Praktikumsprotokoll

Mikrorechentechnik I

Versuch: Versuch 3: C–Programmierung

Betreuer: M. Herhold

Gruppe: 30 **Datum** der Praktikumsdurchführung**:** 18.01.2022

Gruppenmitglieder:	Tobias Brantz
	Dustin Hanusch
	Husam Mohamed
	Eginhardt Richter

Informationen zur Abgabe des Protokolls:

Abgabetermin: eine Woche nach Praktikumstermin

Abgabeart: als PDF per E-Mail an: mario.herhold@tu-dresden.de

Inhaltsverzeichnis

1. Einleitung

Die allgemeine Aufgabe war die Programmierung eines C Programms, welches eine grafischen Animation, nach in der Aufgabenstellung vorgegebenen Regeln generiert. Als Grundlage dafür diente ein .txt Dokument, welches ein vorgegebenes Muster, die Größe der Darstellungsfläche sowie die Anzahl der Animationsschritte enthielt und eingelesen werden musste. Nach dem Einlesen sollte das eingelesene Muster verarbeitet werden, wofür es notwendig war einen Algorithmus zu erstellen, welcher die Animation berechnet. Mithilfe der Grafikbibliothek SDL2 konnte diese dann ausgegeben werden. Zudem sollten bestimmte Anforderungen an eine Interaktion mit dem Programm mittels User Interface erfüllt werden, wozu ein eigenes Modul erstellt wurde.

2. Organisation im Team

Für eine reibungslose Zusammenarbeit erstellten wir ein Repository auf GitHub, was uns eine parallele Entwicklung aller C-Module ermöglicht hat. Zu diesem Zweck wurden die einzelnen Aufgaben wie folgt verteilt:

- Datenstruktur- und Funktionstypen: Tobias Brantz, Dustin Hanusch
  - C-Module: runtime_data.c, runtime_data.h
- Einlesen der Konfigurationsdatei: Tobias Brantz
  - C-Module: config.c, config.h
- Berechnen des nächsten Animationschrittes: Dustin Hanusch
  - C-Module: engine.c, engine.h
- Grafische Darstellung des Animationsbildes: Husam Mohamed
  - C-Module: gfx.c, gfx.h
- Behandlung von Nutzereingaben: Eginhardt Richter
  - C-Module: ui.c, ui.h

runtime_data

Zur Kommunikation der Programmteile entwickelten wir vor dem eigentlichen Code die Schnittstellen zur Interaktion der einzelnen Module. Die wichtigste davon ist die in der runtime_data.h definierte, welche die eingelesenen und zu verarbeitenden Daten zur Verfügung stellt.

Beispiel 1

```
43  //erstellen einer Liste mit den Element (X,Y,AN_COUNTER,AN_MAX,DELAY,ARRAY)
44  list_header* new_specific_list();
```

Im Beispiel 1 wird die Erzeugung der Liste gezeigt, welche alle wichtigen Laufzeitparameter sowie die Eigenschaften der Darstellungsfläche enthält. Diese wurde wie in der Vorlesung gezeigt in einer dynamischen Liste Organisiert, auf welche mit Hilfe des list_header zugegriffen werden kann. Diesen Zugriff verwalten die Getter- und Setter Funktionen in der Form, wie es in Beispiel 2 dargestellt wird.

Beispiel 2

```
59  //getter setter Animationszähler
60  int* get_animation_counter(list_header* list);
61  void set_animation_counter(list_header* list, int* counter);
```

config

In diesem Modul ist nur die Einbindung der runtime_data.h in den Header notwendig, da lediglich die Liste der runtime_data, auf Basis der zur Verfügung gestellten .txt Datei geändert wird.
Der Name der .txt Datei und die Liste sind zudem die Parameter der einzigen Funktion in diesem Modul (Beispiel 3), welche von anderen Modulen aufgerufen wird.

Beispiel 3

```
8   // einlesen der settings.txt in runtime_data
9   void read_settings(list_header *list,char* path);
```

engine

Auch die engine.h enthält nur eine Funktion (Beispiel 4) die von außerhalb des Moduls gerufen werden muss. Durch die Weitergabe des list_headers kann intern auf die Schnittstellen der runtime_data zugegriffen werden um den nächsten berechneten Schritt abzuspeichern. Zu diesem Zweck ist ebenfalls eine Einbindung der runtime_data.h notwendig.

Beispiel 4

```
8   //berechnet den nächsten Animationschritt ausgehend von dem Array in der Liste
9   void cal_nextAnimaionStep(list_header* list);
```

gfx

In diesem für die Grafikausgabe zuständigen Modul wird lediglich die runtime_data.h in den Header mit eingebunden, welches den Zugriff auf die Laufzeitdaten ermöglicht. Dabei werden wie in Beispiel 5 gezeigt diese mittels list_header übergeben und explizit greift dieses Modul dann auf das Array zu, welches die zu zeichnenden Pixel enthält.

Beispiel 5

```
8   //neuzeichnen des Buffers
9   void print_animation_buffer(list_header* list);
10  //Initialisierung des Fensters
11  void init_frame(list_header* list);
```

ui

Die Schnittstelle der ui ist sehr einfach, da in dieser nur die Liste übergeben werden muss. Intern wird der Parameter zur Weitergabe des Headers an andere Funktionen benötigt, um deren Funktionalität zu gewährleisten.

Beispiel 6

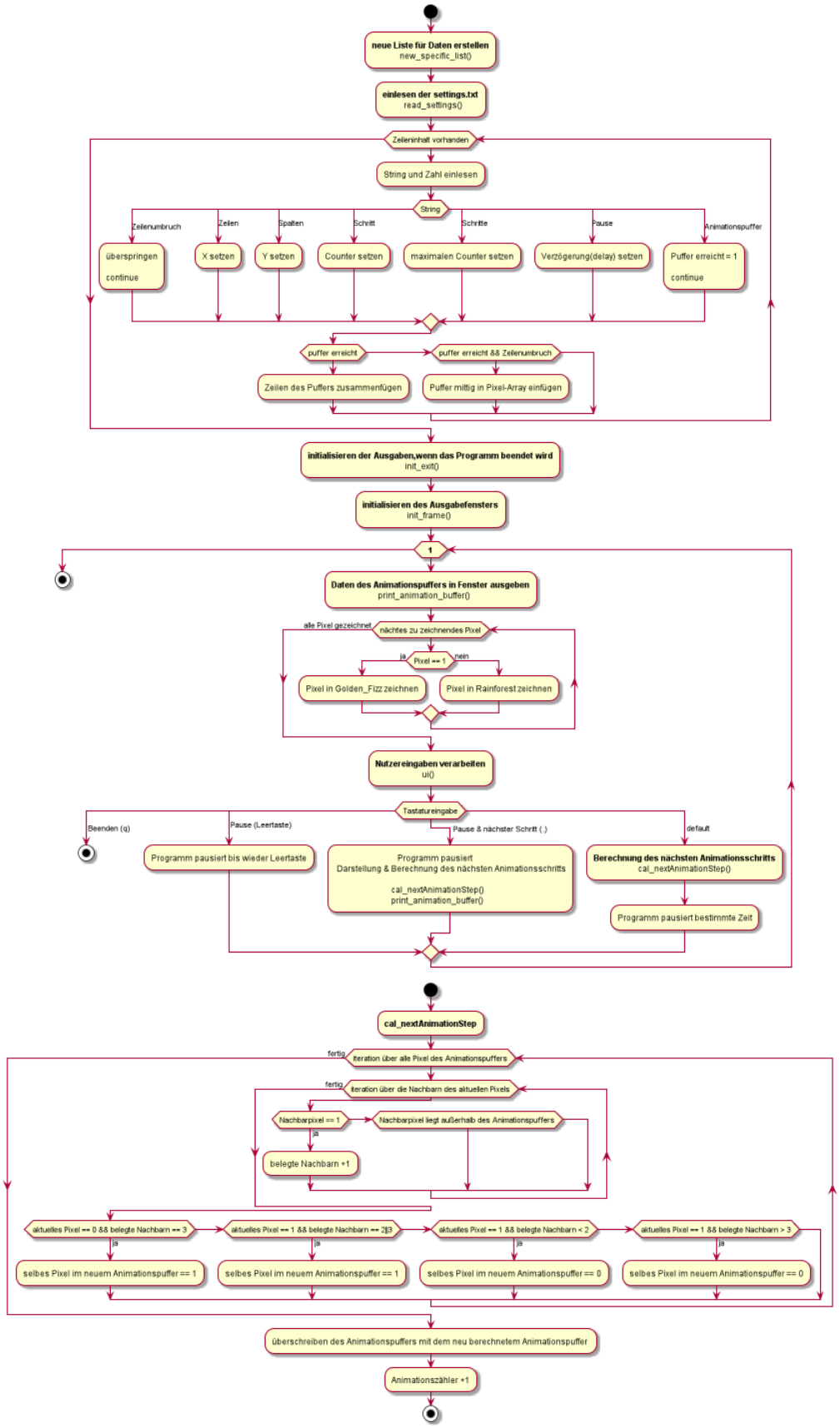
```
4   // UI Schnittstelle
5   void ui(list_header* list);
```

Die Verknüpfung der einzelnen C-Module zu einem Programm wird zum Großteil in der main.c realisiert, wobei die ui.c auch wichtige Verknüpfungen von Programmteilen vornimmt. Diese Module sorgen für das richtige Aufrufen verschiedener Funktionen zum richtigen Zeitpunkt, in Abhängigkeit von den Benutzereingaben. In der while Schleife in Beispiel 7 sieht man das nur zwei Funktionen ständig gerufen werden müssen um den korrekten Ablauf des Programms zu gewährleisten. Da diese Schleife keine Abbruchbedingung besitzt, ist es möglich das, dass Programm nur durch eine User-Interaktion aus der ui.c heraus beendet werden kann.

Beispiel 7

```
20  int main(void)
21  {
22      list_header *list = new_specific_list();
23      read_settings(list, "./settings-1.txt");
24
25      init_exit();
26      init_frame(list);
27
28      while(1){
29
30          print_animation_buffer(list);
31          ui(list);
32
33      }
34      return 0;
35  }
```

3. Aktivitätsdiagramm



source code: https://github.com/duha887b/MRT1_V3_Animation/blob/master/Protokoll/Aktivit%C3%A4tsdiagramm.puml

4. Besonderheit des Codes

Eine Besonderheit ist, dass das Array, welches in der runtime_data abgespeichert wird und die abzubildenden Pixel enthält ein 1D Array ist. Dies bedeutet, dass die Zeilen der gesamten 2D-Darstellungsfläche direkt hintereinander angehängt sind. Aufgrund dessen können über das zeilenweise Auslesen die Pixel dann einfach wieder ihrer 2D Position zugewiesen werden.

1D-Einlesen

```
for (i=0;i<puffer_rows;i++) {                          //Iteration über Reihen des Puffers
    index=index+puffer_cols+2*space_l_r+transfer;           //Sprung um 2*Platz-links/rechts + Spalten des Puffers
    for (j=0;j<puffer_cols;j++) {                      //Iteration über Spalten des Puffers
        field_content= field_content_conversion(puffer[i*puffer_cols+j]);
        array[index+j+1]=field_content;
    }
}
```

Der im Code erwähnte Puffer enthält die aneinander gehängten Zeichen des vorgegebenen Musters. Mit Hilfe der bekannten Zeilen- und Spaltenanzahl, sowohl des Puffers als auch der Darstellungsfläche kann dann einfach über diese iteriert werden.
Jede Schleife “i” sorgt dabei für das Weiterspringen in den Zeilen und die Schleife “j” für das Iterieren über die jeweiligen Zeilen des Puffers.

5. Probleme während der Entwicklung des Programms

Die meisten Fehler, die aufgetreten sind, waren auf Unachtsamkeit während der Programmierung zurückzuführen. Diese konnten häufig durch einen einzigen Durchlauf des Programms mittels Debugger behoben werden. Das Einzige was dieses Projekt für uns schwieriger gemacht hat, war die vergleichsweise komplizierte Speicherverwaltung in C. Nach gründlichem Auseinandersetzen mit der Funktionsweise dieser, konnten alle vorher gemachten Fehler mit der Speicherallokation im Weiteren problemlos behoben werden.