

ArchaeoLogic Visualiser - Code

Student 9217953

January 11, 2020

Abstract

Included here is the full source code for the project, comprising Python, HTML, JavaScript, and CSS. Database connection, object creation, and most of the logic is handled by python. Python objects and variables are passed through to the HTML using Jinja2. User interaction and dynamic webpage changes are handled by JavaScript and CSS, and all styling is handled withing CSS.

1 Python Code - visualiser.py

```
1  #!/usr/bin/env python3
2
3  from jinja2 import Environment, FileSystemLoader
4  import cx_Oracle
5  import cgi
6  import cgitb
7
8  cgitb.enable(format='text')
9
10 class GraphicsArea:
11
12     # Defines an area for SVG graphics to be rendered on the website
13
14     def __init__(self, width, height, viewBox_x, viewBox_y, viewBox_width, viewBox_height):
15
16         # Parameters passed in during creation
17         self.width = f"{width}px"           # width of SVG element on page
18         self.height = f"{height}px"         # height of SVG element on page
19         self.viewBox_x = viewBox_x          # viewBox min x
20         self.viewBox_y = viewBox_y          # viewBox min y
21         self.viewBox_width = viewBox_width  # viewBox width
22         self.viewBox_height = viewBox_height # viewBox height
23
24         # string representation of the whole viewBox:
25         self.viewBox_custom = f"{viewBox_x} {viewBox_y} {viewBox_width} {viewBox_height}"
26
27
28 class Field:
29
30     # Field Object Creator
31
32     def __init__(self, field_id, lowx, lowy, hix, hiy, area, owner, crop_id):
33
34         # Parameters passed in during creation (ie fetched from database)
35         self.field_id = field_id           # a number uniquely identifying the field
36         self.lowx = lowx                    # x-coordinate of lower left corner
37         self.lowy = lowy                    # y-coordinate of lower left corner
38         self.hix = hix                     # x-coordinate of upper right corner
39         self.hiy = hiy                     # y-coordinate of upper right corner
40         self.area = f"{area:.2f}"          # area, formatted to 2 decimal places
41         self.owner = owner                  # name of the farmer who owns the field
42         self.crop_id = crop_id              # a number identifying the crop in the field
43
44         # Derived Attributes (populated at runtime)
45         self.finds_in_this_field = []      # a list containing the finds found in this field
46
47         # Attributes calculated from object properties
48         self.width = hix - lowx             # field width (in map units)
49         self.height = hiy - lowy            # field height
50         self.centroidx = (hix + lowx)/2 + lowx # field centroid x coordinate
51         self.centroidy = (hiy + lowy)/2 + lowy # field centroid y coordinate
52
53         # Default value for fill is 'none'. This property is dynamically added at runtime
```

```

54         self.fill = 'none'
55
56     # User & coder friendly representations of Field objects
57     def __repr__(self):
58         return f"Field({self.field_id}, {self.lowx}, {self.lowy}, {self.hix}, {self.hiy})"
59
60     def __str__(self):
61         return f"Field {self.field_id} - Bottom Left({self.lowx},{self.lowy}) Top Right({self.hix},{self.hiy})"
62
63
64 class Find:
65
66     # Find Object Creator
67
68     def __init__(self, find_id, xcoord, ycoord, find_type, depth, field_notes):
69         self.find_id = find_id          # a number uniquely identifying the find
70         self.xcoord = xcoord            # x coordinate of find location
71         self.ycoord = ycoord            # y coordinate of find location
72         self.find_type = find_type      # a number identifying the find type (class)
73         self.depth = f"{depth:.2f}"    # the depth of the find, formatted to 2 decimal places
74         self.field_notes = field_notes  # a string field with field notes
75         self.class_name = 'none'       # holds the text name of the crop populated at runtime
76
77     # Derived Attributes
78
79     # a list of which field(s) the find is found in
80     # (it can be more than 1 field if on the border between fields!)
81     self.in_which_fields = []          # populated at runtime
82
83     self.fill = get_find_colour(self.find_type) # applies a colour based on the find type
84
85     # User & coder friendly representations of Find objects
86     def __repr__(self):
87         return f"Find({self.find_id}, {self.xcoord}, {self.ycoord})"
88
89     def __str__(self):
90         return f"Find {self.find_id} - Coordinates : ({self.xcoord}, {self.ycoord})"
91
92
93 class MyClass:
94
95     # MyClass Object Creator
96
97     def __init__(self, class_type, name, period, use):
98         self.class_type = class_type    # a number identifying the class type
99         self.name = name                 # a string with the class name
100        self.period = period              # a string with the period of the class
101        self.use = use                    # a string with the use of the class
102
103        self.fill = 'none'                # the colour of a particular class, populated at runtime
104
105     # User & coder friendly representations of MyClass objects
106     def __repr__(self):
107         return f"Class({self.class_type}, {self.name}, {self.period}, {self.use})"
108
109     def __str__(self):
110         return f"Class # {self.class_type} - {self.name}, Period : {self.period}, Use: {self.use}"
111
112
113 class Crop:
114
115     # Crop Object Creator
116
117     def __init__(self, crop, name, startseason, endseason):
118         self.crop = crop                 # a number identifying the crop type
119         self.name = name                 # a string identifying the crop name
120         self.startseason = startseason   # the start of the growing season
121         self.endseason = endseason       # the end of the growing season
122
123        self.fill = 'none'                # the colour assigned to each crop type, populated at runtime
124
125     # User & coder friendly representations of Crop objects
126     def __repr__(self):
127         return f"Crop({self.crop}, {self.name}, {self.startseason}, {self.endseason})"
128
129     def __str__(self):

```

```

130         return f"Crop # {self.crop} - {self.name}, Start of Season: {self.startseason}, End of Season
131         : {self.endseason})"
132
133 def getDBdata(table_name, order_column):
134     # Accesses the Oracle database and creates Find, Field, Crop & MyClass
135     # objects with which to create the website visualiser tools
136     results = []
137     with open('../../details.txt', 'r') as f:
138         pwd = f.readline().strip()
139     try:
140         conn = cx_Oracle.connect(f"s0092179/{pwd}@geoslearn")
141         c = conn.cursor()
142         c.execute(f"SELECT * FROM {table_name} ORDER BY {order_column}")
143     except:
144         print("Failed to connect to Database Server...")
145
146     if table_name == "MY_FIELDS":
147         fields_list = [] # initialise an empty list
148         for row in c:
149             (a, b, c, d, e, f, g, h) = row # pack the results of the query into a tuple
150             field_name = table_name[:-1] + str(a)
151             field_name = Field(a, b, c, d, e, f, g, h) # create a new Field object for each row in
the table
152             fields_list.append(field_name) # add the newly created field to the "
fields_list" list
153             results = fields_list # return the list of Field objects
154
155     # The following elif statements handle creation of other objects from the
156     # different tables. I will not comment these further as the process is the same as for Field
objects
157     elif table_name == "MY_FINDS":
158         finds_list = []
159         for row in c:
160             (a, b, c, d, e, f) = row
161             find_name = table_name[:-1] + str(a)
162             find_name = Find(a, b, c, d, e, f)
163             finds_list.append(find_name)
164             results = finds_list
165
166     elif table_name == "MY_CLASS":
167         classes_list = []
168         for row in c:
169             (a, b, c, d) = row
170             my_class = MyClass(a, b, c, d)
171             classes_list.append(my_class)
172             results = classes_list
173
174     elif table_name == "MY_CROPS":
175         crops_list = []
176         for row in c:
177             (a, b, c, d) = row
178             my_crop = Crop(a, b, c, d)
179             crops_list.append(my_crop)
180             results = crops_list
181     else: # if no table name matches are made, go to the
else...
182         print("Table Name not supported...")
183     conn.close() # close the connection to the Oracle server
184     return results
185
186
187 def get_which_finds(fields, finds):
188     # loops through fields and finds, and identifies which finds fall within which field,
189     # and also which fields a particular find falls in (could be more than one if on border...)
190     # The program appends any items fulfilling the basic spatial criteria to the relevant field and
find objects attribute lists
191     for field in fields:
192         for find in finds:
193             if find.xcoord >= field.lowx and find.xcoord <= field.hix and find.ycoord >= field.lowy
and find.ycoord <= field.hiy:
194                 field.finds_in_this_field.append(find.find_id)
195                 find.in_which_fields.append(field.field_id)
196
197
198 def get_field_colour(field_crop):
199     # takes a crop name and returns the correct colour for rendering to the web
200     if field_crop == 'TURNIPS':

```

```

201         return '#A647FF' # purple
202     elif field_crop == 'OIL SEED RAPE':
203         return '#F3FC30' # pale yellow
204     elif field_crop == 'STRAWBERRIES':
205         return '#FD5959' # orangey red
206     elif field_crop == 'PEAS':
207         return '#91F708' # light green
208     elif field_crop == 'POTATOES':
209         return '#F9C89A' # lightish orange
210     else:
211         return 'none'
212
213
214 def get_find_colour(find_class):
215     # takes a class name and returns the correct colour for rendering to the web
216     if find_class == 1:
217         return '#9AA8F9' # light blue
218     elif find_class == 2:
219         return '#C8C8C8' # light grey
220     elif find_class == 3:
221         return '#ABC349' # flinty green
222     elif find_class == 4:
223         return '#D1BB00' # mustard colour
224     else:
225         return 'none'
226
227
228 def get_crop_name(crops, crop_id):
229     # takes a crop id number, and returns a string of the actual crop name
230     for crop in crops:
231         if crop.crop == crop_id:
232             return crop.name
233     else:
234         continue
235
236
237 def get_class_name(my_class, find_type):
238     # takes a find type, and returns a string of its class name
239     for cls in my_class:
240         if cls.class_type == find_type:
241             return cls.name
242     else:
243         continue
244
245
246 def get_unique_owners(fields):
247     # simply takes the list of all owners and returns a list of just the unique values
248     # which is used for displaying dropdown menus in the website with no duplicate values
249     unique = []
250     for field in fields:
251         if field.owner not in unique:
252             unique.append(field.owner)
253     return unique
254
255
256 def get_max_find_coordinates(finds):
257     # returns a list of the maximum x and y co-ordinates for any find
258     # this is used for rendering the co-ordinate options in the dropdown menus
259     # to make sure that the values are sensible and go high enough but not any further than needed
260     max = [0,0]
261     for find in finds:
262         if find.xcoord > max[0]:
263             max[0] = find.xcoord
264         if find.ycoord > max[1]:
265             max[1] = find.ycoord
266
267     return max
268
269
270 def print_svg(width, height, viewBox):
271     # if required, can return a string of the basic SVG tag
272     return f'<svg width="{width}" height="{height}" viewBox="{viewBox}">'
273
274
275 def assign_field_colours(fields, crops):
276     # takes a lists of fields and crops, and populates the fill colour attributes
277     # by matching crop id from field objects with crop from crop objects
278     # This results in all fields with a particular crop having the same colour,
279     # and all crop name cells in the tables having the same colour.
280     for field in fields:

```

```

279     for crop in crops:
280         if field.crop_id == crop.crop:
281             field.fill = get_field_colour(crop.name)
282             crop.fill = field.fill
283         else:
284             continue
285
286
287 def assign_find_colours(finds, classes):
288     # takes a lists of finds and classes, and populates the fill colour attributes
289     # by matching find type from find objects with class_type from class objects
290     # This results in all finds with a particular class having the same colour,
291     # and all class name cells in the tables having a matching colour.
292     for find in finds:
293         for cls in classes:
294             if find.find_type == cls.class_type:
295                 find.fill = get_find_colour(cls.class_type)
296                 cls.fill = find.fill
297             else:
298                 continue
299
300
301 def assign_crop_names(fields, crops):
302     # populates the crop_name attribute for each field by linking the crop id to the list of crop
303     # objects
304     for field in fields:
305         field.crop_name = get_crop_name(crops, field.crop_id)
306
307
308 def assign_class_names(finds, classes):
309     # populates the class_name attribute of each find by linking the find_type to the list of
310     # MyClass objects
311     for find in finds:
312         find.class_name = get_class_name(classes, find.find_type)
313
314
315 def render_html():
316     # Uses Jinja2 to render an html template and passing in a long list of objects and variables
317     # from python to the html page
318     env = Environment(loader=FileSystemLoader('.'))
319     temp = env.get_template('index.html')
320     print(temp.render(fields=field_objects, finds=find_objects, classes=my_classes, crops=my_crops, g
321 =graphics_area_for_svg, unique_owners=unique_owners, max_find_coords=max_find_coords))
322
323
324 if __name__ == '__main__':
325
326     print("Content-type: text/html\n")          # Ensures that the webpage is rendered correctly using
327     HTML code
328
329     # Create new lists of objects for each of the main types of interest
330     my_classes = getDBdata("MY_CLASS", "TYPE")
331     my_crops = getDBdata("MY_CROPS", "CROP")
332     field_objects = getDBdata("MY_FIELDS", "FIELD_ID")
333     find_objects = getDBdata("MY_FINDS", "FIND_ID")
334
335     # run the required functions to complete the population of the important fields
336     # in each of the objects by cross referencing between object attributes
337     assign_field_colours(field_objects, my_crops)
338     assign_find_colours(find_objects, my_classes)
339     assign_crop_names(field_objects, my_crops)
340     assign_class_names(find_objects, my_classes)
341     unique_owners = get_unique_owners(field_objects)
342     get_which_finds(field_objects, find_objects)
343     max_find_coords = get_max_find_coordinates(find_objects)
344
345     # Define a new GraphicsArea object which contains the necessary attributes for the SVG rendering
346     # on the website
347     # (SVG container width in pixels, SVG container height in pixels, viewBox min x, viewBox min y,
348     # viewBox width, viewBox height)
349     graphics_area_for_svg = GraphicsArea(420, 530, -1, 1, 16, 18)
350
351     # render the html template to the browser
352     render_html()

```

Listing 1: Python3 Code

2 HTML Markup - index.html

```
1 <!DOCTYPE html>
2
3 <!-- TIGIS Assignment 3 - ArchaeoLogic Visualiser -->
4 <!-- January 2020 -->
5 <!-- Student Number 9217953 -->
6
7 <html lang="en" dir="ltr">
8 <head>
9   <meta charset="utf-8">
10  <meta name="viewport" content="width=device-width, initial-scale=1">
11  <!-- Add link to external stylesheet -->
12  <link rel="stylesheet" href="../../style.css">
13  <title>ArchaeoLogic Visualiser</title>
14 </head>
15 <!-- -->
16 <body>
17   <div class="content">
18     <div class="header">
19       <h1 id="main_title">ArchaeoLogic</h1>
20     </div>
21     <div class="column-map">
22
23       <h2>Map of Fields & Finds</h2>
24       <h3>Axes labelled in Map Units - Click to Select</h3>
25
26       <!-- Main Container DIV for the SVG Map -->
27
28       <svg width="{{ g.width }}" height="{{ g.height }}" viewBox="{{ g.viewBox_custom }}">
29         <!-- The SVG Map is rendered dynamically using Jinja2 to access variables in the Python classes associated with Finds & Fields
30             Furthermore, the variable "g" is a GraphicsArea class defined in python and containing information on the SVG viewBox,
31             height, and width -->
32         <!--<svg width="100%" height="100%" viewBox="{{ g.viewBox_custom }}">-->
33             <!-- The transform(scale(1,-1)) flips the coordinates around the x-axis, and translate relocates the flipped axes in the
34             correct place by shifting
35             by an amount equal to the total SVG viewBox height. This is done because the SVG co-ordinates are a graphical
36             co-ordinate system and we need to display a
37             geographical co-ordinate system (with origin at bottom left rather than top right...)-->
38             <g id="draw_svg" transform="scale(1,-1) translate(0, -{{ g.viewBox_height }}">
39               <!-- Draw the background grid for the field and find map -->
40               <!-- Two patterns are used : smallGrid to do small grid squares, and grid to render the larger ones with slightly
41               thicker lines -->
42               <pattern id="smallGrid" width="0.25" height="0.25" patternUnits="userSpaceOnUse">
43                 <path d="M 1 0 0 1" fill="none" stroke="#e3effc" stroke-width="0.04"/>
44               </pattern>
45               <pattern id="grid" width="1" height="1" patternUnits="userSpaceOnUse">
46                 <rect width="13" height="17" fill="url(#smallGrid)"/>
47                 <path d="M 10 0 0 10" fill="none" stroke="#ccddf0" stroke-width="0.1"/>
48               </pattern>
49               <rect class="rect-grid" width="13.03" height="17.03" fill="url(#grid)" />
50             </g>
51             <!-- Loop through the fields and draw the SVG rectangle for each field in the variable "fields" -->
```

```

48 <!-- Note that both fields and finds have JS code associated so that when they are clicked, they are toggled between "
49 selected" (appearance changes and information appears) and "unselected" -->
50   {% for field in fields %}
51     <rect class="field" id="field{{ field.find_id }}" onclick="setFieldSelected(this)" x="{{ field.lowx }}" y="{{
52       field.lowy }}" width="{{ field.width }}" height="{{ field.height }}" fill="{{ field.fill }}" stroke="black"/>
53     {% endfor %}
54 <!-- Loop through the finds and draw the SVG circle for each find in the variable "finds" -->
55   {% for find in finds %}
56     <circle class="find" id="find{{ find.find_id }}" onclick="setFindSelected(this)" cx = "{{ find.xcoord }}" cy="{{
57       find.ycoord }}" r="0.4" fill="{{ find.fill }}" />
58     {% endfor %}
59 </g>
60 <g id="axes">
61 <!-- The following 2 loops go through the x and y co-ordinates and render numbers in the appropriate places (using x,y
62 co-ordinates and "translate" to tweak the positions) -->
63   {% for x in range(g.viewBox_width-2) %}
64     <text class="x_axis_label" transform="translate(-0.1, 0)" x="{{ x }}" y="{{ g.viewBox_height + 0.5 }}" fill="gray"
65       font-size="0.3">{{ x }}</text>
66     {% endfor %}
67     {% for y in range(g.viewBox_height) %}
68       <text class="y_axis_label" transform="translate(0, 0.1)" x="{{ -0.7 }}" y="{{ g.viewBox_height-y }}" fill="gray"
69         font-size="0.3">{{ y }}</text>
70       {% endfor %}
71 </g>
72 <g id="text_numbers">
73 <!-- The following 2 loops go through the fields and finds and render numbers in the appropriate places (using x,y
74 co-ordinates and "translate" to tweak the positions) -->
75   {% for field in fields %}
76     <text id="text_field{{ field.find_id }}" class="field_number" transform="translate(-0.2, 0)" x="{{ field
77       centroidx }}" y="{{ g.viewBox_height-field.centroidy }}" font-size="0.5">{{field.find_id}}</text>
78     {% endfor %}
79     {% for find in finds %}
80       <text id="text_find{{ find.find_id }}" class="find_number" onclick="setFindSelected(find{{ find.find_id }})"
81         transform="translate(-0.15, 0.2)" x="{{ find.xcoord }}" y="{{ g.viewBox_height-find.ycoord }}" font-size="0.5">{{find.find_id}}</text>
82       {% endfor %}
83 </g>
84 </svg>
85
86 <!-- Render three buttons, each with JS functions triggered when clicked, to clear and reset the various page elements -->
87 <div class="centering">
88   <input class="button" type="button" name="clear_map" value="Clear Map" onclick="clearMap()" />
89   <input class="button" type="button" name="clear_tables" value="Clear Tables" onclick="clearTables()" />
90   <input class="button" type="reset" value="Reset All..." onclick="resetForms()" />
91 </div>
92
93 <!-- The main container DIV for the Search and Query Forms -->
94 <div class="column-form">
95
96 <!-- The most basic search form, showing 2 dropdown menus to select by Farmer and/or Crop -->
97
98 <p class="instructions_text">Basic Search - Select items matching the following criteria:</p>
99 <form id="main_form" name="main_form" action="#" method="post" onsubmit="return processForm(this)">

```

```

93 <div class="form_elements">
94
95 <!-- Fieldsets are used to group similar form elements together -->
96 <fieldset class="selectors">
97
98 <!-- Dropdown menus are rendered by looping through the required variable and creating an "option" for each one
-->
99
100 <select class="dropdown" name="option1">
101     <option selected disabled>Field Owner...</option>
102     {% for owner in unique_owners %}
103         <option value="{{owner}}">{{ owner.title() }}</option>
104     {% endfor %}
105 </select>
106 <select class="dropdown" name="option2">
107     <option selected disabled>Crop Type...</option>
108     {% for crop in crops %}
109         <option value="{{crop.name}}">{{ crop.name.title() }}</option>
110     {% endfor %}
111 </select>
112 </fieldset>
113
114 <div>
115     <br>
116     <br>
117     <p></p>
118     <!-- Render the buttons that will control the form above -->
119     <p class="instructions_text">Match...</p>
120     <fieldset class="radio_button_fieldset">
121         <input class="radio_button" type="radio" name="and_or" id="select_or" checked value="ANY">Any<br>
122         <input class="radio_button" type="radio" name="and_or" id="select_and" value="ALL">Both<br>
123     </fieldset>
124     <fieldset class="buttons">
125         <input class="button_right" type="submit" value="Search...">
126     </fieldset>
127 </div>
128 </form>
129
130 <!-- The first spatial search form, which triggers the JS function "processSpatialForm()", and passes the form itself through to
the function.
131 The layout is similar to the above, but x and y coordinates are rendered up to the values of the highest
132 coordinates of any find (calculated in the python code and passed through using Jinja2)
133 This ensures that the values in the dropdown menus are sensible and cover all necessary values.
134 NOTE: +1 is added to each list because of the index values starting at 0 and only
135 reaching up to the (length of the list - 1). -->
136
137 <form id="spatial_form" name="spatial_form" action="#" method="post" onsubmit="return processSpatialForm(this)">
138     <div class="form_elements">
139         <br>
140         <hr>
141         <p class="instructions_text">Spatial Search - Select finds within an area:</p>
142         <fieldset class="selectors">
143             <select class="dropdown" name="lowleftx">

```



```

145 <option selected disabled>Lower Left X Coordinate</option>
146 {% for xcoord in range(0,max_find_coords[0]+1) %}
147 <option value="{{xcoord}}">{{ xcoord }}</option>
148 {% endfor %}
149 </select>
150 <select class="dropdown" name="lowlefty">
151 <option selected disabled>Lower Left Y Coordinate</option>
152 {% for ycoord in range(0,max_find_coords[1]+1) %}
153 <option value="{{ycoord}}">{{ ycoord }}</option>
154 {% endfor %}
155 </select>
156 </fieldset>
157 <fieldset class="selectors">
158 <select class="dropdown" name="highrightx">
159 <option selected disabled>Upper Right X Coordinate</option>
160 {% for xcoord in range(0,max_find_coords[0]+1) %}
161 <option value="{{xcoord}}">{{ xcoord }}</option>
162 {% endfor %}
163 </select>
164 <select class="dropdown" name="highrighty">
165 <option selected disabled>Upper Right Y Coordinate</option>
166 {% for ycoord in range(0,max_find_coords[1]+1) %}
167 <option value="{{ycoord}}">{{ ycoord }}</option>
168 {% endfor %}
169 </select>
170 </fieldset>
171 </div>
172 <div>
173 <fieldset class="buttons">
174 <input class="button_right" type="submit" value="Search...">
175 <br>
176 </fieldset>
177 </div>
178 </form>
179
180 <!-- The second spatial search form. This is very similar to the first, but calls the functions
181 "processSpatialForm2()" instead, and again passes the form itself through to the function so that it has
182 access to the user input -->
183
184 <form id="spatial_form2" name="spatial_form2" action="#" method="post" onsubmit="return processSpatialForm2(this)">
185 <div class="form_elements">
186 <br>
187 <hr>
188
189 <p class="instructions_text">Spatial Search - Select finds within a radius of a point:</p>
190 <fieldset class="selectors">
191 <select class="dropdown" name="centre_x">
192 <option selected disabled>Centre Point X</option>
193 {% for xcoord in range(0,max_find_coords[0]+1) %}
194 <option value="{{xcoord}}">{{ xcoord }}</option>
195 {% endfor %}
196 </select>
197 <select class="dropdown" name="centre_y">
198 <option selected disabled>Centre Point Y</option>

```

```

199     {% for ycoord in range(0,max_find_coords[1]+1) %}
200         <option value="{{ycoord}}">{{ ycoord }}</option>
201     {% endfor %}
202 </select>
203 </fieldset>
204 <fieldset class="selectors">
205     <select class="dropdown" name="radius">
206         <option selected disabled>Radius</option>
207         {% for xcoord in range(0,max_find_coords[0]+1) %}
208             <option value="{{xcoord}}">{{ xcoord }}</option>
209         {% endfor %}
210     </select>
211 </fieldset>
212 </div>
213 <div>
214     <fieldset class="buttons">
215         <input class="button_right" type="submit" value="Search...">
216     <br>
217 </fieldset>
218 </div>
219 </form>
220
221
222 </div>
223
224 <!-- The main container DIV for the Field and Find Tables to display all the necessary information
225 on whichever fields and finds are selected -->
226
227 <div class="column-table">
228
229 <!-- These tables are created dynamically, again by looping through variables and adding rows for each field and find.
230 The key is that by default they are rendered with class="hidden", which means they are not visible on the screen. When
231 a given entity is selected by the user or by JS code, the class attribute is altered to remove "hidden", and "find_selected"
232 or "field_selected" is added instead -->
233
234 <!-- RENDER FIELD INFORMATION IN TABLE FORM -->
235
236     <div class="field_table" id="field_table1">
237         <h2>Field Information</h2>
238         <table>
239             <tr>
240                 <th>Field Number</th>
241                 <th>Owner</th>
242                 <th>Lower Left</th>
243                 <th>Upper Right</th>
244                 <th>Area</th>
245                 <th>Crop Number</th>
246                 <th>Crop Name</th>
247                 <th>Contains Find(s)</th>
248             </tr>
249             {% for field in fields %}
250                 <tr class="hidden" id="tablerow_field{{field.field_id}}">
251                     <td class="field_number"> {{field.field_id}} </td>
252                     <td class="owner"> {{field.owner.title()}}</td>

```

```

253 <td class="lower_left"> {{field.lowx}}, {{field.lowy}} </td>
254 <td class="upper_right"> {{field.hix}}, {{field.hiy}} </td>
255 <td class="area"> {{field.area}} </td>
256 <td class="crop_number"> {{field.crop_id}}</td>
257 <td class="crop_name" style="background-color:{{field.fill}}"> {{field.crop_name.title()}}</td>
258 <td class="finds_in_this_field"> {% for find in field.finds_in_this_field %} {{find}} {% endfor %}</td>
259
260     </tr>
261     {% endfor %}
262 </table><br>
263
264 <!-- RENDER FIND INFORMATION IN TABLE FORM -->
265
266 <div class="find_table" id="find_table1">
267 <h2>Find Information</h2>
268 <table>
269 <tr>
270 <th>Find Number</th>
271 <th>X</th>
272 <th>Y</th>
273 <th>Type</th>
274 <th>Class</th>
275 <th>Depth</th>
276 <th>Is In Field(s)</th>
277 <th>Field Notes</th>
278 </tr>
279 {% for find in finds %}
280 <tr class="hidden" id="tablerow_find{{find.find_id}}">
281 <td class="find_number"> {{find.find_id}} </td>
282 <td class="x" id="find{{find.find_id}}_xcoord"> {{find.xcoord}} </td>
283 <td class="y" id="find{{find.find_id}}_ycoord"> {{find.ycoord}} </td>
284 <td class="find_type"> {{find.find_type}} </td>
285 <td class="find_class_name" style="background-color:{{find.fill}}"> {{find.class_name.title()}} </td>
286 <td class="depth"> {{find.depth}} </td>
287 <td class="in_which_fields">{% for field in find.in_which_fields %} {{field}} {% endfor %}</td>
288 <td class="field_notes"> {{find.field_notes.title()}} </td>
289 </tr>
290 {% endfor %}
291 </table><br>
292 </div>
293 </div>
294
295 <!-- A basic footer with design, author, and date information -->
296
297 <div class="footer">
298 <p>designed by student number 9217953, january 2020</p>
299 </div>
300
301 <div>
302 <script type="text/javascript" src="../../main.js"></script>
303 </body>
</html>

```

Listing 2: HTML Markup

3 JavaScript - main.js

```
1 // JavaScript to support the operation of the ArcheoLogic Visualiser website
2 // January 2020
3 // Student Number 9217953
4
5 function processSpatialForm2(f) {
6
7     // This function takes a spatial search form as an argument, and processes
8     // it to identify finds within a certain radius of a central point
9
10    // NOTE: All calcalations are done using arbitrary Map Units
11
12    // First clear the map and tables for a clean slate
13    clearMap();
14    clearTables();
15
16    // Set up variables by getting user input data from the form
17    var centre_x = f.centre_x.value // search centre point x-coordinate
18    var centre_y = f.centre_y.value // search centre point y-coordinate
19    var radius = f.radius.value // search radius
20
21    // Get all find x and y coordinates by selecting table cells with a class of "x" or "y"
22    var find_x_elements = document.querySelectorAll('td[class~="x"]')
23    var find_y_elements = document.querySelectorAll('td[class~="y"]');
24
25    // A variable to keep track of how many finds are identified
26    var finds_found = 0;
27
28
29    // Loop through the x and y elements, convert the text to integer values,
30    // then perform simple Pythagorean geometry to identify those within a certain radius
31    for (i=0; i < find_x_elements.length; i++) {
32        let x = parseInt(find_x_elements[i].innerText, 10);
33        let y = parseInt(find_y_elements[i].innerText, 10);
34        let y_difference = Math.abs(y - centre_y);
35        let x_difference = Math.abs(x - centre_x);
36        let straight_line_distance = Math.sqrt(Math.pow(x_difference, 2) + Math.pow(y_difference, 2))
37        // If a find is closer than or equal to the radius set, run through this code:
38        if (straight_line_distance <= radius){
39            finds_found +=1;
40            var find_number = i+1;
41            var local_find_id = ("find").concat(find_number);
42            setSelectedFromFindID(local_find_id); // Calls JS function to set the find as
43            // selected and show it on the map and tables
44            console.log("Find ", i+1 , "is within ", radius, "map units of the centre point and
45            has find_id of ", local_find_id)
46        }
47        // If no successful finds were identified, clear the map from any previous selections,
48        // and alert the user to try again:
49        if (finds_found == 0) {
50            clearMap();
51            var alert_string = ("No match within " + radius + " map units of (" + centre_x + "," +
52            centre_y + ") : please try again...");
53            alert(alert_string);
54        }
55        return false
56    }
57
58    function processSpatialForm(f) {
59
60        // This function takes a spatial search form as an argument, and processes
61        // it to identify finds that fall within the search area, a rectangle
62        // defined by the x,y coordinates of the lower left and top right corners.
63
64        // NOTE: All calcalations are done using arbitrary Map Units
65
66        // First clear the map and tables for a clean slate
67        clearMap();
68        clearTables();
69
70        // Set up variables by getting user input data from the form
71        // In this case they represent the lower left and upper right coordinates
```

```

71 // of the search area
72 var lowleftx = f.lowleftx.value
73 var lowlefty = f.lowlefty.value
74 var highrightx = f.highrightx.value
75 var highrighty = f.highrighty.value
76
77 // Get all find x and y coordinates by selecting table cells with a class of "x" or "y"
78 var find_x_elements = document.querySelectorAll('td[class~="x"]')
79 var find_y_elements = document.querySelectorAll('td[class~="y"]');
80
81 // Loop through the find x, y coordinates and check if that find falls within
82 // the search rectangle (NOTE: or right on the border of it!)
83 for (i=0; i < find_x_elements.length; i++) {
84     let x = parseInt(find_x_elements[i].innerText, 10); // converts text string to
integer
85     let y = parseInt(find_y_elements[i].innerText, 10);
86     // Search logic : && operator is used to match all conditions
87     if ((x >= lowleftx) && (x <= highrightx) && (y >= lowlefty) && (y <= highrighty)){
88         var find_number = i+1;
89         var local_find_id = ("find").concat(find_number);
90         setSelectedFromFindID(local_find_id); // Trigger JS function to handle selecting
this find
91         console.log("Find ", find_number , "is in the search area and has find_id of ",
local_find_id)
92     }
93 }
94 return false
95 }
96
97
98 function clearMap() {
99
100     // This funtions simply identifies all Fields and Finds marked as "selected"
101     // by including the class of "find_selected" or "field_selected", and loops through
102     // each one to reset the class to "find" or "field" respectively
103
104     var selected_fields = document.querySelectorAll('rect[class~="field_selected"]')
105     var selected_finds = document.querySelectorAll('circle[class~="find_selected"]')
106
107     // Loop through selected fields on the map, and reset to unselected state
108     for (i=0; i<selected_fields.length; i++){
109         console.log(selected_fields[i])
110         selected_fields[i].setAttribute("class", "field")
111     }
112     // Loop through selected finds on the map, and reset to unselected state
113     for (i=0; i<selected_finds.length; i++){
114         console.log(selected_finds[i])
115         selected_finds[i].setAttribute("class", "find")
116     }
117 }
118
119 function clearTables() {
120
121     // This funtions simply identifies all field and find table rows that do NOT have a
class of "hidden"
122     // (ie are visible), and loops through each one to reset the class to "hidden"
123
124     // Get all the visible field and find table rows and store in 2 variables
125     var field_tablerow_elements = document.querySelectorAll('tr[class=""]');
126     var find_tablerow_elements = document.querySelectorAll('tr[class=""]');
127
128     console.log("field_tablerow_elements: ", field_tablerow_elements)
129     console.log("find_tablerow_elements: ", find_tablerow_elements)
130
131     // Loop through all visible field table rows and reset to hidden
132     for (i=0; i<field_tablerow_elements.length; i++){
133         console.log(field_tablerow_elements[i])
134         field_tablerow_elements[i].setAttribute("class", "hidden")
135     }
136     // Loop through all visible find table rows and reset to hidden
137     for (i=0; i<find_tablerow_elements.length; i++){
138         console.log(find_tablerow_elements[i])
139         find_tablerow_elements[i].setAttribute("class", "hidden")
140     }
141 }
142
143 function processFormAll(field_owner, owner_elements, crop_name, crop_elements) {
144

```

```

145 // this function takes user-selected elements and tablerow elements
146 // and compares them to identify fields that match ALL criteria, ie that have both
147 // the chosen owner name and also crop name.
148
149 // Set up local variables
150
151 var owner_index;
152 var crop_index;
153 var class_index;
154
155 // Get all field and find table rows by selecting table rows with an id matching the
pattern
156 var all_field_rows = document.querySelectorAll('tr[id^="tablerow_field"]');
157 var all_find_rows = document.querySelectorAll('tr[id^="tablerow_find"]');
158
159 // Loop through all the rows, and get local variables from their children elements
160 // NOTE: The children elements are the <td> elements ie the individual cells in the
tables
161 // (ie to "get inside" the table rows and extract info, then set variables equal to that
info)
162 // Note also that the trim() and toUpperCase() functions are used to "clean" up and
standardise the data
163 for (i=0; i < all_field_rows.length; i++) {
164     let this_field = ("field").concat(all_field_rows[i].children[0].innerText.trim());
165     let this_owner = all_field_rows[i].children[1].innerText.trim().toUpperCase();
166     let this_crop = all_field_rows[i].children[6].innerText.trim().toUpperCase();
167     let finds_in_this_field = all_field_rows[i].children[7].innerText.trim();
168     if ((field_owner == this_owner) && (crop_name == this_crop)){
169         console.log("Match FOUND"); // Console logging only used for debugging
170         setSelectedFromFieldID(this_field); // Use external JS function to set the field
as selected
171     } else {
172         console.log("NO MATCH")
173     }
174 }
175 return false;
176 }
177
178 function processFormAny(field_owner, owner_elements, crop_name, crop_elements) {
179
180     // this function takes user-selected elements and tablerow elements
181     // and compares them to identify fields that match ANY criteria, ie that have either
182     // the chosen owner name or crop name. Those that are matched are selected.
183
184     // Loop through all the owner attributes in each field tablerow, and
185     // check if the user selected owner is the same. If so, select that field.
186     // In order to go "up one" from owner elements to a field element, the parentElement
operator is used.
187     // Note also that the trim() and toUpperCase() functions are used to "clean" up and
standardise the data
188     for (i=0; i < owner_elements.length; i++){
189         if(owner_elements[i].outerText.toUpperCase().trim() == field_owner.trim()) {
190             owner_elements[i].parentElement.setAttribute("class", "");
191             var parent_id = owner_elements[i].parentElement.getAttribute("id").split('_')
[1];
192             setSelectedFromFieldID(parent_id);
193         } else {
194             console.log("went to the owner else...");
195         }
196     }
197
198     // Loop through all the crop name attributes in each field tablerow, and
199     // check if the user selected crop is the same. If so, select that field.
200     // In order to go "up one" from owner elements to a field element, the parentElement
operator is used.
201     for (i=0; i < crop_elements.length; i++){
202         if(crop_elements[i].outerText.toUpperCase().trim() == crop_name.trim()) {
203             crop_elements[i].parentElement.setAttribute("class", "");
204             var parent_id = crop_elements[i].parentElement.getAttribute("id").split('_')
[1];
205             setSelectedFromFieldID(parent_id);
206         } else {
207             console.log("went to the crop else...");
208         }
209     }
210 }
211
212 function processForm(f){

```

```

213
214 // This function takes a form as an argument and processes that basic search
215 // form. There are two "routes" depending on which of the radio buttons were selected
216 // by the user (ANY or ALL).
217
218 // Start with clean slate
219 clearMap()
220 clearTables()
221
222 // Set the following variables to the options chosen by the user
223 let field_owner = f.option1.value;
224 let crop_name = f.option2.value;
225
226 // Store the state of the radio button in a variable (either 'ALL' or 'ANY')
227 let radio_select = f.and_or.value;
228
229 var owner_elements = document.querySelectorAll('td[class^="owner"]');
230 var crop_elements = document.querySelectorAll('td[class^="crop_name"]');
231
232 // Select route (next function) depending on radio button selection
233 if (radio_select == 'ALL') {processFormAll(field_owner, owner_elements, crop_name,
crop_elements)}
234 if (radio_select == 'ANY') {processFormAny(field_owner, owner_elements, crop_name,
crop_elements)}
235
236 return false
237 }
238
239 function resetForms(){
240
241 // Gets all forms by ID, and resets them
242 // Also clears the map and tables for good measure
243
244 document.getElementById("main_form").reset();
245 document.getElementById("spatial_form").reset();
246 document.getElementById("spatial_form2").reset();
247 console.log("Resetting the forms....")
248 clearMap();
249 clearTables();
250 }
251
252 function setFieldSelected(id){
253
254 // Checks if a field is unselected, and if so, selects it,
255 // and vice-versa
256 var trow_id = "tablerow_".concat(id.id)
257 var trow = document.getElementById(trow_id)
258 if (id.getAttribute("class") == "field_selected") {
259 id.setAttribute("class", "field");
260 trow.setAttribute("class", "hidden");
261 } else {
262 id.setAttribute("class", "field_selected");
263 trow.setAttribute("class", "");
264 }
265 }
266
267 function setFindSelected(id){
268
269 // Checks if a field is unselected, and if so, selects it,
270 // and vice-versa
271 var trow_id = "tablerow_".concat(id.id)
272 var trow = document.getElementById(trow_id)
273 if (id.getAttribute("class") == "find_selected") {
274 id.setAttribute("class", "find");
275 trow.setAttribute("class", "hidden");
276 } else {
277 id.setAttribute("class", "find_selected");
278 trow.setAttribute("class", "")
279 }
280 }
281
282 function setSelectedFromFindID(find_id){
283 // Takes a find_id (eg find6) and selects that find
284 var trow_id = "tablerow_".concat(find_id)
285 var trow = document.getElementById(trow_id);
286 var this_find = document.getElementById(find_id);
287 this_find.setAttribute("class", "find_selected");
288 trow.setAttribute("class", "");

```

```

289     }
290
291     function setSelectedFromFieldID(field_id){
292         // Takes a field (eg field6) and selects that field
293         var trow_id = "tablerow_".concat(field_id)
294         var trow = document.getElementById(trow_id);
295         var this_field = document.getElementById(field_id);
296         this_field.setAttribute("class", "field_selected");
297         trow.setAttribute("class", "");
298     }
299
300     function setUnselectedFromFindID(find_id){
301         // Takes a find_id (eg find6) and unselects that find
302         var trow_id = "tablerow_".concat(find_id)
303         var trow = document.getElementById(trow_id);
304         var this_find = document.getElementById(find_id);
305         this_find.setAttribute("class", "find");
306         trow.setAttribute("class", "");
307     }
308
309     function setUnselectedFromFieldID(field_id){
310         // Takes a field (eg field6) and unselects that field
311         var trow_id = "tablerow_".concat(field_id)
312         var trow = document.getElementById(trow_id);
313         var this_field = document.getElementById(field_id);
314         this_field.setAttribute("class", "field");
315         trow.setAttribute("class", "hidden");
316     }
317
318     function toggleFieldHighlight(field_id){
319
320         // Takes a field_id as an argument - if the field is selected, it unselects
321         // If the field is unselected, it will select it. This is done by changing
322         // the classes between "field" and "field_selected" as necessary
323
324         field = document.getElementById(field_id)
325         if (field.getAttribute("class") == "field_selected") {
326             field.setAttribute("class", "field");
327         } else {
328             field.setAttribute("class", "field_selected");
329         }
330     }
331
332
333     function toggleFindHighlight(find_id){
334
335         // Takes a find_id as an argument - if the find is selected, it unselects
336         // If the find is unselected, it will select it. This is done by changing
337         // the classes between "find" and "find_selected" as necessary
338
339         find = document.getElementById(find_id)
340         if (find.getAttribute("class") == "find_selected") {
341             find.setAttribute("class", "find");
342         } else {
343             find.setAttribute("class", "find_selected");
344         }
345     }

```

Listing 3: JavaScript Code

4 CSS - style.css

```
1  /*
2
3  CSS for styling ArchaeoLogic visualiser
4
5  TIGIS Assignment 3
6  January 2020
7  Student Number 9217953
8
9  */
10
11  *{
12  box-sizing: border-box;
13  margin: 0;
14  padding: 0;
15  }
16
17  body {
18  font-family: "Lucida Sans Unicode", "Lucida Grande", "sans-serif";
19  font-size: 1em;
20  align-items: center;
21  justify-content: center;
22  text-align: center;
23  }
24
25  .header {
26  width: 100%;
27  height: 40px;
28  border-radius: 5px;
29  background-color: #4CAF50;
30  color: white;
31  margin: auto;
32  padding-bottom: 0px;
33  text-align: center;
34  vertical-align: middle;
35  }
36
37  .footer {
38  width: 100%;
39  background-color: #4CAF50;
40  color: white;
41  margin: auto;
42  padding-top: 10px;
43  text-align: center;
44  vertical-align: middle;
45  }
46
47  .content {
48  width: 100%;
49  max-width: 1000px;
50  margin: 0px auto 30px;
51  max-height: 100vh;
52  }
53
54  svg {
55  margin: 0px auto 0px;
56  max-width: 640px;
57  }
58
59  .rect-grid {
60  margin: auto;
61  }
62
63  .column-form {
64  width: 20%;
65  float: left;
66  margin: 5px auto 10px;
67  padding-top: 5px;
68  padding-bottom: 5px;
69  border: dashed #e3effc 1px;
70  }
71
72  .column-map {
73  border: dashed #e3effc 1px;
74  width: 80%;
75  float: left;
```

```

76     margin: 5px auto 10px;
77     clear: both;
78 }
79
80 .column-table {
81     vertical-align: middle;
82     width: 100%;
83     padding-top: 0px;
84     padding-bottom: 5px;
85     margin: 0px auto 0px;
86     clear: both;
87 }
88
89 #field_table1 {
90     padding-top: 2px;
91     border: dashed #e3effc 1px;
92     margin-bottom: 1px;
93 }
94
95 #find_table1 {
96     border: dashed #e3effc 1px;
97     padding-top: 0px;
98     margin-bottom: 0px;
99 }
100 }
101
102 table {
103     border-collapse: collapse;
104     width: 100%;
105 }
106
107 table, th, td {
108     border: 1px solid #ddd;
109 }
110
111 th {
112     height: 2em;
113     font-size: 0.7rem;
114     text-align: center;
115     background-color: #4CAF50;
116     color: white;
117 }
118
119 th.field_number, th.area, th.crop_number {
120     width: 60px;
121 }
122
123 th.lower_left, th.upper_right {
124     width: 100px;
125 }
126
127 th.crop_name, th.find_class_name {
128     width: 120px;
129 }
130
131 th.find_number, th.x, th.y, th.find_type, th.depth {
132     width: 60px;
133 }
134
135 tr {
136     padding: 10px;
137     font-size: 0.7em;
138 }
139
140 tr:hover {background-color: #f5f5f5;}
141
142 h1 {
143     font-size: 1.5em;
144     text-align: center;
145     margin: auto;
146     padding-top: 5px;
147     padding-bottom: 5px;
148 }
149
150 h2 {
151     font-size: 1em;
152     margin-top: 5px;
153     margin-bottom: 0px;

```

```

154     color: black;
155     opacity: 0.6;
156 }
157
158 h3 {
159     font-size: 0.8em;
160     font-style: normal;
161     font-weight: normal;
162     margin-top: 5px;
163     margin-bottom: 0px;
164     color: black;
165     opacity: 0.4;
166 }
167
168 .instructions_text {
169     float: left;
170     margin: auto;
171     padding: 10px;
172     font-size: 0.9em;
173     font-style: oblique;
174     color: #4CAF50;
175 }
176
177 .find {
178     cursor: pointer;
179     fill-opacity: 1.0;
180     stroke: black;
181     stroke-width: 0.02;
182 }
183
184 .find_selected {
185     cursor: pointer;
186     fill-opacity: 1.0;
187     stroke: red;
188     stroke-width: 0.1;
189 }
190
191 .field {
192     cursor: pointer;
193     fill-opacity: 0.5;
194     stroke-width: 0.03;
195     stroke: black;
196 }
197
198 .field_selected {
199     cursor: pointer;
200     fill-opacity: 0.9;
201     stroke-width: 0.1;
202     stroke: blue;
203 }
204
205 .field_number, .find_number {
206     cursor: pointer;
207 }
208
209 .field_number {
210     font-family: Verdana;
211     fill: black;
212 }
213
214 .find_number {
215     font-family: Verdana;
216     fill: white;
217 }
218
219 .form_elements {
220     float: left;
221 }
222
223 .selectors {
224     width: 100%;
225     display: block;
226 }
227
228 hr {
229     width: 99%;
230     margin: auto;
231     border-top: 1px dashed #4CAF50;

```

```

232 }
233
234 .radio_button {
235     width: 20%;
236     margin: 0px;
237     padding: 0px;
238     font-size: 0.75em;
239     align-content: center;
240     vertical-align: middle;
241     float: left;
242     clear: right;
243 }
244
245 .radio_button_fieldset {
246     width: 30%;
247     margin-top: 0px;
248     vertical-align: middle;
249     display: block;
250 }
251 }
252
253 .field_selections {
254     border: 1px solid #ddd;
255 }
256
257 .find_selections {
258     border: 1px solid #ddd;
259 }
260
261 .centering {
262     width: inherit;
263     margin: auto;
264 }
265
266 .dropdown {
267     width: 100%;
268     border: 1px solid #4CAF50;
269     background-color: white;
270     border-radius: 5px;
271     font-style: italic;
272     color: gray;;
273     margin: 2px auto 2px;
274     float: left;
275     padding: 2px auto 2px;
276 }
277 }
278
279 .button_right {
280     border: 1px solid #4CAF50;
281     align-items: center;
282     background-color: white;
283     color: #4CAF50;
284     border-radius: 5px;
285     width: 100%;
286     padding: 3px;
287     margin: 2px auto 2px ;
288     display: block;
289 }
290
291
292 .button {
293     border: 1px solid #4CAF50;
294     align-items: center;
295     background-color: white;
296     border-radius: 5px;
297     width: 25%;
298     margin: 10px;
299     color: #4CAF50;
300     text-align: center;
301     font-size: 0.8em;
302     vertical-align: middle;
303     padding: 5px;
304     float: left;
305 }
306
307 .footer {
308     height: 30px;
309     color: white;

```

```
310     font-size: 0.6em;
311     font-style: italic;
312     border-radius: 5px;
313     background-color: #4CAF50;
314 }
315
316 fieldset {
317     border: none;
318 }
319
320 .hidden {
321     display: none;
322 }
```

Listing 4: CSS Styles