Praktikumstermin Nr. 11, INF: Operator-Methode, Vererbung

Abgabe im GIP-INF Praktikum der Woche 8.-12.1.2024.

(Pflicht-) Aufgabe INF-11.01: Operator-Methode `operator*`

Kopieren Sie die Dateien `MyCanvas.h`, `MyCanvas.cpp`, `MyRectangle.h` und `MyRectangle.cpp` aus ihrem letzten Praktikum in ein neues Projekt.

Kopieren Sie die Dateien `gip_mini_catch_v_2_3.h` , `main.cpp` , `gip_mini_catch_heap_delete.h` , `test_MyCanvas.cpp` , `test_MyRectangle.cpp` , `test_operator_MyRectangle.cpp` aus Ilias in das Projekt.

Programmieren Sie für die Klasse `MyRectangle` einen `operator * (unsigned int i)`, so dass man für ein `MyRectangle r1` einen Ausdruck `r1 * i` (z.B. `r1 * 3`) schreiben kann und dies bedeutet, dass ein *neues* Rechteck entsteht (und *nicht* das aktuelle Rechteck modifiziert wird), welches die gleiche linke obere Ecke `(x1, y1)` hat wie das ursprüngliche Rechteck `r1`, aber um den Faktor `d` verlängerte oder verkürzte Kanten (z.B. im Fall `d = 2` doppelt so lange Kanten).

Testen Sie Ihr Programm, indem Sie schauen, dass alle automatisch durchgeführten Tests erfolgreich sind.

(Pflicht-) Aufgabe INF-11.02: Vererbung, Klasse `MyFramedRectangle`

Kopieren Sie die Dateien `MyCanvas.h`, `MyCanvas.cpp`, `MyRectangle.h` , `test_MyCanvas.cpp` , `test_MyRectangle.cpp` , `test_operator_MyRectangle.cpp` und `MyRectangle.cpp` aus der vorigen Aufgabe in ein neues Projekt.

Kopieren Sie zusätzlich die Datei `test_MyFramedRectangle.cpp` aus Ilias in das Projekt.

Legen Sie neue Dateien `MyFramedRectangle.h` und `MyFramedRectangle.cpp` an und programmieren Sie dort eine Klasse `MyFramedRectangle`, die von `MyRectangle` erbt.

Die Klasse `MyFramedRectangle` besitze keine eigenen Attribute, sondern nutze die Attribute der Basisklasse `MyRectangle`.
Prüfen Sie, ob der Zugriffsschutz der Basisklassen-Attribute so ist, dass diese Attribute in der abgeleiteten Klasse sichtbar sind und trotzdem nicht von außen sichtbar sind.

Der Konstruktor von `MyFramedRectangle` habe die gleichen Parameter wie der Konstruktor der Basisklasse `MyRectangle` und initialisiere die Attribute durch Aufruf des Basisklassen-Konstruktors mit den Parametern des eigenen Aufrufs.

Die `draw()` Methode von `MyFramedRectangle` soll erst einmal mit Hilfe der `draw()` Methode von `MyRectangle` ein Rechteck zeichnen.
Dann soll sie die äußeren Hash `#` Zeichen des Rechtecks mit Pluszeichen `+` überschreiben, so dass das Rechteck einen äußeren Rahmen aus Pluszeichen hat (aber seine Größe dadurch nicht verändert).

Testen Sie Ihr Programm, indem Sie schauen, dass alle automatisch durchgeführten Tests erfolgreich sind. Das Hauptprogramm in `main.cpp` zeichnet außerdem ein `MyFramedRectangle` mit zufälligen Koordinaten.

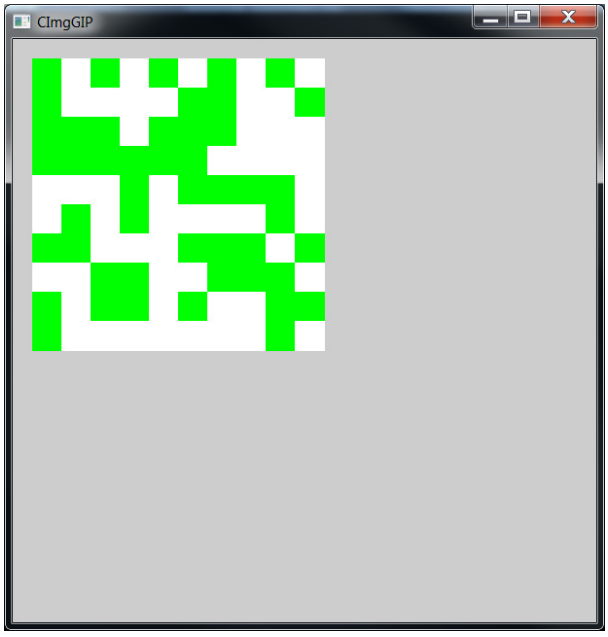
(Freiwillige) Aufgabe INF-11.03: Conway's *Game of Life*

Schreiben Sie unter Benutzung der *`CImg`* Library (mittels der Headerdatei `CImgGIP06.h`, die Sie schon aus einem vorherigen Praktikumsversuch kennen) ein C++ Programm, welches das Spiel *Game of Life* realisiert. Dieses stellt die zeitliche Entwicklung einer „Zellkolonie" dar.

https://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens

Die Größe des Feldes sei als Konstante `grid_size` vorgegeben. Benutzen Sie ein zweidimensionales statisches Array zur Realisierung des Spielfeldes.

Das Spiel startet je nach Benutzereingabe entweder mit einer zufälligen Belegung des Feldes oder mit einer vorgegebenen Belegung. Benutzen Sie für die zufällige Belegung den Funktionsaufruf `gip_random(0,1);` um jeweils eine Zufallszahl „entweder 0 oder 1" zu erzielen.



Orientieren Sie sich an folgendem Programmgerüst (welches auch als Datei in Ilias vorliegt) für Ihre Lösung:

```
#include <iostream>

#define CIMGGIP_MAIN
#include "CImgGIP08.h"

using namespace std;
using namespace cimg_library;

const int grid_size = 10; // Anzahl an Kaestchen in x- und y-Richtung
const int box_size = 30;  // size der einzelnen Kaestchen (in Pixel)
const int border = 20;   // Rand links und oben bis zu den ersten Kaestchen (in Pixel)

// Prototyp der Funktionen zum Vorbelegen des Grids ...
void grid_init(bool grid[][grid_size]);

int main()
{
    bool grid[grid_size][grid_size] = { 0 };
    bool next_grid[grid_size][grid_size] = { 0 };

    // Erstes Grid vorbelegen ...
    grid_init(grid);

    while (gip_window_not_closed())
    {
        // Spielfeld anzeigen ...
        // gip_stop_updates(); // ... schaltet das Neuzeichnen nach
                               //     jeder Bildschirmänderung aus

        // TO DO

        // gip_start_updates(); // ... alle Bildschirmänderungen (auch die
```

```
                                //     nach dem gip_stop_updates() ) wieder anzeigen
        gip_sleep(3);

        // Berechne das naechste Spielfeld ...
        // Achtung; Für die Zelle (x,y) darf die Position (x,y) selbst *nicht*
        // mit in die Betrachtungen einbezogen werden.
        // Ausserdem darf bei zellen am rand nicht über den Rand hinausgegriffen
        // werden (diese Zellen haben entsprechend weniger Nachbarn) ...

        // TO DO


        // Kopiere das naechste Spielfeld in das aktuelle Spielfeld ...

        // TO DO

    }
    return 0;
}

void grid_init(bool grid[][grid_size])
{
    int eingabe = -1;
    do {
        cout << "Bitte waehlen Sie die Vorbelegung des Grids aus:" << endl
            << "0 - Zufall" << endl
            << "1 - Statisch" << endl
            << "2 - Blinker" << endl
            << "3 - Oktagon" << endl
            << "4 - Gleiter" << endl
            << "5 - Segler 1 (Light-Weight Spaceship)" << endl
            << "6 - Segler 2 (Middle-Weight Spaceship)" << endl
            << "? ";
        cin >> eingabe;
        cin.clear();
        cin.ignore(1000, '\n');
    } while (eingabe < 0 || eingabe > 6);

    if (eingabe == 0)
    {
        // Erstes Grid vorbelegen (per Zufallszahlen) ...

        // TO DO

    }
    else if (eingabe == 1)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '*', '.', '*' },
            { '.', '*', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
```

```
    }
    else if (eingabe == 2)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '.', '*', '.' },
            { '.', '*', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 3)
    {
        const int pattern_size = 8;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '.', '.', '*', '*', '.', '.', '.' },
            { '.', '.', '*', '.', '.', '*', '.', '.' },
            { '.', '*', '.', '.', '.', '.', '*', '.' },
            { '*', '.', '.', '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '.', '.', '.', '*' },
            { '.', '*', '.', '.', '.', '.', '*', '.' },
            { '.', '.', '*', '.', '.', '*', '.', '.' },
            { '.', '.', '.', '*', '*', '.', '.', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+1] = true;
    }
    else if (eingabe == 4)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '.', '.', '*' },
            { '*', '*', '*' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 5)
    {
        const int pattern_size = 5;
        char pattern[pattern_size][pattern_size] =
        {
            { '*', '.', '.', '*', '.' },
            { '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '*' },
            { '.', '*', '*', '*', '*' },
            { '.', '.', '.', '.', '.' },
```

```
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 6)
    {
        const int pattern_size = 6;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '*', '*', '*', '*' },
            { '*', '.', '.', '.', '.', '*' },
            { '.', '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '*', '.' },
            { '.', '.', '*', '.', '.', '.' },
            { '.', '.', '.', '.', '.', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
}
```

Eine Zelle wird in einem leeren Feld neu geboren, wenn im vorigen Grid drei der umgebenden Nachbarzellen belebt waren.

Nachbarzellen: Eine Zelle, die nicht am Rand des Spielfelds liegt, hat 8 Nachbarn: oben, unten, links, rechts und 4x diagonal. Die Zelle selbst zählt nicht zu ihren eigenen Nachbarn. Zellen am Rand des Spielfelds haben weniger Nachbarn.

Eine Zelle bleibt in einem Feld am Leben, wenn im vorigen Grid zwei oder drei der umgebenden Nachbarzellen belebt waren.

In allen anderen Fällen wird keine Zelle geboren bzw. eine dort lebende Zelle stirbt wegen zu vieler oder zu weniger Nachbarn, d.h. das Feld wird im nächsten Spielfeld unbewohnt sein.

Stellen Sie das Spielfeld über die Zellgenerationen hinweg graphisch dar.

Hinweis: Sie werden sehen, dass die Aktualisierungen des "Spielfelds" recht langsam "Kästchen für Kästchen" passieren … Das ist eine Konsequenz der GIP-spezifischen Änderungen der CImg Library (http://cimg.eu/), die ich vorgenommen habe: Ich "zwinge die Library", jede Graphik-Operation auch sofort auf dem Bildschirm sichtbar zu machen. Das bremst die Library sehr stark, hat aber den Vorteil, dass Sie im Debugger jeden Schritt des Programms einzeln nachverfolgen können und wirklich auf dem Bildschirm sehen, was ihre einzelnen Graphik-Operationen wie `gip_draw_rectangle()` *tun. Normalerweise wäre das nicht der Fall …*
Wenn Sie dann sicher sind, dass ihr Code wohl funktioniert, dann fügen Sie die im Programmrahmen noch auskommentierten Funktionsaufrufe `gip_stop_updates()` *und* `gip_start_updates()` *hinzu.*
`gip_stop_updates()` *"schaltet CImg in seinen üblichen Modus um": Graphik-Änderungen*

*("Updates") werden **nicht** mehr direkt angezeigt, sondern nur noch intern durchgeführt und gespeichert.*
Erst mit dem Aufruf von `gip_start_updates()` wird die Library wieder in den "GIP Modus geschaltet": Alle "in der Zwischenzeit" intern gespeicherten Änderungen werden dann mit einem Schlag sichtbar gemacht und alle danach stattfindenden Änderungen werden dann auch wieder sofort sichtbar gemacht …

Testläufe: *(Animation der vorgegebenen Muster siehe Wikipedia Seite)*

```
Bitte waehlen Sie die Vorbelegung des Grids aus:
0 - Zufall
1 - Statisch
2 - Blinker
3 - Oktagon
4 - Gleiter
5 - Segler 1 (Light-Weight Spaceship)
6 - Segler 2 (Middle-Weight Spaceship)
?
```