```
/**
* @ProgramName: Program-1
* @Author: Revathi chikoti
* @Description:
*     This program displays coutries, recognizes mouse click and a thick border ap
pers with rectangle
       over borders of country and displays counytry name.
* @Course: 4553 spatial data structures
* @Semester: Summer-1 2017
* @Date: 06-14-2017
*/


import json
import os,sys
import pygame
import random
import math
import pprint as pp

# Get current working path
DIRPATH = os.path.dirname(os.path.realpath(__file__))

###################################################################################
##
###################################################################################
##

class Colors(object):
    """
    Opens a json file of web colors.
    """
    def __init__(self,file_name):

        with open(file_name, 'r') as content_file:
            content = content_file.read()

        self.content = json.loads(content)

    def get_random_color(self):
        """
        Returns a random rgb tuple from the color dictionary
        Args:
            None
        Returns:
            color (tuple) : (r,g,b)
        Usage:
            c = Colors()
            some_color = c.get_random_color()
            # some_color is now a tuple (r,g,b) representing some lucky color
        """
        r = random.randint(0,len(self.content)-1)
        c = self.content[r]
        return (c['rgb'][0],c['rgb'][1],c['rgb'][2])

    def get_rgb(self,name):
        """
        Returns a named rgb tuple from the color dictionary
        Args:
            name (string) : name of color to return
        Returns:
            color (tuple) : (r,g,b)
        Usage:
            c = Colors()
            lavender = c.get_rgb('lavender')
            # lavender is now a tuple (230,230,250) representing that color
        """
        for c in self.content:
            if c['name'] == name:
                return (c['rgb'][0],c['rgb'][1],c['rgb'][2])
        return None

    def __getitem__(self,color_name):
        """
        Overloads "[]" brackets for this class so we can treat it like a dict.
        Usage:
            c = Colors()
            current_color = c['violet']
            # current_color contains: (238,130,238)
        """
        return self.get_rgb(color_name)

###################################################################################
##
###################################################################################
##

class StateBorders(object):
    """
    Opens a json file of the united states borders for each state.
    """
    def __init__(self,file_name):
        """
        Args:
            filename (string) : The path and filename to open
        Returns:
            None
        """
        with open(file_name, 'r') as content_file:
            content = content_file.read()

        self.content = json.loads(content)
    def get_state(self,name):
        """
        Returns a polygon of a single state from the US.
        Args:
            name (string): Name of a single state.

        Returns:
            json (string object): Json representation of a state

        Usage:
            sb = StateBorders()
            texas = sb.get_state_polygon('texas')
            # texas is now a list object containing polygons
        """
        for s in self.content:
            if s['name'].lower() == name.lower() or s['code'].lower() == name.low
er():
                t = []
                for poly in s['borders']:
                    np = []
                    for p in poly:
                        np.append((p[0],p[1]))
                    t.append(np)
                return(t)

        return None

    def get_continental_states(self):
        """
        Returns a list of all the continental us states as polygons.
        Args:
            None

        Returns:
            list (list object): list of Json objects representing each continental
state.

        Usage:
            sb = StateBorders()
            states = sb.get_continental_states()
            # states is now a list object containing polygons for all the continent
al states
        """
        states = []
        for s in self.content:
            t = []
            if s['name'] not in ['Alaska','Hawaii']:
                for poly in s['borders']:
                    np = []
                    for p in poly:
                        np.append((p[0],p[1]))
                t.append(np)
            states.append(t)
        return(states)

    def key_exists(self,key):
        """
        Returns boolean if key exists in json
        Args:
            key (string) : some identifier

        Returns:
            T/F (bool) : True = Key exists
        """
        for s in self.content:
            if s['name'].lower() == key.lower():
                return True
            elif s['code'].lower() == key.lower():
                return True
        return False

##################################################################################
##
##################################################################################
##

class WorldCountries(object):
    """
    Opens a json file of the united states borders for each state.
    """
    def __init__(self,file_name):
        with open(file_name, 'r') as content_file:
            content = content_file.read()

        self.content = json.loads(content)

    def get_all_countries(self):
        """
        Returns a list of all the countries us states.
        Args:
            None

        Returns:
            list (list object): List of Json objects representing each country

        Usage:
            wc = WorldCountries()
            countries = wc.get_all_countries()
            # countries is now a list object containing polygons for all the countr
ies
        """
        all_countries = []
        for c in self.content['features']:
            if c['id'] in ["ATA"]:
                continue
            all_countries.append(c['geometry']['coordinates'])
        return all_countries

    def get_country(self,id):
        """
        Returns a list of one country.
        Args:
            None

        Returns:
            list (list object): List of Json object representing a country

        Usage:
            wc = WorldCountries()
            country = wc.get_country('AFG')
            # country is now a list object containing polygons for 'Afghanistan'
        """
        country = []
        for c in self.content['features']:
            if c['id'].lower() == id.lower() or c['properties']['name'].lower() =
= id.lower():
                country.append(c['geometry']['coordinates'])
        return country

    def key_exists(self,key):
        """
        Returns boolean if key exists in json
        Args:
            key (string) : some identifier

        Returns:
            T/F (bool) : True = Key exists
        """
        for c in self.content['features']:
            if c['id'].lower() == key.lower():
                return True
            elif c['properties']['name'].lower() == key.lower():
                return True
        return False


##################################################################################
##
##################################################################################
##

class DrawGeoJson(object):
    __shared_state = {}
    def __init__(self,screen,width,height):
        """
        Converts lists (polygons) of lat/lon pairs into pixel coordinates in order 
to do some
        simple drawing using pygame.
        """
        self.__dict__ = self.__shared_state

        self.screen = screen    # window handle for pygame drawing

        self.polygons = []      # list of lists (polygons) to be drawn


        self.all_lats = []      # list for all lats so we can find mins and max's
        self.all_lons = []

        # New added June 13
        self.adjusted_polys = []
        # New added June 13
        self.adjusted_poly_dict = {}

        self.mapWidth = width       # width of the map in pixels
        self.mapHeight = height     # height of the map in pixels
        self.mapLonLeft = -180.0    # extreme left longitude
        self.mapLonRight = 180.0    # extreme right longitude
        self.mapLonDelta = self.mapLonRight - self.mapLonLeft # difference in long
itudes
        self.mapLatBottom = 0.0     # extreme bottom latitude
        self.mapLatBottomDegree = self.mapLatBottom * math.pi / 180.0 # bottom in 
degrees

        self.colors = Colors(DIRPATH + '/../Json_Files/colors.json')

    def convertGeoToPixel(self,lon, lat):
        """
        Converts lat/lon to pixel within a set bounding box
        Args:
            lon (float): longitude
            lat (float): latitude

        Returns:
            point (tuple): x,y coords adjusted to fit on print window
        """
        x = (lon - self.mapLonLeft) * (self.mapWidth / self.mapLonDelta)

        lat = lat * math.pi / 180.0
        self.worldMapWidth = ((self.mapWidth / self.mapLonDelta) * 360) / (2 * mat
h.pi)
        self.mapOffsetY = (self.worldMapWidth / 2 * math.log((1 + math.sin(self.m
apLatBottomDegree)) / (1 - math.sin(self.mapLatBottomDegree))))
        y = self.mapHeight - ((self.worldMapWidth / 2 * math.log((1 + math.sin(la
t)) / (1 - math.sin(lat)))) - self.mapOffsetY)

        return (x, y)
    def add_polygon(self,poly,id=None):
        """
        Add a polygon to local collection to be drawn later
        Args:
            poly (list): list of lat/lons

        Returns:
            None
        """
        self.polygons.append(poly)
        # New added June 13
        if id is not None:
            # if country not in dict, make a list for its polygons
            # to be appended to.
            if id not in self.adjusted_poly_dict:
                self.adjusted_poly_dict[id] = []
            # append poly to dictionary with country as key (id).
            self.adjusted_poly_dict[id].append(poly)
        for p in poly:
            x,y = p
            self.all_lons.append(x)
            self.all_lats.append(y)
        self.__update_bounds()

    # We should use recursion on these containers with arbitrary depth, but oh well
.
    def adjust_poly_dictionary(self):
        #pp.pprint(self.adjusted_poly_dict)
        for country,polys in self.adjusted_poly_dict.items():
            new_polys = []
            print(country)
            for poly in polys:
                new_poly = []
                for p in poly:
                    x,y = p
                    new_poly.append(self.convertGeoToPixel(x,y))
                new_polys.append(new_poly)
            self.adjusted_poly_dict[country] = new_polys
        pp.pprint(self.adjusted_poly_dict)

    def drawrect(self,position):
        """
        Determines which polygon contains the click
        Border that polygon with a thick black border.
        Prints the countries name or states name on the screen
        Draw bounding box rectangle around the country or state you clicked.
        """
        black=(0,0,0)
        red = (255,0,0)
        lon = []
        lat = []
        #polylist=[]
        for i in self.polygons :
            polylist=[]
            for val in i:
                polylist.append(self.convertGeoToPixel(val[0],val[1]))
            if point_inside_polygon(position[0],position[1],polylist):
                pygame.draw.polygon(self.screen,black,polylist,8)
                for k in range(len(polylist)):
                    lon.append(polylist[k][0])
                    lat.append(polylist[k][1])
                point1 = min(lon), min(lat)
                point2 = min(lon), max(lat)
                point3 = max(lon), max(lat)
                point4 = max(lon), min(lat)

                pygame.draw.line(self.screen, red, point1, point2,4)
                pygame.draw.line(self.screen, red, point2, point3,4)
                pygame.draw.line(self.screen, red, point3, point4,4)
                pygame.draw.line(self.screen, red, point4, point1,4)
                self.addText(position)
    def addText(self,position):
            black = (0,0,0)
            for key,value in self.adjusted_poly_dict.items():
                name_list = []
                for v in value:
                    for m in v:
                        name_list.append(m)
                    if point_inside_polygon(position[0],position[1],name_list):

                        pygame.init()
                        myfont = pygame.font.SysFont('Comic Sans MS', 30)
                        textsurface = myfont.render(key, False, (0, 0, 0))
                        screen.blit(textsurface,(200,300))

               # pygame.draw.line(self.screen, red, point1, point3)

    def draw_polygons(self):
        """
        Draw our polygons to the screen
        Args:
            None

        Returns:
            None
        """

        for poly in self.polygons:
            adjusted = []
            for p in poly:
                x,y = p
                adjusted.append(self.convertGeoToPixel(x,y))
            # New added June 13
            self.adjusted_polys.append(adjusted)
            pygame.draw.polygon(self.screen, self.colors.get_random_color(), adjus
ted, 0)

    def __update_bounds(self):
        """
        Updates the "extremes" of all the points added to be drawn so
        the conversion to x,y coords will be adjusted correctly to fit
        the "bounding box" surrounding all the points. Not perfect.
        Args:
            None

        Returns:
            None
        """
        self.mapLonLeft = min(self.all_lons)
        self.mapLonRight = max(self.all_lons)
        self.mapLonDelta = self.mapLonRight - self.mapLonLeft
        self.mapLatBottom = min(self.all_lats)
        self.mapLatBottomDegree = self.mapLatBottom * math.pi / 180.0


    def __str__(self):
        return "[%d,%d,%d,%d,%d,%d,%d]" % (self.mapWidth,self.mapHeight,self.mapL
onLeft,self.mapLonRight,self.mapLonDelta,self.mapLatBottom,self.mapLatBottomDegree
)


##################################################################################
##
##################################################################################
##

class DrawingFacade(object):
    def __init__(self,width,height):
        """
        A facade pattern is used as a type of 'wrapper' to simplify interfacing wit
h one or
        more other classes. This 'facade' lets us interface with the 3 classes inst
antiated
        below.
        """
        self.sb = StateBorders(DIRPATH + '/../Json_Files/state_borders.json')
        self.wc = WorldCountries(DIRPATH + '/../Json_Files/countries.geo.json')
        self.gd = DrawGeoJson(screen,width,height)

    def add_polygons(self,ids):
        """
        Adds polygons to the 'DrawGeoJson' class using country names or id's, state
 names or code's. It
        expects a list of values.
```

```python
        Args:
            ids (list) : A list of any state or country identifiers

        Returns:
            None

        Usage:
            df.add_polygons(['FRA','TX','ESP','AFG','NY','ME','Kenya'])
        """
        for id in ids:
            if self.wc.key_exists(id):
                self.__add_country(self.wc.get_country(id),id)
            elif self.sb.key_exists(id):
                self.__add_state(self.sb.get_state(id),id)

    def __add_country(self,country,id=None):
        for polys in country:
            for poly in polys:
                if type(poly[0][0]) is float:
                    gd.add_polygon(poly,id)
                else:
                    for sub_poly in poly:
                        self.gd.add_polygon(sub_poly,id)

    def __add_state(self,state,id=None):
        for poly in state:
            self.gd.add_polygon(poly,id)



def point_inside_polygon(x,y,poly):
    """
    determine if a point is inside a given polygon or not
    Polygon is a list of (x,y) pairs.
    http://www.ariel.com.au/a/python-point-int-poly.html
    """
    n = len(poly)
    inside =False

    p1x,p1y = poly[0]
    for i in range(n+1):
        p2x,p2y = poly[i % n]
        if y > min(p1y,p2y):
            if y <= max(p1y,p2y):
                if x <= max(p1x,p2x):
                    if p1y != p2y:
                        xinters = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
                    if p1x == p2x or x <= xinters:
                        inside = not inside
        p1x,p1y = p2x,p2y

    return inside
```

```
##################################################################################
##################################################################################

def mercator_projection(latlng,zoom=0,tile_size=256):
    """
    ******NOT USED******
    The mapping between latitude, longitude and pixels is defined by the web mercat
or projection.
    """
    x = (latlng[0] + 180) / 360 * tile_size
    y = ((1 - math.log(math.tan(latlng[1] * math.pi / 180) + 1 / math.cos(latlng[
1] * math.pi / 180)) / math.pi) / 2 * pow(2, 0)) * tile_size

    return (x,y)

if __name__ == '__main__':

    # if there are no command line args
    if len(sys.argv) == 1:
        width = 1024    # define width and height of screen
        height = 512
    else:
        # use size passed in by user
        width = int(sys.argv[1])
        height = int(sys.argv[2])

    # create an instance of pygame
    # "screen" is what will be used as a reference so we can
    # pass it to functions and draw to it.
    screen = pygame.display.set_mode((width, height))

    # Set title of window
    pygame.display.set_caption('Draw World Polygons')

    # Set background to white
    screen.fill((255,255,255))

    # Refresh screen
    pygame.display.flip()

    # Instances of our drawing classes
    gd = DrawGeoJson(screen,width,height)
    df = DrawingFacade(width,height)

    print(gd.__dict__)

    # Add countries and states to our drawing facade.
    # df.add_polygons(['FRA','TX','ESP','AFG','NY'])
    # df.add_polygons(['TX','NY','ME','Kenya'])
    df.add_polygons(['Spain','France','Belgium','Ireland', 'Scotland','Greece','Ger
many','Egypt','Morocco','India'])


    # Call draw polygons to "adjust" the regular polygons
    gd.draw_polygons()
    # Call my new method to "adjust" the dictionary of polygons
    gd.adjust_poly_dictionary()

    # Main loop
    running = True
    while running:
        gd.draw_polygons()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                print(event.pos)
                gd.drawrect(event.pos)


            pygame.display.flip()
```