LENOIR Rachel, HUBERT Margot, ROMBAUT Doriane, CHATELUS Eolia

SCAN 71

Separation into MVC

Model

class SeaLevel:

def load_data_sea_level(file_sea_level)**:**

"""

Load the data from a file containing the average sea level on Earth for all past years since 2015. Convert them into a dictionary dico_sea_level where each key corresponds to a year and the associated value corresponds to the average sea elevation.

Parameters:

----

file_sea_level: str

name of the csv file containing the data for the sea level elevation for the past 10 years

Returns:

----

dico_sea_level: dict

associating each year to the average sea level

"""

def compute_sea_level_1 (year)**:**

"""

Controller

class Controller:

def set_views (self, mainview, secondaryview)**:**

"""

Initializes the attributes mainview and secondaryview using the parameters which are respectively an instance of the MainView class and an instance of the SecondaryView class.

Returns

None.

"""

def count_refugees(self)**:**

"""

Compute the number of climatic refugees using the function compute_refugees from the class ElevationData, according to the year chosen by the user.

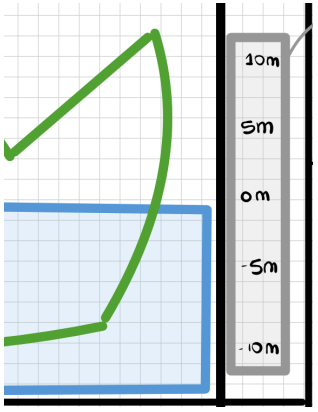
Returns

----

nb_refugees : string

View

class ProfileView :



def country_profile (self, frame, width, height, dico_per_long, sea_level) :

"""

Create the profile view in the main window of the interface. The sea is placed at the bottom in blue,

Height of the canva = sea height + maximum elevation average +1, Width of the canva = nb longitudes + 2

Place a point for each longitude with the associated average latitude and draw lines between neighbor points to create the profile shape of the country.

class CoordinateConverter:

def canvas_to_geo(self, x, y, canvas, base_image, lats, lons, pan_x, pan_y, zoom, lat_indices, lon_indices):

"""

Converts coordinates (x, y) of a point on the canva with the map of the Earth into real coordinates on Earth (latitude, longitude).

```
    Compute the average sea level elevation
    for a given year (int) in the future using
    the model prediction from the GIEC
    scenario 1.

    Parameter:
    ----------
    year: int
    above 2025 corresponding to a year in
    the future at which we want to compute
    the sea level according to the predictions
    Returns:
    ----------
    sea_level:  float
    value of the average sea level elevation
    for the given year
    """

def retrieve_sea_level (year,
dico_sea_level):
    """
    Retrieve the sea level for a year given as
    a parameter. If the value of the sea level
    is already in the dictionary
    dico_sea_level, no computation is
    needed, the value is retrieved from the
    dictionary. If the value of the given year
    is not in dico_sea_level, compute the sea
    level  using the function
    compute_sea_level_i according to the
    scenario chosen by the user.

    Parameters:
```

```
        number of climatic refugees in the
    form 'nb_refugees' millions.
    """

def top_or_side(self):
    """
     Define if we want to display the profile
    view or the global one.

    Returns
    -------
    None.
    """
def create_top_map(self):
    """
     Create a map adapted to the user's
    choice (reuse of a function from
    SecondaryView)

    Returns
    -------
    None.
    """
def create_profile_map(self):
    """
     Prepare and draw the profile view of a
    country for which it is available if the
    user has clicked on it.

    Returns :
    -------
     None
    """
```

```
    frame : canva
        canva in which the profile view is
    displayed
    width : int
        Width of the profile view
     height : int
         Height of the profile view
    dico_per_long : dict
        Dictionary containing the longitudes
    and their associated elevation
    sea_level : float
        Sea level for the concerned year


    Returns
    -------
    None.
    """

def load_sky_image(self):
    """
    Load a background image for the profile
    view.

    Returns
    -------
    None.
    """
def redraw(self):
```

```
    Parameters
     ----------
    x : float
        x coordinate on the canvas.
    y : float
        y coordinate on the canvas.
    canvas: canva
        canva containing the image of the
    Earth (base_image)
    base_image : PIL image
        image generated using the .nc file with
    all points on Earth
    lats : numpy arrays
        array of the latitudes, from the .nc file
    lons : numpy arrays
        array of the longitudes, from the .nc
    file
    pan_x : float
        Horizontal pan offset to draw the
    image
    pan_y : float
        Vertical pan offset to draw the image
    zoom : float
        Zoom scale factor (1.0 = no zoom)
    lat_indices : numpy linspace
        array with regularly spaced values of
    the latitudes
    lon_indices : numpy arrays
        array with regularly spaced values of
    the longitudes

    Returns
     -------
    lat : float
```

----
year: int
year at which we want to retrieve the sea level
scenario: int
GIEC scenario the user chose on the interface to model the evolution of sea level rise

Returns:
----
sea_level : float
value of the sea level elevation for the given year
"""
Algo:
If the year (int) is in dico_sea_level (dict):
Retrieve the measured value of the ocean elevation from the dico_sea_level
Else if the year is not in dico_sea_level: Compute sea level based on year and scenario using the function compute_sea_level_i

class ElevationData:

def create_elevation (selfe):
"""

def get_where_clicked(self):
"""
Retrieve the geographical coordinates (latitude and longitude) of a point from the x and y coordinates of the window

Returns
----
lat : float
Latitude of the concerned point
lon : float
Longitude of the concerned point
"""

def get_sea_level(self, year, scenario):
"""
Retrieve the sea level from the SeaLevel class and its functions

Parameters
----
year : int
Year chosen by the user to visualize
scenario : int
GIEC scenario chosen by the user to visualize

Returns
----
self.sea_level_value : float
Sea level for the corresponding year and scenario
"""

def run(self):
"""
Draw the profile view and adapt it to the window's size

Returns

None.
"""

def on_resize(self, event):
"""
Reinitialize the drawing.
Returns

None.
"""

class MainView:

def create_widgets():
"""
Creates and place the widgets of the main interface (around the map): a scale to allow the user to choose a year, a button to generate the map, a label to indicate if the interface is in profile or top view, a button to exit the profile view, a button to show the number of refugees, a label displaying the number of refugees.. Get the information on the chosen year (scale). Gets the information on chosen zoom (roll of middle mouse button).

Latitude of a point on Earth.
lon : float
Longitude of a point on Earth.
"""

```
    Load the data from a .nc file containing
    the elevation, longitude and altitude of
    all points on Earth (with a precision to
    the 60 arc-minute). Convert the data to
    create a dictionary with the key being an
    elevation and the value a list of tuples
    (latitude, longitude), in degrees, of all
    points being at the given elevation (in
    meters).

    elevation_dico = { “elevation1” : [ [lat1,
    long1], [lat2, long2], ...],
            “elevation2” : [ [lat1, long1],
    [lat2, long2], ...],
             .... ,
          }
    Parameters:
    -------
    self.netcdf_files: string
    corresponding to the name of the .nc file
    containing the data for the latitude,
    longitude and elevation of all points on
    Earth.

    Returns:
    -------
    elevation_dict: dict
    associates each elevation in meters to a
    list of tuples (latitude, longitude) in
    degrees at the given elevation.
    """

def create_polygon(self, csv_file):
    """
```

```
        Run the code to display the interface to
    the user.
    Returns
     -------
     None
      """
```

```
            Returns:
            -------
            None.
             """

def on_scale_change(self, value):
    """
    Adapt the year entered by the user for a
    one that is a multiple of 5 (i.e, 2020,
    2025...).
    Retrieve the sea level (computed in the
    other classes' functions) and display it to
    the user.

     Parameters
     ----------
     value : int
         Year entered by the user

     Returns
      -------
     None.
     """

def increase_scale(self):
    """
    Increases the year on the scale by 5.

     Returns
     -------
     None.
     """
def decrease_scale(self):
```

Load a csv file with coordinates and create a polygon shape from it. *The csv has columns 'latitude' and 'longitude'.* *Parameters* *----------* *fichier_csv : str* *The path to the CSV file that contains the boundary coordinates.* *Returns* *-------* *polygon : shapely.geometry.Polygon* *The polygon created from the coordinates.* *"""* **def test_if_point_in**(self, where_clicked)**:** *"""* *Test if a clicked point is inside the polygon.* *The polygon should already be created before calling this function.* *Parameters* *----------* *where_clicked : tuple* *Coordinates of the point clicked (lat, lon)* *Returns* *-------* *is_inside : bool*		*"""* *Decreases the year on the scale by 5.* *Returns* *-------* *None.* *"""* **def get_ipcc_value**(self): *"""* *Store the user's choice of scenario.* *Returns* *-------* *int* *Chosen scenario* *"""* **def get_user_year**(self): *"""* *Retrieve the year chosen by the user and return it rounded for it to correspond to a step of 5 years.* *Returns* *-------* *int* *Chosen year* *"""* **def show_map**(self):	

```
        True if the point is inside the polygon, False otherwise
    """

def build_dico_per_long (self, sea_level) :
    """
    Reads a csv file of France mainland elevation points and creates a dictionary where each key is a longitude (rounded to 1 decimal) and the value is the average elevation above sea level at that longitude (only if it is above sea level).

    Parameters
    ----------
    sea_level : float
        The reference sea level we want to compare elevation to.

    Returns
    -------
    dico_per_long : dict
        Dictionary of the form {longitude_rounded : avg_elevation_above_sea_level}
        Only includes longitudes where the average elevation is above the sea level.
    """

def compute_refugees (self, year, elevation_year, elevation_2022):
    """
    Compute the number of climatic refugees due to the elevation of sea level.
    """
    Display the map as well as a loading label while waiting.

    Returns
    -------
    None.
    """
def generate_map_canvas(self):
    """
    Create the map of the Earth from the top or side view depending on the user's choice of display.

    Returns
    -------
    None.
    """
def count_refugees(self):
    """
    Retrieve the number of climatic refugees computed in the controller and display an adapted message in the text zone dedicated on the interface n clicking on the button generate_refugees.

    Returns
    -------
    None.
    """
def change_mode_value(self, value):
```

Define the limits of the continents considering a polygon containing the whole continent, including sea borders. *Store the limit coordinates in lists of tuples (lat, long) in order north, east, south and west to have a closed polygon.* *Initialize the number of refugees known in 2022.* *For each continent define an average population density and store it in a dictionary associating a continent name to a density.* *Define a dictionary associating to a continent's name the annual population growth to adjust the average population density according to the year.* *Define the surface in km squared covered by a point on the dictionary eleavtion_dict as the product of one degree in latitude and one in longitude converted in kilometer times 25 since we have a 5° resolution.* *At the year chosen by the user (limited to 500 years after 2022 to avoid unrealistic projections), compute the corresponding population density for each continent.* *If the user chooses a year in the future, for each point in elevation_dict, if the point has been submerged, computes the number of refugees due to sea level rise by multiplying the surface submerged by the population density of the continent.*		""" *Change the value of the exit button and mode text depending on whether the user displays the top or profile view.* *Parameters* *----------* *value : string* *mode of display, 'top' if we see the whole map of the Earth, 'profile' for the profile view of a country.* *Returns* *-------* *None.* """ **def exit_profile_view**(self): """ *Allow to exit the profile view after having displayed the profile view of a country when clicking on the button exit_profile_button.* *Returns* *-------* *None.* """ **class SecondaryView:** **def generate_base_image**(self, base_width, base_height, sea_level)**:**	

To check if a point have been submerged, we check that it is below sea level in the year chosen by the user, but was above sea level in 2022. Then add the refugees due to other climatic events using the function estimate_other_climatic_refugees. *Parameters:* *-------* *year : int* *year chosen by the user on the interface, at which we want to compute the number of climatic refugees* *elevation_year : float* *sea level in meters in the year chosen by the user* *elevation 2022 : float* *sea level elevation in 2022* *Returns:* *-------* *nb_refugees: (int) number of climatic refugees according to the year chosen by the user and the scenario.* *"""* **def estimate_other_climatic_refugees**(self, year): *"""* *Estimate number of climatic refugees due to the climatic events different from sea level rise.* *Parameters* *----------*		*"""* *Generate the base image (PIL.Image) sized (base_width x base_height) showing land and sea colors.* *Uses elevation data from netCDF file to color pixels blue if below sea level, green if above.* *Only regenerates if the water level has changed since last call.* *"""* **def redraw**(self): *"""* *Redraw the base image on the canvas, applying zoom and pan offsets.* *This method handles scaling the base image to fit current zoom and placing it correctly on the canvas based on pan_x and pan_y.* *Returns* *-------* *None.* *"""* **def on_resize**(self, event):	

year : int *The year chosen by the user to compute the number of climatic refugees.* *Returns* *-------* *refugees : int* *Estimated number of additional climatic refugees due to different climate features like floods, heatwaves, drought and wildfire.* *"""*		*"""* *Resize the window according to the zoom chosen by the user with its mouse scroll.* *Recalculate the pan offsets to center the zoomed image within the new canvas size.* *Then redraw the map.* *Returns* *-------* *None.* *"""* **def on_zoom**(self, event): *"""* *Allow to zoom in and out when scrolling with the mouse, centered on the point below the cursor of the mouse.* *Prevents zooming out too much so that the image does not fit in the canvas anymore.* *Returns* *-------* *None.* *"""* **def on_click**(self, event):	

		""" *Check if the user clicked on a country for which a profile view is available. Converts the clicked canvas coordinates into image (pixel) coordinates, taking into account the zoom on the interface and pan. Stores the result as self.x and self.y.* *Then notifies the controller to trigger a generation of the profile view.* *Returns* *-------* *None.* *"""* **def create_map**(self, frame, width, height, sea_level):	

		""" Create and set up the CTkinter canvas with the base image loaded and display it. Also binds resize and zoom events to allow the user to interact with the interface. Parameters ------- frame : container of the canvas containing the map width : float width of the frame height : float height of the frame sea_level : float value of the sea level at the year chosen by the user Returns ------- None. """	