Milestone 1

Die Frist für die Abgabe des Milestone 1 ist Mittwoch der 13.11.2019 23:59 und erfolgt über OLAT

In diesem Milestone ist das Ziel die Grundlagen von C++ zu erlernen. Die Aufgaben sollen durch ein Team von 3 bis 4 Studierenden bearbeitet werden. Um die Teamarbeit am Anfang zu erleichtern sind Aufgabe 1 und 2 unabhängig voneinander und müssen erst in Aufgabe 3 zusammengeführt werden.

Aufgabe 1 Zellulärer Automat $(\Sigma\ 100\ P.)$

a) Erstellt ein statisches zweidimensionales Array, welches ein Feld mit 30x30 Zellen darstellt, und füllt jede Zelle mit einer rein zufälligen[1] ganzen Zahl aus dem Intervall [0, 9]. Erstellt zusätzlich ein eindimensionales dynamisches Array mit der gleichen Zellen Anzahl. Kopiert dann die Inhalte der Zellen des statischen Arrays in die des dynamischen Arrays.
Dabei soll das statische Array vor dem Kopieren und das dynamische Array nach dem Kopieren auf der Standardausgabe in Form einer Matrix ausgegeben werden. (10)

b) Erstellt eine Benutzersteuerung für ein Konsolenprogramm mit der jede Aktion aus Aufgabenteil (a) einzeln und in beliebiger Reihenfolge ausgeführt werden kann. Zusätzlich soll eine Aktion implementiert werden mit welcher man das Konsolenprogramm verlassen kann. (10)

c) Erstellt eine Klasse für den Zellulären Automaten, welcher Cornway's Game of Life umsetzt. Dabei soll die Feldgröße 30x30 Zellen betragen, jedoch vom Benutzer des Programmes verändert werden können. Es sollen im Wesentlichen nur zwei Arrays verwendet werden, eines für den alten Zustand und eines für den neuen Zustand. Der Benutzer soll den Zustand der Zellen über ihre Koordinaten einzeln setzen (lebend, tot), aber auch ändern können (lebend zu tot, tot zu lebend). Erstellt eine Funktion, welche die Evolution für genau einen Zeitschritt durchführt, sowie eine Funktion, welche einen String erstellt, der den Zustand des Automaten in Form einer Matrix repräsentiert. (45)

d) Erstellt eine Funktionalität um den Zustand des Zellulären Automaten aus Dateien zu importieren und in Dateien zu exportieren. Das Format der Datei soll sich an der bereitgestellten Datei orientieren. (20)

e) Implementiert eine Steuerung für ein Konsolenprogramm, wie in Aufgabenteil (b), mit welcher der Benutzer sämtliche Funktionalitäten des Zellulären Automaten steuern kann. Der Zustand des Automaten soll nach jeder Aktion visualisiert werden. (15)

[1] Verwendet dazu einen mindestens C++ 11 konformen Zufallszahlengenerator.

Aufgabe 2 Visuelle Kryptographie (Σ 100 $P.$)

a) Erstellt eine Klasse *(NBild)*[2] mit der ihr den Inhalt der vorgegebenen Dateien importieren und exportieren könnt. Es soll möglich sein auf die einzelnen Bildpunkte mittels Indexzugriff lesend und schreibend zu zugreifen (also `operator() const` und `&operator()`). Die Farben des Bildes sollen sich auf die beiden Farben (Weiß / Schwarz) beschränken und geeignet codiert werden. (30)

b) Erstellt eine weitere Klasse *(CBild)*, im Unterschied zu *(NBild)* sollen deren Elemente die Blöcke A und B repräsentieren. Es sollen ebenfalls Methoden zum Importieren und Exportieren von Bildern implementiert werden. (25)

c) Erstellt eine statische Funktion, die ein Bild mit rein zufällig[3] gesetzten Bildpunkten erzeugt. Die Größe des Bildes soll durch Funktionsparameter bestimmt werden. (10)

d) Erstellt eine Funktion mit der man ein Bild mit Hilfe von einem Schlüssel, gemäß der Vorgaben, verschlüsseln kann und eine weitere Funktion, welche die Verschlüsselung umkehrt. (15)

e) Erstellt eine Funktion die zwei Bilder als Eingabe erwartet und als Ausgabe ein Bild erzeugt, so als würde man die beiden Bilder übereinanderlegen. (10)

f) Speichert die Ausgabe die entsteht, wenn man zwei Bilder übereinander legt die mit dem gleichen Schlüssel verschlüsselt wurden. (3)

g) Implementiert ein Konsolenprogramm (`visualcrypt`) welches die Funktionen aus Aufgabenteil (d) bis (e) in Abhängigkeit von Parameter ausgeführt wird die beim Programmaufruf gesetzt werden (7)

```
visualcrypt encode <source> <result> <key>
visualcrypt decode <image_a> <image_b> <result>
visualcrypt overlay <image_a> <image_b> <result>
```

[2] Die Klassennamen dürfen frei gewählt werden

[3] Auch hier soll mindestens ein C++ 11 konformer Zufallszahlengenerator verwendet werden.

Aufgabe 3 Visualisierung mit QT (Σ 100 $P.$)

Erstellt mit dem QT-Creator eine QT-Widget-Anwendung. Diese soll sowohl den Funktionsumfang von Aufgabe 1, als auch den von Aufgabe 2 auf eine Benutzeroberfläche abbilden, so dass ein Benutzer grafisch mit dem Programm interagieren kann.

a) Erstellt die Benutzeroberfläche und verwendet ein TabWidget um zwischen den Aufgabenteilen wechseln zu können. (10)

b) Erstellt ein Widget für Game-of-Life welches die Evolution automatisch nach einer bestimmten Zeitspanne erneut ausführt. Verwendet dazu einen Zeitgeber (QTimer*), für den der Benutzer selbst die Zeitspanne festlegen kann. Visualisiert das Feld von Game of Life (QPainter-Widget) und achtet darauf, dass man die Zellen visuell voneinander abgrenzen kann. Der Zustand von lebenden und toten Zellen soll über einen Klick auf die dementsprechende Zelle verändert werden können. Es soll weiterhin möglich sein die Größe des Feldes zu ändern (QSpinBox) und Dateien zu importieren und exportieren (QFileDialog). Damit der Benutzer genug Zeit hat das Feld zu erstellen soll es erst nach aktivieren eines Steuerelementes gestartet werden (QPushButton) und es sollen Elemente vorhanden sein mit denen der Benutzer die Evolution anzuhalten und das Feld zu bereinigen. (45)

c) Erstellt ein Widget für die Visuelle Kryptographie, bei dem der Benutzer den Modus [Verschlüsseln, Entschlüsseln, Übereinanderlegen] wählen kann (QComboBox). Zum Öffnen und zum Speichern der Dateien sollen wieder Dialoge verwendet werden. Allerdings soll die Ausgabe nur dann gespeichert werden, wenn der Benutzer eine dementsprechende Aktion ausführt. Die geöffneten Dateien und die Ergebnisse der Berechnungen sollen auf geeignete Weise visualisiert werden, dazu können z.B. Animationen oder Farben verwendet werden (45)

Sonderregeln für Gruppen mit Lehramtsbeteiligung

Die folgenden Aufgaben müssen von den Gruppen mit der entsprechenden Zahl an Lehramtsstudenten nicht bearbeitet werden

1 Lehramtsstudent: Aufgabe 1 e

2 Lehramtsstudenten: Aufgabe 1 e und Aufgabe 2 e, 2 f, 2g

3 Lehramtsstudenten: Aufgabe 3 a 3 c

4 Lehramtsstudenten: Aufgabe 2 e, 2f, 2g und Aufgabe 3 a, 3 c

Hinweise:

Die Abgabe muss fristgerecht erfolgen. Eine Bewertung kann nur dann vorgenommen werden, wenn alle Dateien vorhanden sind und sich das Programm kompilieren lässt. Die Quellcode Dateien sind vollständig zu kommentieren und dokumentieren. Bekannte Fehler sollen in einer `readme.txt` dokumentiert werden. Aus dieser Datei soll auch die Gesamtarbeitszeit jeden Gruppenmitgliedes ersichtlich sein, sowie wer an welchen Funktionen, Klassen oder Programmabschnitten gearbeitet hat.