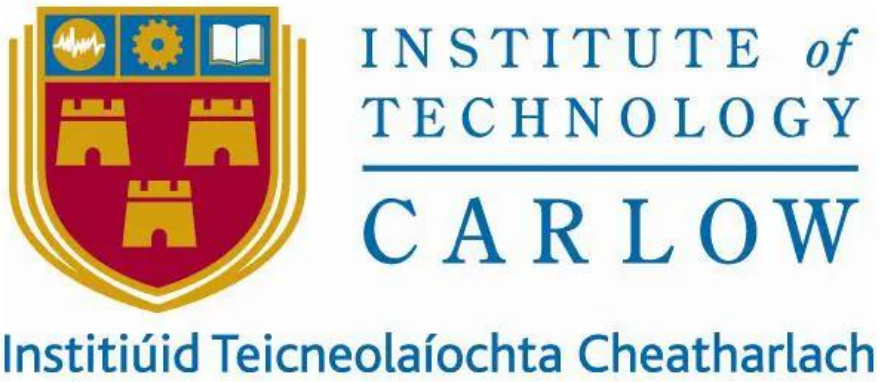
4th year project
Automated Drone Air Traffic Control System

Technical Manual

Institute of Technology Carlow



Supervisor: Dr. Oisin Cawley
Author: James Hall
Submission Date: 27 April 2021

Abstract

The purpose of this document is to explain the technical aspects of the Automated Drone Traffic Control System. This includes installation instructions, and a will showcase the codebase for the project.

Table of Contents

Introduction

The purpose of the Automated Drone Traffic Control System is to allow its user to operate a drone or fleet of drones without the need for constant input. The user can add any number of drones to the system as well as any number of locations for the drone to take off and land. The user may then use the system to schedule flights for the drones to take part in. Once the user begins a flight, the chosen drone will take off and fly to its destination automatically. The system is capable of performing multiple simultaneous flights with any number of drones.

There are collision detection mechanisms in place to ensure that the drones do not collide with one another while in flight. The system also has the ability to abort any flight in case of emergency and direct drones to the nearest safe landing area.

Installation

Firstly the operating system required for the application to run is Ubuntu 18.04, This is due to the use of Olympe and it only being compatible with this build of Ubuntu.

To install the application it must first be cloned from the github repository at:
https://github.com/Jamhougin/DroneTrafficControlSystem

Once downloaded to your system. Open the Setup and use the included README for instructions on how to set up the virtual environment.

README.text

```
Required OS Ubuntu 18.04

The user may firstly open up a terminal in the same folder as this README.

If the setup.sh has been opened in Windows previously the user may need to
remove some hidden Dos commands using:

"sudo apt install dos2unix"
"dos2unix setup.sh"

In the terminal the setup.sh file needs to be given execution permission with:

"chmod 777 setup.sh"

Next run the setup.sh using:

"source ./setup.sh"

With the virtual environment set up the Program may be run from the terminal
using:

"python3 MapViewApp.py"
```

Code

MapViewApp.py

```
# -*- coding: UTF-8 -*-


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0

#Main App Screen and Layers

"""
MapViewApp.py
================================

This is the core module for the ADTCS project
"""
import os
import random
from math import *
from kivy.garden.mapview.mapview.utils import clamp
```

```
import time
import pickle
import threading
from DroneController import *
from AbstractDrone import AbstractDrone
from VirtualDrone import VirtualDrone
from Location import Location
from Flight import Flight
from save_load import *
from PrintLists import *
from FloatInput import FloatInput

from kivy.config import Config
Config.set('graphics','maxfps','60')
from kivy.lang import Builder

from kivy.uix.behaviors.focus import FocusBehavior
from kivy.uix.screenmanager import ScreenManager, Screen

from kivy.uix.widget import Widget
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.textinput import TextInput
from kivy.uix.popup import Popup
from kivy.uix.image import Image
from kivy.properties import ObjectProperty, NumericProperty, StringProperty, Boole
anProperty
from kivy.uix.recycleview import RecycleView
from kivy.uix.scrollview import ScrollView
from kivy.core.window import Window

from kivy.app import App
from kivy.clock import Clock
from kivy.graphics import Color, Line
from kivy.graphics.transformation import Matrix
from kivy.graphics.context_instructions import Translate, Scale
from kivy.garden.mapview.mapview import MapView, MapLayer, MapMarker, MarkerMapLay
er, MIN_LONGITUDE, MIN_LATITUDE, MAX_LATITUDE, MAX_LONGITUDE

from functools import partial

drones = LoadDroneList()
locations = LoadLocationList()
flights = LoadFlightList()
threads = []
coordinates = []
corridors = []
network = []
drone_hidden = True
location_hidden = True
new_location_add = False
name = ''
location_layer = None

class MapViewApp(App):
    """
    Main App Class
```

```
    """
    mapview = None

    def __init__(self, **kwargs):
        super(MapViewApp, self).__init__(**kwargs)
        Clock.schedule_once(self.post, 0)

    def build(self):
        self.title = "Drone Air Traffic Control"
        layout = BoxLayout(orientation='vertical')
        return layout

    def post(self, *args):

        global location_layer
        layout = FloatLayout()
        self.mapview = MapView(zoom=18, lat=52.824725, lon=-6.935661)
        line = LineMapLayer()
        self.mapview.add_layer(line, mode="scatter")  # window scatter
        drone_layer = DroneMarkerLayer()
        self.mapview.add_layer(drone_layer, mode="scatter")
        location_layer = LocationMarkerLayer()
        self.mapview.add_layer(location_layer, mode="scatter")
        flights_layer = FlightLayer()
        self.mapview.add_layer(flights_layer, mode="scatter")
        layout.add_widget(self.mapview)

        dronebuttoncolor = [.3,1,0,1] #Green
        locationbuttoncolor = [.5,1,1,1] #Blue
        flightbuttoncolor = [1,1,.5,1] #Yellow

        #Button containers
        buttonrow1 = BoxLayout(orientation='horizontal',height='30dp',size_hint_y=
None)
        buttonrow2 = BoxLayout(orientation='horizontal',height='30dp',size_hint_y=
None)
        buttonrow3 = BoxLayout(orientation='horizontal',height='30dp',size_hint_y=
None)

        buttonrow1.add_widget(Button(text="RemoveDrone",on_press=lambda a: drone_l
ayer.remove_drone_popup(), background_color = dronebuttoncolor))
        buttonrow1.add_widget(Button(text="AddDrone",on_press=lambda a: drone_laye
r.add_virtual_drone_popup(), background_color = dronebuttoncolor))
        buttonrow1.add_widget(Button(text="ViewLocationsList",on_press=lambda a: l
ocation_layer.view_locations_popup(), background_color = locationbuttoncolor))
        buttonrow1.add_widget(Button(text="ViewLocationMarkers", background_color
= locationbuttoncolor, on_press=lambda a: show_locations_button(self.mapview, loca
tion_layer, drone_layer)))

        buttonrow2.add_widget(Button(text="ViewDrones", background_color = dronebu
ttoncolor,on_press=lambda a: show_drones_button(self.mapview, drone_layer, locatio
n_layer)))
        buttonrow2.add_widget(Button(text="AddVirtualDrone",on_press=lambda a: dro
ne_layer.add_virtual_drone_popup(), background_color = dronebuttoncolor))
        buttonrow2.add_widget(Button(text="AddLocation",on_press=lambda a: locatio
n_layer.add_point_popup(), background_color = locationbuttoncolor))
        buttonrow2.add_widget(Button(text="RemoveLocation",on_press=lambda a: loca
tion_layer.remove_location_popup(), background_color = locationbuttoncolor))
```

```
        buttonrow3.add_widget(Button(text="ViewFlightsList",on_press=lambda a: fli
ghts_layer.view_flights_popup(), background_color = flightbuttoncolor))
        buttonrow3.add_widget(Button(text="CreateFlight",on_press=lambda a: flight
s_layer.create_flight_popup(), background_color = flightbuttoncolor))
        buttonrow3.add_widget(Button(text="RemoveFlight",on_press=lambda a: flight
s_layer.remove_flight_popup(), background_color = flightbuttoncolor))
        buttonrow3.add_widget(Button(text="StartFlight",on_press=lambda a: flights
_layer.start_flight_popup(), background_color = flightbuttoncolor))
        buttonrow3.add_widget(Button(text="AbortFlight",on_press=lambda a: flights
_layer.abort_flight_popup(), background_color = flightbuttoncolor))
        #First button row
        self.root.add_widget(buttonrow1)
        #Second button row
        self.root.add_widget(buttonrow2)
        #Third button row
        self.root.add_widget(buttonrow3)
        self.root.add_widget(layout)

        #Used to draw locationmarkers on screen
        def show_locations_button(mapview, locationslayer, droneslayer):
            global location_hidden
            global drone_hidden
            global threads
            mapv = mapview
            locationsl = locationslayer
            dronesl = droneslayer

            if location_hidden == True:
                location_hidden = False
                locationsl.draw_locations()
            elif location_hidden == False:
                location_hidden = True
                for child in mapv.children:
                    if type(child) is MarkerMapLayer:
                        #Remove all marker widgets and add back in Drone markers i
f not hidden
                        child.clear_widgets()
                        if drone_hidden == False:
                            threads.append(Thread(target = dronesl.draw_drones, ar
gs = (dronesl, locationsl), daemon = True))
                            threads[len(threads)-1].start()
                        break

        #Used to draw drone markers on screen
        def show_drones_button(mapview, droneslayer, locationslayer):
            global drone_hidden
            global location_hidden
            global threads
            mapv = mapview
            dronesl = droneslayer
            locationsl = locationslayer

            if drone_hidden == True:
                drone_hidden = False
                threads.append(Thread(target = dronesl.draw_drones, args = (drones
l, locationsl), daemon = True))
                threads[len(threads)-1].start()
            elif drone_hidden == False:
                drone_hidden = True
```

```
                for child in mapv.children:
                    if type(child) is MarkerMapLayer:
                        #Remove all marker widgets and add back in Location marker
s if not hidden
                        child.clear_widgets()
                        if location_hidden == False:
                            locationsl.draw_locations()
                        break

###Class related to CRUD location operations
class LocationMarkerLayer(MapLayer):
    """
    Class for adding Location and Location Markers
    """
    locationview = None

    def __init__(self, **kwargs):
        super(LocationMarkerLayer, self).__init__(**kwargs)
        self.zoom = 0
        locationview = self.parent

    def on_touch_down(self, touch):
        """
        Adds functionality to kivy's on_touch_down() event, specificly for adding
new Locations
        """
        global new_location_add
        global location_layer

        if new_location_add == True:
            print(touch.pos[0])
            print(touch.pos[1])
            location_layer.add_point(touch.pos[0],touch.pos[1],0)
        return super(LocationMarkerLayer, self).on_touch_down(touch)

    def draw_locations(self, *args):
        """
        Adds location markers to MapLayer
        """
        locationview = self.parent
        self.zoom = locationview.zoom
        global locations

        for location in locations:
            # Draw new Location markers
            locationview.add_marker(MapMarker(lat=location.getlocationlatitude(),
lon=location.getlocationlongitude(), source='images/MapMarker.png'))

    def add_point(self, lat, lon, alt):
        """
        Adds a location to system
        """
        global locations
        global new_location_add
        global name
        if new_location_add == True:
            newlocation = Location(name, self.parent.get_latlon_at(lat,lon)[0], se
lf.parent.get_latlon_at(lat,lon)[1], alt)
            new_location_add = False
```

```
            #: Add a point a chosen coordinates
            print("location to add:")
            print (newlocation.getlocationname())
            locations.append(newlocation)
            self.draw_locations()
            SaveLocationList(locations)


    def set_loc_name(self, nname):
        """
        Set name for new location to be added to system
        """
        global name
        global new_location_add
        name = nname
        new_location_add = True

    def add_point_popup(self):
        """
        Popup used to set name for new location
        """
        mapview = self.parent
        global locations
        global name
        global new_location_add
        self.root = BoxLayout(orientation='vertical',spacing=15)

        nameinput = TextInput(hint_text="name", multiline=False, size_hint=(.9, No
ne), pos_hint ={'center_x': .5,'center_y': 1}, height=40)
        self.root.add_widget(nameinput, index=0)

        self.popup = Popup(title="Add Location", auto_dismiss=True, size_hint=(.5,
.5))
        self.root.add_widget(Button(text="Add Location Name", size_hint=(.9,.7), p
os_hint ={'center_x': .5,'center_y': 1},on_press=lambda a: (self.popup.dismiss(),s
elf.set_loc_name(nameinput.text))))
        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1], size_hint=(.9,.7), pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a
: self.popup.dismiss()))
        self.popup.add_widget(self.root)
        self.popup.open()


    def view_locations_popup(self):
        """
        Popup to display a list of locations on system
        """

        self.root = BoxLayout(orientation='vertical',spacing=15)

        #Scrollable location list
        locationlistview = Label(text= location_list(),size_hint=(1.01, None))
        locationlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.widt
h, Window.height/2),scroll_y=1, do_scroll_x=False, scroll_type=['bars','content'],
 bar_width = 10, pos_hint ={'center_x': .5,'center_y': 1})
        locationlistviewscroll.add_widget(locationlistview)
        locationlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(locationlistviewscroll)
```

```
        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1], size_hint=(.9,.7), pos_hint ={'center_x': .5,'center_y': 1}, on_press=lambda
a: self.popup.dismiss()))
        self.popup = Popup(title="View Location List", auto_dismiss=True, size_hin
t=(.9,.8))
        self.popup.add_widget(self.root)
        self.popup.open()

    def remove_location_popup(self):
        """
        Popup used to remove location from system
        """
        def buttonPressRemoveLocation(selection):
            global locations
            global drone_hidden
            global location_hidden
            global threads
            if int(selection) < 1 or int(selection) > len(locations):
                return

            del locations[int(selection)-1]
            SaveLocationList(locations)


        self.root = BoxLayout(orientation='vertical',spacing=15)

        locationlistview = Label(text= location_list(),size_hint=(1.01, None))
        locationlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.widt
h, Window.height/4),scroll_y=1, do_scroll_x=False, scroll_type=['bars','content'],
 bar_width = 10, pos_hint ={'center_x': .5,'center_y': 1})
        locationlistviewscroll.add_widget(locationlistview)
        locationlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(locationlistviewscroll)

        location = FloatInput(hint_text="Enter Number corresponding to location",
multiline=False, size_hint=(.9, None), pos_hint ={'center_x': .5,'center_y': 1},he
ight=40)
        self.root.add_widget(location)
        self.root.add_widget(Button(text="RemoveLocation", size_hint=(.9,.8), pos_
hint ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressRemoveLocation(
location.text)))

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1], size_hint=(.9,.8), pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a
: self.popup.dismiss()))

        #Add popup widget to root
        self.popup = Popup(title="Remove Location", auto_dismiss=True, size_hint=(
.9,.8))
        self.popup.add_widget(self.root)
        self.popup.open()


class DroneMarkerLayer(MapLayer):
    """
    Class related to CRUD drones operations
    """
    droneview = None
```

```
    def __init__(self, **kwargs):
        super(DroneMarkerLayer, self).__init__(**kwargs)
        self.zoom = 0
        droneview = self.parent

    def add_virtual_drone(self, name, home):
        """
        Adds a virtual drone to the system
        """
        global locations
        if int(home) <= len(locations):
            for location in locations:
                if location.getlocationname() == locations[int(home)-
1].getlocationname():
                    home_loc = location

                    global drones
                    newdrone = VirtualDrone(name, "Virtual", float(home_loc.getloc
ationlatitude()), float(home_loc.getlocationlongitude()), float(home_loc.getlocati
onlatitude()), float(home_loc.getlocationlongitude()), 0, 100, 0, "Grounded", 0)
                    break


            drones.append(newdrone)
            SaveDroneList(drones)

    def add_virtual_drone_popup(self):
        """
        Popup to add virtual drone to the system
        """
        global drones
        global locations
        self.root = BoxLayout(orientation='vertical',spacing=15)
        nameinput = TextInput(hint_text="drone name", multiline=False,size_hint=(.
9, None),pos_hint ={'center_x': .5,'center_y': 1},height=40)
        homeinput = FloatInput(hint_text="home location name", multiline=False,siz
e_hint=(.9, None),pos_hint ={'center_x': .5,'center_y': 1},height=40)

        #Text boxes to add location info
        self.root.add_widget(nameinput, index=0)

        locationlistview = Label(text= location_list(),size_hint=(1, None))

        locationlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.widt
h, Window.height/4), do_scroll_x=False, scroll_type=['bars','content'], bar_width
= 10, pos_hint ={'center_x': .5,'center_y': 1})
        locationlistviewscroll.add_widget(locationlistview)
        locationlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(locationlistviewscroll)

        self.root.add_widget(homeinput, index=0)

        self.root.add_widget(Button(text="AddDrone",size_hint=(.9, .8),pos_hint ={
'center_x': .5,'center_y': 1},on_press=lambda a: self.add_virtual_drone(nameinput.
text, homeinput.text)))
        self.popup = Popup(title="Add Drone", auto_dismiss=True, size_hint=(.9,.8)
)
```

```
        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1],size_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))

        #Add popup widget to root
        self.popup.add_widget(self.root)
        self.popup.open()

    def remove_drone_popup(self):
        """
        Popup the remove a drone from the system
        """

        def buttonPressRemoveDrone(selection):
            """
            Function removes drone when button pressed
            """
            global drones
            if int(selection) < 1 or int(selection) > len(drones):
                return

            del drones[int(selection)-1]
            SaveDroneList(drones)

        self.root = BoxLayout(orientation='vertical',spacing=15)

        #Scrollable Drone list
        dronelistview = Label(text= drone_list(),size_hint=(1.01, None))
        dronelistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
Window.height/4), do_scroll_x=False, scroll_type=['bars','content'], bar_width = 1
0, pos_hint ={'center_x': .5,'center_y': 1})
        dronelistviewscroll.add_widget(dronelistview)
        dronelistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(dronelistviewscroll)

        #Input box
        location = FloatInput(hint_text="Enter Number corresponding to drone", mul
tiline=False, size_hint=(.9,None), pos_hint={'center_x': .5,'center_y': 1},height=
40)
        self.root.add_widget(location)

        self.root.add_widget(Button(text="RemoveDrone", size_hint=(.9,.7), pos_hin
t={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressRemoveDrone(locatio
n.text)))

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1], size_hint=(.9,.7), pos_hint={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))

        #Add popup widget to root
        self.popup = Popup(title="Remove Drone", auto_dismiss=True, size_hint=(.9,
.8))
        self.popup.add_widget(self.root)
        self.popup.open()

    def insert_drone_image(self, drone, *largs):
        """
        Used to choose suitable image for drawing drone marker on screen
        """
```

```
        droneview = self.parent
        if drone.getdronestate() == "Descending" and drone.getbattery() > 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneDescending.png"))
        elif drone.getdronestate() == "Descending" and drone.getbattery() <= 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneDescending50batt.png"))
        elif drone.getdronestate() == "Descending" and drone.getbattery() <= 10:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneDescending10batt.png"))
        elif drone.getdronestate() == "Ascending" and drone.getbattery() > 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneAscending.png"))
        elif drone.getdronestate() == "Ascending" and drone.getbattery() <= 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneAscending50batt.png"))
        elif drone.getdronestate() == "Ascending" and drone.getbattery() <= 10:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/DroneAscending10batt.png"))
        elif drone.getbattery() > 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/Drone.png"))
        elif drone.getbattery() <= 50:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/Drone50batt.png"))
        else:
            droneview.add_marker(MapMarker(lat=drone.getcurrentlatitude(), lon=dro
ne.getcurrentlongitude(), source="images/Drone10batt.png"))

    def draw_drones(self, dronel, locationl):
        """
        Adds drone markers to MapLayer
        """
        dronel = dronel
        locationl = locationl
        droneview = self.parent
        self.zoom = droneview.zoom
        global drones
        global drone_hidden
        global flights

        flightinprogress = False
        for drone in drones:
            if drone.getdronestate() != "Grounded":
                flightinprogress = True
                break
        i = 1 #Used to draw the drones once when not in flight
        while (drone_hidden == False and flightinprogress == True) or i == 1:
            for child in droneview.children:
                if type(child) is MarkerMapLayer:
                    child.clear_widgets()
                    break
            #Draw location markers if they should be seen
            if location_hidden == False:
                Clock.schedule_once(partial(locationl.draw_locations))
            for drone in drones:
                #Draw Drone markers
                Clock.schedule_once(partial(self.insert_drone_image, drone))
```

```
            time.sleep(.05)
            i = i+1


class FlightLayer(MapLayer):
    """
    Class related to flight CRUD operations
    """

    def __init__(self, **kwargs):
        super(FlightLayer, self).__init__(**kwargs)
        self.zoom = 0
        droneview = self.parent

    def view_flights_popup(self):
        """
        Displays list of flights on system
        """

        self.root = BoxLayout(orientation='vertical',spacing=15)

        flightlistview = Label(text= flight_list(),size_hint=(1.01, None))
        flightlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
 Window.height/2),scroll_y=1, scroll_type=['bars','content'], bar_width = 10, pos_
hint ={'center_x': .5,'center_y': 1})
        flightlistviewscroll.add_widget(flightlistview)
        flightlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(flightlistviewscroll)

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1], size_hint=(.9,.7), pos_hint={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))
        self.popup = Popup(title="View Flight List", auto_dismiss=True, size_hint=
(.9,.7))
        self.popup.add_widget(self.root)
        self.popup.open()


    def create_flight_popup(self):
        """
        Popup used to add flights to the system
        """
        mapview = self.parent
        global flights
        global locations
        global drones
        new_flight = Flight("F", 0, 0, 0, 0, "D", False, False)

        def buttonPressAddFlightName(newflight, name):
            newflight.setflightid(name)

        def buttonPressAddDrone(newflight, drones, selection):
            for drone in drones:
                if drone.drone_id == drones[int(selection)-1].drone_id:
                    newflight.setdrone(drone.drone_id)
                    newflight.setstartlatitude(drone.gethomelatitude())
                    newflight.setstartlongitude(drone.gethomelongitude())

        def buttonPressAddDestination(newflight, locations, selection):
```

```
            for location in locations:
                if location.getlocationname() == locations[int(selection)-
1].getlocationname():
                    newflight.setdestinationlatitude(location.getlocationlatitude(
))
                    newflight.setdestinationlongitude(location.getlocationlongitud
e())
        #Adds Flight to system Flight list
        def buttonPressAddFlight(newflight, flights):
            flights.append(newflight)
            SaveFlightList(flights)
            self.popup.dismiss()


        self.root = BoxLayout(orientation='vertical', spacing =5)

        flightname = TextInput(hint_text="Flight Name", multiline=False,size_hint=
(.9, .8),pos_hint ={'center_x': .5,'center_y': 1})
        self.root.add_widget(flightname)

        #Scrollable Drone list
        dronelistview = Label(text= drone_list(),size_hint=(1.01, None))
        dronelistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
Window.height/8), do_scroll_x=False, scroll_type=['bars','content'], bar_width = 1
0, pos_hint ={'center_x': .5,'center_y': 1})
        dronelistviewscroll.add_widget(dronelistview)
        dronelistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(dronelistviewscroll)

        drone = FloatInput(hint_text="Drone choice", multiline=False,size_hint=(.9
, .8),pos_hint ={'center_x': .5,'center_y': 1})
        self.root.add_widget(drone)

        locationlistview = Label(text= location_list(),size_hint=(1.01, None))
        locationlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.widt
h, Window.height/8), do_scroll_x=False,scroll_y=1, scroll_type=['bars','content'],
 bar_width = 10, pos_hint ={'center_x': .5,'center_y': 1})
        locationlistviewscroll.add_widget(locationlistview)
        locationlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(locationlistviewscroll)

        location = FloatInput(hint_text="Destination Choice", multiline=False,size
_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1})
        self.root.add_widget(location)

        self.root.add_widget(Button(text="AddFlightName",size_hint=(.9, .8),pos_hi
nt ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressAddFlightName(new
_flight, flightname.text)))
        self.root.add_widget(Button(text="AddDrone",size_hint=(.9, .8),pos_hint ={
'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressAddDrone(new_flight, d
rones, drone.text)))
        self.root.add_widget(Button(text="AddDestination",size_hint=(.9, .8),pos_h
int ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressAddDestination(n
ew_flight, locations, location.text)))
        self.root.add_widget(Button(text="AddFlight",size_hint=(.9, .8),pos_hint =
{'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressAddFlight(new_flight,
 flights)))
```

```
        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1],size_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))
        self.popup = Popup(title="Add Flight", auto_dismiss=True, size_hint=(.9,.8
))
        self.popup.add_widget(self.root)
        self.popup.open()

    def remove_flight_popup(self):
        """
        Remove flight popup
        """

        def buttonPressRemoveFlight(selection):
            """
            Deletes flight from system on button press
            """
            global flights
            if int(selection) <= len(flights):
                del flights[int(selection)-1]
                SaveFlightList(flights)

        self.root = BoxLayout(orientation='vertical',spacing=15)

        flightlistview = Label(text= flight_list(),size_hint=(1.01, None))
        flightlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
 Window.height/4),scroll_y=1, scroll_type=['bars','content'], bar_width = 10, pos_
hint ={'center_x': .5,'center_y': 1})
        flightlistviewscroll.add_widget(flightlistview)
        flightlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(flightlistviewscroll)

        flight = FloatInput(hint_text="Enter Number corresponding to flight", mult
iline=False, size_hint=(.9, None), pos_hint ={'center_x': .5,'center_y': 1}, heigh
t = 40)
        self.root.add_widget(flight)
        self.root.add_widget(Button(text="CancelFlight",size_hint=(.9, .8),pos_hin
t ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressRemoveFlight(fligh
t.text)))

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1],size_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))
        self.popup = Popup(title="Cancel Flight", auto_dismiss=True, size_hint=(.9
,.8))
        self.popup.add_widget(self.root)
        self.popup.open()


    def abort_flight_popup(self):
        """
        Abort flight popup
        """

        def buttonPressAbortFlight(selection):
            """
            Set flight parameter flight_abort to True, this will be checked
            by the DroneController.MoveDrone() function and the drone will be redi
rected
```

```
            """
            global flights
            flights[int(selection)-1].setflightabort(True)
            SaveFlightList(flights)

        self.root = BoxLayout(orientation='vertical',spacing=15)

        #Scrollable list of flights
        flightlistview = Label(text= flight_list(),size_hint=(1.01, None))
        flightlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
 Window.height/4),scroll_y=1, scroll_type=['bars','content'], bar_width = 10, pos_
hint ={'center_x': .5,'center_y': 1})
        flightlistviewscroll.add_widget(flightlistview)
        flightlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(flightlistviewscroll)

        #Input Box
        flight = FloatInput(hint_text="Enter Number corresponding to flight", mult
iline=False, size_hint=(.9, None), pos_hint ={'center_x': .5,'center_y': 1},height
=40)
        self.root.add_widget(flight)

        self.root.add_widget(Button(text="AbortFlight",size_hint=(.9, .8),pos_hint
 ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressAbortFlight(flight.
text)))

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1],size_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))
        self.popup = Popup(title="Abort Flight", auto_dismiss=True, size_hint=(.9,
.8))
        self.popup.add_widget(self.root)
        self.popup.open()

    def start_flight_popup(self):
        """
        Begin a flight popup
        """

        def buttonPressBeginFlight(selection):
            """
            Begins flight on button press
            """
            global flights
            global locations
            global drones
            global threads

            for drone in drones:
                if drone.drone_id == flights[int(selection)-1].getdrone():
                    flight_drone = drone
            threads.append(Thread(target = TakeOffFlyLand,  args = (flight_drone,
flights[int(selection)-1], drones, locations), daemon = True))
            threads[len(threads)-1].start()

        self.root = BoxLayout(orientation='vertical',spacing=15)

        flightlistview = Label(text= flight_list(),size_hint=(1.01, None))
```

```
        flightlistviewscroll = ScrollView(size_hint=(1, None), size=(Window.width,
 Window.height/4),scroll_y=1, scroll_type=['bars','content'], bar_width = 10, pos_
hint ={'center_x': .5,'center_y': 1})
        flightlistviewscroll.add_widget(flightlistview)
        flightlistviewscroll.effect_cls.spring_constant=0
        self.root.add_widget(flightlistviewscroll)

        flight = FloatInput(hint_text="Enter Number corresponding to flight", mult
iline=False, size_hint=(.9, None), pos_hint ={'center_x': .5,'center_y': 1}, heigh
t = 40)
        self.root.add_widget(flight)
        self.root.add_widget(Button(text="BeginFlight",size_hint=(.9, .8),pos_hint
 ={'center_x': .5,'center_y': 1},on_press=lambda a: buttonPressBeginFlight(flight.
text)))

        self.root.add_widget(Button(text="Close window", background_color = [1,0,0
,1],size_hint=(.9, .8),pos_hint ={'center_x': .5,'center_y': 1},on_press=lambda a:
 self.popup.dismiss()))
        self.popup = Popup(title="Start Flight", auto_dismiss=True, size_hint=(.9,
.8))
        self.popup.add_widget(self.root)
        self.popup.open()

class LineMapLayer(MapLayer):
    """
    Class related to displaying lines between destinations !Currently incomplete
    """

    def __init__(self, **kwargs):
        super(LineMapLayer, self).__init__(**kwargs)
        self.zoom = 0
        mapview = self.parent
        global coordinates


    def reposition(self):
        mapview = self.parent
        global corridors

        #: Must redraw when the zoom changes
        #: as the scatter transform resets for the new tiles
        if (self.zoom != mapview.zoom):
            self.draw_line()

    def create_corridor_popup(self):
        mapview = self.parent
        global coordinates
        global locations
        global corridors
        corridortoadd = []

        def buttonPressAddLocation(coord, corr, corridortoadd, selection):
            corridortoadd.append(coord[int(selection)-1])
            print(coord[int(selection)-1])
            print(corridortoadd)

        def buttonPressAddCorridor(corr):
            global corridors
            corridors.append(corridortoadd)
```

```
            #self.draw_line(corr)
            self.draw_line()
            print("Full corridors list:")
            print(corridors)


        self.root = BoxLayout(orientation='vertical', spacing =5)

        self.root.add_widget(Label(text= self.location_list()))

        location = TextInput(hint_text="Enter Number corresponding to flight", mul
tiline=False)
        self.root.add_widget(location, index=0)

        self.root.add_widget(Label(text= str(corridortoadd)))

        self.root.add_widget(Button(text="AddLocation",on_press=lambda a: buttonPr
essAddLocation(coordinates, corridors, corridortoadd, location.text)))
        self.root.add_widget(Button(text="AddCorridor",on_press=lambda a: buttonPr
essAddCorridor(corridortoadd)))

        self.root.add_widget(Button(text="Close window",on_press=lambda a: self.po
pup.dismiss()))
        self.popup = Popup(title="Add Corridor", auto_dismiss=True, size_hint=(.5,
.5))
        self.popup.add_widget(self.root)
        self.popup.open()



    def get_x(self, lon):
        """Get the x position on the map using this map source's projection
        (0, 0) is located at the top left.
        """
        return clamp(lon, MIN_LONGITUDE, MAX_LONGITUDE)

    def get_y(self, lat):
        """Get the y position on the map using this map source's projection
        (0, 0) is located at the top left.
        """
        lat = clamp(-lat, MIN_LATITUDE, MAX_LATITUDE)
        lat = lat * pi / 180.
        return ((1.0 - log(tan(lat) + 1.0 / cos(lat)) / pi))

    def draw_line(self, *args):
        mapview = self.parent
        self.zoom = mapview.zoom
        global coordinates
        global corridors
        #corridor = corr

        # When zooming we must undo the current scatter transform
        # or the animation distorts it
        scatter = mapview._scatter
        map_source = mapview.map_source
        sx,sy,ss = scatter.x, scatter.y, scatter.scale
        vx,vy,vs = mapview.viewport_pos[0], mapview.viewport_pos[1], mapview.scale

        # Account for map source tile size and mapview zoom
```

```
        ms = pow(2.0,mapview.zoom) * map_source.dp_tile_size
        #: Since lat is not a linear transform we must compute manually

        #line_points = []
        with self.canvas:
                # Clear old line
                self.canvas.clear()

        for corridor in corridors:
            line_points = []
            for lat,lon in corridor:
                line_points.extend((self.get_x(lon),self.get_y(lat)))
                print("linepointslist:")
                print (line_points)
                #line_points.extend(mapview.get_window_xy_from(lat,lon,mapview.zoo
m))


            with self.canvas:
                # Clear old line
                #self.canvas.clear()
                # Undo the scatter animation transform
                Scale(1/ss,1/ss,1)
                Translate(-sx,-sy)

                # Apply the get window xy from transforms
                Scale(vs,vs,1)
                Translate(-vx,-vy)

                # Apply the what we can factor out
                # of the mapsource long,lat to x,y conversion
                Scale(ms/360.0,ms/2.0,1)
                Translate(180,0)

                # Draw new
                Color(1, 0, 0, 1)
                print("linepointslist Line:")
                print (line_points)
                Line(points=line_points, width=1)#4/ms)#, joint="round",joint_prec
ision=100)
                self.canvas.ask_update()

if __name__ == "__main__":
    MapViewApp().run()
```

AbstractDone.py

```
# -*- coding: UTF-8 -*-


#Name:        James Hall
#Student No.: C00007006
#Institute:   Institute of Technology Carlow
#Project:     Drone Traffic Control System
#Date:        April 2021
#License:     GNU Affero General Public License v3.0

#Abstract Drone class
```

```python
from abc import ABC, abstractmethod

class AbstractDrone(ABC):
    """
    Abstract class for drone types
    """
    home_latitude = 0
    home_longtitude = 0
    current_latitude = 0
    current_longtitude = 0
    destination_latitude = 0
    destination_longtitude = 0
    altitude = 0
    battery = 100
    speed = 0
    flying_state = "Grounded"
    drone_id = 0
    drone_type = "virtual"
    heading = 0

    @abstractmethod
    def takeoff(self):
        pass

    @abstractmethod
    def hover(self):
        pass

    @abstractmethod
    def moveto(self):
        pass

    @abstractmethod
    def changealtitude(self):
        pass

    @abstractmethod
    def land(self):
        pass

    @abstractmethod
    def getgpsposition(self):
        pass

    @abstractmethod
    def setdronestate(self, state):
        pass

    @abstractmethod
    def getdronestate(self):
        pass

    @abstractmethod
    def setbatteryinflight(self):
        pass

    @abstractmethod
    def setbatterytotal(self, percent):
```

```
        pass

    @abstractmethod
    def setbatterychange(self, percent):
        pass

    @abstractmethod
    def getbattery(self):
        pass

    @abstractmethod
    def sethomelatitude(self,lat):
        pass

    @abstractmethod
    def gethomelatitude(self):
        pass

    @abstractmethod
    def setdestinationlatitude(self,lat):
        pass

    @abstractmethod
    def getdestinationlatitude(self):
        pass

    @abstractmethod
    def setdestinationlongitude(self,lon):
        pass

    @abstractmethod
    def getdestinationlongitude(self):
        pass

    @abstractmethod
    def setcurrentlatitude(self,lat):
        pass

    @abstractmethod
    def getcurrentlatitude(self):
        pass

    @abstractmethod
    def setcurrentlongitude(self,lon):
        pass

    @abstractmethod
    def getcurrentlongitude(self):
        pass

    @abstractmethod
    def getdistancetodestination(self):
        pass

    @abstractmethod
    def getdistancetoposition(self):
        pass

    @abstractmethod
```

```
    def createvector(self):
        pass

    @abstractmethod
    def getvectorlatitude(self):
        pass

    @abstractmethod
    def getvectorlongitude(self):
        pass

    @abstractmethod
    def updatecurrentposition(self):
        pass

    @abstractmethod
    def getbearing(self, startlat, startlong, destlat, destlong):
        pass

    @abstractmethod
    def getdronetype(self):
        pass

    @abstractmethod
    def setdronetype(self, dronetype):
        pass
```

VirtualDrone.py

```
# -*- coding: UTF-8 -*-


#Name:        James Hall
#Student No.: C00007006
#Institute:   Institute of Technology Carlow
#Project:     Drone Traffic Control System
#Date:        April 2021
#License:     GNU Affero General Public License v3.0


import math
import time
import threading
from geographiclib.geodesic import Geodesic
from threading import Thread
from AbstractDrone import AbstractDrone


battery_drain = .055

class VirtualDrone(AbstractDrone):
    """
    Virtual Drone class
    """
    def __init__(self, drone_id, dronetype, home_latitude, home_longitude, destina
tion_latitude, destination_longitude, altitude, battery, speed, flying_state, head
ing):
```

```
        self.home_latitude = home_latitude
        self.home_longitude = home_longitude
        self.current_latitude = home_latitude
        self.current_longitude = home_longitude
        self.destination_latitude = destination_latitude
        self.destination_longitude = destination_longitude
        self.altitude = altitude
        self.battery = battery
        self.speed = speed
        self.flying_state = flying_state
        self.drone_id = drone_id
        self.drone_type = dronetype
        self.heading = heading

    def takeoff(self):
        self.altitude = 2
        self.flying_state = "Hovering"

    def hover(self):
        self.flying_state = "Hovering"

    def moveto(self, longitude, latitude, altitude, spe):
        self.altitude = altitude
        self.destination_latitude = latitude
        self.destination_longitude = longitude
        self.speed = spe
        self.createvector()#better name?
        self.updatecurrentposition()
        self.flying_state = "MovingTo"
        self.setbatteryinflight()

    def changealtitude(self, alt):
        self.altitude = alt
        #self.flying_state = "Changing Altitude"

    def land(self):
        self.altitude = 0
        self.flying_state = "Grounded"

    def getgpsposition(self):
        name = self.drone_id
        lat = float(self.current_latitude)
        lon = float(self.current_longitude)
        alt = self.altitude

        return name, lat, lon, alt

    def getaltitude(self):
        return self.altitude

    def setdronestate(self, state):
        self.flying_state = state

    def getdronestate(self):
        flystate = self.flying_state

        return flystate

    #Set/Get Battery
```

```
    def setbatteryinflight(self):
        global battery_drain
        # .055 is estimated drain on battery in flight per second as a %
        self.battery = (self.battery - battery_drain)

    def setbatterytotal(self, percent):
        self.battery = percent

    def setbatterychange(self, percent):
        self.battery = self.battery - percent

    def getbattery(self):
        return self.battery

    #Set/Get home lat and lon
    def sethomelatitude(self,lat):
        self.home_latitude = lat

    def gethomelatitude(self):
        return self.home_latitude

    def sethomelongitude(self,lon):
        self.home_longitude = lon

    def gethomelongitude(self):
        return self.home_longitude

    #Set/Get destination lat and lon
    def setdestinationlatitude(self,lat):
        self.destination_latitude = lat

    def getdestinationlatitude(self):
        return self.destination_latitude

    def setdestinationlongitude(self,lon):
        self.destination_longitude = lon

    def getdestinationlongitude(self):
        return self.destination_longitude

    #Set/Get current lat and lon
    def setcurrentlatitude(self,lat):
        self.current_latitude = lat

    def getcurrentlatitude(self):
        return float(self.current_latitude)

    def setcurrentlongitude(self,lon):
        self.current_longitude = lon

    def getcurrentlongitude(self):
        return float(self.current_longitude)

    def getdistancetodestination(self):
        #Get travel distance from point to point
        #Formula adopted from top answer and source of answer:
        #https://stackoverflow.com/questions/27928/calculate-distance-between-two-
latitude-longitude-points-haversine-formula
        #http://www.movable-type.co.uk/scripts/latlong.html
```

```
        radius = 6371 #Radius in km of Earth

        deg_latitude = math.radians(self.destination_latitude -
float(self.current_latitude))
        deg_longitude = math.radians(self.destination_longitude -
float(self.current_longitude))

        #a = Square of half the chord length between two points
        a = math.sin(deg_latitude/2) * math.sin(deg_latitude/2) + math.cos(math.ra
dians(self.current_latitude)) * math.cos(math.radians(self.destination_latitude))
* math.sin(deg_longitude/2) * math.sin(deg_longitude/2)
        #c = Angular distance in radians
        c = 2 * math.atan2(math.sqrt(a),math.sqrt(1-a))
        d = radius * c

        return d

    def getdistancetoposition(self,poslat,poslon):
        #Get travel distance from point to point
        #Formula adopted from top answer and source of answer:
        #https://stackoverflow.com/questions/27928/calculate-distance-between-two-
latitude-longitude-points-haversine-formula
        #http://www.movable-type.co.uk/scripts/latlong.html

        radius = 6371 #Average Radius in km of Earth

        deg_latitude = math.radians(poslat - float(self.current_latitude))
        deg_longitude = math.radians(poslon - float(self.current_longitude))

        #a = Square of half the chord length between two points
        a = math.sin(deg_latitude/2) * math.sin(deg_latitude/2) + math.cos(math.ra
dians(float(self.current_latitude))) * math.cos(math.radians(float(self.destinatio
n_latitude))) * math.sin(deg_longitude/2) * math.sin(deg_longitude/2)
        #c = Angular distance in radians
        c = 2 * math.atan2(math.sqrt(a),math.sqrt(1-a))
        d = radius * c

        return d

    def createvector(self):
        self.vectorlat = float(self.destination_latitude) -
float(self.current_latitude)
        self.vectorlon = float(self.destination_longitude) -
float(self.current_longitude)

        return self.vectorlat, self.vectorlon

    def getvectorlatitude(self):
        vectorlat = float(self.destination_latitude) - self.home_latitude

        return vectorlat

    def getvectorlongitude(self):
        vectorlon = float(self.destination_longitude )- self.home_longitude

        return vectorlon

    def updatecurrentposition(self):
```

```
        direction_length = math.sqrt(math.pow(self.getvectorlatitude(),2) + math.p
ow(self.getvectorlongitude(),2))
        normalised_vector_lat = self.getvectorlatitude() / direction_length
        normalised_vector_lon = self.getvectorlongitude() / direction_length
        self.current_latitude=float(self.current_latitude)+ (normalised_vector_lat
*(self.speed*1))
        self.current_longitude =float(self.current_longitude) + (normalised_vector
_lon*(self.speed*1))


    def getbearing(self, startlat, startlong, destlat, destlong):
        heading = Geodesic.WGS84.Inverse(startlat,startlong, destlat, destlong)["a
zi1"]
        return heading

    def getdronetype(self):
        return self.drone_type

    def setdronetype(self, dronetype):
        self.drone_type = dronetype
```

BebopDrone.py

```
# -*- coding: UTF-8 -*-


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0
"""
Bebop Drone class
"""

import olympe
from olympe.messages.ardrone3.Piloting import TakeOff, moveBy, moveTo, Landing
from olympe.messages.ardrone3.PilotingState import GpsLocationChanged, FlyingState
Changed, moveToChanged, PositionChanged

DRONE_IP = "10.202.0.1"
def getgpsposition(drone):
    lat = drone.get_state(PositionChanged)["latitude"]
    lon = drone.get_state(PositionChanged)["longitude"]
    alt = drone.get_state(PositionChanged)["altitude"]

    print("Latitude:", lon)
    print("Longitude:", lat)
    print("Altitude:", alt)

def takeoff(drone):
    drone(TakeOff()
    >> FlyingStateChanged(state="hovering", _timeout=5)).wait()
    getgpsposition(drone)
```

```
def moveto(drone, longt, lati, alti):
    drone(moveTo(longt,lati,alti,0,0)
    >> FlyingStateChanged(state="hovering", _timeout=5)).wait()
    getgpsposition(drone)

def changealtitude(drone, alt):
    lat = drone.get_state(PositionChanged)["latitude"]
    lon = drone.get_state(PositionChanged)["longitude"]

    drone(moveTo(lon,lat,alt,0,0)
    >> FlyingStateChanged(state="hovering", _timeout=5)).wait()
    getgpsposition(drone)

def land(drone):
    assert drone(Landing()).wait().success()
    getgpsposition(drone)


if __name__ == "__main__":
    with olympe.Drone(DRONE_IP,) as drone:
        drone.connect()

        #drone.FlyingStateChanged(state="hovering", _policy="check")
        #TakeOffLocation = drone.get_state(GpsLocationChanged)

        takeoff(drone)
        TakeOffLocation = drone.get_state(GpsLocationChanged)

        getgpsposition(drone)
        moveto(drone, .00001, 0, 1)
        getgpsposition(drone)
        changealtitude(drone, 5)
        getgpsposition(drone)
        moveto(drone, TakeOffLocation["latitude"], TakeOffLocation["longitude"], 1
)
        getgpsposition(drone)
        land(drone)

        # Leaving the with statement scope: implicit drone.disconnect() but that
        # is still a good idea to perform the drone disconnection explicitly
        drone.disconnect()
```

DroneController.py

```
# -*- coding: UTF-8 -*-


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0
"""
Drone Controller
==========================
Contains all drone control algorithms
```

```
"""

import pickle
import math
import time
import threading
import multiprocessing
from threading import Thread, Lock
from AbstractDrone import AbstractDrone
from VirtualDrone import VirtualDrone
from Location import Location
from Flight import Flight
from save_load import *

#Global thread variables used to kill threads
takeoffThread = False
moveThread = False
altitudeThread = False
hoverThread = False
landThread = False
data_lock = Lock()

def MoveDrone(drone, flight, dronelist, lat, lon, locations):
    global moveThread
    drone.destination_latitude = lat
    drone.destination_longitude = lon
    #Following variable needed to know when to stop
    distancetodest = getdistancetodestination(drone, flight)

    #While destination latitude and longitude not reached and distance from origin
 to destination not exceeded
    while ((drone.getdistancetoposition(float(drone.home_latitude), float(drone.ho
me_longitude)) <= distancetodest)):
        canmove = True
        for d in dronelist:
            if (CollisionDetection(drone, d) and d.getdronestate() != "Grounded" a
nd flight.flight_drone != d.drone_id and ((int(drone.getbattery()) < int(d.getbatt
ery())) or (float(drone.gethomelatitude())>float(d.gethomelatitude()) and float(dr
one.gethomelongitude())>float(d.gethomelongitude())))):
                canmove = False
                break
        if(canmove and flight.getflightabort() == False):
            drone.moveto(flight.getdestinationlongitude(), flight.getdestinationla
titude(), drone.getaltitude(), .0001)
            time.sleep(1)
        elif(canmove and flight.getflightabort() == True):
            newlat, newlon = findnearestlocation(drone, locations)
            drone.sethomelatitude(drone.getcurrentlatitude())
            drone.sethomelongitude(drone.getcurrentlongitude())
            flight.setdestinationlatitude(newlat)
            drone.setdestinationlatitude(newlat)
            flight.setdestinationlongitude(newlon)
            drone.setdestinationlongitude(newlon)
            drone.moveto(flight.getdestinationlongitude(), flight.getdestinationla
titude(), drone.getaltitude(), .0001)
            distancetodest = getdistancetodestination(drone, flight)
            flight.setflightabort(False)
            time.sleep(1)
        else:
```

```
            ChangeAltitude(drone,drone.getaltitude()-10)
            for i in range(10):
                Hover(drone)
            ChangeAltitude(drone,drone.getaltitude()+10)
            time.sleep(1)
    drone.sethomelatitude(float(drone.getcurrentlatitude()))
    drone.sethomelongitude(float(drone.getcurrentlongitude()))

def ChangeAltitude(drone,newalt):
    global altitudeThread
    while drone.getaltitude() != newalt:
        if (newalt < drone.getaltitude()):
            drone.setdronestate("Descending")
            drone.changealtitude(drone.getaltitude() - 1)
            drone.setbatterychange(.225)
            time.sleep(.5)
        elif (newalt > drone.getaltitude()):
            drone.setdronestate("Ascending")
            drone.changealtitude(drone.getaltitude() + 1)
            drone.setbatterychange(.225)
            time.sleep(.5)
    #Hover when change complete
    '''while altitudeThread:
        drone.hover()
        drone.setbatteryinflight()
        time.sleep(1)'''

def Hover(Drone):
    global hoverThread
    global takeoffThread
    while hoverThread or takeoffThread:
        Drone.hover()
        Drone.setbatteryinflight()
        time.sleep(1)

#Function to land the drone
def LandDrone(Drone):
    #global landThread
    #while Drone.getaltitude() != 0:
    #    Drone.changealtitude(Drone.getaltitude() - 1)
    #    Drone.setbatterychange(.225)
    #    time.sleep(.5)
    ChangeAltitude(Drone,0)
    print("Drone Landed: ", Drone.drone_id)
    Drone.setdronestate("Grounded")

def getdistancetodestination(drone, flight):
        #Get travel distance from point to point
        #Formula adopted from top answer and source of answer:
        #https://stackoverflow.com/questions/27928/calculate-distance-between-two-
latitude-longitude-points-haversine-formula
        #http://www.movable-type.co.uk/scripts/latlong.html

        radius = 6371 #Radius in km of Earth

        deg_latitude = math.radians(float(flight.getdestinationlatitude()) - float
(drone.current_latitude))
        deg_longitude = math.radians(float(flight.getdestinationlongitude()) - flo
at(drone.current_longitude))
```

```
        #a = Square of half the chord length between two points
        a = math.sin(deg_latitude/2) * math.sin(deg_latitude/2) + math.cos(math.ra
dians(float(drone.current_latitude))) * math.cos(math.radians(float(flight.getdest
inationlatitude()))) * math.sin(deg_longitude/2) * math.sin(deg_longitude/2)
        #c = Angular distance in radians
        c = 2 * math.atan2(math.sqrt(a),math.sqrt(1-a))
        d = radius * c

        return d

def getdistancetoposition(drone,poslat,poslon):
        #Get travel distance from point to point
        #Formula adopted from top answer and source of answer:
        #https://stackoverflow.com/questions/27928/calculate-distance-between-two-
latitude-longitude-points-haversine-formula
        #http://www.movable-type.co.uk/scripts/latlong.html

        radius = 6371 #Average Radius in km of Earth

        deg_latitude = math.radians(poslat - float(drone.current_latitude))
        deg_longitude = math.radians(poslon - float(drone.current_longitude))

        #a = Square of half the chord length between two points
        a = math.sin(deg_latitude/2) * math.sin(deg_latitude/2) + math.cos(math.ra
dians(float(drone.current_latitude))) * math.cos(math.radians(float(drone.destinat
ion_latitude))) * math.sin(deg_longitude/2) * math.sin(deg_longitude/2)
        #c = Angular distance in radians
        c = 2 * math.atan2(math.sqrt(a),math.sqrt(1-a))
        d = radius * c

        return d

def getdistancetodrone(drone,otherdrone):
        #Get travel distance from point to point
        #Formula adopted from top answer and source of answer:
        #https://stackoverflow.com/questions/27928/calculate-distance-between-two-
latitude-longitude-points-haversine-formula
        #http://www.movable-type.co.uk/scripts/latlong.html

        radius = 6371 #Average Radius in km of Earth

        deg_latitude = math.radians(float(otherdrone.current_latitude) - float(dro
ne.current_latitude))
        deg_longitude = math.radians(float(otherdrone.current_longitude) - float(d
rone.current_longitude))

        #a = Square of half the chord length between two points
        a = math.sin(deg_latitude/2) * math.sin(deg_latitude/2) + math.cos(math.ra
dians(float(drone.current_latitude))) * math.cos(math.radians(float(otherdrone.cur
rent_latitude))) * math.sin(deg_longitude/2) * math.sin(deg_longitude/2)
        #c = Angular distance in radians
        c = 2 * math.atan2(math.sqrt(a),math.sqrt(1-a))
        d = radius * c

        return d

def TakeOffFlyLand(drone, flight, dronelist, locationlist):
    drone.takeoff()
```

```
    destlat = flight.getdestinationlatitude()
    destlong = flight.getdestinationlongitude()

    ChangeAltitude(drone, 50)
    MoveDrone(drone, flight, dronelist, destlat, destlong, locationlist)
    LandDrone(drone)
    print(drone.drone_id, 'complete:', flight.getflightcomplete())
    flight.setflightcomplete(True)
    print(drone.drone_id, 'complete:', flight.getflightcomplete())
    with data_lock:
        flights = LoadFlightList()
        for f in flights:
            if f.getflightid() == flight.getflightid():
                print(f.getflightid())
                print(flight.getflightid())
                f.setflightcomplete(True)
                SaveDroneList(dronelist)
                SaveFlightList(flights)
                print('complete:', f.getflightcomplete())
                break

def CollisionDetection(drone, otherdrone):
    if ((getdistancetodrone(drone,otherdrone)<=.03) and (abs(drone.getaltitude()-
otherdrone.getaltitude())<=9)):
        return True
    else:
        return False

def findnearestlocation(drone, locations):

    newlocationlat = drone.getdestinationlatitude()
    newlocationlon = drone.getdestinationlongitude()

    for location in locations:
        if (getdistancetoposition(drone,float(location.getlocationlatitude()),floa
t(location.getlocationlongitude())) < getdistancetoposition(drone,float(newlocatio
nlat),float(newlocationlon))):
            newlocationlat = location.getlocationlatitude()
            newlocationlon = location.getlocationlongitude()

    return newlocationlat, newlocationlon
```

Flight.py

```
# -*- coding: UTF-8 -*-


#Name:        James Hall
#Student No.: C00007006
#Institute:   Institute of Technology Carlow
#Project:     Drone Traffic Control System
#Date:        April 2021
#License:     GNU Affero General Public License v3.0

#Flight class
```

```
class Flight():
    """
    Class for flight object
    """
    def __init__(self, name, startlatitude, startlongitude, destlatitude, destlong
itude, drone, flightcomplete, flightabort):
        self.flight_id = name
        self.start_latitude = startlatitude
        self.start_longitude = startlongitude
        self.destination_latitude = destlatitude
        self.destination_longitude = destlongitude
        self.flight_drone = drone
        self.flight_complete = flightcomplete
        self.flight_abort = flightabort

    def setflightid(self, name):
        self.flight_id = name

    def getflightid(self):
        name = self.flight_id
        return name

    def setstartlatitude(self, startlatitude):
        self.start_latitude = startlatitude

    def getstartlatitude(self):
        startlatitude = self.start_latitude
        return startlatitude

    def setstartlongitude(self, startlongitude):
        self.start_longitude = startlongitude

    def getstartlongitude(self):
        startlongitude = self.start_longitude
        return startlongitude

    def setdestinationlatitude(self, destlatitude):
        self.destination_latitude = destlatitude

    def getdestinationlatitude(self):
        destlatitude = self.destination_latitude
        return destlatitude

    def setdestinationlongitude(self, destlongitude):
        self.destination_longitude = destlongitude

    def getdestinationlongitude(self):
        destlongitude = self.destination_longitude
        return destlongitude

    def setdrone(self, drone):
        self.flight_drone = drone

    def getdrone(self):
        drone = self.flight_drone
        return drone
```

```
    def setflightcomplete(self, flightcomplete):
        self.flight_complete = flightcomplete

    def getflightcomplete(self):
        flightcomplete = self.flight_complete
        return flightcomplete

    def setflightabort(self, flightabort):
        self.flight_abort = flightabort

    def getflightabort(self):
        flightabort = self.flight_abort
        return flightabort
```

Location.py

```
# -*- coding: UTF-8 -*-

#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0

#Location class


class Location():
    """
    Class for location object.
    """
    def __init__(self, name, latitude, longitude, altitude):
        self.location_name = name
        self.location_latitude = latitude
        self.location_longitude = longitude
        self.location_altitude = altitude

    def setlocationname(self, name):
        self.location_name = name

    def getlocationname(self):
        name = self.location_name
        return name

    def setlocationlatitude(self, latitude):
        self.location_latitude = latitude

    def getlocationlatitude(self):
        latitude = self.location_latitude
        return latitude

    def setlocationlongitude(self, longitude):
        self.location_longitude = longitude

    def getlocationlongitude(self):
        longitude = self.location_longitude
        return longitude
```

```
    def setlocationaltitude(self, altitude):
        self.location_altitude = altitude

    def getlocationaltitude(self):
        altitude = self.location_altitude
        return altitude
```

PrintLists.py

```
# -*- coding: UTF-8 -*-


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0
"""
Used to store functions for printing drone/location/flight lists
"""

from save_load import *

def flight_list():
    flights = LoadFlightList()
    string = ""
    number = 1

    for flight in flights:
        string += str(number)
        string += ": "
        string += flight.getflightid()
        string += "  Start Lat/Lon:"
        string += str(round(flight.getstartlatitude(), 4))
        string += "/"
        string += str(round(flight.getstartlongitude(), 4))
        string += "  End Lat/Lon:"
        string += str(round(flight.getdestinationlatitude(), 4))
        string += "/"
        string += str(round(flight.getdestinationlongitude(), 4))
        string += " Drone:"
        string += flight.getdrone()
        string += " Complete:"
        string += str(flight.getflightcomplete())
        string += "\n"
        number += 1

    return string

def location_list():
    locations = LoadLocationList()
    string = ""
    number = 1

    for location in locations:
        string += str(number)
```

```
        string += ": "
        string += location.getlocationname()
        string += " Lat:"
        string += str(round(location.getlocationlatitude(),4))
        string += " Lon:"
        string += str(round(location.getlocationlongitude(),4))
        string += "\n"
        number += 1

    return string

def drone_list():
    drones = LoadDroneList()
    string = ""
    number = 1

    for drone in drones:
        string += str(number)
        string += ": Name:"
        string += drone.drone_id
        string += " Lat:"
        string += str(round(drone.gethomelatitude(),4))
        string += " Lon:"
        string += str(round(drone.gethomelongitude(),4))
        string += "\n"
        number += 1

    return string
```

save_load.py

```
# -*- coding: UTF-8 -*-


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0

#save_load.py
"""
Used for functions relating to saving and loading Item Lists
"""
import pickle

def SaveDroneList(dronelist):
    """
    Saves List of drones using pickle.
    """
    with open("database//config.dronelist", "wb") as config_dronelist_file:
        pickle.dump(dronelist, config_dronelist_file)

def SaveLocationList(locationlist):
    """
    Saves List of locations using pickle
    """
```

```
    with open("database//config.locationlist", "wb") as config_locationlist_file:
        pickle.dump(locationlist, config_locationlist_file)

def SaveFlightList(flightlist):
    """
    Save List of flights using pickle
    """
    with open("database//config.flightlist", "wb") as config_flightlist_file:
        pickle.dump(flightlist, config_flightlist_file)

def LoadDroneList():
    """
    Loads list of drones using pickle
    """
    with open("database//config.dronelist", "rb") as drones:
        dronelist = pickle.load(drones)
        return dronelist

def LoadLocationList():
    """
    Loads list of locations using pickle
    """
    with open("database//config.locationlist", "rb") as locations:
        locationlist = pickle.load(locations)
        return locationlist

def LoadFlightList():
    """
    Loads list of flights using pickle
    """
    with open("database//config.flightlist", "rb") as flights:
        flightlist = pickle.load(flights)
        return flightlist
```

FloatInput.py

```
import re
from kivy.uix.textinput import TextInput


#Name:         James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0

#FloatInput class

class FloatInput(TextInput):
    """
    Class designed to ensure only floats can be used for text input for lat/long v
alues.

    Source: https://kivy.org/doc/stable/api_kivy.uix.textinput.html
    """
    pat = re.compile('[^-|0-9]')
    def insert_text(self, substring, from_undo=False):
```

```
        pat = self.pat
        if '.' in self.text:
            s = re.sub(pat, '', substring)
        else:
            s = '.'.join([re.sub(pat, '', s) for s  in substring.split('.', 1)])
        return super(FloatInput, self).insert_text(s, from_undo=from_undo)
```

MainMenu2.py (deprecated)

```
# -*- coding: UTF-8 -*-


#Name:          James Hall
#Student No.: C00007006
#Institute:    Institute of Technology Carlow
#Project:      Drone Traffic Control System
#Date:         April 2021
#License:      GNU Affero General Public License v3.0
"""
Original command line script #deprecated
"""

import pickle
import math
import time
import threading
import multiprocessing
import DroneController
from threading import Thread
from AbstractDrone import AbstractDrone
from VirtualDrone import VirtualDrone
from Location import Location
from Flight import Flight
from save_load import *

#Global thread variables used to kill threads
takeoffThread = False
moveThread = False
altitudeThread = False
hoverThread = False
landThread = False

##########CRUD Drones methods
def CreateDrone(dronelist, locationlist):
    dronename = input("\nEnter drone name:")
    dronetype = input("\nEnter drone type:")
    homelocation = input("\n Enter drone home location:")
    for i in range(len(locationlist)):
        if locationlist[i].getlocationname() == homelocation:
            homelatitude = locationlist[i].getlocationlatitude()
            homelongitude = locationlist[i].getlocationlongitude()

    dronelist.append(VirtualDrone(dronename, dronetype, homelatitude, homelongitud
e, homelatitude, homelongitude, 0, 100, 0, "Grounded", 0))

def ViewSystemDrones(dronelist):
    for i in range(len(dronelist)):
```

```
        print(" " + dronelist[i].drone_id +"," + dronelist[i].getdronetype() + " L
at:" + str(dronelist[i].current_latitude) + " Lon:" + str(dronelist[i].current_lon
gitude) + " State:" + str(dronelist[i].getdronestate()) + " Altitude:" + str(drone
list[i].getaltitude()) + "\n")

def ViewActiveDrones(dronelist):
    for i in range(len(dronelist)):
        if dronelist[i].flying_state != "Grounded":
            print(" " + dronelist[i].drone_id + " Lat:" + str(dronelist[i].current
_latitude) + " Lon:" + str(dronelist[i].current_longitude) + "\n")

def RemoveDrone(dronelist):
    for i in range(len(dronelist)):
        print(" " + dronelist[i].drone_id)
    dronename = input("\nEnter drone name to remove:")

    for i in range(len(dronelist)):
        if dronelist[i].drone_id == dronename:
            dronelist.remove(dronelist[i])
            print(" " + dronename + " removed from Drone List")
            break

##########CRUD Location methods
def CreateLocation(locationlist):
    locationname = input("\nEnter Location Name:")
    locationlatitude = float(input("\nEnter Location Latitude:"))
    locationlongitude = float(input("\nEnter Location Longitude:"))

    locationlist.append(Location(locationname, locationlatitude, locationlongitude
, 0))

def ViewLocations(locationlist):
    for i in range(len(locationlist)):
        print(" " + locationlist[i].getlocationname() + " Lat:" + str(locationlist
[i].getlocationlatitude()) + " Lon:" + str(locationlist[i].getlocationlongitude())
 + "\n")

def RemoveLocation(locationlist):
    for i in range(len(locationlist)):
        print(" " + locationlist[i].getlocationname())
    locationname = input("\nEnter location name to remove:")

    for i in range(len(locationlist)):
        if locationlist[i].getlocationname() == locationname:
            locationlist.remove(locationlist[i])
            print(" " + locationname + " removed from Locations")
            break

##########CRUD Flight methods
def CreateFlight(flightlist, locationlist, dronelist):
    flightname = input("\nEnter Flight Name:")
    dronename = input("\nEnter Drone Name:")

    for i in range(len(dronelist)):
        if dronelist[i].drone_id == dronename:
            drone = dronelist[i].drone_id
            startlatitude = dronelist[i].gethomelatitude()
            startlongitude = dronelist[i].gethomelongitude()
```

```
    destlocation = input("\nEnter Destination Location:")

    for i in range(len(locationlist)):
        if locationlist[i].getlocationname() == destlocation:
            destlatitude = locationlist[i].getlocationlatitude()
            destlongitude = locationlist[i].getlocationlongitude()

    flightlist.append(Flight(flightname, startlatitude, startlongitude, destlatitu
de, destlongitude, drone, False))

def ViewFlights(flightlist):
    for i in range(len(flightlist)):
        print(" " + flightlist[i].getflightid() + " Start Lat:" + str(flightlist[i
].getstartlatitude()) + " Start Lon:" + str(flightlist[i].getstartlongitude()) + "
 Dest Lat:" + str(flightlist[i].getdestinationlatitude()) + " Dest Lon:" + str(fli
ghtlist[i].getdestinationlongitude()) + " Complete:" + str(flightlist[i].getflight
complete()) +"\n")

def RemoveFlight(flightlist):
    for i in range(len(flightlist)):
        print(" " + flightlist[i].getflightid())
    flightname = input("\nEnter location name to remove:")

    for i in range(len(flightlist)):
        if flightlist[i].getflightid() == flightname:
            flightlist.remove(flightlist[i])
            print(" " + flightname + " removed from Flights")
            break

##########BeginFlight uses TakeOffFlyLand as thread
def BeginFlight(flightlist, locationlist, dronelist, threads):
    takeoffThread = False
    global moveThread
    altitudeThread = False
    hoverThread = False
    global landThread

    for i in range(len(flightlist)):
        print(" " + flightlist[i].getflightid())
    flightname = input("\nEnter flight you want to start:")

    for i in range(len(flightlist)):
        if flightlist[i].getflightid() == flightname:
            for j in range(len(dronelist)):
                if flightlist[i].getdrone() == dronelist[j].drone_id:
                    threads.append(Thread(target = DroneController.TakeOffFlyLand,
  args = (dronelist[j], flightlist[i],dronelist)))
                    threads[len(threads)-1].start()
                    return None

##########Main Menu
def MainMenu():

    DronesList = LoadDroneList()
    LocationList = LoadLocationList()
    FlightList = LoadFlightList()

    threads = []
```

```
    loop_true = True


    '''init of game;

    game loop start
        check for inputs
        //update all game objects
        for d in drones
            d.moveTo
        for a certain frame rate
            render all game objects
    end loop'''


    while loop_true == True:
        print(" Enter Number corresponding to your choice:\n")
        print(" 1 Create Drone (Must have location to add to home)")
        print(" 2 View System Drones")
        print(" 3 View Active Drones")
        print(" 4 Remove Drone")
        print(" 5 Create Location")
        print(" 6 View Locations")
        print(" 7 Remove Location")
        print(" 8 Create Flight (Must have minimum 2 locations and 1 drone created
)")
        print(" 9 View Flights")
        print(" 10 Cancel Flight")
        print(" 11 Begin Flight")
        print(" 12 exit program")

        loopinput = int(input("\nEnter number choice:"))

        if loopinput == 12:
            SaveDroneList(DronesList)
            SaveLocationList(LocationList)
            SaveFlightList(FlightList)
            break
        elif loopinput == 1:
            CreateDrone(DronesList, LocationList)
        elif loopinput == 2:
            ViewSystemDrones(DronesList)
        elif loopinput == 3:
            ViewActiveDrones(DronesList)
        elif loopinput == 4:
            RemoveDrone(DronesList)
        elif loopinput == 5:
            CreateLocation(LocationList)
        elif loopinput == 6:
            ViewLocations(LocationList)
        elif loopinput == 7:
            RemoveLocation(LocationList)
        elif loopinput == 8:
            CreateFlight(FlightList, LocationList, DronesList)
        elif loopinput == 9:
            ViewFlights(FlightList)
        elif loopinput == 10:
            RemoveFlight(FlightList)
        elif loopinput == 11:
```

```
            BeginFlight(FlightList, LocationList, DronesList, threads)


if __name__ == "__main__":

    M = Thread(target = MainMenu)
    M.start()
```

Plagiarism Declaration

Declaration

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.

- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.

- I have provided a complete bibliography of all works and sources used in the preparation of this submission.

- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offense.

Student Name: James Hall

Student Number: C00007006

Signature: James Hall