Programmieren C++: Copy-Konstruktor, ...: Fortsetzung Grafik

Klaus Kusche

Dieses Beispiel ist eine *Fortsetzung des vorigen Beispiels*, nimm deine Lösung (oder meine Musterlösung) des vorigen Schrittes als Ausgangsbasis!

1. Unsere **`Rect`**-Klasse hat implizit einen *Copy-Konstruktor*, der eine *exakt idente Kopie* des Rechteckes an derselben Stelle erzeugt.

   Das wollen wir ändern: Erweitere die **`Rect`**-Klasse um einen **Copy-Konstruktor**, der die Kopie *in der Breite und in der Höhe um 3 Pixel kleiner* als das Original macht und auch *in jeder Farbkomponente um 5 dunkler* (beides solltest Du *gleich in der Initialisierungsliste* des Copy-Konstruktors machen). *Position und Flugrichtung* werden *unverändert* übernommen.

   Außerdem zeigt der implizite Copy-Konstruktor das kopierte Rechteck nicht gleich an (d.h. ruft kein **`draw()`** auf). Dein selbstgeschriebener Copy-Konstruktor sollte die *Kopie gleich zeichnen*!

   Wenn du ganz vorsichtig bist, *verhinderst* du mit einer **`max`**-Funktion (selbstgeschrieben oder aus **`<algorithm>`**), dass *Breite oder Höhe kleiner* **`1`** oder *eine Farb-Komponente kleiner* **`0`** wird.

   Da der resultierende Code nicht ganz kurz ist, sollte die *Implementierung nicht* im **`class`** erfolgen.

2. Schreib eine **Methode** **`moveOnTop`**, die ein *Rechteck als Parameter* hat und das eigene Rechteck *genau auf das als Parameter übergebene Rechteck legt* (d.h. das eigene Rechteck an der alten Position löscht, den Mittelpunkt des eigenen Rechteckes auf den Mittelpunkt des übergebenen Rechteckes setzt und dann das eigene Rechteck neu zeichnet).

   Übergib den *Rechteck-Parameter* vorläufig einmal *by Reference*.

3. Dazu habe ich folgendes **Hauptprogramm** geschrieben:

   - Ich verwende wieder ein *Array mit dynamisch angelegten Objekten*, in diesem Beispiel 40 Stück.
   - Auch eine *Zufalls-Hilfsfunktion* ist wieder ganz praktisch: **`int randPos(int range, int dist)`** soll eine Zufallszahl zwischen **`dist`** und **`(range-1)-dist`** (jeweis einschließlich) liefern.
   - Das Hauptprogramm speichert zuerst *im ersten Array-Element* ein Rechteck, dass *in Fenstermitte* liegt, *maximal gelb* ist, und ein *Fünftel der Fensterbreite* als Breite sowie ein *Fünftel der Fensterhöhe* als Höhe hat (die Flugrichtung ist egal, das Default **`0`** passt). Danach: Fenster *aktualisieren* und ein bisschen *warten*.
   - *Alle weiteren Array-Elemente* werden in einer Schleife mit einer *dynamisch angelegten Kopie* des jeweis vorigen Array-Elementes befüllt.

     Jedes neue Rechteck wird gleich nach dem Kopieren *an eine zufällige Stelle des Fensters verschoben*. Die neue Position wird durch Aufruf von **`randPos`**

mit der *Breite bzw. Höhe des Fensters* und der *Breite bzw. Höhe des Rechtecks* ermittelt, damit das Rechteck nicht über den Fensterrand hinausragt. Wieder nach jedem einzelnen Rechteck: Fenster *aktualisieren* und ein bisschen *warten*.

- Als nächstes habe ich das *letzte Rechteck in die Mitte der linken Fensterhälfte* gelegt und *alle anderen* Rechteckte in *verkehrter* Reihenfolge (*abwärts* zählende Schleife) mittels **`moveOnTop`** daraufgelegt. Wieder nach jedem einzelnen Rechteck: Fenster aktualisieren und ein bisschen warten.
- Dann kommt dasselbe *in umgekehrter Reihenfolge*: *Erstes Rechteck* in die Mitte der *rechten* Fensterhälfte, *alle anderen in richtiger Reihenfolge* mit **`moveOnTop`** drauf (wieder anzeigen und warten nach jedem Rechteck).
- Und als letztes habe ich *alle Objekte vernichtet*, in *verkehrter* Reihenfolge (Abwärts-Schleife) (wieder mit anzeigen und warten nach jedem Rechteck).

Wenn das ungefähr richtig aussieht: *Experimentiere mit der Parameter-Übergabe:*
Deklariere den Parameter von **`moveOnTop`** einmal ***"by Value"***
und einmal ***"by Reference"***.
Siehst du einen Unterschied? Kannst du den Unterschied erklären?