Entwicklerdokumentation von Quiz App

Zsolt Gaál

1. Dezember 2024

Inhaltsverzeichnis

1 Einführung

In jedem Fall enthalten die Header-Dateien die Gebrauchsanweisungen für die einzelnen Funktionen, d.h. eine detaillierte Dokumentation über die Funktionsweise, Parameter und Rückgabewerte.

2 Dateistruktur

2.1 `main.c`

Enthält die `main` Funktion, die das Programm startet und alle Hauptprozesse aufruft:

```c
int main(int argc, char *argv[]) {
    srand(time(NULL));  // random seed
    clear_screen();     // Konsole leeren
    init_quiz();        // Speicherplatz allokieren

    read_all_input(argc, argv); // Einlesen
    shrink_quiz_size();         // Speicherplatzminimierung
    play_quiz();                // Spiel
    evaluate_quiz();            // Auswertung

    free_quiz();        // Speicherplatz freigeben
    return 0;
}
```

2.2 `comm.c`

Dient der Vereinfachung der Kommunikation mit dem Benutzer. Anstelle von `printf` wird eine eigene Nachrichtenanzeigefunktion mit verschiedenen Nachrichtentypen implementiert. Enthält die Menüanzeige, den Spielmoduswähler und den Tastenanschlagsbeobachter.

2.3 `fs_utils.c`

Enthält kürzere Hilfsfunktionen: Zeichenkettenmanipulationen, Anforderungsüberprüfungen.

2.4 `fs_read.c`

Verantwortlich für das Einlesen der Eingabedateien. Navigiert im Dateisystem, benachrichtigt den Benutzer bei Fehlern und fordert gegebenenfalls eine Intervention an.
Einige globale Konstanten:

```c
const char * const input_root = "input";
const char * const allowed_extensions[] = {".txt", ".tsv"};
const int allowed_extensions_size = sizeof(allowed_extensions) / sizeof(
    allowed_extensions[0]);
```

Das Haupteinlesen:

```c
void read_all_input(int argc, char *argv[]) {
    int i;

    // wenn kein Eingabeverzeichnis vorhanden ist, erstellen wir eines
    if (no_input_root()) {
        print_message(ERROR, "Input directory not found.");
        print_message(QUESTION, "Would you like to create the input directory? (y/n)
            ");
```

```
1
2           if (getchar_equals('y') && create_input_root()) {
3               print_message(INFO, "Input directory created.");
4               print_message(INFO, "Place your input files in the input directory, then
                    press any key to continue.");
5               print_message(QUESTION, "If there is no ANY key on your keyboard, consult
                     your local keyboard vendor...");
6               getchar_equals(0);
7           }
8           else {
9               print_message(FATAL, "Could not create input directory.");
10          }
11          print_message(INFO, "");
12      }
13
14      // wenn keine Parameter angegeben sind, lesen wir den gesamten Input-Ordner ein
15      if (argc == 1) {
16          print_message(WARNING, "No input parameters specified in the command line.");
17          print_message(QUESTION, "Would you like to read all files in the input
                directory? (y/n) ");
18          if (getchar_equals('y'))
19              try_read_input(path_join(input_root, NULL));
20      }
21      else // ohne Fehlerbehandlung wäre es nur so :)
22          for (i = 1; i < argc; i++)
23              // da wir relativ zum Eingabeverzeichnis suchen, fügen wir den gesuchten
                    Pfad mit dem Eingabeverzeichnis zusammen
24              try_read_input(path_join(input_root, argv[i]));
25
26      return;
27  }
```

So versuchen wir, vom angegebenen Pfad zu lesen:

```
1           void try_read_input(char *input_path) {
2               if (!inside_input_root(input_path)) {   // es wurde gesagt, dass wir aus
                    dem input-Verzeichnis lesen
3                   print_message(ERROR, "Input '%s' is not within the input directory.",
                         input_path);
4               }
5               // zuerst versuchen wir, die Eingabe als Datei zu lesen, und falls das
                    nicht klappt, als Ordner
6               else if (!try_read_file(input_path) && !try_read_folder(input_path)) {
7                   print_message(ERROR, "Could not read input path: '%s'", input_path);
8               }
9
10              free(input_path);   // Speicherplatz freigeben
11              return;
12          }
```

2.5 `quiz.c`

Verwalten der Quiz-Datenbank und Implementierung der Quiz-Logik. Hier erfolgt die Speicherzuweisung und freigabe, das Hinzufügen neuer Fragen und die Steuerung der Hauptspielschleife.

Die Quiz-Datenbank:

```c
typedef struct Quiz {
    QAPair **qas;    // die Liste der Frage-Antwort-Paare
    int size;        // die Anzahl der gespeicherten Daten
    int capacity;    // die Größe des für die Liste reservierten Speichers
} Quiz;
```

Der Aufbau der einzelnen Daten ist äußerst einfach:

```c
typedef struct QAPair {
    char *question; // die Frage im Speicher
    char *answer;   // die Antwort
} QAPair;
```

Das Hinzufügen neuer Daten zur Datenbank erfolgt wie folgt:

```c
int extend_quiz(QAPair *qa) {
    int success = 0;    // ob es erfolgreich war

    // es ist schwierig, nach nichts zu fragen
    if (qa != NULL) {
        // wenn wir mehr Platz brauchen
        if (quiz->size == quiz->capacity) {
            quiz->capacity *= 2;

            quiz->qas = realloc(quiz->qas, quiz->capacity * sizeof(QAPair*));
            if(quiz->qas == NULL)
                print_message(FATAL, "Memory allocation failed.");
        }

        // wir haben schon genug Platz
        quiz->qas[quiz->size++] = qa;
        success = 1;
    }

    return success;
}
```

Die Hauptspielschleife:

```c
void play_quiz() {
    int aktuelle;    // der Index der zufällig ausgew  ählten Frage
    int exit = 0;   // ob eine Beendigungsanforderung eingegangen ist
    int range = quiz->size;
    if (range == 0)
        print_message(FATAL, "Could not read any data from the specified inputs.");

    gamemode_select();  // Auswahl des Spielmodus
    start_timer();      // Uhr startet

    // solange wir noch Fragen haben oder keine Beendigungsanforderung eingegangen
       ist
    while (range && !exit) {
        QAPair *qa = random_question(range, &current);  // wir wählen eine Frage aus
        exit = ask_and_correct_question(qa);            // Frage stellen und Antwort
           korrigieren
        swap_qas(current, --range);         // wir tauschen mit der letzte Element
```

```
        // Im Endloserlebnis, wenn keine Fragen mehr übrig sind, werden alle Fragen
            wieder in den Pool zurückgelegt
        if ((gamemode == INFINITE || gamemode == INFINITE_REVERSED) && range == 0)
            range = quiz->size;
    }

    stop_timer();       // Uhr anhalten

    return;
}
```

2.6 evaluation.c

Verantwortlich für die Nachverfolgung und Bewertung der Benutzerleistung.
Hier wird die Frage gestellt und die Antwort korrigiert:

```
int ask_and_correct_question(QAPair *qa) {
    char user_answer[ANSWER_SIZE];  // hier wird die Antwort eingelesen
    char* question;         // das werden wir fragen
    char* correct_answer;   // die richtige Antwort
    int exit = 0;           // ob eine Beendigungsanforderung eingegangen ist

    // Im umgekehrten Spielmodus vertauschen wir die Frage und die Antwort
    if (gamemode == ONEROUNDER_REVERSED || gamemode == INFINITE_REVERSED) {
        question = qa->answer;
        correct_answer = qa->question;
    }
    else {
        question = qa->question;
        correct_answer = qa->answer;
    }

    // für die Ergebnisanzeige erforderlich (global)
    asked_questions++;

    // wir stellen die Frage und lesen die Antwort ein
    print_message(INFO, "%s ", question);
    fgets(user_answer, sizeof(user_answer), stdin);

    // wir entfernen das Enter-Zeichen am Ende
    user_answer[strcspn(user_answer, "\n")] = '\0';

    // wir erfassen die Beendigungsanforderung
    if (strcmp(user_answer, "!exit") == 0)
        exit = 1;
    // ob die Antwort richtig war
    else if (strcmp(user_answer, correct_answer) == 0)
        correct_answers++;
    // wenn nicht, hätte dies geschrieben werden sollen
    else
        print_message(INCORRECT, "%s", correct_answer);

    // ästhetischer Zeilenumbruch
    print_message(INFO, "");

    return exit;
}
```