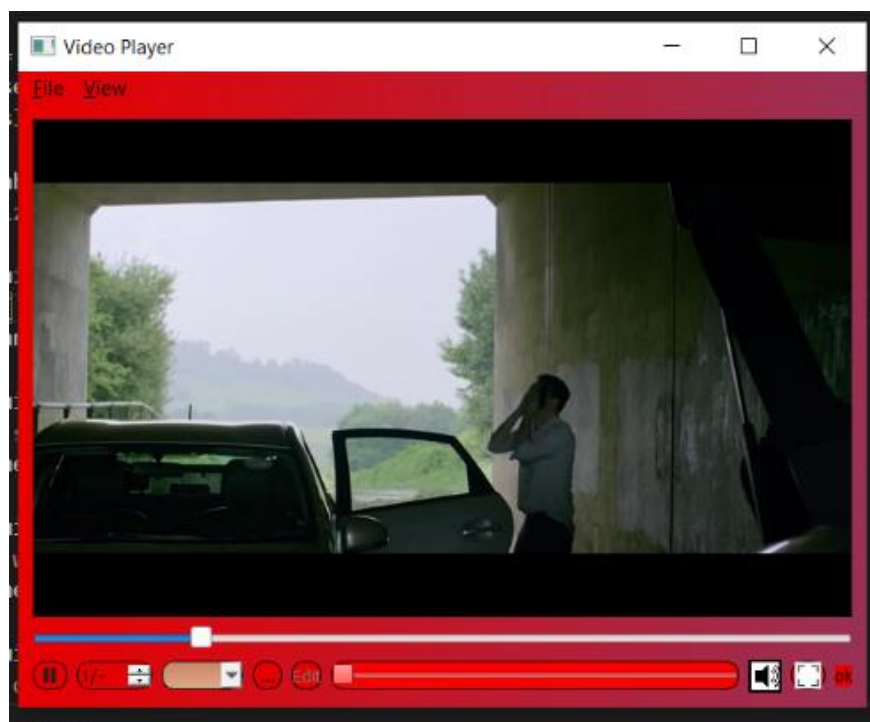


گزارش:

شمایی از خروجی برنامه



قابلیت های برنامه به صورت کلی:

پخش ویدیو

کم و زیاد کردن سرعت ویدیو

با یک فایل اکسل می توان فیلم را بخش بندی کرد وبا اضافه کردن به زمان های مختلف فیلم پرش کرد و در خود برنامه امکان به روز رسانی می باشد.

برای نوشتن یک برنامه پخش ویدیو از کلاس QMediaPlayer یک instance میسازیم. این کلاس توابع و اسلات ها و سیگنال های متفاوتی دارد. برای مثال در این پروژه از اسلات ها، سیگنال ها و توابع زیر استفاده کرده ایم.

- Play ()
- setVideoOutput ()
- stateChanged ()
- pause ()
- durationChanged ()
- error ()

- `setVolume ()`
- `setPlaybackRate ()`
- `setPosition ()`
- `setMedia ()`
- `state ()`
- `PlayingState ()`
- `setPosition ()`
- `errorString ()`

در برنامه کلاس های زیر تعریف شده است. کلاس `VideoWindow` که از کلاس `QMainWindow` ارث بری می کند و کلاس `EditMessageBox` که از `QtWidgets.QMessageBox` ارث بری می کند، در ادامه توضیحات آن ها بیان می شود.

VideoWindow – ۱

این کلاس برای توصیف ویژگی های پخش کننده ویدیو می باشد که ترتیب متغیرهای آن به ترتیب زیر است:

`Menu bar`: شامل گزینه های `file` و `view` است که برای نوشتن آنها از کد زیر استفاده شده است. برای `file` دو زیرمنوی `open` و `exit` برای باز کردن و بستن ویدیوی موردنظر گذاشته شده اند. کلیدهای میانبر `ctrl+o` و `ctrl+q` را میتوان استفاده کرد.

```
# Create menu bar and add action
menuBar = self.menuBar()
fileMenu = menuBar.addMenu('&File')
ViewMenu = menuBar.addMenu('&View')
ViewMenu.addAction(colormodeAction)
fileMenu.addAction(openAction)
fileMenu.addAction(exitAction)
# Create a widget for window contents

# Create open action
openAction = QAction(QIcon('open.png'), '&Open', self)
openAction.setShortcut('Ctrl+O')
openAction.setStatusTip('Open movie')
openAction.triggered.connect(self.openFile)

# Create exit action
exitAction = QAction(QIcon('exit.png'), '&Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.setStatusTip('Exit application')
exitAction.triggered.connect(self.exitCall)
```

بعد از کلیک بر روی `open` تابع `openfile` فراخوانی می شود که با استفاده از `QFileDialog` به کاربر امکان انتخاب فایل را می دهد. آدرس فایل انتخاب شده با استفاده از تابع `QMediaContent` فایل ویدیویی را از آدرسی که در `openfile` کاربر داده لود میکند. در تابع `openfile` بعد از لود ویدیو دکمه ی پخش فعال می شود.

```
def openFile(self):
    fileName, _ = QFileDialog.getOpenFileName(self, "Open Movie",
                                              QDir.homePath())

    if fileName != '':
        self.mediaPlayer.setMedia(
            QMediaContent(QUrl.fromLocalFile(fileName)))
        self.playButton.setEnabled(True)
```

view: – 2

در قسمت `view` می توان تم محیط پخش ویدیو را تغییر داد. در اینجا چند تم متفاوت تعریف شده است.

```
#change theme
default= QAction(QtGui.QIcon('color.jpg'), '&default color', self)
default.setStatusTip('change color')
default.triggered.connect(self.default_color)

theme1 = QAction(QtGui.QIcon('color.jpg'), '&sky mode', self)
theme1.setStatusTip('sky mode')
theme1.triggered.connect(self.theme1)

theme2 = QAction(QtGui.QIcon('color.jpg'), '& white color', self)
theme2.setStatusTip('white color mode')
theme2.triggered.connect(self.theme2)

theme3 = QAction(QtGui.QIcon('color.jpg'), '&dark mode', self)
theme3.setStatusTip('dark')
theme3.triggered.connect(self.theme3)
```

3- برای responsive بودن صفحه ی پلیر از دو نوع layout، QVBoxLayout() و QHBoxLayout() استفاده کردیم که یکی افقی و دیگری عمودی است. در QVBoxLayout()، دو شی videoWidget و Qslider ساخته شده اند.

۴-VideoWidget

یک شی از این کلاس تعریف میکنیم که باید در آخر آن را به عنوان ورودی () setVideoOutput که متدی از mediaPlayer است بدهیم.

```
#place item for second layout
layout = QVBoxLayout()
layout.addWidget(videoWidget)
layout.addWidget(self.positionSlider)
layout.addLayout(controllLayout)

# Set widget to contain window contents
wid.setLayout(layout)
```

یک QWidget میسازیم تا بتوانیم آیتم های layout را نمایش دهیم و آن را به عنوان ورودی به تابع setlayout میدهیم. در controllayout دکمه های پخش، سرعت دوبرابر، ویرایش و اضافه کردن تگ، کم و زیاد کردن صدا، برچسب برای بلندگو و full screen کردن می باشد.

```
#place items for first layout
controllayout.setContentsMargins(0, 0, 0, 0) #set cor
controllayout.addWidget(self.playButton)
controllayout.addWidget(self.douplex_btn)
controllayout.addWidget(self.cb)
controllayout.addWidget(self.test_btn)
controllayout.addWidget(self.edit_btn)
controllayout.addWidget(self.sld)
controllayout.addWidget(self.label)
controllayout.addWidget(self.fullscreenButton)
```

توضیح به انواع آتم های استفاده شده:

۱- QPushButton

دکمه ی اجرای ویدیو می باشد که با متد `setStyleSheet` استایل آن را تغییر میدهیم. با استفاده از `setEnabled` در ابتدا آن را غیرفعال میکنیم. با استفاده از `setIcon` آیکون آن را تغییر میدهیم و از آیکون استاندارد `SP_MediaPlay` در `Qstyle` استفاده میکنیم. سیگنال `clicked` آن تابع `play` را فراخوانی میکند.

```
#play_button
self.playButton = QPushButton()
self.playButton.setStyleSheet("border-radius : 50;")
self.playButton.setEnabled(False)
self.playButton.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))
self.playButton.clicked.connect(self.play)
```

در تابع `play()` ابتدا چک میکنیم که اگر `state` الان با `state` در حال پخش یکی باشد `pause` بشه. اگر `playingstate==1` و `mediaplayer.state==1` شد ویدیو `pause` میشه در غیر اینصورت پخش می شود که اسلایدر مربوط به `play` هم `state` جدید میگیرد.

```
def play(self):
    if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
        print(self.mediaPlayer.state(), QMediaPlayer.PlayingState)
        self.mediaPlayer.pause()

    else:
        self.mediaPlayer.play()
        self.mediaPlayer.setVolume(self.sld.value())
```

spin_speed-۲

برای تغییر سرعت پخش ویدیو از یک شی از کلاس QDoubleSpinBox استفاده کردیم تا کاربر با کم یا زیاد کردن آن سرعت مناسب پخش خود را تنظیم کند که تابع doublex بعد از تغییر مقدار spin box فراخوانی می شود که مقدار آن را به setPlaybackRate می دهد.

```
#set speed for play |
self.spin_speed = QDoubleSpinBox (self)
self.spin_speed.setStyleSheet(style)
self.spin_speed.setDecimals(1)
self.spin_speed.setSingleStep(0.1)
self.spin_speed.setProperty("value", 1.0)
self.spin_speed.valueChanged.connect(self.doublex)
```

```
#play video with selected speed
def doublex(self):
    value = self.spin_speed.value()
    self.mediaPlayer.setPlaybackRate(value)
```

addtag-۳

یک QFileDialog باز میشود و کاربر تگ مورد نظرش را انتخاب میکند که یک فایل excel است. بعد از آنکه QFileDialog آدرس آن فایل را به دست آورد آن را با کتابخانه ی pandas میخوانیم و آن را به لیست تبدیل میکنیم. لیست ما شامل دو لیست زمان و موضوعات می باشد که در ادامه آیتم های موضوعات را در آیتم های combo box اضافه میکنیم. یک attribute برای کلاس VideoWindow تعریف کردیم که هدف از آن نگهداری لیست اکسل دریافتی میباشد و در اینجا آن را مقداردهی میکنیم.

```
#add tag for video
def addtag(self):

    fileName, _ = QFileDialog.getOpenFileName(self, "Open Tag",QDir.homePath())

    if fileName != '':
        print(fileName)
        df = pd.read_excel(fileName)
        s = df.values
        self.value=s
        for i in s:
            self.cb.addItem(i[1])
```

edit-۴

به منظور ادیت کردن لیست تگ ها به کار می رود که بعد از کلیک کردن آن تابع `edit_tag` فراخوانی میشود. برای ادیت یک پنجره ی جدید باز می شود که در کلاس دیگری به نام `EditMessageBox` تعریف شده است که ورودی `EditMessageBox` ، `attribute value` می باشد. مقدار بازگشتی از کلاس `EditMessageBox` لیست ادیت شده می باشد به همین منظور مقادیر قبلی `combo box` را حذف می کنیم و آنرا با مقادیر جدید پر می کنیم. و در آخر در این تابع لیست جدید را در مسیری که کاربر فایل را انتخاب کرده ذخیره می شود.

```
def edit_tag(self):
    ex = EditMessageBox (self.value) #Create Object and sent value for fill the table for editing
    a = ex.r()

    if not isinstance(a,np.ndarray):          #call fuction to return modified data
        self.value = a.values
        self.cb.clear()
        for i in a.values: #update combobox
            self.cb.addItem(i[1])

        #save edited value to current address
        a.to_excel(self.path,index=0,header = None, engine='xlsxwriter')
```

توضیح کلاس `EditMessageBox`

این کلاس از کلاس `QtWidgets.QMessageBox` ارث می برد که در آن دو دکمه اجرا و لغو و یک `QTableWidget` اضافه کردیم. توجه داشته باشید در سازنده آن مقدار `VALUE` که شامل لیست مقادیر گرفته شده از اکسل می باشد را به عنوان ورودی به آن دادیم و در سازنده آن متغیر `a` را پر کردیم. در تابع `addTableWidget` با توجه به لیست دریافتی جدول را پر می کنیم. تابع `event` برای تنظیم آیتم های موجود برای تنظیم فاصله بین آنها و تعیین مقدار مناسب برای نمایش مناسب نوشته شده است. کاربر با کلیک کردن روی جدول می تواند مقدار مورد نظر خود را وارد کند و ویرایش را انجام میدهد.

```
class EditMessageBox(QtWidgets.QMessageBox):
    def __init__(self,items):
        QtWidgets.QMessageBox.__init__(self)
        self.setSizeGripEnabled (True)

        self.setWindowTitle('ویرایش تگ ها')
        self.setIcon(self.Question)
        self.setText("ویرایش")
        self.a=items
        self.addButton (
            QtWidgets.QPushButton('ذخیره'),
            QtWidgets.QMessageBox.YesRole
        )

        self.addButton(
            QtWidgets.QPushButton('لغو'),
            QtWidgets.QMessageBox.RejectRole
        )

        self.addTableWidget (self,items)

        currentClick = self.exec_()
```

و بعد از کلیک بر روی دکمه ذخیره مقدار `currentClick` یک می شود. و در حلقه مقادیر جدول خوانده می شود. و در `self.a` ریخته می شود. اگر کلید لغو را بزند اتفاق خاص نمی افتد.

```
if currentClick==0 :
    rows = self.tableWidget.rowCount()
    columns = self.tableWidget.columnCount()
    df = pd.DataFrame()
    for i in range(rows):
        for j in range(columns):
            df.loc[i, j] = str(self.tableWidget.item(i, j).text())
    self.a=df

if currentClick==2 :
    pass
```

توضیحات مربوط به مدیاپلیر و اسلایدر:

اگر `state` مدیا تغییر کند تابع `mediastateChanged` فراخوانی میشود که در آن اگر در حالت مکث باشد آیکون مربوط به مکث را نشان می دهد و اگر در حالت پخش باشد آیکون مربوط به پخش فعال می شود.

```
def mediaStateChanged(self, state):
    print("mediaStateChanged")
    if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
        self.playButton.setIcon(
            self.style().standardIcon(QStyle.SP_MediaPause))
    else:
        self.playButton.setIcon(
            self.style().standardIcon(QStyle.SP_MediaPlay))
```

تابع `positionchanged`: مقدار جدید اسلایدر را آپدیت میکند.

```
def positionChanged(self, position):
    self.positionSlider.setValue(position)
```

تابع `durationchanged`: باتوجه به طول ویدیو مقدار `range` اسلایدر را تعیین میکند.

```
def durationChanged(self, duration):  
    print("duration",duration)  
    self.positionSlider.setRange(0, duration)
```

توضیح تابع `skip`

```
#combi box for displaying tag  
self.cb = QComboBox()  
self.cb.setStyleSheet(style+"background-image : url(../image/icon/index.png);")  
self.cb.activated.connect(self.skip)
```

بعد از انتخاب آیتم مورد نظر در `combo box` تابع `skip` صدا زده می شود. ابتدا اندیس آیتم انتخاب شده مشخص می شود بعد با توجه به مقادیر `self.value` ثانیه آن به دست می آید و به ورودی تابع `setPosition` زمان را در ۱۰۰۰ می دهیم چرا که ورودی تابع بر اساس میلی ثانیه می باشد. و مقدار اسلایدر را آپدیت می کنیم. تا بعد از پرش ویدیو `slider` هم پرش کند.

```
def skip(self):  
    print(self.value)  
    index = self.cb.currentIndex()  
    print("index",index,type(index))  
    t = self.value[index][0]  
    self.setPosition(int(t)*1000)  
    self.positionSlider.setValue(t*1000)
```

تابع `handleError`:

اگر فیلم در حین لود دچار مشکل شد بر چسبی که به صورت پیش فرض مقدار `OK` را دارد پیام خطا را می نویسد. و دکمه اجرا را غیر فعال می کند.

```
def handleError(self):  
    self.playButton.setEnabled(False)  
    self.errorLabel.setText("Error: " + self.mediaPlayer.errorString())
```

کلمات کلیدی و امکانات تعریف شده:

با click بر روی دکمه full screen صفحه تمام صفحه می شود. یک پیغامی از اینکه برای خروج دکمه esc را فشار دهید نمایش داده می شود. همچنین توجه داشته باشید با دو بار click کردن در صفحه چنان که صفحه تمام صفحه نبود تمام صفحه می شود. و اگر در حالت تمام صفحه بود به حالت اول خود باز می گردد.

```
def show_fullscreen(self):
    msg = QMessageBox()
    QMessageBox.about(self, " ", "را فشار دهید esc برای خروج دکمه")

    # retval = msg.exec_()
    self.showFullScreen()
    self.fullscreenButton.setIcon(QtGui.QIcon('outzoom.png'))

def exit_fullscreen(self):
    self.showNormal()
```

```
# DOUBLE click the full screen enabled
def mouseDoubleClickEvent(self, event):
    if self.isFullScreen():
        self.showNormal()
    else:
        self.showFullScreen()
```

با فشار دادن کلید های Key_Right تابع forwardSlider فراخوانی میشود، ویدیو در حدود یک دقیقه به جلو می کشد. با فشار دادن کلید Key_Left ویدیو را حدود یک دقیقه از موقعیت فعلی به عقب می برد. با فشار دادن کلید Key_Escape از حالت تمام صفحه خارج می شود.

```
#keyboard shortcut
#key right forward 1000
self.shortcut = QShortcut(QtGui.QKeySequence(QtCore.Qt.Key_Right), self)
self.shortcut.activated.connect(self.forwardSlider)
self.shortcut = QShortcut(QtGui.QKeySequence(QtCore.Qt.Key_Left), self)
self.shortcut.activated.connect(self.backwardSlider)
self.shortcut = QShortcut(QtGui.QKeySequence(QtCore.Qt.Key_Escape), self)
self.shortcut.activated.connect(self.exit_fullscreen)

#set value for slider
```

```
def forwardSlider(self):
    self.mediaPlayer.setPosition(self.mediaPlayer.position() + 1000*60)

#backward the video for 1000*60 ms
def backwardSlider(self):
    self.mediaPlayer.setPosition(self.mediaPlayer.position() - 1000*60)
```