МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ГЕОДЕЗИИ И КАРТОГРАФИИ

(МИИГАиК)

Кафедра ВТиАОАИ

(КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И АВТОМАТИЗИРОВАННОЙ ОБРАБОТКИ АЭРОКОСМИЧЕСКОЙ ИНФОРМАЦИИ)

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

**Автоматизация проектирования аэросъемочных работ**

**Выполнил:** студент ФПКиФ группы ИСиТ III-1б

Зубов Святослав Владимирович

**Преподаватель:** Груздев Сергей Сергеевич

**Содержание**

[Введение 3](#_Toc13160938)

[Работа с AerialWare 3](#_Toc13160939)

[Установка 3](#_Toc13160940)

[Знакомство с программой 3](#_Toc13160941)

[Этап 1. Загрузка изображения. 4](#_Toc13160942)

[Этап 2. Ввод координат. 5](#_Toc13160943)

[Этап 3. Выбор участков и сохранение результатов 6](#_Toc13160944)

[Создание AerialWare 9](#_Toc13160945)

[Введение 9](#_Toc13160946)

[Создание GUI 9](#_Toc13160947)

[Подготовительные работы 11](#_Toc13160948)

[Этап 1. Загрузка изображения. 12](#_Toc13160949)

[Этап 2. Ввод координат. 13](#_Toc13160950)

[Этап 3. Выбор участков и сохранение результатов. 13](#_Toc13160951)

[Дополнительные методы класса окна 16](#_Toc13160952)

[Подклассы элементов Qt 18](#_Toc13160953)

[Класс ячейки сетки 18](#_Toc13160954)

[Класс сцены 19](#_Toc13160955)

# Введение

Практическая работа заключается в разработке программы, позволяющей автоматически строить маршрут полета самолета в зависимости от выбранных пользователем участков на изображении, которые необходимо заснять.

**Особенностью программы** является то, что трансформируется не изображение под координатную сетку в какой-либо проекции, а трансформирование координатной сетки под изображение.

**Работа с программой делится на три этапа:**

1. Загрузка изображения.
2. Ввод координат углов снимка в географической системе координат и шаг сетки по широте и долготе.
3. Выбор нужных участков, которые необходимо заснять, и сохранение результата в виде векторного изображения.

Соответственно, **алгоритм работы программы состоит из следующих этапов:**

1. Загрузка и отображение изображения пользователя.
2. Чтение введенных пользователем данных.
3. Построение и отображение сетки, реагирующей на нажатия, с учетом вышеописанной особенности.
4. Анализ выделенных пользователем участков и построение маршрута полета самолета по меридианам и по горизонталям.
5. Сохранение результатов работы программы.

При разработке использовался язык программирования Python 3, фреймворк Qt 5 и биндинги Qt для Python под названием PyQt5.

Разработанная программа называется **AerialWare** и является бесплатным и открытым программным обеспечением. Исходный код можно найти и скачать по ссылке: [github.com/matafokka/AerialWare](https://github.com/matafokka/AerialWare)

# Работа с AerialWare

## Установка

Программа поставляется без установщика, поэтому установка состоит из следующих этапов:

1. Установка зависимостей:

* Python 3 – можно скачать с сайта [python.org](https://www.python.org/)
* PyQt5 – можно скачать из репозитория используемого дистрибутива или установить через pip, поставляемый с Python 3

1. Скачивание программы по ссылке выше
2. Запуск файла AerialWare.py

## Знакомство с программой

После запуска программы появится такое окно:



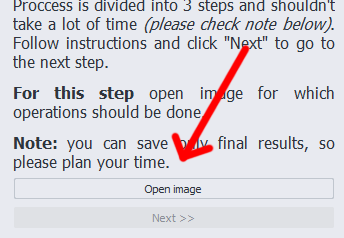
**Описание элементов программы** (отмечены красным на скриншоте выше)**:**

1. Область с изображением. Изображение появится после открытия.
2. Элементы изменения масштаба изображения. Кнопки увеличивают и уменьшают изображение на 10% в соответствии со знаком на них. В поле ввода можно ввести произвольный масштаб.
3. Область, содержащая информацию о каждом этапе и элементы взаимодействия с пользователем.
4. Кнопка «Далее». Переводит пользователя на следующий этап.

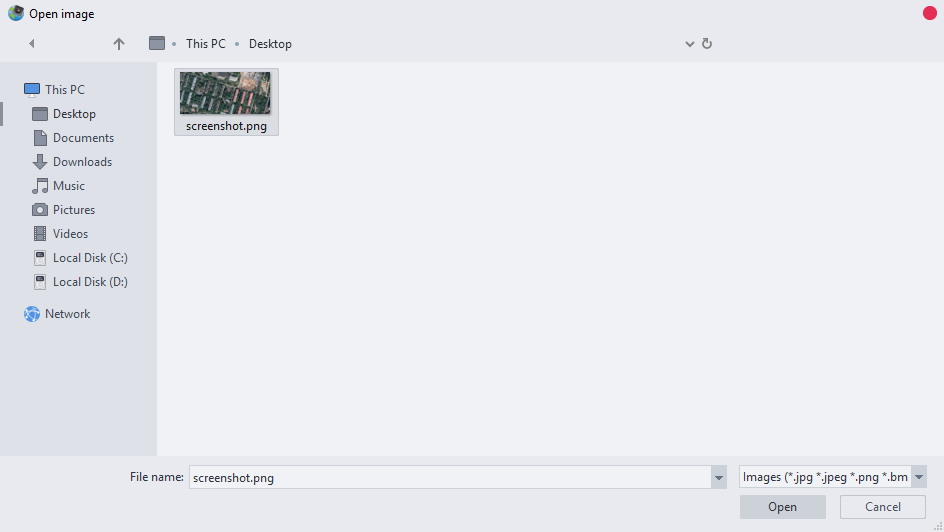
Если навести курсор между областью с изображением и панелью справа, можно изменять размер этих элементов.

## Этап 1. Загрузка изображения.

Для загрузки изображения необходимо нажать кнопку «Open image», после чего появится диалоговое окно выбора файла, в котором необходимо выбрать изображение.



Для проектирования реальных работ используются изображение или карты мелкого масштаба, но для демонстрации работы программы будет использоваться изображение с Google Maps.



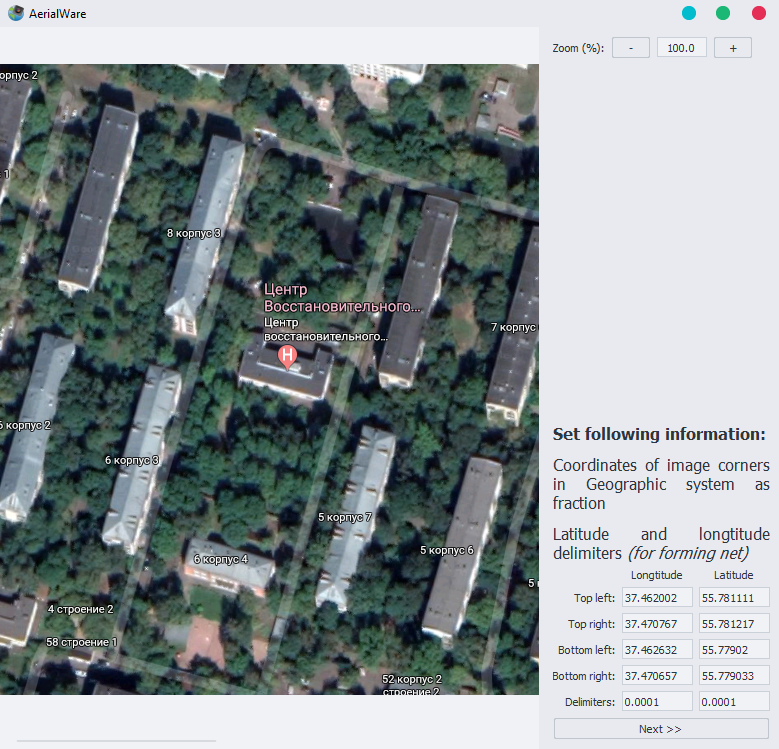
После выбора изображения оно отобразится в программе. Также станут активны средства масштабирования. Для перехода на следующий этап необходимо нажать кнопку «Next».



## Этап 2. Ввод координат.

Обычно координаты уже известны, и идут в комплекте со снимками, либо их берут из ГИС. Google Maps позволяет получить координаты любой точки, они и будут использованы в примере.

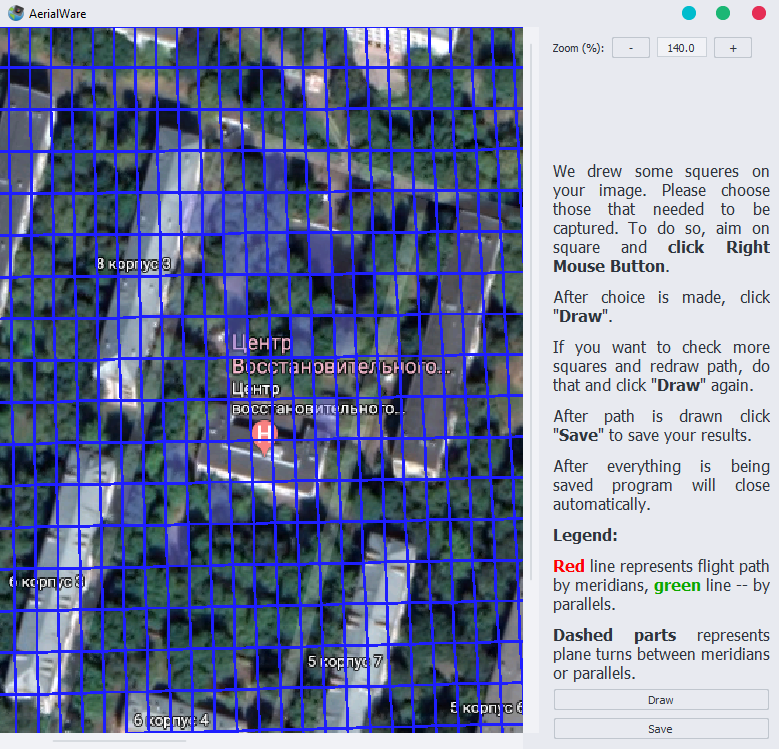
Координаты должны быть введены в географической системе координат в виде десятичной дроби (градусы). Также необходимо ввести шаг сетки по широте и долготе.



После ввода координат необходимо нажать кнопку «Next»

## Этап 3. Выбор участков и сохранение результатов

После выполнения предыдущего этапа программа генерирует координатную сетку, трансформированную в соответствии с геометрией изображения.



Программа может построить пути между участками, не прилегающими друг к другу, поэтому делать выбор можно в абсолютно произвольном порядке.

**Порядок работы с программой на данном этапе:**

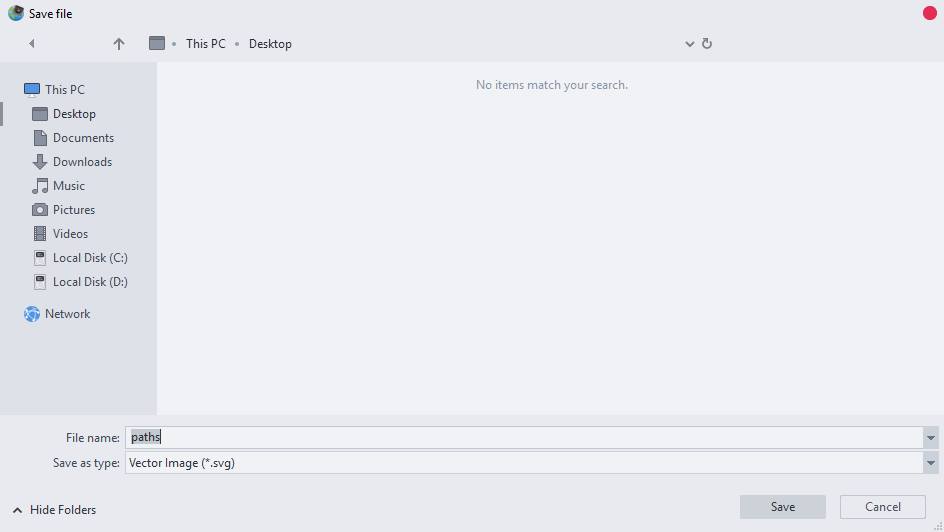
1. Чтобы выбрать участок, нужно кликнуть по нему правой кнопкой мыши. Чтобы отменить выбор, нужно снова кликнуть.
2. Чтобы построить пути, нужно нажать кнопку «Draw».
3. Если необходимо произвести выбор заново, нужно выбрать недостающие квадраты или отменить выбор ненужных и заново нажать кнопку «Draw».
4. Когда получен требуемый результат, нужно нажать на кнопку «Save»



**Легенда построенных путей:**

* Красные линии показывают маршрут при съемке по меридианам, зеленые – по параллелям.
* Пунктирные части линий показывают путь от одного участка съемки до другого (например, переход с одной «строки» на следующую, где самолету необходимо сделать поворот).

При нажатии на кнопку «Save» отобразится диалоговое окно сохранения, в котором нужно выбрать папку, куда сохранить файл, и ввести его имя. Поддерживается сохранение только в формате SVG.



# Создание AerialWare

## Введение

Изначально задумывалась поэтапная работа с программой, что упростит ее использование, поэтому код тоже делится на «этапы». Далее будут разобраны эти «этапы», после – используемые подклассы элементов Qt и функции.

Стоит отметить, что в дальнейшем слово «функция» будет обозначать метод класса, так как все действия являются методами подкласса QMainWindow.

Код достаточно подробно прокомментирован на английском языке, поэтому в отчете описано только общее назначение частей кода.

## Создание GUI

GUI создавался в программе Qt Designer. В таблице ниже приведена иерархическая структура элементов окна, их назначение и лейаут – порядок расположения элементов внутри данного элемента.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Объект** | | | | | | | | **Класс** | **Назначение** | **Лейаут** |
| MainWindow | | | | | | | | QMainWindow | Окно программы | Вертикальный |
|  | centralwidget | | | | | | | Qwidget | Контейнер для элементов окна | Вертикальный |
|  |  | splitter | | | | | | Qsplitter | Разделяет изображение и панель с элементами управления | Горизонтальный |
|  |  |  | Image | | | | | QGraphicsView | Изображение | Нет |
|  |  |  | rightPanel | | | | | Qframe | Панель с элементами управления | Вертикальный |
|  |  |  |  | topPanel | | | | QHBoxLayout | Панель с элементами масштабирования | Горизонтальный |
|  |  |  |  |  | btnDecreaseZoom | | | QPushButton | Кнопка уменьшения масштаба | Нет |
|  |  |  |  |  | btnIncreaseZoom | | | QPushButton | Кнопка увеличения масштаба | Нет |
|  |  |  |  |  | editZoom | | | QLineEdit | Поле ручной установки масштаба | Нет |
|  |  |  |  |  | lblZoom | | | Qlabel | Подпись "Zoom" | Нет |
|  |  |  |  | Steps | | | | QStackedWidget | Элемент, содержащий "шаги" программы | Нет |
|  |  |  |  |  | Step1 | | | Qwidget | Первый шаг | Вертикальный |
|  |  |  |  |  |  | btnOpenImage | | QPushButton | Кнопка открытия изображения | Нет |
|  |  |  |  |  |  | lblTask | | Qlabel | Информация для первого шага | Нет |
|  |  |  |  |  | Step2 | | |  | Второй шаг | Вертикальный |
|  |  |  |  |  |  | gridLayout | | QGridLayout | Лейаут для элементов второго шага | Сетка |
|  |  |  |  |  |  |  | label | Qlabel | Подписи для соответствующих элементов управления | Нет |
|  |  |  |  |  |  |  | label\_2 |
|  |  |  |  |  |  |  | label\_3 |
|  |  |  |  |  |  |  | label\_4 |
|  |  |  |  |  |  |  | label\_5 |
|  |  |  |  |  |  |  | label\_6 |
|  |  |  |  |  |  |  | label\_7 |
|  |  |  |  |  |  |  | xBottomLeft | QLineEdit | Поля ввода значений долготы для соответствующих углов изображения |
|  |  |  |  |  |  |  | xBottomRight |
|  |  |  |  |  |  |  | xTopLeft |
|  |  |  |  |  |  |  | xTopRight |
|  |  |  |  |  |  |  | yBottomLeft | Поля ввода значений широты для соответствующих углов изображения |
|  |  |  |  |  |  |  | yBottomRight |
|  |  |  |  |  |  |  | yTopLeft |
|  |  |  |  |  |  |  | yTopRight |
|  |  |  |  |  |  |  | xDelimeter | Деление сетки по долготе |
|  |  |  |  |  |  |  | yDelimeter | Деление сетки по широте |
|  |  |  |  |  |  | lblDataError | | Qlabel | Элемент для вывода ошибок |
|  |  |  |  |  |  | lblTask\_2 | | Информация для второго шага |
|  |  |  |  |  | Step3 | | | Qwidget | Третий шаг | Вертикальный |
|  |  |  |  |  |  | btnDraw | | QPushButton | Кнопка рисования путей полета | Нет |
|  |  |  |  |  |  | lblTask3 | | Qlabel | Информация для третьего шага |
|  |  |  |  |  | btnNext | | | QPushButton | Кнопка перехода на следующий шаг |

## Подготовительные работы

Потребуется импорт следующих библиотек и классов Qt:

from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog, QMessageBox, QGraphicsPixmapItem, QGraphicsScene, QGraphicsRectItem, QGraphicsLineItem, QGraphicsPolygonItem

from PyQt5.QtSvg import QSvgGenerator

from PyQt5.QtGui import QPixmap, QPolygonF, QBrush, QPen, QColor, QTransform, QPainter

from PyQt5.QtCore import QPointF, QLineF, Qt

# For loading ui file

from PyQt5 import uic

import sys

Инициализация класса окна и его конструктор:

# Main window of application

class Window(QMainWindow):

    def \_\_init\_\_(self):

        # Make this app super awesome

        super().\_\_init\_\_()

        # Load UI from file

        uic.loadUi('ui/mainwindow.ui', self)

        # Initialize step 1

        self.stepOne()

Формирование окна приложения:

if \_\_name\_\_ == '\_\_main\_\_':

    # Create an instance of QApplication

    app = QApplication(sys.argv)

    # Create an instance of our Window class

    window = Window()

    # Show window

    window.show()

    # Execute app and also write action for exiting

    sys.exit(app.exec\_())

## Этап 1. Загрузка изображения.

На данном этапе происходит работа с элементами окна, прописываются слоты для сигналов кнопок:

def stepOne(self):

    """Step 1 -- loading image, connecting events to controls

    """

    self.disableItems()

    self.comboLang.setVisible(False) # Hide language panel because there are no additional languages yet

    # Connect events

    self.btnOpenImage.clicked.connect(self.loadImage)

    self.btnNext.clicked.connect(self.stepTwo)

    self.btnIncreaseZoom.clicked.connect(self.increaseZoom)

    self.btnDecreaseZoom.clicked.connect(self.decreaseZoom)

    self.editZoom.textEdited.connect(self.setZoom)

Функция загрузки изображения. Если пользователь выбрал файл, не являющийся изображением, программа выведет окно с ошибкой.

def loadImage(self):

    # Open file dialog and get path

    file = QFileDialog.getOpenFileName(self, 'Open image', '', 'Images (\*.jpg \*.jpeg \*.png \*.bmp);;All files (\*)')[0]

    # Create a pixmap from file

    img = QPixmap(file)

    # Check if it's an image

    if not img.isNull():

        # Set opened image to the scene

        item = QGraphicsPixmapItem(img)

        self.scene = QCustomScene()

        self.scene.addItem(item)

        self.Image.setScene(self.scene)

        self.height = img.height()

        self.width = img.width()

        self.enableItems()

    elif file != "":

        # If it's not image and user didn't press 'Cancel' raise an error message

        self.disableItems()

        msg = QMessageBox()

        msg.setIcon(QMessageBox.Critical)

        msg.setWindowTitle("AerialWare - Invalid image")

        msg.setText("It's not an image. Please open a valid image.")

        msg.exec\_()

## Этап 2. Ввод координат.

Данный этап нужен только для корректной обработки первого нажатия кнопки «Next». Все остальное происходит в функции для третьего этапа

def stepTwo(self):

    """Step 2 -- Needed only for first click. Step 3 does all the job.

    """

    self.turnPage()

    self.btnNext.disconnect()

    self.btnNext.clicked.connect(self.stepThree)

## Этап 3. Выбор участков и сохранение результатов.

В дальнейшем приведенный код находится в функции stepThree, пока не будет сказано обратное.

Чтение и обработка данных, также проверка на ошибки и их вывод в окне программы:

# Validate data

def setError(s):

    """Formats string for displaying

    """

    self.lblDataError.setText("<html><head/><body><div style='font-size: 12pt'>Can't procede, " + s + "</div></body></html>")

# Try to read data

try:

    xTL = float(self.xTopLeft.text())

    xTR = float(self.xTopRight.text())

    xBL = float(self.xBottomLeft.text())

    xBR = float(self.xBottomRight.text())

    yTL = float(self.yTopLeft.text())

    yTR = float(self.yTopRight.text())

    yBL = float(self.yBottomLeft.text())

    yBR = float(self.yBottomRight.text())

    xD = float(self.xDelimiter.text())

    yD = float(self.yDelimiter.text())

except:

    # Write error message

    setError("some fields contains non-numeric values. Please recheck everything and click \"Next\" again.")

    return

# Check for metric errors

errors = ""

if xD <= 0:

    errors += "longtitude"

if yD <= 0:

    if errors != "":

        errors += " and "

    errors += "latitude"

if errors != "":

    setError("delimiters should be more than zero. Please set correct " + errors + " delimiter.")

    return

# And calculate px / deg ratio, i.e. deg \* (px / deg) = px

try:

    self.xDTop = int(self.width / abs(xTL - xTR) \* xD)

    self.xDBottom = int(self.width / abs(xBL - xBR) \* xD)

    self.yDLeft = int(self.height / abs(yTL - yBL) \* yD)

    self.yDRight = int(self.height / abs(yTR - yBR) \* yD)

except ZeroDivisionError:

    setError("some given corners are at the same point. Please recheck every coordinate.")

    return

errors = ""

if self.xDTop > self.width:

    errors += "Longtitude delimiter is less than given size of top of the image. Please check longtitude of top corners.<br>"

if self.xDBottom > self.width:

    errors += "Longtitude delimiter is less thagiven n size of bottom of the image. Please check longtitude of bottom corners.<br>"

if self.yDLeft > self.height:

    errors += "Latitude delimiter is less than given size of left of the image. Please check latitude of left corners.<br>"

if self.yDRight > self.height:

    errors += "Latitude delimiter is less than diven size of right of the image. Please check latitude of right corners.<br>"

if errors != "":

    setError("something's wrong with given data:<br>" + errors)

    return

Генерация точек для создания сетки. Если не была добавлена ни одна точка, программа выдаст ошибку.

# Draw grid

def intersection(x1, y1, x2, y2, x3, y3, x4, y4):

    """Calculates intersection on two lines

    Args:

        x1, y1 -- first point of first line

        x2, y2 -- second point of first line

        x3, y3 -- first point of second line

        x4, y4 -- second point of second line

        These coordintaes must represent coordinates in pixels

    Returns:

        QPointF(x, y) -- intersection of two given lines

    """

    divider = ((x1 - x2) \* (y3 - y4) - (y1 - y2) \* (x3 - x4))

    if divider == 0:

        return None

    x = ((x1 \* y2 - y1 \* x2) \* (x3 - x4) - (x1 - x2) \* (x3 \* y4 - y3 \* x4)) / divider

    y = ((x1 \* y2 - y1 \* x2) \* (y3 - y4) - (y1 - y2) \* (x3 \* y4 - y3 \* x4)) / divider

    return QPointF(int(x), int(y))

# Set points for grid

# Points will look like:

# [point, point, ...],

# [point, point, ...], ...

point\_rows = []

# Let x1, y1; x2, y2 be vertical line

# and x3, y3; x4, y4 be horizontal line.

# Thus, y1 and y2 should be always on top and bottom;

# x3, x4 -- on left and right

y1, y2, x3, x4 = 0, self.height, 0, self.width

# So we have to change:

#   for vertical line: x1, x2

#   for horizontal line: y3, y4

y3 = y4 = 0 # Initial values for horizontal line

# Move horizontal line

while y3 <= self.height or y4 <= self.height:

    x1 = x2 = 0 # Initial values for vertical line

    # Move vertical line

    points = []

    while x1 <= self.width or x2 <= self.width:

        point = intersection(x1, y1, x2, y2, x3, y3, x4, y4)

        if point != None:

            points.append(point)

        x1 += self.xDTop

        x2 += self.xDBottom

    if points != []:

        point\_rows.append(points)

    y3 += self.yDLeft

    y4 += self.yDRight

# If nothing has been added

if points == []:

    setError("can't generate grid. I dont even know how you managed to get this error. Please recheck coordinates and try again.")

    return

Сетка будет представлена в виде полигонов, имеющих общие точки. Это необходимо для реализации событий по клику по сетке.

# Get scene geometry to preserve scene expanding

rect = self.scene.sceneRect()

# And add bounds for the grid

self.bounds = QGraphicsRectItem(rect)

self.bounds.setFlag(self.bounds.ItemClipsChildrenToShape)

self.scene.addItem(self.bounds)

# Create polygons from points

# We'll recheck previous item

i = 1 # Rows

while i < len(point\_rows):

    j = 1 # Points

    while j < len(point\_rows[i]):

        # Add points in following order: top left, top right, bottom right, bottom left

        points = [

            point\_rows[i - 1][j - 1],

            point\_rows[i - 1][j],

            point\_rows[i][j],

            point\_rows[i][j - 1]

        ]

        # We're assigning self.bounds as parent implicitly, so we shouldn't add polygon to scene by ourselves.

        poly = QCustomGraphicsPolygonItem(QPolygonF(points), self.bounds)

        poly.setRowCol(i - 1, j - 1)

        j += 1

    i += 1

# Restore scene geometry

self.scene.setSceneRect(rect)

Переход, собственно, на третий этап и назначение слотов сигналам кнопок:

self.turnPage()

self.btnDraw.clicked.connect(self.scene.drawPaths)

self.btnNext.setText("Save")

self.btnNext.disconnect()

self.btnNext.clicked.connect(self.save)

def save(self):

    """Saves user results

    """

    rect = self.scene.sceneRect()

    file = QFileDialog.getSaveFileName(self, 'Save file', '', 'Vector Image (\*.svg)')[0]

    if file == "":

        return

    gen = QSvgGenerator()

    gen.setFileName(file)

    gen.setSize(rect.size().toSize())

    gen.setViewBox(rect)

    painter = QPainter(gen)

    self.scene.render(painter)

    painter.end()

    sys.exit(None)

## Дополнительные методы класса окна

def enableItems(self):

    """Enables controls

    """

    self.editZoom.setEnabled(True)

    self.btnDecreaseZoom.setEnabled(True)

    self.btnIncreaseZoom.setEnabled(True)

    self.btnNext.setEnabled(True)

def disableItems(self):

    """Disables controls

    """

    self.editZoom.setEnabled(False)

    self.btnDecreaseZoom.setEnabled(False)

    self.btnIncreaseZoom.setEnabled(False)

    self.btnNext.setEnabled(False)

def changeLanguage(self):

    """Changes language. Not implemented yet.

    """

    pass

def turnPage(self):

    """Turns page of "Steps"

    """

    self.Steps.setCurrentIndex(self.Steps.currentIndex() + 1)

def increaseZoom(self):

    """Zooms image in

    """

    value = float(self.editZoom.text()) + 10

    self.editZoom.setText(str(value))

    self.setZoom()

def decreaseZoom(self):

    """Zooms image out

    """

    value = float(self.editZoom.text()) - 10

    if value >= 10:

        self.editZoom.setText(str(value))

        self.setZoom()

def setZoom(self):

    """Sets zoom

    """

    try:

        value = float(self.editZoom.text()) \* 0.01

    except ValueError:

        self.editZoom.setText("100.0")

        self.setZoom()

        return

    self.Image.resetTransform()

    self.Image.scale(value, value)

## Подклассы элементов Qt

### Класс ячейки сетки

Ячейка сетки представляет собой полигон, который может считывать клики по нему.

При инициализации класса задается обводка полигона, цвета его заливки в выделенном и обычном состоянии и вычисляются центры его сторон. Так как все полигоны данного класса будут иметь только четыре стороны, можно не писать обработку исключений.

class QCustomGraphicsPolygonItem(QGraphicsPolygonItem):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super().\_\_init\_\_(\*args, \*\*kwargs)

        # Set appearance

        pen = QPen(QColor(30, 30, 255))

        pen.setWidth(2)

        self.setPen(pen)

        self.checkBrush = QBrush(QColor(130, 130, 255, 100))

        self.uncheckBrush = QBrush(QColor(0, 0, 0, 0))

        # Calculate centers of polygon's sides

        def getSide(p1, p2):

            x1, x2, y1, y2 = p1.x(), p2.x(), p1.y(), p2.y()

            return QPointF(

                min(x1, x2) + abs((x1 - x2) / 2),

                min(y1, y2) + abs((y1 - y2) / 2)

            )

        points = self.polygon()

        # Check Step 3 to understand magic numbers below

        self.left = getSide(points[0], points[3])

        self.right = getSide(points[1], points[2])

        self.top = getSide(points[0], points[1])

        self.bottom = getSide(points[3], points[2])

Дополнительные методы данного класса, используемые другими частями программы:

def getSides(self):

    """Returns dictionary of sides

    """

    return {

        "left": self.left,

        "right": self.right,

        "top": self.top,

        "bottom": self.bottom

    }

def setRowCol(self, row, col):

    """Sets row and column of square in a grid

    """

    self.row = row

    self.col = col

def getRowCol(self):

    """Returns dict with row and column

    """

    return {

        "row": self.row,

        "col": self.col

    }

### Класс сцены

Сцена представляет собой элемент, в который помещаются все элементы формируемого изображения. Подкласс требуется для реализации работы с сеткой. Данный класс также реализует свое выделение элементов.

Конструктор сцены вызывает конструктор родителя и создает список с выделенными элементами:

class QCustomScene(QGraphicsScene):

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super().\_\_init\_\_(\*args, \*\*kwargs)

        self.customSelectedItems = []

Обработка клика по сцене заключается в выделении и снятии выделения с ячейки сетки:

def mousePressEvent(self, event):

    """Checks and unchecks squares under cursor if RMB has been pressed

    """

    if event.button() != Qt.RightButton:

        return

    item = self.itemAt(event.scenePos(), QTransform())

    if not isinstance(item, QCustomGraphicsPolygonItem):

        return

    if item in self.customSelectedItems:

        index = self.customSelectedItems.index(item)

        self.customSelectedItems.pop(index)

        item.setBrush(item.uncheckBrush)

    else:

        self.customSelectedItems.append(item)

        item.setBrush(item.checkBrush)

Функция, возвращающая выделенные элементы (только ячейки):

def selectedItems(self):

    return self.customSelectedItems

Функции, используемые для отрисовки путей. Отрисовка работает только с выделенными ячейками и заключается в следующих этапах:

1. Сортировка ячеек по столбцам и строкам или наоборот в зависимости от того, прокладывается путь по меридианам или горизонталям (в дальнейшем «линия»).
2. Для каждого полигона:
   1. Если данный полигон первый в линии:
      1. Создание переменной с координатами центра стороны первого полигона в линии, ближайшей к ее началу (в дальнейшем «начало строки»).
      2. Удаление обозначения начала новой линии.
   2. Вычисление, является ли данный полигон последним в линии.
   3. Если данный полигон последний в линии:
      1. рисование линии от начала строки до координат стороны текущего полигона, ближайшей к ее концу.
      2. Если ранее были отрисованы линии, рисование соединительной линии от конца предыдущей к началу текущей.
      3. Перенос начала следующей линии к концу текущей.
      4. Запись текущей линии как предыдущей.
      5. Обозначение начала следующей линии

def drawPaths(self):

    """Draws both flight path

    """

    # Remove existing lines

    for item in self.items():

        if isinstance(item, QGraphicsLineItem):

            self.removeItem(item)

    # Draw paths

    items = self.selectedItems()

    self.drawPath(items, True)

    self.drawPath(items, False)

    self.update()

def drawPath(self, items, use\_rows):

    """Draws one flight path

    Args

    items -- squares.

    use\_rows:

        True -- go by rows

        False -- go by cols

    """

    if use\_rows:

        way = "row"

        sort\_second = "col"

        old\_side = "left"

        new\_side = "right"

        color = QColor(255, 10, 10)

    else:

        way = "col"

        sort\_second = "row"

        old\_side = "top"

        new\_side = "bottom"

        color = QColor(10, 255, 10)

    # Pen for main lines

    pen = QPen(color)

    pen.setWidth(2)

    # Pen for turning lines

    pen2 = QPen(color)

    pen2.setWidth(2)

    pen2.setStyle(Qt.DashLine)

    pen2.setDashOffset(2)

    # Sort selected items

    items.sort(key=lambda item: item.getRowCol()[sort\_second])

    items.sort(key=lambda item: item.getRowCol()[way])

    # Draw lines

    new\_pos = True # Shows if we're will be at new col/row at next iteration

    prev\_line = None # Previously drawn line

    is\_last\_point\_reversed = False # Shows from which end draw turning line

    for i in range(len(items)):

        item = items[i]

        if new\_pos:

            side1 = item.getSides()[old\_side]

            new\_pos = False

        pos = item.getRowCol()[way]

        try:

            next\_pos = items[i + 1].getRowCol()[way]

        except:

            next\_pos = pos + 1

        if pos != next\_pos:

            side2 = item.getSides()[new\_side]

            lineF = QLineF(side1, side2)

            line = QGraphicsLineItem(lineF)

            line.setPen(pen)

            self.addItem(line)

            new\_pos = True

            if prev\_line != None:

                if is\_last\_point\_reversed:

                    start = prev\_line.p2()

                    end = side2

                else:

                    start = prev\_line.p1()

                    end = side1

                new\_line = QGraphicsLineItem(QLineF(start, end))

                new\_line.setPen(pen2)

                self.addItem(new\_line)

            is\_last\_point\_reversed = not is\_last\_point\_reversed

            prev\_line = lineF