```python
"""door_select.py | Robin Forestier | 07.03.2022

This file is used to select doors in images.
"""

import cv2
import numpy as np
import pickle


class DoorSelect:
    """The DoorSelect class is a class that contains a method that allows the user to select a door."""
    def __init__(self, img = None):
        """ __init__ is the constructor of the class DoorSelect.
        :param img: the image that will be used to select the doors.
        :type img: numpy.ndarray
        """

        # the image
        self.img = img


        self.doors = []
        """Doors
        List containing the 4 corners coordinations of each door.
        """

        self.fleches = []
        """Fleches
        List containing the coordination of the arrowhead.
        """

        # number of point already placed (return to 0 after each door)
        self.npoints = 0

        if img is not None:
            # a copy of the clean image (used when you delete a selection)
            self.img_copy = img.copy()
            cv2.namedWindow("door select")
            cv2.setMouseCallback('door select', self.mouse_event)
            self.run()

    def run(self):
        """
        [INFO] Doors selection phase.
        [INFO] To select a door:
        [INFO] 1. Left click the four corner of the door.
        [INFO] 2. left click on the side where you enter the room through the door.
        [INFO] Repeat these 2 steps as many times as you have doors.
        [INFO] If you want to delete a selection, right click on it.
        [INFO] Press < SPACE > when you are done.
        """

        print(self.run.__doc__)

        # The loop is infinite until the user press the <SPACE> key.
        while cv2.waitKey(1) != 32:
            cv2.imshow("door select", self.img)

        # check if doors are selected (with an arrow)
        if len(self.doors) % 4 == 0 and len(self.fleches)  == len(self.doors) / 4:
            with open('doors.pickle', 'wb') as f:
                # Save the list self.doors and self.fleches with pickle
                save = (self.doors, self.fleches)
                pickle.dump(save, f)

            print("[INFO] Selected doors successfully saved.\n")

            i = 0
            # This is a way to iterate over the list self.doors 4 by 4.
            for door in zip(*[iter(self.doors)] * 4):
                arr = np.array(door)
                x, y, w, h = cv2.boundingRect(arr)
                #cv2.rectangle(self.img, (x, y), (x + w, y + h), (0, 255, 0), 2)
                img_croped = self.img_copy[y:y+h, x:x+w]
                cv2.imwrite("door_{0}.png".format(i), img_croped)
                i = i + 1

        else:
            print("[ERROR] No door select ! \n")

        print("[INFO] Press < ENTER > to close the app.")
        print("[INFO] Press < SPACE > to select a new door. \n")

        # retry or close the app
        while True:
            key = cv2.waitKey(1)
            if key == 32: # space
                self.run()
            elif key == 13: # enter
                print("[INFO] Stop the app.")
                cv2.destroyAllWindows()
                break

    def sort_points(self, door):
        """sort_points is used for sorting the 4 corner of the door like that:
        0      1
         +-----+
         |     |
         +-----+
        2      3

        :param door: a list of four points that represent the four corners of the door
        :type door: list
        :return: a list of 4 points that represent the corners of the door.
        :rtype: list
        """

        # It creates a numpy array of 4 rows and 2 columns.
        rect = np.zeros((4, 2), dtype="float32")

        # It's summing the y coordinates of the points.
        s = np.sum(door, axis=1)

        # min -> corner 0
        # max -> corner 2
        rect[0] = door[np.argmin(s)]
        rect[2] = door[np.argmax(s)]
        # diff of the poits y

        # It contains the difference between the y coordinates of the points.
        diff = np.diff(door, axis=1)
        # min -> corner 1
        # max -> corner 3
        rect[1] = door[np.argmin(diff)]
        rect[3] = door[np.argmax(diff)]

        return rect

    def draw_door(self):
        """Draw the door outline and the arrow"""

        n_door = 0
        # drawing of the 4 points arround the door
        for point in self.doors:
            cv2.circle(self.img, point, 4, (0, 0, 255), -1)
        # drawing of the arrowhead
        for point in self.fleches:
            cv2.circle(self.img, point, 4, (0, 255, 0), -1)

        # drawing of all the lines
        for door in zip(*[iter(self.doors)] * 4):
            # It's sorting the 4 points of the door in order to draw the door.
```

```python
            rect = self.sort_points(door)

            # It's converting the points from float to integer.
            rect = rect.astype(int)

            # It's drawing the door.
            cv2.line(self.img, tuple(rect[0]), tuple(rect[1]), (0, 0, 255), 1)
            cv2.line(self.img, tuple(rect[1]), tuple(rect[2]), (0, 0, 255), 1)
            cv2.line(self.img, tuple(rect[2]), tuple(rect[3]), (0, 0, 255), 1)
            cv2.line(self.img, tuple(rect[3]), tuple(rect[0]), (0, 0, 255), 1)

            # It's computing the center of the bottom of the door.
            center = (int((rect[3][0] + rect[2][0]) / 2), int((rect[3][1] + rect[2][1]) / 2))

            cv2.circle(self.img, center, 4, (0, 0, 255), -1)

            # This is a way to draw the arrowhead.
            if len(self.fleches) >= n_door + 1:
                cv2.line(self.img, center, self.fleches[n_door], (255, 0, 0), 2)

            n_door = n_door + 1

    def delete_door(self, x, y):
        """Delete a door already selected by right click on it

        :param x: The x-coordinate of the mouse-click
        :type x: int
        :param y: The y-coordinate of the point
        :type y: int
        """

        n_door = 0
        for door in zip(*[iter(self.doors)] * 4):
            # sort the points
            rect = self.sort_points(door)
            rect = rect.astype(int)
            # if you click between the point 0 and 3
            if rect[0][0] < x < rect[2][0] and rect[0][1] < y < rect[2][1]:
                print("[INFO] Door deleted \n")

                # delete the 4 points of the door
                del(self.doors[n_door:n_door + 4])

                # if it's create delete the coresponding arrow
                if len(self.fleches)  >= (len(self.doors) + 4) / 4:
                    del(self.fleches[int(n_door / 4)])
                else:
                    # if the arrow has not been created, the next point should not be the arrow.
                    self.npoints = 0

                self.img = self.img_copy.copy()
                self.draw_door()

            n_door = n_door + 4

    def mouse_event(self, event, x, y, flags, params):
        """Execute when mouse is used on image.

        :param event: The event that took place (left mouse button pressed, left mouse button released, mouse movement, etc)
        :type event: int
        :param x: The x-coordinate of the event
        :type x: int
        :param y: The y-coordinate of the click
        :type y: int
        :param flags: The flags are the optional parameters to the mouse callback function
        :type flags: int
        :param params: extra parameters passed to the callback function
        :type params: int
        """

        # Checking if the left button of the mouse is pressed.
        if event == cv2.EVENT_LBUTTONDOWN:
            # outline selection
            if self.npoints <= 3:
                self.doors.append((x, y))
                self.npoints = self.npoints + 1
            # fleche selection
            elif self.npoints < 5:
                self.fleches.append((x, y))
                self.npoints = self.npoints + 1

            # new selection
            if self.npoints >= 5:
                self.npoints = 0

        # This is a way to draw the door when the mouse is released.
        elif event == cv2.EVENT_LBUTTONUP:
            self.draw_door()

        # Checking if the right button of the mouse is pressed.
        if event == cv2.EVENT_RBUTTONDOWN:
            self.delete_door(x, y)


if __name__ == '__main__':
    # read the file video_d.avi
    cap = cv2.VideoCapture(0)
    # take the first img of the video
    _, img = cap.read()
    img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_AREA)
    # create object DoorSelect
    d = DoorSelect(img)

```