**Projekt Bildverarbeitung-Dokumentation**

Autor: Anita Sherchan, Daniel Salib, Niklas Rother

Datum (letztes Update):

**Übersicht**

Übersicht Beschreibung: In diesem Projekt wollen wir ein interaktives Tool bereitstellen, mit dem man Basis Methoden der Digitalen Bildverarbeitung anwenden kann. Unser Ziel ist es eine möglichst benutzerfreundliche Anwendung zu schaffen, welche nahezu ohne Erklärung und mehrseitige Dokumentation auskommt. „Picture in Picture out“ ist unser Motto, das Tool soll einem die Möglichkeit geben ein Bild zu übergeben, diese zu verändern und es dann wieder abspeichern zu können. Verwendete Bibliotheken: Qimage: Diese Bibliothek benötigen wir um Bilder einzuladen und diese Bibliothek stellt auch Basisfunktionalitäten wie Skalierung oder Interpolation dar.

Beschreibung:

Der BildModell repräsentiert das Modell für ein Bild in der Anwendung. Es enthält Methoden, um Eigenschaften des Bildes zu erhalten, die Bildansicht zu ändern (Zoom, Skalierung, Rotation) und Änderungen im Bild zu signalisieren.

Methoden:

* QPoint BildModell::getPos() const:
* **Beschreibung:** Gibt die aktuelle Position des Mauszeigers zurück, die im Bildmodell gespeichert ist.
* void BildModell::setPos(const QPoint& pos):
* **Beschreibung:** Setzt die aktuelle Mauszeigerposition im Bildmodell.
* QPixmap BildModell::getImage() const:
* **Beschreibung:** Gibt das Bild (als QPixmap) zurück, das im Bildmodell gespeichert ist.
* QRect BildModell::getRecF() const:
* **Beschreibung:** Gibt das Rechteck zurück, das die Dimensionen des aktuellen Bildausschnitts im Bildmodell repräsentiert.
* void BildModell::zoomIn(QString rectangle):
* **Beschreibung:** Vergrößert den Bildausschnitt basierend auf der ausgewählten Rechteckgröße (Pixelgröße, 75x75, 150x150, oder eine benutzerdefinierte Größe). Das Bildmodell emittiert ein Signal (imageChanged) nach der Änderung.
* void BildModell::scaleImage(int scale):
* **Beschreibung:** Skaliert das Bild anhand des übergebenen Skalierungsfaktors. Das Bildmodell emittiert ein Signal (imageChanged) nach der Änderung.
* void BildModell::rotateImage(int degree):
* **Beschreibung:** Rotiert das Bild um den übergebenen Grad. Dabei wird ein Transformationsmatrix verwendet, um die Rotation zu verarbeiten. Das Bildmodell emittiert ein Signal (imageChanged) nach der Änderung.

Beschreibung:

Die Header-Datei BildModell.h definiert die Klasse BildModell, die als Modell für ein Bild in der Anwendung fungiert. Sie enthält Methoden, um Eigenschaften des Bildes zu verwalten, wie die Position des Mauszeigers, die Bildgröße und die Bildmanipulation.

Klassen und Strukturen:

Klassenname: BildModell

**Beschreibung:** Repräsentiert das Modell eines Bildes in der Anwendung.

**Attribute:**

* image: Ein QPixmap, das das Bild repräsentiert.
* currentMousePosition: Die aktuelle Position des Mauszeigers im Bild.
* pixelSize: Die Größe des Bildes in Pixeln.
* rectImage: Ein QRect, das den aktuellen Bildausschnitt repräsentiert.
* transformationMatrix: Eine QTransform, die für Transformationen wie Rotation verwendet wird.
* ImageInput: Eine Konstante, die eine unveränderte Kopie des ursprünglichen Bildes speichert.
* scaleFactor: Ein Faktor, der für die Bildskalierung verwendet wird.
* rotationFactor: Der aktuelle Rotationswinkel des Bildes.

**Methoden:**

* BildModell(QObject\* parent, const QString& file):
* **Beschreibung:** Konstruktor der Klasse. Initialisiert die Attribute und lädt das Bild aus einer Datei.
* void setPos(const QPoint& pos):
* **Beschreibung:** Setzt die aktuelle Position des Mauszeigers im Bildmodell.
* QPoint getPos() const:
* **Beschreibung:** Gibt die aktuelle Position des Mauszeigers im Bild zurück.
* QPixmap getImage() const:
* **Beschreibung:** Gibt das Bild (als QPixmap) zurück, das im Bildmodell gespeichert ist.
* QRect getRecF() const:
* **Beschreibung:** Gibt das Rechteck zurück, das die Dimensionen des aktuellen Bildausschnitts im Bildmodell repräsentiert.

**Signals:**

* posChanged(): Signalisiert eine Änderung der Mauszeigerposition.
* pixelSizeChanged(): Signalisiert eine Änderung der Bildgröße.
* imageChanged(): Signalisiert eine Änderung im Bild.

**Slots:**

* void scaleImage(int scale):
* **Beschreibung:** Skaliert das Bild basierend auf dem übergebenen Skalierungsfaktor.
* void rotateImage(int degree):
* **Beschreibung:** Rotiert das Bild um den angegebenen Winkel.
* void zoomIn(QString rectangle):
* **Beschreibung:** Vergrößert den Bildausschnitt basierend auf der ausgewählten Rechteckgröße.

Beschreibung:

Die Header-Datei BildView.h definiert die Klasse BildView, die eine Ansichtskomponente für die Darstellung eines Bildes in der Benutzeroberfläche repräsentiert.

Klassen und Strukturen:

Klassenname: BildView

**Beschreibung:** Eine Ansichtskomponente für die Darstellung eines Bildes in der Benutzeroberfläche.

**Attribute:**

* pmodell: Eine Referenz auf das zugehörige BildModell, mit dem die BildView interagiert.

**Methoden:**

* BildView(BildModell& modell, QWidget \*parent = nullptr):
* **Beschreibung:** Konstruktor der Klasse. Initialisiert die Ansichtskomponente und verknüpft sie mit dem zugehörigen BildModell.
* ~BildView():
* **Beschreibung:** Destruktor der Klasse. Aufräumarbeiten können hier durchgeführt werden.
* void paintEvent(QPaintEvent \*event):
* **Beschreibung:** Überschriebene Funktion, die beim Zeichnen des Widgets aufgerufen wird. Zeichnet das Bild auf der Ansicht.

**Funktionalität:**

* BildView(BildModell& modell, QWidget \*parent = nullptr):
* Initialisiert die Ansichtskomponente und verbindet sie mit dem zugehörigen BildModell.
* void paintEvent(QPaintEvent \*event):
* Zeichnet das Bild auf der Ansicht. Diese Funktion wird aufgerufen, wenn das Widget gezeichnet wird und verwendet einen QPainter, um das Bild aus dem Modell auf der Ansicht zu zeichnen.

Die BildView Klasse dient dazu, das Bild aus dem BildModell auf der Benutzeroberfläche darzustellen und wird verwendet, um die Ansichtskomponente für das Bild zu erstellen und zu verwalten.

Beschreibung:

Die Klasse BildView ist eine Ansichtskomponente, die für die Darstellung eines Bildes in der Benutzeroberfläche zuständig ist. Sie erhält Informationen aus dem BildModell und aktualisiert sich entsprechend, wenn Änderungen im Modell auftreten.

Klassen und Strukturen:

Klassenname: BildView

**Beschreibung:** Eine Ansichtskomponente, die für die Darstellung eines Bildes in der Benutzeroberfläche verantwortlich ist.

**Attribute:**

* pmodell: Eine Referenz auf das zugehörige BildModell, mit dem die BildView interagiert.

**Methoden:**

* BildView(BildModell& modell, QWidget\* parent):
* **Beschreibung:** Konstruktor der Klasse. Verbindet Signale des Modells mit der Aktualisierung der Ansicht.
* void paintEvent(QPaintEvent \*event):
* **Beschreibung:** Überschriebene Funktion, die beim Zeichnen des Widgets aufgerufen wird. Zeichnet das Bild auf der Ansicht.

**Funktionalität:**

* connect(&modell, &BildModell::posChanged, this, QOverload<>::of(&QWidget::update)):
* Aktualisiert die Ansicht, wenn sich die Mauszeigerposition im Modell ändert.
* connect(&modell, &BildModell::pixelSizeChanged, this, QOverload<>::of(&QWidget::update)):
* Aktualisiert die Ansicht, wenn sich die Bildgröße im Modell ändert.
* connect(&modell, &BildModell::imageChanged, this, QOverload<>::of(&QWidget::update)):
* Aktualisiert die Ansicht, wenn sich das Bild im Modell ändert.
* void paintEvent(QPaintEvent \*event):
* Zeichnet das Bild auf der Ansicht. Dabei wird ein QPainter verwendet, um das Bild aus dem Modell zu zeichnen.

Die BildView Klasse reagiert auf Änderungen im BildModell und aktualisiert sich entsprechend, um die neuesten Informationen des Modells darzustellen.

Beschreibung:

Der BildController fungiert als Vermittler zwischen einem Bildmodell und einer Bildansicht. Seine Rolle besteht darin, eine effiziente Kommunikation zwischen dem Modell und der Ansicht zu ermöglichen.

Attribute:

* pmodell: Ein Wegweiser zum Bildmodell, der es dem Controller ermöglicht, mit dem Modell zu interagieren.
* pview: Ein Wegweiser zur Bildansicht, der dem Controller den Zugriff auf die Ansicht gewährt.

Methoden:

* BildController(BildModell\* modell, BildView\* view, QObject \*parent):
* **Beschreibung:** Erschafft den Controller und teilt ihm mit, welches Bildmodell und welche Bildansicht er kontrollieren soll.
* bool eventFilter(QObject\* watched, QEvent\* event) override:
* **Beschreibung:** Überwacht und entscheidet über die Behandlung der aufgetretenen Ereignisse.
* void mousePressEvent(QMouseEvent\* event):
* **Beschreibung:** Behandelt Mausklick-Ereignisse auf dem Bild.
* void mouseReleaseEvent(QMouseEvent\* event):
* **Beschreibung:** Kümmert sich um das Loslassen der Maustaste auf dem Bild.

Diese Funktionen und Attribute des BildController dienen dazu, die Interaktionen zwischen dem Modell und der Ansicht zu orchestrieren und zu steuern, um eine reibungslose Kommunikation und Interaktion zwischen den beiden zu gewährleisten.

**Klassen und Strukturen**

Klassenname: **BildController**

**Beschreibung:** Klasse zur Steuerung der Interaktionen zwischen einem Bildmodell und einer Bildansicht.

**Attribute:**

* pmodell: Ein Zeiger auf ein BildModell.
* pview: Ein Zeiger auf eine BildView.

**Methoden:**

* BildController(BildModell\* modell, BildView\* view, QObject \*parent):  
  Beschreibung: Konstruktor der Klasse.  
  Parameter:
* modell: Ein Zeiger auf ein BildModell.
* view: Ein Zeiger auf eine BildView.
* parent: Ein Zeiger auf ein QObject. **Rückgabetyp:** Keiner (void).
* bool eventFilter(QObject\* watched, QEvent\* event) override:  
  Beschreibung: Funktion zur Filterung von Ereignissen.  
  Parameter:
* watched: Ein Zeiger auf das beobachtete Objekt.
* event: Ein Zeiger auf das aufgetretene Ereignis. **Rückgabetyp:** Ein bool-Wert, der anzeigt, ob das Ereignis bearbeitet wurde (true) oder nicht (false).
* void mousePressEvent(QMouseEvent\* event):  
  Beschreibung: Behandlung von Mausklick-Ereignissen.  
  Parameter:
* event: Ein Zeiger auf das aufgetretene Mausklick-Ereignis. **Rückgabetyp:** Keiner (void).
* void mouseReleaseEvent(QMouseEvent\* event):  
  Beschreibung: Behandlung von Mausfreigabe-Ereignissen.  
  Parameter:
* event: Ein Zeiger auf das aufgetretene Mausfreigabe-Ereignis. **Rückgabetyp:** Keiner (void).

Beschreibung:

Die Header-Datei BildWidget.h definiert die Klasse BildWidget, die ein Widget ist und von QMainWindow erbt. Es repräsentiert ein Hauptfenster, das mit einem BildModell verknüpft ist und vermutlich zur Anzeige und Interaktion mit dem Bild verwendet wird.

Klassen und Strukturen:

Klassenname: BildWidget

**Beschreibung:** Ein Hauptfenster-Widget, das mit einem BildModell verknüpft ist.

**Attribute:**

* bildModell: Ein Zeiger auf das zugehörige BildModell, mit dem das BildWidget interagiert.

**Methoden:**

* BildWidget():
* **Beschreibung:** Konstruktor der Klasse. Initialisiert das Hauptfenster-Widget.
* ~BildWidget():
* **Beschreibung:** Destruktor der Klasse. Aufräumarbeiten können hier durchgeführt werden.

**Funktionalität:**

* BildWidget():
* Initialisiert das Hauptfenster-Widget. Es wird angenommen, dass hier das BildWidget erstellt wird und möglicherweise eine Verknüpfung mit einem BildModell hergestellt wird.

Die BildWidget-Klasse repräsentiert ein Hauptfenster-Widget und wird vermutlich dazu verwendet, mit einem Bild zu interagieren und das BildModell in der Benutzeroberfläche anzuzeigen.

Beschreibung:

Der Code in der Datei implementiert die Konstruktionslogik für ein BildWidget, das ein Hauptfenster für die Anzeige und Interaktion mit einem Bild bereitstellt. Es verwendet Elemente aus der Qt-Bibliothek, um verschiedene Bedienelemente und Widgets zu erstellen und sie mit einem BildModell, einer BildView und einem BildController zu verknüpfen, um die Bildinteraktion zu ermöglichen.

Funktionalität:

Der Code erstellt verschiedene Qt-Widgets wie Schieberegler, Buttons, ComboBoxes usw., um die Interaktion mit einem Bild zu ermöglichen. Einige der wichtigen Funktionalitäten und Elemente sind:

* Ein Schieberegler (rotate\_slider) für die Einstellung der Bildrotation.
* Ein SpinBox (scale\_button) zur Skalierung des Bildes.
* Ein ComboBox (button\_auswahl\_pixelgroeße) zur Auswahl verschiedener Bildgrößen.
* Ein QPushButton (speicher\_button) zum Speichern von Bildänderungen.
* Ein QPushButton (edge\_detektion\_button) für Kanten-Erkennung (Edge Detection).
* Ein QDockWidget (dock) auf der rechten Seite für zusätzliche Einstellungen.
* Verbindung der Bedienelemente mit entsprechenden Methoden im BildModell für die Skalierung, Rotation und Auswahl der Bildgröße.

Das BildWidget wird mit einem BildModell, einer BildView und einem BildController verknüpft, um die Bildinteraktion innerhalb des Widgets zu handhaben und darzustellen. Es verwendet verschiedene Signale und Slots, um Änderungen im Bildmodell zu verarbeiten und die Ansicht zu aktualisieren.

**Anmerkungen** **Herausforderungen**: [Eventuelle Herausforderungen oder spezielle Entscheidungen während der Entwicklung]

**Weiterentwicklung**: [Pläne für zukünftige Entwicklungen oder Verbesserungen]

**Hinweise für Entwickler Einrichtung**: [Anleitungen zur Einrichtung des Codes auf anderen Systemen]

**Tests**: [Informationen über vorhandene Tests und wie sie ausgeführt werden können]

**Abschluss** Diese Dokumentation bietet eine Übersicht über den Code und erleichtert anderen Entwicklern das Verständnis und die Arbeit am Projekt. Regelmäßige Aktualisierungen und detaillierte Kommentare tragen zur Pflege und Weiterentwicklung des Codes bei.