

Universitat Politècnica de València

**School of Engineering in Geodesy,
Cartography and Surveying**

FINAL PROJECT

Desarrollo de aplicaciones SIG

Vicente Ortí Llull

Eliška Vlčková



València 2017

INDEX

1	GOAL	3
2	DESCRIPTION	4
2.1	Interface	4
2.2	Map template.....	5
2.3	Functionality	6
2.3.1	Interface functionality	6
2.3.2	Enabling and disabling widgets	9
2.3.3	Automatic loading data.....	10
2.4	Creating thematic maps	11
2.4.1	Access to the list of layers.....	11
2.4.2	Apply interpolation	11
2.4.3	Enable wather layer.....	12
2.4.4	Draw graphic elements	12
2.4.5	Zooming to selected territory	13
2.4.6	Exporting maps.....	14
2.4.7	Template restart.....	14
3	CODE	15
4	RESULTS	22
5	CONCLUSION	24
6	BIBLIOGRAPHY	25

1 GOAL

The main goal was to create stand-alone tool, what performs creating maps using interpolation algorithm (*Figure 1*). Based on input data, there is several types of maps, what can be generated: actual temperature, maximum temperature, minimum temperature, precipitations, humidity and pressure – both have output option for png format and pdf format. Configuration options let user to display legend, scale, time and logo. Additionally, the user is allowed to define supplementary elements of the topographic background. For example, the administrative boundaries (region, province, etc.) and other backgrounds (hillshade, rivers, cities, or meteorological data input data) can be selected. The last option is the „zoom option“ on the given territory, which is mentioned in the last tab of the interface.

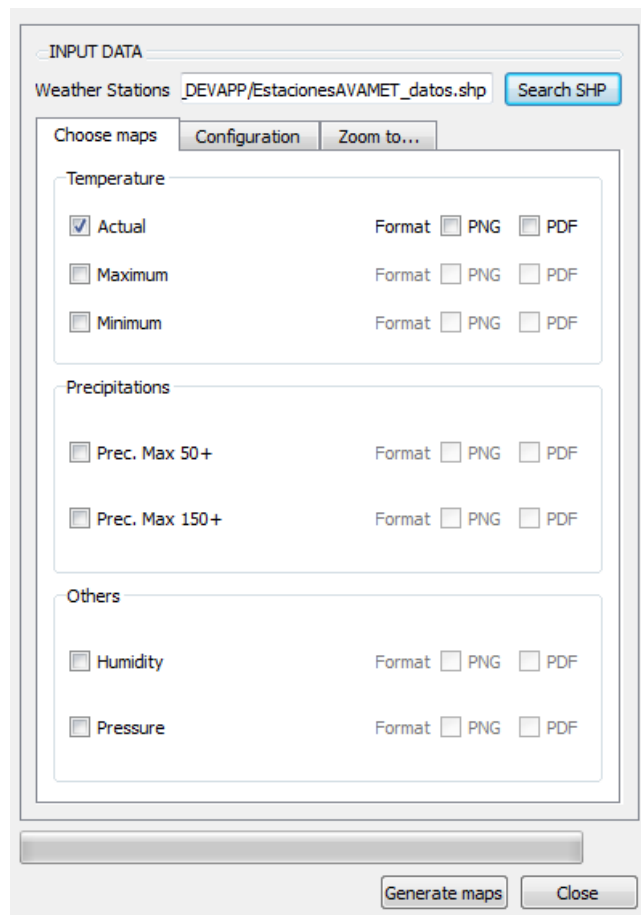


Figure 1: interface of application

2 DESCRIPTION

A graphical interface with PyQt4 had been created to build this tool. Once the interface had been built, an .ui file was generated and edited with Python consoles in order to assign functions to the buttons.

2.1 Interface

Since the objects and their functions and classes have been sufficiently described in the previous personal assignment, space in this project is devoted to another things. The interface itself was divided into three parts: choosing propriert data for interpolation (*Figure 2*), configuration parameters for map creating (*Figure 3*) and possibility to zoom to given territory (*Figure 4*).

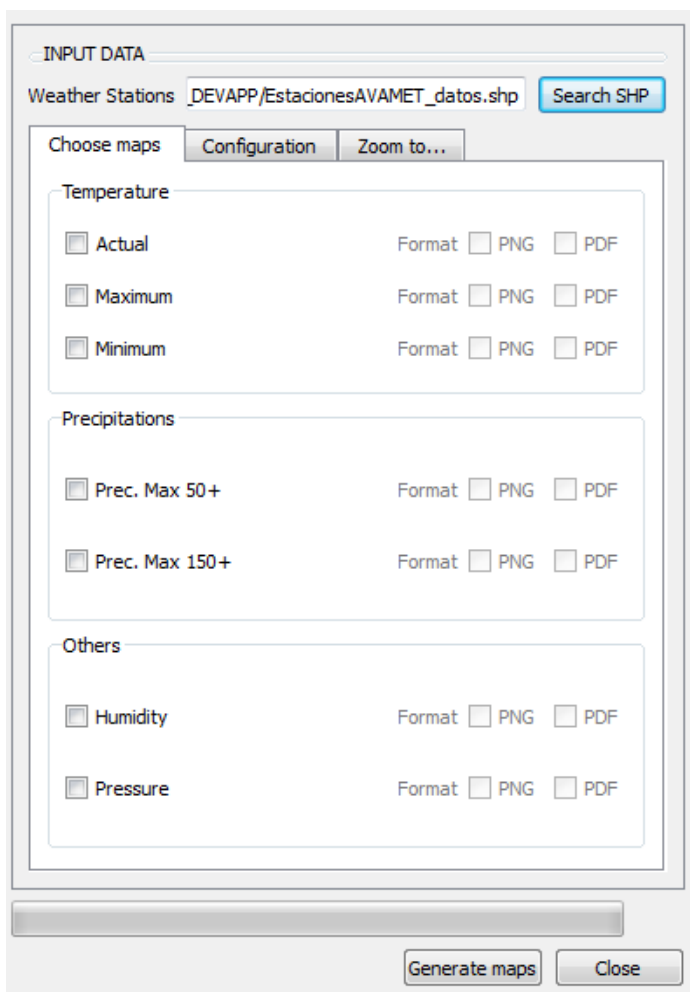


Figure 2: choose maps interface

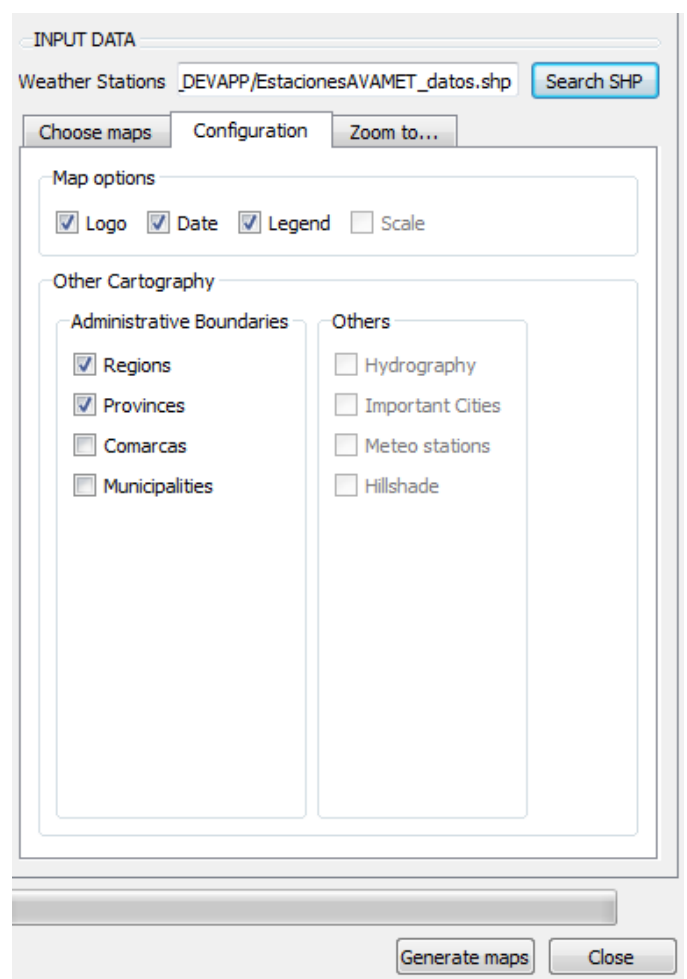


Figure 3: configuration map interface

The application requires as input shapefile with weather data. Those data must unfortunately contain already joined weather conditions. And as was said in previous chapture, the application allows to choose different weather maps and their formats (Figure 2). In configuration tab can user define, whether want to draw into the map

administrative boundaries, another topographic background, or to enable or disable basic map components, such as logo, date and legend.

It's possible to zoom the map to a determined zone. As default, map will be generated in regional scale. However, it's possible to zoom the map to a specific province or region (Figure 4)

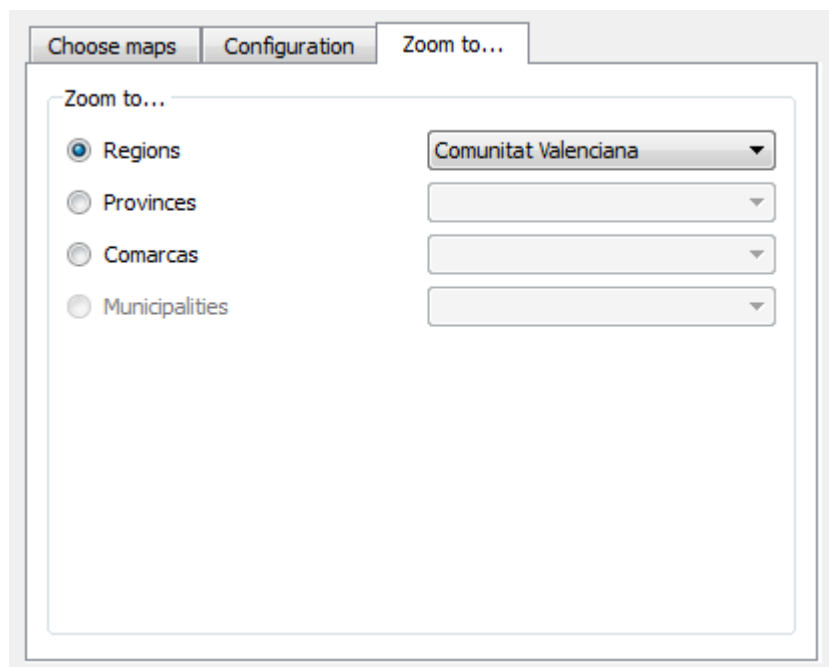


Figure 4: zooming options in interface

Once are all options defined, the programme will generate the weather maps according to user's specifications (Figure 5).

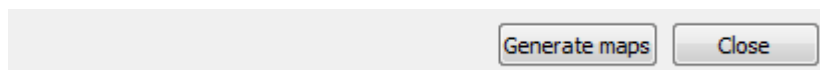


Figure 5: maps generating button

2.2 Map template

After interface was created, a print template map has been built, attending to the user specifications defined on the interface (Figure 6).

An easy-identification nomenclature of the map objects (logo, legend, date, layers) has been used, as in the software interface, to facilitate the implementation of functions with python.

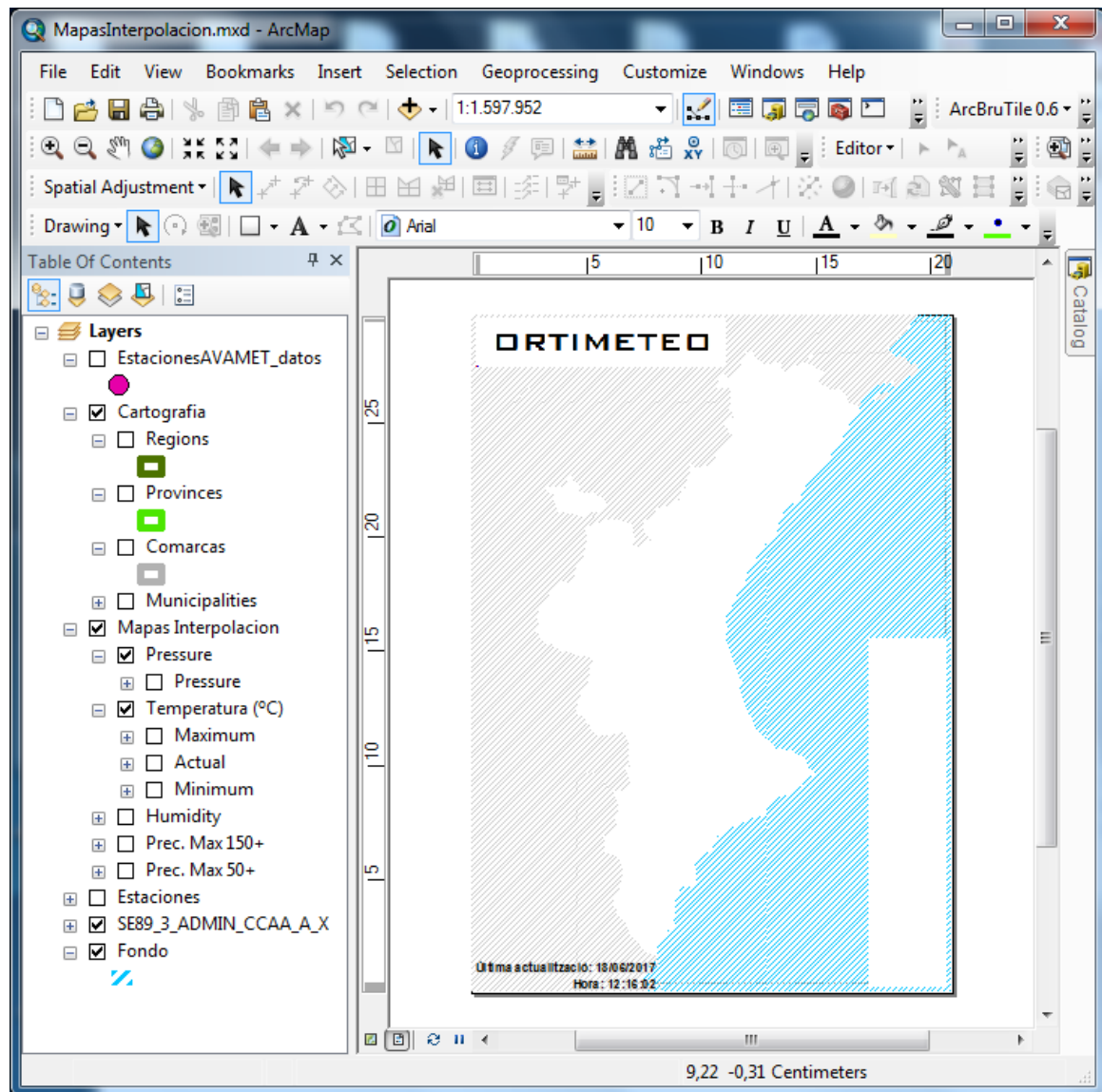


Figure 6: map template shown in ArcMap

2.3 Functionality

This chapter contains a light introduction to the functions and buttons what were used and their connection to the user environment.

2.3.1 Interface functionality

Firstly, a connection between buttons in interface and functions written in Python had to be introduced (see the following code).

```

#Load the file .ui
form = uic.loadUiType("MapsGenerator.ui")[0]
class MyDialogClass(QtGui.QDialog, form):
    def __init__(self, parent=None):
        QtGui.QDialog.__init__(self, parent)
        self.setupUi(self)

        #connections of events with functions
        self.btn_shp_path.clicked.connect(self.get_shp_file)
        self.btn_run.clicked.connect(self.run)
        self.btn_cancel.clicked.connect(self.cancel)

        #Progress bar
        self.progressBar.setMinimum(0)
        self.progressBar.setMaximum(7)

        #Enable and disable widgets automatically
        self.check_temp_mit.clicked.connect(self.enable_choose_map_buttons)
        self.check_temp_max.clicked.connect(self.enable_choose_map_buttons)
        self.check_temp_min.clicked.connect(self.enable_choose_map_buttons)
        self.check_prec50.clicked.connect(self.enable_choose_map_buttons)
        self.check_prec150.clicked.connect(self.enable_choose_map_buttons)
        self.check_hume.clicked.connect(self.enable_choose_map_buttons)
        self.check_pres.clicked.connect(self.enable_choose_map_buttons)

        #Enable and disable zoom widgets automatically
        self.rbtn_zoom_to_regional.clicked.connect(self.enable_zoom_values_cmb)
        self.rbtn_zoom_to_province.clicked.connect(self.enable_zoom_values_cmb)
        self.rbtn_zoom_to_comarca.clicked.connect(self.enable_zoom_values_cmb)
        #self.rbtn_zoom_to_municipality.clicked.connect(self.enable_zoom_values_cmb)

        #Load comarca values for the Qcombobox of zooming
        self.rbtn_zoom_to_comarca.clicked.connect(self.load_comarca_values_cmb)

```

As in the interface are 3 buttons, the functions for them had to be introduced as well. First one contains of function for loading shp data, second one for running the program itself and maps generating and the last one for closing the window. On the code below, is firstly shown the function for getting shapefile from user. This function uses the QFileDialog class, what provides a dialog that allow users to select files or directories. As can be seen, after putting the shapefile inside the programme, another functions in the interface are getting enabled.

```

def get_shp_file(self):
    #Open file. Only SHP files can be chosen
    fname = QFileDialog.getOpenFileName(self, 'Open file', '.', \
                                         "Shapefile /files (*.shp)")

    #Get SHP file path
    self.txt_shp_path.setText(fname)
    #Enable MAP OPTIONS
    self.tabWidget.setEnabled(True)
    self.btn_run.setEnabled(True)

```

After was created the function for closing the window.

```

def cancel(self):
    self.close()

```

The last function, „generate maps“ is connected to run function written in Python. After clicking on this button, the function will generate the map according to the data specifications. For understanding the code bellow, here is a little introduction. Firstly, it was necessary to access a MXD file (this was created in paralel with map template what

is described in previous chapter). And after, for better asseability and simplification, the widgets of the interface had been saved in different lists (see the code below).

```
def run(self):
    #Acces to the MDX file
    path_mxd = r"MapasInterpolacion.mxd"
    mxd = map.MapDocument(path_mxd)
    #Save buttons on Lists
    progressBar = [self.progressBar]
    input_files_path = [self.txt_shp_path.text()]
    weather_layer_list = [self.check_temp_mit, self.check_temp_max, \
                          self.check_temp_min, self.check_prec50, \
                          self.check_prec150, self.check_hume, \
                          self.check_pres]
    btn_png_list = [self.check_temp_mit_png, self.check_temp_max_png, \
                    self.check_temp_min_png, self.check_prec50_png, \
                    self.check_prec150_png, self.check_hume_png, \
                    self.check_pres_png]
    btn_pdf_list = [self.check_temp_mit_pdf, self.check_temp_max_pdf, \
                    self.check_temp_min_pdf, self.check_prec50_pdf, \
                    self.check_prec150_pdf, self.check_hume_pdf, \
                    self.check_pres_pdf]
    administrative_layer_list = [self.check_insert_regions_shp, \
                                self.check_insert_provinces_shp, \
                                self.check_insert_comarcas_shp, \
                                self.check_insert_municipalities_shp]
    other_layer_list = [self.check_insert_hydro_shp, \
                        self.check_insert_cities_shp, \
                        self.check_insert_stations_shp, \
                        self.check_insert_hillshade_shp]
    graphic_list = [self.check_logo, self.check_date, \
                    self.check_legend, self.check_scale]
    btn_zoom_to_layer_list = [self.rbtn_zoom_to_regional, \
                              self.rbtn_zoom_to_province, \
                              self.rbtn_zoom_to_comarca, \
                              self.rbtn_zoom_to_municipality]
    btn_zoom_to_layer_attribute_list= [self.cmb_zoom_to_region, \
                                       self.cmb_zoom_to_province, \
                                       self.cmb_zoom_to_comarca, \
                                       self.cmb_zoom_to_municipality]
```

The last section in running function contained the creating thematic map function (described later). This function will be applied to all possible maps and saved in list („weather layer list“).

```
i = 0 #Contador
for weather_layer in weather_layer_list:
    #Create thematic maps
    create_thematic_map(progressBar[0], mxd, input_files_path[0], i, weather_layer, btn_pdf_list[i], \
                        btn_png_list[i], administrative_layer_list, \
                        other_layer_list, graphic_list, \
                        btn_zoom_to_layer_list, \
                        btn_zoom_to_layer_attribute_list)
    i += 1
```


2.3.2 Enabling and disabling widgets

In order to facilitate the order of the parameters for checking the program, two enabling and disabling widgets functions were created. First one for the format output map widgets, and other for the zooming ones.

```
def enable_choose_map_buttons(self):
    #Temp_mit
    if self.check_temp_mit.isChecked() == False:
        self.check_temp_mit_png.setEnabled(False)
        self.check_temp_mit_pdf.setEnabled(False)
        self.lbl_temp_mit.setEnabled(False)
    elif self.check_temp_mit.isChecked() == True:
        self.check_temp_mit_png.setEnabled(True)
        self.check_temp_mit_pdf.setEnabled(True)
        self.lbl_temp_mit.setEnabled(True)

    #Temp_max
    if self.check_temp_max.isChecked() == False:
        self.check_temp_max_png.setEnabled(False)
        self.check_temp_max_pdf.setEnabled(False)
        self.lbl_temp_max.setEnabled(False)
    elif self.check_temp_max.isChecked() == True:
        self.check_temp_max_png.setEnabled(True)
        self.check_temp_max_pdf.setEnabled(True)
        self.lbl_temp_max.setEnabled(True)
```

The code above shows only an example of enabling and disabling buttons, accordingly, the rest of meteorological variables was created in the same way.

For zooming enable options were used similar process (the code below). As in the code above, the whole code for all options is not shown.

```
#ZOOM ENABLING FUNCTIONS
def enable_zoom_values_cmb(self):
    #Zoom to region
    if self.rbtn_zoom_to_regional.isChecked() == True:
        self.cmb_zoom_to_region.setEnabled(True)
        self.cmb_zoom_to_province.setEnabled(False)
        self.cmb_zoom_to_comarca.setEnabled(False)
        self.cmb_zoom_to_municipality.setEnabled(False)
    #Zoom to province
    elif self.rbtn_zoom_to_province.isChecked() == True:
        self.cmb_zoom_to_region.setEnabled(False)
        self.cmb_zoom_to_province.setEnabled(True)
        self.cmb_zoom_to_comarca.setEnabled(False)
        self.cmb_zoom_to_municipality.setEnabled(False)
    #Zoom to comarca
    elif self.rbtn_zoom_to_comarca.isChecked() == True:
        self.cmb_zoom_to_region.setEnabled(False)
        self.cmb_zoom_to_province.setEnabled(False)
        self.cmb_zoom_to_comarca.setEnabled(True)
        self.cmb_zoom_to_municipality.setEnabled(False)
```

An example of disabling zooming options is shown on the following picture (*Figure 6*)

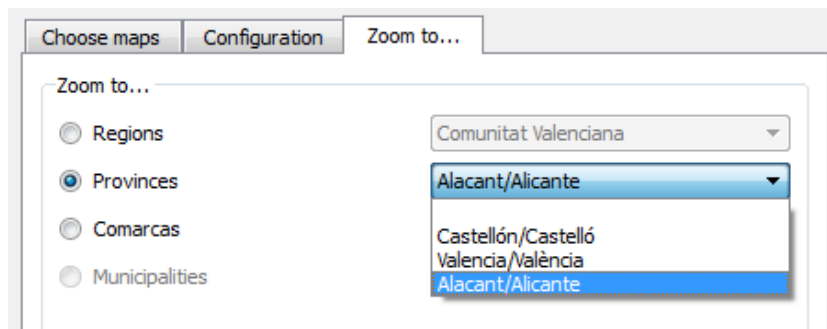


Figure 6: zooming options (enable for provinces)

2.3.3 Automatic loading data

Writing about zooming options, there were one thing what had to be done more – automatic loading data into QComboBox (Figure 7). It consists of automatically loaded field names from the layer what had been created before.

```
def load_comarca_values_cmb(self):
    list1 = []
    #Acces to the MDX file
    path_mxd = r"MapasInterpolacion.mxd"
    mxd = map.MapDocument(path_mxd)
    #Acces to the list of layers
    dataframe = map.ListDataFrames(mxd)[0]
    layers = map.ListLayers(mxd, "", dataframe)
    for layer in layers:
        #if layer.name == btn_layer.text():
        if layer.name == self.rbtn_zoom_to_comarca.text():
            cursor = arcpy.da.SearchCursor(layer, ['COMARCA_'])
            self.cmb_zoom_to_comarca.clear()
            for row in cursor:
                self.cmb_zoom_to_comarca.addItem(row)
```

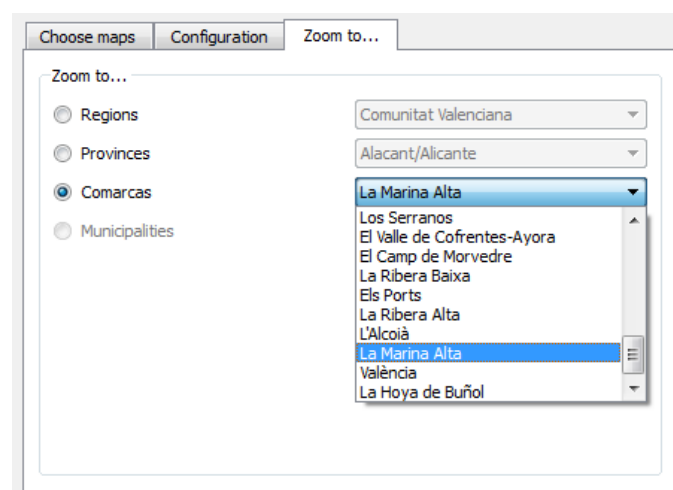


Figure 7: automatically loading data

2.4 Creating thematic maps

In this section are described several functions for generating weather maps. The workflow for generating was following (Figure 8):

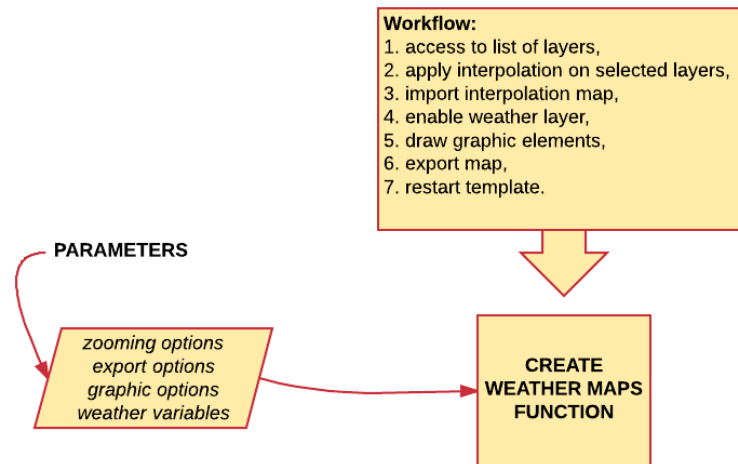


Figure 8: workflow for creating thematic map

2.4.1 Access to the list of layers

The following code shows definition of function with its parameters and accessing to the list of layers.

```
def create_thematic_map(progressBar, mxd, input_files_path, i, btn_layer, btn_export_pdf,\
                        btn_export_png, administrative_layer_list, other_layer_list, \
                        graphic_list, btn_zoom_to_layer_list, btn_zoom_to_layer_attribute_list):
    #Access to the list of layers
    dataframe = map.ListDataFrames(mxd)[0]
    layers = map.ListLayers(mxd,"",dataframe)
    if btn_layer.isChecked()==True:
```

2.4.2 Apply interpolation

Even though in the code (see below) are implemented two interpolation algorithms (IDW and Kriging), it was decided to use IDW, because in Kriging is appropriate manual data exploration analysis. After checking extension, the code was following:

```
shp_path = str(input_files_path)
inFeatures = shp_path
weather_variables = ['temp_min', 'temp_max', 'temp_min', 'prec', 'prec', 'hume', 'pres']
vble = weather_variables[i]
Z_value = vble
print Z_value
outRaster = "interpolation_maps/"+str(vble)
cellSize = 300
kModel = "Spherical 663.040376"

for layer in layers: #(group of layer name is read as one layer else)
    if layer.name == administrative_layer_list[2].text():
        arcpy.env.mask = layer.dataSource
        arcpy.env.extent = "626638.8673 4190956.536 815377.0266 4519162.5229"
        #arcpy.Kriging_3d(inFeatures, Z_value, outRaster, kModel, cellSize, "VARIABLE 12", "")
        arcpy.Idw_3d(inFeatures, Z_value, outRaster, cellSize, "2", "VARIABLE 12", "")
    #In order to automatically change the Source directory of the .shp inside the .mxd file
    #this is assuming that the user doesn't change the directory manually in the .mxd
    last_workspace_path = str(arcpy.env.workspace)
    mxd.findAndReplaceWorkspacePaths(last_workspace_path,arcpy.env.workspace)
```

2.4.3 Enable wather layer

The programme enables the weather layer chosen by user (*Figure 9*).

```
for layer in layers: #(group of layer name is read as one layer else)
    check_enable_layer (layer, btn_layer, True) #function created before
```

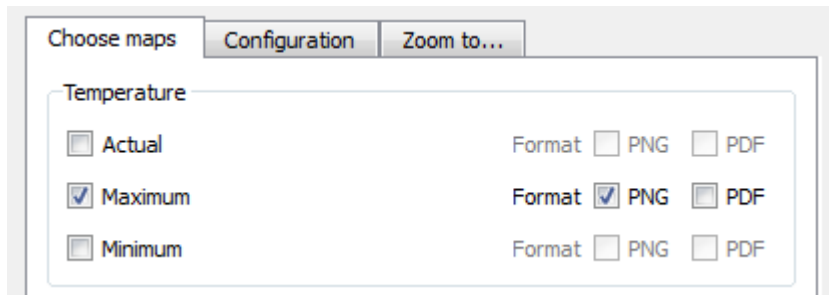


Figure 9: enabling layers

Moreover, other layers for users are enabled in configuration window.

```
#Enable other layers
for administrative_layer in administrative_layer_list:
    check_enable_layer(layer, administrative_layer, True)
for other_layer in other_layer_list:
    check_enable_layer(layer, other_layer, True)
```

In order to do this, programme uses *check_enable_layer* function. By default, no layer is enabled. This function enables the layer introduced by user on the interface.

```
''' Enables or disables layer according to the parameters introduced by user '''
def check_enable_layer (layer, btn_layer, check):
    if btn_layer.isChecked()==True:
        if layer.name == btn_layer.text():
            layer.visible = check
            arcpy.RefreshActiveView()
            arcpy.RefreshTOC()
```

2.4.4 Draw graphic elements

Next step was to draw graphic elements (logo, date, legend) chosen by user in Configuration layer (*Figure 10*).

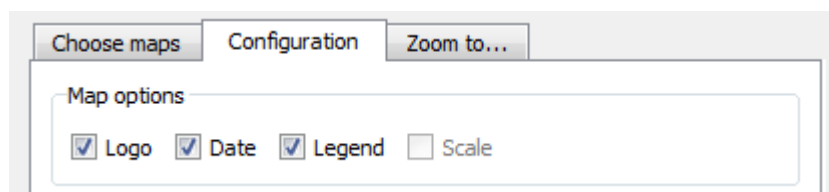


Figure 10: draw graphic elemens

The code for drawing graphics element is following:

```
'''Draws graphic elements according to the parameters introduced by user'''
def draw_graphic_maps(mxd, graphic_list):
    check_logo, check_date, check_legend, check_scale = graphic_list
    graphic_list_map = map.ListLayoutElements(mxd)
    for graphic in graphic_list_map:
        #Draw logo
        draw_graphic(graphic, 'logo_map', check_logo) #function created before
        #Draw Date
        draw_graphic(graphic, 'date_map', check_date)
        draw_graphic(graphic, 'hour_map', check_date)
        #Draw Legend
        draw_graphic(graphic, 'legend_map', check_legend)
```

This function uses another function called *draw_graphic* that draws the graphic element out of layout if user haven't chosen it.

```
def draw_graphic(graphic, graphic_name_map, graphic_btn):
    if graphic.name == graphic_name_map:
        if graphic_btn.isChecked()==False:
            #Graphic element out of layout
            graphic.elementPositionX = 33.3523
            graphic.elementPositionY = 0.4023
```

2.4.5 Zooming to selected territory

When layer maps are generated and added to the template and graphic objects are enabled or disabled, program zooms the layout to a specific area, according to the user's specifications.

```
def zoom_to_value(dataframe, btn_zoom_to_layer_list, layers, \
                  btn_zoom_to_layer_attribute_list):
    j = 0 #Contador
    for btn_zoom_to_layer in btn_zoom_to_layer_list:
        if btn_zoom_to_layer.isChecked()==True:
            for admin_layer in layers:
                if admin_layer.name == btn_zoom_to_layer.text():
                    if admin_layer.name == btn_zoom_to_layer_list[2].text():
                        SQL = "[COMARCA_] = '" + \
                            unicode(btn_zoom_to_layer_attribute_list[j].currentText()) + "'"
                    else:
                        SQL = "[ROTULO] = '" + \
                            unicode(btn_zoom_to_layer_attribute_list[j].currentText()) + "'"

                    arcpy.SelectLayerByAttribute_management(admin_layer, "NEW_SELECTION", SQL)
                    dataframe.zoomToSelectedFeatures()
                    arcpy.SelectLayerByAttribute_management(admin_layer, "CLEAR_SELECTION")
                    arcpy.RefreshActiveView()

            break
    j += 1
```

2.4.6 Exporting maps

Once all output maps are configured according to the parameters introduced in the interface, maps are exported in the formats what were selected.

```
def export_maps(mxd, btn_layer, btn_export_pdf, btn_export_png):
    map_name = str(btn_layer.objectName())[6:]
    if btn_export_png.isChecked() == True:
        #print btn_layer.text(), ' Print PNG'
        map.ExportToPNG(mxd, map_name+".png", resolution=300, \
            transparent_color='255, 255, 255')
    if btn_export_pdf.isChecked() == True:
        #print btn_layer.text(), ' Print PDF'
        map.ExportToPDF(mxd, map_name+".pdf" )
```

2.4.7 Template restart

Because programs allows generating more than one map, it's important to restart the layer's template configuration. So after exporting map all layer will began disable again.

```
#Disable layers (It's important due to clean de template for the next map)
for layer in layers: #(group of layer name is read as one layer else)
    check_enable_layer (layer, btn_layer, False)
    for administrative_layer in administrative_layer_list:
        check_enable_layer(layer, administrative_layer, False)
    for other_layer in other_layer_list:
        check_enable_layer(layer, other_layer, False)
```

3 CODE

```
1. # -*- coding: utf-8 -*-
2.
3.
4. #####
5. ### LIBRARIES ###
6. #####
7. import sys
8. from PyQt4 import QtCore, QtGui, uic
9. from PyQt4.QtCore import *
10. from PyQt4.QtGui import *
11. import arcpy
12. import arcpy.mapping as map
13. import sys, csv, os
14. from arcpy import env
15.
16. #####
17. ### FUNCTIONS ###
18. #####
19. ''' Enables or disables layer according to the parameters introduced by user '''
20. def check_enable_layer (layer, btn_layer, check):
21.     if btn_layer.isChecked()==True:
22.         if layer.name == btn_layer.text():
23.             layer.visible = check
24.             arcpy.RefreshActiveView()
25.             arcpy.RefreshTOC()
26.
27. '''Draws graphic element'''
28. #Without position
29. def draw_graphic(graphic, graphic_name_map, graphic_btn):
30.     if graphic.name == graphic_name_map:
31.         if graphic_btn.isChecked()==False:
32.             #Graphic element out of layout
33.             graphic.elementPositionX = 33.3523
34.             graphic.elementPositionY = 0.4023
35.             #print graphic_name_map, ": Disabled"
36.
37.
38. '''Draws graphic elements according to the parameters introduced by user'''
39. def draw_graphic_maps(mxd, graphic_list):
40.     check_logo, check_date, check_legend, check_scale = graphic_list
41.     graphic_list_map = map.ListLayoutElements(mxd)
42.     for graphic in graphic_list_map:
43.         #Draw logo
44.         draw_graphic(graphic, 'logo_map', check_logo) #function created before
45.         #Draw Date
46.         draw_graphic(graphic, 'date_map', check_date)
47.         draw_graphic(graphic, 'hour_map', check_date)
48.         #Draw Legend
49.         draw_graphic(graphic, 'legend_map', check_legend)
50.
51.
52. ##          #Draw Scale
53. ##          draw_graphic(graphic, 'scale_map', check_scale)
54.
55. ''' Zooms the map according to the parameters introduced by user'''
56. def zoom_to_value(dataframe, btn_zoom_to_layer_list, layers, \
57.                   btn_zoom_to_layer_attribute_list):
58.     j = 0 #Contador
59.     for btn_zoom_to_layer in btn_zoom_to_layer_list:
60.         if btn_zoom_to_layer.isChecked()==True:
61.             for admin_layer in layers:
```

```

62.         if admin_layer.name == btn_zoom_to_layer.text():
63.             if admin_layer.name == btn_zoom_to_layer_list[2].text():
64.                 SQL = "[COMARCA] = '" + \
65.                     unicode(btn_zoom_to_layer_attribute_list[j].currentText()
66.                 ) + "'"
67.             else:
68.                 SQL = "[ROTULO] = '" + \
69.                     unicode(btn_zoom_to_layer_attribute_list[j].currentText()
70.                 ) + "'"
71.             arcpy.SelectLayerByAttribute_management(admin_layer, "NEW_SELECTION", SQL)
72.             dataframe.zoomToSelectedFeatures()
73.             arcpy.SelectLayerByAttribute_management(admin_layer, "CLEAR_SELECTION")
74.             arcpy.RefreshActiveView()
75.             break
76.         j += 1
77.     '''Export maps according to the parameters introduced by user '''
78. def export_maps(mxd, btn_layer, btn_export_pdf, btn_export_png):
79.     map_name = str(btn_layer.objectName())[6:]
80.     if btn_export_png.isChecked() == True:
81.         #print btn_layer.text(), ' Print PNG'
82.         map.ExportToPNG(mxd, map_name+".png", resolution=300, \
83.             transparent_color='255, 255, 255')
84.     if btn_export_pdf.isChecked() == True:
85.         #print btn_layer.text(), ' Print PDF'
86.         map.ExportToPDF(mxd, map_name+".pdf" )
87.
88.     '''Creates thematic maps according to the parameters introduced by user '''
89.     '''This function uses other functions: draw_graphic(),
90.     check_enable_layer(), export_maps()'''
91. def create_thematic_map(progressBar, mxd, input_files_path, i, btn_layer, btn_export_pdf, \
92.     btn_export_png, administrative_layer_list, other_layer_list, \
93.     graphic_list, btn_zoom_to_layer_list, btn_zoom_to_layer_attribute_list):
94.     #Acces to the list of layers
95.     dataframe = map.ListDataFrames(mxd)[0]
96.     layers = map.ListLayers(mxd, "", dataframe)
97.     if btn_layer.isChecked() == True:
98.
99.         # Check out any necessary licenses
100.         arcpy.CheckOutExtension("3D")
101.         arcpy.CheckOutExtension('Spatial')
102.         arcpy.env.overwriteOutput=True
103.
104.         #Join Input data
105.         progressBar.setValue(1)
106.
107.         shp_path = str(input_files_path)
108.
109.         inFeatures = shp_path
110.         weather_variables = ['temp_mit', 'temp_max', 'temp_min', 'prec', 'prec',
111.             'hume', 'pres']
112.         vble = weather_variables[i]
113.         Z_value = vble
114.         print Z_value
115.         outRaster = "interpolation_maps/"+str(vble)
116.         cellSize = 300
117.         kModel = "Spherical 663.040376"
118.         for layer in layers: #(group of layer name is read as one layer else)

```



```

119.         if layer.name == administrative_layer_list[2].text():
120.             arcpy.env.mask = layer.dataSource
121.             arcpy.env.extent = "626638.8673 4190956.536 815377.0266 4519162.
5229"
122.             #arcpy.Kriging_3d(inFeatures, Z_value, outRaster, kModel, cellSi
ze, "VARIABLE 12", "")
123.             arcpy.Idw_3d(inFeatures, Z_value, outRaster, cellSize, "2", "VAR
TABLE 12", "")
124.             #In order to automatically change the Source directory of the .shp insid
e the .mxd file
125.             last_workspace_path = str(arcpy.env.workspace)
126.             mxd.findAndReplaceWorkspacePaths(last_workspace_path,arcpy.env.workspace
)
127.
128.
129.             #Enable weather layer
130.             progressBar.setValue(3)
131.             for layer in layers: #(group of layer name is read as one layer else)
132.                 check_enable_layer (layer, btn_layer, True) #function created before
133.
134.             #Enable other layers
135.             progressBar.setValue(4)
136.             for administrative_layer in administrative_layer_list:
137.                 check_enable_layer(layer, administrative_layer, True)
138.             for other_layer in other_layer_list:
139.                 check_enable_layer(layer, other_layer, True)
140.
141.
142.             #Draws graphic elements
143.             progressBar.setValue(5)
144.             draw_graphic_maps(mxd, graphic_list) #function created before
145.
146.             #Zoom to
147.             progressBar.setValue(6)
148.             zoom_to_value(dataframe, btn_zoom_to_layer_list, layers, \
149.                 btn_zoom_to_layer_attribute_list) #function created before
150.
151.             #Export map
152.             progressBar.setValue(7)
153.             export_maps(mxd, btn_layer, btn_export_pdf, \
154.                 btn_export_png) #function created before
155.
156.             #Disable layers
157.             for layer in layers: #(group of layer name is read as one layer else)
158.                 check_enable_layer (layer, btn_layer, False)
159.                 for administrative_layer in administrative_layer_list:
160.                     check_enable_layer(layer, administrative_layer, False)
161.                 for other_layer in other_layer_list:
162.                     check_enable_layer(layer, other_layer, False)
163.
164.             #####
165.             ### INTERFACE ###
166.             #####
167.
168.             #load the file .ui
169.             form = uic.loadUiType("MapsGenerator.ui")[0]
170.
171.             class MyDialogClass(QtGui.QDialog, form):
172.                 def __init__(self,parent=None):
173.                     QtGui.QDialog.__init__(self,parent)
174.                     self.setupUi(self)
175.
176.             #connections of events with functions

```

```

177.         self.btn_shp_path.clicked.connect(self.get_shp_file)
178.         self.btn_run.clicked.connect(self.run)
179.         self.btn_cancel.clicked.connect(self.cancel)
180.
181.         #Progress bar
182.         self.progressBar.setMinimum(0)
183.         self.progressBar.setMaximum(7)
184.
185.         #Enable and disable widgets automatically
186.         self.check_temp_mit.clicked.connect(self.enable_choose_map_buttons)
187.         self.check_temp_max.clicked.connect(self.enable_choose_map_buttons)
188.         self.check_temp_min.clicked.connect(self.enable_choose_map_buttons)
189.         self.check_prec50.clicked.connect(self.enable_choose_map_buttons)
190.         self.check_prec150.clicked.connect(self.enable_choose_map_buttons)
191.         self.check_hume.clicked.connect(self.enable_choose_map_buttons)
192.         self.check_pres.clicked.connect(self.enable_choose_map_buttons)
193.
194.         #Enable and disable zoom widgets automatically
195.         self.rbtn_zoom_to_regional.clicked.connect(self.enable_zoom_values_cmb)
196.
197.         self.rbtn_zoom_to_province.clicked.connect(self.enable_zoom_values_cmb)
198.         self.rbtn_zoom_to_comarca.clicked.connect(self.enable_zoom_values_cmb)
199.         #self.rbtn_zoom_to_municipality.clicked.connect(self.enable_zoom_values_
200.         cmb)
201.
202.         #Load comarca values for the Qcombobox of zooming
203.         self.rbtn_zoom_to_comarca.clicked.connect(self.load_comarca_values_cmb)
204.
205.         #self.rbtn_zoom_to_municipality.clicked.connect(self.load_municipality_v
206.         alues_cmb)
207.
208.
209.         #####
210.         ### ENABLE AND DISABLE FUNCTIONS ###
211.         #####
212.
213.         def enable_choose_map_buttons(self):
214.             #Temp_mit
215.             if self.check_temp_mit.isChecked() == False:
216.                 self.check_temp_mit_png.setEnabled(False)
217.                 self.check_temp_mit_pdf.setEnabled(False)
218.                 self.lbl_temp_mit.setEnabled(False)
219.             elif self.check_temp_mit.isChecked() == True:
220.                 self.check_temp_mit_png.setEnabled(True)
221.                 self.check_temp_mit_pdf.setEnabled(True)
222.                 self.lbl_temp_mit.setEnabled(True)
223.
224.             #Temp_max
225.             if self.check_temp_max.isChecked() == False:
226.                 self.check_temp_max_png.setEnabled(False)
227.                 self.check_temp_max_pdf.setEnabled(False)
228.                 self.lbl_temp_max.setEnabled(False)
229.             elif self.check_temp_max.isChecked() == True:
230.                 self.check_temp_max_png.setEnabled(True)
231.                 self.check_temp_max_pdf.setEnabled(True)
232.                 self.lbl_temp_max.setEnabled(True)
233.
234.             #Temp_min
235.             if self.check_temp_min.isChecked() == False:
236.                 self.check_temp_min_png.setEnabled(False)
237.                 self.check_temp_min_pdf.setEnabled(False)
238.                 self.lbl_temp_min.setEnabled(False)
239.             elif self.check_temp_min.isChecked() == True:

```

```

237.         self.check_temp_min_png.setEnabled(True)
238.         self.check_temp_min_pdf.setEnabled(True)
239.         self.lbl_temp_min.setEnabled(True)
240.
241.     #Prec_50
242.     if self.check_prec50.isChecked() == False:
243.         self.check_prec50_png.setEnabled(False)
244.         self.check_prec50_pdf.setEnabled(False)
245.         self.lbl_prec50.setEnabled(False)
246.     elif self.check_prec50.isChecked() == True:
247.         self.check_prec50_png.setEnabled(True)
248.         self.check_prec50_pdf.setEnabled(True)
249.         self.lbl_prec50.setEnabled(True)
250.
251.     #Prec_150
252.     if self.check_prec150.isChecked() == False:
253.         self.check_prec150_png.setEnabled(False)
254.         self.check_prec150_pdf.setEnabled(False)
255.         self.lbl_prec150.setEnabled(False)
256.     elif self.check_prec150.isChecked() == True:
257.         self.check_prec150_png.setEnabled(True)
258.         self.check_prec150_pdf.setEnabled(True)
259.         self.lbl_prec150.setEnabled(True)
260.
261.     #Hume
262.     if self.check_hume.isChecked() == False:
263.         self.check_hume_png.setEnabled(False)
264.         self.check_hume_pdf.setEnabled(False)
265.         self.lbl_hume.setEnabled(False)
266.     elif self.check_hume.isChecked() == True:
267.         self.check_hume_png.setEnabled(True)
268.         self.check_hume_pdf.setEnabled(True)
269.         self.lbl_hume.setEnabled(True)
270.
271.     #Pres
272.     if self.check_pres.isChecked() == False:
273.         self.check_pres_png.setEnabled(False)
274.         self.check_pres_pdf.setEnabled(False)
275.         self.lbl_pres.setEnabled(False)
276.     elif self.check_pres.isChecked() == True:
277.         self.check_pres_png.setEnabled(True)
278.         self.check_pres_pdf.setEnabled(True)
279.         self.lbl_pres.setEnabled(True)
280.
281.     #ZOOM ENABLING FUNCTIONS
282.     def enable_zoom_values_cmb(self):
283.         #Zoom to region
284.         if self.rbtn_zoom_to_regional.isChecked() == True:
285.             self.cmb_zoom_to_region.setEnabled(True)
286.             self.cmb_zoom_to_province.setEnabled(False)
287.             self.cmb_zoom_to_comarca.setEnabled(False)
288.             self.cmb_zoom_to_municipality.setEnabled(False)
289.         #Zoom to province
290.         elif self.rbtn_zoom_to_province.isChecked() == True:
291.             self.cmb_zoom_to_region.setEnabled(False)
292.             self.cmb_zoom_to_province.setEnabled(True)
293.             self.cmb_zoom_to_comarca.setEnabled(False)
294.             self.cmb_zoom_to_municipality.setEnabled(False)
295.         #Zoom to comarca
296.         elif self.rbtn_zoom_to_comarca.isChecked() == True:
297.             self.cmb_zoom_to_region.setEnabled(False)
298.             self.cmb_zoom_to_province.setEnabled(False)
299.             self.cmb_zoom_to_comarca.setEnabled(True)
300.             self.cmb_zoom_to_municipality.setEnabled(False)
301.

```

```

302.     ##         #Zoom to municipality
303.     ##         elif self.rbtn_zoom_to_municipality.isChecked() == True:
304.     ##             self.cmb_zoom_to_region.setEnabled(False)
305.     ##             self.cmb_zoom_to_province.setEnabled(False)
306.     ##             self.cmb_zoom_to_comarca.setEnabled(False)
307.     ##             self.cmb_zoom_to_municipality.setEnabled(True)
308.
309.
310.     def load_comarca_values_cmb(self):
311.         list1 = []
312.         #Acces to the MDX file
313.         path_mxd = r"MapasInterpolacion.mxd"
314.         mxd = map.MapDocument(path_mxd)
315.         #Acces to the list of layers
316.         dataframe = map.ListDataFrames(mxd)[0]
317.         layers = map.ListLayers(mxd,"",dataframe)
318.         for layer in layers:
319.             #if layer.name == btn_layer.text():
320.             if layer.name == self.rbtn_zoom_to_comarca.text():
321.                 cursor = arcpy.da.SearchCursor(layer, ['COMARCA_'])
322.                 self.cmb_zoom_to_comarca.clear()
323.                 for row in cursor:
324.                     self.cmb_zoom_to_comarca.addItem(row)
325.
326.
327.     #####
328.     ### BUTTON FUNCTIONS ###
329.     #####
330.     def get_shp_file(self):
331.         #Open file. Only SHP files can be chosen
332.         fname = QFileDialog.getOpenFileName(self, 'Open file', '.', \
333.                                             "Shapefile /files (*.shp)")
334.
335.         #Get SHP file path
336.         self.txt_shp_path.setText(fname)
337.         #Enable MAP OPTIONS
338.         self.tabWidget.setEnabled(True)
339.         self.btn_run.setEnabled(True)
340.
341.     def run(self):
342.         #Acces to the MDX file
343.         path_mxd = r"MapasInterpolacion.mxd"
344.         mxd = map.MapDocument(path_mxd)
345.
346.         #Save buttons on lists
347.         progressBar = [self.progressBar]
348.
349.         input_files_path = [self.txt_shp_path.text()]
350.
351.         weather_layer_list = [self.check_temp_mit, self.check_temp_max, \
352.                               self.check_temp_min, self.check_prec50, \
353.                               self.check_prec150, self.check_hume, \
354.                               self.check_pres]
355.
356.         btn_png_list = [self.check_temp_mit_png, self.check_temp_max_png, \
357.                         self.check_temp_min_png, self.check_prec50_png, \
358.                         self.check_prec150_png, self.check_hume_png, \
359.                         self.check_pres_png]
360.
361.         btn_pdf_list = [self.check_temp_mit_pdf, self.check_temp_max_pdf, \
362.                         self.check_temp_min_pdf, self.check_prec50_pdf, \
363.                         self.check_prec150_pdf, self.check_hume_pdf, \
364.                         self.check_pres_pdf]
365.
366.         administrative_layer_list = [self.check_insert_regions_shp, \
367.                                       self.check_insert_provinces_shp, \
368.                                       self.check_insert_comarcas_shp, \
369.                                       self.check_insert_municipalities_shp]

```

```

367.         other_layer_list = [self.check_insert_hydro_shp, \
368.                             self.check_insert_cities_shp, \
369.                             self.check_insert_stations_shp, \
370.                             self.check_insert_hillshade_shp]
371.         graphic_list = [self.check_logo, self.check_date, \
372.                         self.check_legend, self.check_scale]
373.
374.         btn_zoom_to_layer_list = [self.rbtn_zoom_to_regional, \
375.                                   self.rbtn_zoom_to_province, \
376.                                   self.rbtn_zoom_to_comarca, \
377.                                   self.rbtn_zoom_to_municipality]
378.
379.         btn_zoom_to_layer_attribute_list= [self.cmb_zoom_to_region, \
380.                                            self.cmb_zoom_to_province, \
381.                                            self.cmb_zoom_to_comarca, \
382.                                            self.cmb_zoom_to_municipality]
383.
384.         i = 0 #Contador
385.         for weather_layer in weather_layer_list:
386.             #Create thematic maps
387.             create_thematic_map(progressBar[0], mxd, input_files_path[0], i, wea
ther_layer, btn_pdf_list[i], \
388.                                     btn_png_list[i], administrative_layer_list, \
389.                                     other_layer_list, graphic_list, \
390.                                     btn_zoom_to_layer_list, \
391.                                     btn_zoom_to_layer_attribute_list)
392.             i += 1
393.
394.
395.         def cancel(self):
396.             self.close()
397.
398.
399.         app = QtGui.QApplication(sys.argv)
400.         myDialog = MyDialogClass(None)
401.         myDialog.show()
402.         app.exec_()

```

4 RESULTS

The tool was tested several times based on shapefile “EstacionesAVAMET_datos”, what contains 144 records from weather stations (after join). On the picture below (Figure 11) can be seen the first result – interpolation of humidity.

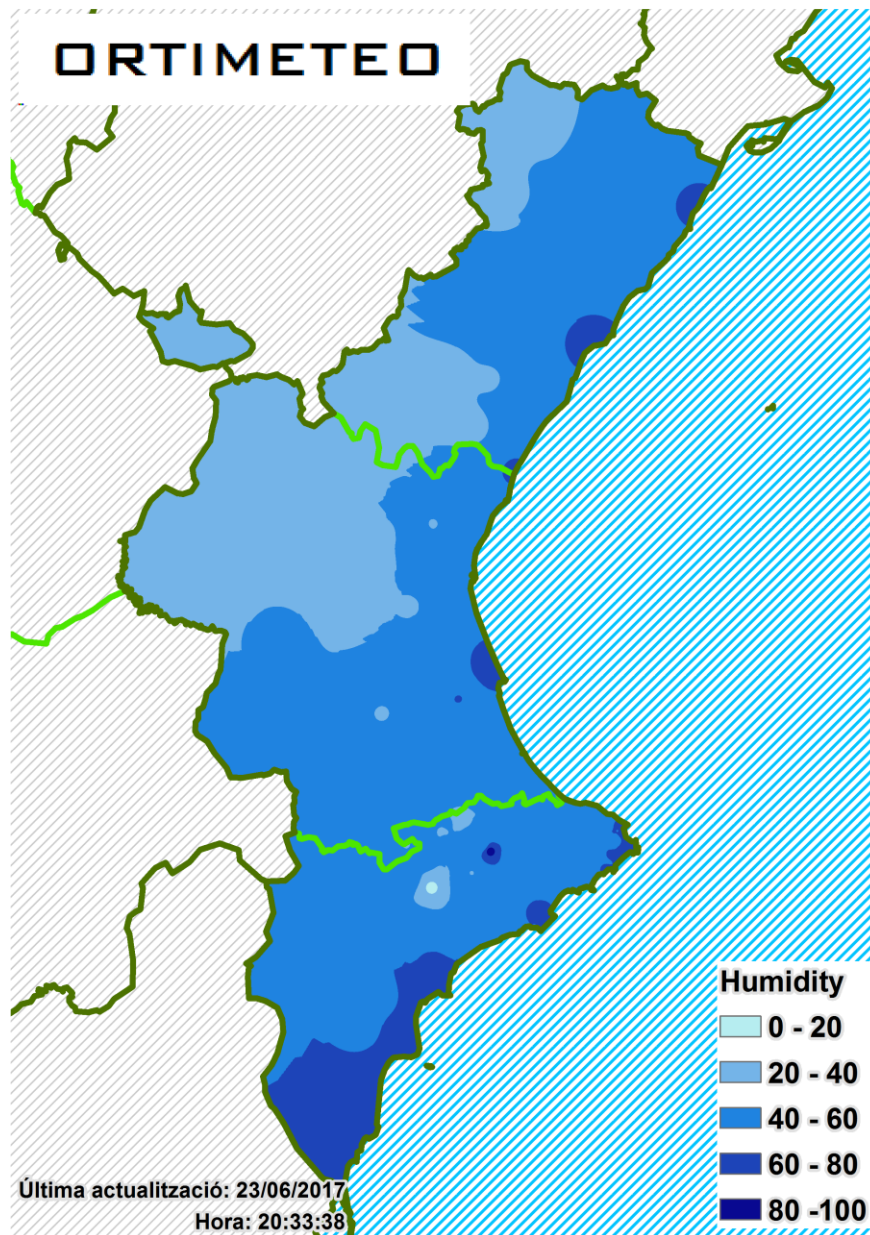


Figure 11: humidity weather map

Next were tested temperature in its maximum and minimum (*Figure 12*)

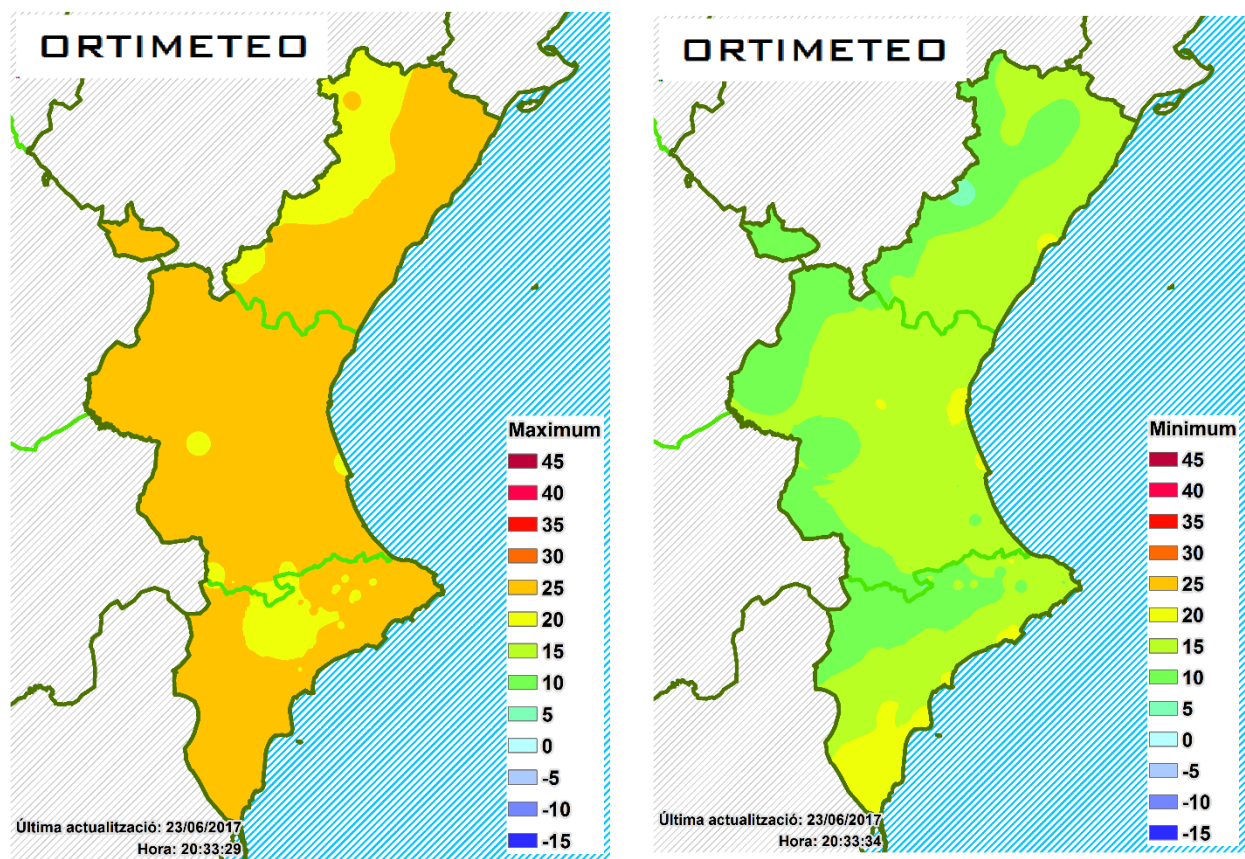
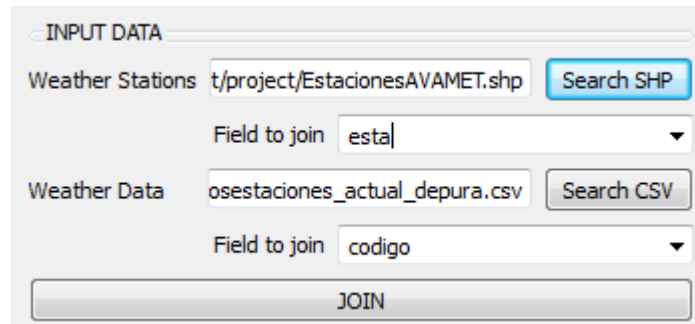


Figure 12: minimum and maximum temperature maps

5 CONCLUSION

The application, what was created, can find a wide use in different fields of research. Firstly, it is a pity, that the join function wasn't performed. This would simplify the process on one level higher (*Figure 12*). Unfortunately, even it was managed to load both shp and csv shapefile and get their field names, the join function itself didn't work. And after browsing discussing forums, it was found out, that there is huge amount of people whom both function what can perform join in arcmap don't work.



INPUT DATA

Weather Stations t/project/EstacionesAVAMET.shp Search SHP

Field to join esta

Weather Data osestaciones_actual_depura.csv Search CSV

Field to join codigo

JOIN

Figure 12: performing join

But even without join function (it can be considered, that the user is able to perform join function in GIS software very easily) can bet his application very helpful. But still, there are possibilities, how to improve this tool, for example creating map with different interpolation algorithms, in case of kriging compute the uncertainty of interpolation and so on.

In conclusion, an application was created. As input is necessary to use already joined data of meteorological stations and after that, can be chosen what variable user want to interpolate (can be interpolated all in the same time).

6 BIBLIOGRAPHY

- LLOYD, C.D a P.M ATKINSON. Assessing uncertainty in estimates with ordinary and indicator kriging. *Computers & Geosciences* [online]. 2001, **27**(8), 929-937 [cit. 2017-06-07]. DOI: 10.1016/S0098-3004(00)00132-1. ISSN 00983004.
- Turning off layers in ArcMap via ArcPy? *GIS Stack Exchange* [online]. [cit. 2017-06-21]. Available at: <https://gis.stackexchange.com/questions/73433/turning-off-layers-in-arcmap-via-arcpy>
- Using Arcpy to zoom to selected feature? *GIS Stack Exchange* [online]. [cit. 2017-06-21]. Available at: <https://gis.stackexchange.com/questions/1711/using-arcpy-to-zoom-to-selected-feature>
- How to get content to ComboBox. *Stack Overflow* [online]. [cit. 2017-06-21]. Available at: <https://stackoverflow.com/questions/6061893/how-do-you-get-the-current-text-contents-of-a-qcombobox>
- Improve you python classes. *Jeff Knupp: python programmer* [online]. [cit. 2017-06-21]. Available at: <https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/>
- Technical Vignette 3:: Kriging, interpolation, and uncertainty. *Berkley university* [online]. [cit. 2017-06-21]. Available at: <http://www.stat.berkeley.edu/~paciorek/research/techVignettes/techVignette3.pdf>
- Geostatistical Analyst Tutorial. *ArcGIS Help* [online]. [cit. 2017-06-21]. Available at: <http://help.arcgis.com/en/arcgisdesktop/10.0/pdf/geostatistical-analyst-tutorial.pdf>
- Exercise 40: Modules, Classes, and Objects. *Learn Python* [online]. [cit. 2017-06-21]. Available at: <https://learnpythonthehardway.org/book/ex40.html>
- Writing a QTableWidgetItem to a .csv or .xls. *Stack Overflow* [online]. [cit. 2017-06-21]. Available at: <https://stackoverflow.com/questions/12608835/writing-a-qtablewidget-to-a-csv-or-xls>
- Change QLabel text dynamically in PyQt4. *Stack Overflow* [online]. [cit. 2017-06-21]. Available at: <https://stackoverflow.com/questions/12608835/writing-a-qtablewidget-to-a-csv-or-xls>
- Problem adding a join to a table via Python using addJoin_management. *GIS Stack Exchange* [online]. [cit. 2017-06-21]. Available at: <https://gis.stackexchange.com/questions/71116/problem-adding-a-join-to-a-table-via-python-using-addjoin-management>
- How do you get the current text contents of a QComboBox? *Stack Overflow* [online]. [cit. 2017-06-21]. Available at: <https://stackoverflow.com/questions/6061893/how-do-you-get-the-current-text-contents-of-a-qcombobox>
- Select Layer By Attribute. *ArcGIS for Desktop* [online]. [cit. 2017-06-21]. Available at: <http://desktop.arcgis.com/en/arcmap/10.3/tools/data-management-toolbox/select-layer-by-attribute.htm>

- Export to PNG. *ArcGIS for Desktop* [online]. [cit. 2017-06-21]. Available at: <http://desktop.arcgis.com/es/arcmap/10.3/analyze/arcpy-mapping/exporttopng.htm>
- QProgressBar. *PyQt4* [online]. [cit. 2017-06-21]. Available at: <http://pyqt.sourceforge.net/Docs/PyQt4/qprogressbar.html#setFormat>
- Syntax Highlight Code In Word Documents. *Planet B* [online]. [cit. 2017-06-21]. Available at: <http://www.planetb.ca/syntax-highlight-word>