Übungsblatt 4

In dieser Übung arbeiten wir ein weiteres mal mit der `ViewPortGL`-Umgebung. Ich habe die Implementationen ein kleines bisschen erweitert und geändert, weil da noch unschöne Warnungen beim Compilieren erzeugt wurden, die eigentlich nicht nötig sind. Außerdem habe ich drei kleine Funktionen hinzugefügt, die Sie aber erst mal nicht weiter beachten müssen. Verwenden Sie aber bitte die aktualisierten Klassen in Ihren Übungen.

1. Wir hatten in den letzten Übungen gesehen, dass sich Farben über Rot-, Grün- und Blauwerte definieren lassen. Nun ist es nicht immer schön, wenn man mit drei `int`-Werten hantieren muss. Zur schöneren Darstellung von Farben sollen Sie darum in dieser Aufgabe eine Klasse `Color` implementieren.

   (a) Erstellen Sie eine Klasse `Color`, die drei privaten `int`-Attribute `red`, `green` und `blue` besitzt. Fügen Sie dieser Klasse einen Konstruktor mit dem Kopf

   ```
   Color(int r, int g, int b)
   ```

   hinzu, der diese Attribute initialisiert. Der Konstruktor soll prüfen, ob sich alle drei Farbkomponenten im Bereich $\{0, \ldots, 255\}$ befinden. Ist das nicht so, so soll eine Exception vom Typ `logic_error` mit aussagekräftiger Fehlermeldung geworfen werden.

   Fügen Sie außerdem einen Default-Konstructor

   ```
   Color()
   ```

   hinzu, der alle drei Farbkomponenten mit 0 initialisiert.

   (b) Fügen Sie der Klasse die folgenden Getter- und Setter-Methoden hinzu:

   - `int getRed()`
   - `int getGreen()`
   - `int getBlue()`
   - `void setRed(int r)`
   - `void setGreen(int g)`
   - `void setBlue(int b)`

   Die Setter-Methoden prüfen dabei wieder, ob die Eingaben im richtigen Bereich sind (siehe oben) und werfen im Fehlerfall wieder entsprechende Exceptions.

   (c) Fügen Sie der Klasse eine weitere Methode mit dem Kopf

   ```
   Color darken(int amount)
   ```

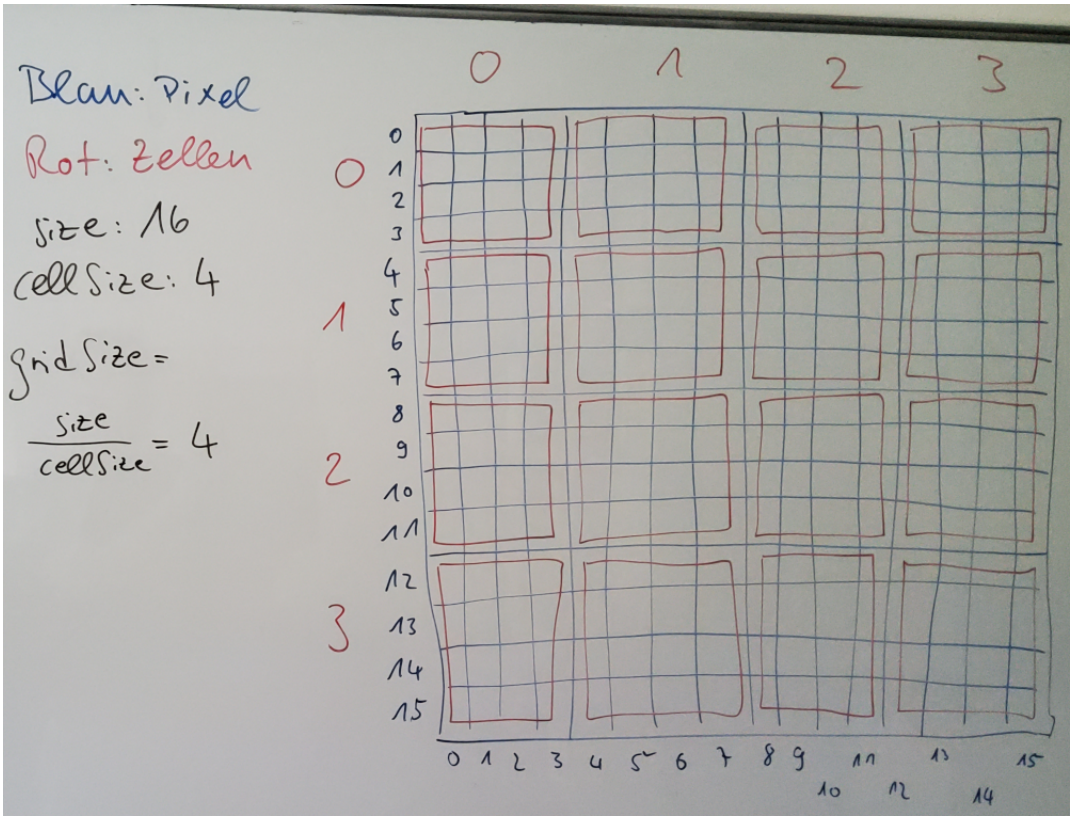
   hinzu, die eine neue Farbe zurückliefert, die diese (also im Sinne von `this`) Farbe um den angegebenen Faktor `amount` abdunkelt. Dieses Argument muss wieder der Menge $\{0, \ldots, 255\}$ entstammen. Die zurückgelieferte Farbe hat Farbanteile, die aus den Farbanteilen von `this` durch Abzug von `amount` hervorgehen. Ist der Rotanteil von `this` also zum Beispiel gleich `100` und ist `amount` gleich `70`, so hat die zurückgelieferte Farbe den Rotanteil `30`. Der zurückgelieferte Farbanteil soll aber niemals negativ sein. Ist die oben beschriebene Differenz also kleiner als `0`, so soll dieser Farbanteil auf `0` in der neuen Farbe gesetzt werden.

   Schreiben Sie analog dazu eine Methode mit dem Kopf

   ```
   Color lighten(int amount)
   ```

   Achten Sie darauf, dass die hier zurückgelieferte Farbe keinen Farbanteil größer `255` enthält!

2. In dieser Aufgabe sollen Sie die Klasse `ViewPortGL` in einer Unterklasse `GridViewer` erweitern. Dabei soll eine zusätzliche Funktionalität eingebaut werden, die anhand der folgenden Darstellung erläutert wird:



Wie in Übungsblatt 1 besprochen, können in einem `ViewPortGL` einzelne Pixel adressiert werden, so wie im Bild über das blaue Koordinatensystem dargestellt.

In der zu implementierenden Unterklasse `GridViewer` soll ein zweites Koordinatensystem (im Bild rot) hinzugefügt werden über das sich quadratische Gruppen von Pixeln ansprechen lassen. Diese Quadrate nennen wir im Folgenden *Zellen*. Um Rundungsproblemen aus dem Weg zu gehen, fordern wir in der Klasse, dass alle verwendeten Größen auf Zweierpotenzen aufbauen.

Pro Zelle merkt sich der `GridViewer` einen `int`-Wert, der auf unterschiedliche Weise im darunter liegenden `ViewPortGL` dargestellt werden kann.

Im Moodle finden Sie die Headerdatei einer Klasse `GridViewer`, die von `ViewPortGL` abgeleitet ist.

Instanzen dieser Klasse sollen immer einen quadratischen View-Port repräsentieren, dessen Seitenlänge eine Zweierpotenz ist, die 4 nicht unter- und 1024 nicht überschreitet.

Ihre Aufgabe besteht darin, die Definitionen zu den Deklarationen im Header zu programmieren. Wie das geschehen soll, wird in den folgenden Teilaufgaben beschrieben:

(a) Implementieren Sie zunächst den Konstruktor mit dem Kopf

```
GridViewer(string titleP, int size, int cellSizeP)
```

der zunächst den Konstruktor der Superklasse aufruft (wobei Höhe und Breite des zu erstellenden `ViewPortGL` beide gleich `size` sind).

Die beiden konstanten Attribute sollen mit den Werten `cellSizeP` für `cellSize` und `size/cellSizeP` für `gridSize` initialisiert werden.

Der Konstruktor prüft, ob `size` eine Zweierpotenz im Bereich zwischen 4 und 1024 ist. Ist dem nicht so, so wirft er einen aussagekräftigen `logic_error`.

Der Parameter `cellSizeP` soll ebenfalls eine Zweierpotenz zwischen 4 und 1024 sein und darf zusätzlich höchstens so groß sein wie `size`. Ihr Konstruktor soll auch diesen Parameter prüfen und im Fehlerfall eine eine Ausnahme erzeugen.

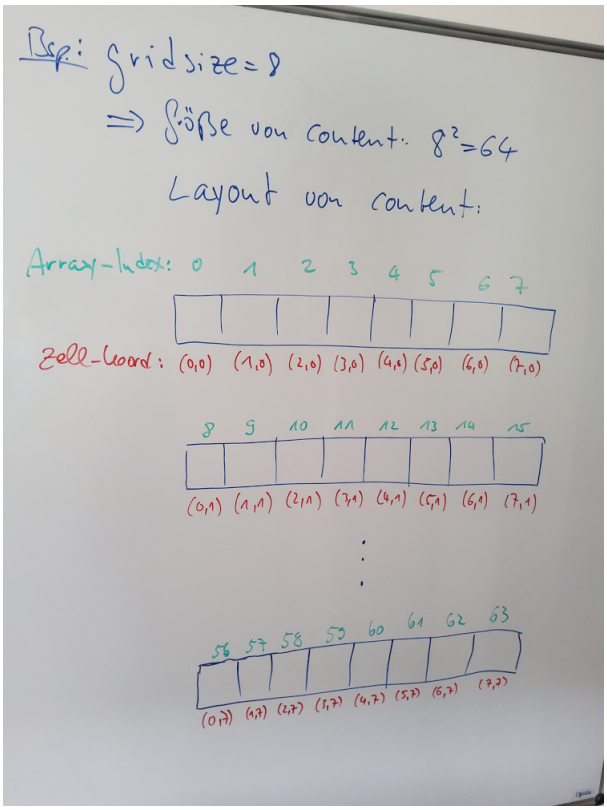
Im Attribut `content` wird sich der Inhalt der Zellen gemerkt. Dieses Array muss im Konstruktor initialisiert werden, d.h. es muss Speicher der Größe $\texttt{gridSize}^2$ `int` dafür reserviert werden. Zudem setzt der Konstruktor den Inhalt aller Zellen zu beginn auf 0.

(b) Implementieren Sie einen passenden, sinnvollen Destruktor für die Klasse `GridViewer`!

(c) Implementieren Sie eine Methode

```
int getCell(int x, int y)
```

Die den Inhalt der Zelle mit den Koordinaten `(x, y)` zurückgibt. Die Zelleninhalte werden sich im Attribut `content` gemerkt. Es handelt sich dabei um ein eindimensionales `int`-Array der Größe `gridSize`^{2} (sh. Konstruktor). Denken Sie sich die Zellen in diesem Array wie folgt angeordnet vor:



Implementieren Sie in diesem Sinne auch die Methode

```
void setCell(int x, int y, int value)
```

Beide Methoden sollen einen aussagekräftigen `logic_error` werfen, wenn die Koordinaten `(x,y)` außerhalb des gültigen Bereichs liegen.

(d) Implementieren Sie die private Methode mit dem Kopf

```
Color colorFor(int value)
```

die einem `int`-Wert `value` eine Farbe wie folgt zuordnet:

- Der Rotanteil der Farbe wird den Bits 31 bis 24 entnommen.
- Der Grünanteil der Farbe wird den Bits 16 bis 23 entnommen.
- Der Blauanteil der Farbe wird den Bits 8 bis 15 entnommen.

Die Bits 0 bis 7 werden ignoriert.

(e) In der Vorlage für `GridViewer.cpp` ist die Methode `draw` vorimplementiert. Diese Methode läuft über alle Zellen und veranlasst, dass sie jeweils gezeichnet werden. Dazu ruft sie die Methoden `prepareCell` und `prepareCellBorder` auf. Lesen und verstehen Sie, was in der `draw`-Methode passiert!
Implementieren Sie die Methode mit dem Kopf

```
void prepareCell(int x, int y)
```

die veranlasst, dass für den Wert, der den Koordinaten `(x, y)` zugeordnet ist (steht in `content`), ein farbiges Quadrat für die Zelle gezeichnet wird. Die Farbe wird dabei über die Methode `colorFor` bestimmt.

Die Methode `prepareCellBorder` können Sie leer lassen (so dass sie nichts tut), falls sie die folgende Bonusaufgabe nicht bearbeiten wollen.

Im Moodle finden Sie eine Datei `Ue4Main.cpp`, die einen `GridViewer` instanziiert und in zwei Funktionen für Testbilder zur Verfügung stellt. Wenn Sie alles richtig gemacht haben, sollte das Programm jetzt compilieren und die Testbilder erzeugen. Testen Sie Ihre Implementation!

(f) (Bonusaufgabe)

Implementieren Sie die Methode mit dem Kopf

```
void prepareCellBorder(int x, int y)
```

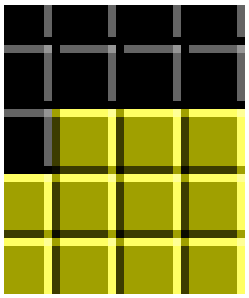
die eine ausgefüllte Zelle mit einem Rand versieht, der eine Beleuchtung, die von rechts oben kommt andeutet. Bestimmen Sie dazu zuerst die Farbe `c`, die der Zelle zugeordnet ist (Wert aus `content` auslesen, Methode `colorFor` aufrufen).

Hellen Sie diese Farbe auf (`Color::lighten`) und zeichnen Sie eine waagerechte Linie am oberen Rand der Zelle so wie eine senkrechte Linie am rechten Rand der Zelle mit der aufgehellten Farbe.

Verfahren Sie analog mit einer abgedunkelten Farbe am unteren und linken Zellenrand.

Wenn Sie wollen können Sie die rechte, obere Ecke noch stärker aufhellen und die linke, untere noch stärker abdunkeln.

Beispiel:



Wenn Sie alle Aufgaben bis hierhin gemacht haben, sollten Sie etwa in der Größenordnung von 130 Zeilen Code geschrieben haben. Das ist nicht wirklich viel. Darum gibt es auf den folgenden Seite weitere Aufgaben. Das sind alles Bonusaufgaben, ich empfehle aber dringend, sie trotzdem zu machen, gerade wenn Sie Schwierigkeiten mit der Programmierung haben!

3. (Bonusaufgabe)

In dieser Aufgabe sollen Sie eine Unterklasse von `GridViewer` mit dem Bezeichner `PaletteGridViewer` entwickeln. Diese Unterklasse besitzt zusätzlich die Funktion eine *Farbpalette* zu verwalten. Dabei handelt es sich um eine Möglichkeit eine endlichen Menge von selbst definierten Farben `int`-Werte zuzuordnen und die Farben über diese Werte anzusprechen.

In der Klasse `GridViewer` kann auf standardmäßige Art und Weise der Farbwert für einen `int` ermittelt werden (sh. Aufgabe 2d). Die Klasse `PaletteGridViewer` ermöglicht es, einzelnen `int`-Werten spezielle Farben zuzuweisen, die sich nicht unmittelbar aus den Bits des `int`-Werts ablesen lassen.

Beispiel: Es sei der `int`-Wert 879470848 gegeben, in Binärdarstellung ist das

```
00110100 01101011 10101001 00000000
```

und nach der Logik der `forColor`-Methode von `GridViewer` entspricht das einem Rot-Anteil von 52, einem Grün-Anteil von 107 und einem Blau-Anteil von 169.

Manchmal möchte man aber Farben nicht immer über Bits zusammenstellen, sondern schnellen Zugriff auf spezielle Farben haben (etwa: Weiß, Schwarz, Blau, Braun, ...). Dafür soll man in der Klasse `PaletteGridViewer` solchen speziellen Farben Indizes zuweisen können (Methode `void setColor(unsigned int value, Color c)` und diese Farben über diese Indizes auch ansprechen können (Methode `Color getColor(unsigned int value)`).

Zum Beispiel soll es möglich sein mit dem Aufruf `setColor(7, Color(255, 255, 0))`, dieser gelben Farbe den Wert 7 zuzuordnen. Bemerken Sie, dass ein `GridViewer` den Wert 7 mit der Farbe Schwarz interpretieren würde, weil die Bits 8 bis 31 in der Binärdarstellung von 7 alle 0 sind.

Der `PaletteGridViewer` arbeitet genauso wie der `GridViewer`, kann nun aber die Farbwerte 0 bis 255 anders, und zwar benutzerdefiniert interpretieren.

Im Moodle finden Sie die Datei `PaletteGridViewer.h`. Implementieren Sie die dort deklarierten Methoden wie folgt:

(a) Implementieren Sie den Konstruktor

```
PaletteGridViewer(string title, int size, int cellSize, int paletteSizeP)
```

der die Argumente`title`, `size` und `cellSize` gleich an den Konstruktor von `GridViewer` weiterreicht. Außerdem prüft er, ob `paletteSizeP` im Bereich $\{1, \ldots, 256\}$ liegt. Falls nein, wird ein aussagekräftiger `logic_error` geworfen. Der Konstruktor reserviert außerdem Speicher für das `palette`-Array, in dem die Zuordnung der Farbindizes zu konkreten Farben gespeichert wird.

(b) Implementieren Sie die Methoden

```
void setColor(unsigned int value, Color c)
```

und

```
Color getColor(unsigned int value)
```

mit deren Hilfe man Farben einen Index zuweisen kann, bzw. abfragen kann, welche Farbe unter einem Index gespeichert ist. Beide Methoden prüfen den Parameter `value` darauf hin, ob er im Bereich $\{0, \ldots, 255\}$ liegt (Exception, falls nicht!).

(c) Überschreiben Sie die virtuelle Methode `colorFor` von `GridViewer` in `PaletteGridViewer`, so dass die Farbe, die in `palette` gespeichert ist zurückgegeben wird, falls `value` sich im Bereich $\{0, \ldots, 255\}$ befindet. Ansonsten soll die Farbe zurückgegeben werden, die auch ohne Überschreibung zurückgegeben würde. Versuchen Sie dabei keinen Code zu duplizieren!

In `Ue4Main.cpp` ist eine auskommentierte Funktion, die mit einem `PaletteGridViewer` arbeitet. Kommentieren Sie die Funktion ein und testen Sie damit Ihre Implementation!