Zufallszahlen in C++11

Ab C++11 stehen neuere und wesentlich bessere Algorithmen für die Erzeugung von Zufallszahlen zur Verfügung. Diese sind im Headerfile `<random>` definiert. Man benötigt insgesamt 2 Objekte:

1) den **Generator**: dieser liefert gleichverteilte `int` aus einem großen Intervall
2) die **Distribution:** Diese erzeugt aus den ganzzahligen Zufallszahlen des Generators die Zufallszahlen mit der gewünschten Verteilung**.**

Wir verwenden folgende 2 „gute" Generatoren (es gibt noch einige andere):

`std::mt19937_64` (Mersenne-Twister Generator, 64bit Version)
`std::random_device` (erzeugt echte Zufallszahlen)

`std::mt19937_64` erzeugt Pseudo-Zufallszahlen (d.h. er berechnet diese mit einer Formel), die aber sehr zufällig aussehen (und jeden Test auf Zufälligkeit bestehen!).
Vorteil: sehr hohe Geschwindigkeit, fast unbegrenzte Kapazität (ist aber periodisch)
Nachteil: bei gleichem Startwert (= Seed) erzeugt er immer dieselbe Zahlenreihe

`std::random_device` erzeugt echte Zufallszahlen
Nachteil: kann nur ein paar erzeugen, dann steht er still, bis er neue zufällige Werte (Entropie) aus Userinput, Spannungsschwankungen, ... sammeln kann.

Alle wichtigen mathematischen Verteilungen stehen als **Distribution** zur Verfügung:

```
std::uniform_int_distribution
std::uniform_real_distribution
std::bernoulli_distribution
std::binomial_distribution
std::exponential_distribution
std::normal_distribution
 …
```

Diese Funktionen sind eigentlich Funktionstemplates, bei denen man noch den gewünschten Datentyp angeben muss. Für die Initialisierung muss man zusätzlich einige Parameter angeben (welche und wie viele hängt von der Distribution ab) **.**

Wie baut man das nun zusammen:

1) Wir wollen eine diskrete Gleichverteilung simulieren (Fairer Würfel `wuerfel`)**:**

```
std::mt19937_64 gen;                    // Generator gen, ohne Seed
std::uniform_int_distribution<int>      // Verteilung mit int-Werten
     wuerfel{1, 6};                     // untere und obere Grenze
```

```
wuerfel(gen);                    // Zufallszahl
wuerfel(gen);                    // die nächste Zufallszahl
```

2) Eine Exponentialverteilung `w` mit Erwartungswert 6 (z.B. Wartezeit):

```
std::mt19937 gen;          // Generator gen, ohne Seed
std::exponential_distribution<double> // Verteilung
     wartezeit{1.0/6.};    // 1./Erwartungswert
wartezeit(gen);            // die Zufallszahl
wartezeit(gen);            // die nächste Zufallszahl
```

3) Eine Normalverteilung `nv` mit Erwartungswert 4.0, Standardabweichung 5.0:

```
std::mt19937_64 gen;       // Generator gen, ohne Seed
std::normal_distribution<double>   // die Verteilung
     nv{4.0, 5.0};         // Erwartungswert und  Standardabweichung
nv(gen);                   // die Zufallszahl
nv(gen);                   // die nächste Zufallszahl
```

Jeder Aufruf der Distributions-Funktion (mit dem Generator als Argument) erzeugt eine neue Zufallszahl. Will man nicht jedes Mal dieselbe Zahlenfolge generieren, sollte man den Generator mit einem zufälligen Seed-Wert instanzieren. Viele Programmierer verwenden dazu die aktuelle Zeit, aber einfacher geht das, indem man den Seed-Wert mit dem „echten“ Zufallszahlengenerator erzeugt:

```
std::random_device rd;         // „echter“ Zufallszahlen-Generator
std::mt19937_64 gen{rd()};     // Generator gen mit Seed rd()
```