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

```python
#!/usr/bin/env python3

from jinja2 import Environment, FileSystemLoader
import cx_Oracle
import cgi
import cgitb

cgitb.enable(format='text')

class GraphicsArea:

    # Defines an area for SVG graphics to be rendered on the website

    def __init__(self, width, height, viewBox_x, viewBox_y, viewBox_width, viewBox_height):

        # Parameters passed in during creation
        self.width = f"{width}px"                 #  width of SVG element on page
        self.height = f"{height}px"               #  height of SVG element on page
        self.viewBox_x = viewBox_x                #  viewBox min x
        self.viewBox_y = viewBox_y                #  viewBox min y
        self.viewBox_width = viewBox_width        #  viewBox width
        self.viewBox_height = viewBox_height      #  viewBox height

        #  string representation of the whole viewBox:
        self.viewBox_custom = f"{viewBox_x} {viewBox_y} {viewBox_width} {viewBox_height}"


class Field:

    #  Field Object Creator

    def __init__(self, field_id, lowx, lowy, hix, hiy, area, owner, crop_id):

        # Parameters passed in during creation (ie fetched from database)
        self.field_id = field_id        #  a number uniquely identifying the field
        self.lowx = lowx                #  x-coordinate of lower left corner
        self.lowy = lowy                #  y-coordinate of lower left corner
        self.hix = hix                  #  x-coordinate of upper right corner
        self.hiy = hiy                  #  y-coordinate of upper right corner
        self.area = f"{area:.2f}"       #  area, formatted to 2 decimal places
        self.owner = owner              #  name of the farmer who owns the field
        self.crop_id = crop_id          # a number identifying the crop in the field

        #  Derived Attributes (populated at runtime)
        self.finds_in_this_field = []   # a list containing the finds found in this field

        #  Attributes calculated from object properties
        self.width = hix - lowx                         #  field width (in map units)
        self.height = hiy - lowy                        #  field height
        self.centroidx = (hix - lowx)/2 + lowx          #  field centroid x coordinate
        self.centroidy = (hiy - lowy)/2 + lowy          #  field centroid y coordinate

        # Default value for fill is 'none'.  This property is dynamically added at runtime
        self.fill = 'none'

    #  User & coder friendly representations of Field objects
    def __repr__(self):
        return f"Field({self.field_id}, {self.lowx}, {self.lowy}, {self.hix}, {self.hiy})"

    def __str__(self):
        return f"Field {self.field_id} - Bottom Left({self.lowx},{self.lowy}) Top Right({self.hix},{
    self.hiy})"


class Find:

    #  Find Object Creator

    def __init__(self, find_id, xcoord, ycoord, find_type, depth, field_notes):
        self.find_id = find_id           #  a number uniquely identifying the find
        self.xcoord = xcoord             #  x coordinate of find location
        self.ycoord = ycoord             #  y coordinate of find location
        self.find_type = find_type       #  a number identifying the find type (class)
        self.depth = f"{depth:.2f}"      #  the depth of the find, formatted to 2 decimal places
        self.field_notes = field_notes   #  a string field with field notes
        self.class_name = 'none'         #  holds the text name of the crop populated at runtime

        # Derived Attributes

        #  a list of which field(s) the find is found in
        #  (it can be more than 1 field if on the border between fields!)
        self.in_which_fields = []        #  populated at runtime

        self.fill = get_find_colour(self.find_type)  #  applies a colour based on the find type

    #  User & coder friendly representations of Find objects
    def __repr__(self):
        return f"Find({self.find_id}, {self.xcoord}, {self.ycoord})"

    def __str__(self):
        return f"Find {self.find_id} - Coordinates : ({self.xcoord}, {self.ycoord})"


class MyClass:

    #  MyClass Object Creator

    def __init__(self, class_type, name, period, use):
        self.class_type = class_type         #  a number identifying the class type
        self.name = name                     #  a string with the class name
        self.period = period                 #  a string with the period of the class
        self.use = use                       #  a string with the use of the class

        self.fill = 'none'                   #  the colour of a particular class, populated at runtime

    #  User & coder friendly representations of MyClass objects
    def __repr__(self):
        return f"Class({self.class_type}, {self.name}, {self.period}, {self.use})"

    def __str__(self):
        return f"Class # {self.class_type} - {self.name}, Period : {self.period}, Use: {self.use})"


class Crop:

    #  Crop Object Creator

    def __init__(self, crop, name, startseason, endseason):
        self.crop = crop                     #  a number identifying the crop type
        self.name = name                     #  a string identifying the crop name
        self.startseason = startseason       #  the start of the growing season
        self.endseason = endseason           #  the end of the growing season

        self.fill = 'none'                   #  the colour assigned to each crop type, populated at
    runtime

    #  User & coder friendly representations of Crop objects
    def __repr__(self):
        return f"Crop({self.crop}, {self.name}, {self.startseason}, {self.endseason})"

    def __str__(self):
        return f"Crop # {self.crop} - {self.name}, Start of Season: {self.startseason}, End of Season: {self.endseason})"


def getDBdata(table_name, order_column):
    #  Accesses the Oracle database and creates Find, Field, Crop & MyClass
    #  objects with which to create the website visualiser tools
    results = []
    with open('../../../details.txt', 'r') as f:
        pwd = f.readline().strip()
    try:
        conn = cx_Oracle.connect(f"s0092179/{pwd}@geoslearn")
        c = conn.cursor()
        c.execute(f"SELECT * FROM {table_name} ORDER BY {order_column}")
    except:
        print("Failed to connect to Database Server...")

    if table_name == "MY_FIELDS":
        fields_list = []                                #  initialise an empty list
        for row in c:
            (a, b, c, d, e, f, g, h) = row              # pack the results of the query into a tuple
            field_name = table_name[:-1] + str(a)
            field_name = Field(a, b, c, d, e, f, g, h)  # create a new Field object for each row in the table
            fields_list.append(field_name)              # add the newly created field to the "fields_list" list
            results = fields_list                       # return the list of Field objects

    #  The following elif statements handle creation of other objects from the
    #  different tables.  I will not comment these further as the process is the same as for Field ojects
    elif table_name == "MY_FINDS":
        finds_list = []
        for row in c:
            (a, b, c, d, e, f) = row
            find_name = table_name[:-1] + str(a)
            find_name = Find(a, b, c, d, e, f)
            finds_list.append(find_name)
            results = finds_list

    elif table_name == "MY_CLASS":
        classes_list = []
        for row in c:
            (a, b, c, d) = row
            my_class = MyClass(a, b, c, d)
            classes_list.append(my_class)
            results = classes_list

    elif table_name == "MY_CROPS":
        crops_list = []
        for row in c:
            (a, b, c, d) = row
            my_crop = Crop(a, b, c, d)
            crops_list.append(my_crop)
            results = crops_list
    else:                                            #  if no table name matches are made, go to the else...
        print("Table Name not supported...")
    conn.close()                                     #  close the connection to the Oracle server
    return results


def get_which_finds(fields, finds):
    #  loops through fields and finds, and identifies which finds fall within which field,
    #  and also which fields a particular find falls in (could be more than one if on border...)
    #  The program appends any items fulfilling the basic spatial criteria to the relevant field and find objects attribute lists
    for field in fields:
        for find in finds:
            if find.xcoord >= field.lowx and find.xcoord <= field.hix and find.ycoord >= field.lowy and find.ycoord <= field.hiy:
                field.finds_in_this_field.append(find.find_id)
                find.in_which_fields.append(field.field_id)


def get_field_colour(field_crop):
    #  takes a crop name and returns the correct colour for rendering to the web
    if field_crop == 'TURNIPS':
        return '#A647FF'  # purple
    elif field_crop == 'OIL SEED RAPE':
        return '#F3FC30'  # pale yellow
    elif field_crop == 'STRAWBERRIES':
        return '#FD5959'  # orangey red
    elif field_crop == 'PEAS':
        return '#91F708'  # light green
    elif field_crop == 'POTATOES':
        return '#F9C89A'  # lightish orange
    else:
        return 'none'


def get_find_colour(find_class):
    #  takes a class name and returns the correct colour for rendering to the web
    if find_class == 1:
        return '#9AA8F9'  # light blue
    elif find_class == 2:
        return '#C8C8C8'  # light grey
    elif find_class == 3:
        return '#ABC349'  # flinty green
    elif find_class == 4:
        return '#D1BB00'  # mustard colour
    else:
        return 'none'


def get_crop_name(crops, crop_id):
    #  takes a crop id number, and returns a string of the actual crop name
    for crop in crops:
        if crop.crop == crop_id:
            return crop.name
        else:
            continue


def get_class_name(my_class, find_type):
    #  takes a find type, and returns a string of its class name
    for cls in my_class:
        if cls.class_type == find_type:
            return cls.name
        else:
            continue

def get_unique_owners(fields):
    #  simply takes the list of all owners and returns a list of just the unique values
    #  which is used for displaying dropdown menus in the website with no duplicate values
    unique = []
    for field in fields:
        if field.owner not in unique:
            unique.append(field.owner)
    return unique

def get_max_find_coordinates(finds):
    #  returns a list of the maximum x and y co-ordinates for any find
    #  this is used for rendering the co-ordinate options in the dropdown menus
    #  to make sure that the values are sensible and go high enough but not any further than needed
    max = [0,0]
    for find in finds:
        if find.xcoord > max[0]:
            max[0] = find.xcoord
        if find.ycoord > max[1]:
            max[1] = find.ycoord

    return max


def print_svg(width, height, viewbox):
    #  if required, can return a string of the basic SVG tag
    return f'<svg width="{width}" height="{height}" viewBox="{viewbox}">'


def assign_field_colours(fields, crops):
    #  takes a lists of fields and crops, and populates the fill colour attributes
    #  by matching crop id from field objects with crop from crop objects
    #  This results in all fields with a particular crop having the same colour,
    #  and all crop name cells in the tables having the same colour.
    for field in fields:
```

```python
        for crop in crops:
            if field.crop_id == crop.crop:
                field.fill = get_field_colour(crop.name)
                crop.fill = field.fill
            else:
                continue


def assign_find_colours(finds, classes):
    #  takes a lists of finds and classes, and populates the fill colour attributes
    #  by matching find type from find objects with class_type from class objects
    #  This results in all finds with a particular class having the same colour,
    #  and all class name cells in the tables having a matching colour.
    for find in finds:
        for cls in classes:
            if find.find_type == cls.class_type:
                find.fill = get_find_colour(cls.class_type)
                cls.fill = find.fill
            else:
                continue


def assign_crop_names(fields, crops):
    #  populates the crop_name attribute for each field by linking the crop id to the list of crop
    objects
    for field in fields:
        field.crop_name = get_crop_name(crops, field.crop_id)


def assign_class_names(finds, classes):
    #  populates the class_name attribute of each find by linking the find_type to the list of
    MyClass objects
    for find in finds:
        find.class_name = get_class_name(classes, find.find_type)


def render_html():
    #  Uses Jinja2 to render an html template and passing in a long list of objects and variables
    from python to the html page
    env = Environment(loader=FileSystemLoader('.'))
    temp = env.get_template('index.html')
    print(temp.render(fields=field_objects, finds=find_objects, classes=my_classes, crops=my_crops, g
    =graphics_area_for_svg, unique_owners=unique_owners, max_find_coords=max_find_coords))


if __name__ == '__main__':

    print("Content-type: text/html\n")      #  Ensures that the webpage is rendered correctly using
    HTML code

    #  Create new lists of objects for each of the main types of interest
    my_classes = getDBdata("MY_CLASS", "TYPE")
    my_crops = getDBdata("MY_CROPS", "CROP")
    field_objects = getDBdata("MY_FIELDS", "FIELD_ID")
    find_objects = getDBdata("MY_FINDS", "FIND_ID")

    #  run the required functions to complete the population of the important fields
    #  in each of the objects by cross referencing between object attributes
    assign_field_colours(field_objects, my_crops)
    assign_find_colours(find_objects, my_classes)
    assign_crop_names(field_objects, my_crops)
    assign_class_names(find_objects, my_classes)
    unique_owners = get_unique_owners(field_objects)
    get_which_finds(field_objects, find_objects)
    max_find_coords = get_max_find_coordinates(find_objects)

    #  Define a new GraphicsArea object which contains the necessary attributes for the SVG rendering
     on the website
    #  (SVG container width in pixels, SVG conatiner height in pixels, viewBox min x, ViewBox min y,
    viewBox width, viewBox height)
    graphics_area_for_svg = GraphicsArea(420, 530, -1, 1, 16, 18)

    #  render the html template to the browser
    render_html()
```

Listing 1: Python3 Code

2 HTML Markup - index.html

```html
<!DOCTYPE html>

<!-- TIGIS Assignment 3 - ArchaeoLogic Visualiser -->
<!-- January 2020 -->
<!-- Student Number 9217953 -->

<html lang="en" dir="ltr">
  <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- Add link to external stylesheet    -->
      <link rel="stylesheet" href="../../style.css">
      <title>ArchaeoLogic Visualiser</title>
  </head>
  <!--     -->
  <body>
    <div class="content">
        <div class="header">
            <h1 id="main_title">ArchaeoLogic</h1>
        </div>
          <div class="column-map">

              <h2>Map of Fields & Finds</h2>
              <h3>Axes labelled in Map Units - Click to Select</h3>

                <!-- Main Container DIV for the SVG Map    -->

                <svg width="{{ g.width }}" height="{{ g.height }}" viewBox="{{ g.viewBox_custom }}">
                <!-- The SVG Map is rendered dynamically using Jinja2 to access variables in the Python classes associated with Finds & Fields
                    Furthermore, the variable "g" is a GraphicsArea class defined in python and containing information on the SVG viewBox,
    height, and width -->
                <!--<svg width="100%" height="100%" viewBox="{{ g.viewBox_custom }}">-->
                  <!-- The transform(scale(1,-1)) flips the coordinates around the x-axis, and translate relocates the flipped axes in the
    correct place by shifting
                      by an amount equal to the total SVG viewBox height.  This is done because the SVG co-ordinates are a graphical
    co-ordinate system and we need to display a
                      geographical co-ordinate system (with origin at bottom left rather than top right...)-->
                    <g id="draw_svg" transform="scale(1,-1) translate(0, -{{ g.viewBox_height }})">
                        <!-- Draw the backgrond grid for the field and find map -->
                        <!-- Two patterns are used : smallGrid to do small gird squares, and grid to render the larger ones with slightly
    thicker lines    -->
                        <pattern id="smallGrid" width="0.25" height="0.25" patternUnits="userSpaceOnUse">
                            <path d="M 1 0 L 0 0 0 1" fill="none" stroke="#e3effc" stroke-width="0.04"/>
                        </pattern>
                        <pattern id="grid" width="1" height="1" patternUnits="userSpaceOnUse">
                            <rect width="13" height="17" fill="url(#smallGrid)"/>
                            <path d="M 10 0 L 0 0 0 10" fill="none" stroke="#ccddf0" stroke-width="0.1"/>
                        </pattern>
                        <rect class="rect-grid" width="13.03" height="17.03" fill="url(#grid)" />

                        <!-- Loop through the fields and draw the SVG rectangle for each field in the variable "fields" -->
```

```
               <!-- Note that both fields and finds have JS code associated so that when they are clicked, they are toggled between "selected" (appearance changes and information appears) and "unselected"  -->
               {% for field in fields %}
                 <rect class="field" id="field{{ field.field_id }}" onclick="setFieldSelected(this)" x="{{ field.lowx }}" y="{{ field.lowy }}" width="{{ field.width }}" height="{{ field.height }}" fill="{{ field.fill }}" stroke="black"/>
               {% endfor %}
               <!-- Loop through the finds and draw the SVG circle for each find in the varable "finds" -->
               {% for find in finds %}
                 <circle class="find" id="find{{ find.find_id }}" onclick="setFindSelected(this)" cx = "{{ find.xcoord }}" cy="{{ find.ycoord }}" r="0.4" fill="{{ find.fill }}"/>
               {% endfor %}
             </g>
             <g id="axes">
              <!-- The following 2 loops go through the x and y co-ordinates and render numbers in the appropriate places (using x,y co-ordinates and "translate" to tweak the positions)    -->
               {% for x in range(g.viewBox_width-2) %}
               <text class="x_axis_label" transform="translate(-0.1, 0)" x="{{ x }}" y="{{ g.viewBox_height + 0.5 }}" fill="gray" font-size="0.3">{{ x }}</text>
               {% endfor %}
               {% for y in range(g.viewBox_height) %}
               <text class="y_axis_label" transform="translate(0, 0.1)" x="{{ -0.7 }}" y="{{ g.viewBox_height-y }}" fill="gray" font-size="0.3">{{ y }}</text>
               {% endfor %}

             </g>
             <g id="text_numbers">
               <!-- The following 2 loops go through the fields and finds and render numbers in the appropriate places (using x,y co-oridinates and "translate" to tweak the positions)      -->
               {% for field in fields %}
                 <text id="text_field{{ field.field_id }}" class="field_number" transform="translate(-0.2, 0)" x="{{ field.centroidx }}" y="{{ g.viewBox_height-field.centroidy }}" font-size="0.5">{{field.field_id}}</text>
               {% endfor %}
               {% for find in finds %}
                 <text id="text_find{{ find.find_id }}" class="find_number" onclick="setFindSelected(find{{ find.find_id }})" transform="translate(-0.15, 0.2)" x="{{ find.xcoord }}" y="{{ g.viewBox_height-find.ycoord }}" font-size="0.5">{{find.find_id}}</text>
               {% endfor %}
             </g>
           </svg>

           <!--  Render three buttons, each with JS functions triggered when clicked, to clear and reset the various page elements   -->
           <div class="centering">
             <input class="button" type="button" name="clear_map" value="Clear Map" onclick="clearMap()">
             <input class="button" type="button" name="clear_tables" value="Clear Tables" onclick="clearTables()">
             <input class="button" type="reset" value="Reset All..." onclick="resetForms()">
           </div>
         </div>

         <!--  The main container DIV for the Search and Query Forms  -->
        <div class="column-form">

          <!-- The most basic search form, showing 2 dropdown menus to select by Farmer and/or Crop     -->

          <p class="instructions_text">Basic Search - Select items matching the following criteria:</p>
          <form id="main_form" name="main_form" action="#" method="post" onsubmit="return processForm(this)">
```

```html
              <div class="form_elements">

                  <!--  Fieldsets are used to group similar form elements together     -->
                  <fieldset class="selectors">

                      <!--  Dropdown menus are rendered by looping through the required variable and creating an "option" for each one -->
                         <select class="dropdown" name="option1">
                               <option selected disabled>Field Owner...</option>
                           {% for owner in unique_owners %}
                               <option value="{{owner}}">{{ owner.title() }}</option>
                           {% endfor %}
                         </select>
                         <select class="dropdown" name="option2">
                                 <option selected disabled>Crop Type...</option>
                             {% for crop in crops %}
                                 <option value="{{crop.name}}">{{ crop.name.title() }}</option>
                             {% endfor %}
                         </select>
                  </fieldset>
              </div>
              <div>
                  <br>
                  <br>
                  <p></p>
                  <!--  Render the buttons that will control the form above     -->
                    <p class="instructions_text">Match...</p>
                    <fieldset class="radio_button_fieldset">
                      <input class="radio_button" type="radio" name="and_or" id="select_or" checked value="ANY">Any<br>
                      <input class="radio_button" type="radio" name="and_or" id="select_and" value="ALL">Both<br>
                      <br>
                    </fieldset>
                  <fieldset class="buttons">
                      <input class="button_right" type="submit" value="Search...">
                  </fieldset>
              </div>
          </form>

          <!-- The first spatial search form, which triggers the JS function "processSpatialForm()", and passes the form itself through to the function.
          The layout is similar to the above, but x and y coordinates are rendered up to the values of the highest
          coordinates of any find (calculated in the python code and passed through using Jinja2)
          This ensures that the values in the dropdown menus are sensible and cover all necessary values.
          NOTE:  +1 is added to each list because of the index values starting at 0 and only
          reaching up to the (length of the list - 1). -->

          <form id="spatial_form" name="spatial_form" action="#" method="post" onsubmit="return processSpatialForm(this)">
              <div class="form_elements">
                <br>
                <hr>

                <p class="instructions_text">Spatial Search - Select finds within an area:</p>
                  <fieldset class="selectors">
                      <select class="dropdown" name="lowleftx">
                  <option selected disabled>Lower Left X Coordinate</option>
              {% for xcoord in range(0,max_find_coords[0]+1) %}
                  <option value="{{xcoord}}">{{ xcoord }}</option>
              {% endfor %}
            </select>
            <select class="dropdown" name="lowlefty">
                  <option selected disabled>Lower Left Y Coordinate</option>
              {% for ycoord in range(0,max_find_coords[1]+1) %}
                  <option value="{{ycoord}}">{{ ycoord }}</option>
              {% endfor %}
            </select>
        </fieldset>
        <fieldset class="selectors">
            <select class="dropdown" name="highrightx">
                  <option selected disabled>Upper Right X Coordinate</option>
              {% for xcoord in range(0,max_find_coords[0]+1) %}
                  <option value="{{xcoord}}">{{ xcoord }}</option>
              {% endfor %}
            </select>
            <select class="dropdown" name="highrighty">
                  <option selected disabled>Upper Right Y Coordinate</option>
              {% for ycoord in range(0,max_find_coords[1]+1) %}
                  <option value="{{ycoord}}">{{ ycoord }}</option>
              {% endfor %}
            </select>
        </fieldset>
    </div>
    <div>
        <fieldset class="buttons">
            <input class="button_right" type="submit" value="Search...">
            <br>
        </fieldset>
    </div>
</form>

<!-- The second spatial search form.  This is very similar to the first, but calls the functions
"processSpatialForm2()" instead, and again passes the form itself through to the function so that it has
access to the user input -->

<form id="spatial_form2" name="spatial_form2" action="#" method="post" onsubmit="return processSpatialForm2(this)">
    <div class="form_elements">
      <br>
      <hr>

      <p class="instructions_text">Spatial Search - Select finds within a radius of a point:</p>
        <fieldset class="selectors">
            <select class="dropdown" name="centre_x">
                  <option selected disabled>Centre Point X</option>
              {% for xcoord in range(0,max_find_coords[0]+1) %}
                  <option value="{{xcoord}}">{{ xcoord }}</option>
              {% endfor %}
            </select>
            <select class="dropdown" name="centre_y">
                  <option selected disabled>Centre Point Y</option>
```

```
         {% for ycoord in range(0,max_find_coords[1]+1) %}
           <option value="{{ycoord}}">{{ ycoord }}</option>
         {% endfor %}
        </select>
      </fieldset>
      <fieldset class="selectors">
        <select class="dropdown" name="radius">
           <option selected disabled>Radius</option>
         {% for xcoord in range(0,max_find_coords[0]+1) %}
           <option value="{{xcoord}}">{{ xcoord }}</option>
         {% endfor %}
        </select>
      </fieldset>
    </div>
    <div>
      <fieldset class="buttons">
        <input class="button_right" type="submit" value="Search...">
        <br>
      </fieldset>
    </div>
  </form>


</div>

<!--  The main container DIV for the Field and Find Tables to display all the necessary information
on whichever fields and finds are selected  -->

<div class="column-table">

<!--  These tables are created dynamically, again by looping through variables and adding rows for each field and find.
The key is that by default they are rendered with class="hidden", which means they are not visible on the screen.  When
a given entity is selected by the user or by JS code, the class attribute is altered to remove "hidden", and "find_selected"
or "field_selected" is added instead -->

<!-- RENDER FIELD INFORMATION IN TABLE FORM   -->

   <div class="field_table" id="field_table1">
       <h2>Field Information</h2>
       <table>
         <tr>
           <th>Field Number</th>
           <th>Owner</th>
           <th>Lower Left</th>
           <th>Upper Right</th>
           <th>Area</th>
           <th>Crop Number</th>
           <th>Crop Name</th>
           <th>Contains Find(s)</th>
         </tr>
         {% for field in fields %}
           <tr class="hidden" id="tablerow_field{{field.field_id}}">
             <td class="field_number"> {{field.field_id}} </td>
             <td class="owner"> {{field.owner.title()}}</td>
```

```html
                  <td class="lower_left"> {{field.lowx}}, {{field.lowy}} </td>
                  <td class="upper_right"> {{field.hix}}, {{field.hiy}} </td>
                  <td class="area"> {{field.area}} </td>
                  <td class="crop_number"> {{field.crop_id}}</td>
                  <td class="crop_name" style="background-color:{{field.fill}}"> {{field.crop_name.title()}}</td>
                  <td class="finds_in_this_field"> {% for find in field.finds_in_this_field %} ({{find}}) {% endfor %}</td>
                </tr>
              {% endfor %}
            </table><br>
        </div>

        <!-- RENDER FIND INFORMATION IN TABLE FORM   -->

       <div class="find_table" id="find_table1">
        <h2>Find Information</h2>
          <table>
            <tr>
              <th>Find Number</th>
              <th>X</th>
              <th>Y</th>
              <th>Type</th>
              <th>Class</th>
              <th>Depth</th>
              <th>Is In Field(s)</th>
              <th>Field Notes</th>
            </tr>
            {% for find in finds %}
              <tr class="hidden" id="tablerow_find{{find.find_id}}">
                <td class="find_number"> {{find.find_id}} </td>
                <td class="x" id="find{{find.find_id}}_xcoord"> {{find.xcoord}} </td>
                <td class="y" id="find{{find.find_id}}_ycoord"> {{find.ycoord}} </td>
                <td class="find_type"> {{find.find_type}} </td>
                <td class="find_class_name" style="background-color:{{find.fill}}"> {{find.class_name.title()}} </td>
                <td class="depth"> {{find.depth}} </td>
                <td class="in_which_fields">{% for field in find.in_which_fields %} ({{field}}) {% endfor %}</td>
                <td class="field_notes"> {{find.field_notes.title()}} </td>
              </tr>
            {% endfor %}
          </table><br>
        </div>
      </div>

      <!-- A basic footer with design, author, and date information   -->

      <div class="footer">
        <p>designed by student number 9217953, january 2020</p>
      </div>
  </div>
  <script type="text/javascript" src="../../main.js"></script>
 </body>
</html>
```

Listing 2: HTML Markup

3 JavaScript - main.js

```
//  JavaScript to support the operation of the ArcheoLogic Visualiser website
//  January 2020
//  Student Number 9217953

    function processSpatialForm2(f) {

      //  This function takes a spatial search form as an argument, and processes
      //  it to identify finds within a certain radius of a central point

      //  NOTE:  All calcalations are done using arbitrary Map Units

      // First clear the map and tables for a clean slate
      clearMap();
      clearTables();

      // Set up variables by getting user input data from the form
      var centre_x = f.centre_x.value  // search centre point x-coordinate
      var centre_y = f.centre_y.value  // search centre point y-coordinate
      var radius = f.radius.value      // search radius

      // Get all find x and y coordinates by selecting table cells with a class of "x" or "y"
      var find_x_elements = document.querySelectorAll('td[class^="x"]')
      var find_y_elements = document.querySelectorAll('td[class^="y"]');

      // A variable to keep track of how many finds are identified
      var finds_found = 0;


      // Loop through the x and y elements, convert the text to integer values,
      // then perform simple Pythagorean geometry to identify those within a certain radius
      for (i=0; i < find_x_elements.length; i++) {
          let x = parseInt(find_x_elements[i].innerText, 10);
          let y = parseInt(find_y_elements[i].innerText, 10);
          let y_difference = Math.abs(y - centre_y);
          let x_difference = Math.abs(x - centre_x);
          let straight_line_distance = Math.sqrt(Math.pow(x_difference, 2) + Math.pow(
    y_difference, 2))
          //  If a find is closer than or equal to the radius set, run through this code:
          if (straight_line_distance <= radius){
            finds_found +=1;
            var find_number = i+1;
            var local_find_id = ("find").concat(find_number);
            setSelectedFromFindID(local_find_id);  //  Calls JS function to set the find as
    selected and show it on the map and tables
            //  Console.log used for debugging
            console.log("Find ", i+1 , "is within ", radius, "map units of the centre point and
    has find_id of ", local_find_id)
          }
       }
       //  If no successful finds were identified, clear the map from any previous selections,
    and alert the user to try again:
       if (finds_found == 0) {
          clearMap();
          var alert_string = ("No match within " + radius + " map units of (" + centre_x + "," +
    centre_y + ") : please try again...");
          alert(alert_string);
       }
       return false
      }


    function processSpatialForm(f) {

      //  This function takes a spatial search form as an argument, and processes
      //  it to identify finds that fall within the search area, a rectangle
      //  defined by the x,y coordinates of the lower left and top right corners.

      //  NOTE:  All calcalations are done using arbitrary Map Units

      // First clear the map and tables for a clean slate
      clearMap();
      clearTables();

      //  Set up variables by getting user input data from the form
      //  In this case they represent the lower left and upper right coordinates
      //  of the search area
      var lowleftx = f.lowleftx.value
      var lowlefty = f.lowlefty.value
      var highrightx = f.highrightx.value
      var highrighty = f.highrighty.value

    // Get all find x and y coordinates by selecting table cells with a class of "x" or "y"
      var find_x_elements = document.querySelectorAll('td[class^="x"]')
      var find_y_elements = document.querySelectorAll('td[class^="y"]');

      //  Loop through the find x, y coordinates and check if that find falls within
      //  the search rectangle (NOTE: or right on the border of it!)
      for (i=0; i < find_x_elements.length; i++) {
          let x = parseInt(find_x_elements[i].innerText, 10);  //  converts text string to integer
          let y = parseInt(find_y_elements[i].innerText, 10);
          //  Search logic :  && operator is used to match all conditions
          if ((x >= lowleftx) && (x <= highrightx) && (y >= lowlefty) && (y <= highrighty)){
            var find_number = i+1;
            var local_find_id = ("find").concat(find_number);
            setSelectedFromFindID(local_find_id);  //  Trigger JS function to handle selecting this find
            console.log("Find ", find_number , "is in the search area and has find_id of ", local_find_id)
          }
      }
       return false
      }


    function clearMap() {

        //  This funtions simply identifies all Fields and Finds marked as "selected"
        //  by including the class of "find_selected" or "field_selected", and loops through
        //  each one to reset the class to "find" or "field" respectively

        var selected_fields = document.querySelectorAll('rect[class^="field_selected"]')
        var selected_finds = document.querySelectorAll('circle[class^="find_selected"]')

        //  Loop through selected fields on the map, and reset to unselected state
        for (i=0; i<selected_fields.length; i++){
            console.log(selected_fields[i])
            selected_fields[i].setAttribute("class", "field")
        }
        //  Loop through selected finds on the map, and reset to unselected state
        for (i=0; i<selected_finds.length; i++){
            console.log(selected_finds[i])
            selected_finds[i].setAttribute("class", "find")
        }
    }

    function clearTables() {

        //  This funtions simply identifies all field and find table rows that do NOT have a class of "hidden"
        //  (ie are visible), and loops through each one to reset the class to "hidden"

        //  Get all the visible field and find table rows and store in 2 variables
        var field_tablerow_elements = document.querySelectorAll('tr[class=""]');
        var find_tablerow_elements = document.querySelectorAll('tr[class=""]');

        console.log("field_tablerow_elements: ", field_tablerow_elements)
        console.log("find_tablerow_elements: ", find_tablerow_elements)

        //  Loop through all visible field table rows and reset to hidden
        for (i=0; i<field_tablerow_elements.length; i++){
            console.log(field_tablerow_elements[i])
            field_tablerow_elements[i].setAttribute("class", "hidden")
        }
        //  Loop through all visible find table rows and reset to hidden
        for (i=0; i<find_tablerow_elements.length; i++){
            console.log(find_tablerow_elements[i])
            find_tablerow_elements[i].setAttribute("class", "hidden")
        }
    }

    function processFormAll(field_owner, owner_elements, crop_name, crop_elements) {

```

```
    //  this function takes user-selected elements and tablerow elements
    //  and compares them to identify fields that match ALL criteria, ie that have both
    //  the chosen owner name and also crop name.

    //  Set up local variables

    var owner_index;
    var crop_index;
    var class_index;

    // Get all field and find table rows by selecting table rows with an id matching the pattern
    var all_field_rows = document.querySelectorAll('tr[id^="tablerow_field"]');
    var all_find_rows = document.querySelectorAll('tr[id^="tablerow_find"]');

    //  Loop through all the rows, and get local variables from their children elements
    //  NOTE: The children elements are the <td> elements ie the individual cells in the tables
    //  (ie to "get inside" the table rows and extract info, then set variables equal to that info)
    //  Note also that the trim() and toUpperCase() functions are used to "clean" up and standardise the data
    for (i=0; i < all_field_rows.length; i++) {
        let this_field = ("field").concat(all_field_rows[i].children[0].innerText.trim());
        let this_owner = all_field_rows[i].children[1].innerText.trim().toUpperCase();
        let this_crop = all_field_rows[i].children[6].innerText.trim().toUpperCase();
        let finds_in_this_field = all_field_rows[i].children[7].innerText.trim();
        if ((field_owner == this_owner) && (crop_name == this_crop)){
          console.log("Match FOUND");  //  Console logging only used for debugging
          setSelectedFromFieldID(this_field);  //  Use external JS function to set the field as selected
        } else {
          console.log("NO MATCH")
        }
      }
  return false;
}

function processFormAny(field_owner, owner_elements, crop_name, crop_elements) {

  //  this function takes user-selected elements and tablerow elements
  //  and compares them to identify fields that match ANY criteria, ie that have either
  //  the chosen owner name or crop name.  Those that are matched are selected.

  //  Loop through all the owner attributes in each field tablerow, and
  //  check if the user selected owner is the same.  If so, select that field.
  //  In order to go "up one" from owner elements to a field element, the parentElement operator is used.
  //  Note also that the trim() and toUpperCase() functions are used to "clean" up and standardise the data
    for (i=0; i < owner_elements.length; i++){
        if(owner_elements[i].outerText.toUpperCase().trim() == field_owner.trim()) {
              owner_elements[i].parentElement.setAttribute("class", "");
              var parent_id = owner_elements[i].parentElement.getAttribute("id").split('_')[1];
              setSelectedFromFieldID(parent_id);
        } else {
              console.log("went to the owner else...");
        }
    }

    //  Loop through all the crop name attributes in each field tablerow, and
    //  check if the user selected crop is the same.  If so, select that field.
    //  In order to go "up one" from owner elements to a field element, the parentElement operator is used.
    for (i=0; i < crop_elements.length; i++){
        if(crop_elements[i].outerText.toUpperCase().trim() == crop_name.trim()) {
              crop_elements[i].parentElement.setAttribute("class", "");
              var parent_id = crop_elements[i].parentElement.getAttribute("id").split('_')[1];
              setSelectedFromFieldID(parent_id);
        } else {
              console.log("went to the crop else...");
        }
    }
  }

function processForm(f){

    //  This function takes a form as an argument and processes that basic search
    //  form.  There are two "routes" depending on which of the radio buttons were selected
    //  by the user (ANY or ALL).

    //  Start with clean slate
    clearMap()
    clearTables()

    //  Set the following variables to the options chosen by the user
    let field_owner = f.option1.value;
    let crop_name = f.option2.value;

    //  Store the state of the radio button in a variable (either 'ALL' or 'ANY')
    let radio_select = f.and_or.value;

    var owner_elements = document.querySelectorAll('td[class^="owner"]');
    var crop_elements = document.querySelectorAll('td[class^="crop_name"]');

    //  Select route (next function) depending on radio button selection
    if (radio_select == 'ALL') {processFormAll(field_owner, owner_elements, crop_name,
crop_elements)}
    if (radio_select == 'ANY') {processFormAny(field_owner, owner_elements, crop_name,
crop_elements)}

    return false
}

function resetForms(){

    //  Gets all forms by ID, and resets them
    //  Also clears the map and tables for good measure

    document.getElementById("main_form").reset();
    document.getElementById("spatial_form").reset();
    document.getElementById("spatial_form2").reset();
    console.log("Resetting the forms....")
    clearMap();
    clearTables();
}

function setFieldSelected(id){

    //  Checks if a field is unselected, and if so, selects it,
    //  and vice-versa
    var trow_id = "tablerow_".concat(id.id)
    var trow = document.getElementById(trow_id)
    if (id.getAttribute("class") == "field_selected") {
        id.setAttribute("class", "field");
        trow.setAttribute("class", "hidden");
    } else {
        id.setAttribute("class", "field_selected");
        trow.setAttribute("class", "");
    }
}

function setFindSelected(id){

  //  Checks if a field is unselected, and if so, selects it,
  //  and vice-versa
    var trow_id = "tablerow_".concat(id.id)
    var trow = document.getElementById(trow_id)
    if (id.getAttribute("class") == "find_selected") {
        id.setAttribute("class", "find");
        trow.setAttribute("class", "hidden");
    } else {
        id.setAttribute("class", "find_selected");
        trow.setAttribute("class", "")
    }
}

function setSelectedFromFindID(find_id){
    //  Takes a find_id (eg find6) and selects that find
    var trow_id = "tablerow_".concat(find_id)
    var trow = document.getElementById(trow_id);
    var this_find = document.getElementById(find_id);
    this_find.setAttribute("class", "find_selected");
    trow.setAttribute("class", "");
}

function setSelectedFromFieldID(field_id){
  //  Takes a field (eg field6) and selects that field
    var trow_id = "tablerow_".concat(field_id)
    var trow = document.getElementById(trow_id);
    var this_field = document.getElementById(field_id);
    this_field.setAttribute("class", "field_selected");
    trow.setAttribute("class", "");
}

function setUnselectedFromFindID(find_id){
    //  Takes a find_id (eg find6) and unselects that find
    var trow_id = "tablerow_".concat(find_id)
    var trow = document.getElementById(trow_id);
    var this_find = document.getElementById(find_id);
    this_find.setAttribute("class", "find");
    trow.setAttribute("class", "");
}

function setUnselectedFromFieldID(field_id){
    //  Takes a field (eg field6) and unselects that field
    var trow_id = "tablerow_".concat(field_id)
    var trow = document.getElementById(trow_id);
    var this_field = document.getElementById(field_id);
    this_field.setAttribute("class", "field");
    trow.setAttribute("class", "hidden");
}

function toggleFieldHighlight(field_id){

    //  Takes a field_id as an argument - if the field is selected, it unselects
    //  If the field is unselected, it will select it.  This is done by changing
    //  the classes between "field" and "field_selected" as necessary

    field = document.getElementById(field_id)
    if (field.getAttribute("class") == "field_selected") {
        field.setAttribute("class", "field");
    } else {
        field.setAttribute("class", "field_selected");
    }
}


function toggleFindHighlight(find_id){

    //  Takes a find_id as an argument - if the find is selected, it unselects
    //  If the find is unselected, it will select it.  This is done by changing
    //  the classes between "find" and "find_selected" as necessary

    find = document.getElementById(find_id)
    if (find.getAttribute("class") == "find_selected") {
        find.setAttribute("class", "find");
    } else {
        find.setAttribute("class", "find_selected");
    }
}
```
Listing 3: JavaScript Code

4 CSS - style.css

```css
/*

CSS for styling ArchaeoLogic visualiser

TIGIS Assignment 3
January 2020
Student Number 9217953

*/

*{
box-sizing: border-box;
margin: 0;
padding: 0;
}

body {
    font-family: "Lucida Sans Unicode", "Lucida Grande", "sans-serif";
    font-size: 1em;
    align-items: center;
    justify-content: center;
    text-align: center;
}

.header {
    width: 100%;
    height: 40px;
    border-radius: 5px;
    background-color: #4CAF50;
    color: white;
    margin: auto;
    padding-bottom: 0px;
    text-align: center;
    vertical-align: middle;
}

.footer {
    width: 100%;
    background-color: #4CAF50;
    color: white;
    margin: auto;
    padding-top: 10px;
    text-align: center;
    vertical-align: middle;
}

.content {
    width: 100%;
    max-width: 1000px;
    margin: 0px auto 30px;
    max-height: 100vh;
}

svg {
  margin: 0px auto 0px;
  max-width: 640px;
}

.rect-grid {
    margin: auto;
}

.column-form {
    width: 20%;
    float: left;
    margin: 5px auto 10px;
    padding-top: 5px;
    padding-bottom: 5px;
    border: dashed #e3effc 1px;
}

.column-map {
    border: dashed #e3effc 1px;
    width: 80%;
    float: left;
    margin: 5px auto 10px;
    clear: both;
}

.column-table {
    vertical-align: middle;
    width: 100%;
    padding-top: 0px;
    padding-bottom: 5px;
    margin: 0px auto 0px;
    clear: both;
}

#field_table1 {
  padding-top: 2px;
  border: dashed #e3effc 1px;
  margin-bottom: 1px;
}

#find_table1 {
  border: dashed #e3effc 1px;
  padding-top: 0px;
  margin-bottom: 0px;

}

table {
    border-collapse: collapse;
    width: 100%;
}

table, th, td {
    border: 1px solid #ddd;
}

th {
    height: 2em;
    font-size: 0.7rem;
    text-align: center;
    background-color: #4CAF50;
    color: white;
}

th.field_number, th.area, th.crop_number {
    width: 60px;
}

th.lower_left, th.upper_right {
    width: 100px;
}

th.crop_name, th.find_class_name {
    width: 120px;
}

th.find_number, th.x, th.y, th.find_type, th.depth {
    width: 60px;
}

tr {
    padding: 10px;
    font-size: 0.7em;
}

tr:hover {background-color: #f5f5f5;}

h1 {
    font-size: 1.5em;
    text-align: center;
    margin: auto;
    padding-top: 5px;
    padding-bottom: 5px;
}

h2 {
    font-size: 1em;
    margin-top: 5px;
    margin-bottom: 0px;
    color: black;
    opacity: 0.6;
}

h3 {
    font-size: 0.8em;
    font-style: normal;
    font-weight: normal;
    margin-top: 5px;
    margin-bottom: 0px;
    color: black;
    opacity: 0.4;
}

.instructions_text {
  float: left;
  margin: auto;
  padding: 10px;
  font-size: 0.9em;
  font-style: oblique;
  color: #4CAF50;
}

.find {
    cursor: pointer;
    fill-opacity: 1.0;
    stroke: black;
    stroke-width: 0.02;
}

.find_selected {
    cursor: pointer;
    fill-opacity: 1.0;
    stroke: red;
    stroke-width: 0.1;
}

.field {
    cursor: pointer;
    fill-opacity: 0.5;
    stroke-width: 0.03;
    stroke: black;
}

.field_selected {
    cursor: pointer;
    fill-opacity: 0.9;
    stroke-width: 0.1;
    stroke: blue;
}

.field_number, .find_number {
    cursor: pointer;
}

.field_number {
    font-family: Verdana;
    fill: black;
}

.find_number {
  font-family: Verdana;
  fill: white;
}

.form_elements {
  float: left;
}

.selectors {
  width: 100%;
  display: block;
}

hr {
  width: 99%;
  margin: auto;
  border-top: 1px dashed #4CAF50;
```

```
}

.radio_button {
  width: 20%;
  margin: 0px;
  padding: 0px;
  font-size: 0.75em;
  align-content: center;
  vertical-align: middle;
  float: left;
  clear: right;
}

.radio_button_fieldset {
  width: 30%;
  margin-top: 0px;
  vertical-align: middle;
  display: block;

}

.field_selections {
  border: 1px solid #ddd;
}

.find_selections {
  border: 1px solid #ddd;
}

.centering {
  width: inherit;
  margin: auto;
}

.dropdown {
  width: 100%;
  border: 1px solid #4CAF50;
  background-color: white;
  border-radius: 5px;
  font-style: italic;
  color: gray;;
  margin: 2px auto 2px;
  float: left;
  padding: 2px auto 2px;

}

.button_right {
  border: 1px solid #4CAF50;
  align-items: center;
  background-color: white;
  color: #4CAF50;
  border-radius: 5px;
  width: 100%;
  padding: 3px;
  margin: 2px auto 2px ;
  display: block;
}


.button {
  border: 1px solid #4CAF50;
  align-items: center;
  background-color: white;
  border-radius: 5px;
  width: 25%;
  margin: 10px;
  color: #4CAF50;
  text-align: center;
  font-size: 0.8em;
  vertical-align: middle;
  padding: 5px;
  float: left;
}

.footer {
  height: 30px;
  color: white;
```

```css
  font-size: 0.6em;
  font-style: italic;
  border-radius: 5px;
  background-color: #4CAF50;
}

fieldset {
  border: none;
}

.hidden {
    display: none;
}
```

Listing 4: CSS Styles