Entwicklung eines intelligenten 10-Finger-Schreibtrainers unter C++

Diplomarbeit im Studiengang Technische Informatik

Technische Fachhochschule Berlin

Fachbereich VI - Informatik

Vorgelegt von Tom Thielicke, s712715

im Sommersemester 2006

Betreuende Lehrkraft: Frau Prof. Dr. rer. nat. Ripphausen-Lipa

Zweitgutachterin: Frau Prof. Dr. Schiele

Eingereicht am 05.07.2006

Inhaltsverzeichnis

1. Einleitung

1.1 Vorwort

In unserer heutigen Gesellschaft stellt der Computer ein wichtiges Medium dar. Nicht nur im beruflichen, sondern auch im privaten Alltag ist der Computer zur Datenverarbeitung, Informationsbeschaffung und als Kommunikationsmittel nicht mehr wegzudenken.

Um einen Computer effektiv als Werkzeug zu nutzen, spielt die Bedienung der Tastatur eine große Rolle. Auch wenn sich heute dank grafischer Oberflächen ein Großteil der Funktionen bequem per Maus ausführen lässt, kommt man spätestens bei der Eingabe von Text um die Verwendung einer Tastatur nicht herum.

Eine Tastatur mit dem Zehnfingersystem zu bedienen, bedeutet, deutlich schneller und effizienter arbeiten zu können, als Texteingaben mit weniger als zehn Fingern und wechselndem Blick zwischen Tastatur und Bildschirm zu vollziehen (umgangssprachlich „Zwei-Finger-Suchsystem“). Ein erfolgreich angewandtes Zehnfingersystem verspricht weitaus höhere Anschlagzahlen und eine geringere Fehlerquote. Auch lassen sich Tippfehler durch den festen Blick auf den Bildschirm schneller erkennen und ausbessern.

Es ist zwar voraussehbar, dass die einfache Texteingabe in ferner Zukunft von Techniken wie der Spracherkennung abgelöst wird; bis diese sich aber wirklich auf dem Markt etabliert haben und vor allem auch für Spezialbereiche wie der Programmierung erfolgreich einsetzbar sind, wird das Zehnfingersystem weiterhin ein anerkanntes und verbreitetes Texteingabesystem bleiben.

Um das Zehnfingersystem zu erlernen, gibt es heute zahlreiche Möglichkeiten. Die früher zur Zeit der Schreibmaschine und der ersten Computer angewandten Methoden, das Tippen nach einem mündlichen Diktat oder das sture Abtippen aus entsprechender Literatur (siehe Abb. 1.1), übernehmen heute Programme, die den Benutzer durch verschiedene Lektionen führen und beim Tippen der Texte mit visuellen Hilfestellungen unterstützen.

Abb. 1.1: Literatur zum Erlernen des Zehnfingersystems [Lang 40] / [Moser 91]

Beispiele für auf dem Markt vorhandene Software zum Erlernen des Zehnfingersystems finden sich in nachfolgendem Auszug:

- 10-Finger Assistent
  `http://www.rls.tue.schule-bw.de/lernprogramme/typodrome/`
- Duden Tipptrainer 1.0
  `http://www.duden.de/tipptrainer`
- Oekosoft Zehnfingersystem Schreibtrainer
  `http://www.oekosoft.ch/10f/`
- Der Schreibtrainer
  `http://www.neuber.com/schreibtrainer`
- Tippkönigin
  `http://www.giletech.de/maschinenschreiben.htm`
- Typewriter
  `http://www.augundohr.at/arbeit/duk/ms2/10.php`

Eine Schwachstelle der hier vorgestellten Schreibtrainer liegt im Aufbau der Texte, die vom Benutzer getippt werden sollen. Sie sind vollständig statisch implementiert und ermöglichen einen Lernerfolg lediglich durch einen allgemein festgelegten Ablauf von Buchstaben-, Wort- und Satzfolgen, meist untergliedert in Lektionen. Der Benutzer wird feststellen, dass bei Wiederholung einer Lektion exakt die gleichen Texte diktiert werden wie beim ersten Durchlauf. Die Wiederholung von Lektionen zum Erzielen eines bestimmten Lernerfolgs erscheint dem Benutzer schnell langweilig. Zudem werden Tippfehler, die an einer bestimmten Stelle gemacht wurden, eventuell an gleicher Stelle wiederholt statt verbessert, da ein zu langer Zeitraum zwischen der Wiederholung entstanden ist.

Gegenstand dieser Diplomarbeit soll es sein, einen 10-Finger-Schreibtrainer zu entwickeln, dessen Diktate auf die Fehler des Benutzers reagieren und individuell angepasste Texte präsentieren. Auf diese Weise kann ein schneller und langfristiger Lernerfolg erzielt werden, der zusätzlich durch Abwechslungsreichtum gefördert wird. Individuelle Statistiken und sinnvoll aufgebaute Lektionen sollen zusätzlich die Motivation des Anwenders steigern.

1.1 Geschichtlicher Hintergrund

Die Qwerty-Welt ist nicht die beste aller Welten, aber sie ist wohl auch nicht merklich schlimmer als andere. [WV Zeit]

Das uns heute bekannte Tastaturlayout wurde im Jahre 1868 von Christopher Latham Sholes für die Schreibmaschine entwickelt und wird das QWERTY-System genannt. Den Namen trägt es aufgrund der ersten sechs Tasten links oben auf der englischen Tastatur. Das deutsche System wird aufgrund einer etwas anderen Tastaturbelegung QWERTZ genannt. Die Anordnung der Tasten wurde nicht, wie

man es vermuten sollte, aufgrund rein ergonomischer Gesichtspunkte festgelegt. Vor allem mechanische Aspekte spielten eine Rolle. So wollte man verhindern, dass sich die Typenhebel (Hämmerchen) der Schreibmaschine verhaken und ordnete die zuvor noch alphabetisch sortierten Tasten neu an [WV Zeit] [WV Wikiquertz].

Mit der Einführung von Computern wurde die Tastatur dann zusätzlich um weitere Tasten wie die Funktions- und Navigationstasten sowie um einen abgetrennten Bereich für numerische Tasten erweitert.

Das Zehnfingersystem für die QWERTY-/QWERTZ-Tastatur wurde eingeführt, um jede Taste auf dem kürzesten Weg zu erreichen und damit das Schreiben zu beschleunigen und zu effektivieren. Die Daumen sind dabei, aufgrund der geringen Bewegungsfreiheit, lediglich zur Betätigung der Leertaste vorgesehen, die restlichen Finger übernehmen die übrigen Tasten.

Es existieren heute zahlreiche neu entwickelte Tastaturlayouts, wie beispielsweise das 1936 erfundene Dvorak-Tastaturlayout des Amerikaners August Dvorak [WV Wikidvorak] oder das 2005 von „Jugend forscht“ preisgekrönte Layout namens RISTOME [WV Ristome]. Diese Systeme erlauben schnelleres und für die Gelenke gesünderes Tippen. Aufgrund der weiten Verbreitung des QWERTZ-Systems und des enormen Aufwands, der für die Einführung eines neuen Systems nötig wäre, haben sie sich bislang aber nur in kleinen Kreisen durchgesetzt.

2. Planung

2.1 Pflichtenheft

Das Pflichtenheft enthält eine Zusammenfassung aller Anforderungen an das Software-Produkt. Es stellt die verbindliche Grundlage für die anschließenden Entwurfs- und Implementierungsentscheidungen dar. Das Pflichtenheft ist nach Helmut Balzert [Balzert 00. S.111 ff] aufgegliedert.

2.1.1 Zielbestimmung

Das Software-Produkt soll dem Benutzer ermöglichen, mittels Rechnerunterstützung das Zehnfingersystem auf einer Computertastatur zu erlernen und die eigene Leistung zu verbessern. Das Hauptaugenmerk soll dabei auf die dynamische, von den Tippfehlern des Anwenders abhängige Erstellung von Texten gerichtet werden.

2.1.1.1 Musskriterien

Allgemein

- Verwendung des deutschen Tastaturlayouts (QWERTZ-System)
- Ausreichende Anzahl an Trainingslektionen (mindestens zehn)

Einstellungen

- Auswahl einer Lektion
- Dauer der Lektion
- Reaktion auf Tippfehler
- Hilfestellungen

Schreibtraining

- Diktat über ein Laufband
- Virtuelle Tastatur mit Leuchttasten
- Statusinformationen
- Pausefunktion
- Vorzeitige Beendigung

Ergebnisse

- Auswertung durchgeführter Lektionen
- Fehlerquoten verwendeter Schriftzeichen

Veröffentlichung

- Lauffähigkeit unter Windows XP
- Bedienungsanleitung
- Auslieferung als Installationspaket auf CD

2.1.1.2 Wunschkriterien

- Bewertung durchgeführter Lektionen
- Aktualisierungsfunktion über eine Internetverbindung
- Wizard beim ersten Programmstart für grundlegende Einstellungen
- Veröffentlichung unter Linux (X11)
- Veröffentlichung unter Mac OS
- Downloadmöglichkeit der Software über eine Internetseite
- Einordnung der Lektionen in Themen (vom Benutzer auswählbar)
- Internationalisierung
- Anbindung an eine Onlinedatenbank

2.1.1.3 Abgrenzungskriterien

Keine Diktierfunktion statischer Texte (z.B. vom Benutzer selbst erstellter Lektionen)

2.1.2 Produkteinsatz

Das Produkt dient der Erlernung des Zehnfingersystems im Eigenstudium.

2.1.2.1 Anwendungsbereiche

- Bevorzugt privater Anwendungsbereich, aber auch Weiterbildung für den Beruf
- Für den schulischen Bereich eher ungeeignet, da ein Leistungsvergleich aufgrund dynamischer Texte nicht exakt genug ist

2.1.2.2 Zielgruppen

Der Schreibtrainer richtet sich in erster Linie an Personen, die

- bislang keinen oder wenig Umgang mit der Tastatur hatten
- bereits ohne das Zehnfingersystem viel mit der Tastatur arbeiten und nun das Zehnfingersystem erlernen möchten
- bereits erfahrene Anwender des Zehnfingersystems sind und sich nur in Punkto Geschwindigkeit und Fehlerquote verbessern möchten

2.1.2.3 Betriebsbedingungen

Einsatz an herkömmlichen Computern

2.1.3 Produktübersicht

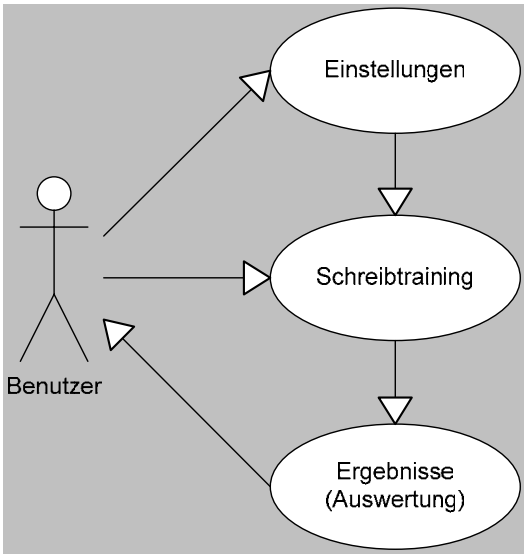


Abb. 2.1: Übersichtsdiagramm

2.1.4 Produktfunktionen

Die Produktfunktionen sollen in drei Darstellungen, *Einstellungen*, *Schreibtraining* und *Ergebnisse* aufgeteilt werden.

Zu Beginn, in der Darstellung *Einstellungen*, kann der Benutzer zwischen aufeinander aufbauenden Lektionen auswählen. Die Lektionen sind für die schrittweise Erlernung der Tastatur vorgesehen, indem zu Beginn nur wenige, später immer mehr Tasten erlernt werden. Zusätzlich sollen sich Einstellungen vornehmen lassen, die die Dauer der Lektion, die visuellen Hilfestellungen und die Reaktion auf Tippfehler beeinflussen.

In der zweiten Darstellung, dem *Schreibtraining*, werden dem Anwender entsprechende Texte aus der gewählten Lektion über ein Laufband diktiert und die zu drückenden Tasten, je nach Einstellungen in der vorherigen Darstellung, visuell auf einer grafischen Tastatur angezeigt. Zusätzlich werden Informationen über eine Statusleiste eingeblendet.

Nach Beendigung des Schreibtrainings soll die Darstellung *Ergebnisse* dem Benutzer aktuelle und vergangene Ergebnisse der Trainingslektionen präsentieren.

2.1.4.1 Einstellungen

Einstellungsmöglichkeit: Auswahl einer Lektion

- Ziel: Der Benutzer kann den Tastaturbereich der Trainingslektion festlegen
- Vorbedingung: -
- Nachbedingung Erfolg: Gewünschte Lektion wird selektiert und im Training verarbeitet
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: -
- Voreinstellung: Nach dem ersten Programmstart ist Lektion 1 selektiert, anschließend immer die jeweils zuletzt trainierte Lektion
- Beschreibung: -

Einstellungsmöglichkeit: Dauer der Lektion

- Ziel: Der Benutzer kann die Dauer der Trainingslektion festlegen
- Vorbedingung: -
- Nachbedingung Erfolg: Das Schreibtraining wird nach Ablauf der Dauer beendet
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: -
- Voreinstellung: Nach dem ersten Programmstart ist ein Zeitlimit von fünf Minuten eingestellt, anschließend die vom Benutzer zuletzt gewählte Dauer
- Beschreibung: Wahl der Dauer entweder über ein Zeitlimit in Minuten oder eine Anzahl zu tippender Zeichen

Einstellungsmöglichkeit: Reaktion auf Tippfehler

- Ziel: Der Benutzer kann festlegen, wie die Software auf Tippfehler reagiert und welches Verhalten daraufhin vom Benutzer erforderlich ist
- Vorbedingung: Für die Korrektur eines Tippfehlers muss die Blockierung (siehe Beschreibung) aktiviert sein
- Nachbedingung Erfolg: Die Software reagiert auf einen Tippfehler und fordert bei Bedarf entsprechende Maßnahmen vom Benutzer
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: -
- Voreinstellung: Nach dem ersten Programmstart ist die Blockierung aktiviert, anschließend die vom Benutzer zuletzt gewählten Einstellungen
- Beschreibung: Vier verschiedene Einstellungen sind vorgesehen:

  1. Tippfehler werden übergangen und es kann das nächste Schriftzeichen getippt werden

  2. Tippfehler werden blockiert und es wird gewartet, bis das korrekte Schriftzeichen eingegeben, der Fehler also verbessert wird

  3. Ist die Blockierung aktiviert, kann zusätzlich eine Korrektur aktiviert werden. Sie verlangt, bevor der Tippfehler verbessert werden kann, die Rücklauftaste (Löschtaste) zu betätigen

  4. Zusätzlich zu den genannten Einstellungen kann ein akustisches Signal, das bei jedem erfolgten Tippfehler ausgegeben wird, aktiviert werden

Einstellungsmöglichkeit: Hilfestellungen

- Ziel: Der Benutzer kann visuelle Hilfestellungen an- und abschalten.
- Vorbedingung: -
- Nachbedingung Erfolg: Die Software unterstützt den Benutzer während des Schreibtrainings mit den aktivierten Hilfestellungen

- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: -
- Voreinstellung: Nach dem ersten Programmstart sind alle visuellen Hilfestellungen aktiviert, anschließend die vom Benutzer zuletzt gewählten Einstellungen
- Beschreibung: Hilfestellungen sind eine virtuelle Tastatur und ein Hilfetext (vgl. Kapitel 2.1.4.2)
    1. Die virtuelle Tastatur stellt während des Schreibtrainings die zu drückenden Tasten farbig dar
    2. Der Hilfetext zeigt unterstützende Texte in einer Statusleiste

2.1.4.2 Schreibtraining

Prozess: Diktat über ein Laufband

- Ziel: Der Text des Diktats bewegt sich einzeilig von rechts nach links
- Vorbedingung: Der Text des Diktats wurde generiert
- Nachbedingung Erfolg: Das Diktat wird während des gesamten Schreibtrainings in einem Laufband wiedergegeben
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Schreibtraining wurde gestartet
- Voreinstellung: -
- Beschreibung:
    1. Von rechts nach links bewegende Schrift ermöglicht, das Diktat einzeilig und so für den Benutzer deutlich darzustellen
    2. Das zu tippende Schriftzeichen wird aufgrund der Bewegung markiert dargestellt

Prozess: Virtuelle Tastatur mit farbigen Tasten

- Ziel: Die virtuelle Tastatur stellt während des Schreibtrainings die anzuschlagenden Tasten farbig dar
- Vorbedingung: Die virtuelle Tastatur (Leuchttastatur) ist aktiviert und ein zu tippendes Schriftzeichen ist vorhanden
- Nachbedingung Erfolg: Alle für ein Schriftzeichen anzuschlagenden Tasten werden farbig gekennzeichnet
- Nachbedingung Fehlschlag: Keine Taste wird markiert
- Auslösendes Ereignis: Schreibtraining wurde gestartet
- Voreinstellung: -
- Beschreibung: Die Farbe der Taste kennzeichnet den Finger, der für dessen Betätigung verwendet werden soll

Prozess: Statusinformationen

- Ziel: Zusatzinformationen werden in einer Statusleiste angezeigt
- Vorbedingung: Die Zusatzinformationen sind aktiviert
- Nachbedingung Erfolg: Informationen über aktuelle Trainingwerte und die zu drückende Taste werden während des Schreibtrainings angezeigt
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Schreibtraining wurde gestartet
- Voreinstellung: -
- Beschreibung: Drei Informationen sind vorgesehen:
    1. Anzahl der Anschläge pro Minute
    2. Anzahl der Tippfehler
    3. Beschreibung der zu verwendenden Finger für die anzuschlagenden Tasten

Funktion: Pause

- Ziel: Das Schreibtraining kann unterbrochen und nach Bedarf fortgesetzt werden
- Vorbedingung: Das Diktat wurde gestartet
- Nachbedingung Erfolg: Das Schreibtraining wird unterbrochen und kann nach Bedarf fortgesetzt werden
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Die Pausefunktion wurde aktiviert
- Voreinstellung: -
- Beschreibung: -

Funktion: Vorzeitige Beendigung

- Ziel: Das Schreibtraining kann vorzeitig beendet bzw. abgebrochen werden
- Vorbedingung: Das Diktat wurde gestartet
- Nachbedingung Erfolg: Das Schreibtraining wird beendet
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Die vorzeitige Beendigung wurde aktiviert
- Voreinstellung: -
- Beschreibung: Zwei Möglichkeiten sind vorgesehen:
    1. Das Schreibtraining wird vorzeitigt beendet, jedoch wie ein abgeschlossenes Training behandelt
    2. Das Schreibtraining wird abgebrochen und zurückgesetzt, es verhält sich wie ein nie stattgefundenes Schreibtraining

2.1.4.3 Ergebnisse

Darstellung: Auswertung durchgeführter Lektionen

- Ziel: Auswertungen durchgeführter Lektionen lassen sich darstellen
- Vorbedingung: Mindestens eine Lektion wurde durchgeführt
- Nachbedingung Erfolg: Ergebnisse und Bewertungen durchgeführter Lektionen werden angezeigt
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Schreibtraining wurde beendet oder Auswertungen werden vom Anwender aufgerufen
- Voreinstellung: -
- Beschreibung: Die im Kapitel 2.1.5.2 aufgeführten Lektionendaten werden tabellarisch dargestellt

Darstellung: Fehlerquoten verwendeter Schriftzeichen

- Ziel: Auswertungen verwendeter Schriftzeichen lassen sich darstellen
- Vorbedingung: Mindestens eine Lektion wurde durchgeführt
- Nachbedingung Erfolg: Ergebnisse und Gewichtungen verwendeter Schriftzeichen werden angezeigt
- Nachbedingung Fehlschlag: -
- Auslösendes Ereignis: Schreibtraining wurde beendet oder Auswertungen werden vom Anwender aufgerufen
- Voreinstellung: -
- Beschreibung: Die im Kapitel 2.1.5.3 aufgeführten Schriftzeichendaten werden tabellarisch dargestellt

2.1.5 Produktdaten

Die Produktdaten sind in einer Datenbank zu speichern.

2.1.5.1 Diktattexte

Alle Texte für die Diktate sind in einer Tabelle zu hinterlegen.

2.1.5.2 Lektionendaten

Über jede durchgeführte Lektion sind folgende Daten zu speichern:
Nummer der durchgeführten Lektion, Zeitpunkt, Dauer in Minuten, Länge in Schriftzeichen, Anzahl der Tippfehler, Anschlagszahl pro Minute

2.1.5.3 Schriftzeichendaten

In den Lektionen verwendete Schriftzeichen sind mit folgenden Informationen zu speichern:
Schriftzeichen, Fehlerzahl und Fehlerquote (in Prozent)

2.1.6 Produktleistungen

Durch Speichern der Daten und Anfordern neuer Diktattexte zur Laufzeit des Schreibtrainings dürfen keine sichtbaren Verzögerungen in der Laufschrift, der Aktualisierung des Diktats und den visuellen Hilfestellungen auftreten.

2.1.7 Qualitätsanforderungen

Qualitätsmerkmale nach DIN ISO 9126 [Balzert 00, S.1102f.]:

Produktqualität	sehr gut	gut	normal	nicht relevant
Funktionalität				
Richtigkeit			x	
Angemessenheit		x		
Interoperabilität			x	
Ordnungsmäßigkeit			x	
Sicherheit				x
Zuverlässigkeit				
Reife			x	
Fehlertoleranz			x	
Wiederherstellbarkeit			x	
Benutzbarkeit				
Verständlichkeit		x		
Erlernbarkeit		x		
Bedienbarkeit	x			
Effizienz				
Zeitverhalten		x		
Verbrauchsverhalten			x	
Änderbarkeit				
Analysierbarkeit			x	
Modifizierbarkeit		x		
Stabilität			x	
Prüfbarkeit			x	
Übertragbarkeit				
Anpassbarkeit		x		
Installierbarkeit			x	
Konformität			x	
Austauschbarkeit				x

Tabelle 2.1: Qualitätsmerkmale des Produkts

2.1.8 Benutzungsoberfläche

Die Bedienung sämtlicher Einstellungen, Dialog- und Mitteilungsfenster soll auf Mausbedienung ausgelegt werden. Das Schreibtraining selbst soll ausschließlich über die Tastatur bedient werden. Der Start des Diktats soll ebenfalls über die Tastatur erfolgen, um eine Verzögerung beim Wechsel von der Maus zur Tastatur zu vermeiden.

Sämtliche Einstellungen, die das Schreibtraining beeinflussen und unter Umständen oft geändert werden, sollen im Anwendungsfenster durchführbar sein. Einstellungen, die nach dem ersten Programmstart durchgeführt und voraussichtlich selten geändert werden, sollen in einem extra Dialog aufruf- und editierbar sein.

2.1.9 Technische Produktumgebung

2.1.9.1 Software

Windows XP (Musskriterium)

Mac OS X und Linux (X11) (Wunschkriterien)

2.1.9.2 Hardware

Herkömmlicher PC mit den Eingabegeräten Maus und Tastatur

2.1.9.3 Orgware

Internetverbindung für die Aktualisierungsfunktion (Wunschkriterium)

2.1.9.4 Produkt-Schnittstellen

-

2.1.10 Spezielle Anforderungen an die Entwicklungsumgebung

Keine Abweichungen von der technischen Produktumgebung

2.1.11 Gliederung in Teilprodukte

Vorgesehen sind drei Teilprodukte. Teilprodukt 1 stellt das Intelligenzkonzept für die Generierung des Diktats dar. Teilprodukt 2 beschreibt die für den intelligenten Schreibtrainer notwendige Datenbank. Teilprodukt 3 ist die grafische Benutzungsoberfläche, die das Intelligenzkonzept mit Hilfe der Datenbank umsetzt und darstellt.

2.1.12 Ergänzungen

- Der Schreibtrainer soll eine kostenlose Alternative zu vorhandenen Schreibtrainern darstellen
- Die Entwicklung soll für die Programmiersprache C++ ausgelegt sein

2.2 Auswahl der Entwicklungsumgebung

Für die Realisierung der grafischen Benutzungsoberfläche (engl. Graphical user interface, kurz *GUI*) und der Datenbank sollen verschiedene auf dem Markt vorhandene Systeme verglichen und hinsichtlich ihrer Eignung für das Projekt ausgewählt werden.

2.2.1 Analyse vorhandener *GUI*-Klassenbibliotheken

Für die Umsetzung der grafischen Oberfläche soll eine *GUI*-Klassenbibliothek genutzt werden. Deswegen wurde eine Internet-Recherche durchgeführt, um vorhandene *GUI*-Bibliotheken aufzufinden und zu vergleichen. Es wurden Klassenbibliotheken gesucht, die

- mit der Programmiersprache C++ arbeiten
- in aktuellen Software-Projekten Verwendung finden
- Veröffentlichung unter Windows erlauben
- möglichst unter die *GNU*-Lizenz (*GPL*) gestellt, vor allem aber kostenlos sind
- Veröffentlichung unter der *GNU*-Lizenz (*GPL*) erlauben
- Methoden für die Anbindung an ein Datenbanksystem besitzen
- möglichst Multiplattform-Entwicklung gestatten

Die Suchkriterien schränken die Wahl sehr ein, obwohl eine Vielzahl an Bibliotheken für die *GUI*-Realisierung existiert. Besonders die Lizenzierung und Realisierung unter Windows lässt viele Systeme für diesen Vergleich nicht zu.
In die engere Wahl kamen zwei Klassenbibiotheken, *wxWidgets* und *Qt*.

Alle zwei Bibliotheken bieten die Möglichkeit der Multiplattform-Entwicklung (cross-platform), daher soll diese kurz erläutert werden:
Jedes Betriebssystem verwendet eigene Schnittstellen für die Darstellung grafischer Oberflächen (application programming interface, kurz API). Unter Windows lautet diese Schnittstelle beispielsweise *Windows-API*. Durch die Nutzung von Multiplattform-Entwicklungssystemen wird es möglich, Programme zu erstellen, die unter verschiedenen Betriebssystemen kompiliert und ausgeführt werden können. Die *GUI*-Klassenbibliothek übernimmt dabei die Aufgabe, die Schnittstellen des jeweiligen Systems zu nutzen, ohne dass der Programmierer selbst große Veränderungen im Quelltext vornehmen muss.

Für das vorliegende Projekt ist die Multiplattform-Entwicklung sehr interessant, da auf diese Weise die Möglichkeit besteht, die Anwendung zu einem späteren Zeitpunkt auch für Betriebssysteme wie *MacOS* oder *Linux* anzubieten.

Die zwei Klassenbibliotheken sollen hier kurz vorgestellt werden:

- *wxWidgets*

  Das Projekt *wxWidgets* wurde 1992 von Julian Smart an der Universität von Edinburgh, Schottland ins Leben gerufen. Bis heute hat sich das Projekt beachtenswert entwickelt und bietet eine sehr umfangreiche Klassenbibliothek für die *GUI*-Realisierung und weitergehende Funktionen an. Es werden unter anderem die bekannten Betriebssysteme *Windows*, *MacOS* und *X11* (*X Window System*) unterstützt (Multiplattform-Entwicklung). Zudem ist das System für die Einbindung verschiedener Datenbanktreiber vorbereitet.
  wxWidgets ist kostenlos und erlaubt aufgrund einer eigenen Lizenz jegliche Nutzung, sowohl unter *GNU*-Lizenz (*GPL*) als auch kommerziell. Technische Unterstützung erhält der Anwender über eine umfangreiche Dokumentation

mit Beispielen und Demonstrationen, zudem ist ein passendes Fachbuch erhältlich. *wxWidgets* wird von vielen Organisationen und Einzelentwicklern genutzt, bekannte Firmen sind zum Beispiel *AOL*, *AMD* und *Xerox*. [WV Wxwidgets]

- *Qt*

  Qt wird von der Firma *Trolltech* herausgegeben, die 1994 von Haavard Nord und Eirik Chambe-Eng gegründet wurde. Der Hauptsitz befindet sich heute in Oslo, Norwegen. *Qt* ist mittlerweile eine der bekanntesten Klassenbibliotheken, bestehend aus ca. 400 Klassen für verschiedenste Funktionalitäten. Dazu zählen nicht nur Klassen für die *GUI*-Realisierung; auch für Datenbanknutzung, XML, Netzwerk, Multithreading und viele weitere Bereiche werden hilfreiche Klassen angeboten. Zusätzliche Software-Werkzeuge helfen beispielsweise bei der Erstellung von Build-Umgebungen (*qmake*), der *GUI*-Realisierung (*Qt Designer*) oder der Internationalisierung (*Qt Linguist*). Die Multiplattform-Entwicklung erlaubt, Programme unter Betriebssystemen wie *Windows*, *MacOS* oder *X11* (*X Window System*) zu kompilieren.

  Qt bietet zwei Lizenzen an. Mit einer kostenpflichtigen Lizenz lässt sich kommerzielle Software entwickeln. Eine kostenfreie *GNU*-Lizenz (*GPL*) ermöglicht die Verbreitung von Software ausschließlich als Open-Source-Produkt, also ebenfalls unter die *GNU*-Lizenz gestellter Software. Diese zweiteilige Lizenz wurde schrittweise eingeführt, zuerst für das *X Window System* (X11) und *MacOS*. Mit der Veröffentlichung von *Qt 4* im Jahr 2005 wurde auch die *Windows*-Version unter *GPL* gestellt.

  Dank der weiten Verbreitung von *Qt* findet sich eine große Anzahl unterstützender Literatur auf dem Markt, zudem wird mit *Qt* eine sehr

umfangreiche Dokumentation ausgeliefert. Dank eigenen Hilfe-Browsers wird praktisches Navigieren und Suchen in der Dokumentation, zahlreichen Beispielen und Tutorien ermöglicht.

Das wohl bekannteste System, das mit *Qt* entwickelt wurde, ist die grafische Arbeitsumgebung *KDE* (K desktop environment) für Unix-Systeme. Aber auch Firmen wie *Adobe*, *IBM* oder *Siemens* sowie zahlreiche Einzelentwickler nutzen erfolgreich die *Qt*-Klassenbibiothek. [WV Trolltech]

Beide Klassenbibliotheken eignen sich sehr gut für die Umsetzung der dieser Diplomarbeit zugrunde liegenden Idee. Aufgrund der weiten Verbreitung von Qt, des großen Funktionsumfangs und der ausführlichen Dokumentation fiel die Entscheidung auf die Verwendung der Bibliothek von *Trolltech*. Die erst im Jahr 2005 erfolgte Veröffentlichung als *GNU GPL*-Version unter Windows bietet zudem interessante Anreize, bisher nur kommerziell unter Windows genutzte Fähigkeiten auch auf die Open-Source-Welt anzuwenden. Weitere, im nachfolgenden Kapitel vorgestellte Fähigkeiten der Klassenbibliothek verfestigten die Entscheidung.

2.2.2 Die GUI-Klassenbibliothek *Qt*

Die in der Analyse ausgewählte Klassenbibliothek *Qt* wird in der Version 4.1.1 verwendet. Nachfolgend sollen Fähigkeiten der Bibliothek vorgestellt werden, die für das Projekt besonders von Interesse sind.

- Benutzungsoberfläche

  Qt bietet alle von grafischen Oberflächen bekannten Interaktionselemente. Ein eigenes Werkzeug, *Qt Designer*, bietet die Möglichkeit, Oberflächenlayouts zu generieren, ohne diese über Programmcode zu entwickeln.

Aufgrund der speziellen Oberfläche (Laufband, virtuelle Tastatur, etc.), die für das hier vorgestellte Projekt dienen soll, wird jedoch auf die Anwendung des *Qt Designers* verzichtet.

- Build-Umgebung

  Mit Hilfe von Projektdateien und dem Werkzeug *qmake* lassen sich Build-Umgebungen auf unterschiedlichen Plattformen erzeugen. Das Erstellen an das Betriebssystem angepasster *Make*-Dateien wird so erleichtert.

- Blackbox-Verhalten

  Qt bietet die Fähigkeit, Methoden sowohl innerhalb einer Klasse, als auch klassenübergreifend individuell kommunizieren zu lassen. Über so genannte *Signals* und *Slots* lassen sich Methoden mit Hilfe des Befehls *connect* dynamisch verbinden. Diese Verbindung ermöglicht auch die Übergabe von Parametern.

  Signals und *Slots* können als Alternative zu dem *Callback*-Verfahren angesehen werden, bei dem Zeiger auf Methoden als Parameter übergeben werden, um diese von anderen Methoden aufrufbar zu machen. Vorteile der *Signals* und *Slots* im Gegensatz zu *Callback*-Methoden sind zwei Eigenschaften. Zum einen sind sie typensicher (type save), es kann also davon ausgegangen werden, dass die korrekten Parameter zurückgegeben werden. Zum anderen können ganze Klassen als Blackbox angesehen werden, lediglich *Signals* und *Slots* müssen bekannt sein. Sie können dann beliebig und vor allem dynamisch verbunden werden.

- Internationalisierung

  Ein weiteres praktisches Werkzeug stellt *Qt Linguist* zur Verfügung. Damit lassen sich alle verwendeten Texte der Anwendung aus dem Quelltext extrahieren und zusammenfassen. Der so entstehende Auszug in Form einer Auflistung von Texten kann dann auf einfach Weise angepasst oder übersetzt werden. Das Werkzeug sorgt in einem weiteren Schritt dafür, dass die entstandenen Textdateien zur Laufzeit geladen und alle vorhandenen Texte entsprechend ersetzt werden. *Qt Linguist* macht es überflüssig, Texte in Headerdateien auszulagern, um sie leicht verändern zu können.

2.2.3 Analyse vorhandener Datenbanksysteme

Für die Umsetzung eines dynamischen Schreibtrainers wurde im Pflichtenheft eine Datenbank vorgesehen, die sowohl die Quelle des Diktats darstellen wie auch sämtliche Benutzerdaten speichern soll. Da es sich um relativ einfache, formatierte Datenbestände handelt, die verwaltet werden müssen, eignet sich für diesen Zweck eine relationale Datenbank [Balzert 00, S.743].

Vorhandene relationale Datenbanksysteme lassen sich in drei Kategorien einteilen:

- Datenbankserver

  Bei der Benutzung einer serverbasierten Datenbank verwaltet ein Server das gesamte Datenbanksystem. Eine Anwendung kommuniziert lediglich mit dem Server. Diese Kategorie der Datenbanken wird bevorzugt verwendet, wenn mehrere Anwendungen (meist Clients in einem Netzwerk) auf eine gemeinsame Datenbank zugreifen sollen.

  Beispiele für Datenbankserver sind *MySql*, *PostgreSQL*, *Microsoft SQL Server*, *Oracle Database*

- Dateibasierte Datenbanken

  Bei dateibasierten Datenbanksystemen greift die Anwendung direkt auf Datenbankdateien zu und arbeitet mit ihnen. Das bedeutet, dass alle Datenbankoperationen von der Anwendung selbst durchgeführt werden. Modifizierungen an einer Datenbankdatei können normalerweise nur von einer Anwendung zur gleichen Zeit durchgeführt werden. Dateibasierte Datenbanken werden häufig für Softwareprodukte verwendet, die eine eigene Datenbank besitzen, also ohne zentrale Datenverwaltung auskommen.

  Beispiele: *Microsoft Access*, *SQLite*, *Paradox*, *Embedded Firebird*

- *XML*-Datenbanken

  Datenbanken die auf *XML* (Extensible markup language) basieren, arbeiten wie dateibasierte Datenbanken mit Dateien, die jedoch im *XML*-Format gespeichert sind. Im Gegensatz zu klassischen relationalen Datenbanksystemen, bei denen die Daten in einer eindeutigen Struktur in Form von Tabellen gespeichert sind, werden die Daten einer *XML*-Datenbank in einer komplexen hierarchischen Struktur abgelegt. *XML*-Datenbanken werden für Netzwerkapplikationen, vorwiegend im Internet, eingesetzt.

  Beispiele: *Tamino*, *eXist*, *Apache Xindice*, *Infonyte*

Der Schreibtrainer soll als Einzelanwendung mit integrierter Datenbank ausgeliefert werden. Die Verwendung eines Datenbankservers wäre mit einem zusätzlichen Prozess für den Server verbunden, der für den Zugriff von nur einer Anwendung überflüssig ist. Es ist zwar denkbar, das System um die Verwendung eines zentralen Servers zu erweitern, auf den über eine Internetverbindung zugegriffen werden könnte (siehe Wunschkriterium im Pflichtenheft), dies soll aber vorerst keine Rolle spielen.

XML-Datenbanken gewinnen zunehmend an Bedeutung, sind aber bei großen Datenmengen meist nicht so schnell wie klassische Datenbanksysteme. Zudem existieren (noch) keine standardisierten Abfragesprachen und Schnittstellen. Da gerade die Performance der Datenbank für den geplanten Schreibtrainer eine große Rolle spielt, ist eine *XML*-Datenbank eher ungeeignet.
Die Vorteile für die Einbindung einer dateibasierten Datenbank liegen auf der Hand. Dateibasierte Datenbanken sind schnell, arbeiten mit einer konventionellen Tabellenstruktur und lassen sich einfach in das System einbinden. Sie sind für den Zweck einer eigenständigen Software gut geeignet. Aus diesem Grund soll für den Schreibtrainer eine dateibasierte Datenbank verwendet werden.

Die Klassenbibliothek *Qt* bietet die Möglichkeit, Datenbanktreiber in das System einzubinden und besitzt zahlreiche Methoden zur Datenmanipulation. Treiber für die dateibasierte Datenbank *SQLite* werden sogar mit dem Installationspaket von *Qt* ausgeliefert. Aus diesem Grund soll *SQLite* im Hinblick auf die Eignung für den Schreibtrainer genauer betrachtet werden.
Das Datenbanksystem *SQLite*

- unterliegt keinem Urheberrecht (public domain) und ist daher für jegliche Verwendung kostenlos
- benötigt keine eigene Konfiguration
- speichert die gesamte Datenbank in einer Datei
- verwendet einen Großteil der von der Abfragesprache *SQL* bekannten Befehle
- ist in vielen Bereichen schneller als vergleichbare Serverdatenbanken
- besitzt eine eigene Schnittstelle für die Programmiersprache C++
- unterstützt Datenbanken bis zu einer Größe von zwei Terrabyte

Zudem existieren zahlreiche Software-Werkzeuge von verschiedenen Anbietern zur Administration des Datenbestandes einer *SQLite*-Datenbank.

Aufgrund dieser Eigenschaften und der einfach zu bewerkstelligen Anbindung an *Qt* eignet sich diese Datenbank gut für den geplanten Schreibtrainer. Ein weiterer Vorteil, der sich durch die Verwendung der weit verbreiteten Abfragesprache *SQL* (Structured Query Language) ergibt, ist, dass die Software jederzeit für die Nutzung eines *SQL*-Datenbankservers umgerüstet werden kann, ohne die Abfragesprache ändern zu müssen. Um beispielsweise das unter *GNU*-Lizenz (*GPL*) gestellte Datenbanksystem *MySql* zu nutzen, müssen lediglich die Treiber und die Anbindung der Software an die Datenbank ausgetauscht werden.

2.2.4 Das Datenbanksystem *SQLite*

Das Datenbanksystem *SQLite* wird in der Version 3.3.4 verwendet. Für die Verwendung müssen folgende Eigenschaften beachtet werden:

- Es kennt nur fünf verschiedene Datentypen:

NULL	Nullwert
INTEGER	Vorzeichenbehaftete Ganzzahl (Größe zwischen 1 und 8 Byte)
REAL	Fließkommazahl (Größe: 8 Byte, Format: IEEE)
TEXT	Text (UTF-Format)
BLOB	Es wird exakt das gespeichert, was eingegeben wurde

- Felder lassen sich nur über den Spaltenindex ansprechen, nicht aber über den Namen des Feldes. Daher sollten Abfragen vermieden werden, die das Schriftzeichen * (Stern) verwenden, um alle Felder einer Tabelle anzusprechen. Einer Vertauschung von Feldinhalten kann so entgegengewirkt werden.
- *SQLite* beherrscht eine geringere Anzahl an *SQL*-Befehlen als vergleichbare *SQL*-Serverdatenbanken.
- Die Klassenbiliothek *Qt* stellt mehr *SQL*-Datenbankfunktionen bereit, als die C++ Schnittstelle von *SQLite* bietet. Daher kann über eine Methode *hasFeature*

der *Qt*-Klasse *QSqlDriver* getestet werden, ob das aktuelle Datenbanksystem bestimmte Funktionen besitzt.

Um die *SQLite*-Datenbank zu verwalten, werden einige kostenlose Administrationswerkzeuge von Dritten angeboten. Das bekannteste, *SQLite* Administrator, wird in der vorliegenden Arbeit verwendet, um die Datenbankstruktur festzulegen und Tabellen mit Daten zu füllen.

3. Entwurf

Das vorliegende Kapitel befasst sich mit dem Entwurf der Softwarearchitektur. Grundlage hierfür stellt das in Kapitel 2 aufgeführte Pflichtenheft und die gewählte Entwicklungsumgebung dar.

Die Zielbestimmung im Pflichtenheft legt fest, das Hauptaugenmerk auf die dynamische, von den Tippfehlern des Anwenders abhängige Erstellung von Texten zu richten. Aus diesem Grund werden im vorliegenden Kapitel in erster Linie Bereiche erläutert, die für die Realisierung der Zielbestimmung notwendig sind. Dazu gehören Grundsatzentscheidungen, die Ausarbeitung eines geeigneten Konzepts sowie dessen Auswirkungen auf die Datenbank und den Programmentwurf. Die Wahl der Tastaturoberfläche und des Layouts wird diskutiert und ermöglicht eine Festlegung für die Darstellung der virtuellen Tastatur. Ein Entwurf der Oberflächengestaltung gibt der Implementierung eine Richtung des Designs vor.

Die detaillierte Erläuterung aller Klassen und Methoden würde den formalen Rahmen dieser Diplomarbeit überschreiten. Daher bieten Diagramme im Anhang dieser Arbeit eine Übersicht über die gesamte Datenbankstruktur und den Aufbau der Klassen.

3.1 Grundsatzentscheidungen

Bevor ein geeignetes Konzept für die dynamische Generierung der Texte entwickelt werden kann, müssen Grundsatzentscheidungen getroffen werden. Dazu gehören einige Einsatzbedingungen, die von der Software erfüllt werden müssen; außerdem wird ein geeignetes Layout für die Darstellung der virtuellen Tastatur festgelegt.

3.1.1 Einsatzbedingungen

Das Produkt wird sequentiell entworfen, das heißt, alle Befehle werden nacheinander abgearbeitet. Bei der Implementierung des dynamischen Teils des Schreibtrainers wird darauf geachtet, dass die im Pflichtenheft festgelegten Leistungsanforderungen bezüglich der Zeit (vgl. Kapitel 2.1.5) eingehalten werden.

Das Produkt wird für einen Anwender vorgesehen. Da die Diktate stark von den gespeicherten Fähigkeiten des Nutzers abhängen, wäre für eine Mehrbenutzer-Verwendung eine zusätzliche Anmeldeprozedur nötig. Die Möglichkeit, vorhandene Anwenderdaten zurückzusetzen (zu löschen) wird jedoch vorgesehen, um so den Einsatz für einen neuen Anwender vorzubereiten.

Die Klassenbibliothek *Qt* verwendet den Unicode Standard 4.0. Insbesondere die Typen *QString* und *QChar* verwenden diesen Zeichensatz. Der Coderaum von Unicode umfasst 32 Bit und kann beinahe alle weltweit existierenden Schriftzeichen annehmen. Ein weiterer Vorteil von *Qt* ist es, dass Tastatureingaben mit dem bekannten Coderaum von 8 Bit direkt in ein Unicode-Zeichen übersetzt werden. Um die Vorteile von Unicode, sprich Plattformunabhängigkeit und Vereinfachung zur Internationalisierung, auch für dieses Produkt zu nutzen, werden die Unicode-Fähigkeiten von *Qt* genutzt. Die Datenbank *SQLite* arbeitet, wenn auch etwas eingeschränkt, ebenfalls mit dem Unicode-Zeichensatz und lässt sich daher gut verwenden.

3.1.2 Tastaturoberfläche und -layout

Die Tasten einer Computertastatur erfüllen unterschiedliche Aufgaben. So existieren zusätzlich zu Tasten für Schriftzeichen auch Zusatztasten, wie Funktions-, Umschalt- und Navigationstasten, sowie ein numerisches Tastenfeld. Je nach Hersteller und Computertyp unterscheidet sich die Aufteilung dieser Tasten. Ein *Windows*-PC besitzt ein anderes Tastaturlayout als beispielsweise ein *Apple Macintosh*-PC. Bei Notebooks sind aus Platzmangel oftmals Tasten anders angeordnet als bei Desktop-PCs oder es werden sogar Tasten weggelassen. Zusätzlich bieten einige Hersteller eigene Tastaturoberflächen an, wie zum Beispiel ergonomische Tastaturen (siehe Abbildung 3.1) oder Tastaturen mit extra gekennzeichneten und farbigen Tasten. *Microsoft* führte mit dem Erscheinen von *Windows 95* eine erweiterte *IBM*-Tastatur ein (siehe Abbildung 3.2), die mit zusätzlichen *Windows*-Funktionstasten versehen ist. Sie ist mittlerweile bei *Windows*-PCs weit verbreitet. Auch *Apple* verwendet eigene Funktionstasten, die sich von den Windows-Tasten aber nur minimal unterscheiden (siehe Abbildung 3.3).

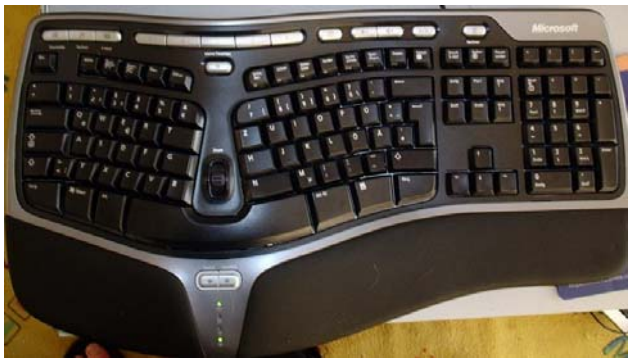


Abb. 3.1: Ergonomische Tastatur Microsoft (Quelle: [WV Wikitastatur])

Abb. 3.2: Windows-Tastatur von Cherry (Quelle: [WV Wikitastatur])

Abb. 3.3: MacOSX-Tastatur von Apple (Quelle: [WV Wikitastatur])

Um diesen Unterschieden gerecht zu werden, wird sich dieser Schreibtrainer auf das eigentliche Tastfeld beschränken, das aus Tasten für Buchstaben und Zahlen besteht, sowie einigen Tasten für Eingabe-, Funktions- und Umschaltmöglichkeiten. Diese sind bei fast allen Tastaturen gleich angeordnet und für die Erlernung des Zehnfingersystems besonders von Bedeutung.

Da die gebräuchlichsten Computer heute mit *Windows*- und *Apple*-Tastaturen arbeiten und der Schreibtrainer im Rahmen dieser Diplomarbeit für *Windows*, später aber auch für *Linux* und besonders für *Mac OS* veröffentlicht werden soll, wird er diese zwei bekannten deutschen Tastaturlayouts berücksichtigen. Sie unterscheiden sich vor allem durch die Anordnung und Verwendung der Modifizierungstasten *Alt* (engl. Alternative), *Alt Gr* (engl. Alternative Graphic) und *Ctrl* (engl. Control), oft auch mit dem deutschen *Strg* (Steuerung) bezeichnet, sowie speziellen Tasten, wie den Windowsfunktionstasten und bei Apple-Computern die „Apfeltaste“. Zudem sind die Sonderzeichen *@* und *€* bei diesen zwei Layouts an verschiedenen Stellen angeordnet (siehe Abb. 3.4 und 3.5).

Es wird daher die Möglichkeit für den Anwender vorgesehen, über eine Grundeinstellung zwischen Windows- und Apple-Tastaturlayout zu wechseln. Das

Layout wird dann über die virtuelle Tastatur während des Schreibtrainings angezeigt und die Unterschiede werden entsprechend behandelt.

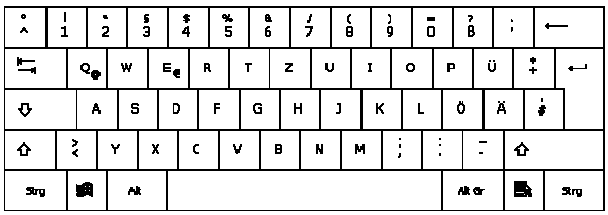


Abb. 3.4: Windows-Tastaturlayout

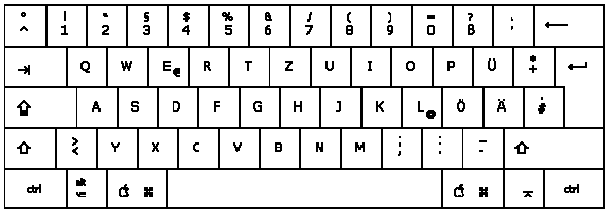


Abb. 3.5: Apple-Tastaturlayout

Um trotzdem ein gleiche Anzahl an Tasten anzubieten und so die Implementierung der virtuellen Tastatur zu vereinfachen, wird die Oberfläche fest auf 61 Tasten aufgeteilt. Die so entstehende Tastatur-Struktur kann beide vorgesehenen Layouts abbilden. Folgende Abbildung zeigt die Struktur der aufgeteilten Tastaturoberfläche.

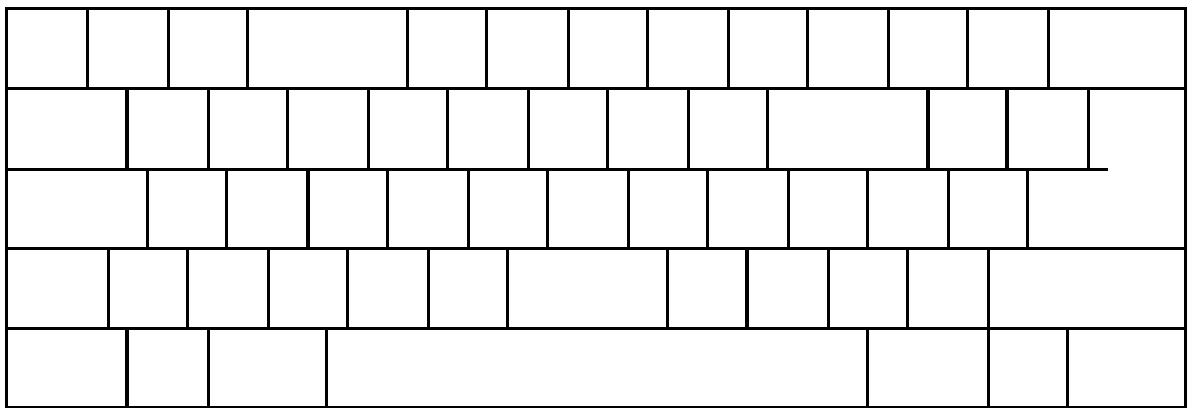


Abb. 3.6: Aufteilung des Tastaturlayouts auf 61 Tasten

Der Gebrauch einer Tastatur sieht außerdem vor, für bestimmte Schriftzeichen zwei Tasten zu betätigen, wie zum Beispiel das zusätzliche Drücken der Umschalttaste für die Großschreibung oder das zusätzliche Drücken der „AltGr“-

Taste für spezielle Sonderzeichen. Daher wird eingeplant, dass jedem Schriftzeichen eine zweite Taste zugeordnet werden kann.

Auf die Darstellung von mehr als zwei Tasten, wie zum Beispiel die Möglichkeit, die Taste *Alt Gr* mit Hilfe der Tasten *Strg* und *Alt* zu simulieren, wird keine Rücksicht genommen. Dies stellt einen Spezialfall dar und ist bei einem normalen Gebrauch der Tastatur nicht vorgesehen. Zudem kann, da es sich nur um eine Simulation einer weiteren Taste handelt, nur eine der Darstellungen angenommen werden. Der Anwender kann aber bei Bedarf entgegen der virtuellen Tastatur auch eine Simulation einer Taste über die Kombination anderer Tasten bevorzugen, da sie der Software den gleichen Tastaturcode übermitteln wie die einfache Konstellation.

3.2 Intelligenzkonzept

„Eine prozedurale Wissensbasis besteht aus weitgehend festgelegten Folgen von Handlungsanweisungen. Prozeduren bewirken in bestimmten Situationen die entsprechenden Aktionen, wobei die richtige Situation durch die Aufrufstelle impliziert wird […].“ [Altenkrüger 92, S. 18]

Das Gebiet der künstlichen Intelligenz ist sehr umfangreich und umfasst eine Vielzahl unterschiedlicher Systeme. In hoch entwickelter Software, die über künstliche Intelligenz verfügt, besitzen auch Prozeduren und Vorgehensweisen die Fähigkeit sich anzupassen. In dieser Arbeit ist das Intelligenzkonzept eingeschränkt, da die intelligente Verarbeitung ausschließlich auf die Eingaben des Nutzers und hinterlegte Daten in einer Datenbank angewandt wird. Ein solches System ist durch feste Regeln bestimmt und wird in Fachkreisen auch regelbasiertes Expertensystem [Balzert 00, S. 311] genannt.

3.2.1 Ziel

Dem Anwender sollen Texte diktiert werden, die ihm ermöglichen, Zeichen, die falsch getippt wurden zu wiederholen, um Fehler schnell in den Griff zu bekommen und das fehlerfreie Tippen zu erlernen. Zu diesem Zweck sollen an die Tippfehler angepasste Texte während des Schreibtrainings als Diktat angeboten und aktualisiert werden. Einfach zusammengefasst heißt das: falsch getippte Schriftzeichen sollen öfter, richtig getippte seltener diktiert werden.

3.2.2 Tippfehlervarianten

Grundsätzlich handelt es sich um einen Tippfehler, wenn eine andere Taste als gewünscht betätigt wird. Meist kommen diese Fehler zustande durch überhastetes Schreiben und fehlende Kenntnisse im Zehnfingersystem. Schnell werden sie so zu Gewohnheitsfehlern, und nur durch gezieltes Training lassen sich diese Gewohnheiten korrigieren.

Folgende Tippfehlervarianten sind denkbar:

- Ein Schriftzeichen wird ausgelassen:

  Dies ist ein häufig vorkommender Fehler, der vor allem durch überhastetes Schreiben entsteht.

  Beispiel: „Schreibtainer“

- Ein Schriftzeichen wird zu viel eingeben:

  Dieser Fehler kommt häufig zustande, wenn zwei gleiche Schriftzeichen aufeinander folgen oder wenn eine Taste zu lang gedrückt wird.

  Beispiele: „Überalll“, „Schreibtrrainer“

- Die Groß- und Kleinschreibung wird falsch angewendet:

  Meist durch zu frühes Loslassen der Umschalttaste oder zu langes Halten der Umschalttaste entsteht diese Variante eines Tippfehlers.

  Beispiel: „schreibtrainer“, „SChreibtrainer“

- Schriftzeichen werden verdreht:

  Besonders häufig entstehen Tippfehler, bei denen zwei Schriftzeichen vertauscht eingegeben werden. Diese Variante wird schnell zu einem Gewohnheitsfehler.

  Beispiel: „Schriebtrainer“

- Es wird ein auf der Tastatur benachbartes Schriftzeichen eingegeben
  Besonders bei der Anwendung des Zehnfingersystems kann es passieren, dass statt der vorgesehenen Taste eine benachbarte Taste betätigt wird.
  Beispiel: „Scjreibtrainer“

Um die verschiedenen Tippfehlervarianten zu erkennen und mit Hilfe des Intelligenzkonzepts gezielt auf sie einzugehen, ist es nötig sehr viele Daten während des Schreibtrainings zu verwalten und mit Hilfe komplexer Algorithmen zu analysieren. Tippfehler müssten mit vorherigen und nachfolgenden Schriftzeichen im Diktat und mit benachbarten Tasten auf der Tastatur verknüpft werden, um sie nach einzelnen Fehlervarianten kategorisieren zu können. Die einzelnen Fehlervarianten müssten wiederum Schriftzeichen zugeordnet werden. Das Intelligenzkonzept müsste dann anhand der Analyse gezielt Diktattexte filtern, die auf bestimmte Fehlervarianten und Schriftzeichen zugeschnitten sind.

Dieser Vorgang ist sehr komplex und soll bei dem ersten Entwurf der Software im Rahmen dieser Diplomarbeit nicht beachtet werden. Da aber alle Tippfehler darauf beruhen, dass eine andere Taste als gewünscht betätigt wurde, können leicht zwei Vorgänge gespeichert werden:

Soll-Tippfehler: Eine bestimmte Taste sollte gedrückt werden

Ist-Tippfehler: Eine bestimmte Taste wurde fälschlicherweise gedrückt

Vor allem der Soll-Tippfehler spielt eine wichtige Rolle, um dem Anwender angepasste Texte zu diktieren (Schriftzeichen mit hohen Soll-Tippfehlern werden im Diktat öfter eingesetzt). Der Ist-Tippfehler kann ebenfalls für eine Textfilterung eingesetzt werden, soll aber im Rahmen der Diplomarbeit vorerst nur gespeichert werden, damit zu einem späteren Zeitpunkt das Intelligenzkonzept einfach erweitern werden kann.

3.2.3 Diskussion verschiedener Lösungsansätze

Um eine dynamische Erstellung der Diktate zu ermöglichen, werden im Folgenden fünf verschiedene Lösungsansätze diskutiert. Es wurde dabei auf die im Pflichtenheft festgelegte Verwendung von Lektionen geachtet, die den Anwender schrittweise mehr Tasten erlernen lassen.

Lösungsansatz 1:

Es sind alle Zeichen hinterlegt, die der Anwender im Laufe der Lektionen erlernen soll (Kleinbuchstaben, Großbuchstaben, Ziffern, Satzzeichen und Sonderzeichen). Dem Anwender werden im Schreibtraining ausschließlich Zeichenfolgen diktiert, die zwar auf die Bedürfnisse des Anwenders zugeschnitten sind, jedoch keinen Wortsinn ergeben.

Beispiel:

> Der Benutzer tippt die Schriftzeichen *d*, *f* und *r* häufig falsch.
>
> Resultierendes Diktat: *dfrdfrdfr*

Vorteile:

- Für jede Lektion lassen sich passende Zeichen anzeigen.
- Jeder Fehler bestimmt genau ein Zeichen, wodurch die intelligente Verarbeitung einfach zu realisieren wäre.
- Das System wäre höchst flexibel und schnell.
- Die Software muss vor der Auslieferung nicht aufwendig mit Worten und Sätzen ausgestattet werden.
- Es müssen keine Worte und Sätze während der Lektion generiert werden.

Nachteile:

- Reine Zeichenfolgen ergeben keinen Sinn und entsprechen nicht dem Vorkommen von Text in der Realität.

- Zeichenfolgen sind für den Anwender viel schwieriger zu tippen als Wörter und Sätze, weil sie schwerer zu erfassen sind.
- Der Anwender erlernt Zeichenkombinationen, die er in der Realität nicht braucht.
- Zeichenfolgen einzugeben, erscheint dem Anwender schnell langweilig.

Lösungsansatz 2:

In einer Datenbank werden ausschließlich Wörter hinterlegt. Diese werden dem Anwender durch Leerzeichen getrennt diktiert.

Beispiel:

Der Benutzer tippt die Schriftzeichen *d*, *f* und *r* häufig falsch.

Resultierendes Diktat: *darf froh doof fad*

Vorteile:

- Das System lässt sich durch eine umfangreiche Wörter-Datenbank umsetzen.
- Die intelligente Verarbeitung kann oft eingreifen und neue Worte an das Diktat anfügen, da die Zeichenzahl eines Wortes im Vergleich zur Zeichenzahl von Sätzen gering ist.

Nachteile:

- Reine Wortfolgen ergeben keinen Sinn und entsprechen nicht der Realität.
- Ziffern, Satzzeichen und Sonderzeichen sind nicht Bestandteil von Worten und lassen sich daher schwer integrieren.
- Aus den Tasten der ersten Lektionen lassen sich nur wenige Worte ableiten.

Lösungsansatz 3:

In einer Datenbank werden ausschließlich Wörter hinterlegt. Diese werden nur in den ersten Lektionen mit Leerzeichen getrennt diktiert und später zu ganzen Sätzen zusammengefügt.

Beispiel:

Der Benutzer tippt die Schriftzeichen *d*, *f* und *r* häufig falsch.

Resultierendes Diktat der ersten Lektionen: *froh er darf*

Resultierendes Diktat der höheren Lektionen: *Er darf froh sein.*

Vorteile:

- Das System ist flexibel und kann beliebige Satzvariationen erstellen.
- Das Diktat kann oft aktualisiert werden, und Sätze können durch Umstrukturierung angepasst werden.

Nachteile:

- Sehr komplexes und schwer realisierbares System, da sich Worte nur mit Hilfe von grammatischen Formen verknüpfen lassen.
- Gefahr, keine sinnvollen Ergebnisse zu erhalten.

Lösungsansatz 4;:

In einer Datenbank werden Wörter, Wortfolgen und Sätze hinterlegt. In den ersten Lektionen werden Worte, später ganze Sätze diktiert.

Beispiel:

Der Benutzer tippt die Schriftzeichen *d*, *f* und *r* häufig falsch.

Resultierendes Diktat der ersten Lektionen: *froh darf freunde*

Resultierendes Diktat der höheren Lektionen: *Die Freunde sind froh.*

Vorteile:

- Der Inhalt der Datenbank kann gefiltert und direkt eingesetzt werden.
- Dem Anwender kann ein abwechslungsreiches Diktat angeboten werden.

Nachteile:

- Die Länge der einzelnen Texte bestimmt den Zeitpunkt der Aktualisierung.
- Ganze Sätze können, aufgrund der vielen Schriftzeichen, schlechter auf die Fehler des Anwenders angepasst werden.

Lösungsansatz 5:

Der Schreibtrainer arbeitet mit statischen Lektionen, wie es bei den meisten auf dem Markt vorhanden Schreibtrainern der Fall ist. Das KI-System wird ausschließlich auf Zusatzlektionen angewandt, die nach Abschluss jeder Lektion durchgeführt werden und falsche getippte Wörter wiederholen.

Beispiel:

Statisches Diktat: *Die Freunde sind froh und daher gut gelaunt.*

Der Benutzer tippt die Schriftzeichen *d*, *f* und *r* häufig falsch.

Resultierendes Zusatzdiktat: *Freunde froh daher*

Vorteile:

- Es lassen sich Worte der vorangegangenen Lektionen, die vom Anwender falsch getippt wurden, gezielt wiederholen.

Nachteile:

- Zusatzlektionen decken nur einen Teil des Schreibtrainings ab.
- Texte der Standardlektionen lassen keine Dynamik zu und haben den Nachteil, nicht an die Eigenschaften des Anwenders angepasst zu werden.

Die Lösungsansätze zeigen, dass bei der Entwicklung eines Systems, das dynamisch Texte zur Verfügung stellen soll, viele Aspekte beachtet werden müssen. Lösungsansatz 4 kristallisiert sich dabei als realisierbares System heraus und ist für die dieser Diplomarbeit zugrunde liegenden Idee sehr geeignet. Die Ansätze 1 und 2 sind aufgrund unsinniger und somit unrealistischer Texte

ungeeignet. Ein für den Benutzer aus sinnvollen Texten zusammengesetztes Diktat stellt einen weitaus wichtigeren Aspekt dar als eine hohe Flexibilität im Bereich der intelligenten Verarbeitung.

Lösungsansatz 3 ist sehr vielversprechend und sollte auf jeden Fall für die weitere Entwicklung des Schreibtrainers in Betracht gezogen werden. Aufgrund der hohen Komplexität soll jedoch ein einfacheres System implementiert werden. So ist gewährleistet, dass die Implementierung das angestrebte Ziel auch erreicht, und dass das System nicht Gefahr läuft, ein unbrauchbares Ergebnis zu erzeugen.

Lösungsansatz 5 stellt nur eine Erweiterung von auf dem Markt vorhandenen Schreibtrainern dar. Da die Idee dieser Diplomarbeit auf einem vollständig dynamischen System basiert, ist dieser Lösungsansatz für die Umsetzung ebenfalls nicht geeignet. Intelligente Zusatzlektionen würden nur einen geringen Teil des Schreibtrainings abdecken. Bei Wiederholung einer Lektion wäre der Benutzer gezwungen, vor Beginn der Zusatzlektion wieder exakt die gleichen Texte einzugeben.

3.2.4 Kriterien für den Programmentwurf

Aus dem Lösungsansatz 4, der für das Intelligenzkonzept ausgewählt wurde, lassen sich Kriterien zusammenfassen und ableiten, die für den Entwurf der Softwarearchitektur entscheidend sind. Sie spielen beim Aufbau der Datenbank eine große Rolle und bestimmen die Vorgehensweise für die technische Verarbeitung des Schreibtrainings.

1. Es muss sichergestellt sein, dass jede Lektion nur bereits erlernte Zeichen beinhaltet.
2. Die ersten Lektionen lassen nur eine sehr geringe Anzahl an Zeichen für die Erstellung von Texten zu. Daher sollen in den ersten Lektionen

ausschließlich Wörter und Zeichenfolgen, in späteren Lektionen ganze Sätze diktiert werden.

3. Jeder zu einer Lektion gehörige Text sollte über möglichst viele in dieser Lektion zu erlernende Schriftzeichen verfügen.
4. Es sollen die Texte einer Lektion diktiert werden, die die am häufigsten falsch getippten Buchstaben enthalten. Tippfehler des Benutzers müssen dazu individuell für jedes Schriftzeichen gezählt und gespeichert werden.
5. Da bestimmte Zeichen im Sprachgebrauch häufiger vorkommen als andere, müssen die gespeicherten Tippfehler gewichtet werden. So wird auch gewährleistet, dass sich die Gewichtung richtig getippter Schriftzeichen „verbessert“ und bei jeder neuen Filterung in den Hintergrund rückt. Für die Gewichtung muss zusätzlich auch das Vorkommen jedes Schriftzeichens gezählt und gespeichert werden.
6. Es sollen möglichst oft neue Texte angefordert und diktiert werden, da so immer wieder auf den aktuellen Fehlerstatus des Anwenders eingegangen werden kann. Worte und Zeichenfolgen bilden dafür geeignete Voraussetzungen. Bei Sätzen soll die Satzlänge möglichst gering gehalten werden.

3.2.5 Definition der Lektionen

Abbildung 3.7 zeigt die allgemeine Festlegung, welcher Finger für welche Taste beim Zehnfingersystem zu verwenden ist (siehe auch Kapitel 1.1 – geschichtlicher Hintergrund).

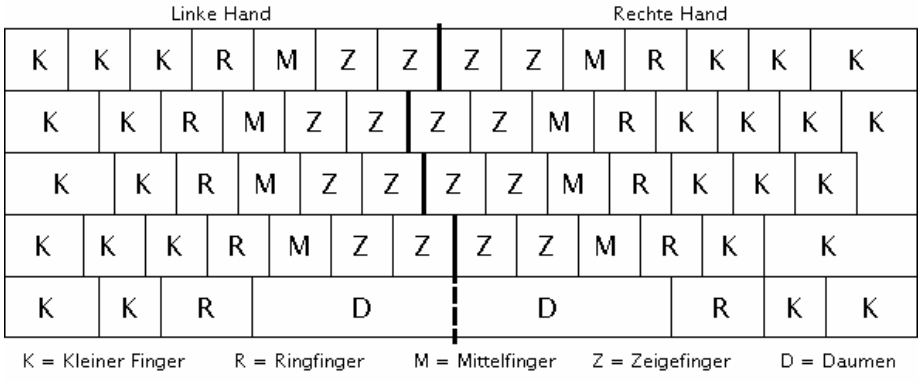


Abb. 3.7: Finger-Tasten-Zuordnung der Tastatur

Die einzelnen Trainingslektionen sollen über die nacheinander zu erlernenden Tasten definiert werden. Dazu ist es sinnvoll, die Tastatur in aufeinander aufbauende Tastenbereiche aufzuteilen, die dem Benutzer ermöglichen, sich das Zehnfingersystem Schritt für Schritt anzueignen.

Es wird von der Grundstellung (siehe Abbildung 3.8) ausgegangen, die der Benutzer von Beginn an mit den Fingern auf der Tastatur einzunehmen und in die er nach jedem Tastvorgang zurückzukehren hat.

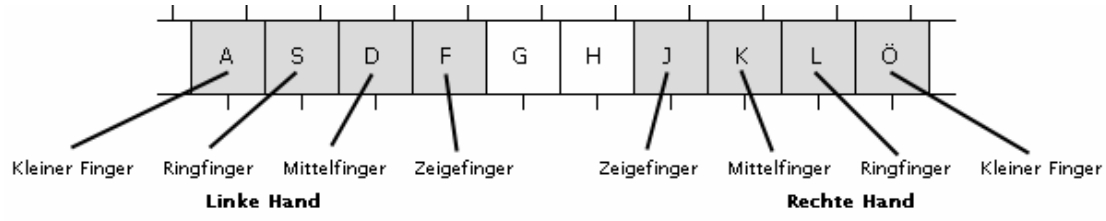


Abb. 3.8: Grundstellung auf der Tastatur

Diese Tasten haben nicht nur durch die Grundstellung eine besondere Wichtigkeit. Sie ermöglichen außerdem einem Anfänger, das Zehnfingersystem zu erlernen, vorerst ohne Tastwege zurücklegen zu müssen. Die Tasten der Grundstellung bilden daher in Form von Kleinbuchstaben einen Teil der

Schriftzeichen der ersten Lektion. Da die Leertaste für die Trennung von Worten und die Eingabetaste für Zeilenumbrüche unabdingbar sind, werden diese ebenfalls in Lektion 1 aufgenommen und schließen sie damit ab.

Anschließend sollen sinnvolle Tastwege und die Möglichkeiten zur Erzeugung dynamischer Texte die Reihenfolge der zu erlernenden Tasten bestimmen. Auf den ersten Blick scheint es schlüssig, linke und rechte Hand exakt die gleichen Tastwege erlernen zu lassen. Bedenkt man jedoch, dass manche Schriftzeichen im Schrift- und Sprachgebrauch häufiger vorkommen als andere, dann ergibt sich daraus, dass diese früher erlernt und trainiert werden sollten als selten verwendete Buchstaben. Zudem bietet dies eindeutig mehr Möglichkeiten zur Erzeugung von Worten und Sätzen für die Diktate.

Das folgende Beispiel veranschaulicht die Problematik:

Während der rechte Zeigefinger den Buchstaben *O* auf der obersten Buchstabenreihe erlernt, müsste, bei gleichen Tastwegen, der linke Zeigefinger entsprechend den Buchstaben *W* annehmen. Da das *O* aber weitaus öfter Verwendung in Texten findet als der Buchstabe *W*, ist es nicht sinnvoll, diese Buchstaben in einer Lektion zu vereinen. Zusätzlich sollte das *O* recht früh, also in einer der ersten Lektionen, erlernt werden. Würde nun der Buchstabe *W* ebenfalls in diese Lektion übernommen, schränkte dies die Auswahl der möglichen Wörter stark ein, da in den ersten Lektionen nur wenige Buchstaben zur Erzeugung der Diktate zur Verfügung stehen.

Um diese Problematik zu vermeiden, wurde versucht, eine Aufteilung der Tasten und damit Lektionen zu finden, die möglichst nach der Häufigkeit eines Buchstabens absteigend erfolgt. Eine hierfür sehr nützliche Informationsquelle stellte die Internetseite des Tastaturlayouts *RISTOME* [WV Ristome] (siehe auch geschichtlicher Hintergrund Kapitel 1.2) zur Verfügung. Unter dem Menüpunkt

Tests & Auswertungen ist eine Häufigkeitsanalyse einzelner Zeichen zu finden. Grundlage bildeten ca. 2,5 Millionen Zeichen unter anderem aus Zeitungsberichten, Erfahrungsberichten und *Chat*-Protokollen in deutscher Sprache. In einer Liste finden sich die meistverwendeten Schriftzeichen und deren Häufigkeit in Prozent.

Die Tabelle auf der nächsten Seite zeigt die Häufigkeitsanalyse.

Zeichen	Häufigkeit
	14,09%
e	13,03%
n	7,93%
i	6,40%
r	5,68%
t	4,87%
s	4,58%
a	4,24%
d	3,51%
h	3,47%
u	2,93%
l	2,81%
c	2,35%
g	2,16%
o	2,07%
m	1,94%
Sonstige	1,46%
b	1,42%
f	1,08%
w	1,05%
,	0,98%
k	0,94%
.	0,87%
z	0,85%
v	0,57%
p	0,51%
ü	0,48%
S	0,47%
ä	0,45%
A	0,41%
D	0,38%
B	0,31%
M	0,29%
E	0,28%
G	0,28%
-	0,23%

Zeichen	Häufigkeit
F	0,22%
K	0,22%
W	0,22%
P	0,21%
ö	0,21%
ß	0,21%
I	0,20%
R	0,18%
H	0,17%
V	0,17%
L	0,16%
T	0,16%
0	0,16%
N	0,13%
"	0,13%
1	0,13%
U	0,12%
:	0,12%
j	0,11%
y	0,11%
Z	0,11%
C	0,10%
J	0,09%
(	0,09%
)	0,09%
2	0,09%
O	0,08%
9	0,07%
x	0,06%
3	0,06%
4	0,05%
5	0,05%
?	0,05%
6	0,04%
!	0,03%
7	0,03%

Zeichen	Häufigkeit
8	0,03%
'	0,02%
/	0,02%
;	0,02%
q	0,01%
Q	0,01%
X	0,01%
%	0,01%
*	0,01%
[	0,01%
]	0,01%
_	0,01%
Ä	0,01%
Ü	0,01%
Y	0,00%
#	0,00%
$	0,00%
&	0,00%
+	0,00%
<	0,00%
=	0,00%
>	0,00%
@	0,00%
\	0,00%
^	0,00%
{	0,00%
|	0,00%
}	0,00%
~	0,00%
€	0,00%
§	0,00%
°	0,00%
²	0,00%
³	0,00%
Ö	0,00%

Tabelle 3.1: Häufigkeitsanalyse (Quelle: *RISTOME* [WV Ristome])

Basierend auf dieser Liste und den bereits festgelegten Zeichen der ersten Lektion wurden nun die restlichen Lektionen definiert. Jeweils zwei neue Tasten und alle Tasten der vorherigen Lektion(en) bilden eine eigene Lektion.

Um das Intelligenzkonzept gut auf die Lektionen anwenden zu können, wurden minimale Modifizierungen an der Reihenfolge der Zeichen und der Anzahl neuer Tasten vorgenommen. So wird der *Punkt* früher als er in der Häufigkeitsanalyse platziert ist erlernt, damit ganze Sätze auch früher diktiert werden können. Zudem wird das Erlernen der Umschalttasten (engl. Shift) vorgezogen und in eine eigene Lektion übernommen. In dieser wird dann die Großschreibung bereits erlernter Zeichen angewendet. Alle Zeichen der folgenden Lektionen beinhalten dann stets Klein- und Großschreibung.

Als sinnvoll hat sich eine Aufteilung der Tastatur in 16 Lektionen erwiesen, die folgendermaßen aufgebaut sind:

Lektion	Zu erlernende Zeichen (getrennt durch Kommas dargestellt)
1	a,s,d,f,j,k,l,ö,*Leerzeichen*,*Zeilenumbruch* (Grundstellung und notwendige Zeichen)
2	e,n
3	r,i
4	t,h
5	c,u
6	A,S,D,F,J,K,L,Ö,E,N,R,I,T,H,C,U (Großschreibung der vorherigen Lektionen)
7	g,G,.,:
8	o,O,m,M
9	b,B,w,W
10	z,Z,*Komma*,;
11	v,V,p,P
12	ü,Ü,ä,Ä
13	ß,?,q,Q
14	y,Y,x,X
15	1,!,2,“,3,§,4,$,5,%,6,&,7,/,8,(,9,),0,= (Ziffern und damit verbundene Zeichen)
16	^,°,´,`,+,*,#,’,<,>,²,³,{,[,],},\,~,@,€,|,µ

Tabelle 3.2: Definition der Lektionen

In Lektion 15 und 16 werden mehr als zwei Tasten in die Lektion übernommen, da die zugehörigen Zeichen sehr selten vorkommen und sonst die Auswahl der Texte zu sehr eingeschränkt wäre.

Eine Übersicht, bei der gefärbte Tasten in 16 Tastaturgrafiken die Aufteilung der Lektionen veranschaulichen, findet sich im Anhang A. Um den Zeilenumbruch, der ab der ersten Lektion Bestandteil des Diktats ist, dem Anwender als Zeichen sichtbar zu machen und so zu veranschaulichen, dass nun die Eingabetaste gedrückt werden soll, wird ein eigenes Zeichen eingeführt. Hierfür eignet sich das im Normalfall sehr selten verwendete und von *Microsoft Word* zur Darstellung des Zeilenumbruchs verwendete Zeichen ¶.

Um Anwendern, die das Zehnfingerschreiben bereits beherrschen, aber sich in Punkto Fehlerquote und Schreibgeschwindigkeit verbessern wollen eine Lektion

anzubieten, die nicht auf bestimmte Schriftzeichen abgeschnitten ist, wird zusätzlich noch eine Lektion 17 vorgesehen. Sie diktiert alle in den Lektionen 1 bis 16 verwendete Sätze.

Da sich durch Testläufe und den Einsatz der Software herausstellen könnte, dass eine andere Definition der Lektionen besser geeignet ist, wird die Definition der Lektionen in einer Tabelle hinterlegt und kann so gegebenenfalls aktualisiert werden.

3.3 Datenbank

Für das Produkt wurde die relationale Datenbank SQLite ausgewählt. In den nachfolgenden Kapiteln sollen die Tabellen (Relationen) der Datenbank schrittweise, mit Beachtung auf die Darstellung der virtuellen Tastatur und die im Kapitel 3.2.3 festgelegten Kriterien für das Intelligenzkonzept entworfen und aufgebaut werden. Für die Realisierung des Intelligenzkonzepts spielt der Entwurf der Datenbankstruktur eine besonders große Rolle, da in der Datenbank alle hierfür relevanten Informationen hinterlegt werden.

3.3.1 Die Tabellen *key_layouts*, *character_list_win* und *character_list_mac*

Um die Darstellung einer virtuellen Tastatur zu ermöglichen, die für jedes zu tippende Schriftzeichen die entsprechenden Tasten farbig darstellt, müssen zu jeder Zeit die dafür nötigen Informationen aus der Datenbank abrufbar sein. Zu diesem Zweck werden die Tabellen *key_layouts*, *character_list_win* und *character_list_mac* eingeführt.

Wie in Kapitel 3.1.2 festgelegt, müssen folgende Faktoren berücksichtigt werden:

1. Aufteilung der Tastatur in 61 Tasten
2. Tastaturlayouts für Windows und Macintosh
3. Zuweisung der Tasten zu entsprechenden Schriftzeichen
4. Möglichkeit, zwei Tasten für ein Schriftzeichen anzugeben

Um Faktor 1 zu realisieren, wird jeder Taste ein fester Wert zugeordnet, wie in Abbildung 3.9 dargestellt. Anschließend werden die sich daraus ergebenden 61 Tasten-Werte in der Tabelle *key_layouts* hinterlegt.

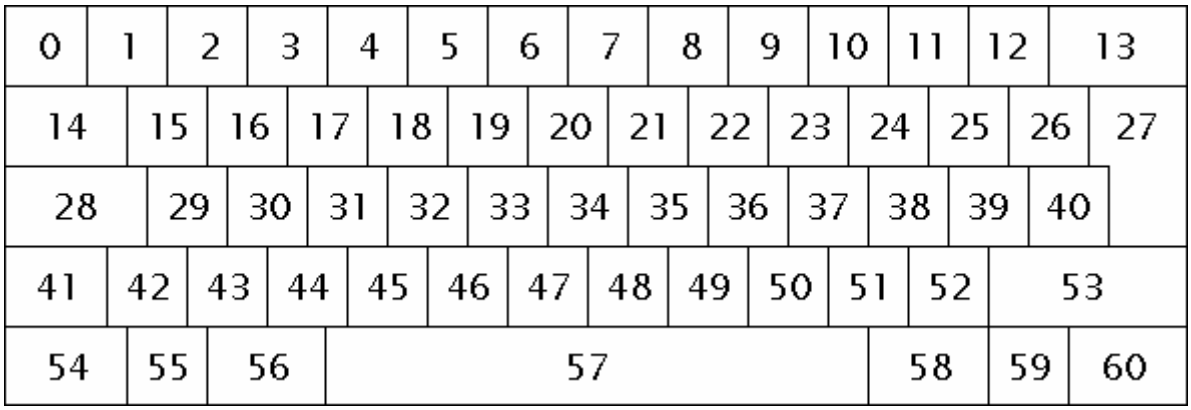


Abb. 3.9: Nummerierung der Tasten

Für die Umsetzung von Faktor 2 wird für jedes Layout eine eigene Tabelle verwendet, im vorliegenden Fall zwei exakt gleich strukturierte Tabellen *character_list_win* und *character_list_mac*. Sie werden redundant mit den Unicode-Werten aller Schriftzeichen bestückt, die für das Schreibtraining möglich sind.

Die Trennung der Layouts in Tabellen ist darin begründet, dass immer nur jeweils eines der Layouts verwendet wird und der Zugriff auf die entsprechende Tabelle häufig erfolgt. Würden die Layouts gemeinsam in einer Tabelle hinterlegt, verdoppelte sich die Anzahl der Datensätze unnötig. Dies würde eine Abfrage auf die Tabelle verlangsamen, insbesondere, falls zu einem späteren Entwicklungszeitpunkt noch weitere Tastaturlayouts eingeführt werden. Durch die

Aufteilung in Tabellen können die Layouts völlig getrennt voneinander betrachtet werden, und durch Anlegen neuer Tabellen kann das System um Layouts erweitert werden.

Erst jetzt, bei der folgenden Zuweisung der Tasten-Werte zu den Schriftzeichen wird ein Unterschied des Inhalts der Tabellen *character_list_win* und *character_list_mac* ersichtlich (Faktor 3 und 4). Da beispielsweise das Schriftzeichen *@* auf der *Windows*-Tastatur von einer andere Taste getragen wird als auf der *Macintosh*-Tastatur, muss diesem Zeichen in jeder Tabelle eine andere Taste zugewiesen werden. Das gleiche gilt auch für andere Zeichen und besonders die Modifizierungstasten.
Das Datenbankdiagramm in Abbildung 3.10 zeigt anhand einer Entity-Relationship-Struktur die Abhängigkeit zwischen den zwei Tabellen und der Zuordnung der Tasten in der Tabelle *key_layouts*.

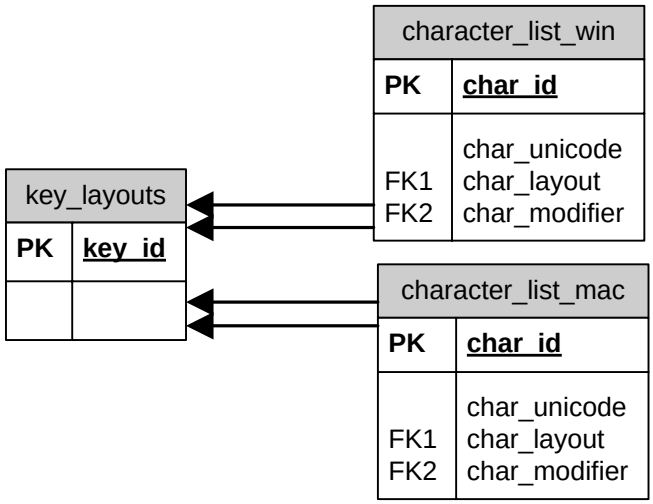


Abb. 3.10: Entity-Relationship-Diagramm des Tastaturlayouts

Das Feld *char_layout* der Tabellen *character_list_win* und *character_list_mac* verweist auf die entsprechende Taste aus der Tabelle *key_layouts*. Wird eine zweite Taste (Modifizierungstaste) für ein Schriftzeichen benötigt, verweist das Feld *char_modifier* auf einen weiteren Tastencode, andernfalls enthält sie die

Zahl 0. Auf diese Weise können jedem Schriftzeichen bis zu zwei Tasten zugeordnet werden.

Abbildung 3.11 veranschaulicht am Beispiel der Schriftzeichen *a* (Unicode Dezimalwert 97) und *A* (Unicode Dezimalwert 65) auf der Windows-Tastatur die Beziehung der Tabellen *key_layouts* und *character_list_win*.

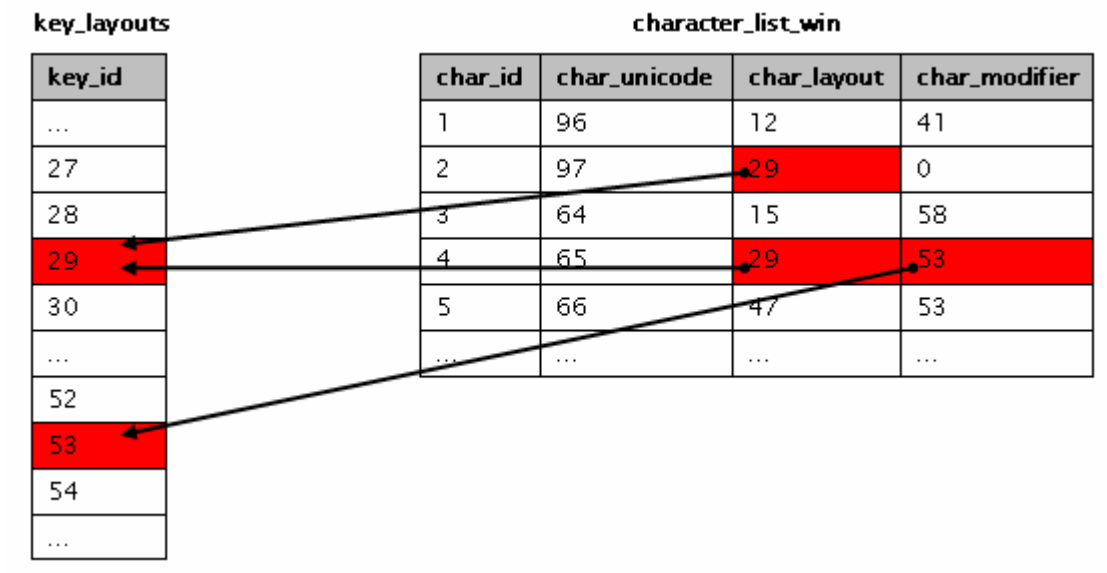


Abb. 3.11: Veranschaulichung der Beziehung der Tabellen *key_layouts* und *character_list_win*

3.3.2 Die Tabellen *lesson_list* und *lesson_content*

In Kapitel 3.1.3 wurden alle Lektionen definiert. Die sich daraus ergebende Liste soll in einer Tabelle gespeichert werden. So wird ermöglicht, die Definition auf einfache Weise zu ändern, zudem können andere Tabellen über eine Beziehung auf bestimmte Lektionen verweisen. Die Tabelle wird auch verwendet, um passende Bezeichnungen der Lektionen für den Anwender zu hinterlegen.

Zu diesem Zweck wird die Tabelle *lesson_list* eingeführt und wie folgt strukturiert:

Feldname	Beschreibung
lesson_id	Kennung der Lektion (Primärschlüssel)
lesson_name	Bezeichnung der Lektion
lesson_description	Beschreibung der Lektion

Tabelle 3.3: Struktur der Tabelle *lesson_list*

Über das Feld *lesson_id* kann die jeweilige Lektion eindeutig identifiziert werden. Die Felder *lesson_name* und *lesson_description* führen Zusatzinformationen, um die Lektion dem Anwender zu beschreiben.

Um ein dynamisches Diktat zur Laufzeit zu erzeugen, müssen viele Texte für die Verwendung in Lektionen in der Datenbank bereitgestellt werden und schnell abrufbar sein. Zu diesem Zweck wird die Tabelle *lesson_content* in der Datenbank eingeführt.
Der Entwurf des Intelligenzkonzepts aus Kapitel 3.2 zeigt auf, dass jede Lektion nur bisher erlernte Schriftzeichen beinhalten darf. Dies zur Laufzeit zu überprüfen, wäre sehr zeitaufwendig, da alle Schriftzeichen aller hinterlegten Texte nach ihrer Tauglichkeit für die jeweilige Lektion analysiert werden müssen. Aus diesem Grund werden die Texte bereits mit zugewiesener Lektion in die Datenbank übernommen. Die Analyse und Zuordnung der Texte soll also bereits beim Erstellen der Texte durchgeführt werden. Dies wird in Kapitel 3.6 genauer beschrieben.
Es ergeben sich folgende Felder für die Tabelle *lesson_content*:

Feldname	Beschreibung
content_id	Kennung des Textes (Primärschlüssel)
content_text	Fester Text (Wort, Wortfolge oder Satz)
content_lesson	Nummer der zugewiesenen Lektion

Tabelle 3.4: Struktur der Tabelle *lesson_content*

Abbildung 3.12 zeigt anhand einer Entity-Relationship-Struktur die Abhängigkeit zwischen den Tabellen *lesson_content* und *lesson_list*.

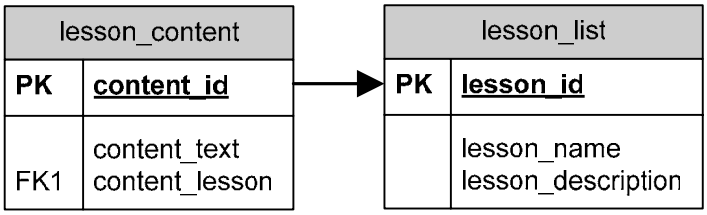


Abb. 3.12: Entity-Relationship-Diagramm der Lektionen

Jedem Text in der Tabelle *lesson_content* wird genau eine Lektion aus der Tabelle *lesson_list* zugeordnet.

3.3.3 Die Tabellen *user_lesson_list* und *user_chars*

Laut Pflichtenheft soll die Anwendung alle Ergebnisse abgeschlossener Lektionen des Anwenders speichern. Dazu wird die Tabelle *user_lesson_list* eingeführt. Sie wird folgendermaßen strukturiert:

Feldname	Beschreibung
user_lesson_lesson	Kennung der Lektion
user_lesson_timelen	Dauer der Lektion in Sekunden
user_lesson_tokenlen	Anzahl der diktierten Zeichen insgesamt
user_lesson_strokesnum	Anzahl der Anschläge insgesamt
user_lesson_errornum	Anzahl der Tippfehler insgesamt
user_lesson_timestamp	Startzeitpunkt der Lektion (Datum und Uhrzeit)

Tabelle 3.5: Struktur der Tabelle *user_lesson_list*

Nach jeder abgeschlossenen Trainingslektion werden die Ergebnisse in einem neuen Datensatz gespeichert. Das Feld *user_lesson_lesson* stellt keinen Primärschlüssel dar, da eine Lektion mehr als einmal durchgeführt werden kann. Die Ergebnisse werden verwendet, um dem Anwender eine Ergebnisübersicht darzustellen und um eine Bewertung jeder Lektion durchzuführen.

Für die Realisierung des Intelligenzkonzepts ist es zudem unerlässlich, jedes während aller Trainingslektionen verwendete Schriftzeichen mit aktuellen Informationen aus dem Schreibtraining zu versehen und in einer Tabelle zu speichern. Dieser Vorgang erfüllt auch das Kriterium Schriftzeichendaten des Pflichtenhefts.

Aus der Tabelle müssen sich für das Intelligenzkonzept jederzeit diejenigen Schriftzeichen filtern lassen, die vom Anwender am häufigsten falsch geschrieben wurden. Würde man dazu die Tabelle nur nach Tippfehlern pro Schriftzeichen aufsteigend sortieren, erhielte man verfälschte Informationen, da bestimmte Zeichen häufiger vorkommen als andere (vgl. Kriterium 5 in Kapitel 3.2.4). Deshalb müssen die Tippfehler des Benutzers gewichtet werden. Dies wird über eine Gewichtung in der Form „*Anzahl der Tippfehler eines Schriftzeichens* geteilt durch die *Anzahl des Schriftzeichens im Diktat insgesamt*" realisiert. So wird sichergestellt, dass jedes Zeichen gleichwertig behandelt wird. Eine absteigende Sortierung nach der Gewichtung bringt zuerst die (im Verhältnis) am häufigsten falsch getippten Schriftzeichen hervor.

Zu diesem Zweck wird die Tabelle *user_chars* eingeführt und durch folgende Felder wiedergegeben:

Feldname	Beschreibung
user_char_unicode	Unicode-Wert des Schriftzeichens (Primary Key)
user_char_target_errornum	Tippfehler: Zeichen sollte getippt werden
user_char_mistake_errornum	Tippfehler: Zeichen wurde fälschlicherweise getippt
user_char_occur_num	Vorkommen des Zeichens insgesamt (Anzahl)

Tabelle 3.6: Struktur der Tabelle *user_chars*

Das Feld *user_char_mistake_num* wird, obwohl es vorerst keine Verwendung findet, bereits vorgesehen, um die Software zu einem späteren Zeitpunkt erweitern zu können (vgl. Kapitel 3.2.2).

Jedes Mal, wenn ein Schriftzeichen diktiert oder ein Tippfehler des Anwenders erkannt wird, werden die Daten in der Tabelle entsprechend aktualisiert. Anhand der Tabellenfelder lautet die Gewichtungsformel dann:

$$\text{Gewichtung des Schriftzeichens} = \frac{user\_char\_target\_errornum}{user\_char\_occur\_num}$$

3.3.4 Die Tabellen *lesson_analysis* und *lesson_chars*

Durch die im vorherigen Kapitel eingeführte Tabelle *user_chars* sind alle Tippfehlerinformationen des Anwenders vorhanden, die für die Erzeugung des dynamischen Diktats erforderlich sind. Die Texte der Lektionen (Tabelle *lesson_content*) lassen sich zudem über die Tabelle *lesson_list* (vgl. Kapitel 3.3.2) auf die aktuell zu trainierende Lektion einschränken.

Ein weiteres Kriterium für die Realisierung des Intelligenzkonzepts stellt die Notwendigkeit dar, die hinterlegten Texte in der Tabelle *lesson_content* zur Laufzeit nach den Tippfehlern des Benutzers aus Tabelle *user_chars* zu filtern. Hierfür ist es nötig, alle Schriftzeichen jedes hinterlegten Textes zu analysieren, indem die Anzahl jedes vorkommenden Schriftzeichens gezählt wird. Diese Analyse würde während des Schreibtrainings sehr viel Zeit in Anspruch nehmen, vor allem bei einer großen Anzahl an hinterlegten Texten. Daher wird sie bereits beim Import der Daten vorgenommen und in einer weiteren Tabelle *lesson_analysis* abgelegt. Diese Tabelle soll die Zählungen der Schriftzeichen beinhalten und so eine schnelle Abfrage der passenden Texte ermöglichen.

Um die Analysetabelle zu generieren, wird eine Hilfstabelle *lesson_chars* vorgesehen. Sie besteht aus nur einem Feld und beinhaltet alle Unicode-Werte der Schriftzeichen, die für Tippfehler überhaupt in Betracht gezogen werden. Struktur der Tabelle *lesson_chars*:

Feldname	Beschreibung
char_unicode	Unicode-Wert des Schriftzeichens (Primärschlüssel)

Tabelle 3.7: Struktur der Tabelle *lesson_chars*

Bei der Analyse der Texte wird dann anhand dieser Schriftzeichen die Tabelle *lesson_analysis* erzeugt. Zusätzlich zu einem auf die Kennung des Textes verweisendes Feld wird die Tabelle für jeden Eintrag der Tabelle *lesson_chars* ein eigenes Feld in der Form analysis_char_<lesson_chars.char_unicode> bekommen. <lesson_chars.char_unicode> steht dabei für jeden Unicode-Wert aus der Tabelle *char_unicode*. Der nachfolgende Auszug der Tabellenstruktur veranschaulicht die Erstellung der Felder:

Feldname	Beschreibung
analysis_content	Kennung des Textes (Primärschlüssel)
…	…
analysis_char_65	Anzahl Schriftzeichen „A“ (Unicode-Dezimalwert 65)
analysis_char_66	Anzahl Schriftzeichen „B“ (Unicode-Dezimalwert 66)
analysis_char_67	Anzahl Schriftzeichen „C“ (Unicode-Dezimalwert 67)
…	…

Tabelle 3.8: Struktur der Tabelle *lesson_analysis*

Nach Erzeugung der Tabellenstruktur kann jeder Text der Tabelle *lesson_content* durchlaufen, sowie die Anzahl jedes vorkommenden Schriftzeichens gezählt und im passenden Feld der Tabelle *lesson_analysis* gespeichert werden.

Folgendes Beispiel demonstriert die Erzeugung der Tabelle *lesson_analysis* anhand der Buchstaben *d*, *e* und *f*, sowie die Bestückung (Zählung und Speicherung) der Datensätze für die Worte *feld*, *der* und *gelder*:

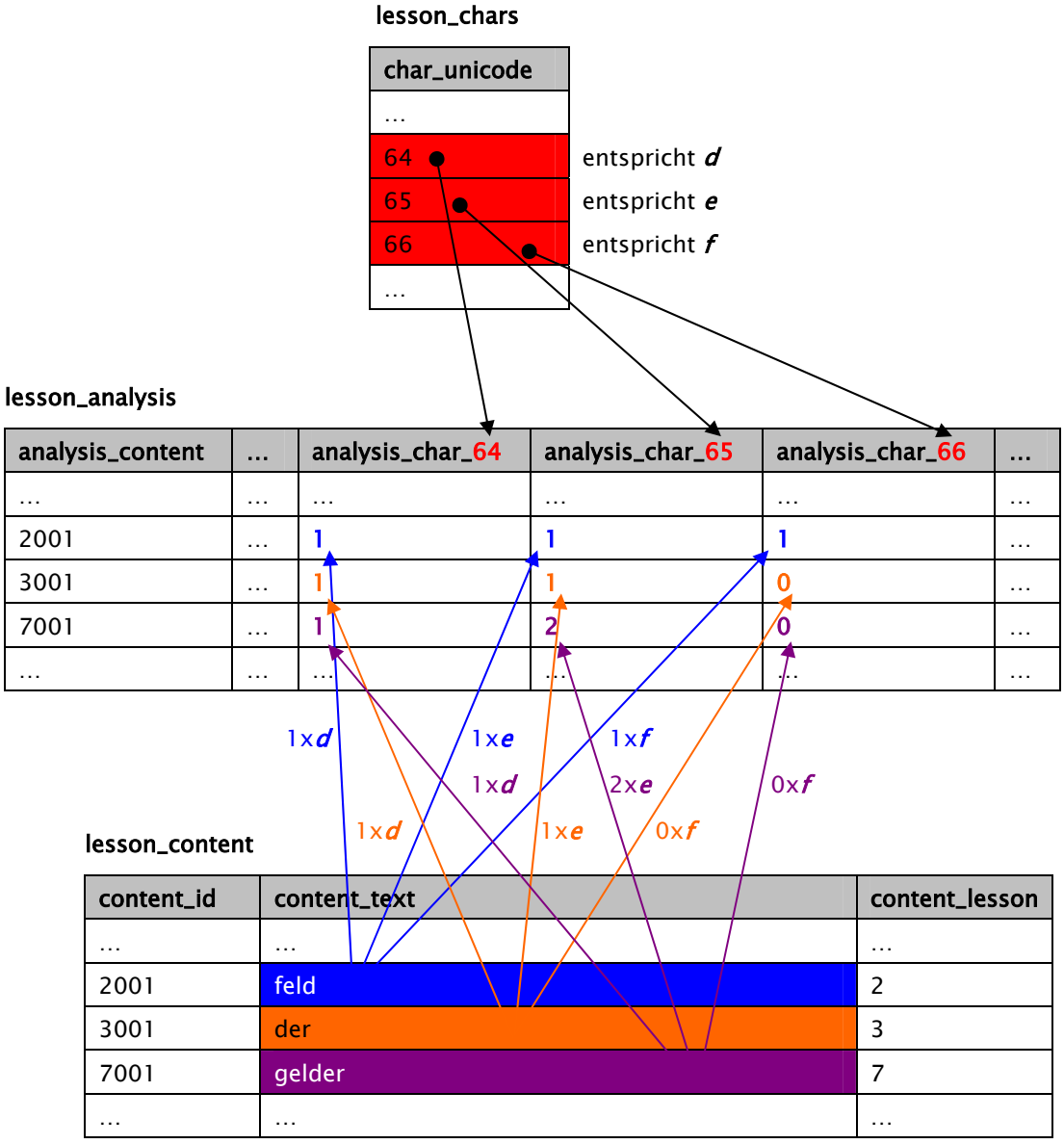


Abb. 3.13: Beispiel zur Generierung der Tabelle *lesson_analysis*

Abbildung 3.14 zeigt anhand einer Entity-Relationship-Struktur die Abhängigkeit zwischen den Tabellen *lesson_content*, *lesson_list* und *lesson_analysis*.

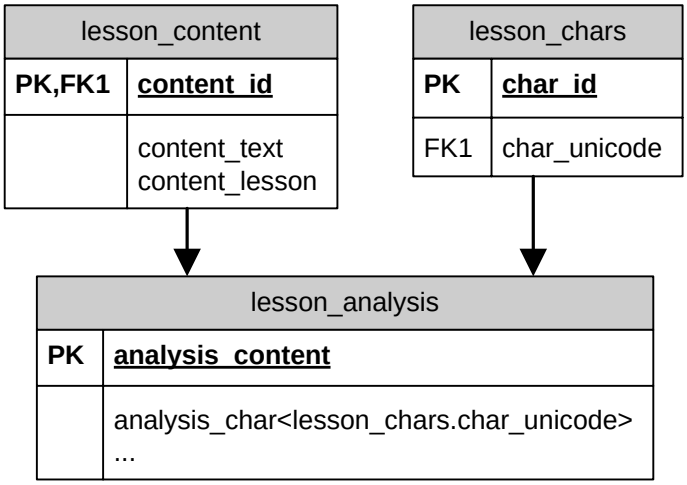


Abb. 3.14: Entity-Relationship-Diagramm der Textanalyse

Mit Hilfe der im Kapitel 3.3.3 erläuterten Tabelle *user_chars* und der damit verbundenen gewichteten Auswertung der Tippfehler lassen sich nun gezielt Datensätze aus der Tabelle *lesson_analysis* filtern, die wiederum auf den passenden Text in der Tabelle *lesson_content* verweisen.

Eine Erweiterung der oben aufgeführten Entity-Relationship-Struktur um die Tabelle *user_chars* verdeutlicht diesen Zusammenhang:

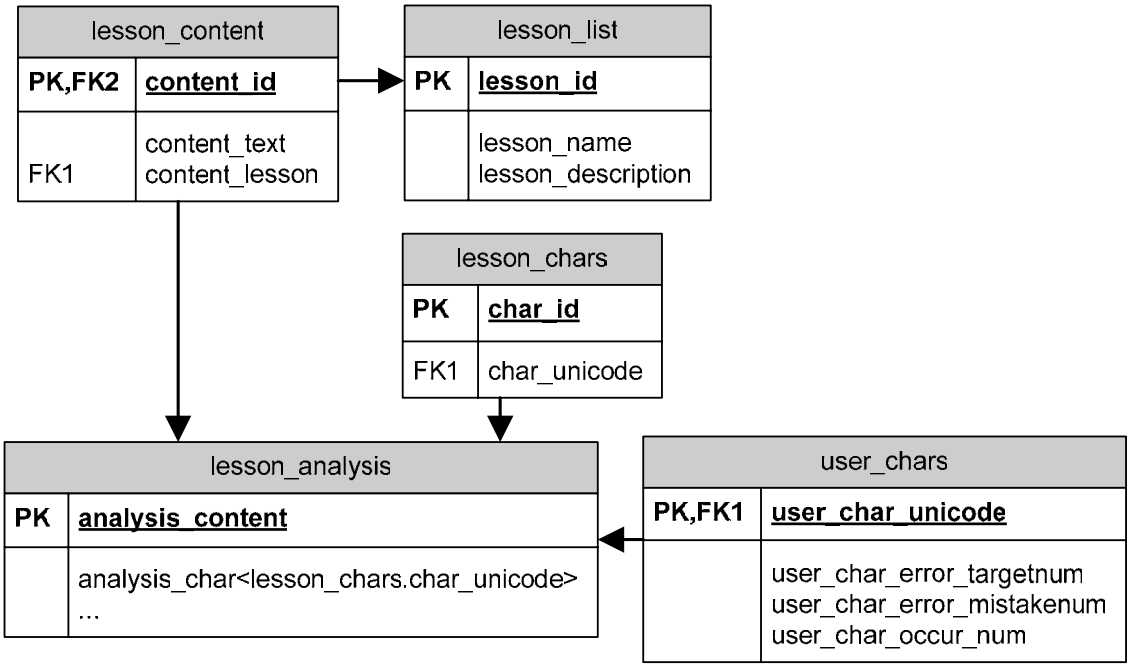


Abb. 3.15: Erweitertes Entity-Relationship-Diagramm der Textanalyse

3.3.5 Indexerstellung

Einige Datenbankabfragen müssen während des Schreibtrainings stattfinden, insbesondere um aktuelle Werte des Trainings zu speichern und um neue Texte für das Diktat anzufordern. Diese Abfragen sollen schnell Ergebnisse liefern, um den Ablauf des Schreibtrainings nicht zu verzögern. Daher wird für jedes Feld, das für Bedingungen in den relevanten Abfragen herangezogen wird, ein eigener Index erstellt.

Wird ein solches indiziertes Feld für die Suche herangezogen, wendet das Datenbanksystem statt einer sequentiellen Suche eine Suche über den Index an. Ein Datenbankindex besteht aus einer Baumstruktur aus Zeigern und verbessert das Zeitverhalten einer Abfrage erheblich.

3.3.6 Datenbankzugriff

Für die Zugriffe auf die Datenbank werden Befehle in der Abfragesprache *SQL* (Structured Query Language) verwendet. Alle Methoden für *SQL*-Abfragen werden in eigene Klassen ausgelagert, um eine gesonderte Schicht für die Datenhaltung zur Verfügung zu stellen (siehe auch Kapitel 3.5.1). Die Namen dieser Klassen werden mit einer zur Benutzungsoberfläche passenden Bezeichnung und der Erweiterung *sql* versehen. Im Klassendiagramm (vgl. Anhang B) lassen sich die Klassen der Datenhaltung auf diese Weise schnell identifizieren.

3.4 Gestaltung der Benutzungsoberfläche

„Die Kunst der Software-Ergonomen besteht darin, Dialogelemente so zu kombinieren, dass sie optimal auf den Nutzungskontext der Benutzerzielgruppe zugeschnitten sind“ [Balzert 00, S. 516]

Der Schreibtrainer stellt eine sehr einfach zu bedienende und klar strukturierte Oberfläche zur Verfügung, um den Ansprüchen aller in Kapitel 2.1.2.2 aufgeführten Zielgruppen gerecht zu werden. Eine im Vergleich zu anderen Software-Produkten verhältnismäßig geringe Anzahl an Funktionen wird ansprechend verteilt und sorgt für Übersichtlichkeit und somit für Klarheit beim Anwender.

Bei der Gestaltung wurde die Verwendung von drei Klassen aus der Klassenbibliothek *Qt* dem Entwurf und der Implementierung des Programmkonzepts vorweggenommen. Sie waren entscheidend für die Wahl der Fenster und die Aufteilung in einzelne Elemente.

3.4.1 Anwendungsfenster

Das Anwendungsfenster stellt das Hauptfenster der Applikation dar. Es erscheint nach dem Aufruf der Anwendung. Die Klassenbibliothek *Qt* bietet eine Vielzahl unterschiedlicher Fenstertypen. Für die Realisierung des *Anwendungsfensters* wird die Klasse *QMainWindow* vorgesehen, da sie bereits über eine Menüleiste verfügt und als Primärdialog hervorragend geeignet ist (siehe Abbildung 3.16).

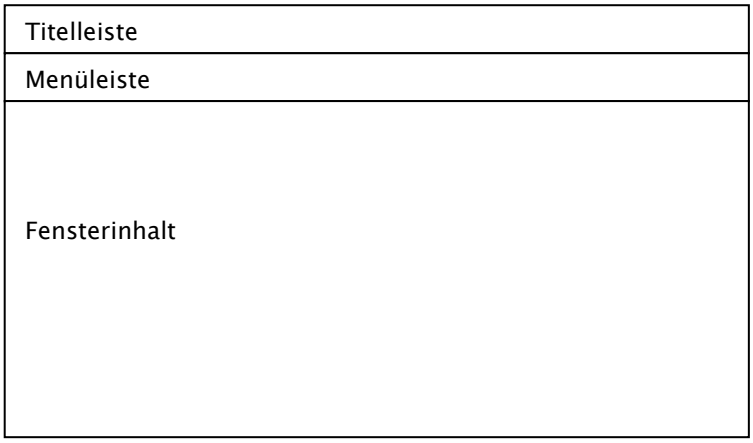


Abb. 3.16: Struktur des *Anwendungfensters*

Die im Pflichtenheft festgelegten Darstellungen *Einstellungen*, *Schreibtraining* und *Ergebnisse* (Kapitel 2.1.4) werden über Unterklassen der von *Qt* zur Verfügung gestellten Klasse *QWidget* realisiert und abwechselnd im *Anwendungsfenster* platziert. Sie werden im weiteren Verlauf Widget genannt, um ihrer Funktion gerecht zu werden.

Da immer nur jeweils ein Widget angezeigt wird, erscheinen die Darstellungen dem Benutzer wie eigene Fenster.

Folgende Abbildung veranschaulicht die Funktion des *Anwendungsfensters*:

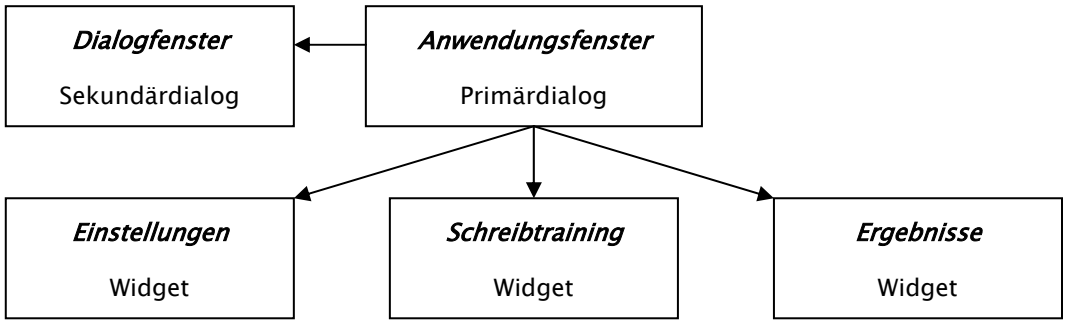


Abb. 3.17: Fenster-Struktur der Software

Nach dem Start der Software, also dem Aufruf des Anwendungsfensters, wird die Menüleiste angezeigt und das Widget *Einstellungen* abgebildet.

3.4.2 Widget *Einstellungen*

Das Widget *Einstellungen* ermöglicht dem Benutzer, eine Lektion auszuwählen und die dafür nötigen Einstellungen vorzunehmen. Eine Menüleiste stellt die Funktionen bereit, um Zusatzfenster (siehe Kapitel 3.4.1) aufzurufen. Zudem kann über das Menü das Auswertungsfenster aufgerufen werden, um aktuelle Ergebnisse auch ohne Durchführen des Schreibtrainings einzusehen. Eine Buttonleiste verfügt über Schaltflächen (Buttons) zum Starten des Trainings und Beenden der Software. Abbildung 3.18 zeigt einen Entwurf des Fensters.

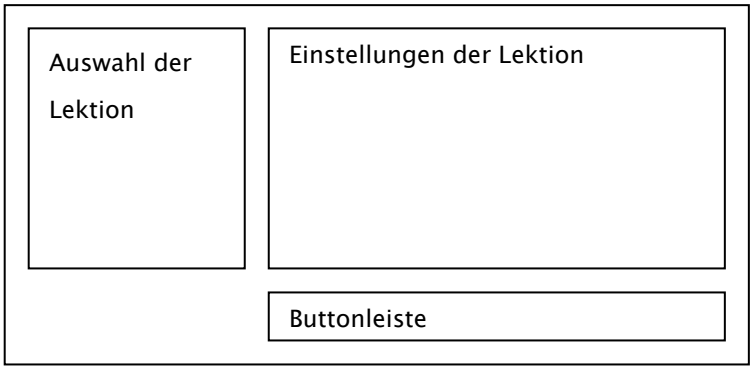


Abb. 3.18: Struktur des Widget *Einstellungen*

3.4.3 Widget *Schreibtraining*

Das Widget *Schreibtraining* besteht aus vier Elementen, einer Laufschrift, einer virtuellen Tastatur, einer Statusleiste und einer Buttonleiste (siehe Abbildung 3.19). Um während des Schreibtrainings keine Ablenkung für den Benutzer zu schaffen, wird das Aussehen auf die nötigsten Funktionen und Informationen beschränkt. Die Menüleiste des Anwendungsfensters wird ausgeblendet. Die Buttonleiste beinhaltet lediglich Schaltflächen für den vorzeitigen Abbruch der Lektion und eine Pausefunktion. Auffällige Farben außerhalb der virtuellen Tastatur werden vermieden.

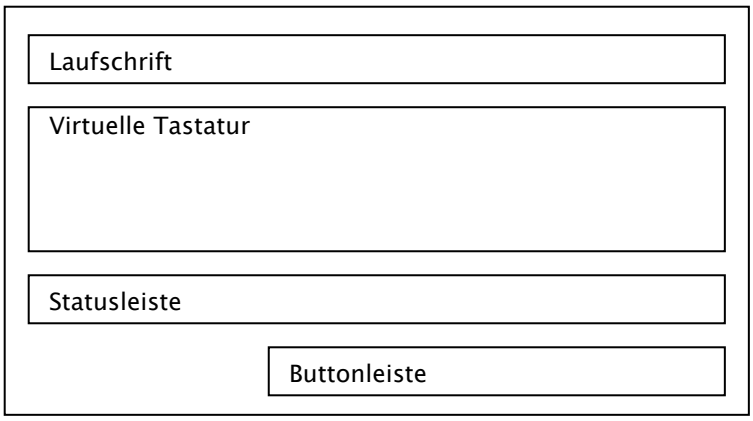


Abb. 3.19: Struktur des Widget *Schreibtraining*

3.4.4 Widget *Ergebnisse*

Um dem Benutzer zwei unterschiedliche Tabellendarstellungen mit Auswertungen anzubieten, teilt sich das Widget *Ergebnisse* durch eine Registerdarstellung in zwei Ansichten (siehe Abbildung 3.20). Das erste Register zeigt die relevanten Ergebnisse der Lektionen und ist auf diese Weise sofort ersichtlich. Wurde ein Schreibtraining durchgeführt, ist der zugehörige Eintrag markiert dargestellt. Das zweite Register bietet eine Auflistung aller verwendeten Schriftzeichen und deren Bewertung. Eine Schaltfläche ermöglicht dem Benutzer, das Auswertungsfenster zu schließen und zum Widget *Einstellungen* zurückzukehren.

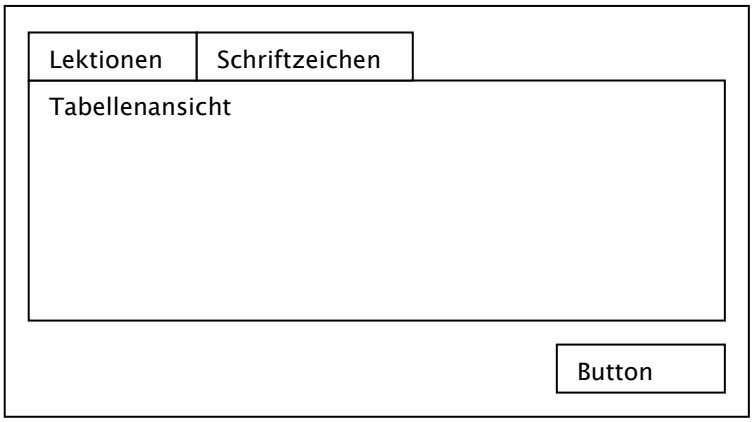


Abb. 3.20: Struktur des Widget *Ergebnisse*

3.4.5 Dialogfenster

Es sind vier Dialogfenster zu implementieren, die sich über die Menüleiste aus dem Anwendungsfenster aufrufen lassen. Eine geeignete Klasse für die Darstellung von Sekundär-Dialogfenstern stellt die *Qt*-Klasse *QDialog* bereit.

Das erste Fenster bietet Grundeinstellungen, die voraussichtlich nur sehr selten vorgenommen werden. Dazu gehören insbesondere Wahl des Tastaturlayouts und Zurücksetzen der Benutzerdaten.

Das zweite Fenster stellt das Hilfesystem dar. Es verfügt über die gleichen Informationen wie die dieser Diplomarbeit beigelegte Bedienungsanleitung (vgl. Kapitel 4.4.1), jedoch in am Bildschirm lesbarer Form. Das dritte Fenster besteht aus einem einfachen Dialog, der lediglich eine Informationsübersicht des Programmnamens, der Programmversion und einer Kurzbeschreibung der *GNU*-Lizenz (*GPL*) beinhaltet.

3.5 Ausgewählte Teile des Programmkonzepts

Im vorliegenden Kapitel werden ausgewählte Teile des Programmkonzepts erläutert, die für die Funktion der Software besondere Bedeutung haben.

In erster Linie handelt es sich hierbei um Entwürfe, die für die grafische Darstellung des Schreibtrainings (Laufband, virtuelle Tastatur, etc.) im Zusammenhang mit der Implementierung des Intelligenzkonzepts notwendig sind. Auf die Erläuterung einfacher Vorgänge wie der Verwendung gebräuchlicher Interaktionselemente der Klassenbibliothek *Qt* (Schaltflächen, Optionsfelder, Listen, Menüleisten, etc.) für die Darstellung von Einstellungsoptionen und Ergebnistabellen wurde an dieser Stelle verzichtet. Auf diese Teile wird im Kapitel 4.2 genauer eingegangen.

3.5.1 Schichtenmodell

Da die Software in erster Linie aus Funktionen für die Benutzungsoberfläche und für den Datenbankzugriff existiert, wird eine Zwei-Schichten-Architektur [Balzert 05, S. 408f.] eingesetzt. Eine Anwendungsschicht realisiert die grafische Oberfläche und den funktionalen Kern der Anwendung, eine Datenhaltungsschicht verwaltet die Zugriffe auf die Datenbank (Auslagerung der Datenhaltung in eigene Klassen siehe auch Kapitel 3.3.6).

Besonders im Schreibtraining wird das Blackbox-Verhalten von Qt genutzt. Einzelne Klassen übernehmen dabei die Funktionalität ganzer grafischer Elemente und werden von einer zentralen, übergeordneten Klasse gesteuert. Auf diese Weise lassen sich Kernfunktionen in der zentralen Klasse einfach erweitern.

3.5.1 *QWidget* der Klassenbibliothek *Qt*

Bei der Gestaltung der Benutzungsoberfläche in Kapitel 3.4 wurden bereits die von *Qt* zur Verfügung gestellten Klassen *QMainWindow*, *QWidget* und *QDialog* angesprochen. Das Klassendiagramm in Abbildung 3.21 zeigt, dass *QMainWindow* und *QDialog* Unterklassen der Klasse *QWidget* darstellen. *QWidget* wiederum erbt von den Klassen *QObject* und *QPaintDevice*, die ein elementarer Teil der Klassenbibliothek *Qt* sind.

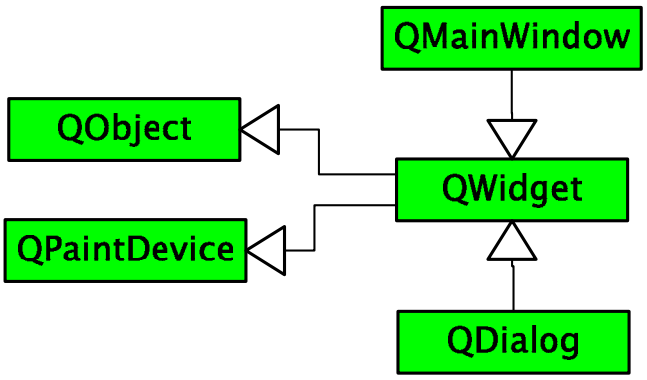


Abb. 3.21: Klassendiagramm: Qt Elementar- und Basisklassen

Für die grafische Benutzungsoberfläche spielt die Klasse *QWidget* eine besondere Rolle. Sie stellt die Basisklasse aller grafischen *Qt*-Objekte für die Benutzerschnittstelle dar. Sie wird durch ein rechteckiges Element auf dem Bildschirm repräsentiert und kann Ereignisse zum Beispiel von Maus oder Tastatur empfangen.

3.5.2 Basiskonzept des Schreibtrainings

Die Aufgaben des Schreibtrainings lassen sich in zwei Bereiche zusammenfassen:

- Grafische Darstellung

  Für den Anwender soll das Diktat in einem Laufband dargestellt, sowie das aktuell zu tippende Schriftzeichen auf einer virtuellen Tastatur und Informationen in einer Statusleiste angezeigt werden.
- Intelligenzkonzept

  Beim Start des Schreibtrainings müssen die ersten Texte für das Diktat erzeugt und während des Diktats regelmäßig aktualisiert werden. Damit die Aktualisierung individuell auf den Anwender zugeschnitten werden kann, müssen aktuelle Schriftzeichen des Diktats und Eingaben des Anwenders laufend gespeichert werden.

Nachfolgend werden die genannten Aufgaben in Klassen unterteilt:
Wie bei der Gestaltung der Benutzungsoberfläche bereits festgelegt, wird das Schreibtraining über ein Widget (*Qt*-Klasse *QWidget*) in einem Anwendungsfenster (*Qt*-Klasse *QMainWindow*) dargestellt. Zu diesem Zweck werden die Klassen ***MainWindow*** und ***TrainingWidget*** definiert.
Um die Elemente Laufschrift, virtuelle Tastatur und Statusleiste individuell behandeln und anzeigen zu können, werden weiterhin die Klassen ***TickerBoard***, ***KeyBoard*** und ***StatusBar*** festgelegt.

Alle Methoden der Datenhaltung sollen in eigene Klassen (Zwei-Schichten-Architektur) ausgelagert werden. Da die Klasse *TrainingWidget* für die Erzeugung der dynamischen Diktate und die Klasse *KeyBoard* für die Tastenmarkierung eines Zeichens Zugriffe auf die Datenbank durchführen, werden für das Schreibtraining zwei weitere Klassen, ***TrainingSql*** und ***KeyboardSql*** vorgesehen.

Abbildung 3.22 veranschaulicht den Klassenaufbau anhand eines Use-Case-Diagramms mit Inkludierungsbeziehungen:

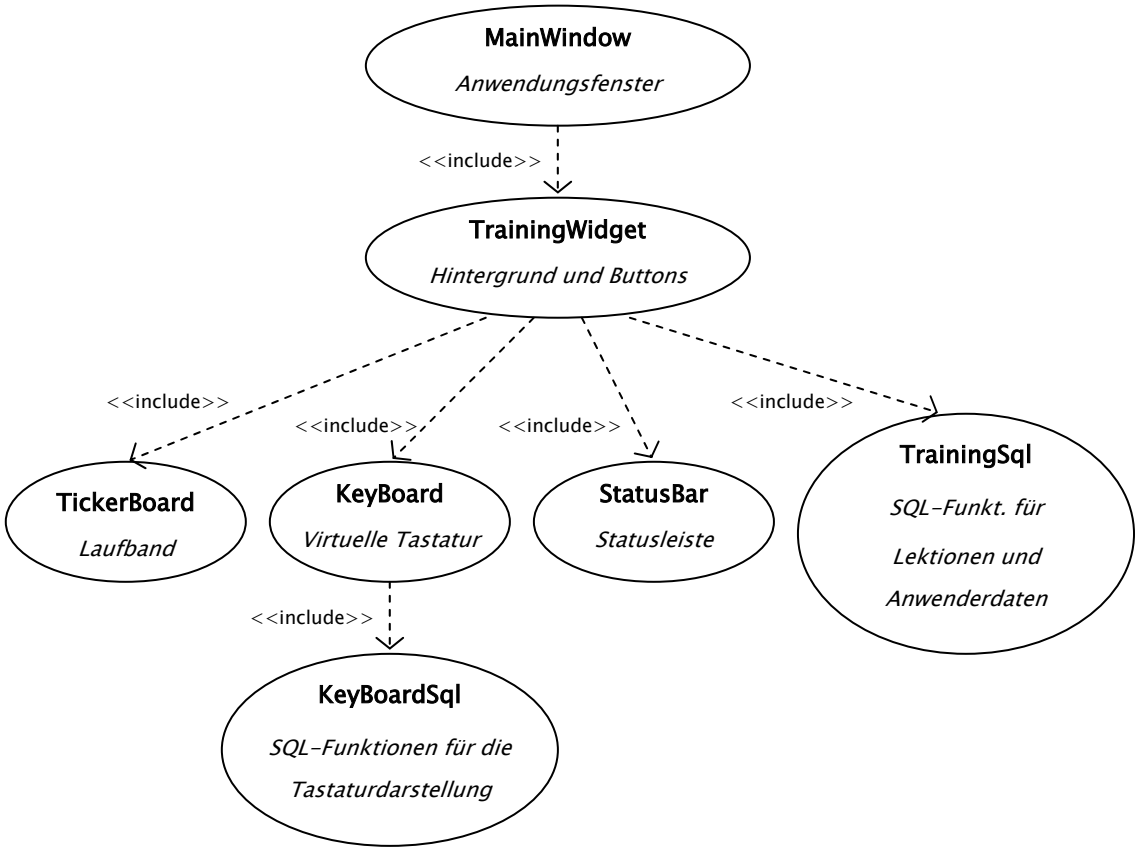


Abb. 3.22: Klassenaufbau und Inkludierungen des Schreibtrainings

Das Klassendiagramm in Abbildung 3.23 zeigt alle für das Schreibtraining festgelegten Klassen und deren Vererbung von der Basisklasse *QWidget*. Ein vollständiges Diagramm aller Klassen, die für die Software erstellt werden, findet sich im Anhang B.

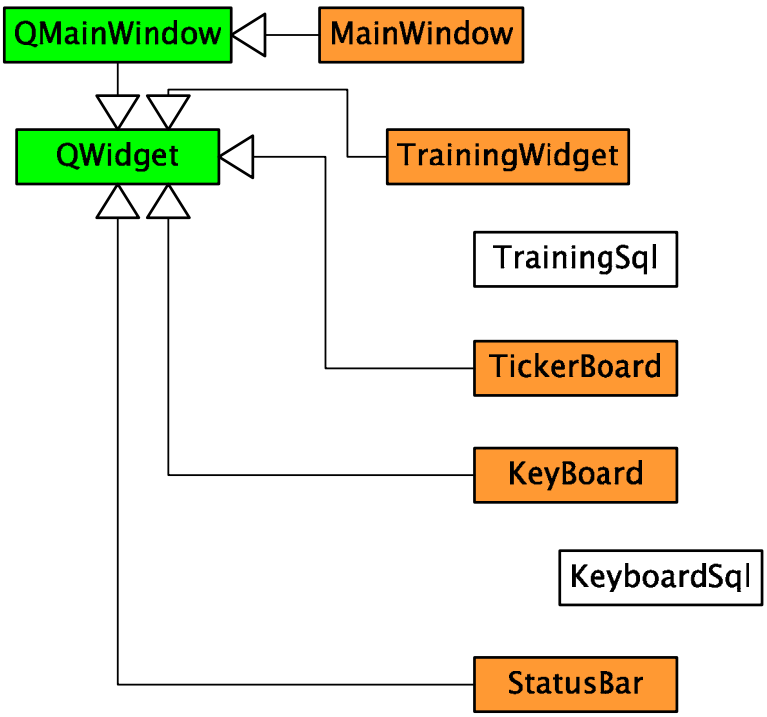


Abb. 3.23: Klassendiagramm: Schreibtraining

Wie die festgelegten Klassen kommunizieren und welche Aufgaben sie erfüllen, wird in den nachfolgenden Kapiteln genauer erläutert.

3.5.2.1 Klasse *MainWindow*

Die Klasse *MainWindow* wird direkt von der *main*-Methode erzeugt und stellt das eigentliche Anwendungsfenster dar. Dazu erbt diese Klasse von der *Qt*-Klasse *QMainWindow* (vgl. Kapitel 3.4.1). Während des Schreibtrainings bildet das Anwendungsfenster lediglich ein Objekt der Klasse *TrainingWidget* ab und stellt die Titelleiste des Fensters zur Verfügung. Die Menüleiste ist während des Schreibtrainings nicht sichtbar.

3.5.2.2 Klasse *TrainingWidget*

Diese Klasse leitet das Schreibtraining und stellt es dar. Sie erzeugt Objekte der Klassen *TickerBoard*, *KeyBoard* und *StatusBar*, sowie zwei Schaltflächen für Pause- und Abbruchfunktion. Zudem verbindet sie eigene Methoden mit allen Objekten über die *Signal*- und *Slot*-Funktionen von *Qt* und verwaltet so das gesamte Schreibtraining. Tippfehler werden erkannt, Zählungen werden durchgeführt und die Dauer der Lektion kontrolliert. Von dieser Klasse werden Methoden der Datenhaltungsklasse *TrainingSql* aufgerufen, um Diktate zu erzeugen und zu aktualisieren. Zudem müssen Anwendereingaben und Lektionenergebnisse an die Klasse *TrainingSql* übermittelt werden, damit sie hier gespeichert werden können.

3.5.2.3 Klasse *TrainingSql*

Die Datenhaltungsklasse TrainingSql stellt vier Methoden mit SQL-Befehlen bereit:

- Über die erste Methode können einzelne Schriftzeichen in der Tabelle *user_chars* inkrementiert werden. Dies ermöglicht es, Tippfehler des Anwenders pro Schriftzeichen und Häufigkeit eines Schriftzeichens im Diktat zu zählen.
- Mit der zweiten Methode lassen sich alle Eigenschaften eines Diktats (Zeitpunkt, Dauer, Fehlerzahl insgesamt, etc.) speichern, um dem Anwender eine Auswertung der Lektionen zu ermöglichen (Tabelle *user_lesson_list*).
- Die dritte Methode dient dazu, den Text des Diktats zu Beginn des Schreibtrainings zu erzeugen. Sie stellt den ersten Text jeder Lektion aus der Tabelle *lesson_content* zur Verfügung.
- Um den Text des Diktats während einer Trainingslektion regelmäßig zu aktualisieren, lassen sich über die vierte Methode neue Texte anfordern. Sie wendet das eigentliche Intelligenzkonzept mit Hilfe von Beziehungen der

Tabellen *user_chars*, *lesson_content* und *lesson_analysis* an. Folgende Schritte sind dazu notwendig:

1. Anhand der Tippfehler-Gewichtung (vgl. Kapitel 3.3.3) werden aus der Tabelle *user_chars* die Schriftzeichen ausgelesen, die am häufigsten falsch getippt wurden.
2. Über die Tabelle *lesson_analysis* lassen sich anschließend die Identifikationsnummern der Texte feststellen, die diese Zeichen am häufigsten beinhalten (vgl. Kapitel 3.3.4).
3. Eine Beziehung der Tabelle *lesson_analysis* und *lesson_content* ermöglicht dann, entsprechende Texte aus der Tabelle *lesson_content* in der Methode zurückzugeben.

3.5.2.4 Klasse *TickerBoard*

Die Klasse TickerBoard speichert den Text des Diktats und stellt diesen in einem Laufband dar. Dazu wird der Text zeichenweise von Anfang bis Ende mit Hilfe einer farbigen Markierung diktiert. Über Methoden lässt sich der Text des Diktats setzen und erweitern. Weitere Methoden dienen der Tippfehlerdarstellung durch andersfarbige Markierung und um durch Verschiebung der Markierung das nächste Zeichen zurückzugeben.

Um zu bestimmen, wann die Klasse neue Texte benötigt, wird eine Konstante eingeführt. Ihr Wert legt die Anzahl der Schriftzeichen fest, die dem Diktat mindestens verbleiben müssen. Wird der Wert erreicht, fordert die Klasse einen neuen Text an.

3.5.2.5 Klasse *KeyBoard*

Das Widget *KeyBoard* stellt eine virtuelle Tastatur zur Verfügung. Die Klasse kann ein Schriftzeichen empfangen und in Form von markierten Tasten auf der virtuellen Tastatur darstellen (die dafür nötige Umwandlung wird von der Datenhaltungsklasse *KeyboardSql* vorgenommen).

Wird vom Anwender eine Taste auf der realen Tastatur gedrückt, erhält die Klasse *KeyBoard* ein Ereignis, setzt es in ein Schriftzeichen um und sendet es über ein Signal aus.

3.5.2.6 Klasse *KeyboardSql*

KeyboardSql stellt eine Datenhaltungsklasse dar, die Methoden für die Konvertierung eines Schriftzeichens in entsprechende Tasten besitzt. Hierzu wird ein Schriftzeichen als Parameter übergeben. Anschließend wird über die Tabelle *key_layouts* und eine der Tabellen *character_list_win* oder *character_list_mac* die zugehörige Taste sowie die Identifikationsnummer einer eventuell notwendigen Zusatztaste ermittelt.

3.5.2.7 Klasse *StatusBar*

Um Werte und Text dem Anwender in einer Statusleiste anzeigen zu können, stellt die Klasse *StatusBar* ein Widget zur Verfügung, das in drei Textbereiche aufgeteilt ist. Über Methoden kann jeder Bereich individuell beschriftet werden.

3.5.3 Dynamisches Konzept des Schreibtrainings

Die Objekte der im vorherigen Kapitel festgelegten Klassen müssen kommunizieren, um das Schreibtraining darzustellen und das Diktat durchzuführen. Im diesem Kapitel wird das dynamische Konzept, also der zeitliche Ablauf der Kommunikation zwischen den Klassenobjekten entworfen.

Die Einstellungen der Lektionen, die der Anwender vornehmen kann, bieten verschiedene Möglichkeiten, den Ablauf des Diktats zu bestimmen. Um den Entwurf übersichtlich zu erläutern, wurde eine typische Konfiguration herausgegriffen: eine aktivierte virtuelle Tastatur und die Reaktion *Tippfehler blockieren* (siehe Kapitel 2.1.4.1 – Reaktion auf Tippfehler). Anhand dieser

Konfiguration lassen sich die wichtigsten Details bei der Kommunikation und die prinzipielle Funktion der Klassen gut erläutern.

Um die Objekte der festgelegten Klassen miteinander kommunizieren zu lassen, eignen sich die *Signal*- und *Slot*-Funktionen von *Qt* sehr gut. Jede Klasse kann als *Blackbox* angesehen werden, die sowohl Signale aussenden als auch empfangen kann (gegebenenfalls mit Parametern). Dazu werden bestimmte Methoden der Klassen als *Signal* oder *Slot* deklariert und mit Hilfe einer *connect*-Methode verbunden. Sie lassen sich aber auch direkt, ohne Verwendung einer Signalverbindung aufrufen.

Die Signalverbindungen der Klassen, die für das Schreibtraining notwendig sind, werden im folgenden Diagramm grafisch dargestellt:

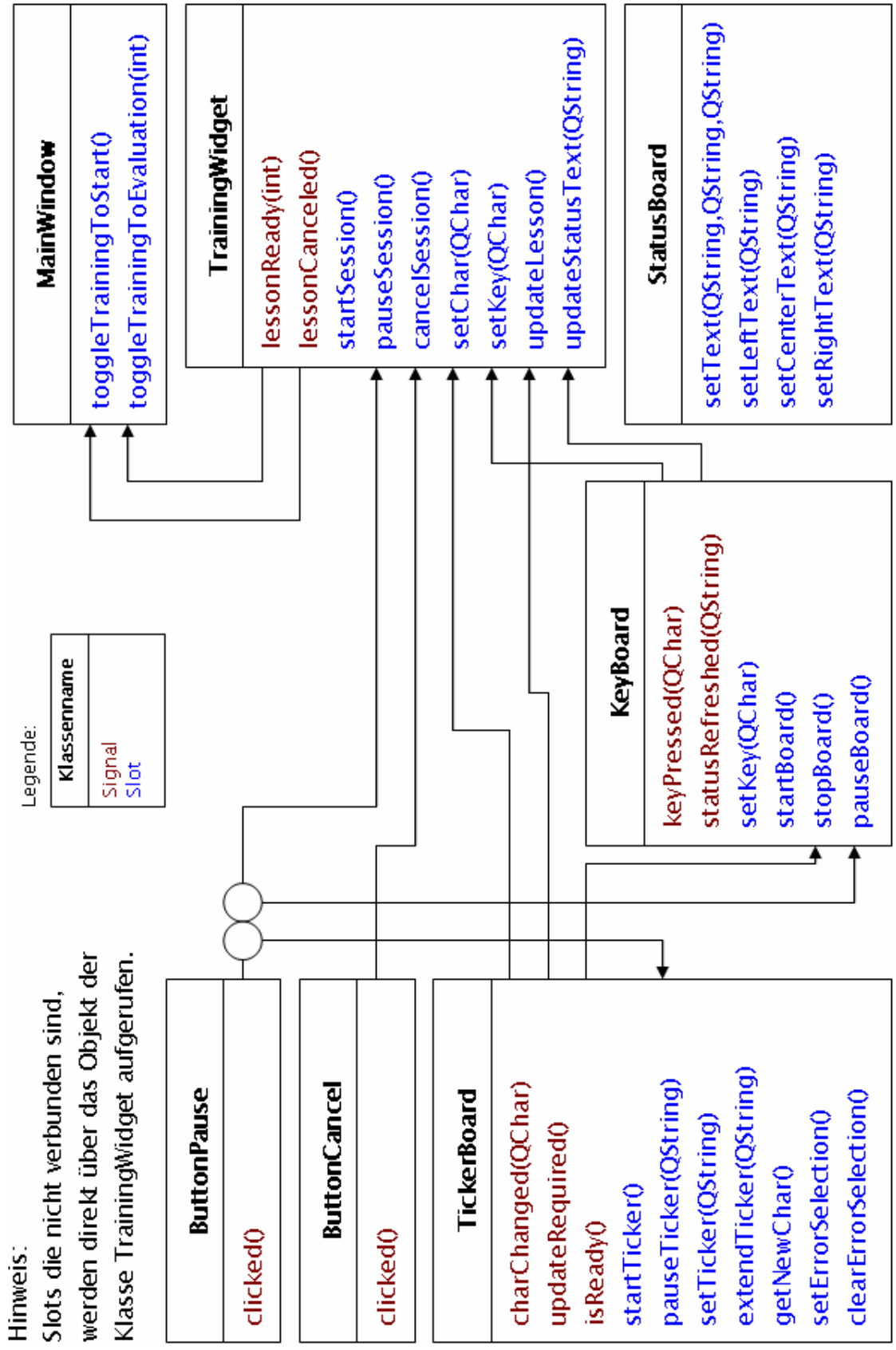


Abb. 3.24: Signalverbindungen während des Schreibtrainings

3.5.3.1 Diktat vorbereiten und starten

Nachdem das Schreibtraining vom Anwender aufgerufen wurde, müssen alle notwendigen Objekte erzeugt und ein Diktat angefordert werden. Damit der Anwender zuerst die Grundstellung auf der Tastatur einnehmen kann, soll das Diktat erst nach dem Drücken der Leertaste gestartet werden.

Das folgende Sequenzdiagramm zeigt den zeitlichen Ablauf nach Erzeugung der Klasse *TrainingWidget* bis zum Start des Diktats durch die Leertaste.

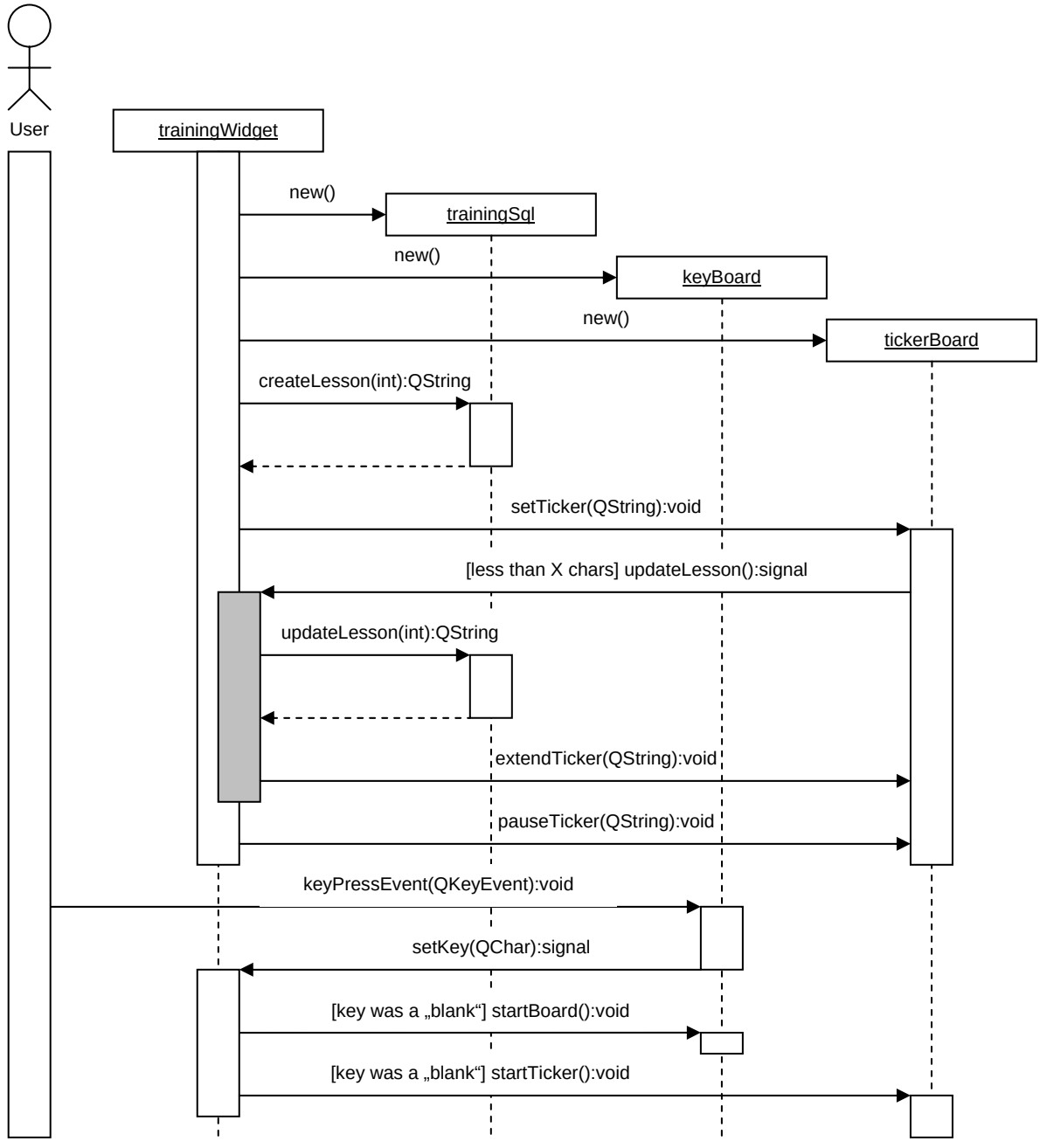


Abb. 3.25: Sequenzdiagramm für den Start des Diktats

Auf die Darstellung der Statusleiste (Klasse *StatusBar*) wurde aus Gründen der Übersichtlichkeit verzichtet. Sie stellt ein einfach zu bedienendes Objekt dar und wird im Kapitel Implementierung erneut aufgegriffen. Zudem werden spezielle Eingriffe in das Schreibtraining, beispielsweise durch die Pausefunktion oder das Beenden der Lektion, außer Acht gelassen.

Das Objekt der Klasse *TrainingWidget* (erzeugt durch die Klasse *MainWindow*) nimmt eine zentrale Funktion ein. Es erzeugt die grafischen Elemente des Schreibtrainings, setzt Verbindungen (*Signal* und *Slot*) und koordiniert das Diktat, indem es mit den einzelnen Elementen kommuniziert.
Zuerst erzeugt das Objekt der Trainingsklasse weitere Objekte für Datenhaltung, virtuelle Tastatur und Laufband. Anschließend wird der erste Text für das Diktat von der Datenhaltung angefordert und an das Laufband übergeben.
Das Laufband übernimmt den Text und überprüft anhand einer Konstante, ob er lang genug ist (vgl. Kapitel 3.5.2.4). Sollte dies nicht der Fall und ein weiterer Text erforderlich sein, wird ein Signal an das Objekt der Trainingsklasse gesendet. Diese fordert daraufhin über die Datenhaltung einen neuen Text an und erweitert die Laufschrift. Dieser Vorgang wird wiederholt, bis das Diktat eine ausreichende Länge besitzt. Nachdem das Diktat bereitgestellt ist, wird das Laufband angehalten, und die virtuelle Tastatur wartet auf einen ersten Tastendruck des Anwenders.
Sobald ein Tastendruck erfolgt, sendet die virtuelle Tastatur ein Signal mit dem getippten Schriftzeichen an das Objekt der Trainingsklasse. Diese überprüft, ob es sich bei dem übermittelten Zeichen um ein Leerzeichen handelt und startet dann gegebenenfalls das Diktat, indem virtuelle Tastatur und Laufschrift gestartet werden.

3.5.3.2 Diktat durchführen

Abbildung 3.26 zeigt das Sequenzdiagramm für den Ablauf nach dem Start des Diktats bis zur Bereitstellung des ersten Schriftzeichens für den Anwender.

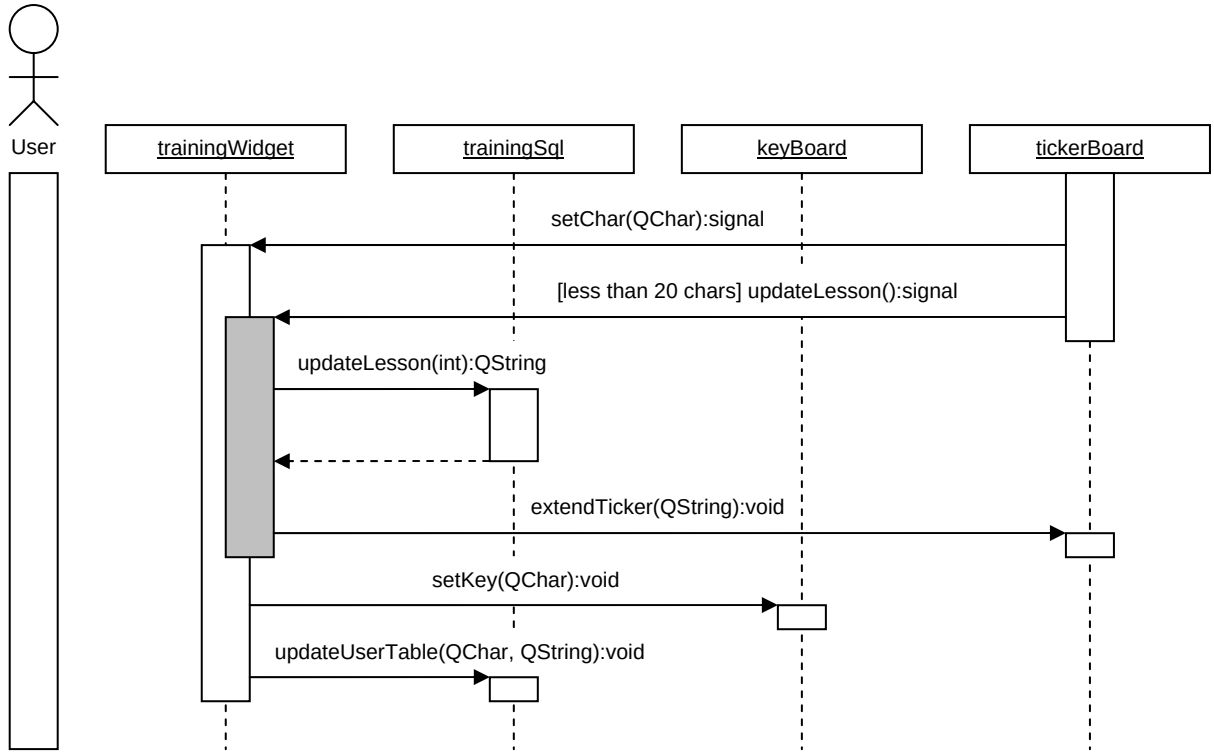


Abb. 3.26: Sequenzdiagramm: Diktat durchführen (1)

Nachdem das Diktat gestartet wurde, übermittelt das Laufband das erste Schriftzeichen des Diktats an das Objekt der Trainingsklasse, stellt das Diktat dem Anwender dar und markiert das erste Zeichen. Zusätzlich überprüft das Laufband erneut die Länge des nun verbleibenden Diktats und fordert gegebenenfalls einen neuen Text an (wie es schon beim Vorbereiten des Diktats erläutert wurde).

Das empfangene Schriftzeichen wird anschließend vom Objekt der Trainingsklasse an die virtuelle Tastatur übermittelt, damit die entsprechenden Tasten markiert werden können. Zudem wird der Zählwert des Zeichens über die Datenhaltung um 1 erhöht. Der Anwender soll nun das Zeichen eingeben, weshalb die virtuelle Tastatur auf den nächsten Tastendruck wartet.

Das nun folgende Sequenzdiagramm kann als Fortsetzung bis zum Abschluss der Lektion angesehen werden, da es in einer Schleife abgearbeitet wird.

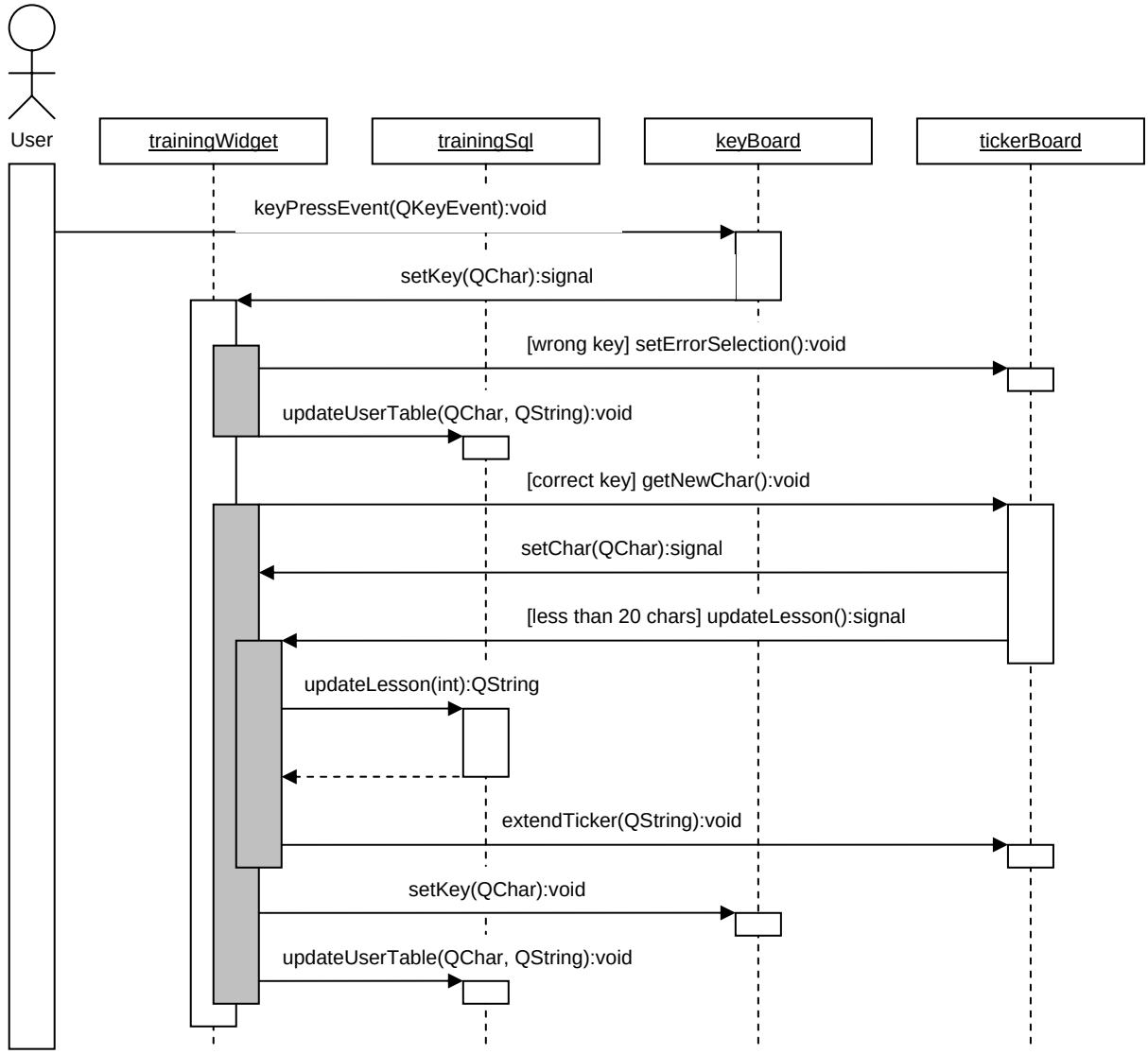


Abb. 3.27: Sequenzdiagramm: Diktat durchführen (2)

Die Tastatur empfängt einen Tastendruck und sendet das empfangene Schriftzeichen an das Objekt der Trainingsklasse. Hier wird nun überprüft, ob es sich um das gleiche Zeichen wie im Diktat handelt, andernfalls handelt es sich um einen Tippfehler. Im Fall eines Tippfehlers wird das Laufband aufgefordert, den Tippfehler durch eine Markierung dem Anwender sichtbar zu machen. Zudem wird die Anzahl der Tippfehler des Schriftzeichens über die Datenhaltung inkrementiert. Die Tastatur ist nun bereit für die nächste Zeicheneingabe des Anwenders.

Wenn nun das korrekte Schriftzeichen eingegeben wird, wird ein neues Schriftzeichen vom Laufband angefordert. Das Laufband gibt dieses daraufhin zurück und erweitert gegebenenfalls das Diktat (wie auch schon nach dem Start des Diktats).

Nachdem das Objekt der Trainingsklasse die virtuelle Tastatur aktualisiert und den Zählvorgang über die Datenhaltung vorgenommen hat, wird erneut auf eine Tasteneingabe gewartet und der gesamte Prozess wiederholt.

3.5.4 Aktualisierungsfunktion

In Kapitel 3.3.3 wurde festgestellt, dass eine Analyse der vorhandenen Texte für das Diktat (hinterlegt in der Tabelle *lesson_content*) durchgeführt und eine entsprechende Analysetabelle (*lesson_analysis*) generiert werden muss. Daher erweist es sich als zweckmäßig, für die Analyse der Daten eine Importfunktion zu nutzen. Dieser Import lässt sich hervorragend mit dem Wunschkriterium des Pflichtenhefts „Aktualisierungsfunktion über eine Internetverbindung“ vereinen. Auf diese Weise können Teile der Datenbank erneuert werden, ohne die Veröffentlichung einer neuen Softwareversion und der damit verbundenen Neuinstallation. Das Wunschkriterium wird daher beim Entwurf der Software einbezogen.

3.5.4.1 Auswahl der zu aktualisierenden Datenbankbereiche

Da die Aktualisierungsfunktion auf die Datenbank angewendet wird, lassen sich zusätzlich zu den Texten der Lektionen auch andere Tabellen in die Aktualisierung einbeziehen.

Das Datenbankdiagramm im Anhang C zeigt alle Tabellen der Datenbank aufgeteilt in drei Bereiche. Der erste Bereich enthält Tabellen, die von der Aktualisierungsfunktion erneuert werden können. Der zweite Bereich, bestehend aus der Analysetabelle, wird automatisch nach der Aktualisierung aus den

vorhandenen Tabellen erzeugt. Der dritte Bereich beinhaltet zwei Tabellen mit anwenderspezifischen Daten, die bestehen bleiben müssen. Dieser Bereich wird daher nicht aktualisiert.

Über die Aktualisierungsfunktion lassen sich so nach der Veröffentlichung in erster Linie eine umfangreichere und/oder verbesserte Tabelle mit Texten für das Diktat, aber auch weitere Tastaturlayouts und Änderungen der Lektionendefinition einbinden. Die Analysetabelle wird dabei stets neu generiert und so an die aktuell vorhandenen Texte angepasst.

3.5.4.2 Format der Aktualisierungsdatei

Da aufgrund des Bestehens der Anwendertabellen nicht die gesamte Datenbankdatei von einem Server heruntergeladen und ersetzt werden kann, muss ein Format festgelegt werden, in dem die Daten zur Verfügung stehen. *XML* ist ein für solche Zwecke häufig verwendetes Format. Allerdings müssten bei dessen Verwendung die Daten nach der Übertragung weiterverarbeitet werden, um sie in die relationale Datenbank *SQLite* zu übernehmen. Eine einfachere Lösung stellt die Möglichkeit dar, eine Liste von *SQL*-Befehlen direkt in einer Datei auf dem Server abzulegen. Nach dem Herunterladen kann diese dann zeilenweise ausgelesen, und die *SQL*-Befehle können direkt an die Datenbank übergeben werden. Zudem kann so beim Erstellen der Aktualisierungsdatei über *SQL*-Befehle entschieden werden, welche Tabellen gelöscht (Befehl *DROP*) und mit neuen Daten wieder erstellt werden sollen (Befehle *CREATE TABLE* und *INSERT INTO*). Einzelne Datensätze auszutauschen, ist aufgrund des umständlichen Vergleichs nicht vorgesehen. Es müsste für jeden Datensatz überprüft werden, ob ein Austausch notwendig ist. Dies wäre insbesondere sehr aufwendig, wenn auch die Definition der Lektionen geändert würde, da jeder hinterlegte Text auf bestimmte Lektionen verweist.

Das in Kapitel 2.2.4 vorgestellte Administrationswerkzeug *SQLite Database Browser* für die Datenbank *SQLite* besitzt eine Exportfunktion, um die gesamte Datenbank als *SQL*-Datei auszugeben. Diese Funktion kann verwendet werden, um nach Veränderungen an der Datenbank eine entsprechende Datei zu generieren, die dann für die Aktualisierungsfunktion eingesetzt werden kann.

Eine Aktualisierung soll jedoch nur durchgeführt werden, wenn sich neue Dateien auf dem Server befinden. Es muss also vor der Aktualisierung festgestellt werden, in welcher Version die Daten vorliegen. Aus diesem Grund wird eine weitere Tabelle *db_version* in die Programmdatenbank eingebunden, die die Version der Datenbank repräsentiert. Sie beinhaltet nur einen Datensatz mit einem ganzzahligen Versionswert. Ein weiterer Wert in einer Versionsdatei auf dem Server dient dann dem Vergleich. Durch das Ablegen der Versionsinformation in einer zweiten Datei (neben der Aktualisierungsdatei) wird verhindert, dass mehr Daten als nötig heruntergeladen werden, falls keine Aktualisierung notwendig sein sollte.

Sind die Werte der Versionstabelle und der Versionsdatei des Servers unterschiedlich, wird der eigentliche Aktualisierungsvorgang eingeleitet. Beim Aktualisierungsvorgang muss daher auf jeden Fall auch der Versionswert in der Datenbank neu gesetzt werden. Dies geschieht ebenfalls über SQL-Befehle in der Aktualisierungsdatei.

Einen Sonderfall stellt der Wert 0 in der Versionsdatei auf dem Server dar. Falls Veränderungen der Software über die Datenbank hinaus eine neue Softwareversion erfordern, ermöglicht dieser Wert, den Anwender darüber zu informieren. Wird also der Wert 0 erkannt, meldet der Schreibtrainer dem Anwender, dass eine Aktualisierung nicht mehr möglich ist, aber eine neue Softwareversion existiert.

Beispiele der Dateien befinden sich auch auf der beigelegten CD-ROM.

3.5.4.3 Aktualisierungsvorgang

Qt bietet eine Klasse *QHttp* die über das Protokoll *HTTP* (Hypertext transfer protocol) eine Verbindung zu einem Server aufnehmen kann. *QHttp* arbeitet wie alle *Qt*-Netzwerkklassen asynchron und akzeptiert einen optionalen Parameter, einen Zeiger auf ein *QIODevice*, in dem die heruntergeladenen Daten landen [Ct 05, S. 244]. Mit diesem Parameter lassen sich die im vorherigen Kapitel besprochene Dateien bequem herunterladen, in eine temporäre Datei speichern und dann auslesen.

Um das Herunterladen einer Datei zu koordinieren, bietet *QHttp* verschiedene Methoden. Für den vorliegenden Fall werden nur die Methoden *setHost()* und *get()* benötigt. Mit ihnen lassen sich ein Server und eine Datei auswählen und das Herunterladen einleiten. Über das Signal *httpDownloadFinished()* lässt sich anschließend feststellen, ob das Herunterladen der Datei erfolgreich beendet wurde.

Die Auflistung der einzelnen Arbeitsschritte veranschaulicht den Vorgang der Aktualisierung:

1. Serveradresse mit Hilfe der Methode *QHttp::setHost()* festlegen
2. Der Methode *QHttp::get()* den Namen der Versionsdatei und einen Zeiger auf eine lokale, temporäre Datei übergeben - dadurch wird das Herunterladen eingeleitet
3. Über das Signal *httpDownloadFinished()* die Beendigung einer fehlerfreien Übertragung feststellen
4. Wert in der temporären Datei mit dem Versionswert in der Datenbank vergleichen
    a. Bei 0 in der temporären Datei das Herunterladen abbrechen und den Anwender über eine neue Softwareversion informieren

   b. Bei identischen Werten das Herunterladen abbrechen und dem Anwender die nicht notwendige Aktualisierung melden
   c. Bei sich unterscheidenden Werten mit Schritt 5 fortfahren
5. Der Methode *QHttp::get()* den Namen der Aktualisierungsdatei und einen Zeiger auf eine lokale, temporäre Datei übergeben
6. Über das Signal *httpDownloadFinished()* die Beendigung einer fehlerfreien Übertragung feststellen
7. Temporäre Datei zeilenweise auslesen und die sich ergebenden *SQL*-Befehle auf die Datenbank anwenden
8. Analyse der Lektionentexte durchführen (siehe Kapitel 3.3.4)
9. Anwender über den erfolgten Aktualisierungsvorgang informieren

Für die Aktualisierungsfunktion wird ein weiteres Dialogfenster (vgl. Kapitel 3.4.5) vorgesehen, das über die Menüleiste des Anwendungsfenster aufgerufen werden kann. Der Dialog wird lediglich mit Startbutton, Fortschrittsbalken und einem Statustext versehen.

3.6 Erstellung der Texte für die Lektionen

Um ein dynamisches und abwechslungsreiches Diktat zu ermöglichen, soll eine möglichst große Anzahl an Texten (Zeichenkombinationen, Wörter und Sätze) mit bestimmten Eigenschaften in der Datenbank hinterlegt werden. Das vorliegende Kapitel befasst sich auch mit dem Entwurf einer geeigneten Methode, um diese Texte zu erstellen.

3.6.1 Notwendige Eigenschaften der Texte

Die Texte für das Diktat müssen bestimmte Eigenschaften erfüllen, die von den Kriterien des Intelligenzkonzepts (Kapitel 3.2.4) und der Definition der Lektionen

(Kapitel 3.2.5) abgeleitet werden können. Nachfolgend werden alle Eigenschaften zusammengefasst, die für die Erstellung der Texte festgelegt wurden:

- Texte müssen einzelnen Lektionen zugeordnet werden
- Die Texte einer Lektion dürfen nur bis dahin erlernte Schriftzeichen beinhalten
- In den ersten Lektionen können auch Zeichenfolgen ohne Sinn verwendet werden (aufgrund der geringen Anzahl möglicher Schriftzeichen)
- Worte bis inklusive Lektion 5 müssen in Kleinschreibung vorliegen
- Ab Lektion 6 müssen ganze Sätze verwendet werden
- Die Satzlänge muss auf 120 Zeichen begrenzt werden

3.6.2 Herkunft der Texte

Für die Auswahl der Texte kann auf verschiedene Weise vorgegangen werden. Zunächst kommt das eigene Erstellen von Texten in Betracht. Aufgrund der vielen einzuhaltenden Kriterien und der großen Anzahl an benötigten Texten, ist das manuelle Erstellen aber sehr (zeit-)aufwendig. Daher sollen bereits bestehende Texte verwendet und an den Schreibtrainer angepasst werden. Es ist also eine Quelle notwenig, über die vorhandene Texte bezogen werden können.

Das Verwenden von Texten aus anderen Schreibtrainern scheidet aus, sie sind urheberrechtlich geschützt und dürfen nicht verwendet werden. Es können jedoch auch nicht beliebig Texte (z.B. aus dem Internet) herangezogen werden. Diese unterliegen, zumindest sobald es sich bei den herausgefilterten Textstellen um ganze Sätze handelt, (ebenso wie die Texte aus vorhandenen Schreibtrainern) prinzipiell dem Schutz des Urheberrechts.

Nach § 24 UrhG ist die Benutzung fremder geschützter Werke ohne Zustimmung ihrer Urheber zwar erlaubt, wenn dadurch eine neue, selbstständige Schöpfung hervorgebracht wird. Um aber die Schwierigkeit zu umgehen, beurteilen zu müssen, ob die Texte eine neue, selbstständige Schöpfung darstellen, soll auf

Texte zurückgegriffen werden, die nicht urheberrechtlich geschützt sind, sogenannte *gemeinfreie* Texte. *Gemeinfrei* und damit für jedermann frei verfügbar werden Texte gemäß § 64 UrhG 70 Jahre nach dem Tod des Urhebers (bei anonymen und pseudonymen Werken 70 Jahre nach dem Erscheinungs- bzw. Entstehungsdatum des Textes). [Haberstumpf, S.168ff]

Man findet im Internet einige Sammlungen von gemeinfreien Texten deutscher Autoren, zum Beispiel unter *http://www.gutenberg.spiegel.de*. Es ergibt sich jedoch ein weiteres Problem - diese Sammlungen beinhalten zwar ausschließlich gemeinfreie Texte, sind jedoch zumeist ihrerseits als sogenanntes *Sammelwerk* gemäß § 4 UrhG oder als sogenannte *Darstellung* i.S. des § 2 Abs. 1 Nr.7 UrhG urheberrechtlich geschützt. [Haberstumpf, S.74, S.89]

Eine Ausnahme stellt die Internetseite *http://www.digbib.org* dar, eine Sammlung gemeinfreier Werke, die ausdrücklich auf eigenen urheberrechtlichen Schutz verzichtet und ihre Inhalte zur freien Verfügung stellt. Daher wird für den Schreibtrainer auf diese Internetseite zurückgegriffen und Texte aus der auf ihr dargestellten Sammlung herausgefiltert.

3.6.3 Werkzeug für die Filterung vorhandener Texte

Für die automatisierte Filterung vorhandener Texte wird ein Werkzeug entwickelt. Das Werkzeug soll benutzt werden, um beliebige Textabschnitte der Sammlung von *DigBib.org*, aber eventuell in Zukunft auch von anderen, nicht geschützten oder freigegebenen Quellen zu analysieren und gezielt Wörter und Sätze mit bestimmten Eigenschaften herauszufiltern. Das Werkzeug direkt auf die Internetseiten zugreifen zu lassen wird nicht vorgesehen, da so auch Texte außerhalb der eigentlichen Sammlung einbezogen werden könnten. Es werden daher manuell Textabschnitte in das Werkzeug kopiert, die dann nach einem Schema geteilt und analysiert werden.

Die grundsätzlichen Arbeitsschritte für die Filterung einzelner Worte aus einem Textabschnitt werden nachfolgend erläutert:

1. Schriftzeichen der vordefinierten Lektionen für eine Analyse werden hinterlegt. Geeignet ist ein zweidimensionales Feld, wobei der erste Index des Feldes die jeweilige Lektion bestimmt und der zweite Index auf alle Schriftzeichen dieser Lektion verweist.
2. Der gesamten Textabschnitt wird anhand des Trennzeichens *Leerzeichen* in Worte geteilt und mit diesen ein weiteres Feld gefüllt. (Für Sätze: der gesamte Textabschnitt wird mittels der Trennzeichen *Punkt* oder *Fragezeichen* oder *Ausrufezeichen* geteilt). Für die ersten fünf Lektionen werden nur Worte in Kleinschreibung beachtet (oder Worte für diesen Zweck in Kleinbuchstaben umgewandelt).
3. Alle Wörter werden in einer Schleife durchlaufen.
    a. Jedes Schriftzeichen eines Wortes wird durchlaufen.
        I. Jedes Schriftzeichen wird mit den Schriftzeichen aller Lektionen (Arbeitsschritt 1) verglichen und der Index der zugehörigen Lektion gemerkt.
4. Jedem Wort wird die höchste Nummer der Lektion (erster Index) zugeordnet, die ein Schriftzeichen des Wortes enthielt
5. Wort mit zugeordneter Lektionennummer wird in einer Tabelle abgelegt, falls es noch nicht existiert. Achtung: Wörter nur bis zur Lektion 6 und Sätze erst ab Lektion 7 speichern

(Für die Filterung ganzer Sätze ist prinzipiell der gleiche Vorgang notwendig, jedoch unterscheiden sich die Trennzeichen, nach denen die Texte getrennt werden (siehe Arbeitsschritt 2), und die Satzlänge muss auf 120 Zeichen begrenzt werden.)

Für die Implementierung eines solchen Werkzeugs kann eine beliebige Plattform und Programmiersprache verwendet werden, da nur die resultierenden Daten in der Datenbank eine Rolle spielen. Eine SQL-Datenbank zu verwenden ist sinnvoll, damit die Texte direkt in Form von SQL-Befehlen in eine Aktualisierungsdatei übernommen werden können.

4. Implementierung

Im Entwurf der Software wurden bereits wichtige Programmteile, wie Intelligenzkonzept, Datenbank und Gestaltung der Oberfläche festgelegt. Für das Schreibtraining wurde eine Aufteilung in Klassen vorgenommen und der zeitliche Ablauf bestimmt.

Im vorliegenden Kapitel soll besonders auf den Einsatz der Klassenbibliothek *Qt* (Installation und Kompilierung), den Test der Software und die Veröffentlichung eingegangen werden. Zudem werden ausgewählte Klassen genauer erläutert, die aus dem Entwurf nicht hervorgingen und eine spezielle Herangehensweise erforderten. Entstandene Schwierigkeiten bei der Realisierung werden dabei ebenfalls berücksichtigt.

4.1 Vorarbeit

4.1.1 Programmierrichtlinien

Um einen einheitlichen Programmierstil zu gewährleisten, richtet sich die Implementierung nach den Richtlinien der *GNU Coding Standards* [WV Gnustandard], sowie dem für die Programmiersprache Java bekannten *Java Coding Style Guide* [WV Javastandard]. Die Standards von *GNU* stellen vor allem Informationen über die Veröffentlichung freier Software sowie allgemeine Richtlinien für die Umsetzung eines sauberen, konsistenten und einfach installierbaren Systems zur Verfügung. Da der Leitfaden von *Java* in einigen Teilen, vor allem in Bezug auf die Namenskonventionen, exaktere Anweisungen beinhaltet als der Standard von *GNU*, wird die Implementierung auch hiervon Gebrauch machen, obwohl es sich um ein Programm in der Sprache C++ handelt.

Nachfolgend sind die wichtigsten Standards, die in der Implementierung Verwendung finden, zusammengefasst:

- Namenskonventionen

  Die Verwendung von Groß- und Kleinschreibung für Klassen, Funktionen, Variablen und Konstanten sind genau geregelt. So beginnen Klassennamen sowie alle folgenden Wörter oder Akronyme stets mit einem Großbuchstaben. Alle anderen Buchstaben sind klein geschrieben. Unterstriche für die Trennung von Namen sind nicht erlaubt.

  Methoden-, Objekt- und Variablennamen dagegen beginnen immer mit einem Kleinbuchstaben, sind aber sonst identisch mit den Konventionen für Klassen.

  Namen für Konstanten bestehen nur aus Großbuchstaben, Wörter und Akronyme werden mit einem Unterstrich abgetrennt.

  Alle Namen erläutern den genauen Zweck der jeweiligen Aufgabe und sind in englischer Sprache verfasst.

- Tabulatorgröße

  Tabulatoren besitzen eine feste Größe von 4 Spalten.

- Maximale Spaltenanzahl einer Zeile

  Um eine gute Lesbarkeit des Quelltextes zu gewährleisten, soll eine Zeile eine Anzahl von 80 Spalten nicht übersteigen. Bei dadurch entstehenden Umbrüchen werden die zugehörigen Folgezeilen um einen Tabulator eingerückt.

- Kommentare

  Der Quelltext muss umfangreich kommentiert sein und auf diese Weise den Lesern ermöglichen, die Programmsprache besser zu verstehen. Es werden drei unterschiedliche Kommentierungsmethoden angewandt: Dokumentations-, Block- und Zeilenkommentierung.

Da für die Dokumentation des Quelltextes das Softwarewerkzeug *DoxyGen* [WV Doxygen] verwendet wird, sind die hierfür nötigen Kommentare mit zugehörigen Kommandos zu versehen, die von *DoxyGen* verstanden werden. *DoxyGen* bietet unterschiedliche Möglichkeiten der Kommentierung, daher wird ein Java-Standard, der von der Kommentierungssoftware *JavaDoc* weit verbreitet ist, eingesetzt.

DoxyGen erstellt aus den Kommentaren eine Dokumentation (z.B. im *HTML*- oder *RTF*-Format), die dazu dienen soll, übersichtlich die Funktionalität einzelner Klassen, Methoden und Eigenschaften zu erläutern.

4.1.2 Programmname und Internetadresse

Der 10-Finger-Schreibtrainer erhält den Namen *Tipp10*. Kriterien für die Wahl des Namens waren:

- Er soll die Funktion der Software beschreiben
- Er soll kurz und einprägsam sein
- Es soll eine passende Internetadresse mit der Endung „de“ verfügbar sein

Heute wird Software vorwiegend über das Internet angeboten. Dabei spielt die Wahl der Internetadresse für die Veröffentlichung eine große Rolle. Die Internetadresse *http://www.tipp10.de* war zur Zeit der Implementierung verfügbar und wurde daher registriert. Sie ist nötig, um die Dateien für die Aktualisierungsfunktion unter einer festen Serveradresse (IP-Adresse) zu erreichen. Zudem kann sie für Internetseiten verwendet werden, über die Programmdateien und Informationen bezogen werden können.

4.1.3 Installation von Qt

Die Open Source Variante von *Qt 4.1.1* für Windows ist als selbstextrahierendes Installationspaket verfügbar und wird im Normalfall zusammen mit dem *GCC*-Compiler *MinGW* verwendet. Dies ist darin begründet, dass der *MinGW* Compiler für Windows ebenfalls unter die Lizenz von *GNU* (*GPL*) gestellt ist. Die Installationsroutine bietet daher die Möglichkeit, zusätzlich zur *Qt*-Installation automatisch den *MinGW* Compiler aus dem Internet zu laden und zu installieren. Befindet sich noch kein Compiler auf dem Entwicklungssystem, ist die Wahl dieser Option sehr empfehlenswert.
Nach der Installation von *Qt* und gegebenenfalls *MinGW* ist es notwendig, *Qt* zu konfigurieren und entsprechend der eigenen Bedürfnisse zu kompilieren (und zu linken). Dies geschieht über das Programm *configure* im Hauptverzeichnis von *Qt*. *configure* kann mit verschiedenen Parametern aufgerufen werden und nimmt anschließend die Konfiguration der *Qt*-Bibliothek vor.

Für die Implementierung des Schreibtrainers wurde eine statische Konfiguration ausgewählt. Sie vereinfacht die Weitergabe, da alle Dateien bei der Kompilierung zu einer einzigen ausführbaren Datei zusammengefasst werden. Zudem werden keine Plugins, z.B. in der Form von *DLL*-Dateien, verwendet, für die eine geteilte (*shared*) Konfiguration notwendig wäre. Weiterhin muss die Konfiguration den entsprechenden *SQL*-Treiber in die Bibliothek einbinden. In Fall des Schreibtrainers handelt es sich hier um das Datenbanksystem *SQLite*.

Zwei Konfigurationen wurden angewandt, eine für den Fehlererkennungs-Modus (*debug mode*) während der Implementierung, und eine für den Veröffentlichungs-Modus (*release mode*). Abbildung 4.1 und 4.2 zeigen den Aufruf der ausführbaren Datei configure.exe mit den besprochenen Parametern.

```
configure -debug -static -qt-sql-sqlite
```

Abb. 4.1: Konfiguration von Qt im Fehlererkennungs-Modus

```
configure -release -static -qt-sql-sqlite
```

Abb. 4.2: Konfiguration von Qt im Veröffentlichungs-Modus

Der Vollständigkeit halber soll abschließend noch auf die Verwendung von Ausnahmebehandlungen (*Exception handling*) eingegangen werden, insbesondere auf die Frage, warum diese für das vorliegende Projekt keine Rolle spielen.
Die Klassen von *Qt* verwenden selbst keine Ausnahmebehandlung. Es ist möglich, Ausnahmebehandlung zu verwenden, aber grundsätzlich wird empfohlen, Fehler über Rückgabewerte abzufangen und bei grafischen Oberflächen mit einer Fehlermeldung an den Benutzer zu melden. Die Verwendung von Ausnahmebehandlungen kann jedoch bei der Installation von Qt über den Parameter *-exceptions* des Befehls *configure* aktiviert werden, sollte er gewünscht oder benötigt werden.

4.1.4 Kompilieren

Um das Kompilieren des Schreibtrainers nachvollziehen zu können, sollen im vorliegenden Kapitel die dafür nötigen Arbeitsschritte erläutert werden.
Qt bietet ein projektorientiertes System mit der Bezeichnung *qmake* an. Dieses ermöglicht, Build-Umgebungen auf verschiedenen Plattformen zu erstellen und zu aktualisieren.

Um *qmake* zu nutzen, muss für das Projekt eine Datei mit der Dateiendung *pro* erstellt werden, die das Projekt beschreibt. Für dieses Projekt wurde eine Datei *tipp10.pro* erzeugt. Diese Datei enthält eine Liste der verwendeten Quelldateien (in einer festen Struktur) und einige Konfigurationsoptionen. Der Inhalt der gesamten Datei findet sich auf der beigelegten CD-ROM. Hier soll lediglich auf zwei wichtige Konfigurationsoptionen hingewiesen werden.

Da für die Software auch externe Bilddateien benötigt werden, die zur Laufzeit geladen werden, wurden Verweise auf zwei weitere Konfigurationsdateien in die Datei *tipp10.pro* eingebunden (siehe Abbildung 4.3), die wiederum auf die Bilddateien verweisen.

```
RC_FILE = tipp10.rc
RESOURCES = tipp10.qrc
```

Abb. 4.3: Verweise auf Konfigurationsdateien in der Projektdatei

Die Datei *tipp10.rc* verweist auf das Icon der Anwendung, die Datei *tipp10.qrc* auf alle *PNG*-Grafiken die in der Anwendung geladen werden. Beide Dateien befinden sich im Hauptverzeichnis der Verzeichnisstruktur für die Implementierung.

Bei der Installation von *Qt* wurde bereits der *SQL*-Treiber implementiert. Der Projektdatei muss nun noch mitgeteilt werden, dass dieser Treiber im Projekt verwendet werden soll. Dies geschieht wie folgt:

```
QT += sql
```

Abb. 4.4: Verweis auf den SQL-Treiber in der Projektdatei

Wird nun die Projektdatei *tipp10.pro* als Parameter des Befehls *qmake* aufgerufen, wird automatisch eine Build-Umgebung mit entsprechendem *Makefile* generiert. Dieser Vorgang muss nur wiederholt werden, wenn die Projektdatei geändert wird, nicht aber, wenn bereits eingebundene Dateien modifiziert werden.

Das entstandene Makefile kann anschließend mit den gewohnten Parametern, wie zum Beispiel *clean*, *release* oder *debug* aufgerufen und so das Programm kompiliert werden.

Nachfolgend finden sich die Befehle für die Erstellung der Build-Umgebung über die Projektdatei und der Kompilierung für eine Veröffentlichung (release mode):

```
qmake tipp10.pro
make release
```

Abb. 4.5: Erstellung der Build-Umgebung und Kompilierung der Software

4.1.5 Verzeichnisstruktur

Für die Implementierung wurde folgende Verzeichnisstruktur angelegt, um die Dateien einzelnen Aufgaben zuordnen zu können:

source	Hauptdateien (*main*-Methode, *Makefile*, Datenbank etc.)
debug	Zielverzeichnis für Debug-Dateien (Fehlersuche)
def	C++ Headerdateien mit Definitionen
img	Verwendete Bilddateien und Icons
release	Zielverzeichnis für Release-Dateien (Veröffentlichung)
sql	C++ Dateien für die Datenhaltung
test	Dateien für Testzwecke
widget	C++ Dateien für GUI und funktionalen Kern

4.2 Ausgewählte Teile der Programmrealisierung

Die Realisierung der Software ist trotz der geringen Anzahl an Programmfunktionen zu umfangreich, um sie in dieser Arbeit detailliert zu erläutern. Die Aufgaben der erstellten Klassen werden durch eine Vielzahl an Eigenschaften und Methoden repräsentiert, deren Zusammenhang aus dem Quelltext und den Kommentaren schnell ersichtlich wird. Eine detaillierte schriftliche Erläuterung wäre sehr unübersichtlich und ausschweifend, insbesondere da auch eine große Anzahl an Eigenschaften und Methoden der Klassenbibliothek *Qt* aufgeführt werden müssten.

Aus diesem Grund werden im vorliegenden Kapitel ausgewählte Teile erläutert, die eine spezielle Herangehensweise erforderten oder Schwierigkeiten aufzeigten. Eine detaillierte Dokumentation aller Eigenschaften und Methoden der implementierten Klassen befindet sich jedoch gesondert auf der CD-ROM. Sie liegt im PDF- und im HTML-Format vor. Für die Erzeugung diente die in Kapitel 4.1.1 eingeführte Dokumentationssoftware *DoxyGen* im Zusammenhang mit den Kommentaren der Quelltext-Dateien.

4.2.1 Auswahl und Einstellungen der Lektion

Direkt nach dem Aufruf des Schreibtrainers wird das im Kapitel 3.4.2 festgelegte Widget *Einstellungen* dargestellt. Abbildung 4.6 zeigt das für den Anwender ersichtliche Fenster:

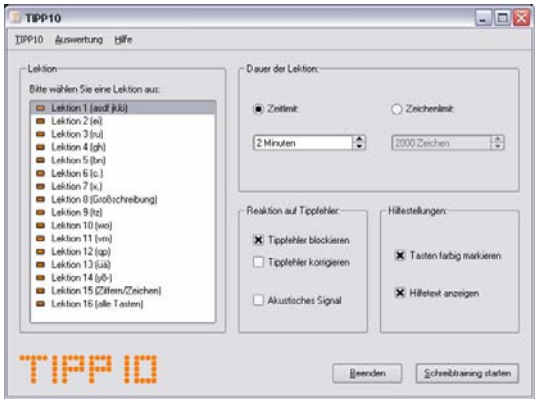


Abb. 4.6: Screenshot: Widget *Einstellungen*

Die Implementierung der für das Fenster nötigen Klassen wurde herangezogen, um zu erläutern wie die Interaktionselemente von *Qt* genutzt und deren Werte gespeichert wurden.

Das Objektdiagramm in Abbildung 4.7 zeigt den Aufbau der erstellten Klassen, um das Fenster darzustellen. Wie im Entwurf erläutert, wird das Anwendungsfenster (Klasse *MainWindow*) von der main-Methode erzeugt. Das Objekt der Klasse MainWindow beinhaltet eine Menüleiste, die beim Start des Schreibtrainers angezeigt wird.

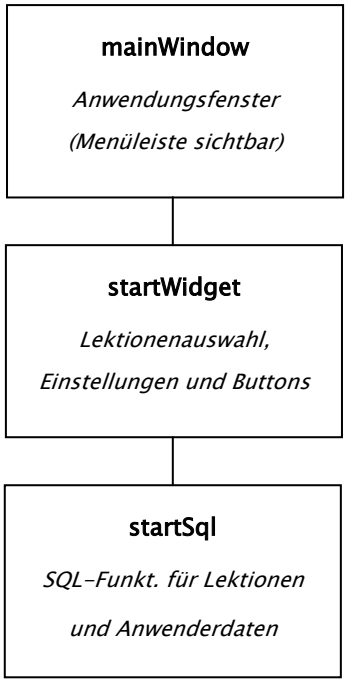


Abb. 4.7: Objektdiagramm: Widget *Einstellungen*

Das Objekt *startWidget* zeigt die Wahl der Lektion, alle Einstellungsmöglichkeiten und Schaltflächen, um das Training zu starten oder das Programm zu beenden. Sie werden mit Hilfe von Interaktionselementen der Klassenbibliothek *Qt* realisiert. Dazu gehören unter anderem die Klassen *QListWidget*, *QPushButton*, *QSpinBox*, *QCheckBox*, *QRadiobutton* und Klassen für die Formatierung wie *QLabel*, *QGroupBox* oder *QBoxLayout*. Auf den Einsatz der einzelnen *Qt*-Klassen soll an dieser Stelle nicht genauer eingegangen werden, jedoch auf nützliche Hilfsmittel, um die Elemente mit Werten zu hinterlegen.

Um die Liste zu füllen, in der der Anwender eine Lektion auswählen kann, wurde die Klasse *StartSql* eingesetzt. Sie ermittelt aus den Tabellen *lesson_list* und *user_lesson_list* die bereits absolvierten Lektionen und stellt alle Lektionen mit einer passenden Grafik in der Liste dar. Auf diese Weise ist dem Anwender sofort ersichtlich, welche Lektionen bereits einmal oder mehrfach durchgeführt wurden. Der dafür nötige *SQL*-Befehl lautet:

```
SELECT lesson_list.lesson_id, lesson_list.lesson_name,
      lesson_list.lesson_description,
      COUNT(user_lesson_list.user_lesson_lesson) FROM lesson_list
      LEFT JOIN user_lesson_list ON lesson_list.lesson_id =
      user_lesson_list.user_lesson_lesson GROUP BY
      lesson_list.lesson_id;
```

Abb. 4.8: SQL-Befehl: Lektionenliste

Die ermittelte Anzahl jeder Lektion aus der Tabelle *user_lesson_list* bestimmt, welche Grafik einer Lektion zugeordnet wird (siehe Abbildung 4.9). Wurde die Lektion bereits einmal durchgeführt, wird ein Haken dargestellt. Wurde sie bereits öfter als einmal trainiert, wird ein Doppelhaken angezeigt. Ein Strich bedeutet, dass die Lektion noch nie durchgeführt wurde.

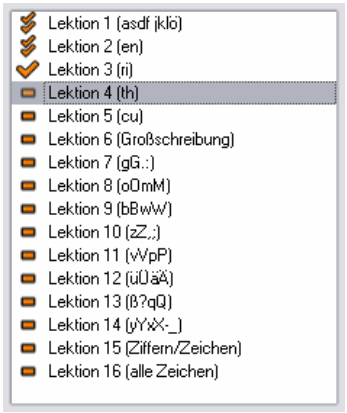


Abb. 4.9: Screenshot: Lektionenliste

Die Einstellungen müssen beim ersten Programmstart Standardwerte annehmen; bei späteren Ausführungen werden die zuletzt gewählten Einstellungen des Anwenders verwendet. Für diesen Zweck wurde eine Methode zum Auslesen der Werte und eine zweite zum Speichern der Einstellungen realisiert. Beide Methoden setzen die *Qt*-Klasse *QSettings* ein.

Die Klasse *QSettings* ermöglicht es, Einstellungen plattformunabhängig und ohne den Einsatz einer Datenbank zu speichern (Methode *setValue*) und auszulesen (Eigenschaft *value*). Je nach Plattform werden die Werte in der *Registry* (*Windows*), in *XML*-Dateien (*MacOS*) oder *INI*-Textdateien (*Unix*) abgelegt. Ein zusätzlicher

Parameter beim Auslesen ermöglicht den Einsatz eines Standardwerts, falls noch kein Wert gespeichert wurde.

Nachfolgend werden die notwendigen Methoden exemplarisch für die Einstellungsoption *Tasten farbig markieren* der virtuellen Tastatur vorgestellt:

```
void StartWidget::readSettings() {
      QSettings settings;
      settings.beginGroup("support");
      checkKeySelection->setChecked(settings.value("check_selection",
            true).toBool());
      settings.endGroup();
}
void StartWidget::writeSettings() {
      settings.beginGroup("support");
      settings.setValue("check_selection",
            checkKeySelection->isChecked());
      settings.endGroup();
}
```

Abb. 4.10: SQL-Befehl: Einstellungen lesen und speichern

In vielen auf dem Markt bekannten Anwendungen werden für Interaktionselemente sogenannte *ToolTips* verwendet. Sie ermöglichen es, eine Kurzbeschreibung eines Interaktionselements einzublenden, wenn der Anwender den Mauszeiger über das Element bewegt. Eine dafür notwendige Methode *setToolTip* bietet auch die Klassenbibliothek. Daher wurde diese Methode für alle Einstellungsoptionen genutzt, damit der Anwender auch ohne Bedienungsanleitung zusätzliche Informationen erhält.

Abbildung 4.11 zeigt die Verwendung der Methode *setToolTip* am Beispiel der Erzeugung des Kontrollkästchens *Tasten farbig markieren*.

```
checkKeySelection = new QCheckBox(QObject::tr("Tasten farbig "
      "markieren"));
checkKeySelection->setToolTip(QObject::tr("Zur visuellen "
      "Unterstützung werden die zu\ndrückenden Tasten farbig "
      "markiert"));
```

Abb. 4.11: Programmauszug: *ToolTips*

4.2.2 Schreibtraining

Anhand des für den Anwender ersichtlichen Fensters zur Durchführung des Schreibtrainings (siehe Abbildung 4.12) werden Herangehensweisen für die Realisierung des Laufschrift, der virtuellen Tastatur und des Intelligenzkonzepts erläutert. Zudem werden Probleme geschildert, die sich bei deren Umsetzung ergeben haben.

Abb. 4.12: Screenshot: Widget *Schreibtraining*

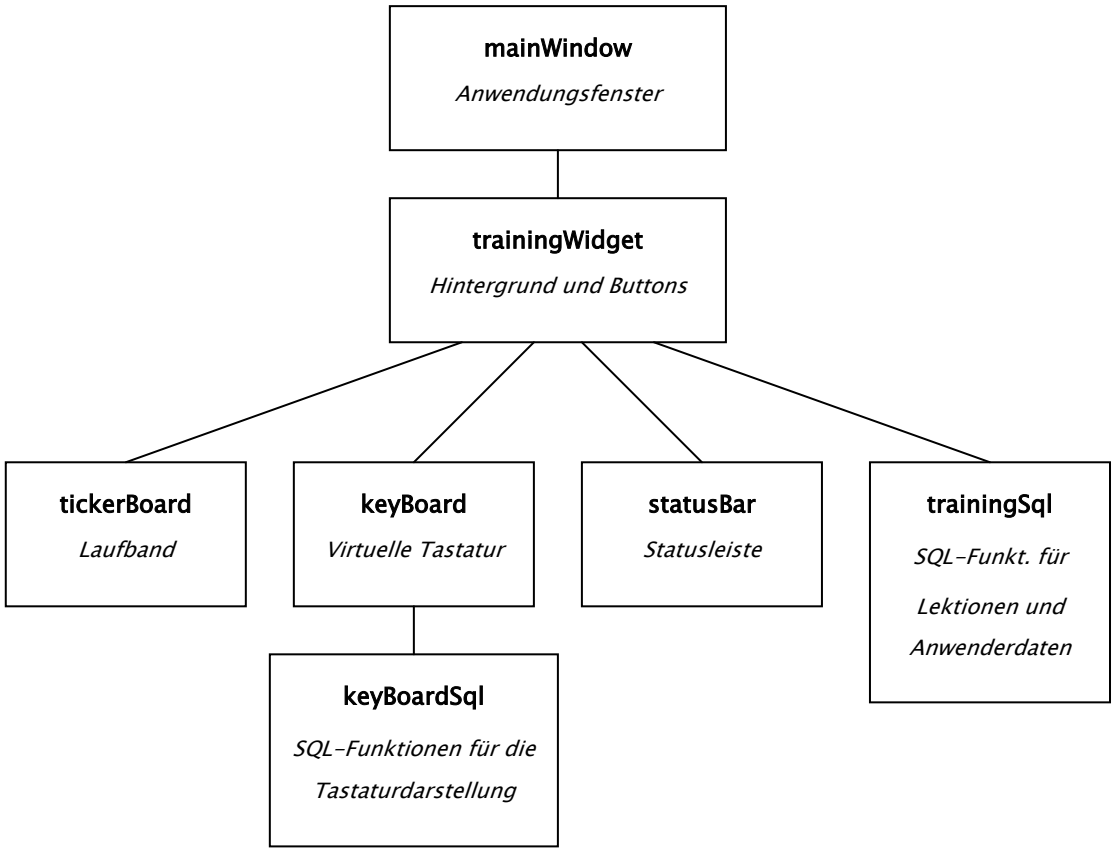


Abb. 4.13: Objektdiagramm: Widget *Schreibtraining*

Der Aufbau der erstellten Klassen wird im Objektdiagramm in der Abbildung 4.13 dargestellt (vgl. auch Kapitel 3.5.2.1):

4.2.2.1 Laufband

Damit sich die Schrift des Diktats gleichmäßig bewegt, bis sich das aktuelle Schriftzeichen in der Mitte des Laufbands befindet (siehe Abbildung 4.14), wurde ein Zeitgeber (Timer) implementiert.

Abb. 4.14: Screenshot: Laufband

Die Geschwindigkeit des Laufbandes lässt sich individuell festlegen und bestimmt die Zeitscheibe des Zeitgebers. Nach jedem Ablauf der Zeitscheibe wird eine Methode aufgerufen, die den Inhalt des Widget bei Bedarf verschiebt. Diese Methode ist wie folgt aufgebaut:

```
void TickerBoard::progress() {
      if (startFlag) {
            if (scrollOffset < lessonOffset) {
                  scrollOffset++;
                  scroll(-1, 0, contentsRect());
            }
            if ((lessonOffset - scrollOffset) > 160) {
                  scrollOffset += (lessonOffset - scrollOffset) - 160;
                  scroll(-((lessonOffset - scrollOffset) - 160), 0,
                        contentsRect());
            }
      }
}
```

Abb. 4.15: Programmauszug: Laufbandbewegung

Hierfür waren zwei Variablen notwendig. *scrollOffset* bestimmt die Verschiebung des Widget. *lessonOffset* bestimmt die Breite des bereits vergangenen Diktats und wird nach jedem neuen Schriftzeichen aktualisiert. Auf diese Weise kann gesteuert werden, wie weit sich das Laufband (noch) bewegen soll. Die Funktion

scroll verschiebt den Inhalt des Widget, der über die Methode *contentsRect* zurückgegeben wird.

Um zu verhindern, dass die Schrift des Diktats bei einem langsamen Laufband und einer hohen Anschlagzahl des Anwenders über den rechten Rand des Laufbands hinausläuft, wurde eine Grenze von 160 Pixel zum rechten Rand festgelegt. Die Grenze wurde so gewählt, dass immer einige Zeichen des Diktats sichtbar bleiben. Erreichen die Variablen *lessonOffset* und *scrollOffset* eine Differenz, die diese Grenze überschreitet, wird das Laufband automatisch beschleunigt (bzw. auf den Grenzwert gesetzt).

Ein weiterer darlegungsbedürftiger Vorgang stellt die Behandlung von Zeilenumbrüchen dar. Das Diktat wird vom Objekt der Klasse *TrainingWidget* regelmäßig, nach einer festen Anzahl Worte oder, bei dem Diktat von Sätzen, nach jedem Satz mit Zeilenumbrüchen bestückt.
Zeilenumbrüche sollen dem Anwender im Laufband so dargestellt werden, dass der Text nach einem Zeilenumbruch erst erscheint, wenn der Anwender das sichtbare Diktat mit der Bestätigungstaste abgeschlossen hat. Deswegen wurde eine Methode implementiert, die stets den aktuellen Text des Diktats bis zum nächsten Zeichen für den Zeilenumbruch abtrennt.

Der folgende Programmauszug veranschaulicht den Vorgang:

```
void TickerBoard::splitLesson() {
      QFontMetrics fm(tickerFont);
      lengthCompleteLesson = txtCompleteLesson.length();
      txtLessonSplited =
            txtCompleteLesson.split(QChar(TOKEN_NEW_LINE),
            QString::SkipEmptyParts, Qt::CaseSensitive);
      txtCurrentLesson = txtLessonSplited.at(counterRow)
            + QChar(TOKEN_NEW_LINE);
      lengthCurrentLesson = txtCurrentLesson.length();
      widthCurrentLesson = fm.width(txtCurrentLesson);
      update();
}
```

Abb. 4.16: Programmauszug: Text-Trennung nach Zeilenumbrüchen

Die Variable *txtCompleteLesson* beinhaltet den gesamten Text des Dikats, *lengthCompleteLesson* dessen aktuelle Zeichenanzahl. Nachdem der Text anhand des Trennzeichens für den Zeilenumbruch aufgeteilt wurde, bestimmt eine Zeilennummer (*counterRow*) den aktuell zu diktierenden Abschnitt. Das Zeichen für den Zeilenumbruch muss dabei erneut angehängt werden, da es durch die *split*-Methode verloren geht.

Anschließend können Zeichenzahl des aktuellen Textes (*length*-Methode) und Textlänge in Pixel (mit Hilfe der Klasse *QFontMetrics*) festgestellt werden. Diese Werte sind notwendig, um den Text im Laufband abzuarbeiten und zu bewegen.

4.2.2.2 Virtuelle Tastatur

Um das festgelegte Tastaturlayout aus 61 Tasten über die Klasse *KeyBoard* darzustellen, wurden Bilddateien (*PNG*-Grafiken) in Zusammenhang mit einem Zeichenereignis *QPaintEvent* verwendet. *PNG*-Dateien lassen sich von einem *QWidget* laden und über Methoden der Klasse *QPainter* darstellen.

Für die Tastaturdarstellung waren sieben verschiedene Tastengrößen in jeweils fünf unterschiedlichen Farben (für fünf Finger) notwendig. Dazu wurden

entsprechende Bilddateien angefertigt und im Programmverzeichnis *source/img/* abgelegt.

Diese Bilddateien werden von der Klasse *KeyBoard* geladen und zu einer Tastatur zusammengesetzt. Mit einer transparenten Bilddatei wird anschließend die Tastatur überdeckt und so nur die Beschriftung der einzelnen Tasten hinzugefügt (siehe Abbildung 4.17). Bislang existieren zwei Bilddateien für die Beschriftung: das Windows- und das Apple-Tastaturlayout.

Abb. 4.17: Tastaturdarstellung mit Bilddateien

Um das aktuelle Schriftzeichen in dieser Form der Tastaturdarstellung anzuzeigen, wurde die Tabelle *key_layouts* um Felder erweitert (siehe Abbildung 4.18), um jeder Taste eigene Koordinaten, eine Farbe und eine Form zuzuordnen. (Zudem wurde ein Feld *key_status* angelegt, das einen Text für die Darstellung in der Statusleiste bereitstellt.)

key_layouts	
PK	key_id
	key_left key_top key_color key_form key_status

Abb. 4.18: Erweiterte Struktur der Tabelle *key_layouts*

Nachdem dem aktuellen Schriftzeichen über das Objekt der Klasse *KeyboardSql* bis zu zwei Tasten zugeordnet wurden (vgl. Kapitel 3.3.1), können auf diese Weise die Eigenschaften der Tasten ausgelesen und die entsprechenden Bilddateien neu dargestellt werden. Abbildung 4.19 zeigt die virtuelle Tastatur mit zwei markierten Tasten, die das Schriftzeichen *D* repräsentieren.

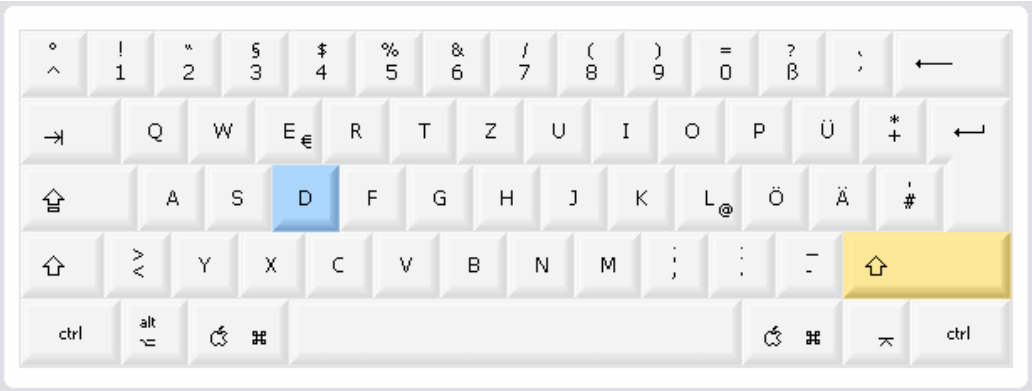


Abb. 4.19: Screenshot: Virtuelle Tastatur

Die virtuelle Tastatur empfängt weiterhin ein Tastenereignis *QKeyEvent*, wenn der Anwender eine Taste drückt. *Qt* bietet die Möglichkeit, dieses Ereignis auf verschiedene Weise auszulesen. Im Normalfall ist es notwendig, Drücken und Loslassen jeder Taste genau zu kontrollieren, um die Verwendung von Modifizierungstasten zu beachten. Eine gedrückte Modifizierungstaste wird erst zu einem Schriftzeichen, wenn eine weitere Taste (mit einem Schriftzeichen!) gedrückt wird, bevor die Modifizierungstaste losgelassen wird.

Qt bietet jedoch auch eine Eigenschaft *text* des Tastenereignisses an, die ermöglicht, direkt einen resultierenden Unicode-Text zu erhalten. Anhand dieser Eigenschaft lässt sich leicht feststellen, ob es sich um ein abgeschlossenes Schriftzeichen oder lediglich um eine Modifizierungstaste, wie zum Beispiel die Umschalttaste, handelt. Wurde ein Schriftzeichen eingegeben, kann dieses von der virtuellen Tastatur zur weiteren Verarbeitung an die Trainingsklasse (Klasse TrainingWidget) übermittelt werden.

Um sicherzustellen, dass ein Tastaturereignis immer im Objekt der Klasse *KeyBoard* ankommt, musste der Klasse ein Fokus zugewiesen werden, der nicht mehr zu anderen Steuerelementen wechseln kann. Dazu wurde die von der Klasse *QWidget* bereitgestellte Methode *setFocusPolicy* verwendet und der Parameter *Qt::StrongFocus* übergeben.

4.2.2.3 Realisierung des Intelligenzkonzepts

Für die Realisierung des Intelligenzkonzepts sind geeignete *SQL*-Befehle von großer Bedeutung. Sie werden vom Objekt der Klasse *TrainingSql* ausgeführt und sollen hier erläutert werden.

Nach der Darstellung von jedem neuen Zeichen im Laufband, aber auch nach jeder falschen Tasteneingabe des Anwenders müssen zugehörige Werte des entsprechenden Schriftzeichens in der Tabelle *user_chars* aktualisiert werden. Im Fall eines Tippfehlers müssen die Felder *user_char_target_errornum* und *user_char_mistake_errornum*, im Fall eines neuen Schriftzeichens das Feld *user_char_occur_num* inkrementiert werden.

Falls das Schriftzeichen noch nicht in der Tabelle existiert, muss es neu angelegt werden (Primärschlüssel). Um die Existenz eines Datensatzes in einer Tabelle zu überprüfen, werden im Normalfall nach einer Abfrage Methoden angewendet, die die Anzahl der Ergebnisse zählen. Leider unterstützt das Datenbanksystem *SQLite* keine dieser Methoden. Alternativ gibt es die Möglichkeit, ans Ende der Abfrage (zum letzten Datensatz) zu springen und dann festzustellen, an welcher Stelle sich der Zeiger (Index) befindet. Dies erfordert jedoch zusätzliche Rechenzeit und ist aufgrund des sehr häufig vorkommenden Aktualisierungsvorgangs für das Vorhaben unbrauchbar (siehe auch *SQL*-Optimierung in Kapitel 4.2.2.4).

Daher wurde ein anderer Weg gewählt, der die Überprüfung der Anzahl überflüssig macht. Der *SQL*-Befehl, der das Schriftzeichen in einem neuen Datensatz anlegt, wird jedes Mal ausgeführt, bevor entsprechende Felder des Datensatzes inkrementiert werden. Existiert also das Schriftzeichen noch nicht in der Tabelle, wird es erst angelegt und anschließend die Inkrementierung durchgeführt. Ist das Schriftzeichen aber bereits in der Tabelle vorhanden, meldet *SQLite* einen Fehler, der aber in diesem Fall nicht behandelt und übergangen wird.

Da es sich bei dem Vorgang um zwei *SQL*-Befehle direkt hintereinander handelt, wurde zusätzlich die Möglichkeit eingesetzt, den gesamten Vorgang in eine Transaktion einzubinden. Befehle in einer Transaktion werden von der Datenbank erst durchgeführt, nachdem die Transaktion beendet wurde. Die dafür nötigen *SQL*-Befehle lauten *BEGIN* und *COMMIT* und beschleunigen eine Verarbeitung mehrerer Befehle hintereinander erheblich.

Transaktionen brechen jedoch den gesamten Vorgang ab, wenn einer der eingeschlossenen Befehle fehlschlägt. Zudem lassen sich einzelne *SQL*-Befehle nicht mehr über Methoden unter C++ prüfen. Da aber im jetzigen Fall das Anlegen eines neuen Datensatzes fehlschlagen darf, wurde eine Möglichkeit angewandt, *SQLite* direkt im *SQL*-Befehl mitzuteilen, wie im Fehlerfall vorzugehen ist. Diese Möglichkeit wird in der *SQLite*-Dokumentation *conflict-algorithm* [WV Sqlite] genannt. Der verwendete Befehl für den vorliegenden Fall lautet *IGNORE*. So kann bei jeder Aktualisierung der gesamte Vorgang (Datensatz anlegen und Wert inkrementieren) durchgeführt werden, fehlerhafte Befehle werden dann von *SQLite* einfach verworfen.

Die Methode für den Aktualisierungsvorgang ist wie folgt aufgebaut:

```
bool TrainingSql::updateUsertable(QChar unicodechar, QString
      columnname) {
      QSqlQuery query;
      QString charToString = QString::number(unicodechar.unicode());

      query.exec("BEGIN;");
      query.exec("INSERT OR IGNORE INTO user_chars VALUES("
            + charToString + ",0,0,0);");
      query.exec("UPDATE user_chars SET " + columnname
            + "=" + columnname + "+1 WHERE user_char_unicode = "
            + charToString + ";");
      if (!query.exec("COMMIT;")) {
            return false;
      }
      return true;
}
```

Abb. 4.20: Programmauszug: Aktualisierung der Anwenderdaten

Parameter sind das Schriftzeichen und der zu inkrementierende Feldname. Das Schriftzeichen wird vor der Verwendung in einen Unicode-Wert umgewandelt.

Zu Beginn einer Lektion wird stets der erste Text einer Lektion diktiert. Er besitzt eine feste Identifikationsnummer in der Form <Nummer der Lektion>000. Er soll den Anwender in die neuen Schriftzeichen einführen und muss nicht unbedingt aus sinnvollen Worten bestehen. Für jede weitere Aktualisierung des Diktats werden die Texte in Abhängigkeit von den Tippfehlern des Anwenders ausgewählt und dem Laufband übergeben. Über die im Kapitel 3.3 festgelegte Tabellenstruktur und die Beziehungen der Tabellen lässt sich dieser Vorgang ebenfalls mit Hilfe von *SQL*-Befehlen durchführen.
Dazu wurden im ersten Schritt die Fehler pro Schriftzeichen aus der Tabelle *user_chars* gewichtet und absteigend nach der Gewichtung sortiert, wie folgender SQL-Befehl veranschaulicht:

```
SELECT user_char_unicode, (user_char_target_errornum * 100) /
      user_char_occur_num AS user_char_weighted FROM user_chars ORDER
      BY user_char_weighted DESC;
```

Abb. 4.21: SQL-Befehl: Fehlergewichtung

Anschließend lassen sich die Unicode-Werte der am häufigsten falsch getippten Schriftzeichen auslesen.
Über die Beziehungen der Tabellen *lesson_content* und *lesson_analysis* werden dann in einem zweiten Schritt passende Worte bzw. Sätze herausgefiltert.

Folgender *SQL*-Befehl zeigt die Filterung exemplarisch für ein Schriftzeichen (Unicode-Wert 34) und die erste Lektion:

```
SELECT lesson_content.content_id, lesson_content.content_text FROM
      lesson_content LEFT JOIN lesson_analysis ON
      lesson_content.content_id = lesson_analysis.analysis_content AND
      lesson_content.content_lesson = 1 AND (lesson_content.content_id
      % 1000) <> 0 GROUP BY lesson_content.content_id ORDER BY
      analysis_char_34 DESC;
```

Abb. 4.22: SQL-Befehl: Textfilterung

Um zu verhindern, dass der erste Text einer Lektion in die Abfrage mit einfließt, wurde die dazugehörige Identifikationsnummer ausgeschlossen. Die Identifikationsnummern der Texte erlauben es, den Ausschluss mit Hilfe des Restwertes einer ganzzahligen Division zu bestimmen.

Bei der Filterung spielen noch weitere Kriterien eine Rolle, zum Beispiel um zu bestimmen, wann ein Text wieder verwendet werden darf, sollte er bereits diktiert worden sein. Diese Kriterien wurden über Konstanten festgelegt, deren Aufgaben im Kapitel 4.2.2.5 genauer erläutert werden.

4.2.2.4 SQLite-Optimierung

SQLite erfordert für jeden Zugriff auf die Datenbank ebenfalls einen Zugriff auf die Festplatte des Computers. Die Datenbank arbeitet mit einer Datenbankdatei, die sich nicht fortlaufend im Arbeitsspeicher befindet.

Bei der Implementierung stellte sich heraus, dass aufgrund der häufigen Zugriffe während des Schreibtrainings einige Elemente wie Laufband und virtuelle Tastatur verzögert reagierten. Um dieses Problem zu umgehen, wurde nach Optimierungen für den Datenbankzugriff gesucht. Nur *SQL*-Befehle zu optimieren, erwies sich als nicht ausreichend. Auf Fehlerprüfungen zu verzichten, wäre nicht sinnvoll. *SQLite* bietet jedoch eine Einstellung, den synchronen Ablauf

der Zugriffe auf die Datenbankdatei zu unterbinden. Sie verhindert, dass der Programmablauf unterbrochen wird, bis der Zugriff beendet wurde.
Über folgenden *SQL*-Befehl wird die synchrone Bearbeitung der Datenbank deaktiviert:

```
PRAGMA synchronous=OFF;
```

Abb. 4.23: SQL-Befehl: Synchronen Ablauf unterbinden

Nachteilig an der Einstellung ist, dass bei einem Absturz des Systems während eines Zugriffs die Datenbankdatei zerstört und damit nicht mehr lesbar werden könnte. Dies wurde aber, zugunsten der Performance, in Kauf genommen.

Diese Einstellung war nur für das Schreibtraining notwendig, in allen anderen Fällen arbeitet die Datenbank synchron. Die Einstellung könnte durch Einsatz schnellerer Prozessoren und Festplatten oder eines anderen Datenbanksystems überflüssig werden, daher wurde im Quelltext explizit darauf hingewiesen.

4.2.2.5 Konstanten

Für den Schreibtrainer wurden einige Konstanten definiert, die bestimmte Programmvorgänge und Bezeichnungen beeinflussen. Diese sind in einer Datei (*defines.h*) ausgelagert, um sie leicht anpassen zu können. Neben Konstanten zum Beispiel für Programmversion, Programmname, Datenbank-Dateiname, Serveradresse für die Aktualisierungsfunktion und Standardwerte für die Einstellungsoptionen wurden hier auch Werte festgelegt, die das Schreibtraining beeinflussen. Da diese Werte eine besondere Bedeutung haben, sollen sie nachfolgend genauer erläutert werden:

- NUM_TOKEN_UNTIL_REFRESH

  Dieser Wert bestimmt, wieviele Zeichen des Diktats, die noch nicht diktiert wurden, mindestens verbleiben müssen. Wird dieser Wert erreicht, werden neue Texte für das Diktat erzeugt (angefordert).

- NUM_TOKEN_UNTIL_NEW_LINE

  Solange nur Worte und keine ganzen Sätze angefordert werden, entscheidet diese Konstante darüber, wann ein Zeilenumbruch gesetzt wird. Wird dieser Wert, ausgehend vom letzten Zeilenumbruch, durch die Zeichenanzahl vorhandener und neuer Worte überschritten, wird ein Zeichen für den Zeilenumbruch an das Diktat angefügt.

- TOKEN_NEW_LINE

  Dieser Wert bestimmt das Zeichen, das für den Zeilenumbruch verwendet wird. Es wird in hexadezimaler Schreibweise angegeben, damit alle Zeichen der Unicode-Familie verwendet werden können.

- NUM_TEXT_UNTIL_REPEAT

  Damit Worte und Sätze nicht zu oft wiederholt werden (falls die Fehlerquoten hintereinander den gleichen Text anfordern), bestimmt diese Konstante, wie viele Texte (Worte oder Sätze) mindestens zwischen der Wiederholung liegen müssen. Über eine Warteschlange nach dem FIFO-Prinzip, die die Größe NUM_TEXT_UNTIL_REPEAT besitzt, lässt sich dieser Vorgang realisieren.

- BORDER_LESSON_IS_SENTENCE

  Der Schreibtrainer muss, besonders für die Verwendung der Zeilenumbrüche wissen, ab welcher Lektion ganze Sätze diktiert werden. Diese Konstante legt die dafür zugehörige Lektionennummer fest.

- LAST_LESSON

  Um in der letzten Lektion die Sätze aller Lektionen (ab BORDER_LESSON_IS_SENTENCE) in das Diktat einzubeziehen, wird über diese Konstante die Nummer der letzten Lektion festgelegt.

4.2.3 Ergebnisse

Die Ergebnisse werden dem Anwender über zwei Register mit Tabellen präsentiert (Abbildung 4.24).

Abb. 4.24: Screenshot: Widget *Ergebnisse*

Abbildung 4.25 zeigt das dazugehörige Objektdiagramm.

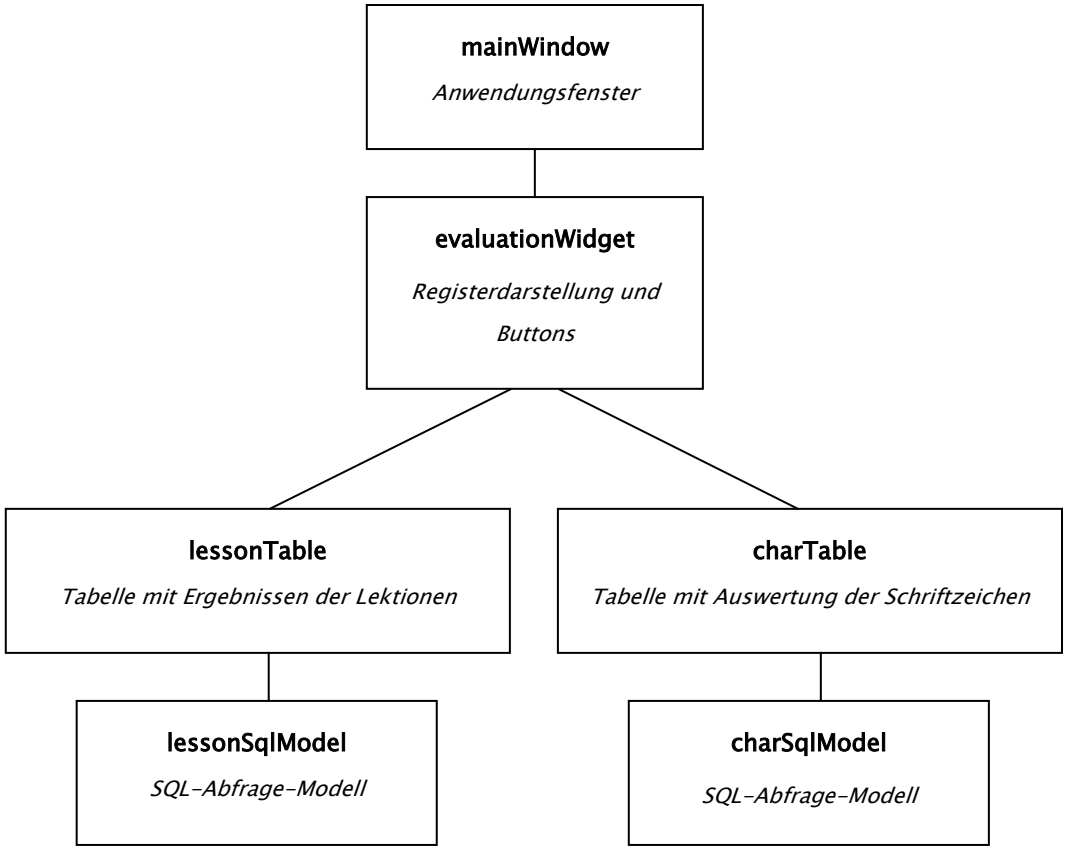


Abb. 4.25: Objektdiagramm: Widget *Ergebnisse*

Die Register und die Schaltfläche werden vom Objekt der Klasse *EvaluationWidget* erzeugt. Jedes Register erhält wiederum ein eigenes Objekt, das erste von der Klasse *LessonTable*, das zweite von der Klasse *CharTable*.
Um die Ergebnisse in Form von Tabellen darzustellen, wurde die *Qt*-Klasse *QSqlQueryModel* verwendet. Über einen *SQL*-Befehl werden Inhalte der Datenbank automatisch in Form einer Tabelle angezeigt, und es lassen sich einzelne Zellen formatieren.

Für die Ergebnisse der Lektionen im ersten Register wurde folgender *SQL*-Befehl übergeben:

```
SELECT user_lesson_lesson, user_lesson_timestamp, user_lesson_timelen,
      user_lesson_tokenlen, user_lesson_errornum,
      ROUND(user_lesson_strokesnum / (user_lesson_timelen / 60.0), 0)
      AS user_lesson_cpm, ROUND(((user_lesson_strokesnum - (20 *
      user_lesson_errornum)) / (user_lesson_timelen / 60.0)) * 0.4, 0)
      AS user_lesson_grade, user_lesson_id FROM user_lesson_list ORDER
      BY user_lesson_lesson DESC, user_lesson_timestamp DESC;
```

Abb. 4.26: SQL-Befehl: Lektionenergebnisse

Aus der Tabelle *user_lesson_list* werden alle Einträge dargestellt. Bei der Abfrage werden aber auch zwei neue Felder erstellt. Das erste, user_lesson_cpm, errechnet die Anschläge pro Minute für die jeweilige Lektion. Das zweite Feld, *user_lesson_grade*, errechnet aus den gegebenen Werten eine Bewertung aus, damit dem Anwender ein besserer Vergleich der absolvierten Lektionen angeboten wird. Die dazugehörige Formel der Bewertung lautet:

$$\frac{(\text{Anschläge} - (20 \times \text{Fehleranzahl}))}{\text{Dauer in Minuten}} \times 0{,}4$$

Die Konstante 0,4 wurde gewählt, um eine Spanne von 0 bis ca. 100 Punkten zu Verfügung zu stellen, die Ergebnisse von Anfängern mit geringer Anschlagzahl und vielen Fehlern bis hin zu sehr guten Ergebnissen mit hoher Anschlagzahl (ca.

300 Anschläge pro Minute [WV Tastaturschreiben]) und ohne Fehler abdeckt. Zudem wird eine hohe Fehlerzahl schlechter bewertet, als eine geringe Anschlagzahl, indem für jeden Fehler 20 Anschläge abgezogen werden.

Der *SQL*-Befehl für das zweite Register erzeugt eine Liste aller Schriftzeichen, die im Laufe aller Lektionen trainiert wurden. Zu jedem Schriftzeichen wird zudem die Fehlerquote errechnet und angezeigt. Der *SQL*-Befehl lautet:

```
SELECT user_char_unicode, user_char_unicode AS user_char_unicode_int,
      user_char_target_errornum, user_char_mistake_errornum,
      user_char_occur_num, ROUND((user_char_target_errornum * 100.0) /
      user_char_occur_num, 0) AS user_char_weighted FROM user_chars
      ORDER BY (user_char_target_errornum * 100.0) /
      user_char_occur_num DESC;
```

Abb. 4.27: SQL-Befehl: Schriftzeichenergebnisse

Die zugehörige Formel der Fehlerquote lautet:

$$\frac{\text{Ist-Fehler}}{\text{Vorkommen}} \qquad \text{Bedingung: Vorkommen} > 0$$

4.2.4 Auszug der verwendeten Programmtexte

Das in Kapitel 2.2.2 vorgestellte Werkzeug *Qt Linguist* wurde für den Schreibtrainer eingesetzt. Es ermöglicht, einen Auszug aller verwendeten Programmtexte zu erstellen, um diese modifizieren und für eine Internationalisierung übersetzen zu können. Voraussetzung dafür war, alle Programmtexte mit dem Aufruf *QObject::tr* zu versehen, der von *Qt Linguist* erkannt und umgesetzt werden kann.

Mit Hilfe des Befehls *lupdate* wurde nach der Implementierung eine Datei *tipp10.ls* erstellt (siehe Abbildung 4.28).

```
lupdate tipp10.pro
```

Abb. 4.28: Textauszug mit *Qt Linguist*

Um die Datei einfach bearbeiten zu können, stellt das Programm *Qt Linguist* eine grafische Oberfläche bereit, mit der sich der Auszug editieren lässt und gegebenenfalls weitere Dateien mit anderen Sprachen erstellt werden können.
Um die erzeugten Dateien in die Software einzubinden, müssen Verweise in der Projektdatei (*tipp10.pro*) hinterlegt werden. Da es sich bislang nur um eine Version der Texte handelt, muss nur die Datei *tipp10.ls* eingebunden werden. Dies geschieht wie folgt:

```
TRANSLATIONS = tipp10.ts
```

Abb. 4.29: Verweis auf den Textauszug

4.2.5 Texte der Lektionen erzeugen

Für die Erzeugung der Texte wurde mit Hilfe der vorgegebenen Arbeitsschritte aus Kapitel 3.6.3 ein Werkzeug realisiert. Es arbeitet mit der Skriptsprache *PHP* und *HTML*-Formularen, um Textabschnitte zu übernehmen und anschließend zu filtern. Über eine *MySql*-Datenbank werden die resultierenden Texte gesammelt. Auf diese Weise ließen sich bequem Texte der Internetseite *http://www.digbib.org* in ein Formular kopieren und per Knopfdruck analysieren.

Die entstandenen Texte konnten aber nicht ohne manuelle Nachbearbeitung einfach in die Datenbank des Schreibtrainers übernommen werden, da aufgrund der heute geltenden Rechtschreibung Änderungen notwendig waren. Zusätzlich

mussten einige Texte angepasst oder entfernt werden, wenn sie zu sehr vom gewöhnlichen Sprachgebrauch abwichen.
Der Datenbank des Schreibtrainers wurde für die erste Veröffentlichung eine ausreichende Anzahl an Texten zugewiesen, um die Funktion der Software zu gewährleisten und den Zweck des Intelligenzkonzepts zu demonstrieren. In Zukunft soll jedoch der Umfang der Texte erhöht werden.

Das entstandene Werkzeug für die Filterung der Texte ist auf der Internetseite *http://www.tipp10.de/textanalyse/* einsehbar.

4.3 Test

Während und nach der Implementierung wurden verschiedene Tests durchgeführt, um die Funktionalität der Software zu prüfen. Sie lassen sich unterscheiden in Programmtests, um einzelne Programmteile und Methoden zu testen, und in Anwendertests, bei denen das veröffentlichte Programm von Personen aus der Zielgruppe angewendet wird.

4.3.1 Programmtest

Die Programmtests sollten vier mögliche Fehlerquellen untersuchen:

1. Liefert die Methode *text* des Tastaturereignisses korrekte Schriftzeichen zurück?
2. Werden die zu einem Schriftzeichen entsprechenden Tasten auf der virtuellen Tastatur richtig angezeigt?
3. Zeigt die Anwendung im Fall eines fehlerhaften Datenbankzugriffs passende Fehlermeldungen an?
4. Werden die gewünschten Texte vom Intelligenzkonzept bereitgestellt?

Zu 1.

Die Klasse *KeyBoard* empfängt ein Tastaturereignis über die Methode *keyPressEvent*. *keyPressEvent* wandelt das Ereignis über die Methode *text* des Tastaturereignisses in ein Schriftzeichen (*QChar*) um und sendet das Zeichen mit Hilfe eines Parameters über das Signal *keyPressed* aus. Der Parameter des Signals soll insbesondere auf den Fall getestet werden, bei dem Sonderzeichen oder Schriftzeichen, für die zwei Tasten verwendet werden, eingegeben werden.

Um *Unit*-Tests durchzuführen, bietet die Klassenbibliothek *Qt* eine eigene Klasse *QTestLib*. Mit ihr lassen sich Methoden aufrufen und die Rückgabewerte oder

Signale mit festgelegten Werten vergleichen. Zudem bietet sie die Möglichkeit, Maus- und Tastaturereignisse zu simulieren.

Für den Test der Klasse *KeyBoard* wurde eine eigene Klasse *UnitTest* erstellt. Sie simuliert über die *QTestLib*-Methode *keyClicks* die Eingabe mehrerer Zeichen auf der Tastatur und empfängt die resultierenden Signale über die *Qt*-Klasse *QSignalSpy*. Im Anschluss überprüft sie die empfangenen Signale mit den zuvor eingegebenen Schriftzeichen.

Zu 2.

Eine Überprüfung der farbigen Tastenmarkierung der virtuellen Tastatur mit Hilfe einer Testklasse ist nicht unbedingt sinnvoll. Eine Klasse, die alle Tasten automatisiert testet, müsste mit den Eigenschaften jeder Taste bestückt werden. Dieser Vorgang wäre für sich gesehen schon sehr fehleranfällig. Daher wurde ein Diktat erzeugt, das alle Schriftzeichen beinhaltet, die im Schreibtrainer verwendet werden. Anschließend konnte durch ein Test-Schreibtraining die Markierung aller Tasten manuell überprüft werden, insbesondere für Schriftzeichen, die zwei Tasten zugleich markieren.

Zu 3.

QTestLib in Zusammenhang mit einer Datenbank zu verwenden, ist leider nicht möglich. Um fehlerhafte Datenbankzugriffe trotzdem zu simulieren, wurden vier Testfälle vorgesehen, die Fehler im Programm provozieren. Für jeden Testfall wurde anschließend die Reaktion der Software und die Richtigkeit der Fehlermeldungen überprüft.

- Der erste Testfall simuliert eine fehlende Datenbankdatei. Dazu wurde die Datei entfernt und der Schreibtrainer ohne Datenbank kompiliert.

- Der zweite Testfall simuliert eine blockierte Datenbankdatei. Dazu wurde die Datenbankdatei mit Hilfe der Software *SQLite Database Browser* modifiziert und gleichzeitig vom Schreibtrainer verwendet.
- Um fehlerhafte *SQL*-Befehle zu testen, wurden im dritten Testfall absichtlich Fehler in die *SQL*-Abfragen eingebaut.
- Der vierte Testfall simuliert eine vorhandene Datenbankdatei, jedoch mit fehlerhaften und fehlenden Daten.

Zu 4.

Um zu testen, ob der Schreibtrainer richtig reagiert und die passenden Texte anfordert, wurden Testfälle entwickelt. Nachfolgend wird die Vorgehensweise exemplarisch an einem Testfall erläutert:

Vorbedingungen:

- Zu Beginn sind keine Benutzerdaten in der Datenbank vorhanden
- Die Warteschleife (Konstante NUM_TEXT_UNTIL_REPEAT) hat die Größe 3
- Die Konstante NUM_TOKEN_UNTIL_REFRESH hat die Größe 2, damit Texte erst angefordert werden, wenn nur noch zwei Zeichen des Diktats verbleiben
- Es werden fünf Worte für Lektion 1 in der Datenbank hinterlegt:

content_id	content_text	content_lesson
1000	anna	1
1001	ananas	1
1002	adam	1
1003	apfel	1
1004	nass	1
1005	anatomie	1

Tabelle 4.1 Testinhalt der Tabelle */esson_content*

Test durchführen:

Nach dem Start des Diktats für Lektion 1 wird zuerst der Text *anna* angezeigt. Nun wird die Zeichenfolge *nnnanaana* auf der Tastatur eingegeben. Die Eingabe bewirkt, dass für das Schriftzeichen *a* drei Tippfehler und für das Schriftzeichen *n* zwei Tippfehler gezählt werden sollen.

Ergebnisse überprüfen:

Nach der Texteingabe muss die Tabelle *user_chars* wie folgt mit zwei Datensätzen gefüllt sein:

user_char_unicode	user_char_target_errornum	user_char_mistake_errornum	user_char_occur_num
110	3	2	2
97	2	3	2

Tabelle 4.2 Testergebnisse in der Tabelle *user_chars*

Nach der festgelegten Vorgehensweise des Intelligenzkonzepts müssen, wenn alles anschließend richtig getippt wird, die Worte der Datenbank in folgender Reihenfolge im Diktat erscheinen:

anna ananas anatomie adam ananas anatomie adam …

(Das Wort *anna* ist für den Start des Diktats festgesetzt und wird nicht in die Aktualisierung einbezogen.)

4.3.2 Anwendertest

Vollständige Anwendertests benötigen viel Zeit, da Personen verschiedener Zielgruppen ihre Erfahrungen über die gesamte Trainingsdauer berichten müssen. Daher konnten diese bislang nur exemplarisch an Lektionen des Schreibtrainings durchgeführt werden. Einige Auswirkungen der Testresultate auf den Entwurf und die Implementierung werden nachfolgend in kurzer Form aufgelistet:

- Der erste Text jeder Lektion ist festgelegt und wird in die dynamische Aktualisierung des Diktats nicht einbezogen.

- Die Aufteilung der Lektionen über die Häufigkeit der Schriftzeichen im Sprach- und Schriftgebrauch vornehmen.
- Das Schriftzeichen . (Punkt) in einer früheren Lektion (als die Häufigkeitsanalyse von *RISTOME* festlegt) diktieren, um ganze Sätze zu ermöglichen.
- Die Beendigung der Lektion über eine Schaltfläche als Abbruch (ohne Nachfrage) behandeln, wenn das Diktat noch nicht begonnen hat.
- Bewertung der Lektionenergebnisse einführen, um den direkten Vergleich zu ermöglichen

Andere Resultate wurden für die Weiterentwicklung des Systems eingeplant und werden im Kapitel Ausblick (5.1) genauer erläutert.

4.4 Veröffentlichung

4.4.1 Bedienungsanleitung

Die Bedienungsanleitung des Schreibtrainers findet sich gesondert im Anhang dieser Diplomarbeit. Da sich unter den im Pflichtenheft festgesetzten Zielgruppen auch Anfänger im Bereich der PC-Bedienung befinden, wurde eine möglichst detaillierte Beschreibung angestrebt. Erfahrene Anwender pflegen im Allgemeinen, sich interessante Bereiche gezielt herauszusuchen, werden aber voraussichtlich detaillierte Erläuterungen wie den Installationsverlauf übergehen.
Die Bedienungsanleitung befindet sich im *PDF*-Format ebenfalls auf der beigelegten CD-ROM.

4.4.2 Windows-Installer

Um unter Windows eine Installationsroutine anzubieten, die alle notwendigen Dateien des Schreibtrainers auf dem Zielsystem installiert und eine Verknüpfung im Startmenü erstellt, wurden zwei Werkzeuge verwendet. Das erste Werkzeug *Dependency Walker* (kurz *depends*) wird in der Dokumentation von *Qt* empfohlen. Es ist kostenlos und kann anhand einer ausführbaren Datei feststellen, welche Systemdateien von dieser verwendet werden und für die Auslieferung der Software relevant sind. Aufgrund der statischen Kompilierung (siehe Kapitel 4.1.2) ist die Anzahl der Dateien des Schreibtrainers gering.

Alle für die Auslieferung notwendigen Dateien sind in folgender Tabelle aufgeführt:

Dateiname	Beschreibung
tipp10.exe	Ausführbare Datei (Programmdatei)
tipp10.db	Datenbankdatei
mingwm10.dll	Systemdatei für den *MinGw*-Compiler
licence.txt	Einfache Textdatei mit Hinweisen auf die *GNU*-Lizenz (*GPL*) (Empfohlene Vorgehensweise in den *GNU Coding Standards*)

Tabelle 4.3 Dateien für die Veröffentlichung

Um diese Dateien in einer gepackten und ausführbaren Installationsdatei weitergeben zu können, die nach der Ausführung alle Dateien auf das Zielsystem kopiert, wurde ein zweites Werkzeug mit der Bezeichnung *Inno Setup Compiler* verwendet. Es ist kostenlos, unterstützt alle bekannten Windowsversionen und ermöglicht einen Deinstallation-Eintrag (*Uninstall*). Über ein eigenes Skriptformat lassen sich alle Eigenschaften und Versionsinformationen der Installationsroutine festlegen.

Das *Inno Setup Compiler* Installationsskript *tipp10_setup.iss* findet sich auch auf der beigelegten CD-ROM.

4.4.3 Verzeichnisstruktur der CD-ROM

CD-ROM

conf	Konfigurationsdateien für *DoxyGen* und *InnoSetup*
doc	Bedienungsanleitung und Quelltext-Dokumentation
setup	Installationsdateien TIPP10 (Veröffentlichung)
software	Verwendete Softwarepakete
source	Quelltext der Software (Unterverzeichnisse vgl. Kap. 4.1.5)
update	Beispieldateien für die Aktualisierungsfunktion

5. Schlussfolgerung

5.1 Ausblick

Nachfolgend aufgelistete Ideen und Verbesserungen sollen einen Einblick geben, wie die weitere Entwicklung des Schreibtrainers ablaufen kann. Dabei werden auch Punkte aufgeführt, die sich während der Implementierung als weniger sinnvoll und verbesserungswürdig herausgestellt haben.

- Veröffentlichung auf verschiedenen Plattformen

  Aufgrund der Multiplattform-Fähigkeit der Klassenbibliothek *Qt* kann der Schreibtrainer auch auf Macintosh- oder Linuxsystemen veröffentlicht werden.

- Feste Einbindung der Tastaturlayouts

  Die Einstellungsmöglichkeit für den Anwender, zwischen zwei Tastaturlayouts (Windows und Macintosh) wechseln zu können, stellt eine überflüssige Anwenderfunktion dar. Es ist nicht gebräuchlich, auf einem Windows-PC eine Macintosh-Tastatur zu verwenden, ebensowenig wie auf einem Macintosh eine Windows-Tastatur. Daher soll die Wahl des Layouts ausschließlich über Konstanten erfolgen, die vor der Veröffentlichung auf einer bestimmten Plattform gesetzt werden.

  Für Testzwecke ist die Einstellung dagegen gut geeignet. So können die Unterschiede der Darstellung auf einem beliebigen System (vorerst Windows) getestet werden.

- Texte für das Diktat erweitern

  Die Texte für das Diktat können bereits über die Aktualisierungsfunktion erweitert werden. Daher können Umfang und Auswahl der Texte erweitert werden, um so das dynamische Verhalten und den Lernerfolg im Schreibtraining zu verbessern. Für diesen Zweck könnten auch neue Quellen für die Herkunft der Texte ausgemacht werden.

- Intelligenzkonzept verbessern

  Das Intelligenzkonzept ist so konzipiert, dass immer wieder die gleichen Texte angefordert werden, solange sich die Fehlerquoten der Schriftzeichen nicht ändern. Dies wird derzeit über eine Warteschlange (vgl. Kapitel 4.2.2.5) unterbunden. Eine zufällige Reihenfolge aller Texte, die für eine bestimmte Fehlerquote geeignet sind, würde diesen Sachverhalt noch einmal verbessern, allerdings ist dies sehr schwer zu realisieren. Eine andere Möglichkeit wäre es, für jedes Schriftzeichen eine eigene Warteschlange zu verwenden. Sie würden gewährleisten, dass auch Texte einer Fehlerquote diktiert werden, die durch ihre Identifikationsnummer am Ende der Abfrage angesiedelt sind.

- Intelligenzkonzept erweitern

  Das Intelligenzkonzept des Schreibtrainers kann auf verschiedene Weise ausgebaut werden. Mehr Tippfehlervarianten des Anwenders könnten beachtet, ausgewertet und in das Konzept einbezogen werden. Bereits durch Zählungen vorgesehen wird die Erweiterung auf Ist-Fehler (vgl. Kapitel 3.2.2). Zudem könnte die Art der Textgenerierung verändert werden, indem zum Beispiel Sätze aus einzelnen Wörtern zusammengesetzt werden (vgl. Kapitel 3.2.3).

- Zeit und Zeichenanzahl im Schreibtraining anzeigen

  Damit der Anwender während des Schreibtrainings die bis zum Ablauf der Lektion ausstehende Zeit erkennen kann, soll zusätzlich eine Zeit- bzw. Zeichenzählung in der Statusleiste angezeigt werden.

- Auswertung grafisch darstellen

  Zur besseren Übersicht könnten Werte der Ergebnistabelle in Form von grafischen Balken angezeigt werden. Alle Lektionenergebnisse in einem Verlaufsdiagramm darzustellen, wäre ebenfalls sinnvoll.

- Bewertung individuell an die Lektionen anpassen

  Die Bewertung wird derzeit auf jede Lektion gleichermaßen angewendet. Um Anfänger zu motivieren, könnte eine Bewertung eingeführt werden, die jede Lektion individuell behandelt und an die Schwierigkeitsstufe der Lektion angepasst ist.

- Aktualisierungsfunktion verbessern

  Die Aktualisierungsfunktion der Anwendung analysiert bislang bei jeder Aktualisierung die vorhandenen Texte und erzeugt eine Analysetabelle. Um die Redundanz dieses Vorgangs zu vermeiden (jede Kopie der Anwendung vollzieht den gleichen Ablauf), könnte er ausgelagert und der Administration überlassen werden. Die passende Analysetabelle kann dann direkt mit in die Aktualisierungsdatei übernommen werden.

- Aktualisierungsfunktion ausweiten

  Die Aktualisierungsfunktion wird bislang nur auf die Datenbank angewendet. Sie könnte erweitert werden, damit Programmteile, zum Beispiel das Intelligenzkonzept, verändert werden könnten, ohne eine neue

Programmversion zu veröffentlichen. Im Normalfall wird dies durch Plugins (meist *DLL*-Dateien) realisiert. Auf das statische Kompilieren der Software müsste dann jedoch verzichtet werden.

- Internationalisierung

  Der Schreibtrainer bietet bereits die Möglichkeit, verschiedene Tastaturlayouts und Texte in den Schreibtrainer einzubinden. Desweiteren wurden die für *Qt Linguist* nötigen Aufrufe (*QObject::tr*) für alle Programmtexte verwendet. Daher kann der Schreibtrainer mit wenig programmtechnischen Änderungen für andere Nationalitäten implementiert und angeboten werden.

5.2 Fazit

Im Rahmen dieser Diplomarbeit ist der 10-Finger-Schreibtrainer TIPP10 entstanden. In der Planungsphase wurde ein Pflichtenheft erstellt und eine passende Entwicklungsumgebung ausgewählt. Ein Entwurf des Intelligenzkonzepts und der damit verbundenen Programmteile Datenbank, Benutzungsoberfläche, Ablauf des Schreibtrainings und Erstellung der Texte für das Diktat ermöglichte, die Implementierung nach einer festgelegten Struktur durchzuführen. In der Implementierungsphase wurden außerdem Erweiterungen des Entwurfs vorgenommen und durch Tests erkannte Probleme gelöst.

Es war vorauszusehen, dass es sich bei der Software vorerst um einen Prototypen handeln würde. Ein wirklich ausgereiftes System ist nur möglich, indem Erfahrungswerte bei der Implementierung und durch Anwendertests nach der Veröffentlichung in die Weiterentwicklung der Software einfließen. Besonders das Intelligenzkonzept kann auf diese Weise weiter ausgebaut und verbessert werden.

Zu diesem Zweck wurde eine Internetseite eingerichtet, die unter der Adresse *http://www.tipp10.de* erreichbar ist. Sie soll zum einen dazu dienen, den Quelltext anderen Software-Entwicklern zur Verfügung zu stellen, um so mehr Strategien und Ideen für die durch die objektorientierte Programmierung zahlreichen möglichen Lösungsansätze zu finden.
Zum anderen soll der ausführbare Schreibtrainer TIPP10 auf der Internetseite mit einer Installationsroutine angeboten werden, damit er durch viele Personen verwendet werden kann und deren Erfahrungen in die Weiterentwicklung einfließen können.

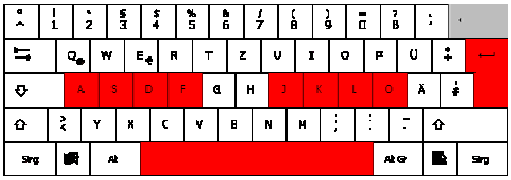
Anhang

A. Lektionenübersicht

Die Aufteilung der Lektionen wird anhand einer deutschen Windows-Tastatur dargestellt:

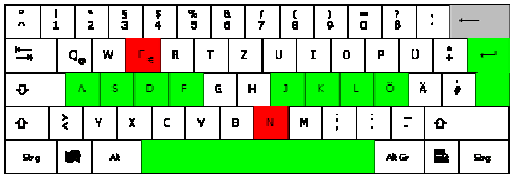
Lektion 1

Zeichen: asdf jklö (und Zeilenumbruch)



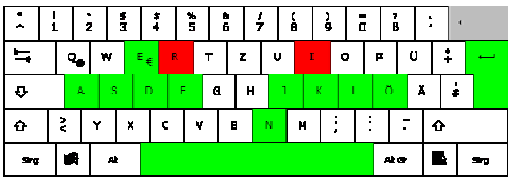
Lektion 2

Zeichen: en



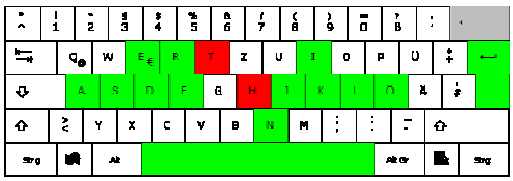
Lektion 3

Zeichen: ri



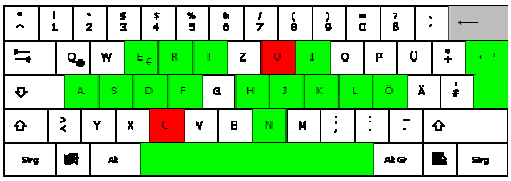
Lektion 4

Zeichen: th



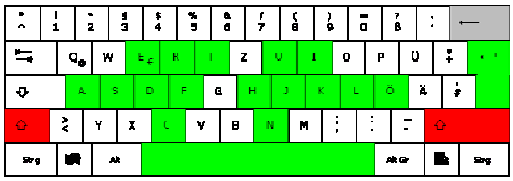
Lektion 5

Zeichen: cu



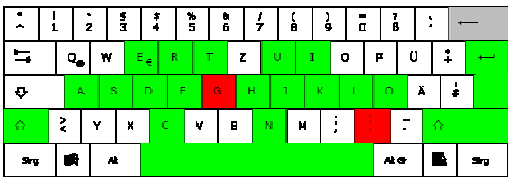
Lektion 6

Zeichen: ASDF JKLÖENRITHCU



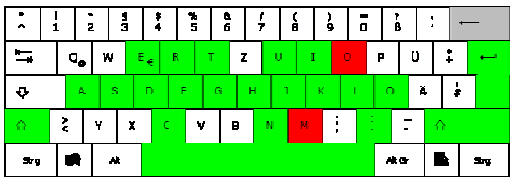
Lektion 7

Zeichen: gG.:



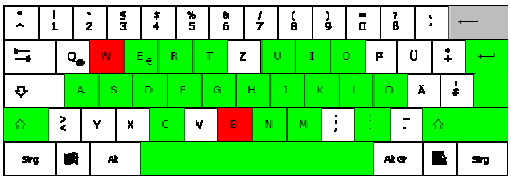
Lektion 8

Zeichen: mMoO



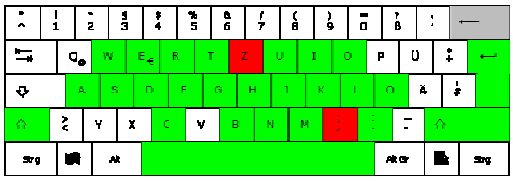
Lektion 9

Zeichen: wWbB



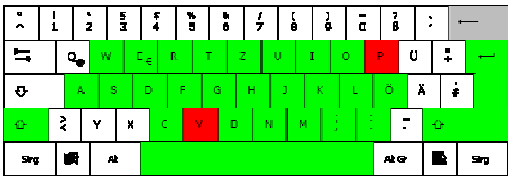
Lektion 10

Zeichen: zZ,;



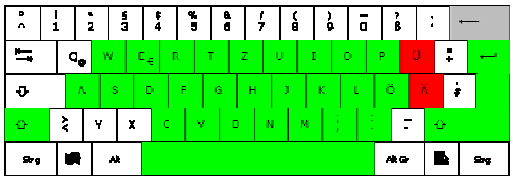
Lektion 11

Zeichen: vVpP



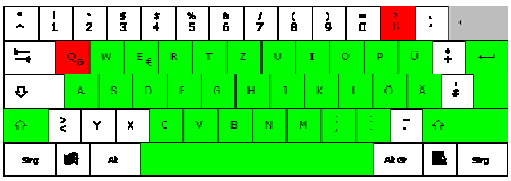
Lektion 12

Zeichen: üÜäÄ



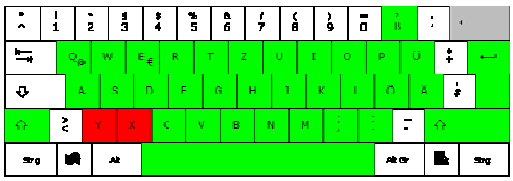
Lektion 13

Zeichen: qQß?



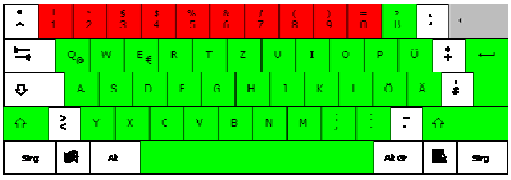
Lektion 14

Zeichen: yYxX



Lektion 15

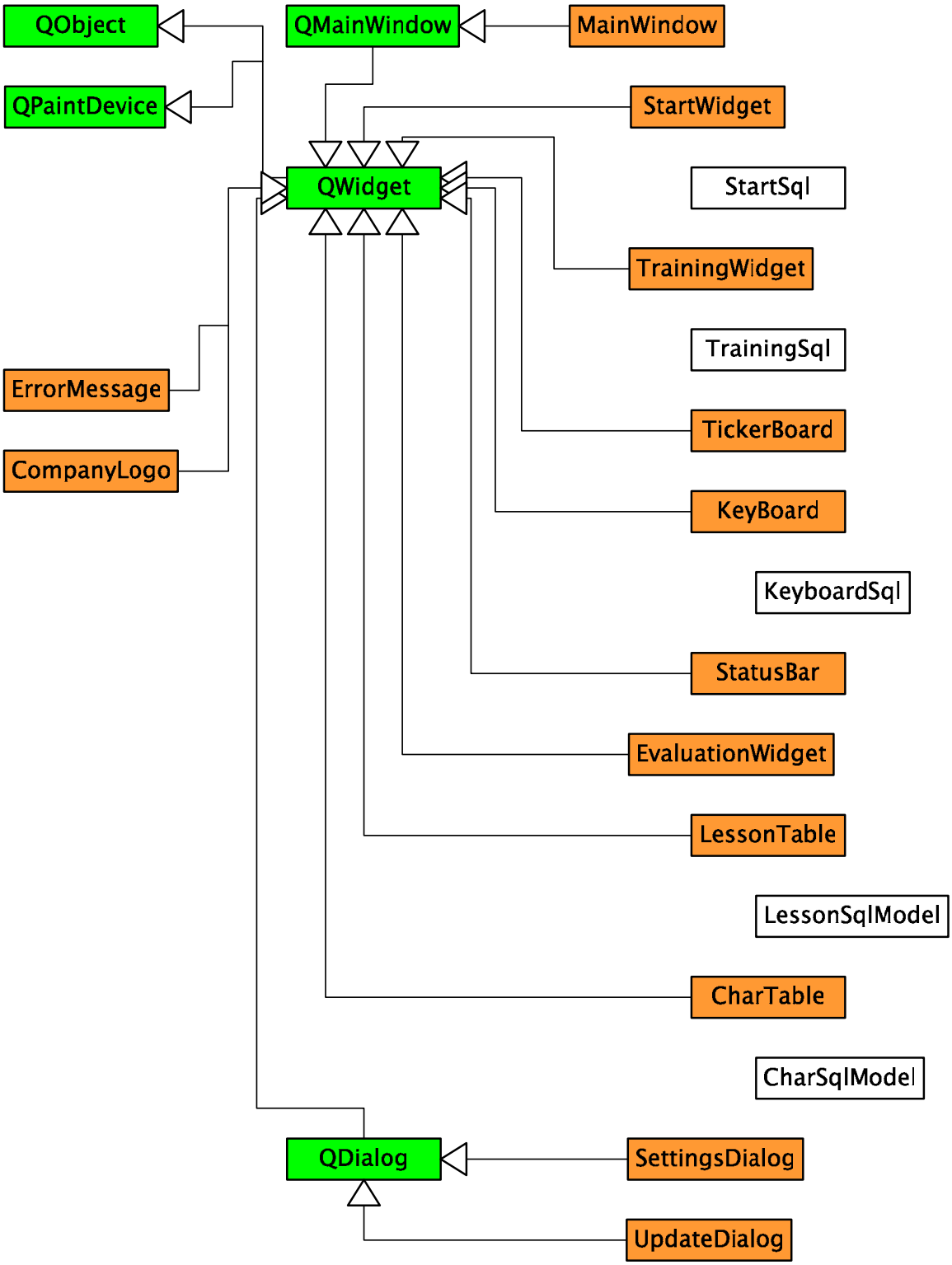
Zeichen: 1!2“3§4$5%6&7/8(9)0=



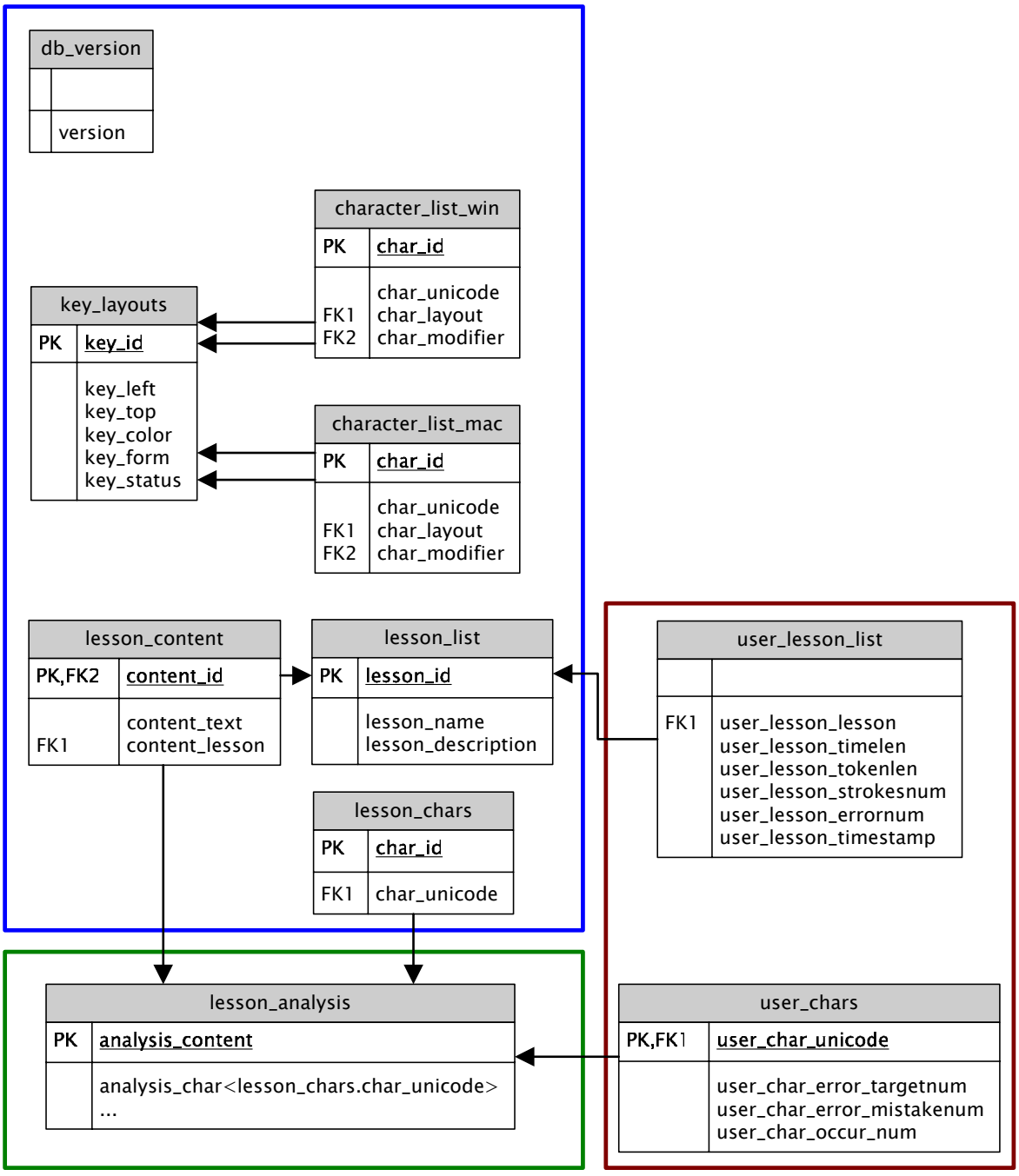
Lektion 16

Zeichen: ^°´`+*#’<>²³{[]}\~@€|µ

B. Klassendiagramm



C. Datenbankdiagramm



Legende:

Tabellen, die über die Aktualisierungsfunktion erneuert werden können

Tabelle, die durch die Aktualisierungsfunktion generiert wird

Anwendertabellen (fester Bestandteil der Datenbank)

D. Literaturverzeichnis

[Altenkrüger 92]

D. Altenkrüger, W. Büttner

Wissensbasierte Systeme: Architektur, Entwicklung, Echtzeitanwendungen

Vieweg Verlag1992, ISBN 3-528-05244-9

[Balzert 00]

H. Balzert

Lehrbuch der Software-Technik

Spektrum, Akademischer Verlag 2000. ISBN 3-8274-0480-0

[Balzert 05]

H. Balzert

Lehrbuch der Objektmodellierung

Spektrum, Akademischer Verlag 2005. ISBN 3-8274-1162-9

[Blanchette 04]

J. Blanchette, M. Sommerfield

C++ GUI Programming with Qt 3

Trolltech AS 2004, ISBN 0-13-124072-2

[Ct 05]

c't, Autor: ola, 2005, Heft 26

Heise Zeitschriften Verlag

[Dalheimer 04]

M. Dalheimer, J. Pedersen et al.

Practical Qt

d.verlag 2004, ISBN 3-89864-260-1

[Haberstumpf 00]

H. Haberstumpf

Handbuch des Urheberrechts

Luchterhand 2000, ISBN 3-472-04381-4

[Lang 40]

K. Lang

Ihr Führer zum Erfolg: Maschinenschreiben 1. Teil

Winklers Verlag, Gebrüder Grimm 1940

[Moser 91]

U. Moser Knaup

Tastmeister: Erlernen des Tastschreibens von der Schreibmaschine zum Computer

Winklers Verlag, Gebrüder Grimm 1991, ISBN 3-8045-7085-2

E. Web-Verzeichnis

[WV Blanchette]

PDF-Version des Buches „C++ GUI Programming with Qt 3“ von J. Blanchette und M. Summerfield (ISBN 0-13-124072-2)
`http://phptr.com/content/images/0131240722/downloads/blanchette_book.pdf`
(Abrufdatum: 13. Mai 2006)

[WV Depends]

Hauptseite des Softwarewerkzeugs „Dependency Walker 2.1“
`http://www.dependencywalker.com/`
(Abrufdatum: 13. Mai 2006)

[WV DocTrolltech]

Dokumentation zu Qt 4.1 der Firma Trolltech
`http://doc.trolltech.com/4.1/index.html`
(Abrufdatum: 13. Mai 2006)

[WV Doxygen]

Hauptseite des Dokumentationssystems „DoxyGen“
`http://www.stack.nl/~dimitri/doxygen/`
(Abrufdatum: 13. Mai 2006)

[WV Gnustandard]

GNU Coding Standards
`http://www.gnu.org/prep/standards/`
(Abrufdatum: 13. Mai 2006)

[WV Inno]

Hauptseite des Softwarewerkzeugs „Inno Setup“

http://www.jrsoftware.org/isinfo.php

(Abrufdatum: 13. Mai 2006)

[WV Javastandard]

Java Coding Style Guide

http://developers.sun.com/prodtech/cc/products/archive/whitepapers/java-style.pdf

(Abrufdatum: 13. Mai 2006)

[WV Ristome]

Hauptseite des RISTOME-Tastaturlayouts

http://www.ristome.de/

(Abrufdatum: 28. Mai 2006)

[WV Sqlite]

Hauptseite der Datenbank „SQLite“

http://www.sqlite.org/

(Abrufdatum: 13. Mai 2006)

[WV SqliteOpt]

Optimierungshinweise und -tips für Sqlite von J. Lyon vom 10. September 2003

http://web.utk.edu/~jplyon/sqlite/SQLite_optimization_FAQ.html

(Abrufdatum: 13. Mai 2006)

[WV Tastaturschreiben]

Schreibleistungen beim Tastaturschreiben

http://www.tastaturschreiben.ch/Index-Dateien/Schreibleistung.htm

(Abrufdatum: 13. Mai 2006)

[WV Trolltech]

Hauptseite der Firma Trolltech
http://www.trolltech.com/
(Abrufdatum: 13. Mai 2006)

[WV Wikidvorak]

Wikipedia-Artikel über das DVORAK-Tastaturlayout
http://de.wikipedia.org/wiki/Dvorak-Tastaturdesign
(Abrufdatum: 28. Mai 2006)

[WV Wikiquertz]

Wikipedia-Artikel über das QWERTZ-Tastaturlayout
http://de.wikipedia.org/wiki/QWERTZ
(Abrufdatum: 28. Mai 2006)

[WV Wikitastatur]

Wikipedia-Artikel über Tastaturen
http://de.wikipedia.org/wiki/Tastatur
(Abrufdatum: 28. Mai 2006)

[WV Wxwidgets]

Hauptseite der Klassenbibliothek „WxWidgets“
http://www.wxwidgets.org/
(Abrufdatum: 13. Mai 2006)

[WV Zeit]

Artikel „Moderne Mythen: Die QWERTY-Tastatur und die Macht des Standards“ von C. Drösser, DIE ZEIT 1997
http://www.zeit.de/archiv/1997/04/qwerty.txt.19970117.xml?page=all
(Abrufdatum: 26. Mai 2006)

F. Abkürzungsverzeichnis

DLL	Dynamic Link Library
GNU-Lizenz (GPL)	GNU General Public Licence
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IP	Internet Protocol
KI	Künstliche Intelligenz
PHP	Hypertext Preprocessor
PNG	Portable Network Graphics
RTF	Rich Text Format
Unicode	Internationaler Zeichenstandard
X11	X Window System
XML	Extensible Markup Language

G. Abbildungsverzeichnis

H. Tabellenverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit *Entwicklung eines intelligenten 10-Finger-Schreibtrainers unter C++* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den 05.07.2006

Tom Thielicke