**Пояснительная записка**

**К проекту**

Тема проекта: основы промышленного программирования на библиотеке PyQt5

Проект выполнил и защитил: Сулимов Сергей

2023/2024 учебный год

2

**Содержание**

Введение…………………………………………………………………3

1. Создания функционала………………………………………………4

1.1 Зависимости…………………………………………………………4

1.2 Основной функционала для подсчёта и отрисовки положения планет…………………………………………………………………….5

1.3 Функционал записи и чтения баз данных или файлов……………8

2. Проектирования классов и константы………………….………….10

2.1 Классы объекты…………………………………………………….10

2.2 Исключения………………………………………………………....11

3. Интерфейс……………………………………………………………12

3.1 Код с помощью qtdesigner…………………………………………12

3.2 Реализация функционала с помощью PyQt5……………………..13

3.3 Запись в базу данных новых планет………………………………14

3.4 Окно карты планет…………………………………………………17

3.5 Обработчик исключений…………………………………………..19

4. Итог…………………………………………………………………..21

3

**Введение**

PyQt -набор расширений(биндингов) графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python.

За первую часть курса “Основы промышленного программирования” я изучил модуль PyQt5, основы языка программирования SQL, научился работать с таблицами csv и работать с СУБД sqlite с помощью модуля sqllite3, научился обрабатывать исключения и создавать собственные.

Целью проекта является показать насколько хорошо я усвоил новые технологии указанные выше. Побочной целью проекта была упрощение игры в **Kerbal Space Program**.

**Kerbal Space Program** (с англ. — «Космическая программа Кербалов»; сокр. **KSP**) — компьютерная игра в жанре симулятор, разработанная и изданная компанией Squad. Игра относится к жанру подлинных космических симуляторов.

Для этого были реализованные следующие функции:

1. Расчёт положения планет для наиболее эффективного перелёта между планетами.
2. Возможность добавлять собственные планеты в систему.
3. Удобный просмотр всех планет, добавленных ранее.

4

**Создания функционала**

Зависимости

Пару слов про зависимости. Все сторонние модули указаны в файле requirements.txt

1. PyQt5==5.15.10
2. Pillow==10.1.0
3. pyqtdarktheme==2.1.0

Всего их три. PyQt5 – основной модуль для работы с интерфейсом.

Pillow – модуль для отрисовки положения планет. Я использовал его, а не встроенную функцию PyQt5 так как должен был отрисовать эти планеты в определённом месте на форме, соответственно точка отсчёта координат была не 0, 0. Но благодаря pillow я могу нарисовать изображение и вставить в виджет Qlabel.

pyqtdarktheme - модуль быстрого создания тёмной и светлой темы приложения.

По мимо сторонних модулей также использовал встроенные модули.

Sys был нужен для корректной работы PyQt5 и обработчика исключений. Traceback нужен был для обработчика исключений. Random для отрисовки карты планет. Math для вычисления положений планет и координат эллипса, который нужно было нарисовать. Os для создания папки crusheslogs, в которую записываются все необработанные исключения. Datetime для названия ошибок. Sqlite3 для чтения и записи в базу данных информации.

Поэтому для работы программы нужно установить все сторонние библиотеки с помощью команды.

>pip install -r requirements.txt

5

Основной функционала для подсчёта и отрисовки положения планет

Для начала нужно бы разобраться как-же всё же рассчитать правильное положения планет. К сожалению для этого школьных знаний недостаточно. Чтобы не тратить 4 – 6 лет для обучения астрономии, воспользуюсь готовой формулой.

На данном [сайте](https://spacedock.ru/kerbal-space-program/guides-ksp/4776-teoriya-i-praktika-mezhplanetnyh-pereletov-chast-1.html) достаточно понятным языком описывается вся теории межпланетных перелётов. Прочитав всю теории можно преобразовать всё в две строчки кода

1. t\_h = math.pi \* math.sqrt(math.pow(first.alt + second.alt, 3) / (8 \* parent.second\_space\_speed))
2. angle = (180 - math.sqrt(parent.second\_space\_speed / second.alt) \* (t\_h / second.alt) \* (180 / math.pi)) % 360

где first - первая планета, second – вторая, parent – общий родитель на орбите которого крутится два тела. Итак, весь функционал записан в файл kspPlanetsTransphere.py. Функция create\_angle(first, second), возвращает приблизительный угол.

1. planets = planet\_classes() # список планет

Первая строчка возвращает список классов всех планет, каждый класс имеет поля для вычисление различных данных. К данной функции вернёмся чуть позже. Далее нужно определить, какие именно перед нами планеты, ведь на вход функции передаются строчки.

1. for el in planets:
2. if el.name == first:
3. first = el
4. elif el.name == second:
5. second = el
6. for el in planets:
7. if el.id == first.parent:
8. parent = el

Второй цикл нацелен на то, чтобы найти родителя планет. Переменная parent это такой-же класс, как и first или second. Осталось подставить значения и получить результат, который вернётся в виде str.

6

Следующая функция draw\_angle(first, second, ...). Как следуют из названия, она рисует положение планет. Функция имеет два позиционных аргумента first и second – это названия планет, и несколько именованных. Width и height – размеры картинки (по умолчанию 1000x1000). Color и color\_text цвета для отрисовки орбит и текста.

В самом начале программы проверка на правильность данных.

1. if first == second:
2. raise SamePlanet('Укажите разные планеты')

Далее вызывается ранее описанная функция create\_angle, чтобы определить угол. Создаётся изображение, где будем рисовать, draw – чем будем рисовать и координаты x, y.

1. angle = float(create\_angle(first, second))
2. im = Image.new("RGB", (width, height), color)
3. draw = ImageDraw.Draw(im)
4. x = 0
5. y = 0

После также строчки заменяются классами Planet. И следует вспомогательная переменная, которая хранит в себе или true или false. Она нужна для того, чтобы правильно отрисовывать планеты. Например, если мы захотим полететь с земли на марс, то программа нарисует в центре солнце, затем первую орбиту земли, а затем вторую орбиту марса. Если же с марса на землю, то нужно сделать так чтобы марс также был на второй орбите, а земля на первой.

1. flag = second.alt < first.alt

**Ещё одна проверка на правильность ввода.**

1. if first.parent != second.parent:
2. raise DifferentParent('укажите планеты с одинаковым "родителем"')

**Далее следует рисование планет. Так как если мы вначале нарисуем в центре объект, а потом захотим нарисовать белый круг с чёрным контуром, этот круг закрасит объект в центре. Поэтому рисовать будем в обратном порядке. Вначале орбиту верхнего объекта.**

**7**

1. draw.ellipse(((int(0.2 \* width), int(0.2 \* height)), (int(0.8 \* width), int(0.8 \* height))),  
    fill=color, outline=color\_text)

**Далее в зависимости от того, какой объект выше. Если second ниже, то он будет отрисован позже, иначе его координаты посчитаются с помощью** [формулы,](https://bez-smenki.ru/material/koordinaty-tochki-na-okruzhnosti/) **которая немного подстроена под специфику координат в pillow. Также рисуется текст – название планеты.**

1. if flag:
2. draw.ellipse(((int(0.77 \* width), int(0.47 \* height)), (int(0.83 \* width), int(0.53 \*
3. height))), fill=first.color)
4. draw.text((int(0.77 \* width), int(0.53 \* height)), first.name, fill=color\_text,
5. font\_size=20)
6. else:
7. R = (int(0.2 \* height) - int(0.8 \* height)) / 2
8. x0 = int(0.5 \* width)
9. y0 = int(0.5 \* height)
10. x = int(x0 + R \* -math.cos(math.radians(angle)))
11. y = int(y0 + R \* math.sin(math.radians(angle)))
12. Draw.ellipse([x - 30, y - 30, x + 30, y + 30], fill=second.color)
13. draw.text((x - 30, y + 30), second.name, fill=color\_text, font\_size=20)

**Далее следует аналогичный код только теперь для нижнего объекта. И kerbol – аналог солнца.**

1. draw.ellipse(((int(0.45 \* width), int(0.45 \* height)), (int(0.55 \* width), int(0.55 \*
2. height))), (238, 210, 2))
3. draw.text((int(0.475 \* width), int(0.55 \* height)), 'kerbol', fill=color\_text, font\_size=20)

**И линии соединяющие kerbol с двумя объектами.**

1. if not flag:
2. draw.line(((int(0.5 \* width), int(0.5 \* height)), (int(0.7 \* width), int(0.5 \* height))),
3. fill='red', width=2)
4. else:
5. draw.line(((int(0.5 \* width), int(0.5 \* height)), (int(0.8 \* width), int(0.5 \* height))),
6. fill='red', width=2)
7. draw.line(((int(0.5 \* width), int(0.5 \* height)), (x, y)), fill='red', width=2)

**Далее мне нужно было конвертировать класс Image из модуля Pillow в QPixmap из PyQt5. Для этого вначале конвертировал изображение в массив байтов, и затем превратил его QPixmap, вернув результат.**

1. data = im.tobytes("raw", "RGB")
2. qim = QtGui.QImage(data, im.size[0], im.size[1], QtGui.QImage.Format\_RGB888)
3. return QtGui.QPixmap.fromImage(qim)

**8**

Функционал записи и чтения баз данных или файлов

**Следующий файл, который мы рассмотрим – WriteAndReadFilesFunctions.py. Название говорит само за себя, в нём описан функционал записи и чтения.**

**Первая функция** write\_exception(message). Она нужна **для записи необработанных исключений в txt файлы в папке crusheslogs.**

**Для начала нужно определить время, чтобы так назвать файл.**

1. now = datetime.datetime.now()
2. path = f"crusheslogs\\{now.strftime('%c').replace(':', '.').replace(' ', '\_')}.txt"

**Затем определяется есть-ли папка crusheslogs в папке с программой, если есть, то просто записывается файл, если нет, то создаётся папка и точно также записывается message в файл path.**

1. if 'crusheslogs' in os.listdir():
2. with open(path, 'w', encoding='utf-8') as f:
3. f.write(message)
4. else:
5. os.mkdir('crusheslogs')
6. with open(path, 'w', encoding='utf-8') as f:
7. f.write(message)

**Следующая функция** planet\_classes()**, которая использовалась раньше.**

**Вначале устанавливается соединение с базой данных, создаётся cursor и отправляется запрос на получение всех данных из таблицы.**

1. con = sqlite3.connect(DATABASE + 'planets.db')
2. cur = con.cursor()
3. data = cur.execute("""select \* from planets""").fetchall()

**Теперь нужно пройтись по всем элементам data и создать класс Planet, добавив его в result, вернуть результат в виде списка экземпляров класса.**

1. result = []
2. for el in data:
3. result.append(Planet(\*el))
4. con.close()
5. return result

**Следующая функция вспомогательная. Я пометил её двумя нижними подчёркиваниями, что говорит о ей приватности. Это функция, нужна для формирования правильного запроса.**

**9**

1. def \_\_convert(obj):
2. if isinstance(obj, bool):
3. if obj:
4. return str(1)
5. return str(0)
6. return f"'{obj}'"

**Последняя функция в данном файле -** add\_obj\_in\_database(path, data, data\_name)**. Как и в функции** planet\_classes()**, создаётся подключение и cursor. Теперь время формировать запрос. Вначале вычленяем из path название таблицы в переменную database. Первая строчка запроса:**

1. **INSERT** into planets

**Затем указываем поля, которые нужно заполнить. Они хранятся в data\_name. Теперь сами данные мы конвертируем с помощью** \_\_convert(obj), все строчи получают кавычки, а логические значения превращаются в 0 или 1. Например, на вход подаётся строчка *“Земля”*, на выходе мы получаем *“’Земля’”*. В итоге мы получаем запрос вида:

1. **INSERT** into planets
2. **(названия столбцов)**
3. **values (данные)**

**В конце запрос отправляется, база данных сохраняется и соединение закрывается.**

**10**

****Проектирование классов****

Классы объекты

**Изначально была задумка добавить несколько классов для объектов, например, двигателей или топливных баков для создание ракетоносителей, но из-за недостатка времени было принято отказаться от этой идей и оставить только класс планет –** Planet**.**

**Данный класс имеет всего два метода.** \_\_init\_\_(self, id, name, g, atmosphere, second\_space\_speed, color, parent, alt), где происходит инкапсуляция, и два одинаково работающих методов \_\_str\_\_(self) и \_\_repr\_\_(self), которые возвращают поле name, где хранится имя планеты.

1. class Planet:
2. def \_\_init\_\_(self, id, name, g, atmosphere, second\_space\_speed, color, parent, alt):
3. self.g = g
4. self.atmosphere = atmosphere
5. self.name = name
6. self.second\_space\_speed = second\_space\_speed
7. self.color = color
8. self.parent = parent
9. self.alt = alt
10. self.id = id
11. def \_\_str\_\_(self):
12. return self.name
13. def \_\_repr\_\_(self):
14. return self.name

Стоит также рассказать о маленьком файле Constants, в котором хранятся переменные, которые используются по всей программе.

1. DATABASE = "datebases\\"
2. IMAGES = 'images\\'
3. TTF = 'TTF\\'
4. OK\_RESULT = 1
5. BAD\_RESULT = 0

**11**

****Исключения****

**Ещё одна новая технология, которую я изучил за последние уроки. Базовое исключение ExceptionGroupKsp, которое наследует Exception. Далее идёт разветвление на ErrorPlenets и IncorrectInput. И далее ещё 4 исключения, ответственные за разные ошибки.**

1. class ExceptionGroupKSP(Exception):
2. def \_\_init\_\_(self, message='Необработанная ошибка'):
3. self.message = message
4. def \_\_str\_\_(self):
5. return f'{self.message}'
6. class ErrorPlanets(ExceptionGroupKSP):
7. pass
8. class NegativeValue(ErrorPlanets):
9. pass
10. class IncorrectInput(ExceptionGroupKSP):
11. pass
12. class NoAnyoneSelect(IncorrectInput):
13. pass
14. class DifferentParent(IncorrectInput):
15. pass
16. class SamePlanet(IncorrectInput):
17. pass

**12**

****Интерфейсы****

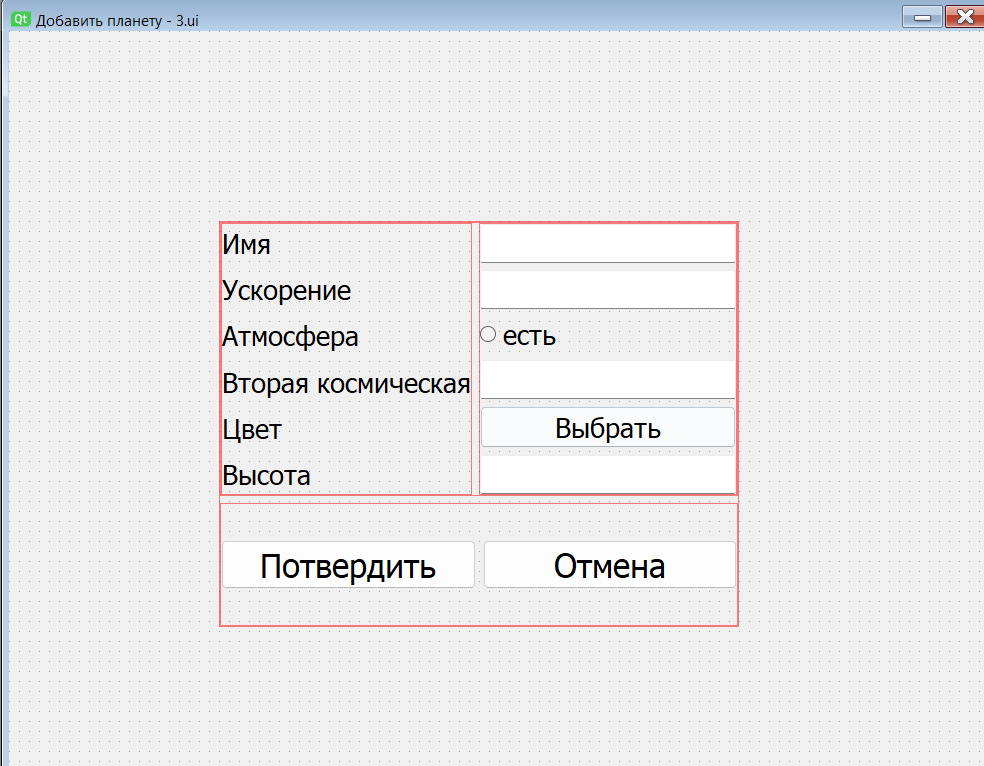
Код с помощью qtdesigner

**Всего создана три окна с помощью qtdesigner.**

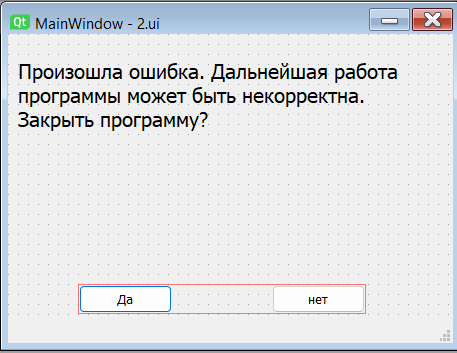
**Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описаниерис. 1 – главное окно**

**13**

**рис. 2 – окно добавление планет**

**рис. 3 – окно ошибки**



На данных окнах использовались виджиты: QPushButton, Qlabel, QComboBox, QLineEdit, QRadioButton.

А также диалоги: QDialog и QColorDialog.

14

Реализация функционала с помощью PyQt5

Класс, который реализовывает главное окно, называется CalculatorKsp. В инициализации запускается функция setupUi(self), которая создаёт все виджеты, функция root(self), которая создаёт все нужные поля, вешает на кнопки функции и запускает функцию fill\_combobox(self), она заполняет combobox`ы, читая из базы данных все планеты, кроме Кербола.

1. def fill\_combobox(self)
2. planets = kspPlanetsTransphere.planet\_classes()
3. planets = list(map(str, planets))[1:]
4. self.start.addItems(planets)
5. self.End.addItems(planets)

Из-за того, QComboBox не имеет метода text(), как у QLineEdit, пришлось на событие currentTextChanged навесить две функции change\_start и change\_end, они сохраняют в соответствующие поля текущий текст QComboBox.

1. def change\_start(self, text):
2. self.start\_text = text
3. def change\_end(self, text):
4. self.end\_text = text

На кнопку Calculate повесил функцию calculate(self, flag=False). Для начало стоит упомянуть, что у приложения есть тёмная тема. И чтобы изображение не сливалось цвет определяется внутри функции и передаётся как именованным аргументом в draw\_angle. Тут возникает следующая проблема. При переключении темы нужно перерисовывать изображение, но поле start\_text и end\_text меняется даже без нажатия на кнопку, т.е если я поменяю планету в Combobox, затем поменяю тему, то прорисуется не старая конфигурация планет, а новая.

Для исправления данной проблемы было добавлены поля first и second, которые хранят в себе планеты, которые были отрисованы последний раз. Также был добавлен именованный аргумент flag, если он True, то планеты, которые будут отрисованы, будут взяты из полей first и second, иначе из текущий значений QComboBox. И в конце функции поля first и second заполняются отрисованными планетами. По мимо всего выше перечисленного, при нажатии на Enter рисуются планеты, путём вызова calculate.

15

1. def calculate(self, flag=False):
2. first = self.start\_text
3. second = self.end\_text
4. if flag:
5. first = self.first
6. second = self.second
7. valid(first)
8. valid(second)
9. color = (248, 249, 250)
10. color\_text = (0, 0, 0)
11. if self.dark:
12. color = (32, 33, 36)
13. color\_text = (255, 255, 255)
14. self.Angel.setText(
15. str(round(float(kspPlanetsTransphere.create\_angle(first, second)), 1)) + '°')
16. self.Image.setPixmap(
17. QPixmap(kspPlanetsTransphere.draw\_angle(first, second, width=self.Image.width(),
18. height=self.Image.height(), color=color,
19. color\_text=color\_text)))
20. self.first = first
21. self.second = second

И при смене темы в функции theme\_change меняется иконка кнопки, и вызывается функция calculate с flag=True.

1. def theme\_change(self):
2. if self.dark:
3. qdarktheme.setup\_theme('light')
4. self.darkTheme.setIcon(QIcon(QPixmap(IMAGES + 'icon-light.png')))
5. else:
6. qdarktheme.setup\_theme('dark')
7. self.darkTheme.setIcon(QIcon(QPixmap(IMAGES + 'icon-dark.png')))
8. self.dark = not self.dark
9. try:
10. self.calculate(flag=True)
11. except ExceptionGroupKSP:
12. pass

16

Запись в базу данных новых планет

Класс DialogAddPlanet отвечает за окно создания планеты. В файле main описано три метода: \_\_init\_\_(self, window=None), save\_func(self) и choice\_color(self), window переменная, в которой хранится главное окно. В инициализации вешаются функции на кнопки и создаются нужные поля. В save\_func(self), проверяется правильность данных с помощью valid(i), данные сохраняются в поле param и запускается accept(). Последний метод, отвечает за вызов диалогового окна выбора цвета.

1. class DialogAddPlanet(QDialog, Ui\_Form):
2. def \_\_init\_\_(self, window=None):
3. super().\_\_init\_\_(window)
4. self.setupUi(self)
5. self.saveBut.clicked.connect(self.save\_func)
6. self.cancel.clicked.connect(self.reject)
7. self.color = ''
8. self.color\_button.clicked.connect(self.choice\_color)
9. self.Icon.setPixmap(QPixmap(IMAGES + 'KspIcon.png'))
10. def save\_func(self):
11. param = (self.name.tex(), self.acceleration.text(), self.atmoph.isChecked(),
12. self.second\_space\_speed.text(),
13. self.color, self.alt.text())
14. for i in param:
15. valid(i)
16. self.param = param
17. self.accept()
18. def choice\_color(self):
19. color = QColorDialog.getColor()
20. if color.isValid():
21. self.color\_button.setStyleSheet(f'background-color: {color.value()}')
22. self.color = color.rgb()

17

*Окно карты планет*

Теперь поговорим об карте планет. Одной из задач являлся удобный просмотр всех планет. Я решил реализовать это виде карты системы планет на плоскости. За это окно отвечает класс DialogMapPlanets. При запуске этого окна рисуется карта. Функция draw(self, qp: Qpainter) показалась мне самой тяжёлой с точки зрении логики. Я попытаюсь объяснить словами без ссылок на код. Вначале нужно понять какое расстояние между планетами, также стоит учесть, что планеты не располагаются от края изображения до его центра. К счастью, в отличии от pil, чтобы нарисовать ellipse, указывать левый и правый угол не обязательно, можно указать радиус. Радиус будет хранится в переменной b. Предположим, что наши планеты будут лежать в промежутке по высоте между 900 и 500 пикселей(если изображение 1000 на 1000). Тогда мы должны уместить все планеты в промежутке по высоте в 400 пикселей. Для этого я 400 делю на количество планет это переменная i. Начинаем наш цикл, по выше указанных причинам он будет идти в обратном порядке (От верхнего до нижнего). Первая планета будет располагаться на 900 пикселе, т.е радиус будет равен 900 / 2. Затем я должен уменьшить радиус на i, которую мы посчитали выше. Таким образом если всё дописать, то можно увидеть такую картину.

рис. 4 – вот такую картину

18

Осталось расставить планеты и написать текст. Он будет расставляться аналогично pil, только рандомно. Надеюсь, после объяснения стало понятно. Весь код рисование карты планет.

1. def draw(self, qp: QPainter):
2. height = 1000
3. width = 1000
4. color = (248, 249, 250)
5. color\_text = (0, 0, 0)
6. if self.window\_main.dark:
7. color = (32, 33, 36)
8. color\_text = (255, 255, 255)
9. planets = WriteAndReadFilesFunctions.planet\_classes()[1:]
10. planets1 = []
11. for planet in planets:
12. if planet.parent == 0:
13. planets1.append(planet)
14. planets = planets1[:]
15. planets = sorted(planets, key=lambda x: x.alt)
16. qp.setBrush(QColor(\*color))
17. pen\_black = QPen()
18. pen\_black.setColor(QColor(\*color\_text))
19. qp.setPen(pen\_black)
20. pen\_light = QPen()
21. pen\_light.setBrush(QColor(\*color))
22. i = 0.4 \* width / len(planets)
23. b = 0.9 \* width / 2
24. center = QPoint(500, 500)
25. for j in range(len(planets), 0, -1):
26. qp.setPen(pen\_black)
27. qp.setBrush(QColor(\*color))
28. # орбита
29. qp.drawEllipse(center, b, b)
30. # планета
31. angle = randint(0, 360)
32. x0 = int(0.5 \* width)
33. y0 = int(0.5 \* height)
34. x = int(x0 + b \* -math.cos(math.radians(angle)))
35. y = int(y0 + b \* math.sin(math.radians(angle)))
36. qp.setPen(pen\_light)
37. qp.setBrush(QColor(planets[j - 1].color))
38. qp.drawEllipse(QPoint(x, y), 25, 25)
39. b -= i
40. # текст
41. qp.setPen(pen\_black)
42. qp.drawText(QPoint(x - 15, y + 35), planets[j - 1].name)
43. # Kerbol
44. qp.setPen(pen\_light)
45. qp.setBrush(QColor('yellow'))
46. qp.drawEllipse(center, 50, 50)
47. qp.setPen(pen\_black)
48. qp.drawText(QPoint(500 - 15, 500 + 65), 'Kerbol')

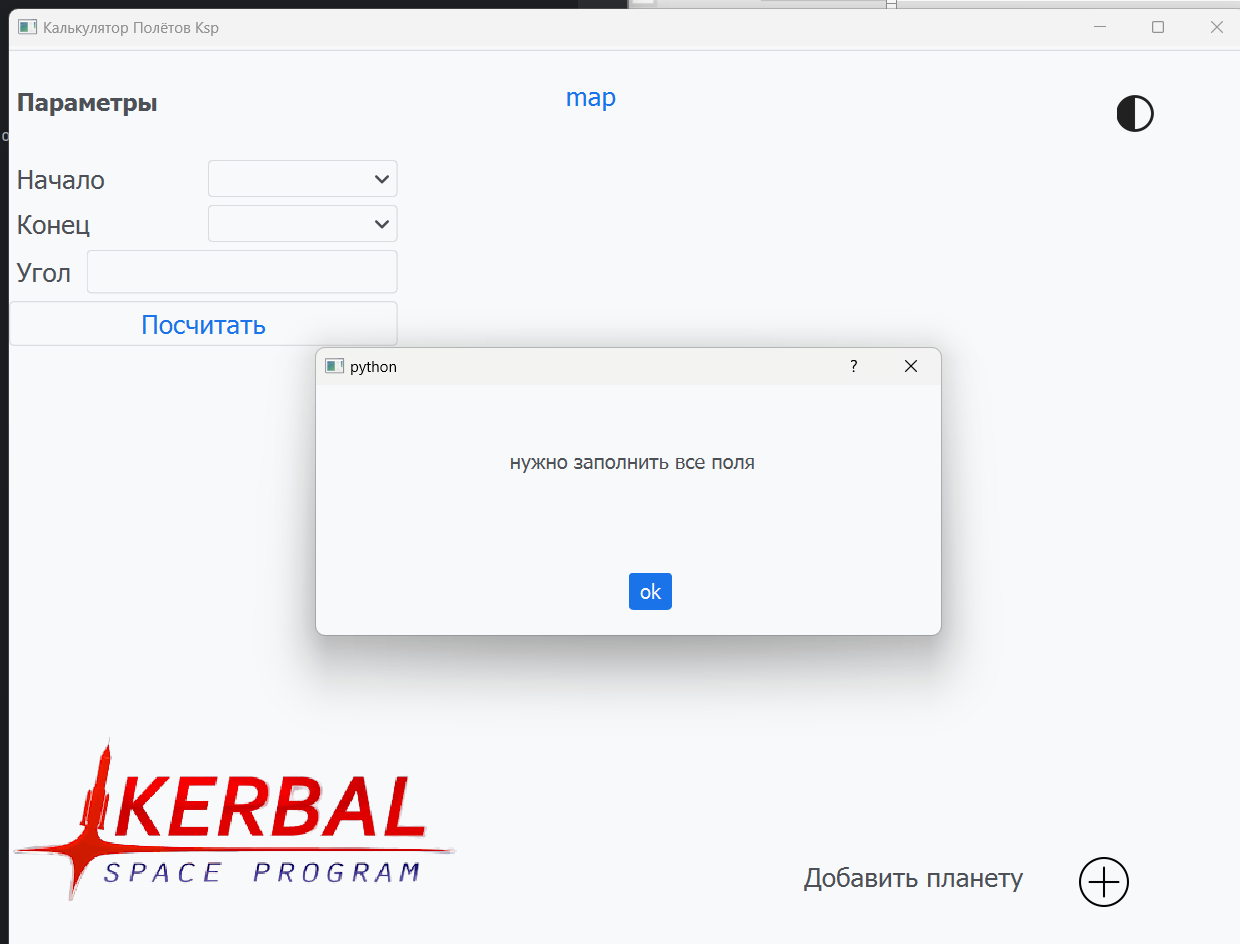
19

Обработчик исключений

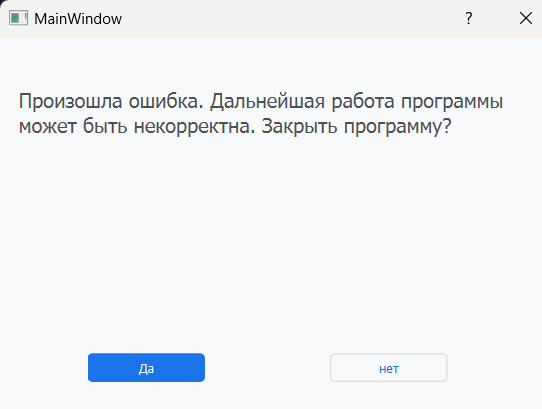
Отдельной темой стал обработчик исключений. В моей программе могут произойти два типа исключений – обработанный и необработанные. Обработанными исключениями я называю те исключения, которые предвидены, например, неправильный ввод. И необработанные ошибки, которые я не предвидел. Все ошибки, которые я предвидел, являются child класса **ExceptionGroupKsp. Sys.excepthook это поле хранит все функцию, которая будет вызываться при исключении.** Except\_hook(exc\_type, exc\_value, exc\_tb), моя функция, которая проверяет, что это за исключение и вызывает нужное окно.

1. def except\_hook(exc\_type, exc\_value, exc\_tb):
2. if not issubclass(exc\_type, ExceptionGroupKSP):
3. tb = "".join(traceback.format\_exception(exc\_type, exc\_value, exc\_tb))
4. WriteAndReadFilesFunctions.write\_exception(tb)
5. res = ex.error\_message(flag=True)
6. if res == OK\_RESULT:
7. sys.exit()
8. else:
9. ex.error\_message(exc\_value.message)

Если ошибка обработана, то создаётся экземпляр ErrorMessage, иначе ErrorCriticalDialog.

рис. 5. – пример обработанной ошибки

20

рис. 6. – пример необработанной ошибки

При нажатии на да программа закроется, иначе не закроется.

21

Итог

В заключении, хотел бы подчеркнуть все используемые технологии.

1. В первую очередь это виджеты PyQt5
2. QPixmap, QPainter (стр.17)
3. Запись ошибок в файл txt (стр. 8)
4. Базы данных (стр. 8)
5. Исключения (стр. 19)
6. Диалоги (стр. 16)
7. Обработка клавиш (стр. 14)

Скачать готовый проект можно по [ссылке](https://github.com/sergnator/kspCalculator).