Programmieren 1 Übung: GUI-Programmierung

Klaus Kusche

TicTacToe

Wir wollen ein Programm schreiben, das bei einem TicTacToe-Spiel (oder einem anderen "*n Steine in einer Reihe*"-Spiel) die Anzeige (nicht die Spiellogik) übernimmt.
Dazu müssen wir das *Spielfeld selbst zeichen* (Alternativen wären z.B. ein Grid-Sizer mit lauter einzelnen Bitmap-Buttons oder eine Table, aber wir wollen ja das Zeichnen üben).

Der Programmrahmen ist einfach:

- Unsere *Applikationsklasse* samt **`OnInit`** sieht aus wie immer.
- Auch unsere *Hauptframe-Klasse* können wir direkt von der vorigen Übung übernehmen: Zwei *Buttons* ("*Reset*" und "*Exit*"), ein horizontaler *Sizer*, der die beiden Buttons nebeneinander anordnet, und ein vertikaler Sizer, der oben unser Spielfeld-Fenster und unten den horizontalen Sizer mit den Buttons plaziert.

  Unsere Hauptfenster-Klasse hat nur *ein Member*: Das *Subfenster* zur Pixel-Anzeige des Spielfeldes (ein Pointer auf ein Objekt unserer *selbstdefinierten Spielfeld-Klasse* **`MyBoard`**, wird im Konstruktor mit einem neu angelegten **`MyBoard`** initialisiert).
- Das einzig neue ist die *Statuszeile*: Sie wird mit **`CreateStatusBar()`** aktiviert. Das liefert einen Pointer auf ein **`wxStatusBar`**-Objekt. Unser Spielfeld-Objekt bekommt diesen Pointer gleich beim Konstruktor übergeben (siehe unten), **`CreateStatusBar()`** steht daher als Argumentwert direkt im **`MyBoard`**-Konstruktoraufruf in der Initliste des Konstruktors unseres Hauptframes.
- Jeder *Button* bekommt wieder eine *Id* und einen *Event-Handler* samt **`Bind`** im Konstruktor des Hauptfensters: Der Handler für "*Exit*" ruft wie gehabt **`Close`** auf, der "*Reset*"-Button nur ruft die **`Reset`**-*Methode unseres Spielfeld-Subfensters* auf.

Die ganze Komplexität steckt in unserer *selbstdefinierten Spielfeld-Klasse* **`MyBoard`**:

Da wir keine vorgefertigte Spezial-Funktionalität nutzen können,
ist sie von der allgemeinen Basis-Fenster-Klasse **`wxWindow`** abgeleitet.

Die Klasse hat folgende *private Member*:

- Das *Spielfeld*, ein statisch dimensioniertes, *zweidimensionales* **`int`**-*Array*.
  Die Elemente enthalten **`-1`** für frei, **`0`** für ein Kreuzerl und **`1`** für ein Ringerl.
- Eine **`int`**-Variable für den *Spieler*, der gerade am Zug ist
  (ebenfalls **`0`** für Spieler "Kreuzerl" und **`1`** für Spieler "Ringerl").
- Die Anzahl der noch *freien Felder* am Spielfeld
  (auch ein **`int`**, wird auf **`0`** gesetzt, wenn das Spiel zu Ende ist).
- Einen **`wxStatusBar`**-*Pointer*, der auf das Statusbar-Objekt des Hauptfensters zeigt
  (weil die Spielfeld-Klasse ja den Text in der Statuszeile ändern muss).

Die Klasse hat folgende *private Hilfsmethode*:

- **Winner** (keine Parameter, Returnwert **bool**):

  Diese Methode hat nichts mit wxWidgets oder dem GUI zu tun: Sie soll einfach **true** zurückliefern, wenn der aktuelle Spieler irgendwo am Spielfeld *genug Steine in einer Linie* hat (zuerst überlegen, das sind viele trickreiche Schleifen und **if**'s!)

  Man könnte natürlich auch die Position des aktuell gesetzten Steines übergeben und nur dessen Nachbarn in jeder Richtung prüfen.
  Das ist deutlich schneller, aber das ist kaum einfacher.

Die Klasse hat folgende *öffentliche Methoden*:

- Einen **Konstruktor** (Parameter sind *zwei Pointer* auf das *übergeordnete Fenster* und auf die *Statuszeile* (Typ **wxStatusBar**)):
  - In der Initialisierungsliste wird zuerst der Konstruktor der *Vaterklasse* **wxWindow** aufgerufen (dabei sollte man das Style-Flag **wxFULL_REPAINT_ON_RESIZE** angeben, sonst gibt es optische Probleme, wenn man die Fenstergröße ändert) und dann der *Statuszeilen-Parameter* in der dafür vorgesehenen Member-Variable gespeichert.
  - Dann sollte man eine *Mindestgröße* für das Spielfeld-Fenster setzen und es durch Aufruf der Methode **Reset** *initialisieren und zeichnen*.
  - Schließlich kommen die beiden **Bind**-*Aufrufe* zum Anbinden der beiden Eventhandler: Der *Maus-Event* für **OnClick** heißt **wxEVT_LEFT_UP**, der *Repaint-Event* für **OnPaint** heißt **wxEVT_PAINT**, beide werden ohne spezielle Id verbunden.
- **Reset** (kein Parameter, kein Returnwert):
  - Als erstes setzt diese Methode die Member für Spielfeld, Spieler und freie Felder auf die *Anfangswerte*.
  - Dann wird in die *Statuszeile* eine Meldung geschrieben, dass der erste Spieler am Zug ist.
  - Zuletzt wird mit **Refresh** ein *Neuzeichnen des Spielfeldes* ausgelöst.
- **OnClick** (**wxMouseEvent**-Referenz-Parameter, kein Returnwert):

  Das ist der Event-Handler, der aufgerufen wird, wenn man mit der Maus in das Spielfeld klickt. Er hat viele Schritte:
  - Wenn die *Anzahl der freien Felder* schon 0 ist, kann man *sofort zurückkehren*: Das Spiel ist schon aus, der Klick wird *ignoriert*.
  - Sonst holt man sich mit **GetClientSize** die Spielfeldfenster-Größe und aus dem Event-Parameter mit **GetPosition** die Klick-Position.

    Liegt der Klick außerhalb des Fensters, kann man auch sofort zurückkehren, sonst muss man die Klick-Koordinaten *auf Brett-Koordinaten (d.h. x- und y-Index) umrechnen* (*"Welches Feld wurde angeklickt?"*).
  - Wenn dieses Feld *schon belegt* ist, gibt man eine Fehlermeldung aus (mittels **wxMessageBox**) und kehrt zurück.

- Sonst *belegt* man das Feld mit einem Stein des aktuellen Spielers und zählt die freien Felder um eins herunter.
- Danach wird ein *Neuzeichnen des Spielfeldes* ausgelöst und mittels **wxSafeYield()** dafür gesorgt, dass dies auch *sofort* passiert (*vor* den Message-Boxen in den nächsten Punkten).
- Dann ruft man unser **Winner** auf. Ist das Ergebnis **true**, gibt man eine *Siegesmeldung* aus (Message-Box und Statuszeile) und setzt die Anzahl der freien Felder auf **0**, um *weitere Züge zu unterbinden*.
- Sonst prüft man, ob das das *allerletzte freie Feld* war. Wenn ja, gibt man eine *"unentschieden"-Meldung* aus (Message-Box und Statuszeile).
- Wenn nein, *wechselt* man den aktuellen Spieler und schreibt eine *Aufforderung zum nächsten Zug* in die Statuszeile.

- **OnPaint** (**wxPaintEvent**-Referenz-Parameter, kein Returnwert):

  Das ist der Event-Handler, den wxWidgets einerseits aufruft, wenn wir mit **Refresh()** bewusst ein *Neuzeichnen des Fensters* auslösen, und andererseits *automatisch intern aufruft*, wenn es das Fenster intern neu zeichnen muss (z.B. bei Resize oder nachdem es verdeckt war).

  Diese Methode *zeichnet* den gesamten Inhalt des Spielfeld-Fensters neu: Die *Trennstriche* und die *bisher gesetzten Ringerl und Kreuzerl*. Dazu dienen vor allem **wxDC**-Methoden:

  - **GetSize** (Größe des Fensters in Pixel, zum *Berechnen der Pixel-Koordinaten* der Kreuzerl, Ringerl und Trennstriche)
  - **Clear** (alles löschen, zuvor **SetBackground** auf **wxWHITE_BRUSH** !)
  - **SetPen**: Das setzt den "Stift" für die nachfolgenden Zeichenoperationen, d.h. Strichfarbe, -breite und -art. Für den richtigen roten, grünen und schwarzen Stift legt man sich am besten drei lokale statische Variablen vom Typ **wxPen**-Pointer an, die man mit **wxThePenList->FindOrCreatePen** entsprechend initialisiert.
  - **DrawLine** und **DrawEllipse** oder **DrawCircle**

  Zum Aufruf dieser Methoden deklariert **OnPaint** als erstes ein *lokales* **wxPaintDC**-Objekt *für das eigene Fenster*.

Hinweise:

- Im Idealfall sollten sowohl die *Spielfeldgröße* als auch die zum Sieg erforderliche *Anzahl von Steinen in einer Reihe* ganz oben als *Konstanten* definiert sein, damit das Programm von simplem TicTacToe (beides 3) bis Gomoku (Fünferreihe auf 19*19-Brett oder größer) anpassbar ist.
- Die Behandlung der *Rundung* von Fenster- auf Spielfeld-Größe im Grafikfenster, d.h. jener Pixel, die als Divisionsrest überbleiben, wenn man die Größe des Grafikfensters durch die Anzahl der Felder teilt, ist ziemlich lästig (man sollte das Raster zentrieren und die Rest-Pixel symmetrisch auf beide Ränder verteilen).

  Du darfst dieses Problem für den Anfang *ignorieren* (d.h. entwerder das Raster links oben im Grafikfenster zeichnen oder die Größe des Grafikfensters auf eine schön teilbare Pixelzahl fixieren).