Programmdokumentation MRT Praktikum 3: Animation

Gliederung:

1. Einführung
2. Projektablauf (zeitlicher Ablauf, Aufgabenverteilung)
3. Vorstellung des Programmes
4. Probleme / Schwierigkeiten bei der Bearbeitung der Aufgabenstellung
5. Einführung

(kurz Aufgabenstellung vorstellen)

1. Projektablauf:

Die Praktikumsaufgabe wurde um den Jahreswechsel 22/23 bearbeitet. Zunächst wurden am 22.12.2022 grundlegende Punkte zur Bearbeitung der Aufgabe besprochen.

Zum einen wurde die Datenstruktur „laufzeitdaten“ festgelegt und die Interpretation des Arrays für den Puffer besprochen. Weiter wurde der grundsätzliche Ablauf des Programmes diskutiert und festgelegt.

Abschließend wurde die Praktikumsaufgabe in kleinere Arbeitspakete unterteilt:

* Einlesen der Datei: Kleon Dingeldein
* Berechnung des nächsten Puffers: Martin Blümel
* Ausgabe des Bildes: Theo Röhl
* Hauptprogramm und Benutzereingabe: Justus Krenkel

In den folgenden Tage wurden die einzelnen Arbeitspakete von den Teammitglieder bearbeitet. Um Probleme beim Programmieren gemeinsam zu besprechen und so schneller Lösungen finden zu können, fand ein regelmäßiger Austausch im Team statt.

Am 02.01.2023 war eine letzte Besprechung geplant bei die einzelnen Programme zusammengefügt wurden und ein finaler Funktionstest durchgeführt wurde.

1. Vorstellung des Programmes:

3.1. Main und Benutzereingabe

3.2. Datenstruktur „laufzeitdaten“

3.3. Einlesen der Datei

Das Einlesen einer Konfigurationsdatei ist im C-Modul config.c implementiert. Über den Funktionsaufruf *x=einlesen(“filename.txt“)* wird die Variable x des Types „laufzeitdaten“ mit den eingelesen Informationen aus der filename.txt befüllt.

Nach Funktionsaufruf springt das Programm über die Headerdatei *config.h* in das gleichnamige C-Module. Dort ist der grundlegende Programmablauf in der Funktion *struct laufzeidaten einlesen (const char\* filename)* programmiert:

1. Öffnen der Datei (ausschließlich zum Einlesen von Informationen) mit evtl. Fehleranzeige falls dies misslingt.
2. Einlesen der Informationen zu Spalten, Zeilen, Schritt, Gesamtschritte, Pausen zwischen den Animationsschritten in der Funktion *nummerneinlesen(input\_file):*

Diese Funktion bekommt den Pointer auf das geöffnete Dokument übergeben. Der Curser befindet sich an erster Stelle. In einer while-Schleife wird ein String eingelesen und dieser mit den Schlüsselbegriffen („Zeilen:“, „Spalten:“, usw.) verglichen. Anschließend wird eine interne Variable des Types „laufzeitdaten“ mit der nun eingelesenen Zahl beschrieben. Sobald die Funktion erkennt, dass als nächstes der Animations-Puffer eingelesen werden soll, wird die Schleife abgebrochen.

* Einlesen des Puffers in der Funktion *puffereinlesen(data.X, data.Y, input\_file):*

Diese Funktion ließt den Puffer der Datei ein und hinterlegt ihn in einem Zwischenspeicher. Dafür bekommt sie die maximalen Maße des Bildschirmes übergeben und den Pointer auf die Einlesedatei. Dessen Curser befindet sich zu Funktionsbeginn vor dem ersten Zeichen des Puffers.

Der Zwischenspeicher ist von der Datenstruktur *eingelesenerpuffer* – einer Struktur die *laufzeitdaten* ähnlich ist, jedoch nur die für diesen Schritt benötigte Zeilen- und Spaltenlänge sowie ein Array für den eingelesen Puffer besitzt. Da nur *config.c* diese Datenstruktur verwendet, ist sie in *config.h* deklariert.

Zunächst wird in das dynamisch erzeugte Array *einlesespeicher* die erste Zeile eingelesen und dessen Länge ermittelt.

Anschließend wird Speicherplatz für den Zwischenspeicher allokiert. Idee dieser etwas umständlicheren Programmierung ist, dass der Puffer deutlich kleiner als der gegebene Bildschirm sein kann. Dadurch wird weniger Speicher zum Einlesen benötigt.

Anschließend wird der Zwischenspeicher mit den vom Puffer eingelesenen Zeichen befüllt und der Speicher des *einlesenspeicher* deallokiert.

Die Funktion gibt den eingelesenen Puffer sowie dessen Abmaße zurück.

* Schließen der Einlesedatei.
* Meldung falls der eingelesen Puffer nicht zentrierbar ist: Aus den Maßen des eingelesen Puffers und den Maßen des Bildschirmes wird nun geprüft, ob das Bild mittig eingegeben werden kann. Es wird eine Meldung ausgegeben, falls dies nicht möglich ist.
* Zentrieren des Puffers in der Funktion *zentrieren(zwischenspeicher, data.X, data.Y):*

Diese Funktion setzt den eingelesen Puffer mittig in den eingelesen Bildschirm. Dafür wird für jedes Pixel des Bildschirmes geprüft, ob es von einem Pixel des eingelesen Puffers beschrieben werden muss.

Rückgegeben wird ein Array das den Bildschirm mit dem zentrierten eingelesen Puffer beschreibt. Das Array berücksichtig außerdem den zur weiteren Berechnung benötigten Rand.

* Rückgabe der befüllten Datenstruktur an die Main-Schleife.

3.4. Berechnung des nächsten Animationsschrittes

Das Berechnen des nächsten Animationsschrittes ist im Modul engine.c implementiert. Durch den Funktionsaufruf calculate\_next\_pic(daten), innerhalb der Mainschleife im Modul main.c, wird über die dem Engine-Modul gleichnamige Header-Datei engine.h, aufgerufen.  
Die Funktion calculate\_next\_pic realisiert eine pixelweise Neuberechnung des übergebenen Animationspuffers, das Hochzählen des Animationsschrittes und die Rückgabe eines Structs vom Typ Laufzeitdaten mit den aktualisierten Parametern.

* Funktionsaufruf im Modul main.c



* Implementierte Funktion im Modul engine.c



* Pixelweise Neuberechnung des Animationspuffers: diese ist erfolgt nach 4 vorgegebenen Regeln:  
  1. Ein freier Pixel mit exakt 3 belegten Nachbarpixeln wird zu einem belegten Pixel.  
  2. ein belegter Pixel mit weniger als zwei belegten Nachbarpixeln wird frei  
  3. ein belegter Pixel mit zwei oder drei belegten Nachbarpixeln bleibt belegt  
  4. ein belegter Pixel mit mehr als drei belegten Nachbarpixeln wird frei  
  Ebenfalls war es gewünscht das auch die Randpixel die zu späterem Zeitpunkt nicht sichtbar sind neu zu berechnen.  
  Um diese Regeln um zusetzen wird in einem weiteren Laufzeitdatenstruct „betdata“ ein um einen Randpixel größerer Animationspuffer allokiert, mit leeren Pixeln befühlt und anschließend mittig durch den übergeben Animationspuffer ergänzt.   
    
    
    
  Damit ist es möglich auch die acht umliegenden Pixel der Randpixel zu betrachten. Über 2 For-Schleifen wird dann der Animationspuffer durchlaufen.  
    
    
    
  Dabei werden die umliegenden und der zu betrachtende Pixel über if-Anweisungen auf Freiheit geprüft. Umliegend belegte Pixel werden über einen Zähler „k“ mitgezählt.  
    
    
  Beispiel:  
  Ein Bild, das Text enthält.

  Automatisch generierte Beschreibung  
    
    
    
  Je nach Regel werden die geänderten Pixel oder gleich gebliebenen Pixel auf den Animationspuffer des Rückgabestructs „nextdata“ geschrieben.  
    
  Beispiel:  
  Ein Bild, das Text enthält.

  Automatisch generierte Beschreibung
* Hochzählen des Animationschrittes  
    
  Ein Bild, das Text enthält.

  Automatisch generierte Beschreibung
* Rückgabe von nextdata über return

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

3.5. Ausgabe des Bildes

4. Probleme / Schwierigkeiten bei der Bearbeitung der Aufgabenstellung

* VM, Git einrichten
* (Übergang von VS zu VM)
* Umgang mit Pointer (bei strings, Dateien,…)
* .c dateien nicht in .h dateien…
* Allokieren und befreien von Speicher