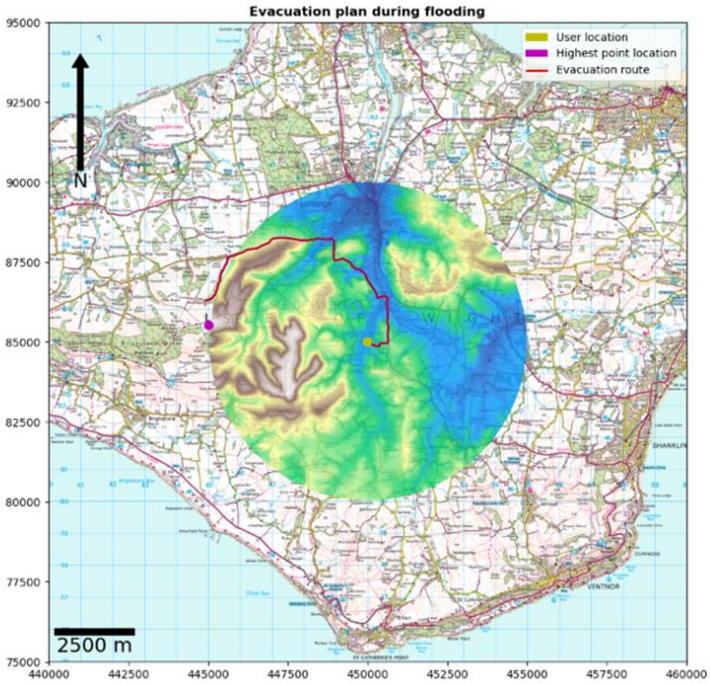
Flood Emergency Planning

Report for the CEGE0096: 2nd Assignment



Megalodon Group

Sijie Cheng
ucessj2
ucessj2@ucl.ac.uk

Nina Moiseeva
ucesois
ucesois@ucl.ac.uk

Pratibha Patel
ucesatb
ucesatb@ucl.ac.uk

Nijun Jiang
zcfbnji
zcfbnji@ucl.ac.uk

Introduction

Software, developed during realization of an Assignment 2 for CEGE 0096: Geospatial Programming aimed to provide for a user an emergency evacuation plan in a case of flooding.

Software step by step implements the following:

- Accepts information about the user's location as input.
- Finds the highest point in a given radius (that is, the point to which it is most reliable to evacuate).
- Finds the nodes of the road graph that are located closest to the user's location and to the highest point.
- Finds the shortest evacuation route between mentioned above nodes according to the Naismith's rule.
- Displays the route on the map for user orientation.

The achieved results of Assignment 2 for CEGE 0096: Geospatial Programming are flagged in the table below.

No.	Task Description	Status
1	User Input	Done
2	Highest Point Identification	Done
3	Nearest Integrated Transport Network	Done
4	Shortest Path	Done
5	Map Plotting	Done
6	Extend the Region	Done
7	Creativity marks are available under certain conditions for adding features that have not been specified	Several improvements made: 7.1 Programme uses a GUI. 7.2. Programme searches highest point within a user-defined radius.

For the code OOP was used (each task is presented in a separate Class with its Attributes and Methods), error handling functionality was used where appropriate (within the user input). PEP 8 style was implemented for the whole code. Commenting on the code was carried out according to the principle of rational necessity. The Git Log is provided as an appendix to the report.

Work on main tasks (1-5) was distributed within a team as follows:

- Task 1 – Jiang Nijun, Pratibha Patel.
- Task 2 – Jiang Nijun.
- Task 3 – Sijie Cheng.
- Task 4 – Pratibha Patel, Sijie Cheng.
- Task 5 – Nina Moiseeva.

During the Assignment 2 Team worked via GIT repository located by a link: https://github.com/CEGE0096/flood-emergency-planning-megalodo

At first, the team members worked separately on their parts in separate branches of Git with cross-validation of the code. Upon completion of the integral parts, the code was undergoing general processing. The software was developed with the usage of PyCharm and Jupyter Notebook with Python 3.8. compilator with predefined Geospatial environment. Throughout the work with the code, the team members coordinated with each other. The development of the code lasted 28 days.

At the moment, the main limitation when using the software may be the performance of the device on which the program will run.

Project Description

Software was developed with the usage of Python compilator 3.8 and Geospatial environment, which could be defined as follows:

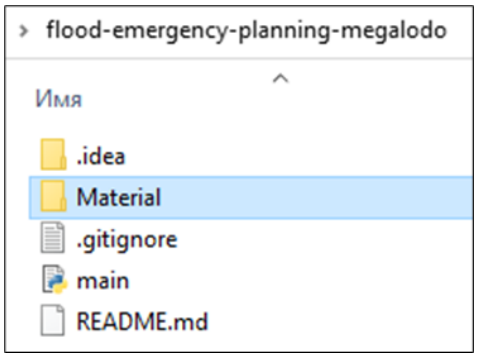
```
name: geospatial
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.8.5
  - cartopy=0.18.0
  - proj=7.1.1
  - numpy=1.19.1
  - shapely=1.7.1
  - rtree=0.9.4
  - pip:
    - jupyter==1.0.0
    - geopandas==0.8.1
    - descartes==1.1.0
    - matplotlib==3.3.3
    - rasterio==1.1.8
    - networkx==2.5
    - pandas==1.1.4
    - pyproj==3.0.0
    - scipy==1.5.4
```

Software contains main.py file to run. To run a software user could use one of following:

- PyCharm.
- Anaconda Prompt.

To use a software user needs to follow the several steps:

- Download a provided archive with a program: flood-emergency-planning-megalodo.zip and unzip archive in a folder on a computer.
- User needs to put a Material folder inside a flood-emergency-planning-megalodo folder, as shown on a picture below.



- User needs to run main.py.
- User can choose between PyCharm or Anaconda Prompt.

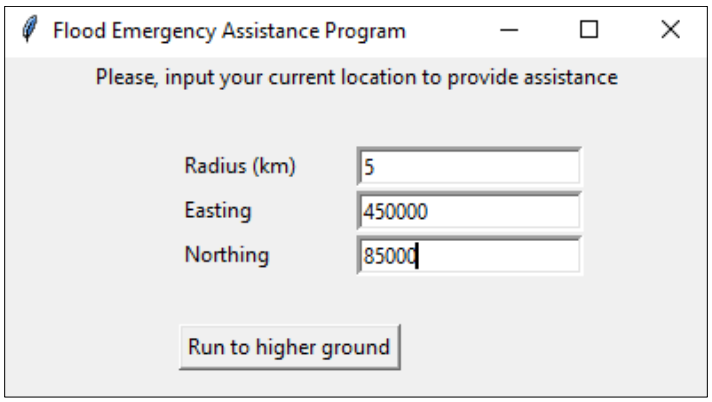
To run a chosen file in PyCharm user must:

- Open PyCharm.
- Press File – Open File from Project and define a path to an unzipped project folder (flood-emergency-planning-megalodo.zip).
- With Edit Configuration choose to run main.py.
- Press a Run button.

To run a chosen file in Anaconda Prompt user must:

- Open Anaconda Prompt.
- Activate Geospatial environment.
- Type a name of a file - main.py and press Enter.

When the user will run a software a GUI dialogue menu will appear. User will need to input 3 parameters – radius in kilometres, easting in meters and northing in meters. GUI dialogue menu is shown on a picture below.



Software

Task 1: User Input

This programme uses a GUI (from package tkinter) as an additional feature to enhance the user input functionality. Some error handling functions are introduced into the GUI class, such as `assert` to constraint the input location range and `try... except ValueError` to handle unexpected values. If the package `tkinter` fails to work, please use commented `UserLocation` class as an alternative input method and activate the class `user_location = UserLocation` and modify other class inputs according to comments.

The section with error handling:

```
try:
    global east1, north1, radius
    east1 = float(self.var1.get())
    north1 = float(self.var2.get())
    radius = float(self.var0.get())
    assert 425000 < east1 < 470000 and 75000 < north1 < 100000, 'please input a location
within the box (430000, 80000)  and (465000, 95000), British National Grid coordinate
(easting and northing)'
    print(east1, north1, radius)

    my_w.t1.delete(0, 'end')
    # remove input everytime after press button
    my_w.t2.delete(0, 'end')
    window.destroy()

except ValueError:
    raise ValueError("Please input integer/float values into both easting and northing
entry")
```

Task 2: Highest Point Identification

In this task, the method of finding the highest point, in short, is transforming the raster file into single points with elevation values within a pandas data frame and looping over the data frame to find the closest high elevation inside a certain search radius. The class method find_high utilises the fact that the ASC raster file has a shape of 9000x5000,which means each raster has a shape of 5x5, and raster.xy can return the representative points of rasters. By rounding the input user location to the nearest 5 (`nearest_easting=5 * round(east1/ 5)`, `nearest_northing=5 * round(north1/ 5)`), we can obtain the raster positions (`east_raster_position and north_raster_position`) ,that can be considered as the centre of the elevation searching, in the matrix-like 9000x5000 data structure.

The `saa` , search area adjust, the radius of the searching in unit of raster number. Input radius has a unit of kilometer which is transformed into raster number by using `saa=int(radius * 1000 / 5`.

Transform raster into dataframe:

```
#minimum bounding box for search area (a circle with radius x)
xmin, ymax = self.elevation_raster.xy(self.elevation_raster.height -
north_raster_position - self.saa,
                        east_raster_position - self.saa)
xmax, ymin = self.elevation_raster.xy(self.elevation_raster.height -
north_raster_position + self.saa,
                        east_raster_position + self.saa)
x = np.linspace(xmin, xmax, self.saa * 2) # saa*2 = diameter of search radius
y = np.linspace(ymax, ymin, self.saa * 2)

# create 2D arrays for data frame
xs, ys = np.meshgrid(x, y)
zs = self.elevation_raster.read(1, window=Window.from_slices(
        (row_start, row_end),
        (column_start, column_end)))

raster_data = {"X": pd.Series(xs.ravel()), #X -> easting
        "Y": pd.Series(ys.ravel()), #Y -> northing
        "Elevation": pd.Series(zs.ravel())}

df = pd.DataFrame(data=raster_data)
```

Check null values and apply trigonometry to find the high ground:

```
# apply trigonometry to select elevation values within xkm radius and form a dataframe
called df_circle
df_circle = df.drop(df[((df.X - self.easting) ** 2 + (
            df.Y - self.northing) ** 2) ** 0.5 > self.radius * 1000].index)
self.df_circle = df_circle
# reset index to properly loop the dataframe
df_circle = df_circle.reset_index()
```

```
# drop any null values if exists
df_after_null = df_circle.dropna()

if df_circle.shape == df_after_null.shape:
    print('No Null Values detected')
else:
    print('Null Values detected, proceed with caution')

# Find maximum elevation value and return the pixel's central coordinates
highest_point_coordinate =
Point(round(df_circle.iloc[df_circle['Elevation'].idxmax()][1]),

round(df_circle.iloc[df_circle['Elevation'].idxmax()][2]))
```

Task 3: Nearest Integrated Transport Network

In this task, we need to use the given itn file to find the user and the path node of the highest point. So in the init function, we need to get the coordinates of the user, the highest point and the elevation data, and create all the attribute data.

```
class Road_Networks():
    def __init__(self, p_location, p_highest, elevation_raster):
        self.p_location = p_location
        self.p_highest = p_highest
        self.elevation_raster = elevation_raster
        self.p_node = None
        self.highest_node = None
        self.s_node_key = None
        self.e_node_key = None
        self.road_links = None
        self.shortest_path_gpd = None
```

In the nearest_node function, first read all the road coordinates, second create an rtree with all the coordinates, traverse the rtree and find the nearest road_node to the user and the highest point, finally find the path where the node is located and get the road start and end nodes.

```
def find_node(self, roadlist, node,str):
    for i in roadlist:
        for j in self.road_links[i]["coords"]:
            if j == node:
                key = self.road_links[i][str]
    return key

def find_nearest(self, idx, loc, list):
    for i in idx.nearest(loc, 1):
        node = list[i]
    return node
```

```
def nearest_node(self):
    # Read itn network information
    itn_path = "Material/itn/solent_itn.json"
    with open(itn_path, "r") as file:
        itn_json = json.load(file)
    # Obtain road node and route information
    road_nodes = itn_json['roadnodes']
    self.road_links = itn_json['roadlinks']
    road_links_keys_list = list(self.road_links.keys())
    road_list = []
    # Get path coordinates
    for i in road_nodes:
        road_list.append(road_nodes[i]['coords'])
    # Use rtree to find the nearest node
    idx = index.Index()
    for i, coords in enumerate(road_list):
        pose = coords + coords
        my_index = MyIndex(pose, 100)
        idx.insert(i, my_index.localPos, obj=my_index)
    # Find the node closest to the user's coordinates
    self.p_node = self.find_nearest(idx, self.p_location, road_list)
    # Find the closest node to the highest point
    self.highest_node = self.find_nearest(idx, self.p_highest, road_list)
    # Find the starting point of the road closest to the user point
    self.s_node_key = self.find_node(road_links_keys_list,self.p_node,"start")
    # Find the ending point of the road closest to the highest point
    self.e_node_key = self.find_node(road_links_keys_list, self.highest_node, "end")
```

Task 4: Shortest Path

To calculate the shortest path, method shortest_path() is created in the class Road_Networks(). In the shortest_path function, the bidirectional network graph is first created, followed by the creation of network nodes and edges and the use of Naismith's rule to assign road weights. Ultimately, the shortest path is calculated.

```
def xy_transform(self, elevation_array, transformer, x, y):
    ele = elevation_array[rowcol(transformer, x, y)]
    return ele


def uphill_time(self, coords, transformer, elevation_array):
    # Calculate the value of elevation increase
    uphill_elevation = 0
    i = 0
    x, y = coords[0]
    s_elevation = self.xy_transform(elevation_array, transformer, x, y)
    for point in coords[1:]:
        x, y = point
        elevation = self.xy_transform(elevation_array, transformer, x, y)
        if elevation > s_elevation:
            # Cumulative increase in meters
            ele_increase = elevation - s_elevation
```

```
            uphill_elevation = ele_increase + uphill_elevation
        s_elevation = elevation
        i += 1
    # an additional minute is added for every 10 meters of climb
    uph_time = uphill_elevation / 10
    return uph_time

def shortest_path(self):
    # Create a bidirectional graph
    road_network = nx.DiGraph()
    elevation_array = self.elevation_raster.read(1)
    # Get the affine matrix
    transform = self.elevation_raster.transform
    for link in self.road_links:
        road_length = self.road_links[link]['length']
        road_coordinates = self.road_links[link]['coords']
        # Uphill time calculation
        additional_minute = self.uphill_time(road_coordinates, transform,
elevation_array)
        # Time spent walking at a constant speed
        initial_minute = road_length * 0.012
        naismith_time = initial_minute + additional_minute
        road_network.add_edge(self.road_links[link]['start'],
                              self.road_links[link]['end'],
                              fid=link, length=self.road_links[link]['length'],
weight=naismith_time)
        # Because the length of the back and forth uphill
        # is inconsistent, so calculate the reverse time weight
        additional_minute = self.uphill_time(road_coordinates[-1:], transform,
elevation_array)
        naismith_time = initial_minute + additional_minute
        road_network.add_edge(self.road_links[link]['end'],
                              self.road_links[link]['start'],
                              fid=link,
                              length=self.road_links[link]['length'],
weight=naismith_time)
    # Use an algorithm to calculate the shortest path
    path = nx.dijkstra_path(road_network, source=self.s_node_key, target=self.e_node_key,
weight="weight")
    links = []
    geom = []
    first_node = path[0]
    for node in path[1:]:
        link_fid = road_network.edges[first_node, node]['fid']
        links.append(link_fid)
        geom.append(LineString(self.road_links[link_fid]['coords']))
        first_node = node
    # Get the shortest path
    shortest_path_gpd = gpd.GeoDataFrame({'fid': links, 'geometry': geom})
    self.shortest_path_gpd = shortest_path_gpd
```

Task 5: Extend the Region

We implemented the code section (in task Highest Point Identification, class method `def find_high(self)`) that ensures that users can be within 5km from the edge of the raster file when looking for high ground.

Code section:

```
row_start = np.array([self.elevation_raster.height - north_raster_position - self.saa])
row_start[row_start[0] < 0] = 0
row_start = int(row_start)

row_end = np.array([self.elevation_raster.height - north_raster_position + self.saa])
row_end[row_end[0] > self.elevation_raster.height] = self.elevation_raster.height
row_end = int(row_end)

column_start = np.array([east_raster_position - self.saa])
column_start[column_start[0] < 0] = 0
column_start = int(column_start)

column_end = np.array([east_raster_position + self.saa])
column_end[column_end[0] > self.elevation_raster.width] = self.elevation_raster.width
column_end = int(column_end)
```

This section constrains the row and column number of the searching area. The maximum number cannot exceed the raster upper limits, and the minimum number cannot be lower than 0 (negative columns or rows do not exist). saa = search area adjust, see task 2 for further explanations.

Task 6: Map Plotting

Code for map plotting located in `class Plot()` and method `def plot(self)`.

All necessary raster layer and vector layers were plotted together in a single map.

Raster layers are:

- Background topographic map (originally presented as a single band raster in TIFF format with 8bit integer values with a colormap). For background raster colormap was read and used.
- Elevation raster (originally presented as a single band raster in ASCII format with float values). For elevation raster “terrain” colormap was used and raster was cropped by a 5 km radius from a user point.

Vector layers are:

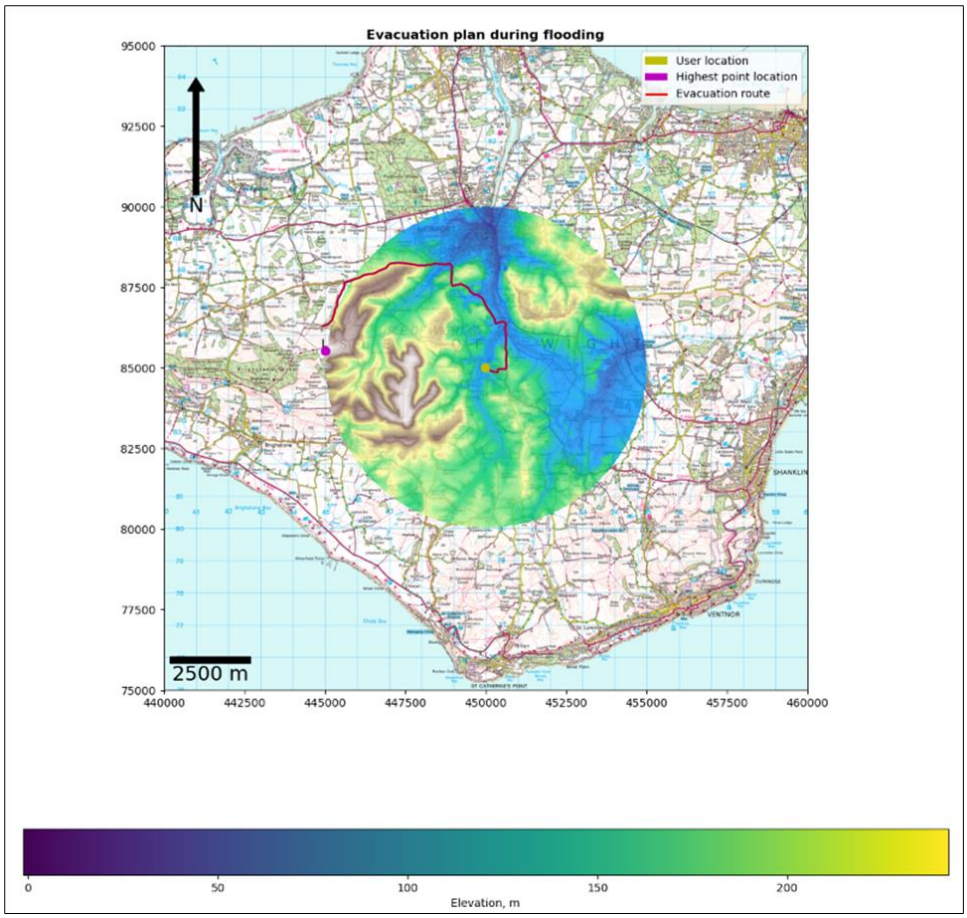
- User location point (user input of latitude and longitude of his position).
- Highest point location (highest point, searched within a radius, defined by user input).
- Evacuation route (route, calculated in method `def nearest_node(self)`).

Map has a size 20×20 km and has the following elements of design and orientation:

- Scale bar.

- North arrow.
- Axis values (used longitude and latitude in meters ESPG 27700).
- Legend for vector data.
- Separate legend for elevation (in meters).
- Title.

View of a final map with all elements is provided below (for point with coordinates 450000 E 85000 N and search radius 5 km).



Code part for plotting with comments is provided below.

```
class Plot:
    def __init__(self, radius, ue, un, p_n, h_n, s_p, elevation_raster, hc):
        self.elevation_raster = elevation_raster
        self.radius = radius
        self.ue = ue
        self.un = un
        self.p_n = p_n
        self.h_n = h_n
        self.s_p = s_p
        self.hc = hc
```

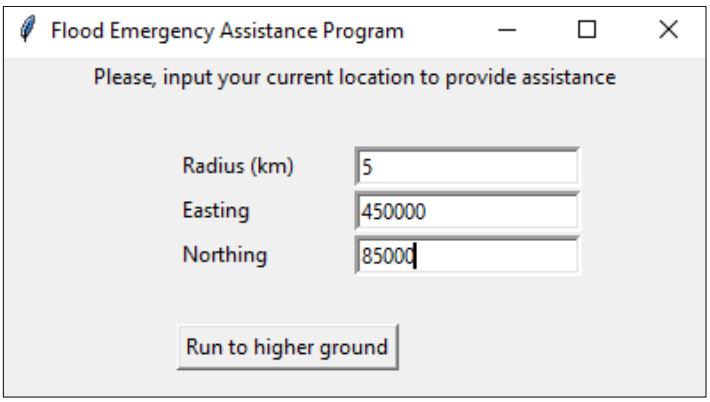
```
    def plot(self):
        # Defining user point, highest point and shortest path
        s_p = self.s_p
        u_location = Point(self.un, self.ue)
        user_buffer = u_location.buffer(5000)
        user_location_c = Point(self.ue, self.un).buffer(150)
        user_e, user_n = user_location_c.exterior.xy
        hc_polygon = self.hc.buffer(150)
        h_e, h_n = hc_polygon.exterior.xy

        # Cropping elevation raster by a circle around user point
        xy = []
        coords = list(user_buffer.exterior.coords)
        for i in coords:
            xy.append([i[1], i[0]])
        features1 = warp.transform_geom(
            {'init': 'EPSG:27700'},
            self.elevation_raster.crs,
            {"type": "Polygon",
             "coordinates": [xy]})

        ele, out_transform = mask.mask(self.elevation_raster,
                                       [features1], filled=False, crop=False, pad=False)

        # Opening background map and defining its colormap and bounds
        background = rasterio.open(str('Material/background/raster-50k_2724246.tif'))
        back_array = background.read(1)
        palette = np.array([value for key, value in background.colormap(1).items()])
        background_image = palette[back_array]
        bounds = background.bounds
        extent = [bounds.left, bounds.right, bounds.bottom, bounds.top]
        fig, ax = plt.subplots(figsize=(15, 15))

        # Plotting background raster
        ax.imshow(background_image,
                  origin='upper',
                  extent=extent,
                  zorder=0)

        # Plotting elevation raster
        rasterio.plot.show(ele,
                           transform=out_transform,
                           ax=ax,
                           zorder=1,
                           alpha=0.8,
                           cmap='terrain',
                           title='Evacuation plan during flooding')

        # Plotting vector data
        self.s_p.plot(ax=ax, edgecolor='blue', linewidth=0.5, zorder=5)
        user_location_df = pd.DataFrame({'name': ['user_point'], 'easting': [self.ue],
'northing': [self.un]})
        user_gdf = gpd.GeoDataFrame(user_location_df,
geometry=gpd.points_from_xy(user_location_df.easting,

user_location_df.northing))
```

```
        new_df = user_gdf.copy()
        new_df['geometry'] = new_df['geometry'].buffer(self.radius * 1000)

        s_p.plot(ax=ax, edgecolor='red', facecolor='none', linewidth=2, label='Evacuation
route')
        plt.fill(user_e, user_n, 'y', label='User location', zorder=2)
        plt.fill(h_e, h_n, 'm', label='Highest point location', zorder=2)

        # Providing a plotting framework and setting the extent limits
        ylim_mi = self.un - 10000
        ylim_mx = self.un + 10000
        xlim_mi = self.ue - 10000
        xlim_mx = self.ue + 10000
        plt.ylim(max(0, ylim_mi), min(100000, ylim_mx))
        plt.xlim(max(0, xlim_mi), min(470000, xlim_mx))

        # Adding legend
        image_hidden = ax.imshow(self.elevation_raster.read(1))
        fig.colorbar(image_hidden, orientation='horizontal', label='Elevation, m')

        # Adding scalebar
        fontprops = fm.FontProperties(size=18)
        scalebar = AnchoredSizeBar(ax.transData,
                                   2500, '2500 m', 'lower left',
                                   pad=0.2,
                                   color='black',
                                   frameon=False,
                                   size_vertical=200,
                                   fontproperties=fontprops)

        # Adding north arrow
        x, y, arrow_length = 0.05, 0.95, 0.2
        ax.annotate('N', xy=(x, y), xytext=(x, y - arrow_length),
                    arrowprops=dict(facecolor='black', width=5, headwidth=15),
                    ha='center', va='center', fontsize=18,
                    xycoords=ax.transAxes)

        # Plotting all map elements on a single map
        ax = matplotlib.pyplot.gca()
        ax.add_artist(scalebar)
        plt.legend()
        plt.show()
```

Task 7: Creativity

7.1 GUI

For creativity part, GUI was created. When the user will run a software a GUI dialogue menu will appear. User will need to input 3 parameters – radius in kilometres, easting in meters and northing in meters. GUI dialogue menu is shown on a picture below.

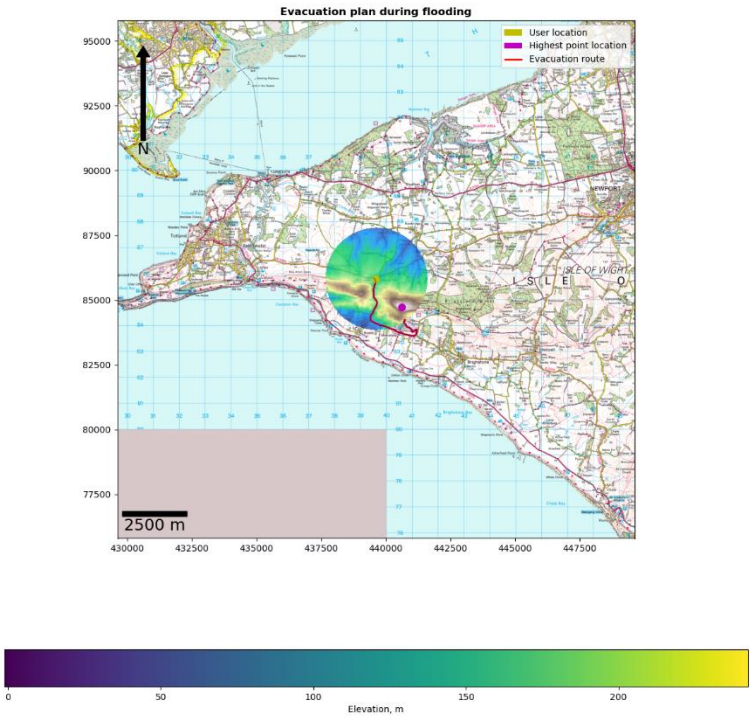



```
class GUI:

    def __init__(self, w):
        self.lbl0 = Label(w, text='Radius (km)')
        self.lbl1 = Label(w, text='Easting')
        self.lbl2 = Label(w, text='Northing')
        self.ibl3 = Label(w, text='Please, input your current location to provide
assistance').pack()
        self.var0 = tk.StringVar("")
        self.var1 = tk.StringVar("")
        self.var2 = tk.StringVar("")
        self.t0 = Entry(bd=3, textvariable=self.var0)
        self.t1 = Entry(bd=3, textvariable=self.var1)
        self.t2 = Entry(bd=3, textvariable=self.var2)
        self.lbl0.place(x=100, y=50)
        self.lbl1.place(x=100, y=75)
        self.t1.place(x=200, y=75)
        self.t0.place(x=200, y=50)
        self.lbl2.place(x=100, y=100)
        self.t2.place(x=200, y=100)
        self.b1 = Button(w, text='Run to higher ground', command=self.add)
        self.b1.place(x=100, y=150)

    def add(self):
        try:
            global east1, north1, radius
            east1 = float(self.var1.get())
            north1 = float(self.var2.get())
            radius = float(self.var0.get())
            assert 425000 < east1 < 470000 and 75000 < north1 < 100000, 'please input a
location within the box (430000, 80000)  and (465000, 95000), British National Grid
coordinate (easting and northing)'
            print(east1, north1, radius)

            my_w.t1.delete(0, 'end')
            # remove input everytime after press button
```

```
            my_w.t2.delete(0, 'end')
            window.destroy()

        except ValueError:
            raise ValueError("Please input integer/float values into both easting and
northing entry")
```

7.2 Modifiable Search Area

The code inside the class `class find_high` and the GUI class enable the user to enter a customized search radius, which should be around or less than 5 km, to avoid the memory space issue.



Git Log

commit da8c948d36495da4df120257c4a76b052141a25d

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Mon Jan 10 14:30:27 2022 +0000

    fix: replace comments

commit 897fb9a7c5ca129bed9e2c577cda8bf7f756cb88

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Mon Jan 10 14:30:06 2022 +0000

    fix: replace misplaced comments

commit 0c3c44d7d7458bebc6642d5eb60f843993555bc2

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Mon Jan 10 14:14:16 2022 +0000

    fix: remove unnecessary print

commit 3f2fe1daa869baec751f19cc2109736fa8423274

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Mon Jan 10 14:13:56 2022 +0000

    fix: remove unnecessary print functions

commit 344e6364e1dd2c4c20ee27857e9807d8b36b8bd3

Author: ucesois <ucesois@ucl.ac.uk>

Date: Sun Jan 9 13:32:16 2022 -0500

    +PEP8 and structured import

commit 29789f3ad1887aee69fac37af6f8f6e95e40b4ff
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Jan 9 13:28:23 2022 -0500

    Last small change inside Plotting, Plotting regarded as finalised

commit e1707b7316804bd518895e690f108f28a79b2dc9
Merge: 32102bd c1f5641
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Jan 9 13:09:46 2022 -0500

    Merge remote-tracking branch 'origin/main'

commit 32102bd39229dc41348dff565debf61020fcf31d
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sun Jan 9 13:09:17 2022 -0500

    Small changes inside Plotting + comments for Ploting

commit c1f5641f21c51841cda4e7be194a303e0dcee9c6
Author: pratibha1708 <92760098+pratibha1708@users.noreply.github.com>
Date: Sun Jan 9 17:33:30 2022 +0000

    updated calorie calculator function

commit 69ff70df3ce6f7f96902f0fb6fccd74f9257164c
Author: SijieCheng <93685301+SijieCheng@users.noreply.github.com>
Date: Sun Jan 9 16:49:53 2022 +0000

fix: add more comments

commit f59f8f7cb444b76fbda540aac9cd26bcadba8fdd

Author: SijieCheng <93685301+SijieCheng@users.noreply.github.com>

Date: Sun Jan 9 16:38:20 2022 +0000

fix: add comments

commit dd64e80520081d35fa876afd9bd265ffb59d1de5

Author: SijieCheng <93685301+SijieCheng@users.noreply.github.com>

Date: Sun Jan 9 16:20:36 2022 +0000

fix: adjust the font size and layer order

commit 845cfde3ea867ccddab9d30357728be952d8bf08

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Sat Jan 8 21:18:03 2022 +0000

feat: graph label

commit 52112eed952832707b15c56e74aacae660629739

Author: ucesois <ucesois@ucl.ac.uk>

Date: Sat Jan 8 15:36:56 2022 -0500

Plotting upgraded (+ legend for raster, colormap for background returned, extra map is deleted, several design adds)

commit 51cf0e102ed7ef248a35c49a5a54373832a4f714

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Sat Jan 8 17:11:22 2022 +0000

feat: basic point/line plotting

commit a0c6ffd32800612633e1df1f15a1d277d93a1d23
Author: ucesois <ucesois@ucl.ac.uk>
Date: Sat Jan 8 03:04:02 2022 -0500

User location, the highest point and shortest path are called in plot class

commit 064f0e32fb66f5e8c66d5b1ecc15a85d8cd2631e
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date: Fri Jan 7 21:55:26 2022 +0000

fix: additional comments, GUI fail-safe

commit 4e5a2fc74ed3b9c2b65bd72a11958dd79cbcfda3
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date: Fri Jan 7 16:59:32 2022 +0000

feat: GUI feature integrated

commit 8ea1f22a083cb33f6974d44685d1f95c2d7fd15e
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date: Fri Jan 7 12:44:20 2022 +0000

fix: df_circle attribute

commit 0aad0f97903735ceb2acc4eda93203eff461ed54
Author: pratibha1708 <92760098+pratibha1708@users.noreply.github.com>
Date: Fri Jan 7 12:04:17 2022 +0000

```
    Added function calories_calculator

commit 73dcd2922aa549578f1a906cd71d96e0b7df79d4
Author: pratibha1708 <92760098+pratibha1708@users.noreply.github.com>
Date:   Fri Jan 7 11:34:23 2022 +0000

    Added class GUI

commit bd6fb85c1d87b1fb81ea6704d5db66bd6a61f1e0
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Fri Jan 7 11:31:27 2022 +0000

    fix: class attributes for plotting

commit 6c507829d9e4e8e8eb0ab27ef6d3d33e4af43a4f
Author: pratibha1708 <92760098+pratibha1708@users.noreply.github.com>
Date:   Fri Jan 7 10:42:46 2022 +0000

    Updated task -4

commit 305c62fcc7ebac01cf4df0f3cc7708a88412fdb2
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Fri Jan 7 00:53:20 2022 +0000

    fix: nearest nodes easting and northing reversed

commit 6ee3848c0b1e8bc3497082c754070a7ba7ab12cf
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Thu Jan 6 19:28:01 2022 +0000
```

```
    fix: read asc file only once across project

    fix: read asc file only once across project

commit 2dd2ab10d0a15486e1108d3cbc7424ac0258c932
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Thu Jan 6 19:26:53 2022 +0000

    Add files via upload

    fix: read asc raster file once across the script

commit ae5edbcd990d7c6a2734019b89de8a55b4579928
Merge: bf965bb 78283d6
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Thu Jan 6 11:16:53 2022 +0000

    Merge pull request #1 from CEGE0096/task2

    merge task 2 branch

commit 78283d6509895e54cb265b46ebf89679f22dcf3c
Merge: 9425c12 bf965bb
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date:   Thu Jan 6 11:16:20 2022 +0000

    Merge branch 'main' into task2

commit 9425c1260032d103807646f5aed8501978d51fac
```

Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>

Date: Thu Jan 6 11:05:25 2022 +0000

  Add files via upload

commit bf965bb8a0fee138648e3cc5d9f7266cb7d889e0

Author: SijieCheng <93685301+SijieCheng@users.noreply.github.com>

Date: Wed Jan 5 14:21:35 2022 +0000

  merge:task1234

commit 12041c97cc5e5d85ac17084ea41069309179cf7d

Author: NijunJiang <suzhou.nijun@gmail.com>

Date: Sun Jan 2 23:24:07 2022 +0000

  feat: modifiable search radius

commit 8ebcfd93de3925c14b0afbe552baea399fb1be9a

Author: ucesois <ucesois@ucl.ac.uk>

Date: Fri Dec 31 19:08:59 2021 +0000

  Added initial map plotting (background and elevation raster, island border, raster legend, arrow and scale)

commit 35ccab5b1faa5b6f1631d5d6c923b3585deac18e

Author: NijunJiang <suzhou.nijun@gmail.com>

Date: Thu Dec 30 12:35:12 2021 +0000

  extend region, make task1 and 2 adaptable to 6

commit 7683e685c5551ddf747a5be0bbf379b28ac36884
Author: NijunJiang <suzhou.nijun@gmail.com>
Date: Wed Dec 29 12:19:48 2021 +0000

    making task 1 and task 2 OPP

commit 7c7f9c3565dae7ef089c4244d7608b5a70ddb731
Author: NijunJiang <suzhou.nijun@gmail.com>
Date: Sun Dec 26 16:49:22 2021 +0000

    data cleaning

commit 592195d51ec02c1fc56ca73530e729b271869a40
Author: NijunJiang <suzhou.nijun@gmail.com>
Date: Sun Dec 26 14:27:42 2021 +0000

    create task2 branch

commit 72a81221be0513e1acc7dc8e768c381da9c6cee2
Author: ucesois <ucesois@ucl.ac.uk>
Date: Mon Dec 20 19:12:27 2021 +0000

    Applied PEP8, Commit to test that commits can be pushed

commit 8257d0c7da4e9f682f0a1dbf5627239894e15ace
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date: Sat Dec 18 19:07:46 2021 +0000

    part1_fixing and form a point Obj of user location

commit 400f64cbeb57408ada5c55dd929a7050225326a4
Author: NijunJiang <92638995+NijunJiang@users.noreply.github.com>
Date: Sat Dec 18 18:52:23 2021 +0000

part1_from_edge

commit 1b90ec947b25d04fe87b7e4ff39a8a3c0f43e530
Merge: 5953815 d185ebc
Author: NijunJiang <suzhou.nijun@gmail.com>
Date: Sun Dec 12 16:17:41 2021 +0000

Merge branch 'main' of https://github.com/CEGE0096/flood-emergency-planning-megalodo

commit 5953815f76e4f1953b6bb7075d5124acf323ad13
Author: NijunJiang <suzhou.nijun@gmail.com>
Date: Sun Dec 12 15:48:22 2021 +0000

user input

commit d185ebcdcb870f96d4e1f5c57602f5bfe373138f
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date: Wed Dec 8 11:19:09 2021 +0000

Initial commit