GeoDraw.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GeoDraw Version 1.0 (23.10) - Last revision: 01/10/2023

- A simple C/C++ 2D drawing library that outputs drawings as JavaScript

- Includes a simplified C function interface

NOTICE: Copyright (C) 2023 Conor McArdle, Dublin City University - All Rights Reserved.

Permission is granted to students registered for DCU module EE219: C/C++ for Engineers

to use this code for their personal use only. No permission is granted to share this file

by any means, including but not limited to, uploading it or submitting it to any private

or public server on the Web/Internet or any online service which publishes or otherwise

processes or stores uploaded content. This notice must stay attached to this file.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef GEODRAW\_H\_INCLUDED

#define GEODRAW\_H\_INCLUDED

#include <iostream>

#include <fstream>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <vector>

#include <math.h>

#include <bits/stdc++.h>

namespace GeoDraw {

using std::cout;

using std::cerr;

using std::endl;

using std::string;

using std::vector;

using std::ofstream;

extern string \_gd\_html\_pre; // html to appear before canvas in output web page

extern string \_gd\_html\_post; // html to appear after canvas in output web page

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Utility methods and defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef unsigned int u\_int;

const double PI = acos(-1);

string gd\_to\_string(int a);

string gd\_to\_string(double a, unsigned short decimalPlaces);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FillState, Font, TextAlign and Color class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

enum FillState { FILLED, UNFILLED };

enum Font { Arial, Courier, Times };

enum TextAlign { LEFT, CENTER, RIGHT };

class Color {

private:

double \_r, \_g, \_b;

string \_24BitColor;

public:

Color(double r, double g, double b) : \_r(r), \_g(g), \_b(b) {

\_24BitColor = "(";

\_24BitColor += gd\_to\_string((int)round(\_r \* 255)) + ",";

\_24BitColor += gd\_to\_string((int)round(\_g \* 255)) + ",";

\_24BitColor += gd\_to\_string((int)round(\_b \* 255)) + ")";

}

double r() const { return \_r; }

double g() const { return \_g; }

double b() const { return \_b; }

string to24BitColorString() const { return \_24BitColor; }

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pre-defined Colors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const Color BLACK(0,0,0);

const Color WHITE(1,1,1);

const Color LIGHT\_GRAY(0.8,0.8,0.8);

const Color GRAY(0.5,0.5,0.5);

const Color DARK\_GRAY(0.2,0.2,0.2);

const Color LIGHT\_RED(1,0.5,0.5);

const Color RED(1,0,0);

const Color DARK\_RED(0.5,0,0);

const Color LIGHT\_GREEN(0.5,1,0.5);

const Color GREEN(0,1,0);

const Color DARK\_GREEN(0,0.5,0);

const Color LIGHT\_BLUE(0.5,0.5,1);

const Color BLUE(0,0,1);

const Color DARK\_BLUE(0,0,0.5);

const Color YELLOW(1,1,0);

const Color TEAL(0,0.5,0.5);

const Color CYAN(0,1,1);

const Color MAGENTA(1,0,1);

const Color PINK(1,0.5,1);

const Color ORANGE(1,0.5,0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Forward Declaratoins

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class LineSeg;

class Circle;

class Polygon;

class Geometry;

class GeometryList;

class Canvas;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Coord class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Coord {

private:

double \_x, \_y;

public:

Coord(double x, double y) : \_x(x), \_y(y) {}

double x() const { return \_x; }

double y() const { return \_y; }

void moveBy(double x, double y) { \_x += x; \_y += y; }

void rotate(double angle) {

double tmp\_x = \_x \* cos(angle) - \_y \* sin(angle);

double tmp\_y = \_x \* sin(angle) + \_y \* cos(angle);

\_x = tmp\_x;

\_y = tmp\_y;

}

void rotate(double angle, double x, double y) {

moveBy(-x, -y);

rotate(angle);

moveBy(x, y);

}

void scale(double factor) { \_x \*= factor; \_y \*= factor; }

double distanceTo(const Coord & otherCoord) const {

double x\_diff = \_x - otherCoord.\_x;

double y\_diff = \_y - otherCoord.\_y;

return sqrt(x\_diff\*x\_diff + y\_diff\*y\_diff);

}

string toString() const {

return gd\_to\_string(\_x) + "," + gd\_to\_string(\_y);

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CoordList class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CoordList {

private:

vector<Coord> coords;

public:

void add(double x, double y) {

coords.push\_back(Coord(x,y));

}

void add(const Coord & c) {

coords.push\_back(c);

}

Coord & operator [] (u\_int i) {

if ((coords.size() == 0) || (i > coords.size()-1))

cerr << "GeoDraw::CoordList::[] - error : index out of bounds at index " << i << endl;

return coords[i];

}

u\_int size() {

return coords.size();

}

void clear() {

coords.clear();

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CanvasElement class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CanvasElement {

friend class Canvas;

private:

u\_int pauseAfter; // milliseconds to pause after drawing this canvas element

virtual CanvasElement \* clone() const = 0;

virtual string toJavaScript() const = 0;

protected:

Color color = BLACK;

u\_int penWidth = 1;

FillState fillState = UNFILLED;

CanvasElement() : pauseAfter(0), color(BLACK), penWidth(1), fillState(UNFILLED) {}

public:

virtual ~CanvasElement() {}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Abstract base class for all geometry objects

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Geometry : public CanvasElement {

friend class GeometryList;

public:

Geometry() {}

virtual ~Geometry() {};

/// Geometry object public interface methods

virtual Geometry & moveBy(double x, double y) = 0;

virtual Geometry & rotate(double angle, double x, double y) = 0;

virtual Geometry & rotate(double angle, const Coord & cRef) = 0;

virtual Geometry & scale(double factor) = 0;

private:

virtual Geometry \* clone() const = 0;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A list of Geomtry objects

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class GeometryList {

private:

vector<Geometry \*> geoms;

public:

void add(const Geometry & geom) {

geoms.push\_back(geom.clone());

}

Geometry & operator [] (u\_int i) {

if ((geoms.size() == 0) || (i > geoms.size()-1))

cerr << "GeoDraw::GeometryList::[] - error : index out of bounds at index " << i << endl;

return \*geoms[i];

}

u\_int size() {

return geoms.size();

}

void clear() {

geoms.clear();

}

virtual ~GeometryList() {

for (u\_int i=0; i<geoms.size(); i++)

if (geoms[i] == NULL)

cerr << "GeoDraw::~GeometryList() - error : null pointer in destructor " << i << endl;

else

delete geoms[i];

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Point class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Point : public Geometry {

private:

Coord coord;

string toJavaScript() const;

string getType() const { return "Point"; }

virtual Geometry \* clone() const { return new Point(\*this); }

public:

Point(double x, double y) : coord(x,y) {}

Point(const Coord & c) : coord(c) {}

double x() { return coord.x(); }

double y() { return coord.y(); }

double distanceTo(const Point & p) {

return coord.distanceTo(p.coord);

}

Coord coordinate() const {

return Coord(coord.x(), coord.y());

}

/// Geometry base class method implementations/overrides

Point & moveBy(double x, double y) { coord.moveBy(x,y); return \*this; }

Point & rotate(double angle, double x, double y) { coord.rotate(angle, x, y); return \*this; }

Point & rotate(double angle, const Coord & cRef) { coord.rotate(angle, cRef.x(), cRef.y()); return \*this; }

Point & scale(double factor) { coord.scale(factor); return \*this; }

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LineSeg class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class LineSeg : public Geometry {

private:

Coord c1, c2;

string toJavaScript() const;

string getType() const { return "LineSeg"; }

virtual Geometry \* clone() const { return new LineSeg(\*this); }

public:

LineSeg(const Coord & \_c1, const Coord & \_c2) : c1(\_c1), c2(\_c2) {}

LineSeg(double x1, double y1, double x2, double y2) : c1(x1,y1), c2(x2,y2) {}

CoordList asCoordList() const {

CoordList coords;

coords.add(c1);

coords.add(c2);

return coords;

}

/// Geometry base class method implementations/overrides

LineSeg & moveBy(double x, double y) {

c1.moveBy(x,y);

c2.moveBy(x,y);

return \*this;

}

LineSeg & rotate(double angle, double x, double y) {

c1.rotate(angle,x,y);

c2.rotate(angle,x,y);

return \*this;

}

LineSeg & rotate(double angle, const Coord & cRef) {

return this->rotate(angle, cRef.x(), cRef.y());

}

LineSeg & scale(double factor) {

c1.scale(factor);

c2.scale(factor);

return \*this;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Circle class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Circle : public Geometry {

private:

Coord cen; double radius;

string toJavaScript() const;

string getType() const { return "Circle"; }

virtual Geometry \* clone() const { return new Circle(\*this); }

public:

Circle(const Coord & c, double r) : cen(c), radius(r) {}

Circle(double c\_x, double c\_y, double r) : cen(c\_x,c\_y), radius(r) {}

/// Geometry base class method implementations/overrides

Circle & moveBy(double x, double y) {

cen.moveBy(x,y);

return \*this;

}

Circle & rotate(double angle, double x, double y) {

cen.rotate(angle,x,y);

return \*this;

}

Circle & rotate(double angle, const Coord & cRef) {

return this->rotate(angle, cRef.x(), cRef.y());

}

Circle & scale(double factor) {

radius \*= factor;

return \*this;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polygon class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Polygon : public Geometry {

private:

vector<Coord> vertices;

string toJavaScript() const;

string getType() const { return "Polygon"; }

virtual Geometry \* clone() const { return new Polygon(\*this); }

public:

Polygon() {}

Polygon(const vector<Coord> & coords) : vertices(coords) { }

Polygon(CoordList & coords) {

for (u\_int i=0; i<coords.size(); i++)

vertices.push\_back(coords[i]);

}

Polygon(const Coord coords[], u\_int size) {

for (u\_int i=0; i<size; i++)

vertices.push\_back(coords[i]);

}

void add(Coord c) { vertices.push\_back(c); }

void add(double x, double y) { vertices.push\_back(Coord(x,y)); }

u\_int size() { return vertices.size(); }

CoordList asCoordList() const {

CoordList coords;

for (u\_int i=0; i<vertices.size(); i++)

coords.add(vertices[i]);

return coords;

}

/// Geometry base class method implementations/overrides

Polygon & moveBy(double x, double y) {

for (u\_int i=0; i<vertices.size(); i++)

vertices[i].moveBy(x,y);

return \*this;

}

Polygon & rotate(double angle, double x, double y) {

for (u\_int i=0; i<vertices.size(); i++)

vertices[i].rotate(angle,x,y);

return \*this;

}

Polygon & rotate(double angle, const Coord & cRef) {

for (u\_int i=0; i<vertices.size(); i++)

vertices[i].rotate(angle, cRef.x(), cRef.y());

return \*this;

}

Polygon & scale(double factor) {

for (u\_int i=0; i<vertices.size(); i++)

vertices[i].scale(factor);

return \*this;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Text class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Text : public CanvasElement {

private:

string text;

Coord position;

Font font;

u\_int fontSize;

TextAlign alignment;

string toJavaScript() const;

virtual Text \* clone() const { return new Text(\*this); }

public:

Text(const string & txt, Coord pos) : text(txt), position(pos), font(Arial), fontSize(20), alignment(LEFT) { fillState = FILLED; }

Text(const string & txt, double x, double y) : text(txt), position(x,y), font(Arial), fontSize(20), alignment(LEFT) { fillState = FILLED; }

Text(const string & txt, Coord pos, Font fnt, u\_int font\_size, TextAlign algn = LEFT) : text(txt), position(pos), font(fnt), fontSize(font\_size), alignment(algn) { fillState = FILLED; }

Text(const string & txt, double x, double y, Font fnt, u\_int font\_size, TextAlign algn = LEFT) : text(txt), position(x,y), font(fnt), fontSize(font\_size), alignment(algn) { fillState = FILLED; }

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Canvas class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Canvas {

private:

vector<CanvasElement \*> elements; // container for canvas drawing elements

u\_int \_xDim; // canvas size in X direction

u\_int \_yDim; // canvas size in Y direction

Color bg\_color; // canvas background color

string outFileName; // output file name

Color default\_pen\_color; // default color of outline for non-filled shapes

Color default\_fill\_color; // default color of filled shapes

u\_int default\_pen\_width; // default width of pen, in pixels

public:

Canvas(u\_int x=600, u\_int y=600, string outFile = "MyDrawing.html", Color color = WHITE) :

\_xDim(x),

\_yDim(y),

bg\_color(color),

outFileName(outFile),

default\_pen\_color(BLACK),

default\_fill\_color(GRAY),

default\_pen\_width(2) { }

unsigned int xDim() { return \_xDim; }

unsigned int yDim() { return \_yDim; }

void setOutFileName(string fileName) { outFileName = fileName; }

Color penColor() { return default\_pen\_color; }

Color fillColor() { return default\_fill\_color; }

u\_int penWidth() { return default\_pen\_width; }

void setBackgroundColor(Color clr) { bg\_color = clr; }

void setPenColor(Color clr) { default\_pen\_color = clr; }

void setPenWidth(u\_int width) { default\_pen\_width = width > 0 ? default\_pen\_width : 1; }

void setFillColor(Color clr) { default\_fill\_color = clr; }

void add(const CanvasElement & elem) { elements.push\_back(elem.clone()); }

void add(const CanvasElement & elem, const Color & color, FillState is\_filled = UNFILLED) {

CanvasElement\* elemPtr = elem.clone();

elemPtr->color = color;

elemPtr->fillState = is\_filled;

elements.push\_back(elemPtr);

}

void add(const CanvasElement & elem, const Color & color, u\_int pen\_width) {

CanvasElement\* elemPtr = elem.clone();

elemPtr->color = color;

elemPtr->penWidth = pen\_width;

elements.push\_back(elemPtr);

}

void add(const CanvasElement & elem, u\_int pen\_width, const Color & color) {

CanvasElement\* elemPtr = elem.clone();

elemPtr->color = default\_pen\_color;

elemPtr->penWidth = pen\_width;

elements.push\_back(elemPtr);

}

void add(GeometryList & geomList) {

for (u\_int i=0; i<geomList.size(); i++)

add(geomList[i]);

}

void add(GeometryList & geomList, const Color & color, FillState is\_filled = UNFILLED) {

for (u\_int i=0; i<geomList.size(); i++)

add(geomList[i], color, is\_filled);

}

void add(GeometryList & geomList, const Color & color, u\_int pen\_width) {

for (u\_int i=0; i<geomList.size(); i++)

add(geomList[i], color, pen\_width);

}

void pause(u\_int delay\_ms) { // pause after after last canvas element added

if (elements.size() == 0) return;

elements[elements.size()-1]->pauseAfter += delay\_ms;

}

void draw(); // send all drawing elements to HTML/JS file

void draw(string filename); // draw to specified file

void clear() {

Polygon p;

p.add(Coord(0, 0));

p.add(Coord(\_xDim, 0));

p.add(Coord(\_xDim, \_yDim));

p.add(Coord(0, \_yDim));

this->add(p, bg\_color , FILLED);

}

~Canvas() {

for (u\_int i=0; i<elements.size(); i++)

if (elements[i] != NULL)

delete elements[i];

}

private:

string generateJSDrawingString(); // helper method for Canvas::draw()

};

} // end GeoDraw namespace

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* SIMPLIFIED C-STYLE INTERFACE for GeoDraw - Function Declarations

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using namespace GeoDraw;

// Returns the width (x direction) of the drawing canvas (number of pixels)

u\_int gd\_getCanvasSizeX ();

// Returns the height (y direction) of the drawing canvas (number of pixels)

u\_int gd\_getCanvasSizeY ();

// Change the canvas size. Side-effect: clears and resets the canvas.

void gd\_resetCanvasSize (u\_int xSize, u\_int ySize);

// Set the canvas background colour with a standard colour

// NOTE: Canvas colour will not change until gd\_clear() is called

// The available colours are:

// BLACK, WHITE, LIGHT\_GRAY, GRAY, DARK\_GRAY,

// LIGHT\_RED, RED, DARK\_RED, LIGHT\_GREEN, GREEN, DARK\_GREEN,

// LIGHT\_BLUE, BLUE, DARK\_BLUE, YELLOW, TEAL, CYAN, MAGENTA,

// PINK, ORANGE.

// Example Usage: gd\_setCanvasColor(DARK\_GRAY);

void gd\_setCanvasColor (Color color);

// Set the canvas background colour with a custom (red,green,blue) colour

// Colour components have values in the range 0.0 to 1.0

// For example (1.0,0.0,0.0) is red

void gd\_setCanvasColor (double r, double g, double b);

// Set the width of the drawing pen (number of pixels)

void gd\_setPenWidth (u\_int width);

// Set the drawing pen colour with a standard colour (see colour list above)

// Example Usage: gd\_setPenColor(BLUE);

void gd\_setPenColor (Color color);

// Set the drawing pen colour with a custom (red,green,blue) colour

// Colour components have values in the range 0.0 to 1.0

// For example, (0.0,1.0,0.0) is green

void gd\_setPenColor (double r, double g, double b);

// Set the colour for filled shapes with a standard colour (see colour list above)

// Example Usage: gd\_setFillColor(GREEN);

void gd\_setFillColor (Color color);

// Set the colour for filled shapes with a custom (red,green,blue) colour

// Colour components have values in the range 0.0 to 1.0

// For example, (0.0,0.0,1.0) is blue

void gd\_setFillColor (double r, double g, double b);

// Set the font to use when drawing text

// Options are: Arial, Courier, Times

// Example Usage: gd\_setFont(Courier);

void gd\_setFont (Font font);

// Set the height of text characters, in units of pixels

void gd\_setTextSize (u\_int font\_size);

// Set the text colour with a standard colour (see colour list above)

// Example Usage: gd\_setTextColor(ORANGE);

void gd\_setTextColor (Color color);

// Set the text colour with a custom (red,green,blue) colour

// Colour components have values in the range 0.0 to 1.0

// For example, (0.0,0.0,1.0) is blue

void gd\_setTextColor (double r, double g, double b);

// Set the text alignment relative to the coordinate at which text is drawn

// Options are: LEFT, CENTER, RIGHT

// Example Usage: gd\_setTextAlignment(CENTER);

void gd\_setTextAlignment (TextAlign alignment);

// Draw a point at the specified coordinate

// The last set pen width and colour will be used

void gd\_point (double x, double y);

// Draw a line segmented between points (x1,y1) and (x2,y2)

// The last set pen width and colour will be used

void gd\_line (double x1, double y1, double x2, double y2);

// Draw a circle centred at (x,y) with the specified radius

// The last set pen width and colour will be used

void gd\_circle (double x, double y, double radius);

// Draw a filled circle centred at (x,y) with the specified radius

// The last set fill colour will be used

void gd\_circleFilled (double x, double y, double radius);

// Draw a triangle with the specified vertices (x1,y1),(x2,y2),(x3,y3)

// The last set pen width and colour will be used

void gd\_triangle (double x1, double y1, double x2, double y2,

double x3, double y3);

// Draw a filled triangle with the specified vertices (x1,y1),(x2,y2),(x3,y3)

// The last set fill colour will be used

void gd\_triangleFilled (double x1, double y1, double x2, double y2,

double x3, double y3);

// Draw a quadrilateral with the specified vertices (x1,y1),(x2,y2),(x3,y3),(x4,y4)

// The last set pen width and colour will be used

void gd\_quad (double x1, double y1, double x2, double y2,

double x3, double y3, double x4, double y4);

// Draw a filled quadrilateral with the specified vertices (x1,y1),(x2,y2),(x3,y3),(x4,y4)

// The last set fill colour will be used

void gd\_quadFilled (double x1, double y1, double x2, double y2,

double x3, double y3, double x4, double y4);

// Draw text at the specified (x,y) coordinate

// The last set font, text size, text colour and text alignment will be used

// txt is a C++ string that may be passed a string literal, such as "Hello World!"

void gd\_text (string txt, double x, double y);

// Pause for a given number of milliseconds before displaying the next drawing element added

// after calling gd\_pause(). Example Usage:

// gd\_circle(100,100,50);

// gd\_pause(1000); // pause 1 second before next circle is drawn

// gd\_circle(200,200,20);

void gd\_pause (u\_int pauseTimeMs);

// Clear the drawing canvas. The canvas will be repainted

// with the last set background colour (see gd\_setCanvasColor())

void gd\_clear ();

// Write the canvas drawing with all previous draw and pause commands to file.

// The output file format is HTML with embedded JavaScript.

// The filename should be given a .html extension so it can be opened easily in a web browser.

// Note: saving does not clear the canvas state.

void gd\_save (string filename);

#endif // GEODRAW\_H\_INCLUDED

GeoDraw.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GeoDraw Version 1.0 (23.10) - Last revision: 01/10/2023

- A simple C/C++ 2D drawing library that outputs drawings as JavaScript

- Includes a simplified C function interface

NOTICE: Copyright (C) 2023 Conor McArdle, Dublin City University - All Rights Reserved.

Permission is granted to students registered for DCU module EE219: C/C++ for Engineers

to use this code for their personal use only. No permission is granted to share this file

by any means, including but not limited to, uploading it or submitting it to any private

or public server on the Web/Internet or any online service which publishes or otherwise

processes or stores uploaded content. This notice must stay attached to this file.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "GeoDraw.h"

namespace GeoDraw {

const string \_gd\_version = "23.10"; // software version number

string \_gd\_html\_pre; // html to appear before canvas element

string \_gd\_html\_post; // html to appear after canvas element

string gd\_to\_string(int a) { // utility string conversion methods

char buffer[50];

sprintf(buffer, "%d", a);

return string(buffer);

}

string gd\_to\_string(double a, unsigned short decimalPlaces = 2) {

char buffer[50];

string format = "%." + gd\_to\_string(decimalPlaces) + "f";

sprintf(buffer, format.c\_str(), a);

return string(buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Canvas class - public methods - Canvas::draw() and Canvas::draw(filename)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Canvas::draw() {

string JSDrawingString;

// HTML output file header with drawing functions

JSDrawingString += "<!DOCTYPE html>\n<html lang=\"en\">\n<head>\n<meta charset=\"utf-8\">\n<title>GeoDraw</title>\n";

JSDrawingString += "<style>\n canvas { border: 1px solid #707070; }\n</style>\n<script>\n\n";

JSDrawingString += "window.onload = function() {\n";

JSDrawingString += "var canvas = document.getElementById(\"myCanvas\");\n";

JSDrawingString += "var ctx = canvas.getContext(\"2d\");\n";

JSDrawingString += "ctx.fillStyle = 'rgb" + bg\_color.to24BitColorString() + "';\n";

JSDrawingString += "ctx.fillRect(0,0,canvas.width,canvas.height);\n\n";

// Generate JavaScript drawing commands from canvas elements

cout << "Generating " << elements.size() << " JavaScipt drawing objects ... ";

clock\_t start, end;

start = clock();

JSDrawingString += this->generateJSDrawingString();

end = clock();

cout << "done in ";

cout << double(end - start)/double(CLOCKS\_PER\_SEC) << " seconds." << endl;

// HTML output file footer

JSDrawingString += "\n};\n</script>\n</head>\n";

JSDrawingString += "<body style=\"background-color:gray;\">\n";

JSDrawingString += \_gd\_html\_pre;

JSDrawingString += " <canvas id=\"myCanvas\" width=\"" + gd\_to\_string((int)\_xDim) + "\" " + "height=\"" + gd\_to\_string((int)\_yDim) + "\"></canvas>\n";

JSDrawingString += "<p style=\"font-family:Arial;font-size:12px;color:LightGray\">&nbsp;Produced by GeoDraw-" + \_gd\_version + " C/C++ Library, conor.mcardle@dcu.ie, 2023</p>";

JSDrawingString += \_gd\_html\_post;

JSDrawingString += "</body>\n</html>\n";

// Save the JS string to file

std::ofstream outHTMLFile;

outHTMLFile.open(outFileName.c\_str());

if(outHTMLFile) {

cout << "Saving to file ... ";

outHTMLFile << JSDrawingString;

outHTMLFile.close();

cout << "done." << endl;

cout << "JavaScript drawing created in " << outFileName << endl;

} else {

cerr << "Error opening output file " << outFileName << endl;

}

}

void Canvas::draw(string filename) {

outFileName = filename;

Canvas::draw();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Canvas class - private helper members

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string Canvas::generateJSDrawingString() {

string JSDrawingString; // string to store JavaScript drawing code

u\_int time = 0; // current canvas element draw time

// Generate JavaScript for each geometry element,

// instering drawing pause events at appropriate times

for (u\_int i=0; i<elements.size(); i++) {

JSDrawingString += elements[i]->toJavaScript() + "\n";

if (elements[i]->pauseAfter != 0) {

// if there was a previous pause event, close the JS timeout function

if (time > 0)

JSDrawingString += "}, " + gd\_to\_string((int)time) + ");\n";

// open new JS timeout function

JSDrawingString += "\nsetTimeout(function() {\n";

// update current display time

time += elements[i]->pauseAfter;

}

}

if (time > 0) // close last pause timeout function

JSDrawingString += "}, " + gd\_to\_string((int)time) + ");\n";

return JSDrawingString;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JavaScript drawing implementations for Drawable objects

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Point::toJavaScript() - private

A friend of the Canvas class

Called by Canvas::draw() to generate JavaScript to draw a Point object

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string Point::toJavaScript() const {

string JS\_string;

JS\_string += "ctx.beginPath();\n";

JS\_string += "ctx.fillStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.arc(" + gd\_to\_string(coord.x()) + "," + gd\_to\_string(coord.y());

JS\_string += "," + gd\_to\_string((int)(penWidth/2)) + ",0,2\*Math.PI);\n";

JS\_string += "ctx.fill();\n";

return JS\_string;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LineSeg::toJavaScript() - private

A friend of the Canvas class

Called by Canvas::draw() to generate JavaScript to draw a LineSeg object

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string LineSeg::toJavaScript() const {

string JS\_string;

JS\_string += "ctx.beginPath();\n";

JS\_string += "ctx.strokeStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.lineWidth = " + gd\_to\_string((int)penWidth) + ";\n";

JS\_string += "ctx.moveTo(" + gd\_to\_string(c1.x());

JS\_string += "," + gd\_to\_string(c1.y()) + ");\n";

JS\_string += "ctx.lineTo(" + gd\_to\_string(c2.x());

JS\_string += "," + gd\_to\_string(c2.y()) + ");\n";

JS\_string += "ctx.stroke();\n";

return JS\_string;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Circle::toJavaScript() - private

A friend of the Canvas class

Called by Canvas::draw() to generate JavaScript to draw a Circle object

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string Circle::toJavaScript() const {

string JS\_string;

JS\_string += "ctx.beginPath();\n";

if (fillState == FILLED)

JS\_string += "ctx.fillStyle = 'rgb" + color.to24BitColorString() + "';\n";

else if (fillState == UNFILLED) {

JS\_string += "ctx.strokeStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.lineWidth = " + gd\_to\_string((int)penWidth) + ";\n";

}

JS\_string += "ctx.arc(" + gd\_to\_string(cen.x()) + "," + gd\_to\_string(cen.y());

JS\_string += "," + gd\_to\_string((int) radius) + ",0,2\*Math.PI);\n";

if (fillState == FILLED)

JS\_string += "ctx.fill();\n";

else if (fillState == UNFILLED)

JS\_string += "ctx.stroke();\n";

return JS\_string;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Polygon::toJavaScript() - private

A friend of the Canvas class

Called by Canvas::draw() to generate JavaScript to draw a Polygon object

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string Polygon::toJavaScript() const {

if (vertices.size() == 0) return "";

string JS\_string;

JS\_string += "ctx.beginPath();\n";

if (fillState == FILLED)

JS\_string += "ctx.fillStyle = 'rgb" + color.to24BitColorString() + "';\n";

else if (fillState == UNFILLED) {

JS\_string += "ctx.strokeStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.lineWidth = " + gd\_to\_string((int)penWidth) + ";\n";

}

JS\_string += "ctx.moveTo(" + vertices[0].toString() + ");\n";

for (unsigned int i=1; i<vertices.size(); i++)

JS\_string += "ctx.lineTo(" + vertices[i].toString() + ");\n";

JS\_string += "ctx.closePath();\n";

if (fillState == FILLED)

JS\_string += "ctx.fill();\n";

else if (fillState == UNFILLED)

JS\_string += "ctx.stroke();\n";

return JS\_string;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Text::toJavaScript() - private

A friend of the Canvas class

Called by Canvas::draw() to generate JavaScript to draw Text on canvas

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

string Text::toJavaScript() const {

string font\_name;

switch (this->font) {

case Arial: font\_name = "Arial"; break;

case Courier: font\_name = "Courier"; break;

case Times: font\_name = "Times"; break;

}

string JS\_string;

JS\_string += "ctx.font = \"" + gd\_to\_string((int)this->fontSize) + "px " + font\_name + "\";\n";

if (alignment == LEFT)

JS\_string += "ctx.textAlign = 'left';\n";

else if (alignment == CENTER)

JS\_string += "ctx.textAlign = 'center';\n";

else

JS\_string += "ctx.textAlign = 'right';\n";

JS\_string += "ctx.textBaseline = 'middle';\n";

if (fillState == FILLED) {

JS\_string += "ctx.fillStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.fillText(\"" + this->text + "\"," + this->position.toString() + ");\n";

}

else if (fillState == UNFILLED) {

JS\_string += "ctx.strokeStyle = 'rgb" + color.to24BitColorString() + "';\n";

JS\_string += "ctx.strokeText(\"" + this->text + "\"," + this->position.toString() + ");\n";

}

return JS\_string;

}

} // end GeoDraw namespace

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* SIMPLIFIED C-STYLE INTERFACE for GeoDraw - Implementation

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace GeoDrawC

{

u\_int \_cgd\_canvas\_size\_x = 600;

u\_int \_cgd\_canvas\_size\_y = 600;

u\_int \_cgd\_pen\_width = 2;

Color \_cgd\_pen\_color = BLACK;

Color \_cgd\_fill\_color = GRAY;

Font \_cgd\_font = Arial;

u\_int \_cgd\_font\_size = 20;

Color \_cgd\_font\_color = BLACK;

TextAlign \_cgd\_text\_alignment = LEFT;

Canvas \_cgd\_canvas(\_cgd\_canvas\_size\_x,\_cgd\_canvas\_size\_y);

}

u\_int gd\_getCanvasSizeX() {

return GeoDrawC::\_cgd\_canvas.xDim();

}

u\_int gd\_getCanvasSizeY() {

return GeoDrawC::\_cgd\_canvas.yDim();

}

void gd\_resetCanvasSize(u\_int xSize, u\_int ySize) {

GeoDrawC::\_cgd\_canvas = Canvas(xSize, ySize);

}

void gd\_setCanvasColor(Color color) {

GeoDrawC::\_cgd\_canvas.setBackgroundColor(color);

}

void gd\_setCanvasColor(double r, double g, double b) {

GeoDrawC::\_cgd\_canvas.setBackgroundColor(Color(r,g,b));

}

void gd\_setPenWidth(u\_int width) {

GeoDrawC::\_cgd\_pen\_width = width;

}

void gd\_setPenColor(Color color) {

GeoDrawC::\_cgd\_pen\_color = color;

}

void gd\_setPenColor(double r, double g, double b) {

GeoDrawC::\_cgd\_pen\_color = Color(r,g,b);

}

void gd\_setFillColor(Color color) {

GeoDrawC::\_cgd\_fill\_color = color;

}

void gd\_setFillColor(double r, double g, double b) {

GeoDrawC::\_cgd\_fill\_color = Color(r,g,b);

}

void gd\_setFont(Font font) {

GeoDrawC::\_cgd\_font = font;

}

void gd\_setTextSize(u\_int font\_size) {

GeoDrawC::\_cgd\_font\_size = font\_size;

}

void gd\_setTextColor(Color color) {

GeoDrawC::\_cgd\_font\_color = color;

}

void gd\_setTextColor(double r, double g, double b) {

GeoDrawC::\_cgd\_font\_color = Color(r,g,b);

}

void gd\_setTextAlignment(TextAlign alignment) {

GeoDrawC::\_cgd\_text\_alignment = alignment;

}

void gd\_point(double x, double y) {

GeoDrawC::\_cgd\_canvas.add(Point(x,y), GeoDrawC::\_cgd\_pen\_color, GeoDrawC::\_cgd\_pen\_width);

}

void gd\_line(double x1, double y1, double x2, double y2) {

GeoDrawC::\_cgd\_canvas.add(LineSeg(x1,y1,x2,y2), GeoDrawC::\_cgd\_pen\_color, GeoDrawC::\_cgd\_pen\_width);

}

void gd\_circle(double x, double y, double radius) {

GeoDrawC::\_cgd\_canvas.add(Circle(x,y,radius), GeoDrawC::\_cgd\_pen\_color, GeoDrawC::\_cgd\_pen\_width);

}

void gd\_circleFilled(double x, double y, double radius) {

GeoDrawC::\_cgd\_canvas.add(Circle(x,y,radius), GeoDrawC::\_cgd\_fill\_color, FILLED);

}

void gd\_triangle(double x1, double y1, double x2, double y2, double x3, double y3) {

Polygon poly;

poly.add(x1,y1);

poly.add(x2,y2);

poly.add(x3,y3);

GeoDrawC::\_cgd\_canvas.add(poly, GeoDrawC::\_cgd\_pen\_color, GeoDrawC::\_cgd\_pen\_width);

}

void gd\_triangleFilled(double x1, double y1, double x2, double y2, double x3, double y3) {

Polygon poly;

poly.add(x1,y1);

poly.add(x2,y2);

poly.add(x3,y3);

GeoDrawC::\_cgd\_canvas.add(poly, GeoDrawC::\_cgd\_fill\_color, FILLED);

}

void gd\_quad(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4) {

Polygon poly;

poly.add(x1,y1);

poly.add(x2,y2);

poly.add(x3,y3);

poly.add(x4,y4);

GeoDrawC::\_cgd\_canvas.add(poly, GeoDrawC::\_cgd\_pen\_color, GeoDrawC::\_cgd\_pen\_width);

}

void gd\_quadFilled(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4) {

Polygon poly;

poly.add(x1,y1);

poly.add(x2,y2);

poly.add(x3,y3);

poly.add(x4,y4);

GeoDrawC::\_cgd\_canvas.add(poly, GeoDrawC::\_cgd\_fill\_color, FILLED);

}

void gd\_text(string txt, double x, double y) {

GeoDrawC::\_cgd\_canvas.add(Text(txt, x, y, GeoDrawC::\_cgd\_font, GeoDrawC::\_cgd\_font\_size, GeoDrawC::\_cgd\_text\_alignment), GeoDrawC::\_cgd\_font\_color, FILLED);

}

void gd\_pause(u\_int pauseTimeMs) {

GeoDrawC::\_cgd\_canvas.pause(pauseTimeMs);

}

void gd\_clear() {

GeoDrawC::\_cgd\_canvas.clear();

}

void gd\_save(string filename) {

GeoDrawC::\_cgd\_canvas.draw(filename);

}

///////////////////////////////////////////////////////////