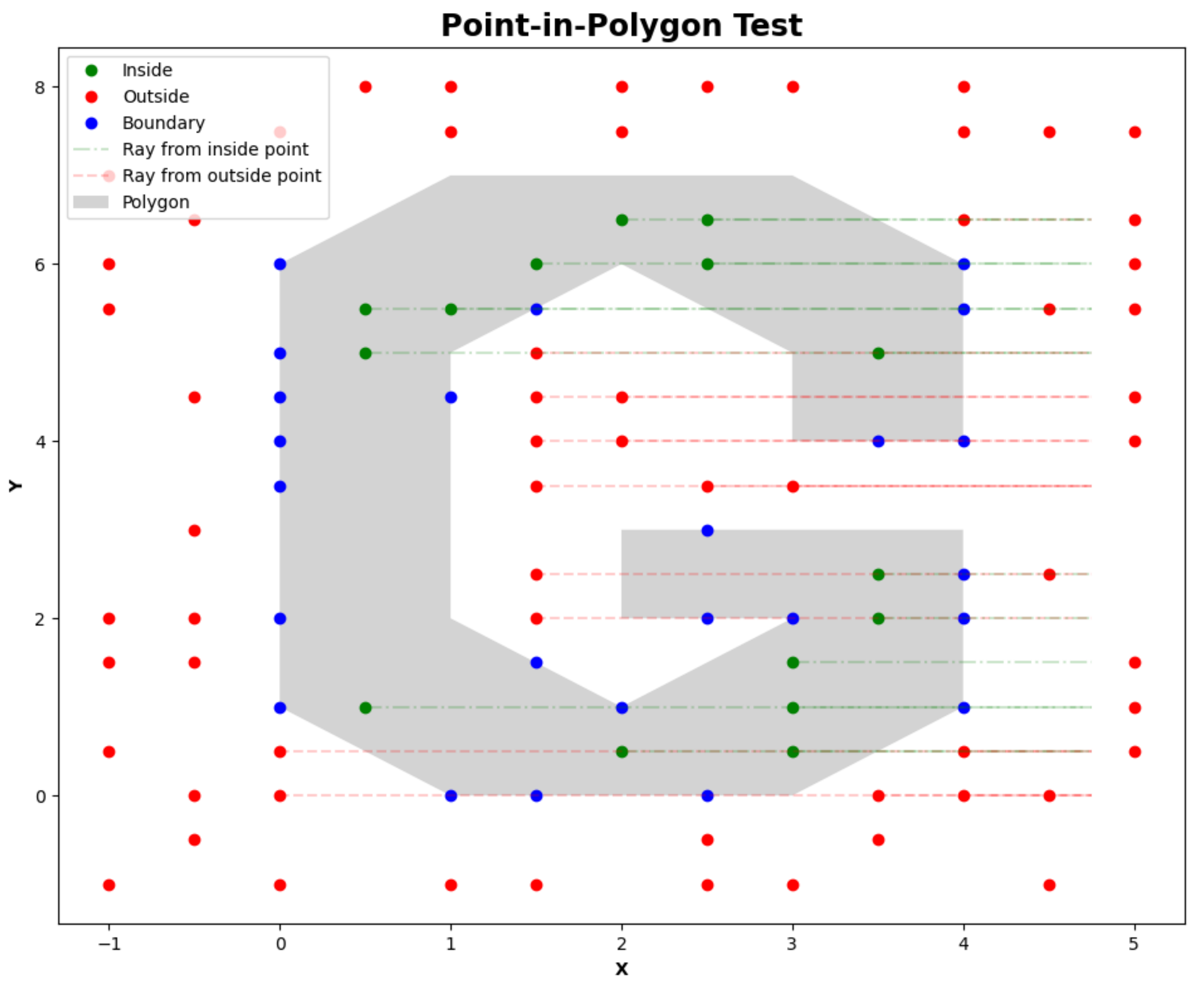
Point-in-Polygon Test

Report for the CEGE0096: 1st Assignment



Name: Zonghe Ma

Portico Number: uceszm4

Email: zonghe.ma.22@ucl.ac.uk

Introduction

The objective of the Point-in-Polygon (PiP) Test is to determine the spatial position relations between points and polygon. Given a point and a polygon, the PiP problem involves determining whether the point lies inside, outside or on the boundary of the polygon.

This software accomplishes the processing of input data from files and users by implementing Minimum Bounding Rectangle (MBR), ray casting algorithm (RCA), classification of special cases, visualisation and interaction functions and outputs PIP results according to user requirements. The software can handle the relationship between points and complex polygons very well, but not complicated polygons with spatial topological relationships, such as those with Boolean operations. The report, however, analyses this situation and suggests possible solutions with examples.

The following critical methods develop the function of the software. MBR constructs the minimum bounding rectangle by finding the maximum and minimum values of the coordinates of the polygon endpoints. RCA from PiP class calculates the total count of intersection points between the ray and each segment and determines the positions between them by using the vector cross product. Special cases that may occur at a vertex of the polygon are classified by the relationship between the test point and the boundaries constructed based on the coordinates before and after that vertex of interest. As mentioned earlier, calculations in this software are based on objects with several reuse classes, so the software has been written in line with object-oriented programming applying PEP8 style within 4 days.

Project Description

You can easily use the software with a basic Python 3.8 environment and Matplotlib, and then you are shown how to run it following the meanings of files and variables.

Filename	**Explanation**
main_from_file.py	Main function file acquires the input points from the file.
main_from_user.py	Main function file to obtain input points from user input.
class_collection.py	Customized classes and functions.
plotter.py	Functions and classes imported from Matplotlib.
input.csv	Input points test file in the format 'ID, X, Y'.
polygon.csv	Input polygon test file in which ordered points form polygon vertices.
output.csv	Results output file corresponding to the test file 'input.csv', in the format ‘ID, Category’.

Run it as follows :

1. Determine the input polygon.

   The default input polygon is 'polygon.csv'. If you want to change the input polygon please place the replacement file in the same directory and change the path variable in the main function to the new file name you have.

```
➢ polygon = Csv('polygon.csv')
```

2. Decide how to input the point data to be tested and execute the different files depending on what you have.
   a. Execute 'main_from_file.py' :

      If you have a CSV file made up of point coordinates (ID, X, Y), put it in the same directory as the test file and change the path variable in 'main_from_file.py' to the name of your file.

```
➢ point = Csv('input.csv')
```

   b. Execute ‘main_from_user.py’ :

      If you have a keyboard connected and want to enter the point coordinates one by one.

```
➢ Please input the points in 'x,y' format
➢ ENTER and continue to input the next point, ending with consecutive
  ENTERs
➢ The point_1 is: 0,0
➢ The point_2 is: 3.5,2
➢ The point_3 is: -1,5.6
```

3. Obtain your PiP test results
   - If you execute the 'main_from_file.py':

     The script will automatically save the output as 'output.csv' in the same directory.

   - If you execute the 'main_from_user.py':

     Select whether to save the output as a CSV file to the same directory when prompted. If you do, continue to enter the file name 'file name.csv’ and enter to view the results. If you do not save the results, enter to continue.

```
➢ Would you like to output the results as a csv? [y/n] y
➢ Please enter a file name as 'name.csv': 'output_from_user'

  or

➢ Would you like to output the results as a csv? [y/n] n
➢ Results will not be stored.

  or

➢ Would you like to output the results as a csv? [y/n] I don't know
➢ Invalid input and no result file saved.
```

Software

Task 1. The MBR Algorithm

MBR script is available in the MBR class of 'class_collection.py'.

MBR.get_polygon():

- Read 'polygon.csv' via the read function in the custom class Csv ().
- Objectify the individual points read.
- Get the polygon vertex coordinate values via the get_x () and get_y () functions in the custom class Point().
- Store the vertex x and y coordinates into the empty list ppx= [ ] and ppy= [ ] respectively via a for loop
- Get the coordinates of the four vertices of the MBR by using the min () and max() functions.
- Return a list of four vertices of MBR endpoint1, endpoint2, endpoint3 and endpoint4.

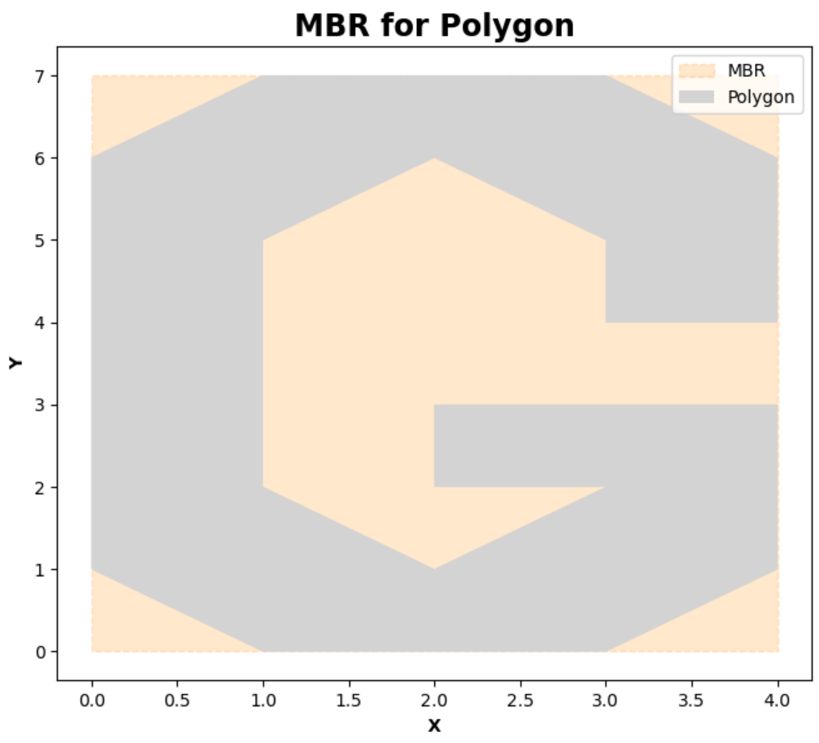
MBR.show()

- Print the four vertices of MBR.
- Example output :

```
➢ *********************From MBR************************
➢ The bottom left point is: (0.0, 0.0)
➢ The upper left point is: (0.0, 7.0)
➢ The upper right point is: (4.0, 7.0)
➢ The bottom right point is: (4.0, 0.0)
➢ ******************************************************
```

MBR.add_mbr( ):

- Visualizing MBR.
- Example output :



Task 2. The RCA Algorithm

RCA script is available in the PiP class of 'class_collection.py'.

PiP.rca ():

- Defining input variables
- Kind='outside' for points outside the MBR via set_kind in the Point ().
- Exclude points where 'Kind' has already been marked.
- Compare each input point with the segment of each polygon side:
    - Set the higher point of the segment to B and the lower one to A according to the y value.
    - Calculate the result of the vector cross product (res1= $\overrightarrow{AB}$ x $\overrightarrow{AP}$) for point P within the range of the y-values of the segment [1].
    - Determine the point-line position relationship based on res1.

      res1 = 0: P is co-linear with the segment AB

      res1 > 0: P is to the left of segment AB

      res1 < 0: P is to the right of segment AB
    - Set 'count' according to the position of the point line (modified by set_count() in Point( )).

      If P is to the left of segment AB, count+=1.

      If P is to the right of segment AB, count+=0.

If P is co-linear with segment AB and the x-value of P is within the x-value domain of segment AB, then P is on the boundary, and set kind= ‘boundary’.

- At this stage for P if the kind is still None and the count is zero, set it to kind='outside'.

Task 3. The Categorisation of Special Cases

RCA script is available in the PiP class of 'class_collection.py'.

- Exclude points where 'Kind' has already been marked.
- Find the point i from vertices of the polygon whose y-value is equal to that of the P and to the right of P, which may occur in special cases.
- Define one point before point i as C, the one after as D and the two after as E.
- Calculate the product of the difference between the vertical coordinates of C, D and P as res2, res2 = $(C_y-P_y) *(D_y-P_y)$
- Determine the position of C, D and the horizontal ray P according to res2.

  res2 < 0: C, D on either side of the ray P

  res2 > 0: C, and D are on the same side of the ray P

  res2 = 0: C, D and P are on the common line
- Special cases to be handled according to C, D and the position of the horizontal ray P:
  - If C, D on either side of the ray P, count-=1.
  - If C, and D are on the same side of the ray P, count+=0.
  - If C, D and P are on the common line
    - Calculate the product of the difference between the vertical coordinates of C, E and P as res2, res2 = $(C_y-P_y) *(E_y-P_y)$
    - Determine the position of C, E and the horizontal ray P according to res3.

      res3 > 0: C, E on either side of the ray P

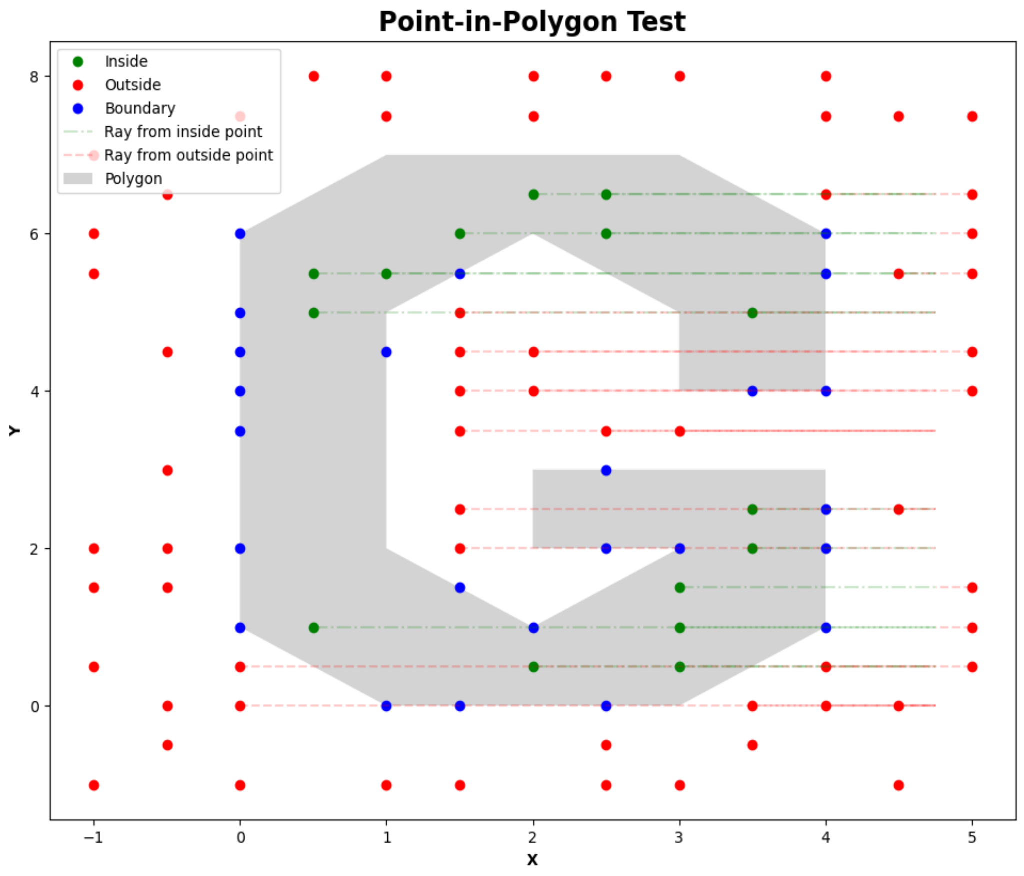
      res3 < 0: C and E are on the same side of the ray P

      res3 = 0 : C, E and P are on the common line again
    - Special cases to be handled according to C, E and the position of the horizontal ray P:
      - If C, E on either side of the ray P, count+=0
      - If C, E on the same side of the ray P, count-=1
      - If C, E and P are on the common line again, count+=0
- After completing the RCA and special case development, determine the position based on the parity of count.

  If the count is odd, the kind of point is inside; if the count is even, the kind of point is outside

PiP.show():

- Visualising the PiP result
- Example of output



Task 4. Object-Oriented Programming

The development of this software follows the principles of Object-Oriented Programming (OOP) and applies the PEP8 style.

The main program 'main_from_file.py' shows that the software passes input and output as parameters between the various classes and functions. The functions constructed are stored in 'class_collection.py’

- Constructed classes in 'class_collection.py’:

```
class Geometry:
class Point(Geometry):
class Line(Geometry):
class Polygon(Geometry):
class Csv:
class MBR(Point):
class PiP(MBR):
```

Task 5. On the Use of Git and GitHub

The development of this software is frequently synchronised and managed using git and GitHub.

Here are some scenes：

- Obtained the content of the assignment via GitHub.
- Create branch "assignmen_1" for editing.
- Commit and push local changes to GitHub regularly 22 times in total.
- Merge the branch "assignment_1" into the default branch "master" when development is complete.
- View and import the log via 'git log>log.text', shown at the end of the report.

Task 6. Plotting

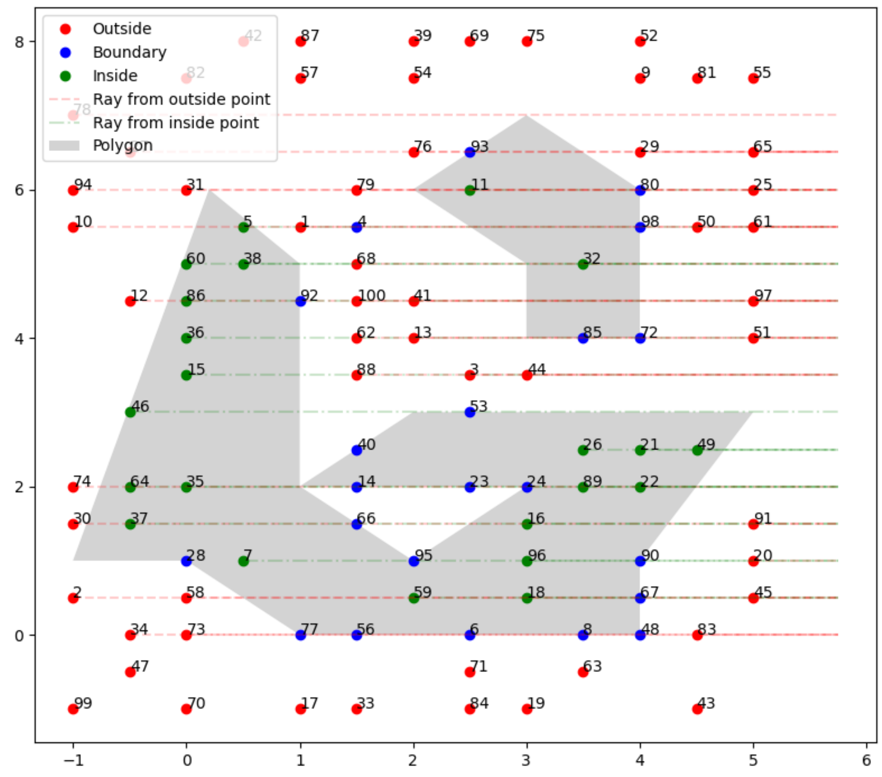
There are four main ways to use the Plotter class, and here are examples of each.

1. Use the initial method directly after instantiation.

- Code:

```
for p in self.point_lst:
    plt.add_point(p.get_x(), p.get_y(), p.get_kind())
plt = Plotter()
plt.add_polygon(ppx, ppy)
plt.show()
```

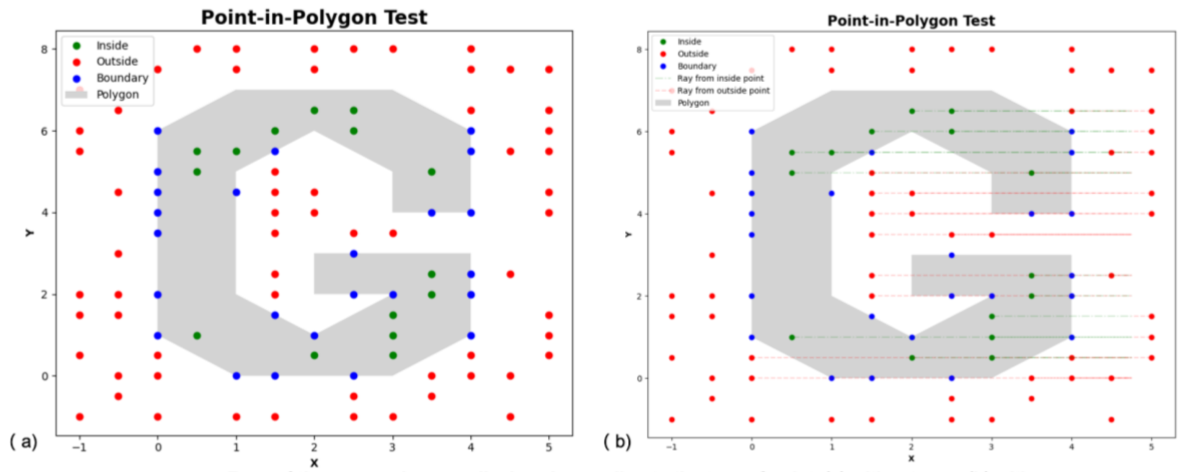
2. Use with changes to the original feature.
    - Add display variable 'id_' to help locate points during debugging.
    - Add title and XY labels, change display font, legend position, transparency and other display effects.
    - Example output：



3. Create a new function

   Adding rays of the same colour and adjusting the legend according to the point type.

   - Example output:



Rays of the same colour are displayed according to the type of point. (a) without rays, (b) with rays

- Code:

➢ `class Plotter():`

```
➢    def add_ray(self, xs, ys, kind):
➢        # plt.plot(xs, ys, label='Ray')
➢        if kind == 'outside':
➢            plt.plot(xs, ys, color='r', label='Ray from outside point', linestyle='--', alpha=0.2)
➢        elif kind == 'boundary':
➢            plt.plot(xs, ys, color='b', label='Ray from points in boundary', linestyle='-', alpha=0.2)
➢        elif kind == 'inside':
➢            plt.plot(xs, ys, color='g', label='Ray from inside point', linestyle='-.', alpha=0.2)
➢        else:
➢            plt.plot(xs, ys, color='k', label='Ray from unclassified point', linestyle='--', alpha=0.2)
➢
```

4. Import Plotter into 'class_collection.py' to join in building new functions.

   Create a function in the MBR using Plotter.plot().

```
➢ from plotter import Plotter
➢ class MBR(Point):
➢    def add_mbr(self):
➢        plt = Plotter()
➢        ppx, ppy, px, py = [], [], [], []
➢        mbr = MBR(self.polygon_lst)
➢        mbr_plst = mbr.get_polygon()
➢        for p in mbr_plst:
➢            px.append(p.get_x())
➢            py.append(p.get_y())
➢        for pp in self.polygon_lst:
➢            ppx.append(pp.get_x())
➢            ppy.append(pp.get_y())
➢        plt.add_mbr(px, py)
➢        plt.add_polygon(ppx, ppy)
```

Task 7. Error Handling

There are 3 main approaches to error handling, the first is (A): to handle possible errors according to the algorithm logic, and the other is (B): to adjust the algorithm path based on user input and program runtime feedback, such as "try" statements and different feedback based on user input. In addition, this part includes an analysis of error cases in the logic of the algorithm and the applicable conditions as (C). This section, therefore, focuses on 3 types of error handling.

A. In Task 3's special case handling, it is necessary to traverse the previous point and the next one and two points of point i. The common practice, in this case, is to treat the end point of polygon_lst separately. The comparison from i+2 onwards avoids the out-of-range error that would occur if the recursion i+2 and i+3 was looped to the end in polygon_lst.

```
➔ i = -2
➔ while i < len(self.polygon_lst) - 2:
➔    if   p.get_y()   ==   self.polygon_lst[i].get_y()   and   p.get_x()   <
self.polygon_lst[i].get_x():
➔        C = self.polygon_lst[i - 1]
➔        D = self.polygon_lst[i + 1]
➔
➔        res2 = (C.get_y() - p.get_y()) * (D.get_y() - p.get_y())
```

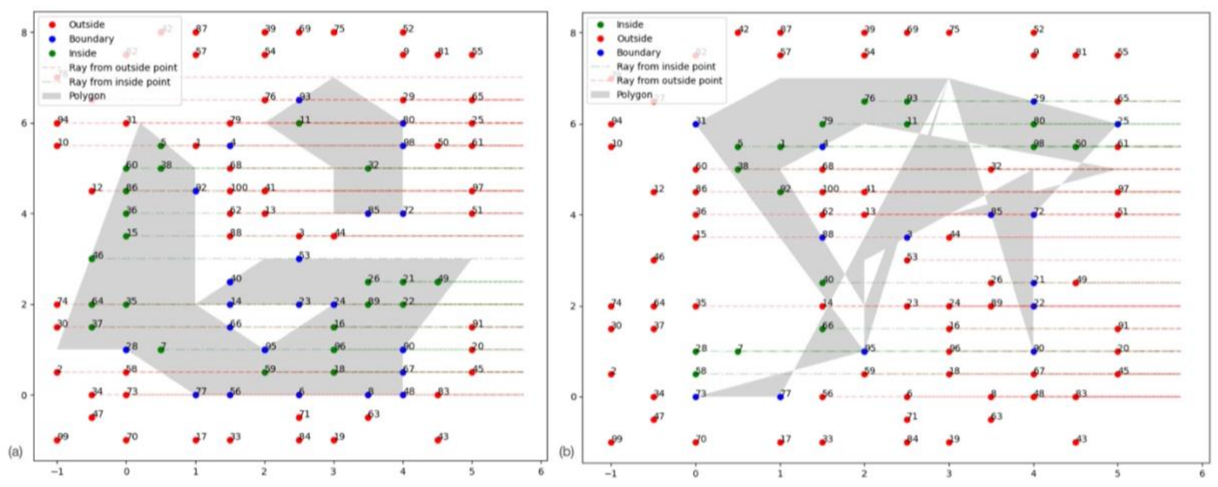
A. When modifying the count and kind attributes based on different results, the logic branch for count+=0 could have been removed for the sake of code simplicity, but in order to improve readability and easier debugging, the possible scenarios were categorised to avoid unknown logic black holes.

```
➔ if res3 > 0:   # C, E on either side of the ray P
➔    p.set_count(p.get_count())
➔ elif res3 < 0:   # C, E on the same side of the ray P
➔    p.set_count(p.get_count() - 1)
➔ else:   # C, E and P are on the common line again
➔    p.set_count(p.get_count())
```

B. Adjust the program logic based on user input, provide feedback on error conditions and avoid black boxes

```
➔ Would you like to output the results as a csv? [y/n]y
➔ Please enter a file name as 'name.csv': 'output_from_user'
➔
➔ Would you like to output the results as a csv? [y/n]n
➔ Results will not be stored.
➔
➔ Would you like to output the results as a csv? [y/n]i don't know
➔ Invalid input and no result file saved.
```

C. The algorithm can handle the relationship between points and complex polygons very well, but not complicated polygons with spatial topological relationships, such as those with Boolean operations.



PiP test results on different types of polygons. (a) Complex polygons (b) Complex polygons with Boolean operations

For example, (a) in the figure above shows a complex polygon after deformation, where points 74, 64, 35, 14, 23, and 89 show that the algorithm can handle the special cases of complex polygons with ease. However, for the points in figure (b) like 60, 86, 28, 58, and 7, the points on the 'boundary' and the points outside the MBR boundary are not judged to be problematic. There are problems with the judgement of 'inside' and 'outside' for certain areas. Analysis reveals that this is due to the Boolean operation on multiple polygons in the area.

The root of this result is the determination of the final total count by the algorithm based on the parity of the ‘kind’. As an example, we have two polygons α and β that overlap spatially and a point P. When P is inside α and inside β, the count of P is odd when judging the relationship between α and P. When judging the relationship between β and P, the count of P is even. At the end of the loop when the parity of count is finally determined, since the sum of the odd and even numbers is odd, the kind of point P will be set to 'inside', as in the case of points 7, 28, 58 in figure (b). When the results of such Boolean operations are ongoing, the points are also judged with different errors.

There are several possible solutions to such situations. The first is to add a point with the attribute ‘boolean_count’ and adjust the point ‘count’ according to this attribute. The second, change the structure of the RCA loop. The third, order the polygon vertices in two dimensions before running the RCA algorithm, which is equivalent to adding polygons that have completed the Boolean operation.

Task 8. Additional Features

Several features have been added based on display and interaction effects.

- Display of the same colour ray within the MBR according to the different categories of points
- Adding elements for the display of the results (XY-axis labels, titles, etc.)
- Ask the user to save the input data and the results and give different feedback depending on the user input values
- Add visualisation methods to the MBR class

- Objectify CSV class, set read, output and save methods
- Define parity determination functions

Git Log

```
➔ commit 84e660cf1c32f39c394f6588e3018bc8c57062f7
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Mon Nov 21 14:05:50 2022 +0000
➔
➔     update
➔
➔ commit b5f27363f77109c00d61994a451ee7a48f0a3ad4
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Mon Nov 21 01:06:14 2022 +0000
➔
➔     Update
➔
➔ commit 1ec456c77819665bb0370826ac75682127b3fc2c
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sun Nov 20 22:38:55 2022 +0000
➔
➔     oop update
➔
➔ commit 2c33499bc4692472829aecb3a2341bfcaa550fc4
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sun Nov 20 22:31:43 2022 +0000
➔
➔     OOP update
➔
➔ commit 76aec4202388a938954db13a9bc1aecd7ad3e7a5
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sun Nov 20 19:37:11 2022 +0000
➔
➔     update
➔
➔ commit 64295b5c2e7f6eac1ea916aa3ef3dc8514d8e78a
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sat Nov 19 23:42:14 2022 +0000
➔
➔     branches merged and comments updated
➔
```

```
➔ commit 35d68780a5c418f248043c7b16ca46f145b08262
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sat Nov 19 22:20:23 2022 +0000
➔
➔    update
➔
➔ commit 8b2b547357d90596ebd8fd4aecfbd04cd531ab9d
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Sat Nov 19 16:55:39 2022 +0000
➔
➔    comments
➔
➔ commit 70828b1000efeb094a01c35bc143bbfb40c9d334
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 18:15:14 2022 +0000
➔
➔    Done with user's input
➔
➔ commit 69cb76efc0784c86f5b723bffd0e8a623b94f9a6
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 17:41:34 2022 +0000
➔
➔    Update main_from_user.py
➔
➔ commit d15aedbd5327bcb42ba3cda260cb6d1b0911985d
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 17:39:20 2022 +0000
➔
➔    Update
➔
➔ commit f66125cb6e94269f3e0d3f65faff23bd645cc1b4
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 16:46:14 2022 +0000
➔
➔    Done with input file
➔
➔ commit f2ab9dbb88f1c673d1275030a01284f9efd68ed8
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 16:05:50 2022 +0000
➔
➔    class_collection
```

```
➔
➔ commit 46fd30d1f649c3a5c5e58666d9c679ef4a68c7a4
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 16:00:15 2022 +0000
➔
➔     result
➔
➔ commit 85b8075374730486353a843853109caf20bbbcd6
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 14:05:43 2022 +0000
➔
➔     Update
➔
➔ commit 51e3c7eddfbe434eaa26582ee1c472f24337f010
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Fri Nov 18 02:46:11 2022 +0000
➔
➔     update
➔
➔ commit d9f37682ebcbe52442535126bf26631820360538
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Thu Nov 17 18:00:20 2022 +0000
➔
➔     Update main_from_file.py
➔
➔ commit ccbc589c648556009db4a31369f488524223cae0
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Thu Nov 17 17:51:29 2022 +0000
➔
➔     Update main_from_file.py
➔
➔ commit 943a1a9047442cf0a9b2b4cbd9013d2198885665
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Thu Nov 17 11:10:06 2022 +0000
➔
➔     update
➔
➔ commit 4809f6fd179745bbc39f727191628639c1b803a1
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Thu Nov 17 01:20:13 2022 +0000
➔
```

```
➔    frame of PIP
➔
➔ commit 4982c7af58757287a007e40e9002f420b0e4c71e
➔ Author: Elio-Zonghe <elioma@163.com>
➔ Date:   Thu Nov 17 00:23:23 2022 +0000
➔
➔    CSV
➔
➔ commit 2dea54a62d72d4913c82216a57076f5b4ff8c523
➔ Author:             github-classroom[bot]             <66690702+github-
  classroom[bot]@users.noreply.github.com>
➔ Date:   Fri Nov 4 11:05:10 2022 +0000
➔
➔    Initial commit
```

Reference

[1] Tomorrow_Maple. (2018) Vector cross-product to determine the position of a point. Retrieved from https://blog.csdn.net/qq_27606639/article/details/80911006.