

Project VON

Belegarbeit im Fach „Entwicklung von
Multimediasystemen“ an der HTW Berlin

Programmierung einer gestengesteuerten Bildergalerie

*Eingereicht bei Sebastian Bauer und
Sebastian Keppler*

*Vorgelegt von Delia Metzler (s0549677) und
Christopher Heiden (s0549453)*

Berlin, 18. Juni 2016

Inhaltsverzeichnis

1. Einleitung	4
2. Planung des Projektes	4
3. Gestellte Anforderungen an dem Projekt	4
4. Aufbau des Projektes	5
4.1 WindowManager	5
4.2 ErstesFenster	5
4.2.1 Sound	6
4.2.2 Vorder- und Hintergrund	6
4.3 ZweitesFenster	6
4.3.1 Optionsleiste	7
4.3.1.1 Vollbild.....	8
4.3.1.2 Hintergrund.....	8
4.3.1.3 Bildergröße.....	8
4.3.1.4 Sprache ändern	9
4.3.1.5 Filtern	9
4.3.1.6 Hilfe	10
4.3.2 Infoleiste	10
4.3.3 Bilder	10
4.3.3.1 Suche	11
4.3.3.2 Umwandlung	11
4.3.3.3 Anzeigen	11
4.3.3.4 MyLabel.....	12
4.4 DrittesFenster	12
4.5 Datenbank-Klasse	13
4.5.1 Datenbankattribute Bild_Dargestellt	13
4.5.1 Funktionen der Datenbank	14
4.6 Kinect v2	14
5. Anforderungen	17
5.1 Gelöste Anforderungen	17
5.2 Ungelöste Anforderungen	18
6. Tests	20
6.2 Unit-Tests	20
6.2 Valgrind/Addresssanitizer	20

7. Fazit	20
I. Anhang	21
Anhang: Bilder	21
Anhang: Anforderungen	22
Anhang: Tests	24

1. Einleitung

In einer Zeit, in der Virtual-Reality und Augmented-Reality durch die Oculus Rift, die HTC VIVE oder die HoloLens real geworden sind, darf man sich nicht nur mit der Programmierung von Spielen oder anderen Arten von Software am PC auseinandersetzen. Aus diesem Grund ist es wichtig sich Wissen über andere Fachgebiete der Informatik zu erschließen, wie zum Beispiel die Programmierung der Kinect von Microsoft, welche einen guten Einstieg in die Welt der Virtualität bietet.

Eine Möglichkeit sein Wissen zu erweitern und einen anderen Weg der Programmierung einzuschlagen, bietet der Kurs „Entwicklung von Multimediasystemen“. In diesem hatte man die Aufgabe, mithilfe des Qt Frameworks, ein vom Dozenten vorgeschlagenes Projekt zu realisieren oder ein eigenes Projekt zu bearbeiten. Dazu konnte man eine Gruppe aus maximal 3 Personen bilden, wobei sich der Umfang, der zu erbringen war, pro Person erhöht.

Für die Entwicklung des Projektes „VON“ bildete sich eine Gruppe aus zwei Personen. Diese setzt sich aus Delia Metzler (s0549677) und Christopher Heiden (s0549453) zusammen. Sie entschieden uns für ein eigenes Projekt, was sie bis zum 18. August 2016 abgeben mussten. Dabei wurde das gesamte Projekt wurde auf einen Monitor programmiert mit einer Auflösung von 1680 x 1050.

2. Planung des Projektes

Um solch ein großes Projekt zu realisieren, galt es als erstes Ideen zu entwickeln, wie die Bildergalerie, die am Ende existieren sollte, aussehen könnte und welche Anforderungen die Applikation haben soll. Dafür skizzierte man zu Beginn mit OneNote (siehe Anhang Bild 1), um sich designerisch an das Projekt anzutasten. Währenddessen entstanden bereits die ersten Anforderungen, wie zum Beispiel, dass man die Hintergrundfarbe ändern oder nach bestimmten Tags filtern kann. Nachdem man die ersten Anforderungen festhielt, wurde ein Papierprototyp entwickelt, welcher das Hauptinterface mit seinen Interaktionsmöglichkeiten und der Darstellung der Bilder, die vom PC des Nutzers stammen, zeigt (siehe Anhang: Bild 2-3). Durch den Prototyp hat man eine bessere Sicht auf das Projekt. Dies hat den Vorteil, dass man sich frühzeitig damit beschäftigt, welche Funktionalitäten in die Applikation existieren sollen und was passieren, wenn der Nutzer bestimmte Buttons bedient. Dadurch entstanden weitere Anforderungen, die mit in die Liste aufgenommen wurden.

3. Gestellte Anforderungen an dem Projekt

Nachdem die designspezifischen Eigenschaften beschlossen waren, war es an der Zeit, die Anforderungen zu spezifizieren. Dazu wurde eine Liste aufgestellt, die sämtliche Eigenschaften enthält (siehe 8.2 Anhang: Anforderungen Tab. 1). Durch die offene Aufgabenstellung, die der Dozent im Kurs „Entwicklung von Multimediasystemen“ stellte, bietet dieser die Möglichkeit mit der Kinect zu arbeiten, was bis zu dem Zeitpunkt nicht Teil des Studiums war.

Eine der wenigen Anforderungen, die der Dozent an alle Teams hatte, war das Nutzen des Qt Frameworks. Man beschloss die Möglichkeit zu ergreifen und entschied sich für die

Integrierung der Kinect in das Projekt. Dafür nutzte man die Kinect v2, mit welcher man die Gestensteuerung realisieren hat.

4. Aufbau des Projektes

Für die Ausarbeitung des Projektes wurde Visual Paradigm 12.2 verwendet, mit dessen Hilfe man ein UML Diagramm skizzierte (siehe Bild 1). Diese theoretische Struktur wurde versucht in das Vorhaben zu implementieren.

Grob lässt sich sagen, dass sich das Projekt aus 5 Hautklassen zusammensetzt, dem „WindowManger“, dem „ErstesFenster“, dem „ZweitesFenster“, dem „DrittesFenster“ und der Datenbank. Diese wiederum haben Unterklassen, die sich um Teile des Programms kümmern, wie die Suche und Darstellung der Bilder.

In den nachfolgenden Unterpunkten wird der Aufbau des Projektes näher erläutert.

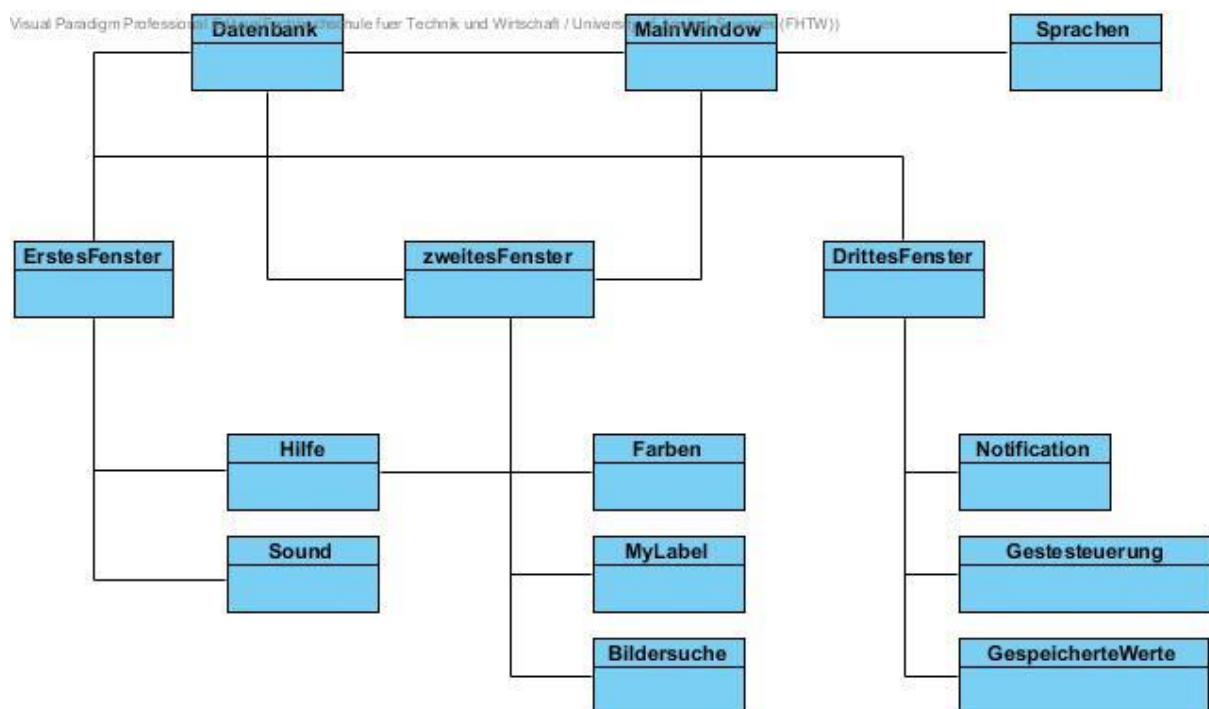


Bild 1 zeigt den Aufbau der Applikation in einem skizzenhaften UML Diagramm.

4.1 WindowManager

Die WindowManager-Klasse dient dazu, zwischen den einzelnen Layouts, die angezeigt werden sollen, zu wechseln, wenn der Nutzer bestimmte Buttons oder andere Aktivitäten betätigt. Dafür wird das Signals & Slots -Prinzip von Qt verwendet. Wenn einer der definierten Signale ausgesendet wird, wird dieses verwertet, indem ein neues Layout gesetzt wird. Zudem wird in dieser Klasse das Icon, welches sich im Ressourcenordner befindet gesetzt, sowie der Name der Applikation.

4.2 ErstesFenster

Dem Nutzer wird beim ersten Öffnen des Programmes nur ein Button präsentiert. Erst beim zweiten Öffnen, wenn die Datenbank bereits Bilder enthält, wird ihm ein zweiter Button zu Verfügung gestellt. (siehe Bild 2 und 3). Durch das Anklicken eines der Buttons gelangt man in das Hauptinterface der Applikation. Auch ist es dem Nutzer möglich die Sprache zu ändern, sowie bei Fragen Hilfestellung zu erhalten



Bild 2 zeigt das erste Fenster, wenn sich in der Datenbank noch kein Bild befinden.



Bild 3 zeigt das erste Fenster, wenn Bilder bereits in der Datenbank enthalten sind.

4.2.1 Sound

Bei jedem Ausführen des Programms wird ein Willkommenston abgespielt. Das Lied, welches abgespielt wird, befindet sich im Ressourcenordner. Da allerdings der Ressourcenordner nicht dazu geeignet ist, Sounds zu beinhalten, wird ein neuer Ordner erzeugt, der das Lied beinhalten. Dieser Ordner wird aufgerufen, um den Sound abzuspielen.

4.2.2 Vorder- und Hintergrund

Der Hintergrund setzt sich aus einer QGraphicsScene und einer QGraphicsView zusammen, die 800 x 600 Pixel groß ist. Dieser wiederum beinhaltenen Linien unterschiedlicher Farben.

Auf dem Hintergrund befinden sich Buttons und das Label, welche in der Klasse Vordergrund definiert sind. Dort werden auch die Stile der Buttons und der Labels gesetzt, sowie deren Signale, die beim Klicken aktiviert werden. Zudem gibt es weitere Buttons, um die Sprache zu ändern und um Hilfe anzufordern.

4.3 ZweitesFenster

Nach der Begrüßung und dem Betätigen eines der Buttons, gelangt der Nutzer der Software zum Hauptinterface (siehe Bild 4). In diesem werden sämtliche vom Nutzer vorher bestimmten Bilder in klein dargestellt und er kann Änderungen am Interface und der Galerie vornehmen, wie zum Beispiel indem er die Farbe der Hintergrundfarbe ändert oder nach bestimmten Bildern filtern.

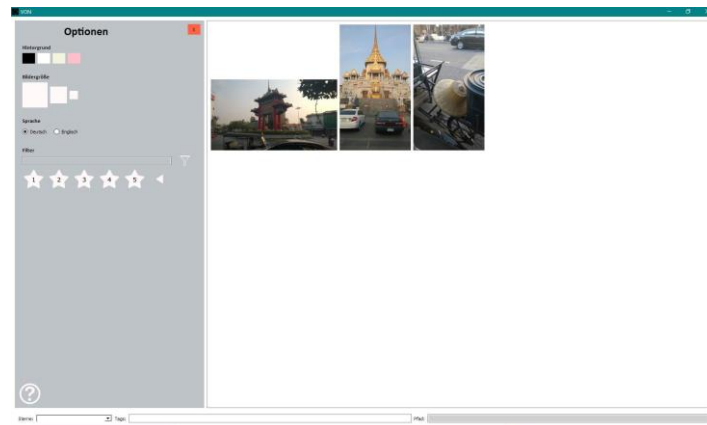


Bild 4 zeigt das Hauptinterface, welches sich zusammensetzt aus einer Optionsleiste, der Informationsleiste und der Bildergalerien.

Hauptaugenmerk des Interfaces ist die Bildergalerie und die Informationsleiste, die sich rechts neben der Optionsleiste bzw. darunter befindet. Das Interface wird im Gegensatz zum ersten Fenster maximal angezeigt.

In den nachfolgenden Kapiteln werden die Options- und Informationsleiste, die Bildergalerie, sowie der Diashowmodus genauer erläutert.

4.3.1 Optionsleiste

Die meisten Möglichkeiten, um mit der Software zu interagieren, ermöglicht die Optionsleiste (siehe Bild 5). Diese bietet dem Nutzer die Option, diese zu entfernen, die Farbe des Hintergrundes und die Größe der Bilder, sowie die Sprache zu ändern. Um sich bei Fragen zu behelfen, bietet man einen Hilfebutton an.

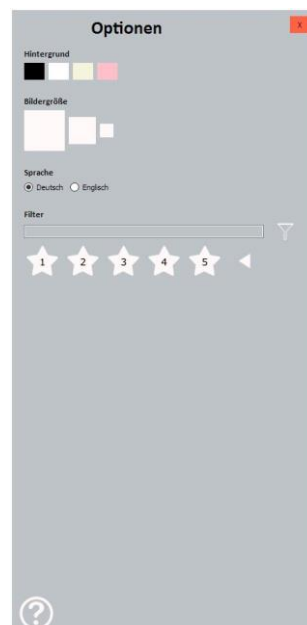


Bild 5 zeigt die Optionsleiste der Software mit der der User mit der Applikation interagieren kann.

4.3.1.1 Vollbild

Der Vollbildmodus wird aktiviert, wenn man auf den roten x-Button (siehe Bild 6) drückt, der sich neben dem Label „Optionen“ befindet. Dieser bewirkt, dass die Optionsleiste verschwindet, sodass man eine größere Fläche für die Bildergalerie hat und dass die Bilder größer dargestellt werden. Um die Optionsleiste wieder sichtbar zu machen, muss man den Button, der sich nun in der Informationsleiste befindet, drücken.



Bild 6 zeigt den Button zur Aktivierung des Vollbildmodus.

Die Informationsleiste existiert trotz dessen, sodass der Nutzer Bilder bewerten und Tags setzen kann. Zudem befindet sich nun ein Button in der Leiste (siehe Bild 7), die den Modus beendet und die Optionsleiste wieder erzeugen kann.



Bild 7 zeigt die Informationsleiste, wenn der Vollbildmodus aktiviert wurde

4.3.1.2 Hintergrund

Die Hintergrundfarbe des Hauptinterfaces kann geändert werden. Der Nutzer kann sich zwischen Schwarz, Weiß, Beige und Pink entscheiden (siehe Bild 8). Bei der Betätigung einer der Buttons wird die Hintergrundfarbe geändert und auch die Farbe der Texte und der Labels werden angepasst.



Bild 8 zeigt die Farben, in der die Hintergrundfarbe der Software verändert werden kann.

4.3.1.3 Bildergröße

Auch die Bildergröße kann vom Nutzer eingestellt werden. Dafür existieren drei Buttons (siehe Bild 9), die bei der Betätigung, die Bilder, die dargestellt werden, vergrößert bzw. verkleinert. Standard ist, dass 20 Bilder dargestellt werden, ohne dass der Nutzer scrollen muss. Dies wurde so implementiert, dass die Bildergröße fest definiert ist. Je nachdem, ob der Nutzer 20, 40 oder 60 Bilder auf einmal sehen möchte.



Bild 9 zeigt, wie der Nutzer der Software die Größe der Bilder anhand eines Klickens auf eine der Buttons beeinflussen kann.

4.3.1.4 Sprache ändern

Auch die Sprache im Hauptinterface kann geändert werden. Zur Auswahl stehen dem Nutzer zwei Sprachen, Deutsch und Englisch. Da das Team davon ausgegangen ist, dass die Applikation für Personen, die als Muttersprache Deutsch gelernt haben, ist, ist standardmäßig die deutsche Sprache gesetzt.

Damit man die Sprache ändern kann, muss der Nutzer auf eine der Flaggen (siehe Bild 10) klicken. Mit diesem Klick werden alle Labels und Texte auf Englisch bzw. Deutsch übersetzt.



Bild 10 zeigt die Möglichkeit die Sprache der Applikation zu ändern, indem man auf einen der beiden Radiobuttons drückt.

4.3.1.5 Filtern

Das Filtern ermöglicht dem Nutzer nach Tags oder nach Bewertungen, welche zuvor in der Informationsleiste für Bilder gesetzt worden sind, zu filtern (siehe Bild 11). Innerhalb des Textfeldes kann man die Tags angeben, nach denen man filtern möchte. Wenn man den Tag gesetzt hat, kann man entweder die Entertaste drücken oder aber auf das „Filter Symbol“ rechts neben dem Textfeld klicken. In der Datenbank wird dann nach Bildern gesucht, die diese Tags enthalten (siehe Kapitel Datenbank 4.5). Die gefundenen Bilder werden in einen Vektor geschrieben, mit dessen Hilfe man die Bilder darstellen kann (siehe Kapitel 4.3.3.2 Umwandlung und 5.3.3.3 Anzeigen).

Um hingegen nach Bewertungen zu filtern, muss man auf einen der Sterne drücken. Dann wird nach einer bestimmten Nummer, die man gedrückt hat, gefiltert. Dabei wird in der Datenbank (siehe Kapitel Datenbank 4.5) nach Bildern gesucht, die diese Bewertung besitzen. Diese werden in einen Vektor geschrieben und dargestellt. Wenn man die Filterung von bestimmten Bewertungen oder Tags beenden möchte, muss man auf den letzten Button in der Reihe der Sterne drücken. Durch diesen werden wieder alle Bilder dargestellt, wie auch beim Öffnen des Programmes erzeugt wurden.



Bild 11 zeigt die Möglichkeiten, die der Nutzer hat, um nach Bildern zu filtern.

4.3.1.6 Hilfe

Der Hilfebutton dient zur Unterstützung des Users bei unverständlichen Symbolen oder Labels. Zudem werden Funktionen dem Nutzer erklärt, die beim ersten Nutzen nicht auffindbar sind, wie das Einfach und Doppelklicken auf ein Bild (siehe Kapitel 4.3.3.4 MyLabel) oder dem Bestätigen durch das Drücken der Taste „Enter“ beim Filtern (siehe Kapitel 4.3.1.4 Filtern).

Beim Anklicken des Buttons (siehe Bild 12) öffnet sich ein weiteres Fenster, welches Erklärungen zu bestimmten Buttons und Funktionen bietet.



Bild 12 zeigt den Button, der bei Fragen gedrückt werden soll.

4.3.2 Informationsleiste

Unterhalb der Optionsleiste und der Bildergalerie befindet sich die Informationsleiste (siehe Bild 13), die Informationen über ausgewählte Bilder anzeigt. Zudem kann man dort die Bilder bewerten und Tags setzen, nach denen man in der Optionsleiste filtern kann (siehe Kapitel 4.3.1.2 Filter). Darüber hinaus wird der Pfad angezeigt, welchen man nicht verändern kann. Aus diesem Grund ist das Textfeld auch ausgegraut, um dies dem Nutzer anzudeuten. Die Funktionen stehen in enger Zusammenarbeit mit der Datenbank (siehe Kapitel 4.5).



Bild 13 zeigt die Informationsleiste, in der die Bewertung, die Tags und der Pfad des Bildes angezeigt werden, welches angeklickt wurde.

4.3.3 Bilder

In diesem Kapitel wird darauf eingegangen, wie der Algorithmus, nach Bildern sucht, sie umwandelt, anzeigt und wie man das Drücken bzw. doppelklicken auf die Bilder, realisiert hat, um die Informationen des jeweiligen Bildes auf der Informationsleiste auszugeben.

4.3.3.1 Suche

Die Suche beginnt mit dem Drücken eines der Buttons aus dem ersten Fenster. Wenn der Nutzer den Button mit der Aufschrift „Startverzeichnis“ drückt, öffnet sich die Ordnerstruktur des Nutzers (siehe Bild 13). Mit dessen Hilfe, kann sich der Nutzer für ein Startverzeichnis von dem aus das Programm nach Bildern suchen soll, entscheiden. Dabei ist zu bemerken, dass nur nach Bildern gesucht werden kann, die vom Datentyp jpg sind. Wenn sich der Nutzer für ein Verzeichnis entschieden hat, wird nach Bildern, die solch eine Endung haben iterativ gesucht. Bilder, die solch eine Endung besitzen, werden als Strings (Pfad des Bilds) in einen Vektor gespeichert, mit dem im Anschluss weitergearbeitet wird.

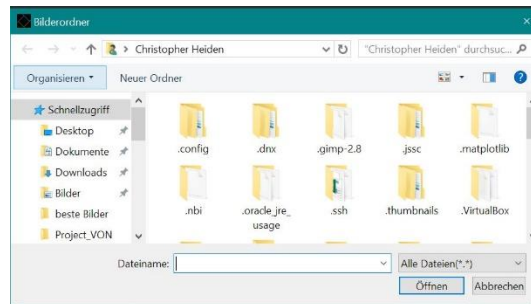


Bild 13 zeigt eine beispielhafte Ordnerstruktur.

Wenn der Nutzer den anderen Button mit der Aufschrift „Letzte Bilder anzeigen“ im ersten Fenster betätigt, wird nach Bildern (deren Pfade) in der Datenbank gesucht, die sich der Nutzer bei letzter Nutzung der Applikation angesehen hat. Diese werden ebenfalls in einem Vektor gespeichert.

4.3.3.2 Umwandlung

Die Pfade der gefundenen Bilder, welche in dem Vektor gespeichert wurden, werden nun zu QImages umgewandelt. Da dies sehr zeitaufwändig ist, entschied man sich QThreads setzt, um die Bearbeitungszeit zu verringern. Bei der Umwandlung spielt auch die Anzahl der Bilder, die ohne scrollen angezeigt werden sollen, eine Rolle. Umso mehr Bilder angezeigt werden sollen, umso kleiner werden die Bilder angezeigt. Die Angaben, wie groß die Bilder angezeigt werden sollen, ist hart kodiert. Die QImages werden wieder in einer Map gespeichert, in der sich als Key auch der Pfad des jeweiligen QImages befindet.

4.3.3.3 Anzeigen

Die Bilder, die sich in einer Map befinden, werden mit einem Iterator durchgegangen. Dabei wird jedes QImage einer vom Team selbst erweiterten Klasse „MyLabel“ zugewiesen und mit Signals versehen. Daraufhin wird jedem Bild ein QPixmap gesetzt.

Wie schon im Kapitel 4.3.3.2 Umwandlung spielt auch hier die Bildgröße eine Rolle. Wenn man 20 Bilder ohne scrollen anzeigen möchte, werden 5 Bilder in einer Reihe angezeigt (siehe Bild 14). Bei 40 bzw. 60 Bildern, die auf einmal sichtbar sein sollen, werden 10 bzw. 15 Bilder nebeneinander angezeigt.

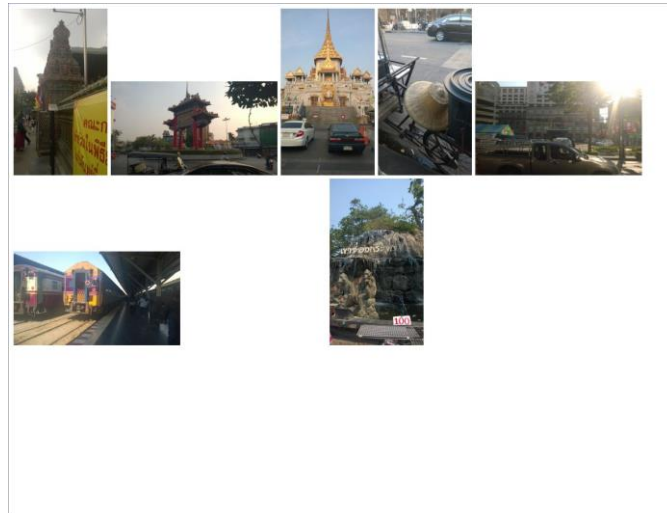


Bild 14 zeigt ein Beispiel für das Anzeigen der Bilder im Hauptinterface.

4.3.3.4 MyLabel

Die Klasse MyLabel erbt von der Klasse QLabel. Somit hat MyLabel alle Funktionen, die auch die Klasse QLabel besitzt sowie weitere, die man ergänzt hat. Die Funktion, um die man die Basisklasse QLabel ergänzt hat, war die Möglichkeit ein Bild einfach und doppelt anzuklicken. Mit diesen Möglichkeiten ergibt sich, dass man beim Anklicken auf ein Bild, der Pfad, die Bewertung und die Tags in der Informationsleiste angezeigt (siehe Kapitel 4.3.2 Informationsleiste) bekommt.

Beim Doppelklicken auf ein Bild wird der Diashowmodus (siehe Kapitel 4.4 Diashowmodus) geöffnet, in dem das Bild in Großformat angezeigt wird.

4.4 DrittesFenster

Der Diashowmodus ermöglicht dem Nutzer das Betrachten seiner Bilder in groß. Dabei spielt es eine Rolle, ob die Bilder im Hoch- oder Querformat sind (siehe Bild 15).

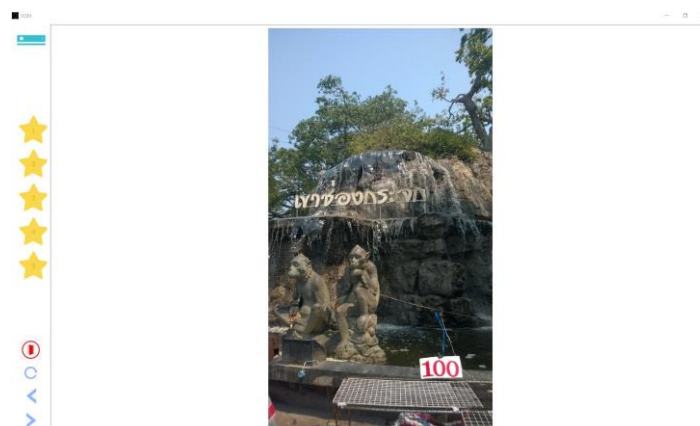


Bild 15 zeigt anhand eines Beispielbildes, wie der Diashowmodus aussieht.

Durch die Datenbank, kann man auf die Bilder, die in dieser Session angezeigt wurden sind, zugreifen, um diese in einem Vektor zu speichern und diese bewerten. Beim Bedienen einer der Pfeilbuttons, wird das nächste bzw. das vorherige Bild im Vektor dargestellt.

Man hat auch die Möglichkeit die Bilder zu drehen. Durch das drücken des Knopfes wird das jeweilige Bild, welches gerade angezeigt wird, nach rechts um 45 Grad gedreht. Die Änderungen werden in der Datenbank gespeichert.

Wenn man aus dem Diashowmodus wieder zu seiner Bildergalerie gehen möchte, drückt man auf den Button mit dem Fluchtsymbol.

Die Funktionalitäten sind auch mit der Kinect nutzbar. Um dies dem Nutzer kenntlich zu machen, wurde ein Symbol innerhalb der Optionsleiste implementiert.

4.5 Datenbank-Klasse

Die Datenbank ist hauptsächlich dazu da, alle Daten zu den Bildern, die der Nutzer in das Programm lädt zu speichern. Mit SQLite wurde eine Datenbank erzeugt, die die Attribute BildID, Bildpfad, Bildwertung, Bildtags, Bildausrichtung, sowie Bild_dargestellt enthält (siehe Bild 16). BildID agiert als Primärschlüssel und wird mit autoincrement automatisch bei jedem neuen Bild erhöht. Der Bildpfad ist zwar auch einzigartig und könnte somit als Primärschlüssel dienen, allerdings ist das Übergeben eines Integers in den späteren Methodenaufrufen performanter, als die Übergabe eines Strings.

```
Datenbank::Datenbank() {  
  
    mydb = QSqlDatabase::addDatabase("QSQLITE");  
    mydb.setDatabaseName("../testdatabase.db");  
  
    if (!mydb.open())  
    {  
        qDebug() << "Error: connection with database fail";  
    }  
    else  
    {  
        qDebug() << "Database: connection ok";  
    }  
  
    QSqlQuery qry;  
    qry.prepare( "CREATE TABLE IF NOT EXISTS Bilder (BildID integer primary key AUTOINCREMENT, Bildpfad text, "  
                "Bildwertung integer check(Bildwertung between 1 and 5), Bildtags text, Bild_dargestellt integer, "  
                "Bildausrichtung integer)" ); //Bild_dargestellt 0 = false, 1 = memory, 2 = true  
    if( !qry.exec() )  
        qDebug() << "Datenbank anlegen Error:" << qry.lastError();  
    else  
        qDebug() << "Table created!";  
}  
}
```

Bild 16 zeigt den Verbindungsaufbau und das Erzeugen der Datenbank.

4.5.1 Datenbankattribut Bild_Dargestellt

Das Attribut Bild_Dargestellt ist zu keiner Zeit für den Nutzer erkenntlich. Es zeigt einen Status an, über den man darauf schließen kann, wann das Bild zuletzt im Programm dargestellt wurde. Ein Bild kann drei Zustände haben, die als Integer in der Datenbank abgespeichert werden (siehe Tabelle 1).

true (2)	Das Bild wird aktuell dargestellt und der Nutzer kann mit dem Bild interagieren.
memory (1)	Das Bild wurde am Anfang der Sitzung in das Programm geladen, wird aber aktuell nicht angezeigt. Der Nutzer hat im weiteren Sitzungsverlauf die Möglichkeit, es sich wieder anzeigen zu lassen.
false (0)	Auf das Bild erfolgt in der aktuellen Sitzung kein Zugriff.

Tabelle 1 zeigt die unterschiedlichen Status, die Bilder einnehmen können.

Vor dem Beginn einer neuen Sitzung, wird Bild_Dargestellt für alle Bilder, die bereits in der Datenbank sind auf false (0) gesetzt. Dann hat der Nutzer die Möglichkeit, Bilder auszuwählen, die im Programm angezeigt werden sollen (siehe 4.2 Begrüßung). Für die Bilder, die der Nutzer in das Programm lädt, wird das Attribut Bild_Dargestellt, unabhängig davon, ob es neue Bilder sind oder ob sie bereits in der Datenbank gespeichert sind, auf 2 gesetzt, weil sie alle für den Nutzer sichtbar sind. Entscheidet er sich dazu nach Bildern mit einer bestimmten Wertung zu Filtern, wird Bild_Dargestellt für alle Bilder, deren Wertung nicht mit der gefilterten übereinstimmen auf memory (1) gesetzt, während bei den anderen die 2 für true stehen bleibt. Betätigt der Nutzer nach dem Filtern den „Zurück“-Button (siehe 4.3.1.5 Filtern), werden alle Bilder, die den Status memory hatten wieder auf true gesetzt und mit angezeigt.

4.5.2 Funktionen der Datenbank

Die Funktionen der Datenbank decken diverse Funktionalitäten ab, die das Hauptinterface bietet, sowie einige, die im Hintergrund ablaufen. Die Funktionsweise aller Funktionen ist sehr ähnlich. Es wird mit der Klasse QSqlQuery von Qt gearbeitet. Mit ihr werden neue SQL-Statements verfasst, gegebenenfalls um zusätzliche Informationen ergänzt und ausgeführt. Als Ergebnis erhält man je nach Anfrage einen Boolean, der erkenntlich macht, ob das Statement erfolgreich ausgeführt werden konnte oder nicht und einen bzw. mehrere Ergebniswert(e), durch die man mit weiteren Methoden der Klasse iterieren kann, um sie dann in andere Datentypen umzuwandeln und mit ihnen weiterarbeiten zu können (siehe Bild 17).

```
int Datenbank::getBildausrichtung(int id){
    int ausrichtung = 0;
    if(bildExists(id)){
        QSqlQuery query;
        query.prepare("SELECT Bildausrichtung FROM Bilder WHERE BildID = (:BildID)"); //vorbereiten eines neuen SQL-Statements
        query.bindValue(":BildID", id); //zusätzliche Informationen werden in das Statement eingebunden
        if(query.exec()){ //Query wird ausgeführt und Ueberprüfung des Rueckgabewerts von exec()
            query.first(); //es ist nur ein Ergebniswert zu erwarten, deswegen direkter Zugriff auf query.first
            ausrichtung = query.value(0).toInt(); //Wert in eine Integer-Variable speichern
            qDebug() << ausrichtung; //Wert zur Ueberprüfung im qDebug ausgeben
        }
        else{ //wenn Rueckgabewert von exec() false ist ...
            qDebug() << "getBildausrichtung error:" << query.lastError(); //... gabe es einen Fehler, der ueber qDebug ausgegeben wird
        }
    }
    return ausrichtung;
}
```

Bild 17 zeigt den Ablauf einer Funktion der Datenbankklasse. Hier im Beispiel das Ermitteln der Bildausrichtung eines bestimmten Bildes.

4.6 Kinect v2

Mit der Kinect v2 hat man beispielweise die Möglichkeiten, Audioinformationen aufzunehmen sowie Gesichter oder Körper zu erfassen. Im gestengesteuerten Diashowmodus treckt die rechte Hand. Beim Trecken der Hand werden deren x- und y-Koordinaten übertragen (siehe Abbildung 1).



Abbildung 1 zeigt das Koordinatensystem der Kinect.

Mit deren Hilfe realisiert man die Gesten zum Wischen, Zoomen und Drehen eines Bildes. Um Gesten zu erkennen, werden die Punkte in einem Vektor gespeichert. Die x- und y-Koordinaten erhält man aus der exe-Datei, die man in der Applikation mithilfe eines QProcesses startet.

Dabei wird zwischen einer offenen und einer geschlossenen Hand unterschieden. Nur bei einer offenen Hand werden Informationen zur Hand übermittelt. Somit kann man eindeutig den Anfangs- und den Endpunkt der Geste bestimmen. Wenn man die Hand allerdings zu lange geschlossen hält deaktiviert sich die Gestensteuerung deaktiviert. Erst beim Anklicken eines Buttons wird die Funktionalität wieder aktiviert (siehe Bild 18).



Bild 18 zeigt das Icon, welches genutzt wird, um die die Gestensteuerung wieder zu aktivieren.

Damit Gesten bestimmt werden können, müssen sich mindestens 81 Handkoordinaten im Vektor gespeichert worden sein.

Beim Wischen spielen vor allem Anfangs- und Endpunkt eine wesentliche Rolle. Diese Punkte müssen unterschiedliche x-Werte besitzen, die y-Werte sollten im günstigsten Fall gleich sein. Da dies nicht der Normalfall ist und der Endpunkt häufig einen anderen y-Wert hat, haben die Entwickler eine Schwankung eingeführt, die für den y-Wert gilt. Diese Schwankung wurde anhand des Startpunktes berechnet. Um zu unterscheiden, ob das vorherige bzw. das nachfolgende Bild aus einem Vektor dargestellt werden soll, untersucht man die x-Werte der beiden Punkte. Wenn der x-Wert vom Startpunkt größer als der x-Wert des Endpunktes ist, wird das nächste Bilder erzeugt. Anders bei einem größeren x-Wert des Endpunktes. In dem Fall wird das vorherige Bilder dargestellt (siehe Abbildung 2).

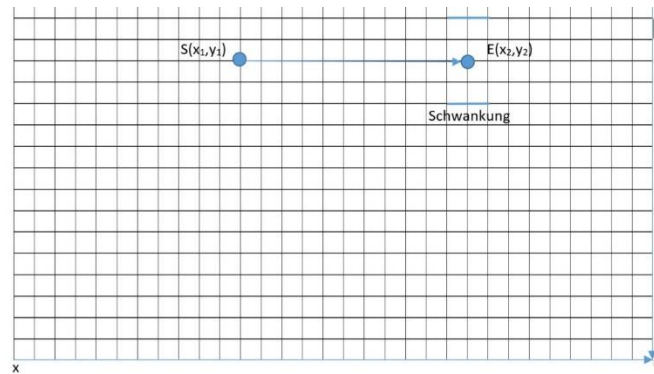


Abbildung 2 zeigt das mathematische Vorgehen zur Erkennung einer Wischgeste anhand einer Skizze.

Bei der Zoomgeste (siehe Abbildung 3) wird ebenfalls unterschieden, in welche Richtung die Senkrechtbewegung stattfindet. Wenn in das Bild hineingezoomt werden soll, muss der y-Wert des Endpunktes kleiner als die Schwankung des y-Startpunktes und der x-Wert größer als die Schwankung des x-Startpunktes sein.

Wenn allerdings der y-Wert des Endpunktes größer als die Schwankung des y-Startpunktes und der x-Wert des Endpunktes kleiner als dessen x-Startpunkt ist, wird hinausgezoomt.

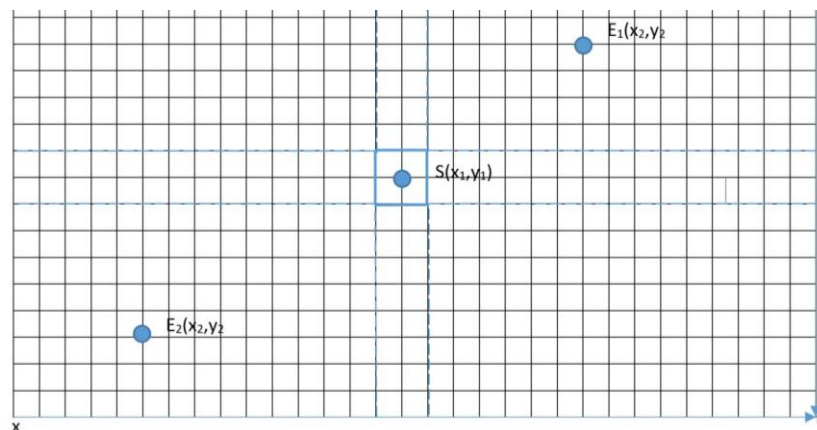


Abbildung 3 zeigt das mathematische Vorgehen zur Erkennung, ob es sich um eine Zoomgeste handelt, anhand einer Skizze.

Bei der Drehung müssen weitere Punkte berücksichtigt werden. So muss der Anfangs- und der Endpunkt und somit der x-Werte und y-Werte ungefähr gleich sein. Wenn das zutrifft, werden Werte, die im ersten Viertel des Vektors stehen, geprüft, ob sich einer dieser in einem bestimmten Feld aufhält. Das gleiche Verfahren wird in den weiteren Vektoren, der sich bei der Hälfte des Vektors und der sich im dritten Viertel des Vektors befindet, berechnet. Zum besseren Verständnis siehe Abbildung 4.

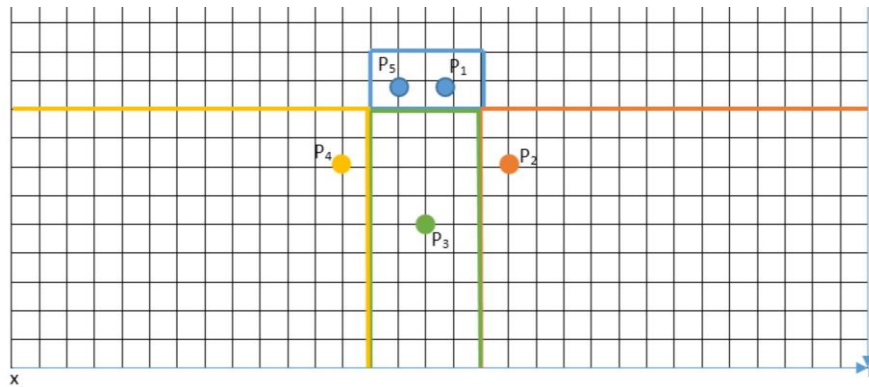


Abbildung 4 zeigt das mathematische Vorgehen, ob es sich um eine Drehung handelt.

5. Anforderungen

Für die Bearbeitung der Projekte innerhalb des Kurses „Entwicklung von Multimediasystemen“ gab es vom Dozenten und vom Projektteam Anforderungen, die es zu erfüllen galt.

In den nachfolgenden Kapiteln geht es um die gelösten Anforderungen und was es bei einigen für Schwierigkeiten gab und die ungelösten Anforderungen mit einer Erklärung, weshalb man diese nicht wie gewünscht lösen konnte.

5.1 Gelöste Anforderungen

Einer der zu Beginn schwierigsten Probleme, die es zu lösen galt, war das Wechseln auf ein neues Layout, um beispielsweise vom Begrüßungslayout zum Hauptinterface zu gelangen. Um dieses Problem zu lösen, nutzt man den WindowManager, der die Layouts bei bestimmten Signalen ändert. Dafür nutzt man die Funktion `setCurrentWidget()` und das `StackedLayout`.

Ein anderes Problem entstand beim Abspielen des Sounds im Begrüßungsinterface. Damit alles in einem Ordner gespeichert wird, entschied man sich den Sound ebenfalls als eine Ressource, wie auch schon das Icon, zu setzen. Bei dem Versuch den Sound abzuspielen, wurde festgestellt, dass der Ressourcenorder nicht geeignet dafür ist Sound abzuspielen. Deshalb entschied man sich einen versteckten Ordner zu erzeugen, in dem sich der Sound befinden soll und abgespielt wird. Dies funktionierte, auch wenn es sehr lange gedauert hat bis ein Ton zu hören war. Aus diesem Grund entschied man sich mit Threads zu arbeiten, in der Hoffnung, dass es schneller ginge. Der Versuch ging schief und es dauerte immer noch zu lange. Schlussendlich hat man sich dafür entschieden, den Sound zu kürzen, so dass es anstelle von 3 Minuten, nur noch sieben Sekunden sind, die es zu laden gilt.

Ein anderes Problem entstand bei der Darstellung der Bilder. Das Problem war, dass zu Beginn anstelle von 98 Bildern nur an die 40 dargestellt wurden. Dies lag wahrscheinlich an der Größe der Bilder, die im MegaByte-Bereich lagen. Aus diesem Grund musste man die Bilder verkleinern. Um dies zu realisieren, durfte man nicht, wie man es zu Beginn implementiert hat, mit `QButtons` arbeiten, auf denen man die Bilder als Icons implementiert hat, sondern musste die Bilder umwandeln von `QString` zu `QImages`. Durch diese Umwandlung war man in der Lage, alle Bilder verkleinert darzustellen.

Doch durch die Lösung entstanden andere Problem. Die Umwandlung von `QString` zu `QImages` dauerte zu lange. Deshalb entschied man sich für die Verwendung von Threads.

Trotz dessen man einen QThread nutzte, nahm es immer noch zu viel Zeit in Anspruch um große Mengen von Bildern in das Programm zu laden. Um dieses Problem zu lösen könnte man in der Datenbank ein Attribut namens „BLOB“ setzen in dem man QImage's speichern kann, wodurch man beim zweimaligen nutzen der Bilder keine Umwandlungen mehr durchführen müsste, sondern man direkt auf die QImage's zugreifen könnte. Aus Zeitgründen wurde dies nicht implementiert.

Ein anderes Problem entstand als man den Button implementiert hat, mit dessen Hilfe man vom Diashowmodus wieder in das Hauptinterface zu gelangen versuchte. Jedes Mal, wenn man auf den Button gedrückt hat und man zurück zur Bildergalerie kehrte, entstanden innerhalb der Optionsleiste weitere Buttons, welche oberhalb der Leiste dargestellt wurden (siehe Bild 19). Das Problem war, dass neue Connections deklariert wurden. Um dies zu umgehen, wurden die Connections von Signals und Slots innerhalb des Konstruktor geschrieben.

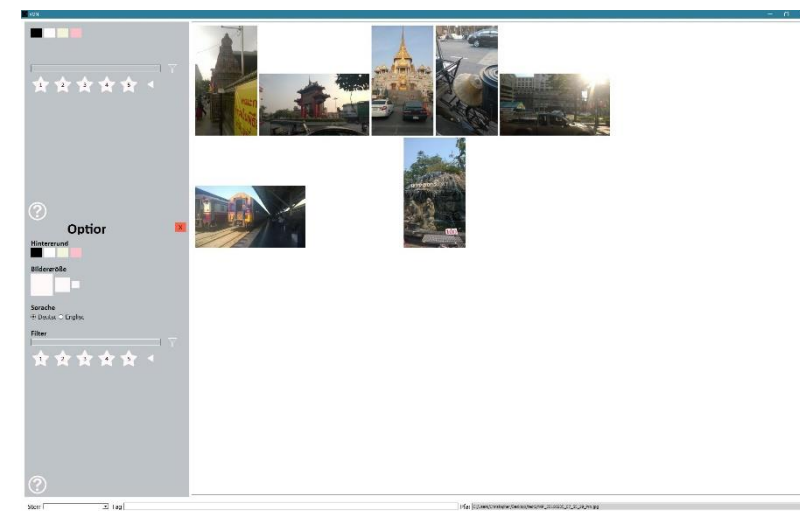


Bild 19 zeigt das Problem beim Zurückkehren vom Diashowmodus zur Bildergalerie.

Dasselbe Problem entstand im Diashowmodus, wenn man sich das nächste Bild ansehen wollte. Dort trat es allerdings so auf, dass man anstelle des nächsten Bilder, man das übernächste Bild dargestellt bekommen hat. Diese Art von Fehler potenzierte sich bei jedem Mal, wenn man das nächste oder das vorherige Bild ansehen wollte.

5.2 Ungelöste Anforderungen

Eine der vom Dozenten spezifizierten Anforderungen an die Applikation, war, dass man diese auch am Smartphone testet. Schließlich sagen die Entwickler von QT, dass man mit diesem Framework gerätunabhängig programmieren kann und das entstandene Projekt leicht auch auf andere Geräte zu bringen ist, bzw. nur wenige Anpassungen benötigt. Aufgrund der Problematik, das Smartphones einen anderen Aufbau und Struktur besitzen als die Betriebssysteme für PCs, ist ein Zugriff auf ein Bildverzeichnis anderes zu bewältigen als bei den klassischen Betriebssystemen. Somit ist das Laden von Bildern nicht möglich. Zudem sind auch das Speichern und der Zugriff auf diese anderes zu realisieren. Aus diesem Grund und durch den Einsatz der Kinect ist die Verwendung des Projektes auf dem Smartphone nicht realisierbar.

Eine andere Anforderung, die es nicht in das Endprojekt geschafft hat, ist die Geste des Puschens eines Bildes. Dies sollte dieselbe Funktion bieten, wie das Doppelklicken auf ein Bild, wodurch der Diashowmodus aktiviert wird. Dies ist vor allem nicht möglich gewesen, da man es nicht geschafft hat den z-Wert der Hand zu ermitteln. Außerdem hätte man die getrackte Hand mit der Maus verknüpfen müssen, sodass, wenn man die Hand bewegt auch der Zeiger der Maus sich in dieselbe Richtung bewegt, wie die Hand.

Eine andere Geste, welches nicht möglich war zu implementieren, war das Beenden und wieder aktivieren der Gestesteuerungen. Zwar kann man die Steuerung beenden doch kann man nicht diese wieder aktivieren indem man eine andere Geste durchführt. Um diese Lücke zu schließen, wurde ein Button hinzugefügt, der diese wieder reaktiviert.

Gravierendere Fehler treten beim Hineinladen der Bilder auf, wenn man sich selbst das Verzeichnis aussucht. Manchmal werden alle oder nur wenige oder manchmal sogar kein Bild dargestellt, obwohl die Bilder in der Datenbank hinterlegt worden sind.

Wenn man dann beispielsweise den Vollbildmodus aktiviert, werden die Bilder aus der DB gelesen und dargestellt. Merkwürdig dabei ist, dass man bei beiden Optionen (Verzeichnis selber aussuchen oder auslesen der zu Letzt gezeigten Bilder aus der Datenbank) den selben Algorithmus nutzt.

Auch tritt ein Fehler auf, wenn man im zweiten Fenster den Hilfebutton betätigt. Dadurch stürzt das Programm ab. Das zumindest ist bei Delia Metzger der Fall. Vielleicht spielt auch die Version des Qt Frameworks eine Rolle, weshalb es dies bei ihr nicht ging. Sie nutzt die Qt Version 5.5 und der Programmierer der Funktion Christopher Heiden nutzt die Qt Version 5.6.

Ein anderer Makel ist, dass die Bilder sich im Diashowmodus überlappen, wenn man Bilder lädt, die horizontal, als auch vertikal sind. Um dieses Problem lösen zu können, müsste man wohl an der richtigen Stelle Speicher freigeben.

Mit Dockern schafft man es Programme, die auf Windowsrechnern laufen, auch auf Linux Rechner zu importieren. Auch dies sollte man als eine der Anforderungen umsetzen. Das ist in dem Fall nicht möglich, da man die Kinect v2 mit in das Projekt integriert hat und man diese mit dem Kinect SDK von Microsoft nutzt. Da die nötigen Treiber auf Linux Rechnern nicht installiert werden können, hat man davon abgesehen, dies zu realisieren. Zudem entstand ein Fehler bei der Ausführung von Docker for Windows, der bis zur Abgabe des Beleges ebenfalls nicht behoben werden konnte.

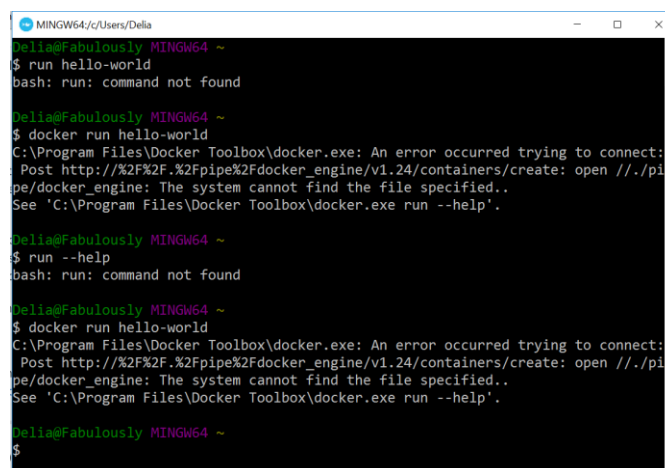
A screenshot of a terminal window titled 'MINGW64/c/Users/Delia'. The prompt is 'Delia@Fabulously MINGW64 ~'. The user enters '\$ run hello-world', and the output is 'bash: run: command not found'. Then the user enters '\$ docker run hello-world', and the output is an error message: 'C:\Program Files\Docker Toolbox\docker.exe: An error occurred trying to connect: Post http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.24/containers/create: open //.pipe/docker_engine: The system cannot find the file specified.. See 'C:\Program Files\Docker Toolbox\docker.exe run --help''. The user then enters '\$ run --help', and the output is 'bash: run: command not found'. Finally, the user enters '\$ docker run hello-world' again, and the same error message appears. The terminal window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Bild 20 zeigt die Fehlermeldung innerhalb des Docker Terminals.

6. Tests

Mithilfe von Unit-Tests soll überprüft und bewiesen werden, ob die Funktionen, welche man programmiert hat, funktionieren. Dies tat man für 25 Methoden, welche alle in der Datenbankklasse implementiert wurden, denn wenn dort Fehler auftreten, ist die ganze Applikation betroffen, da Interface und Datenbank stark miteinander verwoben sind.

6.1 Unit-Tests

Zum Arbeiten mit den Unit-Tests, wurde die Projekt Datei zunächst um testlib erweitert. Für die Tests wurde dann eine neue Klasse angelegt, die von QObject erbt und in der alle Testmethoden als private SLOTS gesetzt wurden. Der Aufruf erfolgt in der Main-Methode mit dem Befehl QTest::qExec(). Die Ergebnisse der Unittests sind im Anhang unter Tests nachlesbar.

6.2 Valgrind/Addresssanitizer

Da unter dem Betriebssystem Windows gearbeitet wurde, für das Valgrind und auch Addresssanitizer nicht verfügbar sind, wurde als Ersatz die Software Dr. Memory genutzt, die die selbe Funktionalität bietet, wie die beiden vom Dozenten vorgegebenen Tools. Die Resultate von Dr. Memory befinden sich genau wie die Ergebnisse der Unit-Tests im Anhang unter Tests.

7. Fazit

Eine solch große Aufgabe wurde zum ersten Mal innerhalb des Studiums gestellt. Allerdings ist das gut, da man dadurch erfasst, wie schnell ein Projekt wachsen kann und wie schwer es ist die letzten 20-30% der Anforderungen zu bearbeiten und wie falsch man mit der Zeit Einplanung liegen kann. Auch wenn man nicht alles geschafft hat, was man sich an Anforderungen gestellt hat, hat es der Kurs geschafft, dass man sich nicht nur mit der Materie „Programmierung“ beschäftigt hat, sondern auch mit Softwareengineering, Kontrollstrukturen und Projektmanagement.

Die Programmierung der Kinect stellte sich im Endeffekt als eine als eine relativ leichte Aufgabe da, da man ein bereits existierendes Projekt von Microsoft nutzen konnte und man dieses nur erweitern musste um die gewünschten Aufgaben zu erfüllen. Weitaus schwieriger war es mit den Werten, die man mit der Kinect erhalten hat zu arbeiten und die Gesten zu erkennen. Somit konnte man allerdings es umgehen, die Kinect in Qt zu integrieren.

8. Anhang

8.1 Anhang: Bilder

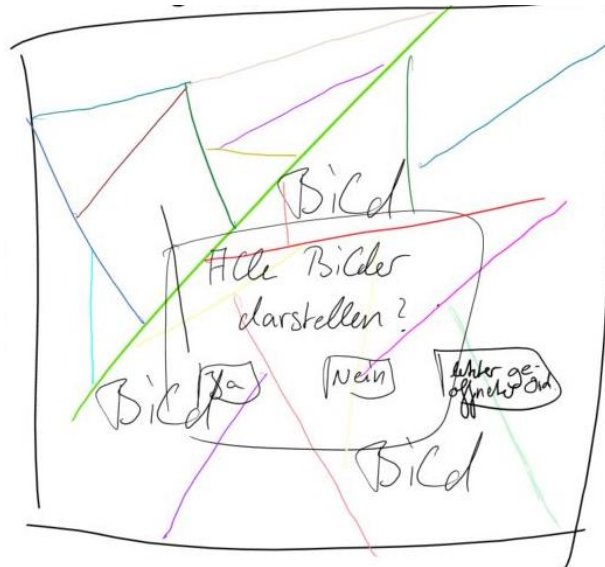


Bild 1 zeigt die Entstehung des Projektes. Für Skizzierung nutze man OneNote. Diese Skizze zeigt den ersten Startbildschirm, welcher beim Öffnen des Programms angezeigt werden soll.

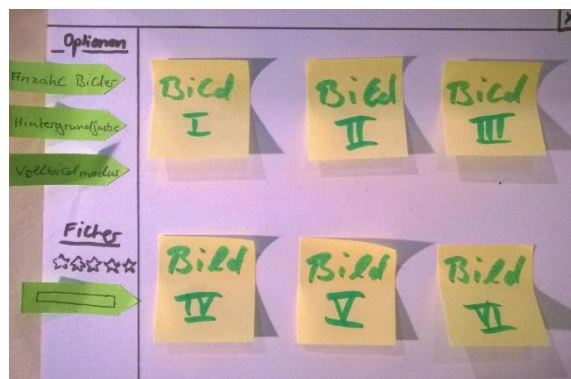


Bild 2 zeigt den Papierprototypen der Bildergalerie und bereits die meisten Funktionen die diese im späteren Verlauf des Projektes implementiert haben soll.

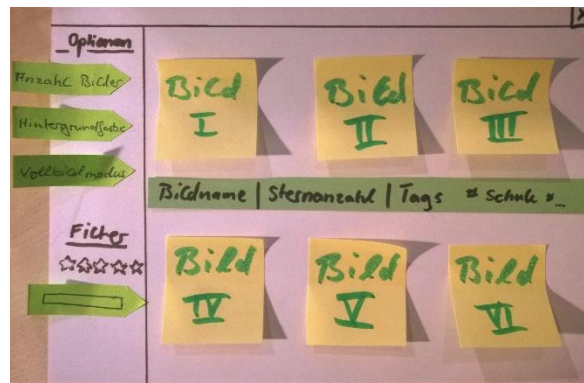


Bild 3 zeigt den Papierprototypen der Bildergalerie und bereits die meisten Funktionen die diese im späteren Verlauf des Projektes implementiert haben soll, sowie, was geschehen soll, wenn der Nutzer auf eines der Bilder geklickt hat.

8.2 Anhang: Anforderungen

Funktion	Punkte	C. Heiden	D. Metzler
kDas Programm ist eine Qt-Anwendung mit einem Hauptfenster.	1	x	
Die Größe des Fensters kann mit den üblichen Methoden verändert werden, das GUI passt sich der Größenveränderung automatisch an.	1	x	
Es gibt es einen Vollbild-Modus	1	x	
Die Anwendung ist lokalisiert und das Benutzerinterface ist in mind. zwei Sprachen (z.B. englisch und deutsch) verfügbar.	4	x	x
Die Applikation läuft auf einem Desktop-Rechner, wird aber auch auf einer mobilen Plattform getestet und wenn nötig angepasst. Das Ergebnis ist in der Dokumentation festzuhalten	4	x	x
Die Anwendung reagiert weiterhin auf Benutzereingaben, wenn längere Berechnungen anstehen. Beispielsweise soll es möglich sein, eine länger dauernde Berechnung abubrechen.	4		
Es wird ein Versionsverwaltungssystem wie git oder svn benutzt. Die Commit-Kommentare beschreiben genau, was gemacht wurde. Build-Artefakte beenden sich nicht im Repository.	4	x	x
Klassen, Funktionen und Schnittstellen sind ausführlich im Quelltext im Doxygen oder qdoc-Format dokumentiert. Die Doxygen- bzw. qdoc-basierende Dokumentation lässt sich einfach mittels make generieren.	4	x	x
Verwendete Algorithmen und Datenstrukturen sind möglichst Qt-unabhängig zu entwickeln, um eine anderweitige Nutzung zu gewährleisten.	4	x	x

Fehlerfreies Funktionieren der implementierten Algorithmen und Datenstrukturen ist mit Unit-Tests zu belegen und deren Erfolg ist im Beleg zu dokumentieren.	5		x
Werkzeuge wie valgrind und AddressSanitizer werfen bei der Ausführung des Programms und der Unit-Tests keine Fehlermeldungen aus. Der Erfolg ist im Beleg zu dokumentieren.	4		x
Das Programm lässt sich einfach für eine Linux-Plattform übersetzen und die Tests laufen dabei automatisch ab. Build-Artefakte werden sowohl für Linux und für eine weitere Plattform wie Windows oder OS X erzeugt und optional für eine mobilen Plattform. Das Bauen der Artefakte wird über einer Containerlösung wie Docker realisiert.	4		
Beim Starten des Programms wird ein Willkommenston abgespielt.	1	x	
Es existiert ein Diashowmodus, der durch einen Doppelklick auf ein Bild gestartet wird.	5	x	x
Die Hintergrundgrafik beim ersten Fenster wird mit QScene erstellt.	1	x	x
Der Benutzer soll Bilder per Dateiauswahl ins Programm laden können.	2	x	
Es besteht die Möglichkeit die Hintergrundfarbe zu ändern.	1	x	
Der Benutzer kann die Anzahl der in einem Fenster dargestellten Bilder ändern.	1	x	
Nach dem Beenden des Programms wird beim nächsten Neustart gefragt, ob der zuletzt geöffnete Ordner wieder geöffnet werden soll.	1	x	x
Es gibt die Menüpunkte Hilfe und Optionen.	3	x	
Erstellen eines Desktop-Icons.	1	x	x
Der Benutzer kann den Bildern Noten von 1 bis 5 geben.	2	x	x
Im Programm ist eine Datenbank integriert, um IDs, Tags, Pfad und Bewertungen abzuspeichern.	4	x	x
Der Benutzer kann die Bilder mit Tags versehen.	2	x	x
Der Benutzer kann nach Tags und nach Noten filtern.	2	x	x
Die Bildinformationen werden beim Anklicken in der Informationsleiste angezeigt	3	x	x
Schnittstelle mit Kinect	5	x	
Der Benutzer soll den Großteil des Diashowmoduses mit Gesten steuern können.	9	x	x
Auf dem Rechner wird nach jpg_Bildern gesucht	2	x	

Tab. 1 zeigt die alle Anforderungen, die es zu erfüllen gab. Zu jeder Anforderung stehen die Punkte und welche Person sie integriert hat. Bei Anforderungen bei dem keine Bearbeiter eingetragen wurde, bedeutet das, dass man jene Anforderungen nicht erfüllt hat.

Geste	Funktion	Punkte	C.Heiden	D.Metzler
Pushen des Bildes	Doppelklick auf das Bild	1		
Hand nach links/rechts bewegen	Folgendes bzw. vorheriges Bild anzeigen	2	x	x
Hand im Kreis drehen	Bild wird um 45° gedreht	1	x	x
Handbewegung senkrecht nach oben bzw. nach unten	Vergrößerung bzw. Verkleinerung des Bildes	2	x	x
Bildbewertung/	Über Sterne wischen	1		
Hand zur Faust ballen für längere Zeit	Beendet Gestensteuerung	1	x	
Aktivierung der Gestensteuerung	Gestenmodus aktivieren	1	x	

Tab. 2 zeigt die alle Gesten, die es mithilfe der Kinect v2 integrieren sollte. Zu jeder Geste stehen die Punkte, die Funktion und welche Person sie integriert hat. Bei Anforderungen bei dem keine Bearbeiter eingetragen wurde, bedeutet das, dass man jene Anforderungen nicht erfüllt hat.

8.3 Anhang: Tests

```
***** Start testing of datenbanktest *****
Config: Using QTest library 5.5.0, Qt 5.5.0 (i386-little_endian-ilp32
shared (dynamic) debug build; by GCC 4.9.2)
QDEBUG : datenbanktest::initTestCase() Database: connection ok
QDEBUG : datenbanktest::initTestCase() Table created!
PASS : datenbanktest::initTestCase()
PASS : datenbanktest::datenbankEmptyTrue()
QDEBUG : datenbanktest::neuesBild() Bild hinzugefuegt!
PASS : datenbanktest::neuesBild()
PASS : datenbanktest::datenbankEmptyFalse()
PASS : datenbanktest::bildpfadExistsFalse()
PASS : datenbanktest::bildpfadExistsTrue()
PASS : datenbanktest::bildExistsTrue()
PASS : datenbanktest::bildExistsFalse()
QDEBUG : datenbanktest::getIDTrue() ID gefunden: 1
PASS : datenbanktest::getIDTrue()
PASS : datenbanktest::getIDFalse()
QDEBUG : datenbanktest::bildBewertenTrue() Bild bewertet!
PASS : datenbanktest::bildBewertenTrue()
QDEBUG : datenbanktest::bildBewertenFalse() Bildbewerten error:
QSqlError("19", "Unable to fetch row", "CHECK constraint failed: Bilder")
PASS : datenbanktest::bildBewertenFalse()
QDEBUG : datenbanktest::bildtagsAendernTrue() Tags geaendert!
PASS : datenbanktest::bildtagsAendernTrue()
PASS : datenbanktest::bildtagsAendernFalse()
QDEBUG : datenbanktest::aktuellenBildPfadAnzeigenTrue() "./Testbild.jpg"
PASS : datenbanktest::aktuellenBildPfadAnzeigenTrue()
PASS : datenbanktest::aktuellenBildPfadAnzeigenFalse()
PASS : datenbanktest::bewertungAnzeigen()
QDEBUG : datenbanktest::bildtagsAnzeigen() "Test"
PASS : datenbanktest::bildtagsAnzeigen()
```



```

QDEBUG : datenbanktest::getBildausrichtung() 0
PASS : datenbanktest::getBildausrichtung()
QDEBUG : datenbanktest::bildDrehen() 0
QDEBUG : datenbanktest::bildDrehen() Bild gedreht!
QDEBUG : datenbanktest::bildDrehen() 90
PASS : datenbanktest::bildDrehen()
QDEBUG : datenbanktest::bewertungFiltern() Alle Bild_dargestellt auf false
gesetzt
QDEBUG : datenbanktest::bewertungFiltern() "../Testbild.jpg"
PASS : datenbanktest::bewertungFiltern()
QDEBUG : datenbanktest::bildtagsFiltern() Alle Bild_dargestellt auf false
gesetzt
QDEBUG : datenbanktest::bildtagsFiltern() "../Testbild.jpg"
PASS : datenbanktest::bildtagsFiltern()
PASS : datenbanktest::bildLoeschenFalse()
QDEBUG : datenbanktest::bildLoeschenTrue() Bild geloescht!
PASS : datenbanktest::bildLoeschenTrue()
QDEBUG : datenbanktest::cleanupTestCase() Table dropped!
PASS : datenbanktest::cleanupTestCase()
Totals: 25 passed, 0 failed, 0 skipped, 0 blacklisted
***** Finished testing of datenbanktest *****

```

Die Programmausgabe zeigt 25 erfolgreich durchlaufene Unit-Tests.

```

Dr. Memory version 1.10.1 build 3 built on Apr 10 2016 18:06:45
Dr. Memory results for pid 2236: "Database_gettingReal.exe"
Application cmdline: "\"C:\\Users\\Delia\\Documents\\Studium\\Sommersemester
2016\\Entwicklung Multimediasysteme\\Projekt\\Datenbank_Test\\build-
Database_gettingReal-Desktop_Qt_5_5_0_MinGW_32bit-
Release\\release\\Database_gettingReal.exe\""
Recorded 115 suppression(s) from default C:\\Program Files (x86)\\Dr.
Memory\\bin\\suppress-default.txt

```

WARNING: application is missing line number information.

```

Error #1: UNINITIALIZED READ: reading register ecx
# 0 sqlite.dll!?          +0x0      (0x6d37b9c4
<sqlite.dll+0x3b9c4>)
# 1 sqlite.dll!?          +0x0      (0x6d37c4b4
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!?          +0x0      (0x6d37c831
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!?          +0x0      (0x6d381027
<sqlite.dll+0x41027>)
# 4 sqlite.dll!?          +0x0      (0x6d381589
<sqlite.dll+0x41589>)
# 5 sqlite.dll!?          +0x0      (0x6d3a25b3
<sqlite.dll+0x625b3>)
# 6 sqlite.dll!?          +0x0      (0x6d3a7d9e
<sqlite.dll+0x67d9e>)
# 7 sqlite.dll!?          +0x0      (0x6d344b2c
<sqlite.dll+0x4b2c>)
# 8 sqlite.dll!?          +0x0      (0x6d346050
<sqlite.dll+0x6050>)
# 9 Qt5Sql.dll!?          +0x0      (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
#10 Database_gettingReal.exe!? +0x0      (0x00401a77
<Database_gettingReal.exe+0x1a77>)

```

```

#11 Qt5Core.dll!?          +0x0      (0x6889b0bf
<Qt5Core.dll+0x1b0bf>)
#12 Qt5Core.dll!?          +0x0      (0x68a0ad1c
<Qt5Core.dll+0x18ad1c>)
#13 Qt5Test.dll!?          +0x0      (0x6edc1693
<Qt5Test.dll+0x1693>)
#14 Database_gettingReal.exe!? +0x0      (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#15 Database_gettingReal.exe!? +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
#16 KERNEL32.dll!BaseThreadInitThunk +0x23      (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:11.201 in thread 3656
Note: instruction: movzx 0x6d3ff283(%ecx) -> %esi

```

```

Error #2: UNINITIALIZED READ: reading register esi
# 0 sqlite.dll!?          +0x0      (0x6d37b89b
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!?          +0x0      (0x6d37c4b4
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!?          +0x0      (0x6d37c831
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!?          +0x0      (0x6d381027
<sqlite.dll+0x41027>)
# 4 sqlite.dll!?          +0x0      (0x6d381589
<sqlite.dll+0x41589>)
# 5 sqlite.dll!?          +0x0      (0x6d3a25b3
<sqlite.dll+0x625b3>)
# 6 sqlite.dll!?          +0x0      (0x6d3a7d9e
<sqlite.dll+0x67d9e>)
# 7 sqlite.dll!?          +0x0      (0x6d344b2c
<sqlite.dll+0x4b2c>)
# 8 sqlite.dll!?          +0x0      (0x6d346050
<sqlite.dll+0x6050>)
# 9 Qt5Sql.dll!?          +0x0      (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
#10 Database_gettingReal.exe!? +0x0      (0x00401a77
<Database_gettingReal.exe+0x1a77>)
#11 Qt5Core.dll!?          +0x0      (0x6889b0bf
<Qt5Core.dll+0x1b0bf>)
#12 Qt5Core.dll!?          +0x0      (0x68a0ad1c
<Qt5Core.dll+0x18ad1c>)
#13 Qt5Test.dll!?          +0x0      (0x6edc1693
<Qt5Test.dll+0x1693>)
#14 Database_gettingReal.exe!? +0x0      (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#15 Database_gettingReal.exe!? +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
#16 KERNEL32.dll!BaseThreadInitThunk +0x23      (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:11.223 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

```

Error #3: UNINITIALIZED READ: reading register esi
# 0 sqlite.dll!?          +0x0      (0x6d37b89b
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!?          +0x0      (0x6d37c4b4
<sqlite.dll+0x3c4b4>)

```

```

# 2 sqlite.dll!? +0x0 (0x6d37c831
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!? +0x0 (0x6d383adb
<sqlite.dll+0x43adb>)
# 4 sqlite.dll!? +0x0 (0x6d387833
<sqlite.dll+0x47833>)
# 5 sqlite.dll!? +0x0 (0x6d3a023f
<sqlite.dll+0x6023f>)
# 6 sqlite.dll!? +0x0 (0x6d3a7d9e
<sqlite.dll+0x67d9e>)
# 7 sqlite.dll!? +0x0 (0x6d344b2c
<sqlite.dll+0x4b2c>)
# 8 sqlite.dll!? +0x0 (0x6d346050
<sqlite.dll+0x6050>)
# 9 Qt5Sql.dll!? +0x0 (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
#10 Database_gettingReal.exe!? +0x0 (0x004026dd
<Database_gettingReal.exe+0x26dd>)
#11 Database_gettingReal.exe!? +0x0 (0x00408cb0
<Database_gettingReal.exe+0x8cb0>)
#12 Qt5Core.dll!? +0x0 (0x68a0ad1c
<Qt5Core.dll+0x18ad1c>)
#13 Qt5Core.dll!? +0x0 (0x68a0f9a1
<Qt5Core.dll+0x18f9a1>)
#14 Qt5Test.dll!? +0x0 (0x6edc7fb8
<Qt5Test.dll+0x7fb8>)
#15 Qt5Test.dll!? +0x0 (0x6edc8aa5
<Qt5Test.dll+0x8aa5>)
#16 Database_gettingReal.exe!? +0x0 (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#17 Database_gettingReal.exe!? +0x0 (0x004013de
<Database_gettingReal.exe+0x13de>)
#18 KERNEL32.dll!BaseThreadInitThunk +0x23 (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:12.525 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

Error #4: UNINITIALIZED READ: reading register esi

```

# 0 sqlite.dll!? +0x0 (0x6d37b89b
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!? +0x0 (0x6d37c4b4
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!? +0x0 (0x6d37c831
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!? +0x0 (0x6d3879bf
<sqlite.dll+0x479bf>)
# 4 sqlite.dll!? +0x0 (0x6d3a023f
<sqlite.dll+0x6023f>)
# 5 sqlite.dll!? +0x0 (0x6d3a7d9e
<sqlite.dll+0x67d9e>)
# 6 sqlite.dll!? +0x0 (0x6d344b2c
<sqlite.dll+0x4b2c>)
# 7 sqlite.dll!? +0x0 (0x6d346050
<sqlite.dll+0x6050>)
# 8 Qt5Sql.dll!? +0x0 (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
# 9 Database_gettingReal.exe!? +0x0 (0x00404892
<Database_gettingReal.exe+0x4892>)

```

```

#10 Database_gettingReal.exe!?      +0x0      (0x004090ca
<Database_gettingReal.exe+0x90ca>)
#11 Qt5Core.dll!?                  +0x0      (0x68a0ad1c
<Qt5Core.dll+0x18ad1c>)
#12 Qt5Core.dll!?                  +0x0      (0x68a0f9a1
<Qt5Core.dll+0x18f9a1>)
#13 Qt5Test.dll!?                  +0x0      (0x6edc7fb8
<Qt5Test.dll+0x7fb8>)
#14 Qt5Test.dll!?                  +0x0      (0x6edc8aa5
<Qt5Test.dll+0x8aa5>)
#15 Database_gettingReal.exe!?      +0x0      (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#16 Database_gettingReal.exe!?      +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
#17 KERNEL32.dll!BaseThreadInitThunk +0x23     (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:12.847 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

```

Error #5: UNINITIALIZED READ: reading register esi
# 0 sqlite.dll!?                   +0x0      (0x6d37b89b
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!?                   +0x0      (0x6d37c4b4
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!?                   +0x0      (0x6d37c831
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!?                   +0x0      (0x6d3879bf
<sqlite.dll+0x479bf>)
# 4 sqlite.dll!?                   +0x0      (0x6d3a023f
<sqlite.dll+0x6023f>)
# 5 sqlite.dll!?                   +0x0      (0x6d3a7d9e
<sqlite.dll+0x67d9e>)
# 6 sqlite.dll!?                   +0x0      (0x6d344b2c
<sqlite.dll+0x4b2c>)
# 7 sqlite.dll!?                   +0x0      (0x6d346050
<sqlite.dll+0x6050>)
# 8 Qt5Sql.dll!?                   +0x0      (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
# 9 Database_gettingReal.exe!?      +0x0      (0x004052a1
<Database_gettingReal.exe+0x52a1>)
#10 Database_gettingReal.exe!?      +0x0      (0x00409188
<Database_gettingReal.exe+0x9188>)
#11 Qt5Core.dll!?                  +0x0      (0x68a0ad1c
<Qt5Core.dll+0x18ad1c>)
#12 Qt5Core.dll!?                  +0x0      (0x68a0f9a1
<Qt5Core.dll+0x18f9a1>)
#13 Qt5Test.dll!?                  +0x0      (0x6edc7fb8
<Qt5Test.dll+0x7fb8>)
#14 Qt5Test.dll!?                  +0x0      (0x6edc8aa5
<Qt5Test.dll+0x8aa5>)
#15 Database_gettingReal.exe!?      +0x0      (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#16 Database_gettingReal.exe!?      +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
#17 KERNEL32.dll!BaseThreadInitThunk +0x23     (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:13.043 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

```

Error #6: UNINITIALIZED READ: reading register esi
# 0 sqlite.dll!?          +0x0      (0x6d37b89b)
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!?          +0x0      (0x6d37c4b4)
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!?          +0x0      (0x6d37c831)
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!?          +0x0      (0x6d386fed)
<sqlite.dll+0x46fed>)
# 4 sqlite.dll!?          +0x0      (0x6d39fb08)
<sqlite.dll+0x5fb08>)
# 5 sqlite.dll!?          +0x0      (0x6d3a7d9e)
<sqlite.dll+0x67d9e>)
# 6 sqlite.dll!?          +0x0      (0x6d344b2c)
<sqlite.dll+0x4b2c>)
# 7 sqlite.dll!?          +0x0      (0x6d346050)
<sqlite.dll+0x6050>)
# 8 Qt5Sql.dll!?          +0x0      (0x6d7c20e9)
<Qt5Sql.dll+0x20e9>)
# 9 Database_gettingReal.exe!? +0x0      (0x0040390e)
<Database_gettingReal.exe+0x390e>)
#10 Database_gettingReal.exe!? +0x0      (0x004094b2)
<Database_gettingReal.exe+0x94b2>)
#11 Qt5Core.dll!?          +0x0      (0x68a0ad1c)
<Qt5Core.dll+0x18ad1c>)
#12 Qt5Core.dll!?          +0x0      (0x68a0f9a1)
<Qt5Core.dll+0x18f9a1>)
#13 Qt5Test.dll!?          +0x0      (0x6edc7fb8)
<Qt5Test.dll+0x7fb8>)
#14 Qt5Test.dll!?          +0x0      (0x6edc8aa5)
<Qt5Test.dll+0x8aa5>)
#15 Database_gettingReal.exe!? +0x0      (0x0040af85)
<Database_gettingReal.exe+0xaf85>)
#16 Database_gettingReal.exe!? +0x0      (0x004013de)
<Database_gettingReal.exe+0x13de>)
#17 KERNEL32.dll!BaseThreadInitThunk +0x23      (0x74ab38f4)
<KERNEL32.dll+0x138f4>)
Note: @0:00:13.372 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

```

Error #7: UNINITIALIZED READ: reading register esi
# 0 sqlite.dll!?          +0x0      (0x6d37b89b)
<sqlite.dll+0x3b89b>)
# 1 sqlite.dll!?          +0x0      (0x6d37c4b4)
<sqlite.dll+0x3c4b4>)
# 2 sqlite.dll!?          +0x0      (0x6d37c831)
<sqlite.dll+0x3c831>)
# 3 sqlite.dll!?          +0x0      (0x6d386fed)
<sqlite.dll+0x46fed>)
# 4 sqlite.dll!?          +0x0      (0x6d39fb08)
<sqlite.dll+0x5fb08>)
# 5 sqlite.dll!?          +0x0      (0x6d3a7d9e)
<sqlite.dll+0x67d9e>)
# 6 sqlite.dll!?          +0x0      (0x6d344b2c)
<sqlite.dll+0x4b2c>)
# 7 sqlite.dll!?          +0x0      (0x6d346050)
<sqlite.dll+0x6050>)

```

```

# 8 Qt5Sql.dll!? +0x0 (0x6d7c20e9
<Qt5Sql.dll+0x20e9>)
# 9 Database_gettingReal.exe!? +0x0 (0x00401fa6
<Database_gettingReal.exe+0x1fa6>)
#10 Database_gettingReal.exe!? +0x0 (0x004094fe
<Database_gettingReal.exe+0x94fe>)
#11 Qt5Test.dll!? +0x0 (0x6edc1693
<Qt5Test.dll+0x1693>)
#12 Database_gettingReal.exe!? +0x0 (0x0040af85
<Database_gettingReal.exe+0xaf85>)
#13 Database_gettingReal.exe!? +0x0 (0x004013de
<Database_gettingReal.exe+0x13de>)
#14 KERNEL32.dll!BaseThreadInitThunk +0x23 (0x74ab38f4
<KERNEL32.dll+0x138f4>)
Note: @0:00:13.511 in thread 3656
Note: instruction: movzx 0x6d3ff283(%esi) -> %edx

```

Error #8: POSSIBLE LEAK 212 direct bytes 0x034f00f8-0x034f01cc + 0 indirect bytes

```

# 0 replace_malloc
[d:\drmmemory_package\common\alloc_replace.c:2576]
# 1 msvcrt.dll!realloc +0x414 (0x77a27e05
<msvcrt.dll+0x47e05>)
# 2 msvcrt.dll!unlock +0x36c (0x77a477dd
<msvcrt.dll+0x677dd>)
# 3 msvcrt.dll!_getmainargs +0x18 (0x77a15799
<msvcrt.dll+0x35799>)
# 4 Database_gettingReal.exe!? +0x0 (0x0040116a
<Database_gettingReal.exe+0x116a>)
# 5 Database_gettingReal.exe!? +0x0 (0x00401451
<Database_gettingReal.exe+0x1451>)
# 6 KERNEL32.dll!BaseThreadInitThunk +0x23 (0x74ab38f4
<KERNEL32.dll+0x138f4>)

```

Error #9: LEAK 24 direct bytes 0x034f0690-0x034f06a8 + 0 indirect bytes

```

# 0 replace_operator_new_array
[d:\drmmemory_package\common\alloc_replace.c:2928]
# 1 Qt5Core.dll!? +0x0 (0x6889d19f
<Qt5Core.dll+0x1d19f>)
# 2 Qt5Test.dll!? +0x0 (0x6edc86df
<Qt5Test.dll+0x86df>)
# 3 Database_gettingReal.exe!? +0x0 (0x0040af85
<Database_gettingReal.exe+0xaf85>)
# 4 Database_gettingReal.exe!? +0x0 (0x004013de
<Database_gettingReal.exe+0x13de>)
# 5 KERNEL32.dll!BaseThreadInitThunk +0x23 (0x74ab38f4
<KERNEL32.dll+0x138f4>)

```

Error #10: LEAK 22 direct bytes 0x034f19d8-0x034f19ee + 0 indirect bytes

```

# 0 replace_operator_new_array
[d:\drmmemory_package\common\alloc_replace.c:2928]
# 1 Qt5Core.dll!? +0x0 (0x6889d19f
<Qt5Core.dll+0x1d19f>)
# 2 Qt5Test.dll!? +0x0 (0x6edc872c
<Qt5Test.dll+0x872c>)
# 3 Database_gettingReal.exe!? +0x0 (0x0040af85
<Database_gettingReal.exe+0xaf85>)

```

```
# 4 Database_gettingReal.exe!?          +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
# 5 KERNEL32.dll!BaseThreadInitThunk    +0x23      (0x74ab38f4
<KERNEL32.dll+0x138f4>)
```

Error #11: POSSIBLE LEAK 5 direct bytes 0x03527b58-0x03527b5d + 0 indirect bytes

```
# 0 replace_malloc
[d:\drmmemory_package\common\alloc_replace.c:2576]
# 1 libgcc_s_dw2-1.dll!?                +0x0      (0x6e9550ce
<libgcc_s_dw2-1.dll+0x150ce>)
# 2 Qt5Core.dll!?                      +0x0      (0x6888dd22
<Qt5Core.dll+0xdd22>)
# 3 Qt5Core.dll!?                      +0x0      (0x6888f507
<Qt5Core.dll+0xf507>)
# 4 Qt5Core.dll!?                      +0x0      (0x68946b59
<Qt5Core.dll+0xc6b59>)
# 5 Database_gettingReal.exe!?          +0x0      (0x004019c4
<Database_gettingReal.exe+0x19c4>)
# 6 Database_gettingReal.exe!?          +0x0      (0x00408c1c
<Database_gettingReal.exe+0x8c1c>)
# 7 Qt5Test.dll!?                      +0x0      (0x6edc1693
<Qt5Test.dll+0x1693>)
# 8 Database_gettingReal.exe!?          +0x0      (0x0040af85
<Database_gettingReal.exe+0xaf85>)
# 9 Database_gettingReal.exe!?          +0x0      (0x004013de
<Database_gettingReal.exe+0x13de>)
#10 KERNEL32.dll!BaseThreadInitThunk    +0x23      (0x74ab38f4
<KERNEL32.dll+0x138f4>)
```

=====

FINAL SUMMARY:

DUPLICATE ERROR COUNTS:

```
Error # 1:      224
Error # 2:        4
Error # 3:        4
Error # 4:        4
Error # 5:        4
Error # 6:        4
Error # 7:        4
```

SUPPRESSIONS USED:

ERRORS FOUND:

```
0 unique,      0 total unaddressable access(es)
7 unique,     248 total uninitialized access(es)
0 unique,      0 total invalid heap argument(s)
0 unique,      0 total GDI usage error(s)
0 unique,      0 total handle leak(s)
0 unique,      0 total warning(s)
2 unique,      2 total,      46 byte(s) of leak(s)
2 unique,      2 total,     217 byte(s) of possible leak(s)
```

ERRORS IGNORED:

```
5 potential error(s) (suspected false positives)
(details: C:\Users\Delia\AppData\Roaming\Dr. Memory\DrMemory-
Database_gettingReal.exe.2236.000\potential_errors.txt)
```

```
1 potential leak(s) (suspected false positives)
  (details: C:\Users\Delia\AppData\Roaming\Dr. Memory\DrMemory-
Database_gettingReal.exe.2236.000\potential_errors.txt)
  26 unique,    73 total,  13371 byte(s) of still-reachable
allocation(s)
  (re-run with "-show_reachable" for details)
Details: C:\Users\Delia\AppData\Roaming\Dr. Memory\DrMemory-
Database_gettingReal.exe.2236.000\results.txt
```

Die Ergebnisse des Überprüfens der Datenbank und der Unit-Tests mit Dr. Memory.