

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| **Институт кибербезопасности и цифровых технологий**  Кафедра КБ-14 «Цифровые технологии обработки данных»  Специальность (направление): 09.03.02 «Информационные системы и технологии»  Дисциплина «Интерпретируемый язык программирования высокого уровня» |

Отчет

по проекту

«Файловый менеджер»

Выполнили:

Студенты 2 курса Группа БСБО-04-20 Нагаев А. С.

Панин А. А.

Москва, 2022 г.

Оглавление

[1. Задание 3](#_Toc103853724)

[2. Код программы 4](#_Toc103853725)

[3. Пояснительная записка 13](#_Toc103853726)

# Задание

Целью данного проекта было выбрано создание собственного файлового менеджера на языке Python с применением технологий PyQT. Файловый менеджер – это компьютерная программа, предоставляющая интерфейс пользователя для работы с файловой системой и файлами. Результатом работы стала самостоятельная программа, в которой были реализованы такие функции, как: создание, изменение, копирование и удаление файлов; поиск файлов; архивирование/распаковка файлов; просмотр свойств файлов и некоторые другие.

# Код программы

Код программы реализован в двух файлах.

Код файла main.py (программа):

import sys  
import os  
import shutil  
import re  
import datetime  
import time  
  
from pathlib import Path  
  
from PyQt5 import QtCore, QtWidgets, QtGui  
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox, \  
 QInputDialog, QTreeView, QWidget, QVBoxLayout, \  
 QLabel, QLineEdit, QHBoxLayout, QListWidget  
from PyQt5.QtCore import QDir, Qt, QThread, pyqtSignal  
  
from ui import main  
  
  
class SearchResults(QWidget):  
 clicked = pyqtSignal(str)  
  
 def \_\_init\_\_(self, filename, rootpath):  
 super().\_\_init\_\_()  
  
 self.list\_view = QListWidget()  
  
 self.list\_view.itemClicked.connect(self.selected)  
 self.setMinimumWidth(1000)  
 vbox = QVBoxLayout()  
 vbox.addWidget(self.list\_view)  
 self.setLayout(vbox)  
  
 self.search\_worker = Searcher(filename, rootpath)  
 self.search\_worker.found.connect(self.add)  
 self.search\_worker.finished.connect(self.finished)  
 self.search\_worker.start()  
  
 def add(self, item):  
 self.list\_view.addItem(item)  
  
 def selected(self, item):  
 self.clicked.emit(item.text())  
  
 def finished(self):  
 info = QMessageBox(self)  
 info.setIcon(QMessageBox.Information)  
 info.setWindowTitle("Error")  
 info.setText("Search ended")  
 info.show()  
  
 def closeEvent(self, a0: QtGui.QCloseEvent) -> None:  
 self.search\_worker.terminate()  
  
  
class AttributeWindow(QWidget):  
 def \_\_init\_\_(self, file, filesize):  
 super().\_\_init\_\_()  
 self.filename = file.name  
 self.filepath = file  
 self.filesize = filesize  
 self.init\_vars()  
 self.init\_ui()  
  
 def init\_vars(self):  
 stats = self.filepath.stat()  
 self.modification\_date = stats.st\_mtime  
 self.access\_date = stats.st\_atime  
  
 def init\_ui(self):  
 self.layout = QVBoxLayout(self)  
 self.setWindowTitle(self.filename)  
 filesize\_label = QLabel("Size:")  
 filesize\_real = format(self.filesize / 1024, '.2f') + " KB"  
 if self.filesize > 1048576:  
 filesize\_real = format(self.filesize / 1048576, '.2f') + " MB"  
 if self.filesize < 1024:  
 filesize\_real = str(self.filesize) + " B"  
 filesize\_value = QLineEdit(filesize\_real)  
 filesize\_value.setEnabled(False)  
 first\_row = QHBoxLayout()  
 first\_row.addWidget(filesize\_label)  
 first\_row.addWidget(filesize\_value)  
 self.layout.addLayout(first\_row)  
  
 modification\_date\_string = datetime.datetime. \  
 fromtimestamp(self.modification\_date).strftime("%Y-%m-%d %H:%M:%S")  
 access\_date\_string = datetime.datetime. \  
 fromtimestamp(self.access\_date).strftime("%Y-%m-%d %H:%M:%S")  
  
 modification\_date\_label = QLabel("Last Modified:")  
 modification\_date\_value = QLineEdit(modification\_date\_string)  
 modification\_date\_value.setEnabled(False)  
 second\_row = QHBoxLayout()  
 second\_row.addWidget(modification\_date\_label)  
 second\_row.addWidget(modification\_date\_value)  
 self.layout.addLayout(second\_row)  
  
 access\_date\_label = QLabel("Last Accessed:")  
 access\_date\_value = QLineEdit(access\_date\_string)  
 access\_date\_value.setEnabled(False)  
 third\_row = QHBoxLayout()  
 third\_row.addWidget(access\_date\_label)  
 third\_row.addWidget(access\_date\_value)  
 self.layout.addLayout(third\_row)  
  
  
class SizeWorker(QThread):  
 finished = pyqtSignal(float)  
  
 def \_\_init\_\_(self, file):  
 super(SizeWorker, self).\_\_init\_\_()  
 self.file = file  
  
 def run(self) -> None:  
 size = 0  
 if self.file.is\_file():  
 size = self.file.stat().st\_size  
 elif self.file.is\_dir():  
 size = round(get\_size(self.file), 3)  
 self.finished.emit(size)  
  
  
class Searcher(QThread):  
 found = pyqtSignal(str)  
 finished = pyqtSignal()  
  
 def \_\_init\_\_(self, name, root):  
 super(Searcher, self).\_\_init\_\_()  
 self.name = name  
 self.root = root  
  
 def run(self) -> None:  
 for path, dirs, files in os.walk(self.root):  
 for d in dirs:  
 if self.name == d:  
 self.found.emit(path)  
 for f in files:  
 if str(f).startswith(self.name):  
 self.found.emit(path)  
 self.finished.emit()  
  
  
def get\_size(filepath):  
 total\_size = 0  
 for dirpath, dirnames, filenames in os.walk(filepath):  
 for f in filenames:  
 fp = Path(dirpath) / Path(f)  
 if not fp.is\_symlink():  
 try:  
 total\_size += fp.stat().st\_size  
 except FileNotFoundError:  
 continue  
 return total\_size  
  
  
class MyWidget(QMainWindow, main.Ui\_MainWindow):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.size\_worker = None  
 self.setupUi(self)  
 self.hidden = False  
 self.copy\_this = set()  
 self.setAcceptDrops(True)  
 self.actionBack.triggered.connect(self.go\_back)  
 self.actionHome.triggered.connect(self.home\_dir)  
 self.actionShowHidden.triggered.connect(self.show\_hid)  
 self.actionSearch.triggered.connect(self.file\_search)  
 self.lineEdit.returnPressed.connect(self.goto)  
 self.treeView.setAcceptDrops(True)  
 self.treeView.setDropIndicatorShown(True)  
  
 self.model = QtWidgets.QFileSystemModel()  
 self.model.setRootPath((QtCore.QDir.rootPath()))  
 self.model.setReadOnly(False)  
  
 self.treeView.setContextMenuPolicy(QtCore.Qt.CustomContextMenu)  
 self.treeView.customContextMenuRequested.connect(self.cont\_menu)  
 self.treeView.doubleClicked.connect(self.open\_file)  
 self.treeView.viewport().installEventFilter(self)  
 self.treeView.setModel(self.model)  
 self.treeView. \  
 setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.treeView.setDragDropMode(QTreeView.InternalMove)  
 self.treeView.setSelectionMode(QTreeView.ExtendedSelection)  
 self.treeView.sortByColumn(0, QtCore.Qt.SortOrder(0))  
 self.treeView.setColumnWidth(0, 200)  
 self.treeView.setSortingEnabled(True)  
  
 self.lineEdit.returnPressed.connect(self.goto)  
 self.comboBox.activated.connect(self.path\_changer)  
  
 def click(self, file):  
 self.treeView.setRootIndex(self.model.index(file))  
 self.search\_results.close()  
  
 def file\_search(self):  
 index = self.treeView.rootIndex()  
 rootpath = self.model.filePath(index)  
 if rootpath == "":  
 self.show\_msg("Warning", "Choose disk to look for file!").show()  
 return  
 s, search = QInputDialog.getText(self, "Search", "Input", text="")  
 if search:  
 self.search\_results = SearchResults(s, rootpath)  
 self.search\_results.show()  
 self.search\_results.clicked.connect(self.click)  
  
 def keyPressEvent(self, event):  
 if event.modifiers() & Qt.ControlModifier:  
 if event.key() == Qt.Key\_H:  
 self.home\_dir()  
 if event.key() == Qt.Key\_C:  
 self.copy()  
 if event.key() == Qt.Key\_V:  
 self.paste()  
 if event.key() == Qt.Key\_R:  
 self.change\_name()  
 if event.key() == Qt.Key\_S:  
 self.file\_search()  
 if event.key() == Qt.Key\_Left:  
 self.go\_back()  
 if event.key() == Qt.Key\_Delete:  
 self.delete\_selected()  
  
 def cont\_menu(self):  
 menu = QtWidgets.QMenu()  
 index = self.treeView.selectedIndexes()  
 if len(index) == 0:  
 new\_dir = menu.addAction("New dir")  
 new\_file = menu.addAction("New file")  
 paste = menu.addAction("Paste")  
 new\_dir.triggered.connect(self.new\_dir)  
 new\_file.triggered.connect(self.new\_file)  
 paste.triggered.connect(self.paste)  
 else:  
 file = Path(self.model.filePath(index[0]))  
 \_open = menu.addAction("Open")  
 change = menu.addAction("Change name")  
 delete = menu.addAction("Delete")  
 copy = menu.addAction("Copy")  
 attributes = menu.addAction("Attributes")  
 arc = menu.addAction("Archive")  
 if file.suffix == ".zip":  
 unpack = menu.addAction("Unpack")  
 unpack.triggered.connect(self.unpack)  
 arc.triggered.connect(self.archive)  
 copy.triggered.connect(self.copy)  
 attributes.triggered.connect(self.show\_atts)  
 delete.triggered.connect(self.delete\_selected)  
 \_open.triggered.connect(self.open\_file)  
 change.triggered.connect(self.change\_name)  
 cursor = QtGui.QCursor()  
 menu.exec\_(cursor.pos())  
  
 def dragEnterEvent(self, event):  
 mime = event.mimeData()  
 if mime.hasUrls():  
 event.acceptProposedAction()  
  
 def dropEvent(self, event):  
 for url in event.mimeData().urls():  
 self.copy\_this.add(url.toLocalFile())  
 self.paste()  
 return super().dropEvent(event)  
  
 def open\_file(self):  
 index = self.treeView.currentIndex()  
 file\_path = self.model.filePath(index)  
 self.treeView.clearSelection()  
 if Path(file\_path).is\_file():  
 os.startfile(file\_path)  
 if Path(file\_path).is\_dir():  
 self.treeView.setRootIndex(index)  
 self.lineEdit.setText(file\_path)  
 self.set\_path(file\_path)  
  
 def set\_path(self, path):  
 path\_list = [x for x in path.split("/") if x != ""]  
 self.comboBox.clear()  
 self.comboBox.addItems(path\_list)  
 self.comboBox.setCurrentText(path\_list[len(path\_list) - 1])  
  
 def goto(self):  
 path = self.lineEdit.text()  
 if Path(path).exists():  
 self.treeView.setRootIndex(self.model.index(path))  
 self.set\_path(path)  
 else:  
 self.show\_msg("Error", "Wrong path!")  
  
 def search\_goto(self, path):  
 self.treeView.setRootIndex(self.model.index(path))  
 self.lineEdit.setText(path)  
 self.set\_path(path)  
  
 def path\_changer(self):  
 path\_l = [self.comboBox.itemText(i)  
 for i in range(self.comboBox.currentIndex() + 1)]  
 path = '/'.join(path\_l)  
 self.lineEdit.setText(path)  
 self.treeView.setRootIndex(self.model.index(path))  
  
 def go\_back(self):  
 index = self.model.parent(self.treeView.rootIndex())  
 self.treeView.setRootIndex(index)  
 self.lineEdit.setText(self.model.filePath(index))  
 if self.comboBox.currentIndex() != 0:  
 self.comboBox.setCurrentIndex(self.comboBox.currentIndex() - 1)  
  
 def home\_dir(self):  
 self.lineEdit.clear()  
 self.comboBox.clear()  
 self.model = QtWidgets.QFileSystemModel()  
 self.model.setRootPath((QtCore.QDir.rootPath()))  
 self.treeView.setModel(self.model)  
  
 def change\_name(self):  
 index = self.treeView.selectedIndexes()  
 if len(index) > 4 or len(index) <= 0:  
 self.show\_msg("Error", "Choose one item").show()  
 return  
 self.treeView.edit(index[0])  
  
 def new\_dir(self):  
 index = self.treeView.rootIndex()  
 path = Path(self.model.filePath(index))  
 i, filled = QInputDialog. \  
 getText(self, "Input name", "Input", text="New folder")  
 if filled:  
 if i == "" or i == "." or \  
 i == ".." or re.match(r'.\*[<>:"/\\|?\*].\*', str(i)):  
 self.show\_msg("Error", "Invalid name!").show()  
 else:  
 (path / Path(i)).mkdir(exist\_ok=True)  
  
 def new\_file(self):  
 index = self.treeView.rootIndex()  
 path = Path(self.model.filePath(index))  
 i, filled = QInputDialog. \  
 getText(self, "Input name", "Input", text="New file")  
 if filled:  
 if i == "" or i == "." or \  
 i == ".." or re.match(r'.\*[<>:"/\\|?\*].\*', str(i)):  
 self.show\_msg("Error", "Invalid name!").show()  
 elif str(path) == "C:\\":  
 self.show\_msg("Error", "Can't create file here!").show()  
 else:  
 (path / Path(i)).touch(exist\_ok=True)  
  
 def delete\_selected(self):  
 index = self.treeView.selectedIndexes()  
 for i in index:  
 self.model.remove(i)  
 self.treeView.clearSelection()  
  
 def show\_hid(self):  
 if not self.hidden:  
 self.model.setFilter(QDir.NoDot | QDir.NoDotDot |  
 QDir.Hidden | QDir.AllDirs | QDir.Files)  
 self.hidden = True  
 else:  
 self.model.setFilter(QDir.NoDot | QDir.NoDotDot |  
 QDir.AllDirs | QDir.Files)  
 self.hidden = False  
  
 def copy(self):  
 self.copy\_this.clear()  
 indexes = self.treeView.selectedIndexes()  
 for i in indexes:  
 self.copy\_this.add(self.model.filePath(i))  
  
 def paste(self):  
 index = self.treeView.rootIndex()  
 try:  
 for i in self.copy\_this:  
 file\_to\_copy = Path(i)  
 if not file\_to\_copy.exists():  
 self.show\_msg("Error", "File not exists!").show()  
 continue  
 path = self.model.filePath(index)  
 if Path(i).is\_dir():  
 path = path + f"/{file\_to\_copy.name}"  
 if Path(path).exists():  
 path += " - copy at " + str(round(time.time() \* 1000))  
 shutil.copytree(i, path)  
 else:  
 if Path(path + '/' + file\_to\_copy.name).exists():  
 path = path \  
 + '/' + file\_to\_copy.stem + " - copy" \  
 + file\_to\_copy.suffix  
 shutil.copy2(i, path)  
 except PermissionError:  
 self.show\_msg("Warning", "Run as admin to do that").show()  
  
 def show\_msg(self, title, text):  
 msg = QMessageBox(self)  
 if title == "Warning":  
 msg.setIcon(QMessageBox.Warning)  
 elif title == "Error":  
 msg.setIcon(QMessageBox.Critical)  
 elif title == "Info":  
 msg.setIcon(QMessageBox.Information)  
 msg.setWindowTitle(title)  
 msg.setText(text)  
 return msg  
  
 def show\_atts(self):  
 if self.size\_worker is not None:  
 self.size\_worker.terminate()  
 index = self.treeView.selectedIndexes()  
 self.file = Path(self.model.filePath(index[0]))  
 self.size\_worker = SizeWorker(self.file)  
 self.size\_worker.finished.connect(self.report)  
 self.size\_worker.start()  
 self.atts\_win = self.show\_msg("Info", "Calculating size...")  
 self.atts\_win.show()  
  
 def report(self, n):  
 self.atts\_win.close()  
 self.atts\_win = AttributeWindow(self.file, n)  
 self.atts\_win.show()  
  
 def archive(self):  
 index = self.treeView.selectedIndexes()  
 file = Path(self.model.filePath(index[0]))  
 if len(index) > 4:  
 self.model.mkdir(self.treeView.rootIndex(), "zip")  
 files\_set = set()  
 file = file.parent / Path("zip")  
 for i in index:  
 files\_set.add(self.model.filePath(i))  
 for i in files\_set:  
 if Path(i).is\_dir():  
 shutil.copytree(i, file / Path(i).name)  
 else:  
 shutil.copy2(i, file)  
 os.chdir(file.parent)  
 i, filled = QInputDialog. \  
 getText(self, "Input name", "Input", text=file.stem)  
 if filled:  
 if i == "" or i == "." or \  
 i == ".." or re.match(r'.\*[<>:"/\\|?\*].\*', str(i)):  
 self.show\_msg("Error", "Invalid name").show()  
 elif (file.parent / Path(i + ".zip")).exists():  
 self.show\_msg("Error", "Already exists!").show()  
 else:  
 shutil.make\_archive(i, "zip", file.parent, file.name)  
 if len(index) > 4:  
 shutil.rmtree(file)  
  
 def unpack(self):  
 index = self.treeView.selectedIndexes()  
 file = Path(self.model.filePath(index[0]))  
 os.chdir(file.parent)  
 i, filled = QInputDialog. \  
 getText(self, "Input name", "Input", text=file.stem)  
 if filled:  
 if i == "" or i == "." or i == ".." or \  
 re.match(r'.\*[<>:"/\\|?\*].\*', str(i)):  
 self.show\_msg("Error", "Invalid name").show()  
 elif (file.parent / Path(i)).exists():  
 self.show\_msg("Error", "Already exists!").show()  
 else:  
 shutil.unpack\_archive(file, i)  
  
 def eventFilter(self, obj, event):  
 if (  
 obj is self.treeView.viewport() and  
 event.type() == QtCore.QEvent.MouseButtonPress  
 ):  
 ix = self.treeView.indexAt(event.pos())  
 if not ix.isValid():  
 self.treeView.clearSelection()  
 return super(MyWidget, self).eventFilter(obj, event)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 ex = MyWidget()  
 ex.show()  
 sys.exit(app.exec\_())

Код файла main.py (интерфейс):

from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(700, 650)  
 MainWindow.setMinimumSize(QtCore.QSize(650, 600))  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.verticalLayout\_2 = QtWidgets.QVBoxLayout(self.centralwidget)  
 self.verticalLayout\_2.setObjectName("verticalLayout\_2")  
 self.frame = QtWidgets.QFrame(self.centralwidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.frame.sizePolicy().hasHeightForWidth())  
 self.frame.setSizePolicy(sizePolicy)  
 self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)  
 self.frame.setFrameShadow(QtWidgets.QFrame.Raised)  
 self.frame.setObjectName("frame")  
 self.verticalLayout = QtWidgets.QVBoxLayout(self.frame)  
 self.verticalLayout.setObjectName("verticalLayout")  
 self.horizontalLayout = QtWidgets.QHBoxLayout()  
 self.horizontalLayout.setObjectName("horizontalLayout")  
 self.comboBox = QtWidgets.QComboBox(self.frame)  
 self.comboBox.setObjectName("comboBox")  
 self.horizontalLayout.addWidget(self.comboBox)  
 self.lineEdit = QtWidgets.QLineEdit(self.frame)  
 self.lineEdit.setObjectName("lineEdit")  
 self.horizontalLayout.addWidget(self.lineEdit)  
 self.verticalLayout.addLayout(self.horizontalLayout)  
 self.treeView = QtWidgets.QTreeView(self.frame)  
 self.treeView.setObjectName("treeView")  
 self.verticalLayout.addWidget(self.treeView)  
 self.verticalLayout\_2.addWidget(self.frame)  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(MainWindow)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 962, 26))  
 self.menubar.setObjectName("menubar")  
 self.menuHome = QtWidgets.QMenu(self.menubar)  
 self.menuHome.setObjectName("menuHome")  
 MainWindow.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName("statusbar")  
 MainWindow.setStatusBar(self.statusbar)  
 self.actionHome = QtWidgets.QAction(MainWindow)  
 self.actionHome.setObjectName("actionHome")  
 self.actionBack = QtWidgets.QAction(MainWindow)  
 self.actionBack.setObjectName("actionBack")  
 self.actionShowHidden = QtWidgets.QAction(MainWindow)  
 self.actionShowHidden.setObjectName("actionShowHidden")  
 self.actionSearch = QtWidgets.QAction(MainWindow)  
 self.actionSearch.setObjectName("actionSearch")  
 self.menuHome.addAction(self.actionHome)  
 self.menuHome.addAction(self.actionBack)  
 self.menuHome.addAction(self.actionShowHidden)  
 self.menuHome.addAction(self.actionSearch)  
 self.menubar.addAction(self.menuHome.menuAction())  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  
 self.menuHome.setTitle(\_translate("MainWindow", "Menu"))  
 self.actionHome.setText(\_translate("MainWindow", "Home"))  
 self.actionBack.setText(\_translate("MainWindow", "Back"))  
 self.actionShowHidden.setText(\_translate("MainWindow", "Show hidden files"))  
 self.actionSearch.setText(\_translate("MainWindow", "Search"))

# Пояснительная записка

Название проекта – File Browser.

Авторами проекта являются Панин А. А. и Нагаев А. С.

Идея проекта заключалась в создании файлового менеджера, не перегруженного множеством функций (и, следовательно, более оптимизированного), как, к примеру, «Проводник» на Windows, но при этом позволяющего свободно выполнять основные функции по работе с файлами.

Основное окно программы – MainWindow.

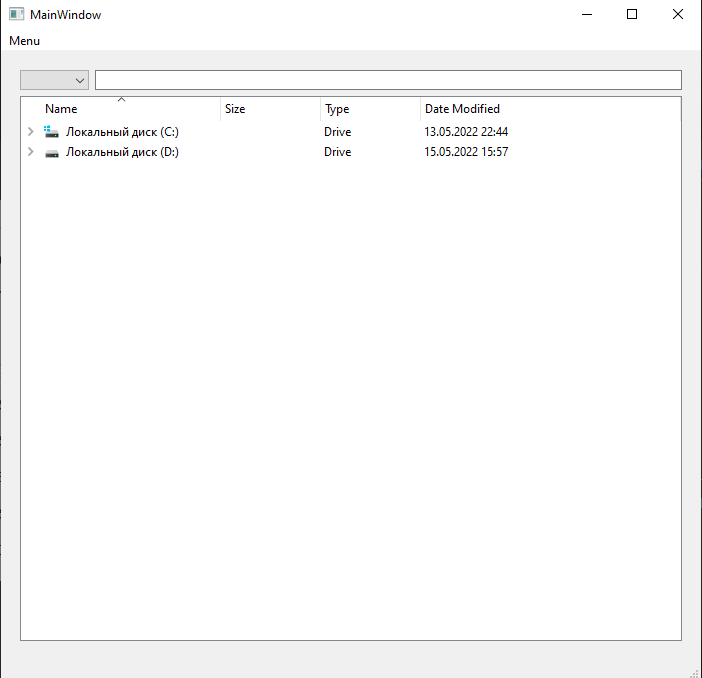


Рис. 3.1 – MainWindow

В этом окне отображается список файлов и директорий с их названиями, указанными размерами, типами (папка, исполняемый файл и т.д.) и датами последней модификации. Открытие файла или папки происходит двойным щелчком левой кнопки мыши.

Щелчок правой кнопкой мыши по файлу, папке или текущей директории вызовет меню действий для этого файла. К ним относятся Открытие, Переименование, Удаление, Копирование, Свойства и Архивирование (если файл уже не является архивом). Нажатие ПКМ по пустому месту предложит создать новый файл, создать новую директорию или вставить скопированный файл.

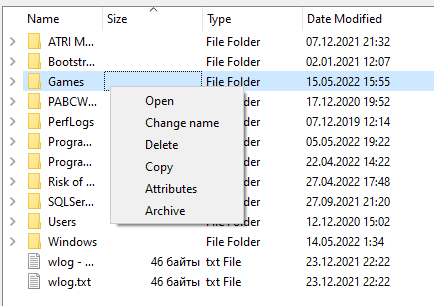


Рис 3.2 – Меню действий с файлом

Перемещение файлов по директориям реализовано с использованием функции Drag and Drop – пользователь может перетащить нужный файл в нужную ему директорию. Более того, пользователь может перемещать файлы между двумя и более окнами File Browser и даже между другими файловыми менеджерами.

Нажатие кнопки Свойств выведет основную информацию о файле: Размер, Дату последнего изменения и Дату последнего доступа. Подсчет размера файла – достаточно трудоемкий процесс, но поскольку сбор свойств файла выведен в отдельный поток, его запуск не мешает работе с менеджером.

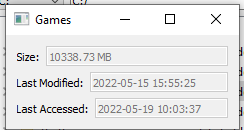


Рис. 3.3 – Свойства файла

В программе также реализован поиск файлов в директориях. Пользователь может ввести искомое имя файла и начать поиск, после чего он получит окно со всеми совпадающими результатами. Нажатие на результат приведет пользователя к директории, где хранится конкретный файл. Поскольку этот процесс также выведен в отдельный поток, использование поиска не затрудняет работу с файловым менеджером.

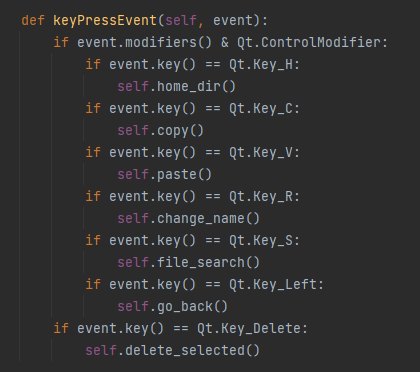
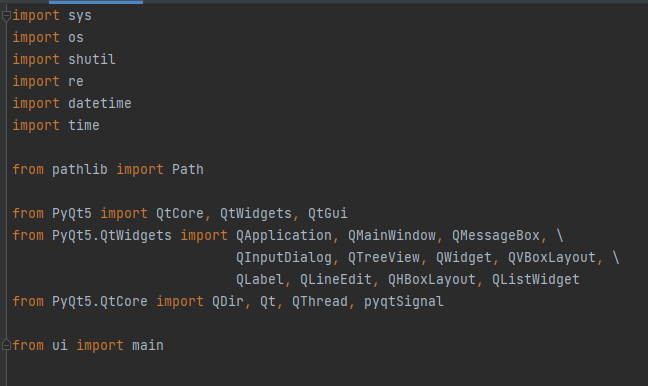


Рис. 3.4 – Комбинации «горячих клавиш» в коде программы

В программе присутствует поддержка комбинаций «горячих клавиш» с зажатой клавишей Ctrl. Реализованы привычные всем Ctrl + C Ctrl + V для копирования и вставки файлов, а также некоторые другие.

Для выполнения данного проекта было использовано несколько библиотек из модуля PyQt5.

Рис. 3.5 – Использованные библиотеки и модули

Таким образом, с использованием большинства изученных в курсе технологий и некоторых особенных приемов, цель проекта была достигнута.