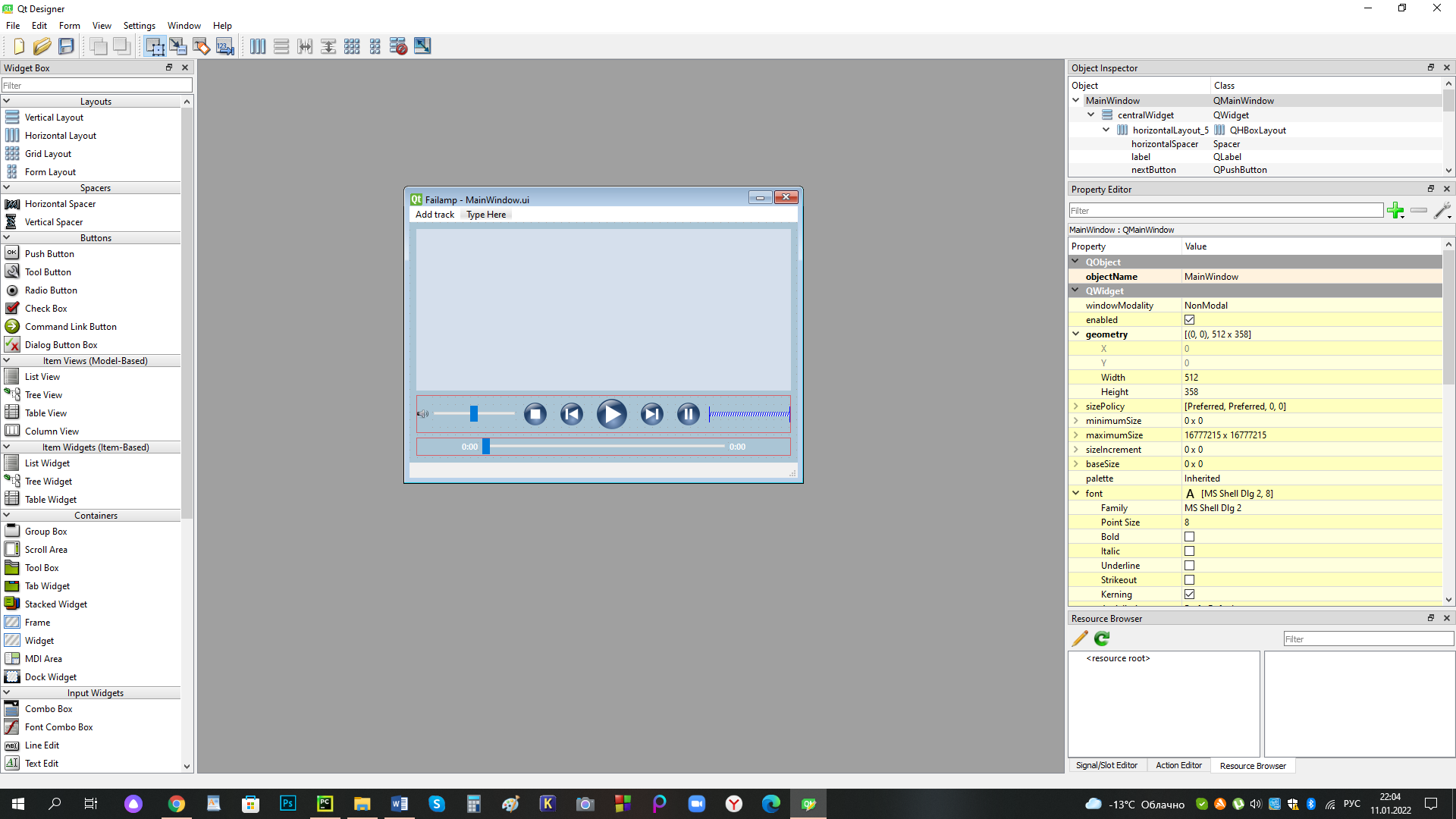
PollyTunes - это простой аудиоплеер, реализованный на Python, использующий встроенный список воспроизведения Qt и функции обработки мультимедиа.

Он очень слабо смоделирован по образцу оригинального Winamp, хотя и далеко не так совершенен (отсюда и ошибка).

**Главное окно**

Пользовательский интерфейс главного окна был построен с использованием Qt Designer, невероятно удобная вещь, всем советую. На приведенном ниже снимке экрана показан созданный макет.



Макет построен в виде QVBoxLayout, который, в свою очередь, содержит QListView (представление списка воспроизведения) и два горизонтальных макета QHBoxLayout, которые содержат ползунок времени, индикаторы времени и кнопки управления.

Кнопки нарисованы вручную, я очень старалась.

if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication([])  
 app.setApplicationName("PollyTunes")  
 app.setStyle("Fusion")  
  
 palette = QPalette()  
 palette.setColor(QPalette.Window, QColor(53, 53, 53))  
 palette.setColor(QPalette.WindowText, Qt.white)  
 palette.setColor(QPalette.Base, QColor(25, 25, 25))  
 palette.setColor(QPalette.AlternateBase, QColor(53, 53, 53))  
 palette.setColor(QPalette.ToolTipBase, Qt.white)  
 palette.setColor(QPalette.ToolTipText, Qt.white)  
 palette.setColor(QPalette.Text, QColor(106, 134, 174))  
 palette.setColor(QPalette.Button, QColor(53, 53, 53))  
 palette.setColor(QPalette.ButtonText, Qt.white)  
 palette.setColor(QPalette.BrightText, Qt.red)  
 palette.setColor(QPalette.Link, QColor(42, 130, 218))  
 palette.setColor(QPalette.Highlight, QColor(42, 130, 218))  
 palette.setColor(QPalette.HighlightedText, Qt.black)  
 app.setPalette(palette)  
 app.setStyleSheet("QToolTip { color: #ffffff; background-color: #2a82da; border: 1px solid white; }")  
  
 window = MainWindow()  
 app.exec\_()

**Игрок**

Что касается управления, тут все сделано невероятно просто. Отдельные кнопки отвечают за пуск и паузу, за переход к следующему и предыдущему треку, так же есть отдельный регулятор громкости. Все это вы можете видеть на приведенном ранее макете.

**Плейлист**

Список воспроизведения можно передать проигрывателю, который затем будет использовать его для автоматического выбора дорожки для воспроизведения после завершения текущей. Проще говоря, вам не нужно вручную переключать на следующий трек, после того, как доиграет текущий, что является неоспоримым плюсом.

Кнопки управления «предыдущий» и «следующий» присутствуют, я считаю, это успех (все это обрабатывается с помощью QMediaPlaylist).

Отображение списка воспроизводимых треков обрабатывается QListViewобъектом. В нашем случае мы сохраняем данные в объекте списка воспроизведения QMediaPlaylist из модуля Qt Multimedia.

**Открытие и запуск**

У нас есть замечательная операция “открыть файл” — которая добавляет файл в список воспроизведения.

def open\_file(self):  
 path, \_ = QFileDialog.getOpenFileName(self, "Open file", "", "mp3 Audio (\*.mp3)")  
 if path:  
 self.playlist.addMedia(  
 QMediaContent(  
 QUrl.fromLocalFile(path)  
 )  
 )  
 self.model.layoutChanged.emit()

**Модель плейлиста**

Как уже упоминалось, мы используем QListViewобъект для отображения списка воспроизведения с данными, хранящимися в QMediaPlaylist. Поскольку хранилище данных уже обработано, все, что нам нужно сделать — это сопоставить список воспроизведения с тем, как это увидит пользователь.

В этом случае требования довольно простые — нам нужно:

* метод rowCount для возврата общего количества строк в списке воспроизведения через .mediaCount()
* метод data, который возвращает данные для определенной строки (мы отображаем только имя файла)

Сохранив ссылку на список воспроизведения \_\_init\_\_ можно легко получить другие данные в любое время. Изменения плейлиста в приложении будут автоматически отражаться в окне.

class PlaylistModel(QAbstractListModel):  
 def \_\_init\_\_(self, playlist, \*args, \*\*kwargs):  
 super(PlaylistModel, self).\_\_init\_\_(\*args, \*\*kwargs)  
 self.playlist = playlist  
  
 def data(self, index, role):  
 if role == Qt.DisplayRole:  
 media = self.playlist.media(index.row())  
 return media.canonicalUrl().fileName()  
  
 def rowCount(self, index):  
 return self.playlist.mediaCount()

**Таймер**

Наконец, нам нужен метод для преобразования времени в миллисекундах в вид h:m:s или m:s, ну, очевидно, что так удобнее. Для этого мы можем использовать divmod с миллисекундами для каждого временного интервала. Это возвращает количество полных делений (div) и остаток (mod). Оставшаяся задача заключается в том, чтобы отображать часовую часть только тогда, когда время превышает час.

def hhmmss(ms):  
 h, r = divmod(ms, 36000)  
 m, r = divmod(r, 60000)  
 s, \_ = divmod(r, 1000)  
 return ("%d:%02d:%02d" % (h, m, s)) if h else ("%d:%02d" % (m, s))