24-780B—ENGINEERING COMPUTATION

Assigned: Wed. Sept. 21, 2022

Problem Set 4

Now Due: Thurs. Sept. 29, 2022

Formalizing ShapeManager

Our old ShapeTester.cpp put a lot of code into the main() function. It is better in the long run, if we move some of those capabilities into a new class of object called *ShapeManager*. I did a lot of the copy/paste work so that the attached ps04_startup.zip file contains all the graphing capabilities of PS03, but making the main() function only responsible for initialization and to keep the loop going.

The most important feature that we want to add is the ability to graphically edit the shapes. First, we want to allow the user to go into and out of an "edit mode", so that they can do all the things they want to the current shapes. When the program is in "edit mode", we want the user to be aware of it, so we need to make it clear on the screen itself. Also, when in "edit mode", all the shapes except the current shape should display black/gray, leaving the current shape in its innate color (see below).

See the ShapeManager::showMenu() function for a complete list of requirements for the additional functionality of ShapeManager

Expanding Shape2D Class and Line2D Class

Our Shape2D class is marvelous, but it can do better. Below is a list of functions to add for PS04. Note that the header declarations of these functions are already included in the attached files, so you don't need copy/paste like you did in PS03 (saves a bit of time).

You may also remember that we want our shape to be non-intersecting (thus representing a real-world shape). We now owe it to ourselves to make sure this is true, so need to put additional conditions on addPoint(), removePoint(), movePoint(), and movePointDelta() to not allow any changes that would create a self-intersecting shape.

The member variables:

```
protected:
	std::vector<Point2D> thePoints; // stores the vertices that define the shape

	float colorHue = 0;  // stores overall color of shape

	// derived parameters
	Point2D lowerBound = { -INFINITY, -INFINITY };
	Point2D upperBound = { -INFINITY, -INFINITY };
	float perim = 0.f;
	float area = 0.f;  // leave blank for now
```

The new or altered functions:

```
public:
	// returns overall width and height of shape
	float getWidth() { if (lowerBound.x > -INFINITY) return (upperBound.x - lowerBound.x); }
	float getHeight() { if (lowerBound.x > -INFINITY) return (upperBound.y - lowerBound.y); }

	// returns the bounding box of the shape
	Point2D getLowerBound() { return lowerBound; }
	Point2D getUpperBound() { return upperBound; }

	// for PS04, simply returns the pre-calculated length of the perimeter of the shape.
	float perimeter() { return perim; }

	// get coordinates of a point. The first point has index 1.
	// Returns {-INFINITY, -INFINITY} if index is invalid.
	Point2D getPoint(int index);

	// moves a point to new coordinates. The first point has index 1.
	// Returns false if index is invalid.
	bool movePoint(Point2D newCoords, int index);
```

```
        // changes the coordinates of a point by the given delta. The first point has index 1.
        // Returns false if index is invalid.
        bool movePointDelta(Point2D deltaCoords, int index);

        // returns the index of the first vertex (first vertex has index 1) that is near the givenCoord.
        // Uses distance as bounding square instead of radius to reduce calculations.
        int getIndex(Point2D givenCoord, float dist = 0.1);

private:
        // determines if any of the edges of the shape intersects with any other edge
        // makes use of Line2D::getTrueIntersection()
        bool selfIntersects();

        // calculates and stores shape-level derived properties so they don't all have to be
        // generated on the fly. (adapt old perimeter function to include upper and lower bound)
        void recalcShape();
```

For Line2D.h, we need to add:

```
        // returns the intersection of the 2 line segments defined by start and end points given.
        // The true intersection differs from the mathematical intersection in that the 2 line segments
        // (not just their infinite linear extensions) must cross. If there is no true intersection
        // (i.e., the lines are parallel or the segments don't cross), the function
        // returns {-INFINITY, -INFINITY}.
        static Point2D getTrueIntersection(Point2D lineAstart, Point2D lineAend,
                Point2D lineBstart, Point2D lineBend);

        // returns the angle of the given line segment, from startPnt to endPnt.
        // The angle returned is in degrees (0-360), measured CCW from zero (the X-axis).
        // Makes use of atan2() function, so function is very short
        static float getAngle(Point2D startPnt, Point2D endPnt);
```

Deliverables

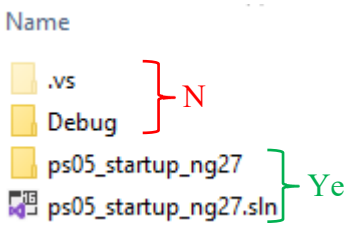
6 files (zipped together):

ShapeManager.h and ShapeManager.cpp

Shape2D.h and Shape2D.cpp

Line2D.h and Line2D.cpp

Alternatively, if you are using Visual Studio, it may be easier to submit your entire solution rather than a collection of files. To do this, create a *zip file* of the whole project (the .sln file and the associated folder), being careful NOT to include the hidden folder called ".vs". This folder is used only to manage the IDE and is typically huge (>100MB). Erasing or omitting it will just force Visual Studio to rebuild it when needed. The Debug folder should also be kept out of the zip file to avoid including executable files that some firewalls may disallow. *The name of the project should include your AndrewID*



Learning Objectives

Use of classes and objects in C++.

Creating and maintaining of sequential data structure (std::vector).

Introductory understanding of graphical data representation and display.

Reading data from a file and understanding how constructors can be effectively used to initialize a data model.

Implementing algorithms developed by others.