**Solution Design Document (SDD)**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to provide a detailed design for an interactive Python application that visualizes and interacts with geometric shapes using the turtle graphics library. This tool will allow users to define and draw rectangles, visualize points, and perform area calculations with interactive feedback.

**1.2 Scope**

This design document covers the system's architecture, class definitions, methods, and user interaction flow. The solution will include:

* Drawing geometric shapes (rectangles and points) on a canvas.
* User inputs for defining rectangles and points.
* Calculations for areas and distance between points.
* Interactive feedback based on user input.

**1.3 Definitions and Acronyms**

* **Turtle**: A Python library used for drawing shapes and handling graphics.
* **Point**: A class representing a point in 2D space.
* **Rectangle**: A class representing a rectangle defined by two corner points.
* **GuiRectangle**: A subclass of Rectangle that handles drawing the rectangle on the canvas.

**2. System Architecture**

**2.1 Overview**

The system will be a Python application using the turtle graphics library. It will have:

* A graphical user interface for drawing and interacting with shapes.
* Input handling to define shapes and perform calculations.
* Feedback mechanisms for user interactions.

**2.2 Components**

* **Turtle Graphics**: For drawing shapes and points on the canvas.
* **Point Class**: Manages point representation and operations.
* **Rectangle Class**: Manages rectangle representation and operations.
* **GuiRectangle Class**: Extends Rectangle to include drawing capabilities.

**3. Design**

**3.1 Classes and Methods**

**3.1.1 Point Class**

* **Attributes:**
  + x: X-coordinate of the point.
  + y: Y-coordinate of the point.
* **Methods:**
  + \_\_init\_\_(self, x, y): Initializes a point with the given coordinates.
  + falls\_in\_rectangle(self, rectangle): Determines if the point is inside the specified rectangle.
  + distance\_from\_point(self, x, y): Calculates the distance from the point to another point.
  + go\_to\_point(self): Moves the turtle to the point and draws a dot.

**3.1.2 Rectangle Class**

* **Attributes:**
  + point1: Bottom-left corner of the rectangle (instance of Point).
  + point2: Top-right corner of the rectangle (instance of Point).
* **Methods:**
  + \_\_init\_\_(self, point1, point2): Initializes the rectangle with two corner points.
  + area\_of\_rectangle(self): Calculates the area of the rectangle.

**3.1.3 GuiRectangle Class**

* **Methods:**
  + draw(self): Draws the rectangle on the turtle canvas.

**3.2 Data Flow**

1. **User Input:**
   * Users provide coordinates for the bottom-left and top-right corners of the rectangle.
   * Users provide coordinates for a point to check if it falls within the rectangle.
   * Users guess the area of the rectangle.
2. **Processing:**
   * The Rectangle class calculates the area based on the corner points.
   * The Point class determines if a point is within the rectangle and calculates distances.
   * The GuiRectangle class draws the rectangle on the canvas.
3. **Output:**
   * Display the drawn rectangle and point on the canvas.
   * Provide feedback on whether the point is inside the rectangle.
   * Show the difference between the guessed area and the actual area.

**3.3 User Interaction**

1. **Input Coordinates:**
   * Prompt the user to enter coordinates for defining the rectangle and the point.
2. **Drawing:**
   * Draw the rectangle and point on the canvas using turtle graphics.
3. **Feedback:**
   * Check if the point is inside the rectangle.
   * Display feedback on the user's area guess.

**4. Implementation**

**4.1 Tools and Technologies**

* **Python**: Programming language.
* **Turtle Library**: For graphical drawing.
* **IDLE or Visual Studio Code**: Development environments.

**4.2 Code Structure**

python

Copy code

import turtle

import random

# Define the Point class

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def falls\_in\_rectangle(self, rectangle):

if rectangle.point1.x < self.x < rectangle.point2.x and rectangle.point1.y < self.y < rectangle.point2.y:

return True

return False

def distance\_from\_point(self, x, y):

return ((x - self.x) \*\* 2 + (y - self.y) \*\* 2) \*\* 0.5

def go\_to\_point(self):

turtle.penup()

turtle.goto(self.x, self.y)

turtle.pendown()

turtle.pencolor("red")

turtle.dot(size=20)

# Define the Rectangle class

class Rectangle:

def \_\_init\_\_(self, point1, point2):

self.point1 = point1

self.point2 = point2

def area\_of\_rectangle(self):

return (self.point2.x - self.point1.x) \* (self.point2.y - self.point1.y)

# Define the GuiRectangle class

class GuiRectangle(Rectangle):

def draw(self):

turtle.penup()

turtle.goto(self.point1.x, self.point1.y)

turtle.pendown()

turtle.pencolor("black")

turtle.left(90)

turtle.forward(self.point2.x - self.point1.x)

turtle.left(90)

turtle.forward(self.point2.y - self.point1.y)

turtle.left(90)

turtle.forward(self.point2.x - self.point1.x)

turtle.left(90)

turtle.forward(self.point2.y - self.point1.y)

# Main code to use the classes

point1 = Point(3, 4)

point2 = Point(10, 20)

rectangle = GuiRectangle(Point(random.randint(0, 500), random.randint(0, 500)), Point(random.randint(500, 1000), random.randint(500, 1000)))

turtle.setup(1000, 1000)

rectangle.draw()

user\_point = Point(float(input("Enter X coordinate for the point: ")), float(input("Enter Y coordinate for the point: ")))

user\_point.go\_to\_point()

print(f"Point is inside rectangle: {user\_point.falls\_in\_rectangle(rectangle)}")

guess\_area = float(input("Guess the area of the rectangle: "))

print(f"Actual area: {rectangle.area\_of\_rectangle()}")

print(f"Difference: {abs(rectangle.area\_of\_rectangle() - guess\_area)}")

turtle.done()

**5. Testing and Validation**

**5.1 Test Cases**

1. **Rectangles and Points:**
   * Test with various rectangle coordinates and point locations to verify correctness.
   * Validate that points are correctly identified as inside or outside the rectangle.
2. **Area Calculation:**
   * Confirm that area calculations are accurate.
   * Verify user guesses against actual area.
3. **User Interaction:**
   * Ensure prompts and feedback are clear and accurate.

**5.2 Validation**

* **Functional Testing:** Verify that all features work as intended.
* **Usability Testing:** Check that the user interface is intuitive and user-friendly.

**6. Deployment**

**6.1 Deployment Strategy**

* **Environment Setup:** Ensure Python and turtle library are installed.
* **Script Execution:** Run the Python script in a development environment to verify functionality.
* **User Documentation:** Provide instructions on how to use the tool.

**6.2 Maintenance**

* Regularly update the tool as needed.
* Address any bugs or issues reported by users.