Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

Лабораторная работа №1

«Системы мультимедиа, прототипирование и разработка интерфейса веб-проекта»

ДИСЦИПЛИНА: «Проектирование программного обеспечения»

Выполнил: студент гр. ИУК4-11М	______________ (подпись)	(Сафронов Н.С.) (Ф.И.О.)
Проверил:	______________ (подпись)	(Белов Ю.С.) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель работы: формирование и закрепление практических навыков по обработки изображений и работы с различными цветовыми системами.

Постановка задачи

Вариант 7

Создать форму с растровым изображением с возможностью регулировки яркости по всем каналам и по C, M, Y в отдельности.

Листинг программы

main.py

```
import sys

from PyQt6.QtWidgets import QApplication

from ui import MainWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec()
```

ui.py

```
import numpy as np
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QPixmap, QImage
from PyQt6.QtWidgets import (
    QMainWindow, QVBoxLayout, QWidget, QLayout,
    QGroupBox, QSlider, QHBoxLayout, QLabel, QPushButton, QFileDialog,
)

from models.images import BrightnessAdjustment
from services.images import ImageService


class MainWindow(QMainWindow):
    """."""

    def __init__(self):
        super(MainWindow, self).__init__()

        self.setWindowTitle('Brightness Adjustment Tool')

        widget = QWidget()
        layout = self._construct_layout()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
        self._images: ImageService | None = None

    def _construct_image_layout(self):
        self._image_layout = QVBoxLayout()
        self._image_view = QLabel()
        self._image_view.setFixedSize(600, 600)
        self._image_view.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self._image_layout.addWidget(self._image_view)
        self.adjustSize()
        return self._image_layout

    def _construct_control_layout(self):
        brightness_control_layout = QVBoxLayout()

        cyan_label = QLabel('Cyan')
        brightness_control_layout.addWidget(cyan_label)
        self._cyan_slider = QSlider(Qt.Orientation.Horizontal)
        self._cyan_slider.setValue(50)
        self._cyan_slider.valueChanged.connect(self._adjust_channel)
        brightness_control_layout.addWidget(self._cyan_slider)

        magenta_label = QLabel('Magenta')
        brightness_control_layout.addWidget(magenta_label)
        self._magenta_slider = QSlider(Qt.Orientation.Horizontal)
        self._magenta_slider.setValue(50)
        self._magenta_slider.valueChanged.connect(self._adjust_channel)
        brightness_control_layout.addWidget(self._magenta_slider)

        yellow_label = QLabel('Yellow')
        brightness_control_layout.addWidget(yellow_label)
        self._yellow_slider = QSlider(Qt.Orientation.Horizontal)
        self._yellow_slider.setValue(50)
        self._yellow_slider.valueChanged.connect(self._adjust_channel)
        brightness_control_layout.addWidget(self._yellow_slider)

        overall_label = QLabel('All Channels')
        brightness_control_layout.addWidget(overall_label)
        self._overall_slider = QSlider(Qt.Orientation.Horizontal)
        self._overall_slider.setValue(50)
        self._overall_slider.valueChanged.connect(self._adjust_overall)
        brightness_control_layout.addWidget(self._overall_slider)

        brightness_control_group = QGroupBox('Brightness Control')
        brightness_control_group.setLayout(brightness_control_layout)

        image_control_group = QGroupBox('Image Loading')
        image_control_layout = QHBoxLayout()
        button = QPushButton('Load Image')
        button.clicked.connect(self._load_image)
        image_control_layout.addWidget(button)
        image_control_group.setLayout(image_control_layout)

        layout = QVBoxLayout()
        layout.addWidget(brightness_control_group)
        layout.addWidget(image_control_group)
        layout.addStretch(1)

        return layout

    def _construct_layout(self) -> QLayout:
        layout = QHBoxLayout()

        layout.addLayout(self._construct_image_layout())
        layout.addLayout(self._construct_control_layout())

        return layout

    def _load_image(self):
        image = QFileDialog.getOpenFileName(
            None, 'OpenFile', '', "Image file(*.jpg)"
            )
        image_path = image[0]
        pixmap = QPixmap(image_path)
        self._initial_image = pixmap
        self._images = ImageService.from_pixmap(self._initial_image)
        self._set_image(self._images.to_pixmap())

    def _set_image(self, pixmap: QPixmap):
        ratio = pixmap.width() / pixmap.height()

        if pixmap.width() < pixmap.height():
            scaled = pixmap.scaled(
                int(600 * ratio),
                600,
                Qt.AspectRatioMode.KeepAspectRatio,
            )
        else:
            scaled = pixmap.scaled(
                600,
                int(600 / ratio),
                Qt.AspectRatioMode.KeepAspectRatio,
            )

        self._image_view.setPixmap(scaled)
        self._image_layout.update()
        self.adjustSize()

    def _get_brightness_adjustment(self) -> BrightnessAdjustment:
        return BrightnessAdjustment(
            cyan=self._cyan_slider.value() - 50,
            yellow=self._yellow_slider.value() - 50,
            magenta=self._magenta_slider.value() - 50,
        )

    def _adjust_channel(self, _: int):
        self._images.adjust_brightness(self._get_brightness_adjustment())
        self._set_image(self._images.adjusted_to_pixmap())

    def _adjust_overall(self, value: int):
        adjustment = BrightnessAdjustment(
            cyan=value - 50,
            yellow=value - 50,
            magenta=value - 50,
        )
        self._images.adjust_brightness(adjustment)
        self._set_image(self._images.adjusted_to_pixmap())
```

services/images.py

```
import typing as t

import numpy as np
from PyQt6.QtGui import QPixmap, QImage
from models.images import CmyChannels, BrightnessAdjustment


class ImageService:

    def __init__(self, rgb_array: np.array):
        self._image: np.ndarray = rgb_array
        self._adjusted: np.ndarray | None = None
        self._channels = None

    @classmethod
    def from_pixmap(cls, pixmap: QPixmap) -> t.Self:
        image = pixmap.toImage()
        image = image.convertToFormat(QImage.Format.Format_RGB888)

        width = image.width()
        height = image.height()
        ptr = image.bits().asstring(width * height * 3)
        image = np.frombuffer(ptr, np.uint8).reshape((height, width, 3))
        return cls(image)

    def adjusted_to_pixmap(self) -> QPixmap:
        return self._to_pixmap(self._adjusted)

    def to_pixmap(self) -> QPixmap:
        return self._to_pixmap(self._image)

    def adjust_brightness(self, adjustment: BrightnessAdjustment) ->
np.array:
        channels = self._channels or self._parse_image_channels(self._image)
        for channel, percent in adjustment.items():
            channels[channel] = self._adjust_channel_brightness(
                channel=channels[channel],
                percent=percent,
            )
        self._adjusted = self._convert_cmy_to_rgb(channels)
        return self._adjusted

    @classmethod
    def _to_pixmap(cls, image: np.array) -> QPixmap:
        channels = cls._parse_image_channels(image)
        image = cls._convert_cmy_to_rgb(channels)

        height, width, channels = image.shape
        bytes_per_line = channels * width
        return QPixmap.fromImage(
            QImage(
                image.data, width, height,
                bytes_per_line, QImage.Format.Format_RGB888,
            )
        )

    @classmethod
    def _parse_image_channels(cls, rgb_array: np.array) -> CmyChannels:
        cyan = 255 - rgb_array[:, :, 0].astype(np.float64)
        cyan /= cyan.max() or 1

        magenta = 255 - rgb_array[:, :, 1].astype(np.float64)
        magenta /= magenta.max() or 1

        yellow = 255 - rgb_array[:, :, 2].astype(np.float64)
        yellow /= yellow.max() or 1

        return CmyChannels(
            cyan=cyan,
            magenta=magenta,
            yellow=yellow,
        )

    @classmethod
    def _convert_cmy_to_rgb(cls, channels: CmyChannels) -> np.array:
        red = 255 - (channels['cyan'] * 255).astype(np.uint8)
        green = 255 - (channels['magenta'] * 255).astype(np.uint8)
        blue = 255 - (channels['yellow'] * 255).astype(np.uint8)
        return np.dstack([red, green, blue])

    @classmethod
    def _adjust_channel_brightness(
            cls,
            channel: np.ndarray[np.float64],
            percent: float,
    ) -> np.ndarray[np.uint8]:
        channel -= percent / 100
        channel[channel > 1] = 1
        channel[channel < 0] = 0
        return channel
```

models/images.py

```
import typing as t
import numpy as np


class CmyChannels(t.TypedDict):
    cyan: np.ndarray[np.float64]
    magenta: np.ndarray[np.float64]
    yellow: np.ndarray[np.float64]


class BrightnessAdjustment(t.TypedDict):
    cyan: float
    magenta: float
    yellow: float
```

Результаты выполнения лабораторной работы

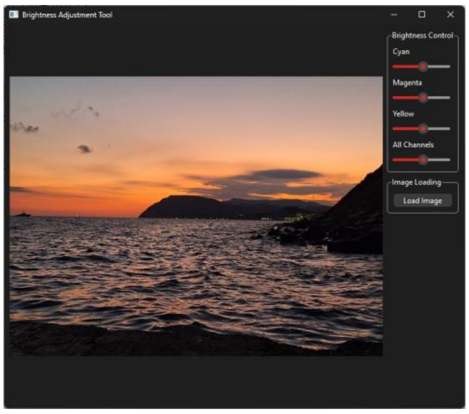

Рисунок 1 – Главное окно приложения

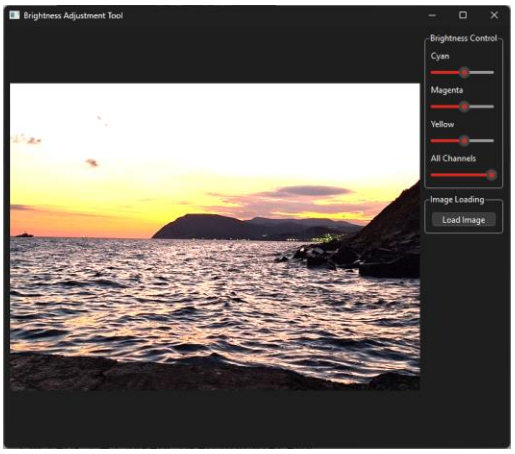

Рисунок 2 – Изображение с отредактированной яркостью

Вывод: в ходе выполнения лабораторной работы были сформированы и закреплены практические навыки по обработки изображений и работы с различными цветовыми системами.