Projektdokumentation Card-Restorer

Aufgabe 1 des Praktikums des Modules Informatik 3 SoSe 24

Gruppe B

Hochschule Reutlingen Fakultät Informatik
Studiengang: Medien-Kommunikations-Informatik

Martin Lauterbach (810087)

Amina Anzorova (752283)

Sissi Wu (808465)

Abstract:
Das Projekt ist eine Lösung zum Problem, eine Reihe von korrupten in Textformat vorliegenden Karten durch eine Referenzdatei zu rekonstruieren. Die Lösung besteht aus dem Einlesen beider Kartensätze, der Möglichkeit des Benutzers, in den Programmablauf Einfluss zu nehmen, das Iterieren über die Referenzen je möglicherweise korrupte Karte, und einem entweder automatischen oder manuellem Auswählen der bestmöglichsten Karte. Die "Bestmöglichste"wird definiert durch ein Levenshtein-Algorithm mit optionaler Gewichtung sowie Transposition, einem optionalem custom Similarity-Algorithmus, sowie einer definierbaren Range, in der eine Karte im Abstand zum geringsten gefundenen Wert nicht gelöscht wird.

Link zur Hochschuloffenen Repository aller Praktikumsaufgaben:
`https://gitlab.reutlingen-university.de/inf3-theteam/aufgaben-repo-inf3-sose14`

Stand: 15. Mai 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Strukturelle Erklärung des Projekts

Das Projekt wird ausgeführt in der Main-Methode der Datei card_restorer.cpp. Zu Beginn werden die hardcodeten relativen Projekt-Dateiwege [1] zu den der Aufgabe vorliegenden .txt-Dateien genutzt, um Zeile für Zeile durch die Methode create_card_list zwei Vector-listen an Card-structs zu erstellen, die die einzelnen spezifischen Unterinformationen pro Karte enthalten.

Daraufhin wird der User gefragt, ob bestimmte Variablen, zusätzliche Algorithmen, Abläufe, Debug-Modi oder Selbstentscheidungen aktiviert werden sollen. Dies geschieht über eine Reihe von user_decide-Methoden, die je einen anderen Dateityp anfragen können.

Nachdem die grundlegenden Argumente des Programms geklärt wurden und in Variablen gesetzt, iteriert das Programm mit jedem der korrupten Karten über alle Referenzkarten, berechnet je LS und (opt) SIM, und schaut bei jeder Findung eines neuen Tiefstwertes danach, alle außerhalb der Range liegenden Karte aus dem Vektor zu löschen.

Die struct *CloseCard* beinhaltet einen double Wert für die LS (da die Gewichtungen double Werte haben darf), einen double Wert für die SIM Berechnung, und dem LS*SIM double Wert, nach welchem die Karten vor Entscheidung sortiert werden.

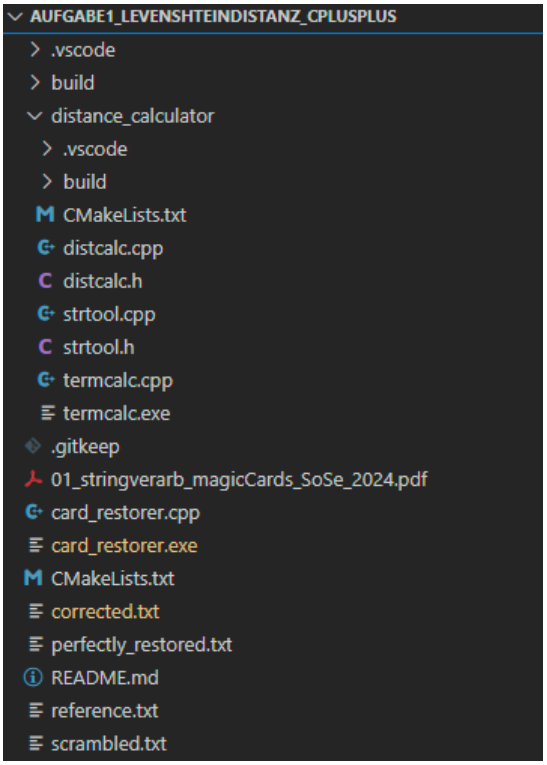

Abbildung 1: Projektstruktur, Screenshot aus Visual Studio Code; Lauterbach; 09.05.24

Nach Ablauf der Iterationen über die Cards mit Abbruchbedingungen , sowie dem darauffolgendem Auswahlverfahren, kann die resultierte Anzahl an Cards mit einer vorherigen Card-Restaurierung prozentual verglichen werden, oder gleich in eine restored.txt-Datei geschrieben werden.

[1] Bitte beachten: Je nach IDE und build-prozess, kann die .exe-Datei an anderen nicht vorhergesehene Stellen erstellt werden. Dies führt zu einem Fehlerauswurf bei Einlesen der Dateien, da diese nicht gefunden werden können.

2 UML-Diagram

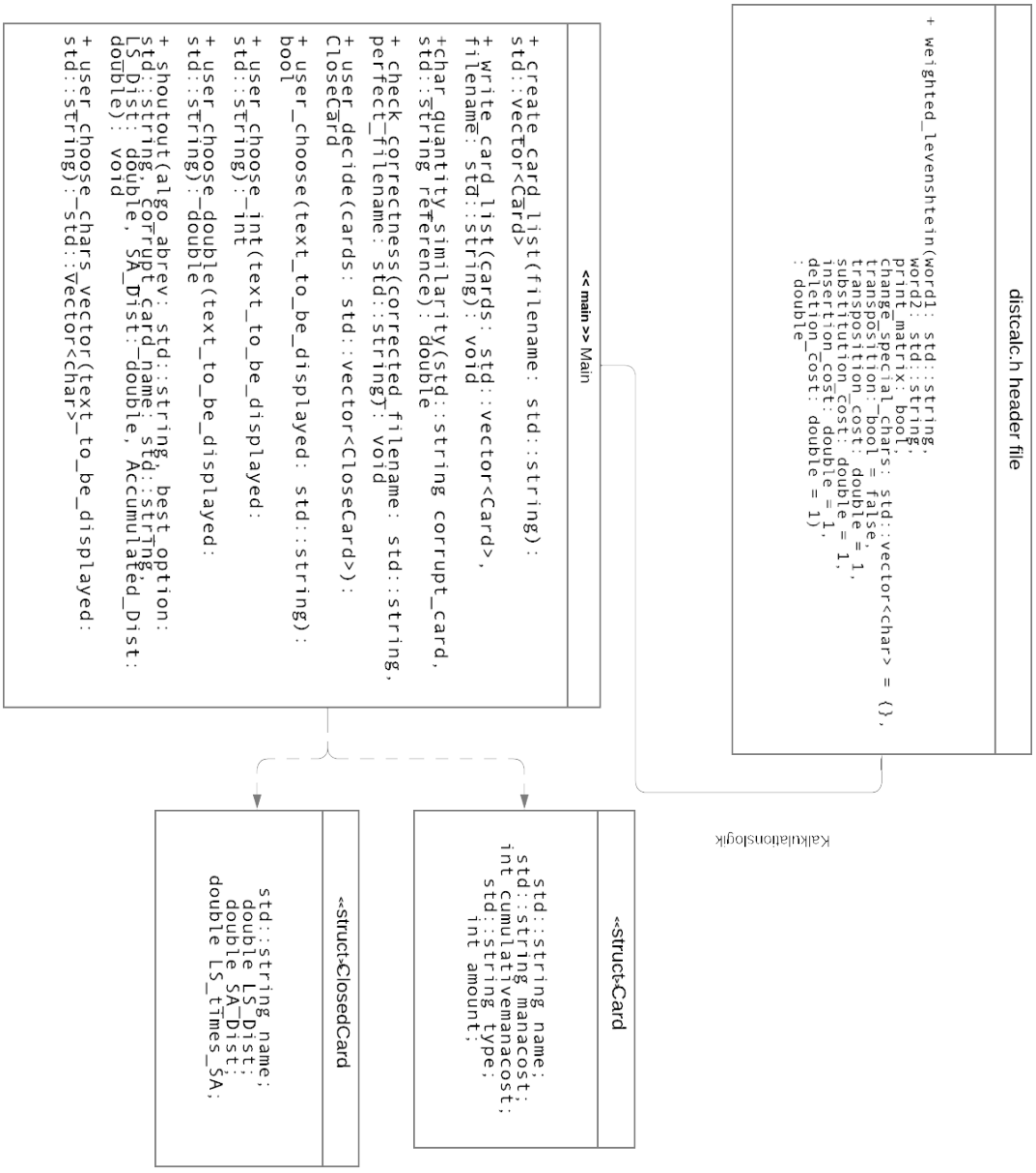

Abbildung 2: UML-Diagram des Programs aus card_restorer.cpp; Anzorova, edited by Lauterbach; 12.05.24

3 Benutzereinstellungen

change_weights: Der User kann die Standard-Werte von int::"1für die folgenden Operation im Levenshtein-Algorithmus verändern: 1. Insertion, 2. Deletion, 3. Substitution, (opt) 4. Transposition

use_unkwnown_chars: Der Benutzer kann festlegen, ob bestimmte Zeichen als sogenannte Wildcard-Charakter für den LS-Algorithmus angesehen werden sollen. Diese werden bei der Berechnung als eine Art Joker angesehen, der jeglichen Char darstellen könnte, und somit keine Erhöhung des LS-Wertes verursacht.

user_decision_LS: Der Benutzer kann selbst entscheiden, ob er jede restaurierte Karte aus einem Pool von maximal 50 nach Endwert sortierten Karten auswählen möchte, oder ob diese Entscheidung automatisch auf die Karte mit dem geringsten Wert fallen sollte.

range_LS: Der Benutzer kann festlegen, in welcher Distanz (double), eine Karte entfernt von der geringsten gefundenen LS/LS_SIM-Distanz sein darf, bis diese gelöscht und nicht in der Entscheidungsfindung teilnimmt.

use_similiarity_algorithm: entscheidet, ob für jede Karte auch zusätzlich der custom Algorithm zur möglichen Verfeinerung der Ergebnisse genutzt werden soll. Der vom Algorithmus errechneter Wert wird, da er zwischen 0 (gleich) bis 1 (keine Änhlichkeiten) reicht, mit dem Wert des Levenshtein-Algorithm multipliziert für die Endergebnisse genutzt.

debug_mode: wenn aktiviert, gibt zusätzliche Ausgaben über aktuelle Zustände aus.

4 Projektmetriken

Tabelle 1: Code-Metriken pro File und hervorgehobenen Methods; Lauterbach, Anzorova; 15.05.2024

Entity	Lines of Code	Lines of Comments
Projektdateien:		
card_restorer.cpp	332	176
distcalc.cpp	33	16
distcalc.h	9	1
Debug-Test-Program:		
strtool.cpp	16	12
strtool.h	6	1
termcalc.cpp	39	12
Wichtige Project-Methods:		
main-method	146	70
weighted_levenshtein	25	10
char_quantity_similarity	28	15
create_card_list	25	7
write_card_list	16	5

5 Antworten zu Aufgabenstellungen

1. Implementiert den Algorithmus zur Berechnung der Levenshtein-Distanz (LD) (5P).

-> weighted_levenshtein in distcalc.cpp

2. Implementiert eine Datenstruktur Card, welche die Attribute name, mana, cmc, type und count beinhaltet (5P). Beruecksichtigt dabei die Prinzipien der Objektorientierung.

-> struct Card in card_restorer.cpp Line 27

3. Implementiert eine Methode, um aus scrambled.txt eine Liste von Cards zu generieren. Haltet euch dabei an das Format, das im Einleitungstext beschrieben ist (3P).

-> std::vector<Card> create_card_list(std::string filename); in card_restorer.cpp Line 375

4. Implementiert eine Methode, um Cards wieder im oben angegebenen For- mat in eine Datei zu speichern (3P).

-> void write_card_list(std::vector<Card> cards, std::string filename); in card_restorer.cpp Line 412

5. Implementiert eine Methode, um die Namen der Karten aus der Referen- zliste zu lesen. In jeder Zeile steht genau ein Name (2P).

-> wird durch das Einlesen beider .txt-Dateien in Card-struct-Vectoren gemacht, die wiederrum dann durch ihren std::string name; Bauteil später iteriert werden.

6. Implementiert eine Methode, welche den Namen einer Card c wiederher- stellt. Berechnet dazu c so lange die LD aus deren Namen und den Namen aus der Referenzliste, bis die LD kleiner als 26,75% der Laenge des Kartennamens ist. Findet ihr so einen Eintrag, ersetzt ihr den Namen von c durch den Referenzwert (5P).

-> Teil der main-methode in card_restorer.cpp zwischen Line 208 bis 277.
-> nested loops, welche CloseCard-structs pro Karte mit berechneten Werten erstellt

7. Schreibt die reparierten Karten wieder in eine Datei (2P).

-> void write_card_list(std::vector<Card> cards, std::string filename); in card_restorer.cpp Line 412