```python
#maim_window.py

from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QPixmap, QColor, QImage
from PyQt5.QtWidgets import QMainWindow, QColorDialog
import os
import cv2
import numpy as np
from PIL import Image

from ui_py.ui_mainwindow import Ui_MainWindow
from dialogs.open_load_dialog import OpenLoadDialog
from dialogs.saved_values_paths import SavedValuesConstants

REPLACED_FILE_NAME = "savedImage.jpg"
COLOR_RANGE = 20


class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()

        # setup ui layout
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        # register event handlers
        self.ui.radioButton_camera.toggled.connect(self.__refresh_ui)
        self.ui.radioButton_picture.toggled.connect(self.__refresh_ui)
        self.ui.radioButton_video.toggled.connect(self.__refresh_ui)
        self.ui.actionLoader.triggered.connect(self.on_action_loader_triggered)
        self.ui.pushButton_replace.pressed.connect(self.show_replace_result)

self.ui.post_it_color_pushButton.pressed.connect(self.on_action_settings_color_picker

        self.post_it_color_rgb_value = np.array([])

        self.replace_image_path = None
        self.original_picture_path = None
        self.original_video_path = None

        self.__refresh_post_it_color_line_edit()

    def on_action_loader_triggered(self):
        dialog = OpenLoadDialog()
        if not dialog.exec_():
            return

        self.replace_image_path = dialog.replace_image_file_path()
        self.original_picture_path = dialog.original_picture_file_path()
        self.original_video_path = dialog.original_video_file_path()

        self.__refresh_ui()

    def on_action_settings_color_picker_triggered(self):
        title = "Choose the color of post-it"
        color = SavedValuesConstants.SettingsColorPicker.CUSTOMIZED_COLOR_POST_IT
        color_pick = QColorDialog.getColor(color, self, title, QColorDialog.ShowAlpha

        self.__store_post_it_color(color_pick)
        self.__update_post_it_color()
```

```
61          self.__refresh_post_it_color_line_edit()
62
63      def show_replace_result(self):
64          assert self.replace_image_path is not None
65          replace_img = self.__convert_rgb_to_bgr(self.replace_image_path)
66          replace_img = Image.fromarray(replace_img)
67
68          # check which radiobutton has been selected
69          if self.ui.radioButton_picture.isChecked():
70              self.ui.pushButton_pause.setEnabled(False)
71              self.on_picture_button_is_checked(replace_img)
72          if self.ui.radioButton_video.isChecked():
73              self.ui.pushButton_pause.setEnabled(True)
74              self.on_video_button_is_checked(replace_img)
75          if self.ui.radioButton_camera.isChecked():
76              self.ui.pushButton_pause.setEnabled(False)
77              self.on_camera_button_is_checked(replace_img)
78
79      def on_picture_button_is_checked(self, replace_img):
80          assert self.original_picture_path is not None
81          self.replace_frame = self.show_result_picture(self.original_picture_path, rep
82          self.__refresh_replaced_result(self.replace_frame)
83
84      def on_video_button_is_checked(self, replace_img):
85          assert self.original_video_path is not None
86          self.cap = cv2.VideoCapture(self.original_video_path)
87          self.replace_img = replace_img
88          self.timer_video = QTimer(self)
89          self.timer_video.timeout.connect(self.show_result_video)
90          self.timer_video.start(50)
91
92      def on_camera_button_is_checked(self, replace_img):
93          self.cap = cv2.VideoCapture(0)
94          self.replace_img = replace_img
95          self.timer_camera = QTimer(self)
96          self.timer_camera.timeout.connect(self.show_result_camera_stream)
97          self.timer_camera.start(5)
98
99      def show_result_picture(self, pic_path, replace_img):
100         # Picture, change the post-it part in pic
101         frame = self.__convert_rgb_to_bgr(pic_path)
102         replace_result = self.detect_and_replace(frame.copy(), replace_img)
103         return replace_result
104
105     def show_result_video(self):
106         # Video, change the post-it part in video
107         replace_img = self.replace_img
108         ret, frame = self.cap.read()
109
110         if frame is not None:
111             replace_result = self.detect_and_replace(frame.copy(), replace_img)
112
113             replace_result = self.__convert_bgr_to_rgb(replace_result)
114             frame = self.__convert_bgr_to_rgb(frame)
115
116             self.__refresh_original_video(frame)
117             self.__refresh_replaced_video(replace_result)
118
119             if self.ui.pushButton_pause.isChecked():
120                 self.timer_video.stop()
121                 return
```

```
122
123         else:
124             self.timer_video.stop()
125             return
126
127     def show_result_camera_stream(self):
128         # webcam, change the post-it part in camera stream
129         replace_img = self.replace_img
130         ret, frame = self.cap.read()
131
132         if frame is not None:
133             replace_result = self.detect_and_replace(frame.copy(), replace_img)
134
135             replace_result = self.__convert_bgr_to_rgb(replace_result)
136             frame = self.__convert_bgr_to_rgb(frame)
137
138             self.__refresh_original_video(frame)
139             self.__refresh_replaced_video(replace_result)
140
141         else:
142             self.timer_camera.stop()
143             return
144
145     def detect_and_replace(self, frame, replace_img):
146         contours, hierarchy = self.find_contours(frame)
147         try:
148             # find biggest bounding box
149             biggest, index = 0, 0
150             rect = list()
151             for i in range(len(contours)):
152                 x, y, w, h = cv2.boundingRect(contours[i])
153                 rect.append(((x, y), (x + w, y + h), (w, h)))
154                 if w * h > biggest:
155                     biggest = w * h
156                     index = i
157
158             # draw bounding box in captured image
159             height = rect[index][1][1]- rect[index][0][1]
160             width = rect[index][1][0] - rect[index][0][0]
161             trans_pt1 = (int(rect[index][0][0]+width/4), int(rect[index][0][1]+height
162             trans_pt2 = (int(rect[index][1][0]-width/4), int(rect[index][1][1]-height
163
164             # frame = cv2.rectangle(frame, trans_pt1, trans_pt2, (0, 0, 255), 2)
165             temp_replace = np.array(replace_img.resize((int(trans_pt2[0]-
trans_pt1[0]),int(trans_pt2[1]-trans_pt1[1]))))
166             frame[trans_pt1[1] : trans_pt2[1], trans_pt1[0] : trans_pt2[0]] = temp_re
167
168         except:
169             pass
170         return frame
171
172     def set_mask(self, frame):
173         lower = self.post_it_color_rgb_value - COLOR_RANGE
174         upper = self.post_it_color_rgb_value + COLOR_RANGE
175         return cv2.inRange(frame, lowerb=lower, upperb=upper)
176
177     def find_contours(self, frame):
178         mask = self.set_mask(frame)
179         return cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
180
181     @staticmethod
```

```
182     def __convert_rgb_to_bgr(rgb_img_path):
183         # open image with Image.open to avoid error from opencv
184         # convert rgb image to bgr for opencv
185         rgb_img = np.array(Image.open(rgb_img_path).convert("RGB"))
186         return cv2.cvtColor(rgb_img, cv2.COLOR_RGB2BGR)
187 
188     @staticmethod
189     def __convert_bgr_to_rgb(bgr_img):
190         return cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)
191 
192     @staticmethod
193     def __store_post_it_color(color_pick):
194         SavedValuesConstants.SettingsColorPicker.CUSTOMIZED_COLOR_POST_IT = color_pic
195 
196     def __update_post_it_color(self):
197         rgb_value = SavedValuesConstants.SettingsColorPicker.CUSTOMIZED_COLOR_POST_IT
198         self.post_it_color_rgb_value = np.array([rgb_value.blue(), rgb_value.green(),
rgb_value.red()])
199 
200     def __refresh_ui(self):
201         self.__update_post_it_color()
202         self.__refresh_replace_image(self.replace_image_path)
203         self.__refresh_original_picture(self.original_picture_path)
204 
205     def __refresh_replace_image(self, replace_image_path):
206         pix_map = QPixmap(replace_image_path)
207         self.ui.replace_image_label.setPixmap(pix_map)
208 
209     def __refresh_original_picture(self, original_picture_path):
210         pix_map = QPixmap(original_picture_path)
211         self.ui.original_source_label.setPixmap(pix_map)
212 
213     def __refresh_replaced_result(self, frame):
214         frame = self.__convert_bgr_to_rgb(frame)
215         source_image = QImage(frame.data, frame.shape[1], frame.shape[0], QImage.Form
216         self.ui.replacement_result_label.setPixmap(QPixmap.fromImage(source_image))
217 
218     def __refresh_original_video(self, frame):
219         source_image = QImage(frame.data, frame.shape[1], frame.shape[0], QImage.Form
220         self.ui.original_source_label.setPixmap(QPixmap.fromImage(source_image))
221 
222     def __refresh_replaced_video(self, replace_result):
223         show_image = QImage(
224                 replace_result.data, replace_result.shape[1], replace_result.shape[0]
QImage.Format_RGB888
225             )
226         self.ui.replacement_result_label.setPixmap(QPixmap.fromImage(show_image))
227 
228     def __refresh_post_it_color_line_edit(self):
229         self.ui.post_it_lineEdit.setStyleSheet(
230             "QLineEdit { background-color: %s}"
231             % SavedValuesConstants.SettingsColorPicker.CUSTOMIZED_COLOR_POST_IT.name(
232         )
233 
```