CS315 – Project Part I

27/11/2016

Atlangoc User Manual

(Figure Drawing Language)

Name	ID Number
Cihangir Mercan	21301813
Mehmet Nuri Yumuşak	21302303
Ahmet Taha Albayrak	21301440

Table of Contents

1. Introduction

Atlangoc is a geometric figure drawing language in which users can draw simple to complex shapes and figures on a canvas. It will be a simple, efficient, understandable and visually beautiful programming language.

“Increasingly, people seem to misinterpret complexity as sophistication, which is baffling. The incomprehensible should cause suspicion rather than admiration.”
(Swiss computer scientist: Niklaus Wirth (1934-))

The purpose of this tutorial is to give an idea about the syntax of the language and the usage of its features.

2. Entry point

Atlangoc programs starts executing at the main function. It has the following function heading:

```
void main(width, height, defaultStrokeWidth) {

}
```

This is the function that does drawings. The canvas has the properties width and height. Also, the shapes inside canvas will have a default stroke width.

3. Built-in shapes

Atlangoc has three types of built-in shapes which are Line, Rectangle and Oval. The syntaxes for these will be explained later at the part 8.

4. Drawing independent of scale

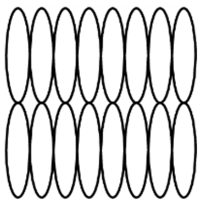
Atlangoc provides user with the ability to drawing with explicit sizes. For example, they will be able to draw 16 ovals in 2 rows and 8 columns or 4 rectangles in 4 rows and 1 column, without specifying absolute values for the oval width and height by the following way.

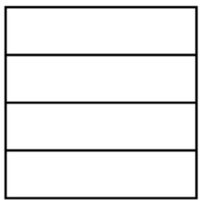
```
void main(200, 200, 1) {

    // sixteen ovals
    drawOval(2, 8);
}

void main(200, 200, 1) {

    // four rectangles
    drawRectangle(4, 1);
}
```





The function has two parameters: first parameter is the number of rows and the second parameter is the number of columns. As it is seen in the figure, in the 200x200 canvas, there are 16 ovals at the first one and 4 rectangles at the second one.

5. Variables

The language will be dynamically typed which means the variable type will be associated with run-time values. Hence, there won't be named variables.

These primitive types will be supported: float, integer, string, and boolean. Their syntax will be the following:

```
void main(200, 200, 1) {
    myName = "Taha"; // string
    myAge = 22; // integer
    myCgpa = 3.82; // float
    isStudent = true; // boolean
}
```

Basic arithmetic and logical operations will be supported on these types. Also, strings will support concatenation via the + operator and conversion from the other types (integers and floats).

```
void main(200, 200, 1) {
    myAge = 22;
    myAge = myAge + 3;

    myName = "Taha";
    me = "Taha: " + myAge; // string must come first in order not to get an
error

    trivialVariable = true && false;
}
```

6. Location, Size and Color objects

Location constructor will have two parameters: x and y which are coordinates. Size constructor will have two parameters: width and height. Color will have two constructors: first one has one parameter which gets color name as a string from predefined set of constants; second one is RGB constructor which gets three integers (smaller than 255).

```
void main(200, 200, 1) {
    location = Location(40, 20); // x-axis and y-axis
    size = Size(50, 50); // width and height
    favoriteColor = Color("yellow"); // with lower-case convention
    uglyColor = Color(120, 120, 120); // rgb
}
```

Predefined colors: pink, red, orange, yellow, brown, green, cyan, blue, purple, magenta, white, gray, black.

7. Bounding box

Bounding boxes will be used for drawing shapes inside. It will define the shape's location and size which will be drawn inside. It has simple constructor which gets Location and Size as parameters.

```
boundingBox = BoundingBox(location, size); // location and size are both
objects not primitive types
```

8. Syntax of built-in shapes (Line, Rectangle and Oval)

8.1 Line

8.1.1 Properties

Line has eight properties: xStart, xEnd, yStart, yEnd, strokeWidth, hasArrow, color, direction.
- xStart, xEnd, yStart and yEnd will be integers. However, user will not manually give them as integers. They will give start location and end location as Location object. Hence, Atlangoc will copy the values from these locations. Example of this will be given at section 8.1.2.
- strokeWidth will be integer. This value will be relative to the default stroke width of the page. For example: if the default stroke width is 1 and user gives 2 as a strokeWidth, the stroke width will be 1 * 2 = 2.
- arrow is an integer. 0 means no arrow, 1 means arrow at the start, 2 means arrow at the end, 3 means arrow at the both start and end.
- color will be Color type.
- direction is a string from predefine set of constants: “NE (north east)”, “NW”, “SE”, “SW”.

8.1.2 Constructors

Line type will have six constructors.

- First one has 2 parameters:
```
line = Line(startLoc, endLoc);
```
In this case, strokeWidth will be equal to defaultStrokeWidth and there will not be any arrow. Color will be black by default. There is no need to specify direction since it has start and end location. This line will not have ability to be drawn inside bounding box since it has locations specified.

- Second one has 4 parameters:
```
line = Line(startLoc, endLoc, strokeWidth, arrow);
```
In this case, strokeWidth and arrow are specified as a difference from the first one.

- Third one has 5 parameters:
```
line = Line(startLoc, endLoc, strokeWidth, arrow, color);
```
In this case, strokeWidth, arrow and color are all specified.

- Fourth one has 1 parameter: (bounding-box suitable)
```
line = Line(direction);
```
It does not take any location; it takes direction instead of location. strokeWidth will be equal to defaultStrokeWidth and there will be no arrow. Color will be black by default. If this line is not drawn inside bounding box, program will give error.

- Fifth one has 3 parameters: (bounding-box suitable)
```
line = Line(direction, strokeWidth, arrow);
```
In this case, strokeWidth and arrow are specified as a difference from the fourth one.

- Sixth one has 4 parameters: (bounding-box suitable)
```
line = Line(direction, strokeWidth, arrow, color);
```
In this case, strokeWidth, arrow and color are all specified.

8.1.3 Drawing Line inside Bounding Box

Lines with these constructors are suitable for bounding boxes:
```
line = Line(direction);
line = Line(direction, strokeWidth, arrow);
line = Line(direction, strokeWidth, arrow, color);
```

Here is the example of a drawing line inside bounding box.

```
void main(200, 200, 1) {
    // define bounding box
    boundingBox = BoundingBox(Location(20, 20), Size(40, 40));

    // define line
    line = Line("NW", 3, 1); // direction, arrow (3 = both sides) and
stroke width = 1 * 1 = 1.

    // draw function which will be explained later
    draw(boundingBox, line);
}
```

8.2 Rectangle

8.2.1 Properties

Rectangle has seven properties: xPos, yPos, width, height, strokeWidth, isRounded, color, isFilled.
- xPos and yPos will be integers. However, user will not manually give them as integers. They will give Location object. Hence, Atlangoc will copy the values from this location. xPos and yPos will represent its left-hand corner coordinates.
- width and height will be integers. Again, user will not manually give them as integers. They will give Size object. Hence, Atlangoc will copy the values from this size.
- strokeWidth will be integer. This value will be relative to the default stroke width of the page. For example: if the default stroke width is 1 and user gives 2 as a strokeWidth, the stroke width will be 1 * 2 = 2.
- isRounded is boolean.
- color is Color type.
- isFilled is boolean.

8.2.2 Constructors

Rectangle type will have six constructors:

- First one has no parameters:
```
rectangle = Rectangle(loc, size);
```
In this case, strokeWidth will be equal to defaultStrokeWidth and the rectangle will not be rounded. Color will be black by default and it will not be filled. This rectangle will not have ability to be drawn inside bounding box since it has location and size specified.

- Second one has four parameters:
```
rectangle = Rectangle(loc, size, strokeWidth, isRounded);
```
In this case, strokeWidth and isRounded are specified as a difference from the first one.

- Third one has six parameters:
```
rectangle = Rectangle(loc, size, strokeWidth, isRounded, color, isFilled);
```
In this case, strokeWidth, isRounded, color and isFilled are all specified.

- Fourth one has no parameters: (bounding-box suitable)
```
rectangle = Rectangle();
```
In this case, it does not take location. Hence, if this rectangle is not drawn inside bounding box, program will give error. strokeWidth will be equal to defaultStrokeWidth and the rectangle will not be rounded. Color will be black by default and it will not be filled.

- Fifth one has two parameters: (bounding-box suitable)
```
rectangle = Rectangle(strokeWidth, isRounded);
```
In this case, strokeWidth and isRounded are specified as a difference from the fourth one.

- Sixth one has four parameters: (bounding-box suitable)
```
rectangle = Rectangle(strokeWidth, isRounded, color, isFilled);
```
In this case, strokeWidth, isRounded, color and isFilled are all specified.

8.2.3 Drawing Rectangle inside Bounding Box

Rectangle with these constructors are suitable for bounding boxes:
```
rectangle = Rectangle();
rectangle = Rectangle(strokeWidth, isRounded);
rectangle = Rectangle(strokeWidth, isRounded, color, isFilled);
```

Here is the example of a drawing rectangle inside bounding box.

```
void main(200, 200, 1) {
    // define bounding box
    boundingBox = BoundingBox(Location(20, 20), Size(40, 40));

    // define rectangle
    rectangle = Rectangle(1, 1) // stroke width = 1 and it is rounded

    // draw function which will be explained later
    draw(boundingBox, rectangle);
}
```

8.3 Oval

8.3.1 Properties

Oval has seven properties: xPos, yPos, width, height, strokeWidth, color, isFilled.
- xPos and yPos will be integers. However, user will not manually give them as integers. They will give Location object. Hence, Atlangoc will copy the values from this location. xPos and yPos will represent its left-hand corner coordinates.
- width and height will be integers. Again, user will not manually give them as integers. They will give Size object. Hence, Atlangoc will copy the values from this size.
- strokeWidth will be integer. This value will be relative to the default stroke width of the page. For example: if the default stroke width is 1 and user gives 2 as a strokeWidth, the stroke width will be 1 * 2 = 2.
- color is Color type.
- isFilled is boolean.

8.3.2 Constructors

Oval type will have four constructors:

- First one has no parameters:

```
oval = Oval(loc, size);
```

In this case, strokeWidth will be equal to defaultStrokeWidth. Color will be black by default and it will not be filled. This oval will not have ability to be drawn inside bounding box since it has location and size specified.

- Second one has four parameters:

```
oval = Oval(loc, size, strokeWidth);
```

In this case, strokeWidth is specified as a difference from the first one.

- Third one has six parameters:

```
oval = Oval(loc, size, strokeWidth, color, isFilled);
```

In this case, strokeWidth, color and isFilled are all specified.

- Fourth one has no parameters: (bounding-box suitable)

```
oval = Oval();
```

In this case, it does not take location. Hence, if this oval is not drawn inside bounding box, program will give error. strokeWidth will be equal to defaultStrokeWidth. Color will be black by default and it will not be filled.

- Fifth one has two parameters: (bounding-box suitable)

```
oval = Oval(strokeWidth);
```

In this case, strokeWidth `is specified` as a difference from the fourth one.

- Sixth one has four parameters: (bounding-box suitable)

```
oval = Oval(strokeWidth, color, isFilled);
```

In this case, strokeWidth, color and isFilled are all specified.

8.3.3 Drawing Oval inside Bounding Box

Oval with these constructors are suitable for bounding boxes:

```
oval = Oval();
oval = Oval(strokeWidth);
oval = Oval(strokeWidth, color, isFilled);
```

Here is the example of a drawing oval inside bounding box:

```
void main(200, 200, 1) {
    // define bounding box
    boundingBox = BoundingBox(Location(20, 20), Size(40, 40));

    // define oval
    oval = Oval(1, Color("red"), true); // stroke width = 1 and filled red

    // draw function which will be explained later
    draw(boundingBox, oval);
}
```

9. draw function

- Pre-created objects with location and size properties can be drawn by draw function with one parameter:

```
void main(200, 200, 1) {
    // define line
    line = Line(Location(20, 20), Location(60, 60), 1, 0, Color("yellow"));
// 1 stroke width, no arrow, yellow

    // draw it
    draw(line);

    // define rectangle
    rectangle = Rectangle(Location(40, 20), Size(60, 20), 1, true,
Color("green"), true); // 1 stroke width, rounded, green, filled

    // draw it
    draw(rectangle);

    // define oval
    oval = Oval(Location(80, 50), Size(30, 80), 1, Color("yellow"), true);
// 1 stroke width, rounded, yellow, filled

    // draw it
    draw(oval);
}
```

- Pre-created bounding-box suitable objects can be drawn inside bounding box by draw function with two parameters:

```
void main(200, 200, 1) {
    // define bounding box
    boundingBox = BoundingBox(Location(20, 20), Size(40, 40));

    // define oval
    oval = Oval(1, Color("red"), true); // stroke width = 1 and filled red

    // draw function which will be explained later
    draw(boundingBox, oval);
}
```

If objects with location and size specified are tried to be drawn inside bounding box, program will give error.

10. drawString function

Drawing strings are supported in these ways:

```
// default font is 11, color is black
drawString(string, location);

// font is integer
drawString(string, location, font);

// color is Color object
drawString(string, location, font, color); // font is integer
```

Example:
```
drawString("programming languages", Location(40,80));
```

11. drawLine, drawRectangle and drawOval functions

11.1 drawLine

Drawing line without creating it before-hand is supported in these ways:
```
// default stroke width, no arrows and black
drawLine(startLoc, endLoc);

// strokeWidth is relative to default stroke width and it is integer
// arrow= 0: no arrow, 1: at start loc, 2: at end loc, 3: at both
locationsdrawLine(startLoc, endLoc, strokeWidth, arrow)

// color is Color object
drawLine(startLoc, endLoc, strokeWidth, arrow, color);
```

Example:
```
drawLine(Location(20,10), Location(60,10), 1, 3, Color("yellow"));
```

11.2 drawRectangle

Drawing rectangle without creating it before-hand is supported in these ways:
```
// default stroke width, not rounded, not filled, black
drawRectangle(loc, size);

// strokeWidth is relative to default stroke width and it is integer
// isRounded is boolean: true or false
drawRectangle(loc, size, strokeWidth, isRounded);

// color is color object, isFilled is boolean: true of false
drawRectangle(loc, size, strokeWidth, isRounded, color, isFilled);
```

Example:
```
drawRectangle(Location(60,30), Size(30,60), 1, true, Color("yellow"),
true);
```

11.3 drawOval

Drawing oval without creating it before-hand is supported in these ways:
```
// default stroke width, not filled, black
drawOval(loc, size);

// strokeWidth is relative to default stroke width and it is integer
drawOval(loc, size, strokeWidth);

// color is color object, isFilled is boolean: true of false
drawOval(loc, size, strokeWidth, color, isFilled);
```

Example:
```
drawOval(Location(60,30), Size(30,60), 1, Color("yellow"), true);
```

12. for, while and do/while

Atlangoc will support loops that can ease repetitive tasks. The syntaxes of those will be similar to the other programming languages.

for loop:

```
void main(200, 200, 1) {
    for (i = 1; i <= 10; i++) {
        drawLine(Location(10 * i, 10), Location(10 * i + 10, 10));
    }
}
```

while loop:

```
void main(200, 200, 1) {
    i = 1;
    while (i <= 10) {
        drawLine(Location(10 * i, 10), Location(10 * i + 10, 10));
        i++;
    }
}
```

do/while loop:

```
void main(200, 200, 1) {
    i = 1;
    do {
        drawLine(Location(10 * i, 10), Location(10 * i + 10, 10));
        i++;
    } while (i <= 10);
}
```

13. if and if/else

Atlangoc will support conditionals. The syntaxes of those will be similar to the other programming languages.

```
void main(200, 200, 1) {
    isWeatherCold = true;
    if (isWeatherCold) {
        drawString("weather is cold", Location(10, 10));
    } else {
        drawString("weather is not cold", Location(10, 10));
    }
}
```

14. Defining own function

Atlangoc will support modularity at the level of functions. Hence, user can define their own functions with or without parameters outside main function. However, these functions cannot return anything, they are only void. The syntax is the following:

```
function functionName(anInteger, aString, aLocation, aSize) {
   // body
}
```

Example:

```
void main(200, 200, 1) {
    // call method
    drawThreeOvalsNextToEachOther(10, 10);
}

function drawThreeOvalsNextToEachOther(xPos, yPos) {
    // define size
    size = Size(10, 10);

    // define locations
    locOne = Location(xPos, yPos);
    locTwo = Location(xPos + 10, yPos + 10);
    locThree = Location(xPos + 20, yPos + 20);

    // draw them
    drawOval(locOne, size);
    drawOval(locTwo, size);
    drawOval(locThree, size);
}
```

15. Extensions of shapes

Atlangoc will support extensions of shapes. Hence, user will be able to create extensions with or without parameters outside main function. Extensions are like collection of shapes. However, user will be able to define everything (string, integer, Location, Size, Color, BoundingBox, Line, Rectangle, Oval etc.) inside extensions. On the other hand, they won't be able to call any function inside. When they want to draw extension inside main with method: `draw(extension)`, only Line, Rectangle and Oval objects will be realized. If they have location and size specified, they will be drawn. Otherwise, they will not be drawn. The syntax is the following:

```
extension ExtensionName(anInteger, aString, aLocation, aSize) {
   // body
}
```

Example:

```
void main(200, 200, 1) {
    // create extension
    myFavoriteExtension = ThreeOvalsNextToEachOther(10, 10);

    // draw extension
    draw(myFavoriteExtension);
}

extension ThreeOvalsNextToEachOther(xPos, yPos) {
    // define size
    size = Size(10, 10);

    // define color
    color = Color(40, 80, 120);
    // define relative stroke width
    relativeStrokeWidth = 1;

    // define isFilled
    isFilled = true;

    // define locations
    locOne = Location(xPos, yPos);
    locTwo = Location(xPos + 10, yPos + 10);
    locThree = Location(xPos + 20, yPos + 20);

    // define ovals that will be drawn
    ovalOne = Oval(locOne, size, relativeStrokeWidth, color, isFilled);
    ovalTwo = Oval(locTwo, size, relativeStrokeWidth, color, isFilled);
    ovalThree = Oval(locThree, size, relativeStrokeWidth, color, isFilled);

}
```