie den Generator mit einem Seed initialisieren. Hierfür eignet sich zum Beispiel die aktuelle Systemzeit. Es reicht, diesen Seed zum Programmstart einmal zu setzen via: `srand (time(NULL));` (hierfür müssen Sie zusätzlich `time.h` inkludieren).

2.3 Spielmodi

Nun gilt es, das Roulette-Spiel um mehrere Wett-Optionen zu erweitern. Lesen Sie daher zusätzlich als Tastatur-Input bei jedem Einsatz einen Spielmodus als char ein:

- 'f' für eine Farbe (Rot oder Schwarz)
- 's' für eine Spalte (1, 2, oder 3)
- 'z' für eine einzelne Zahl (0-36, genau wie in Aufgabe 1)

Nutzen Sie Switch-Case, um auf die drei Spielmodi einzugehen und sie jeweils auszuwerten.
Entsprechend den Roulette-Regeln gilt: alle ungeraden Zahlen sind rot, alle geraden Zahlen sind schwarz. Die Null ist weder rot noch schwarz (sondern grün), der Farbentipp verliert hier also immer. Wurde die Farbe richtig getippt, beträgt der Gewinn die Höhe des Einsatzes.
Das Spielfeld teilt sich zudem in 3 Spalten auf, Spalte eins enthält die Zahlen 1,4,7,10..., Spalte zwei dementsprechend die Zahlen 2,5,8,11,... und Spalte 3 die Zahlen 3,6,9,12,... Die Null lässt sich keiner Spalte zuordnen, auch hier verliert der Spieler immer. Gewinnen lässt sich der doppelte Einsatz.
Achten Sie bei Ihrem Code, wie immer, auf eine saubere Struktur, gute Kommentare sowie den Umgang mit Falscheingaben.

2.4 Bonusaufgabe: Gleichverteilte Zufallszahlen?

Können Sie testweise überprüfen, ob Ihre generierte Zufallszahl auch wirklich gleichverteilte Zahlen generiert? Schreiben Sie ein kleines Testprogramm, schränken Sie dafür den Wertebereich der Zufallszahl auf [0...9] ein, ähnlich wie vorher in Aufgabe 2.2. Generieren Sie Zählvariablen für jede mögliche Zahl im Wertebereich (wenn Sie es bereits kennen, dürfen Sie auch ein array hierfür verwenden).
Lassen Sie nun den Zufallsgenerator immer wieder laufen (beispielsweise eine Million Mal) und erhöhen die jeweilige Zählvariable um 1, wenn die entsprechende Zahl generiert wurde. Abschließend können Sie alle Zählvariablen durch die Anzahl der Tests (also zB eine Million) teilen und erhalten eine prozentuale Verteilung. Je mehr Tests Sie berechnen lassen, desto genauer sollte der Test ausfallen. Waren wirklich alle Zahlen gleichverteilt? Falls nein, warum nicht? Was können Sie dagegen tun?