Programmieren 1 - Praktikum 3

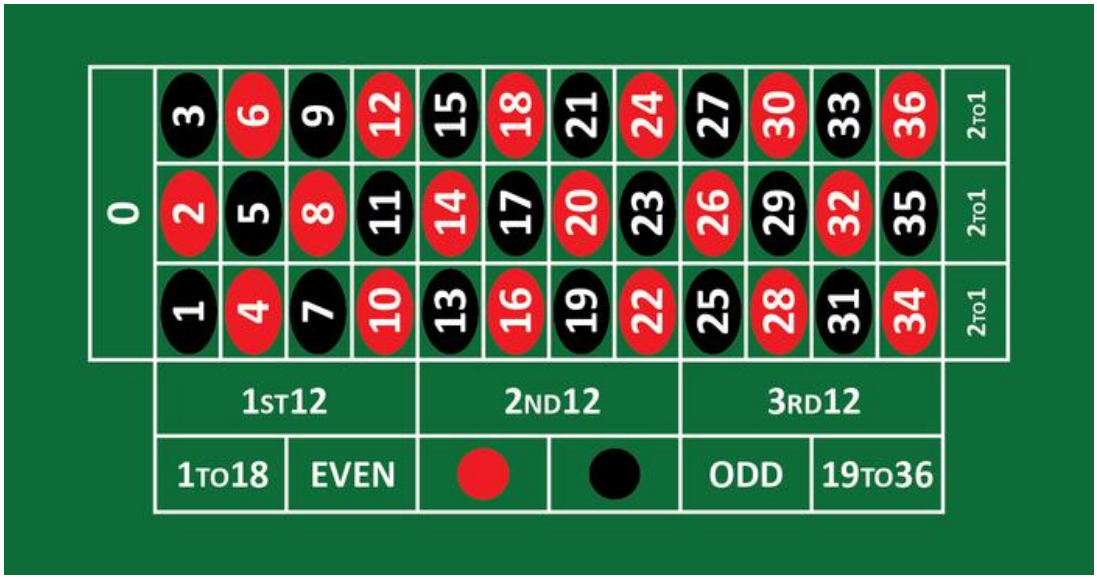
Benjamin Meyer, Michael Roth

Lernziele

- Schreiben eines ersten komplexeren Programms
- Arbeiten mit Strings
- Funktionierende Menüführung

1 Roulette

Wir schreiben unser eigenes Roulette-Spiel!



1.1 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.

Die 'richtige' Roulette-Zahl selber einzugeben ist natürlich keine schöne Lösung. Mithilfe der aus dem letzten Aufgabenblatt bereits bekannten `random.cpp` und `random.h` lässt sich eine Zufallszahl generieren, die ab jetzt das Roulette-Rad simulieren soll.

1.2 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 geraden Zahlen sind rot, alle ungeraden 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 auf eine **saubere Struktur**, gute Kommentare sowie **korrekten** Umgang mit **Falscheingaben** (wobei wir die Eingabe falscher Datentypen an dieser Stelle unberücksichtigt lassen).

2 Caesar-Chiffre

Die Caesar-Chiffre ist eine der ältesten dokumentierten Verschlüsselungsverfahren der Welt. Der Name leitet sich dabei vom römischen Feldherrn Gaius Julius Caesar ab, der diese Art der geheimen Kommunikation für seine militärische Korrespondenz verwendete. Die Grundidee dabei ist, dass jeder Buchstabe einer zu verschlüsselnden Nachricht zyklisch im Alphabet um einen bestimmten Wert (den sog. Schlüssel) verschoben wird. Bei einem Schlüssel von 3 wird beispielsweise aus einem a ein d, aus einem m ein p, und aus einem x wieder ein a. Klein- und Großschreibung wird erstmal ignoriert, verwenden Sie daher nur Kleinbuchstaben. Diese Verschlüsselung wollen wir nachprogrammieren.

Abbildung 1: Chiffrierscheibe für Caesar-Verschlüsselung

Nutzen Sie einen `std::string`, um ein zu verschlüsselndes Wort einzulesen. Gehen Sie nun jeden Buchstaben durch und verschieben ihn im Alphabet (wandeln Sie ihn hierfür zeitweise in einen Integer um). Die Kleinbuchstaben liegen hierbei im Bereich 97 bis 122. Zur Erinnerung: Sie können mit Strings ähnlich arbeiten wie mit Arrays oder Vektoren. Der zu verwendende Schlüssel kann hierbei im Grunde jede ganze Zahl annehmen, auch negative Schlüssel sind erlaubt.

Schreiben Sie abschließend ein kleines Programm drum herum, welches den Benutzer auffordert, den Schlüssel einzugeben, dann das Wort, welches verschlüsselt werden soll.

Beachten Sie bei der Eingabe, dass mit `cin` **keine** Leerzeichen eingelesen werden können! Das eingegeben Wort kann daher wirklich nur **ein** Wort sein!