Flood Emergency Planning

Report for the CEGE0096: 2nd Assignment

Team Hornbill

Harrison Luft
20088591
ucesluf@ucl.ac.uk

YanTing Lam
20144479
stnv525@ucl.ac.uk

Shengze Lin
20096415
ucessl3@ucl.ac.uk

https://github.com/harrisonluft/Assignment_2

Introduction

Consistent with the assignment brief, the aim of the software is to provide an individual evacuation map for flood emergency planning on the Isle of Wight. Specifically, it functions to take in an input point located on the Isle of Wight provided by the user (in National Grid coordinates) and provides the user with the fastest directions from their location to the highest point within a five kilometer radius. The output consists of a map showing the route plotted along the road network with a transparent overlay of the elevation of the surrounding area. In terms of completeness of the assignment, the software contains completed versions of all tasks required to solve the assignment, which will be detailed in the following sections of the report. Limitations of the assignment include the need for the user to supply the point in a specific format and coordinate system, the software would be improved by allowing the user to supply an address or any form of GNSS location. A further limitation is that the software does not limit the fastest route to the aforementioned five kilometer radius. Therefore, depending on where the highest point lies in the five kilometer radius of the user, the route may extend beyond this area at some points (though it will always re-enter).

Development of the software took place primarily over the course of two weeks in which we took shifts developing the tasks sequentially. Git and Github were utilized to share in-process code with group members and resolve coding conflicts when they arose. Group members took turns contributing to the master branch, pushing only functioning versions of their tasks to Github. Meanwhile, the non-master branch members would work coincidently on their local branch tasks, like refining their sections of the software and working to adjust formatting. At the conclusion of each task, project branches were merged and pulled by group members to ensure each member was up to date with the others.

The rest of the report is organized as follows. Section two will explain how to use the software, listing any prerequisites, and required folder structure, and user input specifics. Section three will expand upon the specific project tasks, walking through the steps necessary to complete the assignment. Finally, the git log is shared at the bottom of the report to show group member participation.

Project Description

Our software is run on Python 3.8 and the libraries are listed below:

Libraries: os, sys, rasterio, shapely.geometry, numpy, geopandas, json, cartopy, networkx, matplotlib, matplotlib_scalebar, pyproj, and rtree.

Seven Python scripts are provided in a Scripts folder: 1) main.py 2) bounding_box.py 3) raster_buffer.py 4) nearest_itn.py 5) ShortestPath.py 6) MapPlotting.py and 7) on_island.py. In addition to this a “Materials” folder is provided containing all of the data and geographic layers that the software requires. The Materials and Scripts folders should be kept within a parent folder, Assignment_2. The path to Assignment_2 should be inputted as the working directory at

the top of the ‘main.py’ script, replacing the text ‘INSERT PATH HERE’. All paths to the required inputs are coded in the programs based on file names provided in the project materials. After ensuring all script files and materials are located in their respective folders within the Assignment_2 folder, simply executing the main.py will run the software, first prompting the user to enter their location in National Grid coordinates.

Also note that the software runs via the method described in Task 6 of the assignment brief, by default. This is because if run via the method in Task 1, areas within 5km of the edge of the island will be excluded. Therefore, Task 1 is described below with code examples, though not actually executed in the program.

Object oriented programming is used throughout the software (in every subtask) as a way of making our program usable outside of this specific purpose. Therefore, much of the problem solving takes place outside of the ‘main.py’. In addition to allowing the code to be repurposed outside of this assignment, it also helped keep the subtasks organized for the group. Each step is explained in detail below.

Software

Below, the six necessary tasks to complete the assignment are presented in detail, walking through any necessary classes, methods or functions.

Task 1: User Input

As mentioned above, this Task is not run executed in the program, but is described here to satisfy the solution to the assignment. First the user is prompted to input a point in National Grid coordinates as ‘Eastings, Northings’ without brackets, separated by a comma. In the event the user does not input a correctly formatted coordinate the program will raise an error prompting the user to input a coordinate in the correct format. The input function is presented below:

```
def user_input():
    data_list = []
    while True:
        try:
            user_point = input('Please input coordinate as Eastings, Northings:
')
            user_point_float = [float(user_point.split(',')[0]),
float(user_point.split(',')[1])]
            break
        except:
            print('Invalid input, please try again')
    data_list.append(user_point_float)
    return data_list
```

The ‘data_list’ is a generic list to temporarily hold input coordinates until they are passed to the assigned variable, in this case:

```
input = user_input()
```

Following this, the extent of the map is defined by the class Mbr. In the case of this software the extent is given in the assignment, therefore the coordinates are hard coded to define the mapping extent.

```
# hardcode extent of bounding box
extent = (430000, 80000, 465000, 95000)
mbr = Mbr(extent)
```

The Mbr class, contained within the 'bounding_box.py' script, contains a method, `within_extent()`, which tests the input point against the defined map extent. If the input point resides within the map extent, the program proceeds. If not, the program terminates and the user must re-execute the program and re-enter a new point. The `within_extent()` method is presented below:

```
def within_extent(self, input):
   self.min_x = min(self.E1, self.E2)
   self.min_y = min(self.N1, self.N2)
   self.max_x = max(self.E1, self.E2)
   self.max_y = max(self.N1, self.N2)

   if (input[0][0] > self.max_x) or (input[0][1] > self.max_y) or \
      (input[0][0] < self.min_x) or (input[0][1] < self.min_y):

       print('Number provided is outside map extent')
       return sys.exit(0)
```

Where E1, E2, N1, N2 are the coordinates of the mapping extent, and the input is the coordinate pair inputted by the user.

This method is implemented via the 'bounding_box.py' script and implemented in the 'main.py' commit version: 791c86ac96453d22a301e4cc96ed3855e38bf162 where a toggle is available to run the program as either via task 1 or task 6.

Task 2: Highest Point Identification

Per the assignment brief, the user must be routed to the highest point within a five kilometer radius, so outside of the user point itself, the next step is to define the area in which the highest point will be identified from.

The program utilizes the Shapley Point geometry to transform the user point into a geometry object, then uses Shapley's buffer() method to generate a 5km radius polygon circle around the point. This buffer object is then plugged in to initialize the Rasterbuffer class.

Rasteruffer takes in three arguments 1) a polygon, 2) the path of an elevation raster, and 3) the output path for an elevation raster. These paths are pre-entered, as they are static for the purposes of this assignment. After defining the class over the three inputs, the program runs the method `clip_raster()` which clips the input raster to the bounds of the input polygon and exports it to the specified file path. The method is adopted from Henrikki Tenkanen (2018). Below is the code for the `clip_raster()` method:

```
def clip_raster(self):
   self.geo = gpd.GeoDataFrame({'geometry': self.buffer}, index=[0],
crs=CRS.from_epsg(27700))

   def getfeatures(gdf):
       """Static method to parse features from GeoDataFrame in such a manner
that rasterio wants them"""
       return [json.loads(gdf.to_json())['features'][0]['geometry']]

   self.coords = getfeatures(self.geo)

   with rasterio.open(self.raster_path) as self.src:
       self.out_image, self.out_transform = rasterio.mask.mask(self.src,
self.coords, crop=True)
       self.out_meta = self.src.meta

   self.out_meta.update({"driver": "GTiff",
                         "height": self.out_image.shape[1],
                         "width": self.out_image.shape[2],
                         "transform": self.out_transform})

   with rasterio.open(self.out_path, "w", **self.out_meta) as self.dest:
       self.dest.write(self.out_image)
```

Rasterio requires the coordinates of the input polygon to be formatted as a GeoJSON, therefore the method transforms the buffer polygon into a geodataframe, then passes this geodataframe through the `getfeatures()` static method to transform them into a GeoJSON. Rasterio is then used to import the raw elevation raster and the command `rasterio.mask.mask()` is used to mask out pixels outside the input polygon. Following this the meta data for the clipped raster is updated and then exported to the output path specified in the Rasterbuffer initialization.

To identify the highest point within the specified five kilometer area the clipped raster is imported with Rasterio and transformed into a matrix with Numpy, as below:

```
# import 5km clipped raster
clipped = rasterio.open(os.path.join('Materials', 'elevation', '5k_mask.tif'))

# reading raster as numpy array
matrix = clipped.read(1)
```

Next operations are done to identify the highest number and the index of that number within the matrix:

```
# max height value
max_height = np.amax(matrix)

#  index of max height
result = np.where(matrix == max_height)
```

Finally, the high point's index location is used to identified the x, y point using the following code:

```
#  coordinates of max height and geodataframe construction
high_point = clipped.xy(result[0], result[1])
high_point_obj = Point(float(high_point[0][0]), float(high_point[1][0]))
```

Where 'clipped' is the clipped raster and 'result' is the index of the highest location in the Numpy in the above code. A shapely Point object is constructed out of the coordinates for later use in task 3 and a geodataframe is constructed out of the 'high_point' in order to map the point.

Task 3: Nearest Integrated Transport Network

Tasks 1/6 and 2 define the coordinates of the start and end points of the emergency plan. Now, those points must be translated into nodes along the transportation network in order to solve for the shortest path between the two. Finding the nearest nodes to each of the points requires first loading in the network data, then using the library r-trees index the spatial information, and finally using methods from r-trees to locate the nearest transportation node to each point of interest.

To accomplish the above, the class 'Itn' from the 'nearest_node.py' script is used. The class is initialized by inputting a path to the integrated transport network JSON file. As before, this path is fixed in the code for the purposes of this software. First, the method `itn_index()` imports the JSON file and extracts its nodes. Then, an index is established using the r-trees default constructor, and an 'id_list' is generated. The 'id_list' is used for keeping track of each node inputted into the index, so that it is possible to reference the nearest nodes by number, rather than name, further along in the program. Each set of coordinates is then extracted from the nodes list and inserted into the spatial index. The code for the `itn_index()` method is presented below:

```
def itn_index(self):

    # JSON file
    with open(self.itn_path, "r") as self.f:
        self.itn = json.load(self.f)

    self.nodes = self.itn['roadnodes']

    self.idx = index.Index()
    self.id = 0
```

```
    self.id_list = []
    for key, value in self.nodes.items():
        self.point = Point(float(value["coords"][0]), float(value["coords"][1]))
        self.idx.insert(self.id, (self.point.x, self.point.y))
        self.id_list.append([key])
        self.id += 1
```

After the transport network node values are inserted into the spatial index, the rtrees methods are available in order to find the nearest node to an input point. Therefore, the nearest nodes are found easily using the nearest() method from r-trees. This software's implementation is shown below:

```
def nearest_node(self, user_input):

    for i in self.idx.nearest((user_input.x, user_input.y), 1):
        return print(i, self.id_list[i])
```

In the 'main.py' task 3 is accomplished simply by calling all of the above with the 'user_input' and 'highest_point_obj'. The code is presented below:

```
# Step 3 importing ITN network
step_3 = Itn(os.path.join('Materials', 'itn', 'solent_itn.json'))
# nearest nodes to both the user input and highest points
step_3.itn_index()
print('Nearest ITN node to user: ')
step_3.nearest_node(user_point)
print('Nearest ITN node to highest point:')
step_3.nearest_node(high_point_obj)
```

Task 4: Shortest Path

In this part, we build a class called 'ShortestPath'. It contains four methods. The first one is 'get_itn_and_elevation', this is a simple method to read files of the itn and elevation data and to prepare them for the later process. The code of this method is shown below.

```
def get_itn_and_elevation(self):

    self.itn_json = os.path.join('Materials', 'itn', 'solent_itn.json')
    with open(self.itn_json, 'r') as f:
        self.itn = json.load(f)

    self.dataset = rasterio.open(os.path.join('Materials', 'elevation',
'SZ.asc'))
    self.elevation_raster = self.dataset.read(1)
```

The second method is 'weight_calculate', this method calculates the weight for both directions of the edge based on length and elevation (Naismith's rule). The change of elevation is

calculated by iterating through each link segment. This method returns two arguments for the next method. The code of this method is shown below.

```
temp_coords = []
for coord in coords:
   temp_coords.append([round(coord[0]), round(coord[1])])
temp_elevations = []
for coord in temp_coords:
   row, col = self.dataset.index(coord[0], coord[1])
   temp_elevations.append(self.elevation_raster[row, col])
elevations = np.array(temp_elevations)
array_length = elevations.shape

weight_start_end = length * 12 / 1000
for i in range(1, array_length[0]):
   x = elevations[i] - elevations[i - 1]
   if x > 0:
       weight_start_end += x * 10 / 100

weight_end_start = length * 12 / 1000
for i in range(0, array_length[0] - 1):
   x = elevations[i] - elevations[i + 1]
   if x > 0:
       weight_end_start += x * 10 / 100

return weight_start_end, weight_end_start
```

The third method is to create a DiGraph, this is because it takes different time to walk from different directions. During creating the graph, we would use the second method in this class to give value to both directions of the edge. The last method is to use ‘dijkstra_path()’ method in package ‘networkX’ to calculate the shortest path, and return the shortest path for map plotting in Task 5. The code of the two methods are shown below.

```
# create a Digraph so each direction has its own weight
def create_graph(self):

   self.g = nx.DiGraph()
   road_links = self.itn['roadlinks']
   for link in road_links:
       weight = self.weight_calculate(road_links[link]['length'],
road_links[link]['coords'])
       weight1 = weight[0]
       weight2 = weight[1]
       self.g.add_edge(road_links[link]['start'], road_links[link]['end'],
fid=link, weight=weight1)
       self.g.add_edge(road_links[link]['end'], road_links[link]['start'],
fid=link, weight=weight2)

# get and return shortest path
```

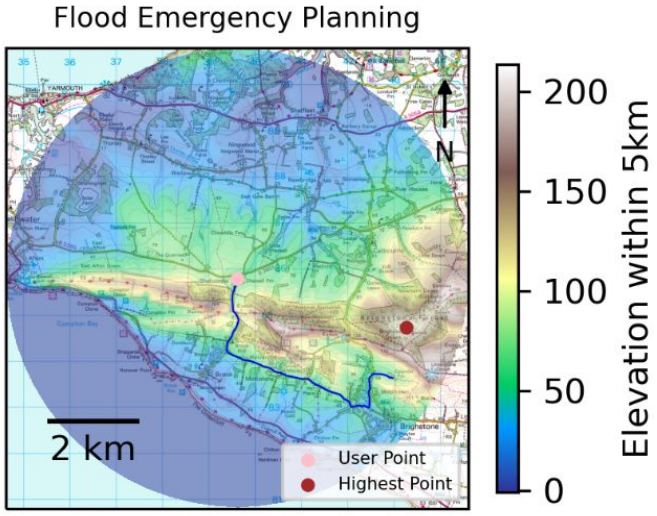
```
def get_shortest_path(self):

    shortest_path = nx.dijkstra_path(self.g, source=str(self.source_point),
                                     target=str(self.target_point),
weight='weight')
    shortest_path_time = nx.shortest_path_length(self.g,
source=str(self.source_point),
                                                 target=str(self.target_point),
weight='weight')

    return shortest_path, shortest_path_time
```

Task 5: Map Plotting

In class ‘MapPlotting’ there is only one method called ‘show_path’. This method is to display the background map and the shortest path, as well as some other information. The plotting result is implemented through several steps.

1. Converting the shortest path to a geodataframe in order to plot it on the map.
2. Importing the background map, and set its extent to 10km * 10km square around the user point.
3. Adding some details to the map including title, scale bar and north arrow.
4. Adding user point and the highest point to the map, as well as the legend.
5. Overlaying a transparent elevation raster and adding the color-bar showing the elevation range.
6. Showing the map.

Here is an example of plotting result (User Point: 439619, 85800):



Task 6: Extending the Region

As mentioned in Task 1 the extent of the route mapping is restricted to a bounding box of within 5km of the edge of the island. This section will present code to overcome this limitation by implementation of an R-tree spatial index used to classify whether or not the user point is contained within the Isle of Wight. The method presented below is an adaptation of Geoff Boeing (2016).

Using class 'Contains', from the 'on_island.py' file, the function `is_within_geo()` generates the classification by the following. First the function reads in the shapefile provided in the assignment materials and transforms it into a Shapely Multipolygon object. Then, an R-tree spatial index is generated for the user point's geometry. The intersect of the polygon's bounds and the R-tree index is taken to establish an index of possible matches within the index of the user points. In this case, it would either return a list containing one index element or none, because the spatial index is only composed of one element. Before proceeding, the program performs a check to make sure of no false-positives. If the R-tree rectangle containing the users point lies both inside and outside the polygon it would cause the program to wrongly consider the point on the Isle of Wight. To account for this, the list of possible matches is intersected directly with the Isle of Wight polygon itself, resulting in a geodataframe containing the points within the bounds of the island.

This r-trees spatial index approach would be much more computationally advantageous when working to determine if several points lie inside or outside the island, though it does provide the accurate result that this assignment requires. Find below the code to solve Task 6:

```
class Contains:

    def __init__(self, user_gdf, shapefile_path):
        self.user_gdf = user_gdf
        self.shapefile_path = shapefile_path

    # solution adopted from
https://geoffboeing.com/2016/10/r-tree-spatial-index-python/
    def is_within_geo(self):
        self.shapefile = gpd.read_file(self.shapefile_path)
        self.polygon = MultiPolygon(self.shapefile.geometry.iloc[0])
        self.gpd_point = self.user_gdf
        self.spatial_index = self.gpd_point.sindex
        possible_matches_index =
list(self.spatial_index.intersection(self.polygon.bounds))
        possible_matches = self.gpd_point.iloc[possible_matches_index]
        precise_matches =
possible_matches[possible_matches.intersects(self.polygon)]
```

```
if precise_matches.empty:
    return False
else:
    return True
```

Where ‘precise_matches’ contains the list of points found to be within the Isle of Wight. As mentioned before, this either contains one or no elements, therefore, if the boolean test passes, the user point is within the borders of the island and the program proceeds.This code, paired with the raster clipping functions from Task 2 allows the user to be anywhere on the map and the buffer region will be populated with the relevant elevation contours. As mentioned previously, and for practical purposes, this is the default method of the software

References:

Tenkanen, Henrikki 2018. Intro to Python GIS - Masking / clipping raster. Accessible at: https://automating-gis-processes.github.io/CSC/notebooks/L5/clipping-raster.html

Boeing, Geoff 2016. R-Tree Spatial Indexing with Python. Accessible at: https://geoffboeing.com/2016/10/r-tree-spatial-index-python/

Git Log

commit e7c31409af1209b7354b19e1d130e4cdd7396581

Author: linshengze <linshengze980605@gmail.com>

Date: Mon Jan 11 22:32:19 2021 +0000

    fix: fix spelling mistake

commit 0f0905c9e398753fcf166b9bfc647ec86b34ce01

Author: linshengze <linshengze980605@gmail.com>

Date: Mon Jan 11 22:27:01 2021 +0000

    fix: change some comment

commit c6f708ee3232460d6a6bcfbc3b6e318f6781521b

Merge: 36a7f1c 7ce5a42

Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 11 22:24:52 2021 +0000

    Merge remote-tracking branch 'origin/master'

commit 36a7f1c79f1d5dbdcccae1192ee037ae084cf625
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 11 22:24:39 2021 +0000

    fix: change some comment

commit 7ce5a42374a933a2602be92ba478e18908ba8a26
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 21:46:44 2021 +0000

    fix: adjust print statements, and map formatting.

commit 209d7b11ace1353a9fd1d68bac9135a88db09a67
Merge: 994e95f 35dbc24
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 21:32:29 2021 +0000

    Merge remote-tracking branch 'origin/master'

    # Conflicts:
    #   Scripts/main.py

commit 994e95f1128f4a8d1a7aa9138d9a8a555c5a3c3a
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 21:30:30 2021 +0000

fix: exclude task 1 from the final script, clean up print statements.

commit 35dbc2412cc8acc51e1d6d1a54c6614d6b262129
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 11 21:26:44 2021 +0000

fix: add title for the map

commit d8be0a415a33ff59fe9388fb127765f50615d3ed
Merge: 546d5ef b9d6c1a
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 11 21:22:20 2021 +0000

Merge remote-tracking branch 'origin/master'

commit 546d5ef7aa56453854515b820ef898d31e7411f1
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 11 21:21:44 2021 +0000

fix: pep 8

commit b9d6c1aa322e4d8319e05ee8689c6813b74794f7
Merge: 12a335f 5fbf366
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Jan 11 21:05:32 2021 +0000

Merge pull request #9 from harrisonluft/hl-branch

fix: format PEP8.

commit 5fbf36692033616e6fa0829857e37280048d717a

Merge: 440f01a 12a335f

Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>

Date: Mon Jan 11 21:04:57 2021 +0000

    Merge branch 'master' into hl-branch

commit 440f01ad8b274992835980e7103d40d5b3349e5e

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Mon Jan 11 21:01:44 2021 +0000

    fix: format PEP8.

commit 12a335f6b3e26a69c15874350234513ce480e9e8

Author: linshengze <linshengze980605@gmail.com>

Date: Mon Jan 11 20:47:48 2021 +0000

    fix: fix step5 in main.py

commit 6b26e5805fe3bee2ee24f82b2b290f322f626b4d

Merge: c37cfc6 6d53c72

Author: linshengze <linshengze980605@gmail.com>

Date: Mon Jan 11 20:46:08 2021 +0000

    Merge remote-tracking branch 'origin/master'

    # Conflicts:

    #   Scripts/main.py

commit c37cfc66f12053f9f79e4e76005bfa0f018615da

Author: linshengze <linshengze980605@gmail.com>

Date: Mon Jan 11 20:45:19 2021 +0000

    feat: add details to the map

commit 6d53c721359e01e048e94ee679b1da162ccdad04

Merge: 2ae10b9 6794ecb

Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>

Date: Mon Jan 11 15:32:03 2021 +0000

    Merge pull request #8 from harrisonluft/hl-branch

    fix: updated print notifications for closing application.

commit 6794ecba8a0cd8d7a19f7799502b97f43af1950b

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Mon Jan 11 15:30:33 2021 +0000

    fix: updated print notifications for closing application.

commit 2ae10b9bbaff7c6fdfcd26e4931091009735782b

Merge: b7c5532 f680715

Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>

Date: Mon Jan 11 15:00:54 2021 +0000

    Merge pull request #7 from harrisonluft/hl-branch

    fix: update to intersecting layers.

commit f6807157b8757b035bc1523f7d2648343fd695af
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 14:59:51 2021 +0000

    fix: update to intersecting layers.

commit b7c55326f3cda47267341218898a064b591fbad3
Merge: 64fca91 232207c
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Jan 11 13:46:10 2021 +0000

    Merge pull request #6 from harrisonluft/hl-branch

commit 232207c24fe31645ec05d3733149387801bb943c
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 13:43:47 2021 +0000

    fix: updates formatting and PEP8.

commit 791c86ac96453d22a301e4cc96ed3855e38bf162
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 13:43:33 2021 +0000

    feat: incorporated task 6 functionality.

commit 75deb2eea6501b0123eee039e555dda07ab425e9
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Jan 11 13:43:05 2021 +0000

    feat: generating spatial index to categorize points as in or out of polygon (task 6).

commit 64fca91b36e3747e05ab7ce4d734e8d67cb6172e
Merge: 3e58e4a c867919
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Jan 4 09:51:16 2021 +0000

    Merge pull request #5 from harrisonluft/hl-branch

    fix: updated PEP8 formatting throughout.

commit c867919a8ca8a7f4900668020184442730143f0a
Merge: 5fe3a20 3e58e4a
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Jan 4 09:51:02 2021 +0000

    Merge branch 'master' into hl-branch

commit 3e58e4adc4f91e021675103fb6284129636e320c
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 06:36:57 2021 +0000

    feat: basic map plotting, not yet completed

commit b2546b8c7daf94da99075ef78e310f771031d6e8
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 05:51:36 2021 +0000

    fix: delete test code

commit 88598de2f45124e5d543332d54a7ed6675dca63d

Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 05:49:59 2021 +0000

    feat: add step 4

commit 07d25b08f26365bd8b808c069641b2fbbec39e15
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 05:06:27 2021 +0000

    feat: add method 'get_nearest_node' for step 4

commit ea21d9465577de686dbb285ee168e3d5ac6fd919
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 05:01:07 2021 +0000

    fix: fix complexity issues, delete 'WeightCalculate.py'

commit efbab6803c80bf8942e88cb3c813da695f091cb8
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 02:14:33 2021 +0000

    feature: shortest path with naismith's rules

commit 0df2a91cc8d8a008f78017f312e0737bde139c54
Author: linshengze <linshengze980605@gmail.com>
Date: Mon Jan 4 00:28:12 2021 +0000

    feature: calculate weight in both direction

commit 5fe3a208aa092bf0f334fbb7fca461e6b7424dc1

Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Dec 28 17:57:41 2020 +0000

    fix: updated PEP8 formatting throughout.

commit 5a135d4cc8bb73977153e62d1d4e4057c292e569
Merge: f0225d7 0e2e556
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Dec 28 13:36:30 2020 +0000

    Merge pull request #4 from harrisonluft/hl-branch

    Hl branch

commit 0e2e556b5f8cc4aaf4a10454706295fa302921da
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Dec 28 13:34:13 2020 +0000

    fix: updated variable names and formatting.

commit 7255dbebd620b9ae60ab0fdd73a1abef4ab3c4c2
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Dec 28 13:16:11 2020 +0000

    fix: made formatting and comments clearer.

commit 7731fe5a000a1bad130e89a246dbdcc880652bef
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Dec 28 13:15:41 2020 +0000

    feat: wrapped nearest_itn in a class, integrated into main function.

commit f0225d7fc91bb75beeea7b3bf40bf6cc66e2e0fe
Merge: 93d16f3 32b6822
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Mon Dec 28 12:30:30 2020 +0000

    Merge pull request #3 from harrisonluft/hl-branch

    feat: script to identify closest node in the itn.

commit 32b6822b08884cfe4ce0342ce6681d73890b7140
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Mon Dec 28 12:27:20 2020 +0000

    feat: script to identify closest node in the itn. Not integrated into main yet.

commit 93d16f3f7f42a6dc79357b09ee0ae784f05f0586
Merge: 101be01 d455b0a
Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>
Date: Sun Dec 27 11:27:38 2020 +0000

    Merge pull request #2 from harrisonluft/hl-branch

    Hl branch

commit d455b0aabf6e1ef193d3193697f289121c750ef2
Author: Harrison Luft <harrisonluft@gmail.com>
Date: Sun Dec 27 10:30:15 2020 +0000

feat: wrapped raster clipping function in a class and moved plotting functionality into main function.

commit 4b3d3316aa744ca290d35ab703bfb82d5e2581b7

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Sun Dec 27 09:40:21 2020 +0000

feat: identify and plot highest point in 5km buffer of input point.

commit 6bcd3c41378f9bc4e5a051926123ff9ac361d69d

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Sun Dec 27 08:22:53 2020 +0000

fix: updated capitalization and comments.

commit 101be01ee184844b1b7a4dffe6826bc50c3c7049

Merge: 1e19902 779d3f9

Author: harrisonluft <72804230+harrisonluft@users.noreply.github.com>

Date: Sat Dec 26 19:47:32 2020 +0000

Merge pull request #1 from harrisonluft/hl-branch

feat: added 5km buffer around input point, yet to add highest point.

commit 779d3f9dc357b65c090c5d1de880e3d674fbbaea

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Sat Dec 26 19:44:48 2020 +0000

feat: added 5km buffer around input point, yet to add highest point.

commit 1e19902c5fd3077d89a629128cd8c512a758e03a

Author: Harrison Luft <harrisonluft@gmail.com>

Date: Thu Dec 17 17:35:54 2020 +0000

    feat: starting commit for assignment 2, main and MBR class