```python
from tkinter import *
import csv
import cv2
from roboflow import Roboflow
from Drone import Drone

from Master import Master
from utils.Utils import Utils


# main.py
# authors: Diogo RosÃ¡rio, JoÃ£o Raposo
# Description:  This file retrieves the trained Roboflow models using the provided APIs and performs predictions
#               on a specified test (image_path). Subsequently, the master agent is invoked to compile and process all images
#               without overlap, ensuring the best possible identification of each corresponding drone.
#               Additionally, the file manages the post-processing steps by saving the master agent's results in a CSV file named
#               "mastersResults.csv" for future reference. Furthermore, it provides real-time visibility by printing the obtained
#               results directly to the console during execution.


image_path = "mastersTests/test-10.jpg"
predicitonA_path = "predictions/predicitonA.jpg"
predicitonB_path = "predictions/predicitonB.jpg"
predicitonC_path = "predictions/predicitonC.jpg"
predictionM_path = "predictions/predictionM.jpg"
confidence = 51
overlap = 0

#Loading models of each drone
rfA = Roboflow(api_key="usQXRh13NGjn2HK6BwFP")
project1 = rfA.workspace().project("drone-a-tb89u")
modelA = project1.version(2).model

rfB = Roboflow(api_key="usQXRh13NGjn2HK6BwFP")
project2 = rfB.workspace().project("drone-b-xkvry")
modelB = project2.version(1).model

rfC = Roboflow(api_key="usQXRh13NGjn2HK6BwFP")
project3 = rfC.workspace().project("drone-c-hytyd")
modelC = project3.version(5).model

# Drones Reputation / confidence
drone_A_confidence = 0.1
drone_B_confidence = 0.5
drone_C_confidence = 0.5

# Open a csv file called "mastersResults.csv" and write the results of the identifications processed by the master.
# In this format: 'A - Cars', 'A - Houses', 'A - Trees', 'B - Cars', 'B - Houses', 'B - Trees','C - Cars', 'C - Houses', 'C - Trees','A Reputation','B

csv_filename = 'masterResults.csv'
with open(csv_filename, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['A - Cars', 'A - Houses', 'A - Trees',
                     'B - Cars', 'B - Houses', 'B - Trees',
                     'C - Cars', 'C - Houses', 'C - Trees',
                     'A Reputation','B Reputation','C Reputation'])

    ################################################################
    ################### DRONE A ####################################

    predictA = modelA.predict(image_path, confidence=confidence, overlap=overlap)
    drone_A = Drone("A", drone_A_confidence, predictA.json())
    predictA.save(predicitonA_path)
    drone_A.saveInCsv()

    ################################################################
    ################### DRONE B ####################################

    predictB = modelB.predict(image_path, confidence=confidence, overlap=overlap)
    drone_B =  Drone("B", drone_B_confidence, predictB.json())
    predictB.save(predicitonB_path)
    drone_B.saveInCsv()

    ################################################################
    ################### DRONE C ####################################

    predictC = modelC.predict(image_path, confidence=confidence, overlap=overlap)
    drone_C = Drone("C", drone_C_confidence, predictC.json())
    predictC.save(predicitonC_path)
    drone_C.saveInCsv()

    ################################################################
    ###################### CALCULATION #############################
    ################################################################

    master = Master(drone_A,drone_B , drone_C)

    clone_img = cv2.imread(image_path)
    masterImage = clone_img.copy()

    # Draw the result identifications
    for box in master.identifications:
        Utils.addLabel(masterImage, box.x, box.y, box.width, box.height, box.class_type, box.confidence, box.drone)

    cv2.imwrite(predictionM_path, masterImage)
    print(str(len(master.identifications)))
```

```python
from Identification import Identification
import csv

# Drone.py
# Authors: Diogo RosÃ¡rio, JoÃ£o Raposo
# Description: This file defines the Drone object utilized in main.py. The objective is to simulate a drone with the following attributes:
# - Name: A or B or C
# - Confidence: Initial reputation
# - Json: JSON file containing predictions/identifications from corresponding models
# - Identifications: List of predictions/identifications from each model

class Drone():
    def __init__(self,name, confidence, json):
        self.name = name
        self.confidence = confidence
        self.json = json
        self.identifications = self.createObjects()



    # Generates the identification list for each Drone object by utilizing the JSON file created during the model's prediction process.
    def createObjects(self):
        objects = []

        for iteration in self.json.get("predictions"):
            x = int(iteration.get("x"))
            y = int(iteration.get("y"))
            width = int(iteration.get("width"))
            height = int(iteration.get("height"))
            ident_confidence = iteration.get("confidence")
            class_type = iteration.get("class")

            x = int(x - round(width / 2))
            y = int(y - round(height / 2))
            obj = Identification(x,y,width,height,ident_confidence,class_type, self.name, self.confidence)
            objects.append(obj)

        return objects

    # Saves the identifications in the corresponding csv file
    def saveInCsv(self):

        if self.name == "A":
            csv_filename = 'droneA_Identification.csv'
            with open(csv_filename, 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(['Identification', 'x', 'y', 'width', 'height', 'confidence', 'class_type'])

                for i, identification in enumerate(self.identifications, start=1):
                    writer.writerow([f'identification {i}', identification.x, identification.y, identification.width,
                                     identification.height, identification.confidence, identification.class_type])

            print(f"CSV file '{csv_filename}' has been created.")

        elif self.name == "B":
            csv_filename = 'droneB_Identification.csv'
            with open(csv_filename, 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(['Identification', 'x', 'y', 'width', 'height', 'confidence', 'class_type'])

                for i, identification in enumerate(self.identifications, start=1):
                    writer.writerow([f'identification {i}', identification.x, identification.y, identification.width,
                                     identification.height, identification.confidence, identification.class_type])

            print(f"CSV file '{csv_filename}' has been created.")

        elif self.name == "C":
            csv_filename = 'droneC_Identification.csv'
            with open(csv_filename, 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(['Identification', 'x', 'y', 'width', 'height', 'confidence', 'class_type'])

                for i, identification in enumerate(self.identifications, start=1):
                    writer.writerow([f'identification {i}', identification.x, identification.y, identification.width,
                                     identification.height, identification.confidence, identification.class_type])

            print(f"CSV file '{csv_filename}' has been created.")
```

```python
from shapely.geometry import Polygon


# Identification.py
# Authors: Diogo RosÃ¡rio, JoÃ£o Raposo
# Description: This file defines the Identification object used in main.py. It aims to represent an identification with the following attributes:
# - x: Float representing the starting width in pixels of the identification
# - y: Float representing the starting height in pixels of the identification
# - width: Width of the identification
# - height: Height of the identification
# - confidence: Confidence value indicating the drone's belief in the identification as a Car, House, or Tree
# - class_type: Type of identification, either "Car," "House," or "Tree"
# - drone_confidence: Confidence/reputation value specific to each drone; a static value
class Identification():

    def __init__(self, x, y, width, height, confidence, class_type, drone, drone_confidence):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.confidence = confidence
        self.class_type = class_type
        self.drone = drone
        self.drone_confidence= drone_confidence

    # Creates two polygons that correspond to two identifications(rectangle)
    # Checks if to identifications collide with each other
    def checkCollision(self, other):
        # Create polygons for each rectangle
        poly1 = Polygon([(self.x , self.y),
                         (self.x + self.width , self.y),
                         (self.x + self.width , self.y + self.height),
                         (self.x , self.y + self.height)])

        poly2 = Polygon([(other.x , other.y),
                         (other.x + other.width , other.y),
                         (other.x + other.width , other.y + other.height),
                         (other.x , other.y + other.height)])

        # Check if the polygons (rectangles) intersect
        return poly1.intersects(poly2) or poly2.intersects(poly1)

    # Function that compares two identifications.
    # Used to sort a list of identifications
    def comparator(this, other):
        confidence_this = this.confidence * this.drone_confidence
        confidence_other = other.confidence * other.drone_confidence

        if(confidence_this > confidence_other):
            return 1
        elif(confidence_this < confidence_other):
            return -1
        else:
            if(this.drone_confidence > other.drone_confidence):
                return 1
            elif(this.drone_confidence < other.drone_confidence):
                return -1
            else:
                return 0


    # String representation of this object (Identification)
    def __str__(self):
        return "[\nClass: " + self.class_type + "\n" + "Confidence: " + str(self.confidence) + "\n" + "x: " + str(self.x) + "\n" + "y: " + str(self.y)

```

```python
from functools import cmp_to_key
from Identification import Identification

# Drone.py
# Authors: Diogo RosÃ¡rio, JoÃ£o Raposo
# Description: Represents the master agent object and conducts all necessary computations to provide the larger identifications among the three drones
# - drone_A: Drone A
# - drone_B: Drone B
# - drone_C: Drone C
# - count_A_cars: Integer value of the number of cars identified by drone A
# - count_A_Houses: Integer value of the number of houses identified by drone A
# - count_A_Trees: Integer value of the number of trees identified by drone A
# - count_B_cars: Integer value of the number of cars identified by drone B
# - count_B_Houses: Integer value of the number of houses identified by drone B
# - count_B_Trees: Integer value of the number of trees identified by drone B
# - count_C_cars: Integer value of the number of cars identified by drone C
# - count_C_Houses: Integer value of the number of houses identified by drone C
# - count_C_Trees: Integer value of the number of trees identified by drone C
# - identifications: List of the result identifications
class Master():

    def __init__(self, droneA, droneB, droneC):
        self.drone_A = droneA
        self.drone_B = droneB
        self.drone_C = droneC

        self.count_A_cars = 0
        self.count_A_Houses = 0
        self.count_A_trees = 0

        self.count_B_cars = 0
        self.count_B_Houses = 0
        self.count_B_trees = 0

        self.count_C_cars = 0
        self.count_C_Houses = 0
        self.count_C_trees = 0


        self.identifications = self.calculateIdentifications(self.drone_A, self.drone_B, self.drone_C)
        for ident in self.identifications:
            if(ident.drone == "A"):
                if(ident.class_type == 'Car'):
                     self.count_A_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_A_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_A_trees += 1

            elif(ident.drone == "B"):
                if(ident.class_type == 'Car'):
                     self.count_B_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_B_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_B_trees += 1

            elif(ident.drone == "C"):
                if(ident.class_type == 'Car'):
                     self.count_C_cars += 1
                elif(ident.class_type == 'House'):
                    self.count_C_Houses += 1
                elif(ident.class_type == 'Tree'):
                    self.count_C_trees += 1

    #Calulate the identifications that are bigger than others. Check the method "comparator" in identifications.py to see what it means "Bigger"
    def calculateIdentifications(self, drone_A, drone_B, drone_C):
        identificationsCheckpoint = []

        for drone_A_ident in drone_A.identifications:
            self.addIdentification(identificationsCheckpoint, drone_A_ident)

        for drone_B_ident in drone_B.identifications:
            self.addIdentification(identificationsCheckpoint, drone_B_ident)

        for drone_C_ident in drone_C.identifications:
            self.addIdentification(identificationsCheckpoint,drone_C_ident)

        return identificationsCheckpoint

    # Tries to add 1 identification.
    # If doesnt collide with another identification -> adds to the list of identifications
    # If does collide with another identification(s) performs the method "resolveCollision"
    def addIdentification(self, list,box2):

        if(box2.drone_confidence * box2.confidence != 0):
            if(self.canCollide(list,box2)):
                indexes = self.findAllThatCollide(list, box2)
                self.resolveCollision(list, indexes, box2)
            else:
                list.append(box2)


    # Identifies all indices of identifications Y with which a given identification X collides.
    # Returns the list of indices
    def findAllThatCollide(self, list, box2):
        indexs = []
        for box1 in list:
            if(box1.checkCollision(box2)):
                indexs.append(list.index(box1))

        return indexs

    # Verifies whether Identification X collides with at least one identification Y from the provided list.
    # Returns true if so.
    def canCollide(self, list, box2):
        for box1 in list:
```

```
        for box1 in list:
            if(box1.checkCollision(box2)):
                return True
        return False

    # In the event of a collision, executes a sequence of methods to eliminate the collision.
    def resolveCollision(self, list, indexes, box2):
        boxes = []

        # Get all the boxes from the indeces
        for i in indexes:
            boxes.append(list[i])

        # Check if box2 has lower confidence that ANY other box that collide with
        # If yes, dont add
        if(self.hasLowerConfidence(boxes, box2)):
            return

        # If last step doesnt work, find the box with lowest confidence and remove it and test
        boxes = sorted(boxes, key=cmp_to_key(Identification.comparator))
        for box1 in boxes:
            if(Identification.comparator(box2, box1) >= 0):
                index = list.index(box1)
                list.pop(index)
                if(not self.canCollide(list, box2)):
                    list.append(box2)
                    return

    # Returns true if at least one identification (box1) from the given list is larger than identification Y (box2).
    def hasLowerConfidence(self, boxes, box2):
        for box1 in boxes:
            if(Identification.comparator(box1, box2) >= 0):
                return True
        return False
```