**Zweck:** Plastische Illustration von fehlerbehaftetem Versenden am Beispiel von ASCII-Texten. Die Lernenden sehen anhand von Textbeispielen, wie stark fehlerkorrigierender Code das Empfangen von Text verbessert.

**Beschreibung:** Simuliert (Monte-Carlo) wird das Versenden eines ASCII-Textes (ein Textvorschlag ist beigefügt). Der Text wird Bit für Bit gesendet, wobei beim Senden sich der Bit mit der Wahrscheinlichkeit *fR* fehlerhafterweise dreht. Wenn z.B. fR = 10%, dann wird ein ASCII-Buchstabe mit W’keit  korrekt empfangen.

Alternativ kann man das Programm ändern, so dass es ein () bzw. zwei Fehler () selbstkorrigiert. Entsprechend werden mehr Bits gesendet. Die Monte-Carlo-Komponente bleibt in allen Fällen dieselbe: Jeder Bit dreht sich mit W’keit *fR*. Sei  die W’keit, dass das Codewort (welches ein Bit repräsentiert) trotz Selbstkorrektur fehlerhaft empfangen wird (mögliche Werte von  erhält man aus der Unterrichteinheit). Dann wird ein ASCII-Buchstabe mit W’keit  korrekt empfangen.

**Ausgabe der Simulation:**

* Der fehlerbehaftete empfangene Text.
* Der Originaltext als Vergleich.
* Die Anzahl Bits, welche die Codierung des Textes voraussetzt und die Anzahl Bits, welche tatsächlich gesendet wurden (dies erlaubt ein Kosten-Nutzen Vergleich für die Selbstkorrektur).

**Anwendung im Unterricht:**

Ich sehe es als Werkzeug für die LP, um den Lernenden den Effekt von Fehlern und von Fehlerkorrektur plastisch zu illustrieren. Als erster Eindruck halte ich es für fassbarer als die Bestimmung der zugehörigen W’keiten.

Da meine Unterrichtseinheit die entsprechende Theorie voraussetzt, kann man es über den ganzen Verlauf der Einheit immer wieder vorführen (ich denke beim ersten Mal 5’ erklären und 5’ vorführen, anschliessend jeweils 2’-3’ zeigen). Die Simulation eignet sich gut, um den Zweck der Bitmanipulationen zu illustrieren. Darum finde ich mehrmaliges Vorführen zweckmässig.

Im späteren Verlauf der Unterrichtseinheit kann man das Programm den Lernenden auch abgeben. Um ein Gespür für stochastische Effekte zu erhalten, ist es sinnvoll, den Run-Knopf ein paarmal selbst zu drücken (höchstens 10’ lang).

Die Arbeitsweise des Simulationsprogramms geht über OFI hinaus (es braucht zwei nicht gängige Python-Bibliotheksfunktionen). Entsprechend werde ich dazu nichts ausarbeiten.