ST1507 DATA STRUCTURES AND ALGORITHMS (AI)

2024/2025 SEMESTER 2 ASSIGNMENT ONE (CA1) REPORT

Name: Phylicia Ng 2308908

Class: DAAA/2A/22

~ Detecting Hidden Objects in Encoded Text ~

Table of Contents

Introduction 3
User Guidelines 3
  Details of options in application menu 3
Object-Oriented Programming Approach 6
  Details of options in application menu 6
  Encapsulation 7
  Function/Operator Overloading 7
Data Structures and Algorithms 8
  Data Structures Used 8
  Performance (Big O Analysis) 8
Challenges and Key Takeaways 9
  Challenges 9
  Key Takeaways 9
  Further Improvements 9
Appendix 10
  Source code 10
  References 23

Introduction

The application "Detecting Hidden Objects in Encoded Text" is designed to help users encode, decode, analyse, and manipulate textual representations of hidden objects. It uses algorithms to detect patterns, classify shapes, and manipulate encoded text files. The application's primary scope includes detecting hidden shapes, analysing frequency distributions, and editing encoded images to provide users with interactive data visualization and manipulation tools.

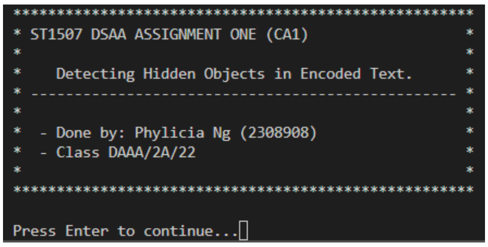
The application helps users perform four core operations:

1. Hide Objects: Encode text images to obscure hidden objects using a randomly reshuffled alphabet.
2. Analyse Frequency: Generate character frequency graphs to understand the distribution of characters in encoded images.
3. Detect Hidden Objects: Uncover hidden objects within encoded text images.
4. Identify and Classify Shapes: Detect the type, location, and count of shapes like triangles, rectangles, and circles in the text images.

User Guidelines

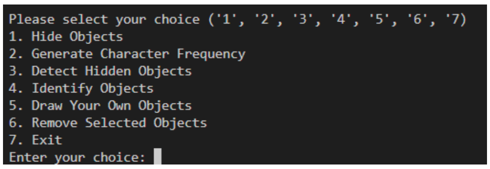
To run the "Detecting Hidden Objects in Encoded Text" application, follow the steps below:

1. Open Anaconda Prompt (or any Python-compatible terminal)
2. Enter the following command:
   `python main.py`
   You will be met with this screen



```
******************************************************
* ST1507 DSAA ASSIGNMENT ONE (CA1)                   *
*                                                    *
*    Detecting Hidden Objects in Encoded Text.       *
* -------------------------------------------------- *
*                                                    *
*  - Done by: Phylicia Ng (2308908)                  *
*  - Class DAAA/2A/22                                *
*                                                    *
******************************************************

Press Enter to continue...
```

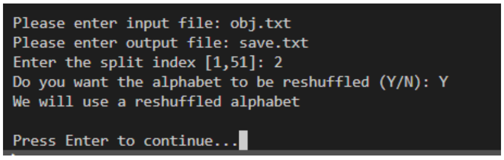
3. Press Enter to continue
4. The next screen will be as shown below



```
Please select your choice ('1', '2', '3', '4', '5', '6', '7)
1. Hide Objects
2. Generate Character Frequency
3. Detect Hidden Objects
4. Identify Objects
5. Draw Your Own Objects
6. Remove Selected Objects
7. Exit
Enter your choice:
```

Details of options in application menu

1. **Option 1: Hide Objects**
   This option allows users to encode a text image, hiding its objects by converting the '*' (object pixels) and '.' (background pixels) into a randomised alphabet.
   Example of the menu

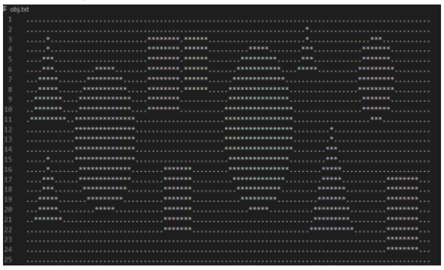


```
Please enter input file: obj.txt
Please enter output file: save.txt
Enter the split index [1,51]: 2
Do you want the alphabet to be reshuffled (Y/N): Y
We will use a reshuffled alphabet

Press Enter to continue...
```

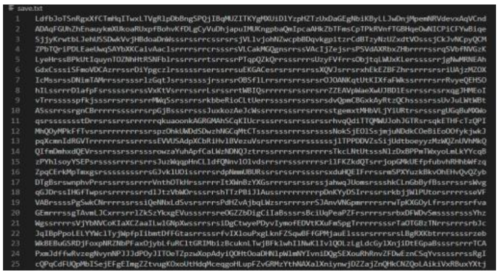
- **Input file**: Type in the name of an existing text image.
- **Output file**: Type in the name of the .txt file you want to save the encoded image to. If the file does not exist, the application will automatically create one for you.
- **Split index**: Choose a number between 1 and 51. This number will determine the number of alphabets used to represent the image.
- **Reshuffle alphabet**:
  - Enter Y for yes, and a random alphabet will be chosen to represent the image.

- Enter N for no, and the alphabets chosen to represent the image will be in the order of abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ.

Example usage:

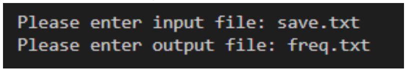


obj.txt



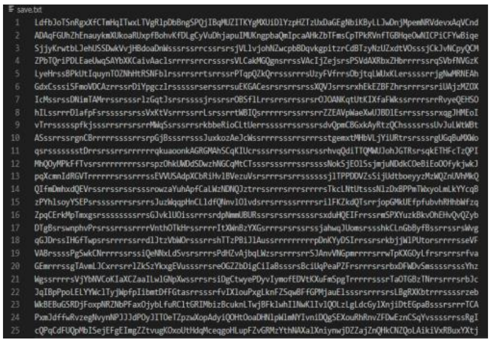
save.txt

2. **Option 2: Generate Character Frequency**
   The application will read the input file, calculate the frequency of each character, and generate a graph representing these frequencies. The graph will then be written to the specified output file.
   Example usage:

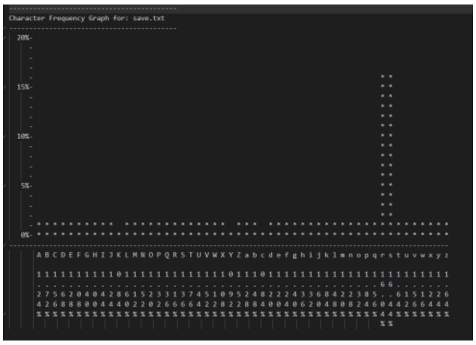


```
Please enter input file: save.txt
Please enter output file: freq.txt
```

- **Input file**: Type in the name of an existing text file to analyse
- **Output file**: Type in the name of the .txt file you want to save the character frequency graph to. If the file does not exist, the application will automatically create one for you



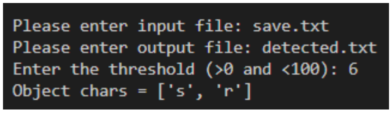
save.txt



```
Character Frequency Graph for: save.txt
```

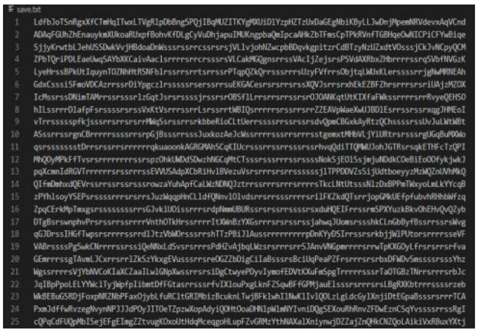
freq.txt

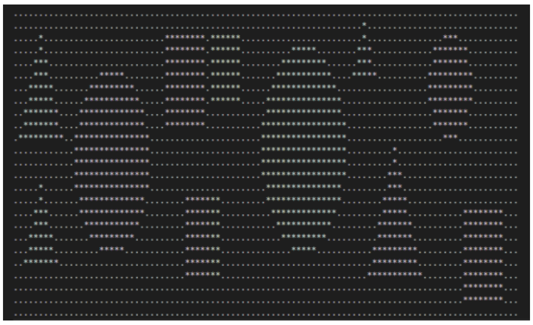
3. **Option 3: Detect Hidden Objects**
   The application will read the input file, analyse the text to detect hidden objects based on a user-defined threshold, and generate an output representing these detections. The output will then be written to the specified output file.
   Example usage:



```
Please enter input file: save.txt
Please enter output file: detected.txt
Enter the threshold (>0 and <100): 6
Object chars = ['s', 'r']
```

- **Input file**: Type in the name of an existing text file to analyse
- **Output file**: Type in the name of the .txt file you want to save the detected to. If the file does not exist, the application will automatically create one for you
- **Threshold**: Characters with a frequency percentage greater than or equal to the threshold (6% in this case) are classified as "object characters". All other characters are classified as "background characters".
- **Object characters (chars)**: This shows the detected characters that make up the image.

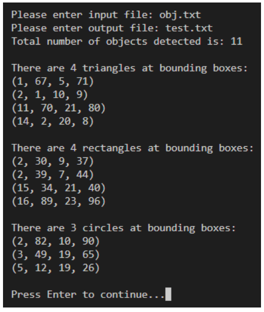

save.txt


detected.txt

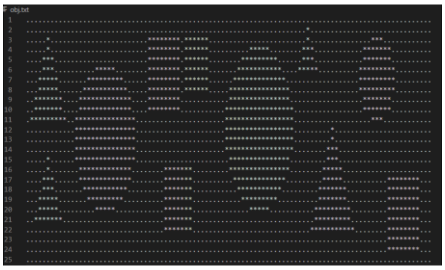
4. **Option 4: Identify Objects**
   The application will identify objects and list them in the order of Triangles, Rectangles, and lastly Circles.

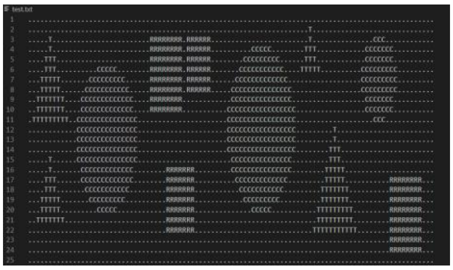
- **Input file**: Type in the name of an existing text image.
- **Output file**: Type in the name of the .txt file you want to save the encoded image to. If the file does not exist, the application will automatically create one for you.
- **Total number of objects detected** – what is detected by the application
- **Bounding box coordinates** – this specifies the area that encloses each detected shape. The coordinates are given as (start_row, start_column, end_row, end_column), indicating the top-left and bottom-right corners of the bounding box.

Example usage:



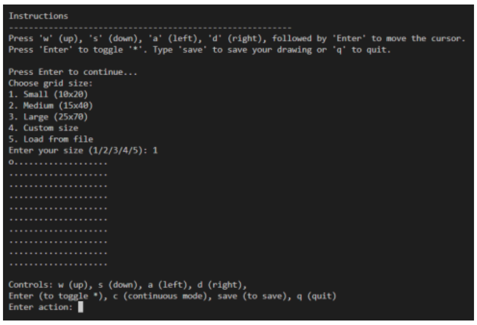
```
Please enter input file: obj.txt
Please enter output file: test.txt
Total number of objects detected is: 11

There are 4 triangles at bounding boxes:
(1, 67, 5, 71)
(2, 1, 10, 9)
(11, 70, 21, 80)
(14, 2, 20, 8)

There are 4 rectangles at bounding boxes:
(2, 30, 9, 37)
(2, 39, 7, 44)
(15, 34, 21, 40)
(16, 89, 23, 96)

There are 3 circles at bounding boxes:
(2, 82, 10, 90)
(3, 49, 19, 65)
(5, 12, 19, 26)

Press Enter to continue...
```


obj.txt


test.txt

5. **Option 5: Draw Your Own Objects**
   This part of the application allows users to create and manipulate a grid of pixels. Users can toggle pixels on and off, move a cursor around the grid, represented by 'o', and save their drawings to a file

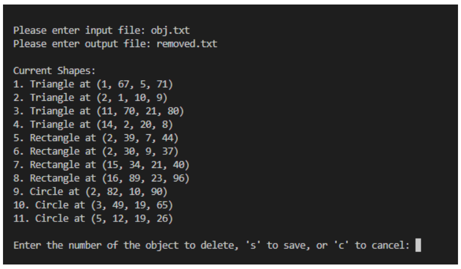
- Basic instructions will be shown
  Press 'w' (up), 's' (down), 'a' (left), 'd' (right), followed by 'Enter' to move the cursor.
  Press 'Enter' to toggle '*'. Type 'save' to save your drawing or 'q' to quit.
- Upon starting, you will be prompted to choose the grid size for a new file. If you choose a custom size, you will be prompted to enter the number of rows and columns. If you choose to load from a file, the application will prompt you to provide the file name.
- **Press 'ENTER' after every input**



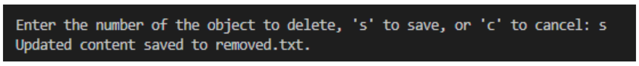
```
Instructions
------------------------------------------------------------
Press 'w' (up), 's' (down), 'a' (left), 'd' (right), followed by 'Enter' to move the cursor.
Press 'Enter' to toggle '*'. Type 'save' to save your drawing or 'q' to quit.

Press Enter to continue...
Choose grid size:
1. Small (10x20)
2. Medium (15x40)
3. Large (25x70)
4. Custom size
5. Load from file
Enter your size (1/2/3/4/5): 1
o...................
....................
....................
....................
....................
....................
....................
....................
....................
....................

Controls: w (up), s (down), a (left), d (right),
Enter (to toggle *), c (continuous mode), save (to save), q (quit)
Enter action: 
```

6. **Option 6: Remove Selected Objects**
   This option identifies shapes in a text-based input file and remove specific shapes based on user input. The application supports saving the updated text file after modifications.

- You will be prompted to enter the number of the object to delete, 's' to save, or 'c' to cancel.
- Enter the unique ID of the shape you want to remove.
- **Save:** Type ‘s’ and press Enter to save the current state of the text file. The application will save the updated text to the specified output file.
- **Cancel**: Type ‘c’ and press Enter to cancel the deletion operation and exit the loop.



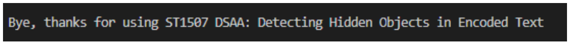
```
Please enter input file: obj.txt
Please enter output file: removed.txt

Current Shapes:
1. Triangle at (1, 67, 5, 71)
2. Triangle at (2, 1, 10, 9)
3. Triangle at (11, 70, 21, 80)
4. Triangle at (14, 2, 20, 8)
5. Rectangle at (2, 39, 7, 44)
6. Rectangle at (2, 30, 9, 37)
7. Rectangle at (15, 34, 21, 40)
8. Rectangle at (16, 89, 23, 96)
9. Circle at (2, 82, 10, 90)
10. Circle at (3, 49, 19, 65)
11. Circle at (5, 12, 19, 26)

Enter the number of the object to delete, 's' to save, or 'c' to cancel: 
```



```
Enter the number of the object to delete, 's' to save, or 'c' to cancel: s
Updated content saved to removed.txt.
```

7. **Option 7: Exit**

Exits the application



```
Bye, thanks for using ST1507 DSAA: Detecting Hidden Objects in Encoded Text
```

Object-Oriented Programming Approach

Details of options in application menu

1. **FileProcessor (Base Class)** This Provides the foundational structure for file processing. writeToFile: A placeholder method designed to be overridden in derived classes. **Encapsulation**: Attributes like inputFile and outputFile are accessible only within the class and derived classes, ensuring proper data protection.	2. **FileChecker (Subclass of FileProcessor)** Validates input and output files, ensuring they are accessible and properly formatted. Checks if input files exist and are not empty. Ensures output files are writable. Handles errors like missing files or incorrect extensions.
3. **GUI** Manages the user interface for the application through a CLI. Displays menus and prompts the user for choices. Handles invalid input dynamically. Connects user actions to the appropriate modules.	4. **encodeObjects (Subclass of FileProcessor)** Encodes text images by replacing * and . with randomized alphabet characters. Randomizes the alphabet and allows user-defined split indices. Encapsulation of encoding logic for reusability
5. **generateGraph (Subclass of FileProcessor)** Analyzes character frequency in a text file and generates a frequency distribution graph. Computes percentages for each character. Generates text-based bar charts for visualization.	6. **detection (Subclass of FileProcessor)** Detects hidden objects in encoded text files by converting characters back into ‘*’ and ‘.’ Identifies objects using a frequency threshold provided by the user. Outputs a simplified text representation of the objects.
7. **Objects (Subclass of FileProcessor)** Identifies and classifies shapes (triangles, rectangles, circles) in text files. Uses bounding boxes and shape classification algorithms. Replaces ‘*’ with labeled characters (T, R, C).	8. **removeObjects (Subclass of FileProcessor)** Removes specific shapes from a text file. Lists all identified shapes with unique IDs. Allows users to delete selected shapes by replacing their characters with background pixels (.)
9. **Grid** Facilitates the creation and manipulation of grid-based drawings. Provides cursor-based drawing functionality. Allows users to toggle pixels, save drawings, or load grids from files.	

Encapsulation

FileProcessor and FileChecker: • **Private Attributes**: The inputFile, outputFile, and text attributes are protected, ensuring only the relevant methods within these classes or their subclasses can access or modify them. • **Encapsulated Logic**: The checkFiles method in FileChecker handles all file validation internally (checking extensions, handling missing files, ensuring write permissions). This logic is hidden from other modules, and they interact with validated files seamlessly.	**encodeObjects Class**: • **Encoding Logic**: The text encoding process is encapsulated within the encodeImage method, which handles: ○ Splitting the alphabet. ○ Randomly assigning labels for object and background pixels. ○ Replacing characters with encoded labels. • Users only interact with the writeToFile method, while the intricate encoding operations remain hidden.
detection Class: • **Detection Process**: The method objectPixels encapsulates the entire process of analyzing character frequencies, determining object/background thresholds, and reconstructing a simplified text image	**Objects Class**: • **Shape Classification Logic**: The identification of shapes (triangles, rectangles, circles) is encapsulated in helper methods like isRectangle, isTriangle, and isCircle, ensuring that the main class method (writeToFile) focuses only on the high-level task of labeling shapes.

Function/Operator Overloading

The writeToFile method in the FileProcessor base class is overridden in its subclasses (e.g., encodeObjects, generateGraph) to implement specific file-writing functionalities. This demonstrates function overloading, where the method performs different tasks depending on the class.

Polymorphism

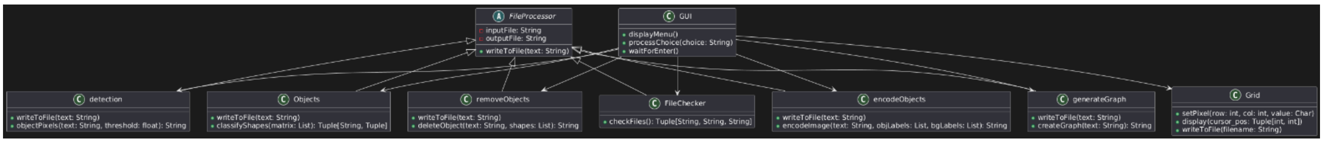
Polymorphism is effectively used to ensure a uniform interface (writeToFile) while enabling different behaviors in subclasses

1. **Grid Class Integration**:
    - The writeToFile method in Grid saves the current drawing state to a text file, showcasing polymorphism in action. The same method name (writeToFile) is used here, but it performs tasks specific to grid-based drawings.
2. **identifyObjects and removeObjects**:
    - Both classes inherit writeToFile from FileProcessor but override it to serve different purposes:
        - identifyObjects.writeToFile labels shapes with T, R, or C.
        - removeObjects.writeToFile removes selected shapes while maintaining the overall grid structure.
3. **Subclasses of FileProcessor**:
   Each subclass (e.g., generateGraph, detection, encodeObjects) uses the same interface (writeToFile) to perform distinct tasks.

Inheritance

Inheritance ensures code reuse and extensibility throughout the application. All subclasses (e.g., encodeObjects, generateGraph, Objects) inherit the file-handling methods (writeToFile) and attributes (inputFile, outputFile) from the FileProcessor base class. This reduces redundancy in initializing file attributes and processing files. The FileProcessor base class makes it easy to extend functionality. For example, you could add a new class, such as highlightObjects, which inherits from FileProcessor and uses the writeToFile interface to highlight specific areas in the text image.

Class Diagram


A FileProcessor
inputFile: String
outputFile: String
writeToFile(text: String)
C GUI
displayMenu()
processChoice(choice: String)
waitForEnter()
C detection
writeToFile(text: String)
objectPixels(text: String, threshold: float): String
C Objects
writeToFile(text: String)
classifyShapes(matrix: List): Tuple[String, Tuple]
C removeObjects
writeToFile(text: String)
deleteObject(text: String, shapes: List): String
C FileChecker
checkFiles(): Tuple[String, String, String]
C encodeObjects
writeToFile(text: String)
encodeImage(text: String, objLabels: List, bgLabels: List): String
C generateGraph
writeToFile(text: String)
createGraph(text: String): String
C Grid
setPixel(row: int, col: int, value: Char)
display(cursor_pos: Tuple[int, int])
writeToFile(filename: String)

Data Structures and Algorithms

Data Structures Used

Data Structure	Purpose	Built-in/Custom
List	Store lines of text, shapes, bounding boxes, classified shapes	Built-in
Dictionary	Store character frequencies, shapes by type	Built-in
Set	Track visited coordinates	Built-in
Tuple	Store cursor positions, bounding boxes	Built-in
Shape Matrix	Store shapes and their bounding boxes	Custom (using built-in lists and tuples)
Classified Shapes	Store classified shapes with their types and bounding boxes	Custom (using built-in lists and tuples)
Labeled Shapes	Store labeled shapes with unique IDs	Custom (using built-in lists and tuples)
Grid	Represent a 2D grid of pixels	Custom (using built-in lists)

Performance (Big O Analysis)

Operation	Complexity	Explanation
Encoding text	O(n)	Linear traversal of text for character replacement.
Character frequency	O(n)	Counts occurrences of each character.
Shape detection (Breadth-First-Search)	O(m×n), where m is the number of rows and n is the number of columns.	Traverses each cell in the grid.
Shape classification	O(s), where s is the size of the bounding box	Processes bounding boxes based on dimensions.

Why Use Breadth-First-Search (BFS) for Grid Traversal?

In the application, objects in the grid (represented by *) are spatially connected. BFS is used to explore all connected * pixels starting from a single *, ensuring that the entire object is grouped together. This approach ensures that all the adjacent pixels belonging to the same object are identified and objects are accurately separated even if they are close to each other.

Challenges and Key Takeaways

Challenges

I struggled to implement accurate shape classification algorithms, especially for option 4. Ensuring that shapes are accurately classified as triangles, rectangles, or circles, especially because the triangles and circles had unique designs was difficult. As such I had to make the classification more lax to accommodate it. Designing a file handling mechanism that accounts for missing or empty files, or wrong file names. Providing an intuitive interface for user interactions with minimal learning curve, especially for my own options as there are no guidelines

Key Takeaways

Through this project, I have gained a better understanding of encapsulation, inheritance and polymorphism. I learned how to design and implement custom data structures, which is the Grid class. I am far more confident in implementing file handling mechanisms to ensure data is correctly read from input files and saved to output files. I also improved skills in testing and debugging by identifying and fixing issues throughout the development process.

Further Improvements

Given time, I would improve the algorithms for detecting and classifying shapes to handle more complex and irregular shapes, such as different polygons or ellipses. I would also have loved to implement real-time drawing and editing capabilities, allowing users to see changes immediately as they interact with the grid, instead of having to press ‘enter’ after every input. Implementing these further improvements would significantly enhance the functionality, performance, and user experience of the application.

Appendix

Source code

Refer to the following files for the code

main.py

```
from gui import GUI
if __name__ == "__main__":
    app = GUI()
    app.displayTitle()
    while True:
        app.waitForEnter()
        while True:
            app.displayMenu()
            choice = app.getChoice()
            if choice == "Invalid choice. Please enter a number between 1 and 7.":
                print(choice)
                continue
            app.processChoice(choice)
```

gui.py

```
# GUI class that displays the title and menu of the application.
from objectEncoding import encodeObjects
from characterFrequency import generateGraph
from detectObjects import detection
from identifyObj import Objects
from fileOperations import FileChecker
from drawOnGrid import draw
from removeObj import removeObjects

class GUI:
    def displayTitle(self):
        # Display the title and author information
        text = [
            "ST1507 DSAA ASSIGNMENT ONE (CA1)",
            "",
            "   Detecting Hidden Objects in Encoded Text.   ",
            "------------------------------------------------",
            "",
            " - Done by: Phylicia Ng (2308908)",
            " - Class DAAA/2A/22",
            ""
        ]

        max_length = max(len(line) for line in text)
        border = '*' * (max_length + 4)

        print(border)
        for line in text:
            print(f"* {line.ljust(max_length)} *")
        print(border)

    def displayMenu(self):
        # Display the menu options
        print("\nPlease select your choice ('1', '2', '3', '4', '5', '6', '7)")
        print("1. Hide Objects")
        print("2. Generate Character Frequency")
        print("3. Detect Hidden Objects")
        print("4. Identify Objects")
        print("5. Draw Your Own Objects")
        print("6. Remove Selected Objects")
        print("7. Exit")

    def getChoice(self):
        # Get the user's choice and validate it
        choice = (input("Enter your choice: "))

        valid_choices = ['1', '2', '3', '4', '5', '6', '7']
        if choice not in valid_choices:
            return "Invalid choice. Please enter a number between 1 and 7."
        print('\n')
        return choice

    def waitForEnter(self):
        # Wait for the user to press Enter to continue
        while True:
            pressEnter = input("\nPress Enter to continue...")

            if pressEnter == "":
                break
            else:
                print("Invalid input. Please press Enter to continue.")

    def processChoice(self, choice):
        if choice == '1':
            # Hide Objects
            file_checker = FileChecker()
            text, inputFile, outputFile = file_checker.checkFiles()
            encoder = encodeObjects(inputFile, outputFile)
            encoder.writeToFile(text)

            self.waitForEnter()

        elif choice == '2':
            # Generate Character Frequency
            file_checker = FileChecker()
            text, inputFile, outputFile = file_checker.checkFiles()
            characterGraph = generateGraph(inputFile=inputFile, outputFile=outputFile)
            characterGraph.writeToFile(text)

            self.waitForEnter()

        elif choice == '3':
            # Detect Hidden Objects
            file_checker = FileChecker()
            text, inputFile, outputFile = file_checker.checkFiles()
            detect = detection(inputFile, outputFile)
            detect.writeToFile(text)

            self.waitForEnter()

        elif choice == '4':
            # Identify Objects
            file_checker = FileChecker()
            text, inputFile, outputFile = file_checker.checkFiles()
            identifier = Objects(inputFile, outputFile)
            identifier.writeToFile(text)

            self.waitForEnter()

        elif choice == '5':
            print("Instructions"
                  "\n---------------------------------------------------------"
                   "Press 'w' (up), 's' (down), 'a' (left), 'd' (right), followed by 'Enter' to move the cursor. "
                  "Press 'Enter' to toggle '*'. Type 'save' to save your drawing or 'q' to quit.")
            self.waitForEnter()

            # Call the draw function
            draw()


        elif choice == '6':
            # Remove selected objects
            file_checker = FileChecker()
            text, inputFile, outputFile = file_checker.checkFiles()
            identifier = removeObjects(inputFile, outputFile)
            identifier.writeToFile(text)

        elif choice == '7':
            print("Bye, thanks for using ST1507 DSAA: Detecting Hidden Objects in Encoded Text")
            exit()
        else:
            print("Invalid choice. Please enter a number between 1 and 7.")
```

fileOperations.py

```python
# Base class for file processing
class FileProcessor:
    def __init__(self, inputFile=None, outputFile=None):
        self.inputFile = inputFile
        self.outputFile = outputFile

    def writeToFile(self, text):
        # placeholder, if not implemented
        raise NotImplementedError("Edit as according to option requirements")

# Class to check input and output files
class FileChecker:
    def __init__(self, inputFile=None, outputFile=None):
        self.inputFile = inputFile
        self.outputFile = outputFile
        self.text = None

    def checkFiles(self):
        # Check both input and output files
        self.checkInputFile()
        self.checkOutputFile()
        return self.text, self.inputFile, self.outputFile

    def checkInputFile(self):
        # Loop until a valid input file is provided
        while True:
            if self.inputFile is None:
                self.inputFile = input("Please enter input file: ")

            # Add .txt extension if not included
            if not self.inputFile.endswith('.txt'):
                self.inputFile += '.txt'

            # Check if the input file is valid
            try:
                with open(self.inputFile, 'r') as infile:
                    self.text = infile.read()

                    if not self.text:  # Check if file is empty
                        print("Error: The input file is empty.")
                        self.inputFile = None
                        continue

            except FileNotFoundError:
                print("Error: The input file was not found.")
                self.inputFile = None
                continue

            except IOError:
                print("Error: Could not read the input file.")
                self.inputFile = None
                continue
            break

    def checkOutputFile(self):
        # Loop until a valid output file is provided
        while True:
            if self.outputFile is None:
                self.outputFile = input("Please enter output file: ")

            # Add .txt extension if not included
            if not self.outputFile.endswith('.txt'):
                self.outputFile += '.txt'

            # Check if the output file can be opened/written
            try:
                with open(self.outputFile, 'w') as outfile:
                    pass  # Just opening and closing to check write permission
            except FileNotFoundError:
                print("Error: The output file was not found.")
                continue

            except IOError:
                print("Error: Could not open the output file for writing.")
                continue
            break
```

objectEncoding.py

```
import string
import random
from fileOperations import FileProcessor

class FileProcessor:
    def __init__(self, inputFile=None, outputFile=None):
        self.inputFile = inputFile
        self.outputFile = outputFile

    def writeToFile(self, text):
        raise NotImplementedError("Subclasses should implement this method")

# Class to encode objects in text
class encodeObjects(FileProcessor):
    def __init__(self, inputFile=None, outputFile=None):
        super().__init__(inputFile, outputFile)
        self.splitIndex = None
        self.alphabet = list(string.ascii_letters)  # 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

    def writeToFile(self, text):
        # Prompt for and validate the split index
        while True:
            splitIndex = input("Enter the split index [1,51]: ")

            try:
                splitIndex = int(splitIndex)
                if splitIndex < 1 or splitIndex > 51:
                    print("Error: The split index must be a number between 1 and 51.")
                    continue
            except ValueError:
                print("Please enter a number between 1 and 51.")
                continue
            self.splitIndex = splitIndex
            break

        # Shuffle the alphabet if desired
        while True:
            shuffle_alphabet = input("Do you want the alphabet to be reshuffled (Y/N): ").upper()

            if shuffle_alphabet == 'Y':
                random.shuffle(self.alphabet)
                print('We will use a reshuffled alphabet')
                break
            elif shuffle_alphabet == 'N':
                print('We will use a non-reshuffled alphabet')
                break
            else:
                print('Invalid choice, please enter Y or N.')

        # Split the alphabet based on the split index
        object_labels = self.alphabet[:self.splitIndex]
        background_labels = self.alphabet[self.splitIndex:]

        # Encode the text image by calling the encodeImage function
        encoded_text = self.encodeImage(text, object_labels, background_labels)

        # Write the encoded content to the output file
        try:
            with open(self.outputFile, 'w') as outfile:
                outfile.write(encoded_text)
            # print(f"Encoded content written to {self.outputFile}")
        except IOError:
            print("Error: Could not write to the output file.")
    def encodeImage(self, text, object_labels, background_labels):
        # Encode the text based on the provided labels
        encoded_image = ""
        for char in text:
            if char == '*':
                encoded_image += random.choice(object_labels)
            elif char == '.':
                encoded_image += random.choice(background_labels)
            else:
                encoded_image += char
        return encoded_image
```

characterFrequency.py

```
import math
from fileOperations import FileProcessor

# Class to generate character frequency graphs
class generateGraph(FileProcessor):
    def __init__(self, inputFile=None, outputFile=None):
        super().__init__(inputFile, outputFile)

    def writeToFile(self, text):
        # Write graph by running createGraph function
        analysis = self.createGraph(text)
        # Write the encoded content to the output file
        try:
            with open(self.outputFile, 'w') as outfile:
                outfile.write(analysis)
            # print(f"Encoded content written to {self.outputFile}")
        except IOError:
            print("Error: Could not write to the output file.")

    def createGraph(self, text):
        frequency = {}
        for char in text:
            if char in frequency:
                frequency[char] += 1
            else:
                frequency[char] = 1

        # Remove \n character from the dictionary
        if '\n' in frequency:
            del frequency['\n']

        total_chars = sum(frequency.values())

        for char in frequency:
            frequency[char] = (frequency[char] / total_chars) * 100

        # Make the max percentage multiples of 5
        max_percentage = math.ceil(max(frequency.values()) / 5) * 5

        graph = "------------------------------------------\n"
        graph += f"Character Frequency Graph for: {self.inputFile}\n"
        graph += "------------------------------------------\n"

        current_percentage = max_percentage
        while current_percentage >= 0:
            if current_percentage % 5 == 0:
                graph += f" {int(current_percentage):3}%-"
            else:
                graph += "     -"

            # Add asterisks for the frequency bars
            for char in sorted(frequency.keys()):
                if frequency[char] >= current_percentage:
                    graph += " *"
                else:
                    graph += "  "
            graph += "\n"
            current_percentage -= 1

        # --------------
        graph += "------" + "-" * (len(frequency) * 2)

        # ABCDEFGHIJK..
        graph += "\n       " + " ".join(sorted(frequency.keys())) + "\n\n"

        # Add the % signs behind the numbers
        frequency_str = {char: f"{frequency[char]:.2f}%" for char in frequency}

        # Vertical numbers
        max_length = max(len(frequency_str[char]) for char in frequency_str)
        for i in range(max_length):
            graph += "       "
            for char in sorted(frequency.keys()):
                value_str = frequency_str[char]
                if i < len(value_str):
                    graph += value_str[i] + " "
                else:
                    graph += "  "
            graph += "\n"

        return graph
```

detectObjects.py

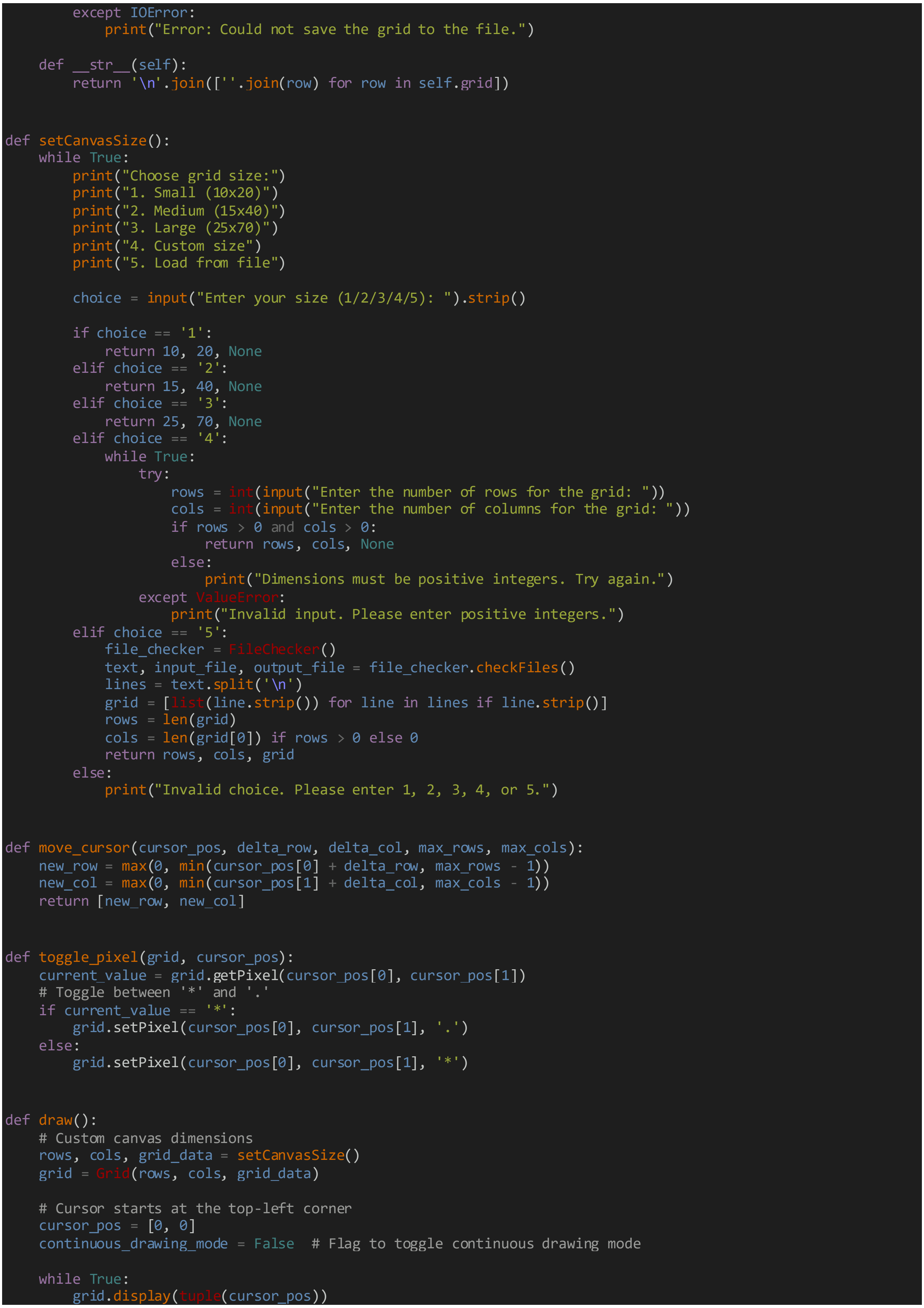
```
from objectEncoding import FileProcessor

# Class to detect hidden objects in text
class detection(FileProcessor):
    def __init__(self, inputFile=None, outputFile=None):
        super().__init__(inputFile, outputFile)

    def writeToFile(self, text):
        # Prompt for and validate the threshold
        while True:
            threshold = input('Enter the threshold (>0 and <100): ')

            try:
                threshold = float(threshold)
                if threshold <= 0 or threshold >= 100:
                    print("Error: The threshold must be a number greater than 0 and less than 100.")
                    continue
            except ValueError:
                print("Please enter a valid number greater than 0 and less than 100.")
                continue
            break

        # Perform analysis and generate output content
        analysis = self.objectPixels(text, threshold)

        # Write the output to the output file
        try:
            with open(self.outputFile, 'w') as outfile:
                outfile.write(analysis)
            # print(f"Processed content written to {self.outputFile}")
        except IOError:
            print("Error: Could not write to the output file.")

    def objectPixels(self, text, threshold):
        # Analyze the text to detect hidden objects
        frequency = {}

        for char in text:
            if char in frequency:
                frequency[char] += 1
            else:
                frequency[char] = 1

        # Remove the newline character from the dictionary
        if '\n' in frequency:
            del frequency['\n']

        total_chars = sum(frequency.values())
        # Calculate percentage frequency
        for char in frequency:
            frequency[char] = (frequency[char] / total_chars) * 100

        # Separate characters into object and background based on the threshold
        object_chars = [char for char, freq in frequency.items() if freq >= threshold]
        background_chars = [char for char in frequency if char not in object_chars]

        print(f'Object chars = {object_chars}')

        output = ''

        for char in text:
            if char in object_chars:
                output += '*'
            elif char in background_chars:
                output += '.'
            else:
                output += char

        return output
```

identifyObj.py

```
from fileOperations import FileProcessor
from shapeIdentifier import identifyObjects, createMatrix, classifyShapes

class Objects(FileProcessor):
    def __init__(self, inputFile=None, outputFile=None):
        super().__init__(inputFile, outputFile)

    def writeToFile(self, text):
        # Identify shapes and their bounding boxes
        objects = identifyObjects(text)

        # Split the input text into lines
        lines = text.split('\n')

        # Create a matrix for shapes
        shape_matrix = createMatrix(objects, lines)

        # Classify shapes based on the number of '*'
        classified_shapes = classifyShapes(shape_matrix)

        # Replace '*' in the text with shape labels ('T', 'R', 'C')
        labeled_text = self.labelObjects(text, classified_shapes)

        # Write the labeled content to the output file
        try:
            with open(self.outputFile, 'w') as outfile:
                outfile.write(labeled_text)
            # debugging
            # print(f"Labeled content written to {self.outputFile}")
        except IOError:
            print("Error: Could not write to the output file.")

        # Print the classified shapes
        self.printShapes(classified_shapes)

    def labelObjects(self, text, classified_shapes):
        # Replace '*' in the input text with labels ('T', 'R', 'C') for shapes
        lines = text.split('\n')
        for shape_type, bounding_box in classified_shapes:
            label = 'T' if shape_type == 'Triangle' else 'R' if shape_type == 'Rectangle' else 'C'
            min_row, min_col, max_row, max_col = bounding_box
            for r in range(min_row, max_row + 1):
                for c in range(min_col, max_col + 1):
                    if lines[r][c] == '*':
                        lines[r] = lines[r][:c] + label + lines[r][c + 1:]
        return '\n'.join(lines)

    def printShapes(self, classified_shapes):
        # Print the classified shapes and their bounding boxes
        # Group shapes by type
        shapes_by_type = {'Triangle': [], 'Rectangle': [], 'Circle': []}

        for shape_type, bounding_box in classified_shapes:
            if shape_type in shapes_by_type:
                shapes_by_type[shape_type].append(bounding_box)

        # Print total number of objects
        total_objects = sum(len(shapes) for shapes in shapes_by_type.values())
        print(f"Total number of objects detected is: {total_objects}")

        # Print each shape type and its bounding boxes, sorted by row and column
        for shape_type, shapes in shapes_by_type.items():
            if shapes:
                # Sort bounding boxes by smallest row first, then smallest column
                shapes.sort(key=lambda box: (box[0], box[1]))
                shape_plural = shape_type.lower() + "s"  # Make the type plural
                print(f"\nThere are {len(shapes)} {shape_plural} at bounding boxes:")
                for bounding_box in shapes:
                    print(f"{bounding_box}")

# # testing
# import os
# if __name__ == "__main__":
#     inputFile = "obj.txt"
#     outputFile = "test.txt"

#     if not os.path.isfile(inputFile):
#         print(f"Error: The file '{inputFile}' does not exist.")
#     else:
#         with open(inputFile, 'r') as infile:
#             text = infile.read()
#         identifier = Objects(inputFile, outputFile)
#         identifier.writeToFile(text)
```

drawOnGrid.py

```
from fileOperations import FileChecker

class Grid(FileChecker):
    def __init__(self, rows, cols, grid=None):
        self.rows = rows
        self.cols = cols
        if grid:
            self.grid = grid
        else:
            self.grid = [['.' for _ in range(cols)] for _ in range(rows)]

    def setPixel(self, row, col, value):
        if 0 <= row < self.rows and 0 <= col < self.cols:
            self.grid[row][col] = value
        else:
            raise IndexError("Pixel out of bounds")

    def getPixel(self, row, col):
        if 0 <= row < self.rows and 0 <= col < self.cols:
            return self.grid[row][col]
        else:
            raise IndexError("Pixel out of bounds")

    def display(self, cursor_pos=None):
        for r, row in enumerate(self.grid):
            for c, pixel in enumerate(row):
                if cursor_pos and (r, c) == cursor_pos:
                    print('o', end='')  # Show cursor as 'o'
                else:
                    print(pixel, end='')
            print()  # New line after each row

    def writeToFile(self, filename):
        try:
            with open(filename, 'w') as file:
                for row in self.grid:
                    file.write(''.join(row) + '\n')
            print(f"Drawing saved successfully to {filename}")
```



```
        except IOError:
            print("Error: Could not save the grid to the file.")

    def __str__(self):
        return '\n'.join([''.join(row) for row in self.grid])


def setCanvasSize():
    while True:
        print("Choose grid size:")
        print("1. Small (10x20)")
        print("2. Medium (15x40)")
        print("3. Large (25x70)")
        print("4. Custom size")
        print("5. Load from file")

        choice = input("Enter your size (1/2/3/4/5): ").strip()

        if choice == '1':
            return 10, 20, None
        elif choice == '2':
            return 15, 40, None
        elif choice == '3':
            return 25, 70, None
        elif choice == '4':
            while True:
                try:
                    rows = int(input("Enter the number of rows for the grid: "))
                    cols = int(input("Enter the number of columns for the grid: "))
                    if rows > 0 and cols > 0:
                        return rows, cols, None
                    else:
                        print("Dimensions must be positive integers. Try again.")
                except ValueError:
                    print("Invalid input. Please enter positive integers.")
        elif choice == '5':
            file_checker = FileChecker()
            text, input_file, output_file = file_checker.checkFiles()
            lines = text.split('\n')
            grid = [list(line.strip()) for line in lines if line.strip()]
            rows = len(grid)
            cols = len(grid[0]) if rows > 0 else 0
            return rows, cols, grid
        else:
            print("Invalid choice. Please enter 1, 2, 3, 4, or 5.")


def move_cursor(cursor_pos, delta_row, delta_col, max_rows, max_cols):
    new_row = max(0, min(cursor_pos[0] + delta_row, max_rows - 1))
    new_col = max(0, min(cursor_pos[1] + delta_col, max_cols - 1))
    return [new_row, new_col]


def toggle_pixel(grid, cursor_pos):
    current_value = grid.getPixel(cursor_pos[0], cursor_pos[1])
    # Toggle between '*' and '.'
    if current_value == '*':
        grid.setPixel(cursor_pos[0], cursor_pos[1], '.')
    else:
        grid.setPixel(cursor_pos[0], cursor_pos[1], '*')


def draw():
    # Custom canvas dimensions
    rows, cols, grid_data = setCanvasSize()
    grid = Grid(rows, cols, grid_data)

    # Cursor starts at the top-left corner
    cursor_pos = [0, 0]
    continuous_drawing_mode = False  # Flag to toggle continuous drawing mode

    while True:
        grid.display(tuple(cursor_pos))
        print("\nControls: w (up), s (down), a (left), d (right), \nEnter (to toggle *), c (continuous mode), save (to save), q (quit)")
        if continuous_drawing_mode:
            print("Continuous Drawing Mode: ON")

        # Get user input for action
        actions = input("Enter action: ").strip().lower()
        invalid_chars = set()

        if actions == '':
            toggle_pixel(grid, cursor_pos)

        if actions == 'save':  # Save the grid to a file
            filename = input("Enter the filename to save the grid: ").strip()
            if filename:
                grid.writeToFile(filename)
            else:
                print("Invalid filename. Try again.")
            continue

        for action in actions:
            if action == 'w':
                cursor_pos = move_cursor(cursor_pos, -1, 0, rows, cols)
                if continuous_drawing_mode:
                    grid.setPixel(cursor_pos[0], cursor_pos[1], '*')
            elif action == 's':
                cursor_pos = move_cursor(cursor_pos, 1, 0, rows, cols)
                if continuous_drawing_mode:
                    grid.setPixel(cursor_pos[0], cursor_pos[1], '*')
            elif action == 'a':
                cursor_pos = move_cursor(cursor_pos, 0, -1, rows, cols)
                if continuous_drawing_mode:
                    grid.setPixel(cursor_pos[0], cursor_pos[1], '*')
            elif action == 'd':
                cursor_pos = move_cursor(cursor_pos, 0, 1, rows, cols)
                if continuous_drawing_mode:
                    grid.setPixel(cursor_pos[0], cursor_pos[1], '*')
            elif action == 'c':  # Toggle continuous drawing mode
                if len(actions) == 1:
                    continuous_drawing_mode = not continuous_drawing_mode
                    if continuous_drawing_mode:
                        print("Continuous Drawing Mode activated. Move to draw!")
                    else:
                        print("Continuous Drawing Mode deactivated.")
                else:
                    invalid_chars.add(action)
            elif action == 'q':
                if len(actions) == 1:
                    print("Stopping... \nReturing to main menu.")
                    return
                else:
                    invalid_chars.add(action)
            else:
                invalid_chars.add(action)

        for char in invalid_chars:
            print(f"Invalid character in input: {char}")
```

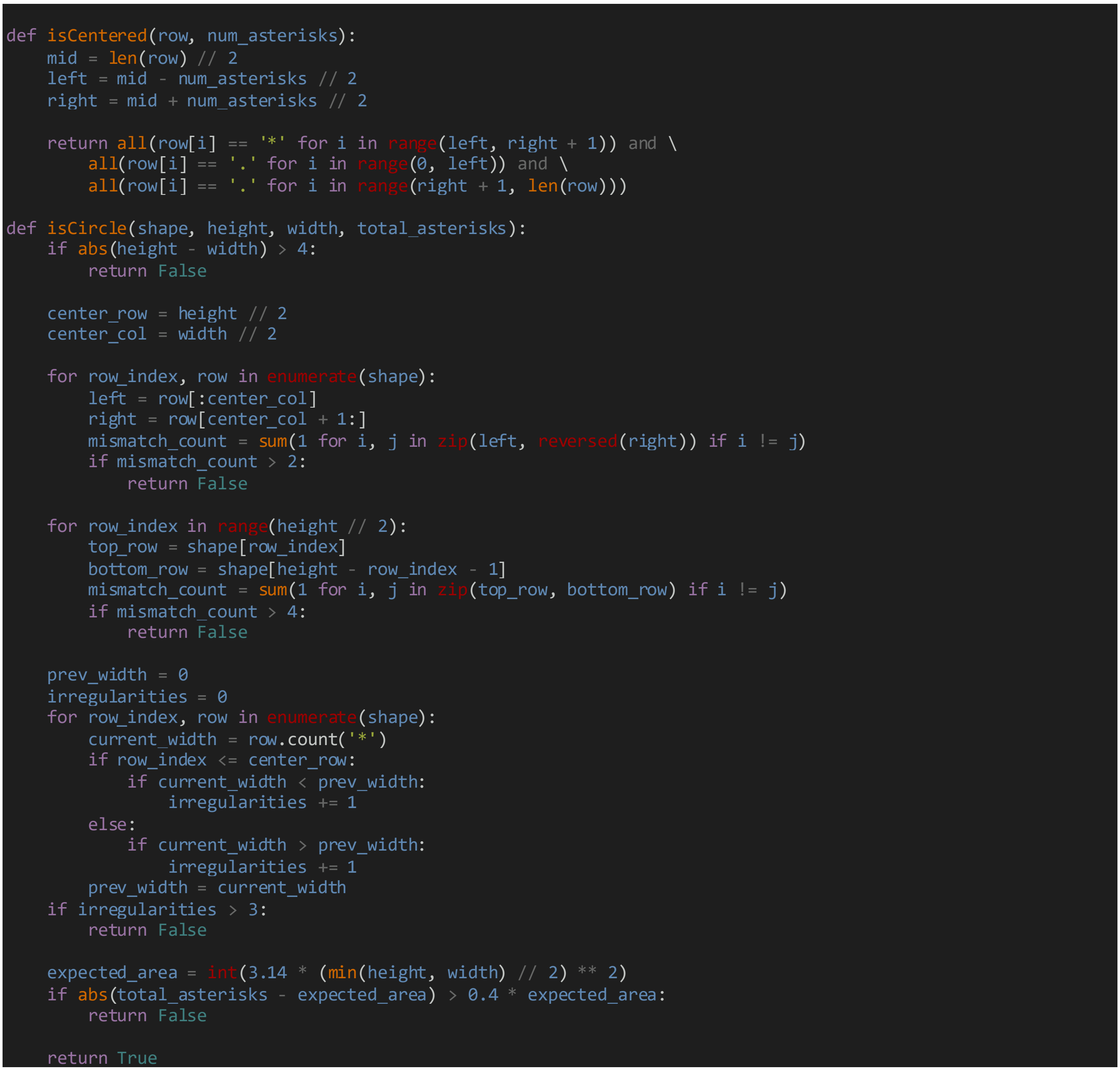
removeObj.py

```
from fileOperations import FileProcessor
from shapeIdentifier import identifyObjects, createMatrix, classifyShapes

class removeObjects(FileProcessor):
    def __init__(self, inputFile=None, outputFile=None):
        super().__init__(inputFile, outputFile)

    def writeToFile(self, text):
        # Identify shapes and their bounding boxes
        objects = identifyObjects(text)

        # Split the input text into lines
        lines = text.split('\n')
        # Create a matrix for shapes
        shape_matrix = createMatrix(objects, lines)

        # Classify shapes based on the number of '*'
        classified_shapes = classifyShapes(shape_matrix)

        # Add unique IDs to shapes
        labeled_shapes = self.labelShapes(classified_shapes)

        # Allow deletion of specific objects
        labeled_text = text  # Start with the original text
        while True:
            labeled_text = self.deleteObject(labeled_text, labeled_shapes)
            if labeled_text is None:
                break

            # Save the current state if the user chooses to save
            try:
                with open(self.outputFile, 'w') as outfile:
                    outfile.write(labeled_text)
                print(f"Updated content saved to {self.outputFile}.")
            except IOError:
                print("Error: Could not write to the output file.")

    def labelShapes(self, classified_shapes):
        # Add unique IDs to each shape
        labeled_shapes = []
        for idx, (shape_type, bounding_box) in enumerate(classified_shapes, start=1):
            labeled_shapes.append((shape_type, bounding_box, idx))
        return labeled_shapes

    def deleteObject(self, labeled_text, labeled_shapes):
        while True:
            # Show current shapes with labels
            print("\nCurrent Shapes:")
            for shape_type, bounding_box, label in labeled_shapes:
                print(f"{label}. {shape_type} at {bounding_box}")

            # Ask the user to delete an object, save, or cancel
            choice = input("\nEnter the number of the object to delete, 's' to save, or 'c' to cancel:
").strip()
            if choice.lower() == 'c':
                return None  # Exit the deletion loop
            elif choice.lower() == 's':
                return labeled_text  # Save the current state

            try:
                choice = int(choice)
                shape_to_delete = next((s for s in labeled_shapes if s[2] == choice), None)
                if not shape_to_delete:
                    print(f"Error: No shape found with number {choice}.")
                    continue

                # Update the text by replacing the shape's '*' with '.'
                labeled_text = self.removeShapeFromText(labeled_text, shape_to_delete)

                # Remove the shape from the labeled_shapes list
                labeled_shapes.remove(shape_to_delete)

                print(f"Shape {choice} removed, enter 's' to save updates.")
            except ValueError:
                print("Error: Please enter a valid number, 's' to save, or 'c' to cancel.")

    def removeShapeFromText(self, labeled_text, shape_to_delete):
        # Replace '*' with '.' for the specified shape
        _, bounding_box, _ = shape_to_delete
        min_row, min_col, max_row, max_col = bounding_box

        # Update the text
        lines = labeled_text.split('\n')
        for r in range(min_row, max_row + 1):
            for c in range(min_col, max_col + 1):
                if lines[r][c] == '*':  # Replace only if it's part of the shape
```

```
                    lines[r] = lines[r][:c] + '.' + lines[r][c + 1:]
        return '\n'.join(lines)

# # testing
# import os
# if __name__ == "__main__":
#     inputFile = "obj.txt"
#     outputFile = "test.txt"

#     if not os.path.isfile(inputFile):
#         print(f"Error: The file '{inputFile}' does not exist.")
#     else:
#         with open(inputFile, 'r') as infile:
#             text = infile.read()
#         remover = removeObjects(inputFile, outputFile)
#         remover.writeToFile(text)
```

shapeIdentifier.py

```
def identifyObjects(text):
    objects = []
    lines = text.split('\n')
    visited = set()

    for row, line in enumerate(lines):
        for col, char in enumerate(line):
            if char in ['*', 'C'] and (row, col) not in visited:
                bounding_box = getBoundingBox(lines, row, col)
                objects.append(bounding_box)

                # Mark all '*' in the bounding box as visited
                min_row, min_col, max_row, max_col = bounding_box
                for r in range(min_row, max_row + 1):
                    for c in range(min_col, max_col + 1):
                        if lines[r][c] in ['*', 'C']:
                            visited.add((r, c))
    return objects

def getBoundingBox(lines, start_row, start_col):
    min_row, max_row = start_row, start_row
    min_col, max_col = start_col, start_col

    stack = [(start_row, start_col)]
    visited = set(stack)

    while stack:
        row, col = stack.pop()
        min_row = min(min_row, row)
        max_row = max(max_row, row)
        min_col = min(min_col, col)
        max_col = max(max_col, col)

        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            r, c = row + dr, col + dc
            if 0 <= r < len(lines) and 0 <= c < len(lines[0]) and (r, c) not in visited and lines[r][c] ==
'*':
                stack.append((r, c))
                visited.add((r, c))

    return min_row, min_col, max_row, max_col

def createMatrix(objects, lines):
    shape_matrix = []
    for bounding_box in objects:
        min_row, min_col, max_row, max_col = bounding_box
        shape = []
        for r in range(min_row, max_row + 1):
            shape.append(list(lines[r][min_col:max_col + 1]))
        shape_matrix.append((bounding_box, shape))
    return shape_matrix

def classifyShapes(shape_matrix):
    classified_shapes = []
    for bounding_box, shape in shape_matrix:
        min_row, min_col, max_row, max_col = bounding_box
        total_asterisks = sum(row.count('*') for row in shape)

        height = max_row - min_row + 1
        width = max_col - min_col + 1

        if total_asterisks == height * width:
            if isRectangle(shape):
                shape_type = 'Rectangle'
                classified_shapes.append((shape_type, bounding_box))
                continue

        if isTriangle(shape, height, total_asterisks):
            shape_type = 'Triangle'
            classified_shapes.append((shape_type, bounding_box))
            continue

        if isCircle(shape, height, width, total_asterisks):
            shape_type = 'Circle'
            classified_shapes.append((shape_type, bounding_box))
            continue

        shape_type = 'Unknown'
        classified_shapes.append((shape_type, bounding_box))

    type_order = {'Triangle': 0, 'Rectangle': 1, 'Circle': 2}
    classified_shapes.sort(
        key=lambda obj: (type_order.get(obj[0], 3), obj[1][0], obj[1][2])
    )

    return classified_shapes

def isRectangle(shape):
    for row in shape:
        if row.count('*') != len(row):
            return False
    return True

def isTriangle(shape, height, total_asterisks):
    expected_asterisks = 0
    current_row_index = 0

    for level in range(1, (height + 1) // 2):
        num_asterisks_in_level = 2 * level - 1
        expected_asterisks += num_asterisks_in_level * 2

        if current_row_index + 1 >= len(shape):
            return False

        row1 = shape[current_row_index]
        row2 = shape[current_row_index + 1]

        if row1.count('*') != num_asterisks_in_level or row2.count('*') != num_asterisks_in_level:
            return False

        if not isCentered(row1, num_asterisks_in_level) or not isCentered(row2, num_asterisks_in_level):
            return False

        current_row_index += 2

    if current_row_index < len(shape):
        last_row = shape[current_row_index]
        last_row_asterisks = 2 * ((height + 1) // 2) - 1
        expected_asterisks += last_row_asterisks

        if last_row.count('*') != last_row_asterisks:
            return False

        if not isCentered(last_row, last_row_asterisks):
            return False

    return total_asterisks == expected_asterisks
```



```
def isCentered(row, num_asterisks):
    mid = len(row) // 2
    left = mid - num_asterisks // 2
    right = mid + num_asterisks // 2

    return all(row[i] == '*' for i in range(left, right + 1)) and \
        all(row[i] == '.' for i in range(0, left)) and \
        all(row[i] == '.' for i in range(right + 1, len(row)))

def isCircle(shape, height, width, total_asterisks):
    if abs(height - width) > 4:
        return False

    center_row = height // 2
    center_col = width // 2

    for row_index, row in enumerate(shape):
        left = row[:center_col]
        right = row[center_col + 1:]
        mismatch_count = sum(1 for i, j in zip(left, reversed(right)) if i != j)
        if mismatch_count > 2:
            return False

    for row_index in range(height // 2):
        top_row = shape[row_index]
        bottom_row = shape[height - row_index - 1]
        mismatch_count = sum(1 for i, j in zip(top_row, bottom_row) if i != j)
        if mismatch_count > 4:
            return False

    prev_width = 0
    irregularities = 0
    for row_index, row in enumerate(shape):
        current_width = row.count('*')
        if row_index <= center_row:
            if current_width < prev_width:
                irregularities += 1
        else:
            if current_width > prev_width:
                irregularities += 1
        prev_width = current_width
    if irregularities > 3:
        return False

    expected_area = int(3.14 * (min(height, width) // 2) ** 2)
    if abs(total_asterisks - expected_area) > 0.4 * expected_area:
        return False

    return True
```

References

1. Find how to open file
   https://dbader.org/blog/python-check-if-file-exists#:~:text=The%20most%20common%20way%20to,module%20in%20the%20standard%20library

```
try:
  f = open('myfile.txt')
  f.close()
except FileNotFoundError:
  print('File does not exist')
except IOError:
  print('File is not accessible')
print('File is accessible')
```

2. What does notimplementederror do → from copilot, 17/11/2024

   The NotImplementedError is a built-in exception in Python. It is used to indicate that a method or function is not yet implemented. In the context of your FileProcessor class, it serves as a placeholder to indicate that the writeToFile method should be overridden by subclasses. If a subclass does not override this method and it is called, the NotImplementedError will be raised, signaling that the method is not implemented.

   Here's a brief explanation of your code:

   - The FileProcessor class is a base class for file processing.
   - The __init__ method initializes the inputFile and outputFile attributes.
   - The writeToFile method is intended to be overridden by subclasses. If it is called without being overridden, it raises a NotImplementedError.

   This design uses inheritance (subclasses will inherit from FileProcessor) and encapsulation (attributes inputFile and outputFile are encapsulated within the class).

3. https://stackoverflow.com/questions/31400769/bounding-box-of-numpy-array
   finding out how do do a bounding box

4. Question: identify the common functions and seperate it out
   GitHub Copilot

   Used 2 references

   To separate the common functions into a separate file, we need to identify the functions that are used in both identifyObj.py and removeObj.py. The common functions are:

   1. identifyObjects
   2. getBoundingBox
   3. createMatrix
   4. classifyShapes
   5. isRectangle
   6. isTriangle
   7. isCentered
   8. isCircle