APES Tutorial

Apes is a simple shape drawing language which is designed to be simple as it can be but having the core functionalities that a figure drawing language must have.

## Primitive Types & Their Operations

Apes supports strings, integers, float numbers, logical functions and operators. In this part you are going to learn how to use those. Initialization of a primitive type is done by using an Identifier and an assignment operation with the respective type, as follows:

stringVariable = “This is a String”;

integerVariable = 2;

floatVariable = 3.4;

booleanVariable = True;

The Language is dynamically typed, therefore, such operations are allowed:

stringVariable = 3.4;

Comments and inner comments are supported in Apes.

/\* Not another comment /\* It is just another comment \*/ Nope \*/

## Location, Size and Colour Types

Location and Size will function in an analogous way. We will create a Location variable and a Size variable called location and size respectively:

location = Location ( 50, 20);

size = Size( 100, 20);

We can access the elements of Location and Size using ‘,’ operator:

widthValue = size.value; *#widthValue will have the value of 100*

yCoordinate = location.y; *#yCoordinate will have the value of 20*

Colour will be created using three integer parameters or using a string matching one of the constants defined in the report:

colourVar = Colour ( 128,128,250);

colourVar2 = Colour (“Green”);

We can also use inline initialization which is demonstrated in the “Draw Function” part.

## Built-in Shapes

To create a new shape, we have to follow this syntax: IDENTIFIER = ShapeName( *Parameters* );. Examples for each type will be shown below:

We will create a line called endLine that is pointing east with an end arrow and no start arrow:

endLine = Line( direction: “E”, endArrow: True);

In this example, we have demonstrated the use of named parameters. We could have also initiated the Line without using named parameters in the format below. However, using named parameters is preferred.

endLine = Line ( “E”, False, True, 1);

We will create an Oval and a rectangle with rounded corners of value 5:

simpleOval = Oval();

cuteRectangle = Rectangle(5);

## Composite Shapes

Apes allows you to create composite shapes from the built-in shapes.

simpleOval = Oval();

cuteRectangle = Rectangle(5);

Composite(“newComp”,simpleOval,cuteRectangle);

Last code line is going to create a composite shape of the cuteRectangle and simpleOval.

simpleOval = Oval();

anotherOval = Oval();

Think that simpleOval and anotherOval are ovals having the same location and size.

Composite(“newComp”,simpleOval,anotherOval);

This line is going to create a composite shape of two ovals differing only x coordinates of their location to prevent the creation of the built-in shapes as composite shape.

## Draw Function

We will draw the shapes we create using the built in draw function. In the draw function, we can specify additional parameters for location, size, fillState, colour, etc.

We will draw endLine that we have created in the previous section starting at coordinates (50,100) with a width of 30 pixels and height of 5. We will fill the line with a red colour. We will also use named parameters:

newLine.draw( Location: Location(50,100), Size: Size(30,5), fillState: True, fillColour: Colour (255,0,0) );

Note the following things: it is possible to create Location. Colour, and Size instances in line without having to create a variable holding their values before. It was also possible to use the following constructor for colour: Colour(“Red”).

## Loops

***While Loop***

while(statement){...}

If the statement inside the parentheses is true, the code block within the curly braces will be executed. It going to continue doing the same procedure, if statement is not false. If statement is false, it is going to skip this code block.

|  |
| --- |
| x=5; |
| while(x<8) { |
| x = x+1; |
| } |

After the execution of this code block the new value of x is 8.

***Do While Loop***

do{...}while(statement)

Without checking the statement inside braces code block is going to be executed once. After that, it behaves like an ordinary while loop. If the statement is true the code block inside the curly braces is going to be executed. If it is not, it is going to skip this code block.

|  |
| --- |
| x=5; |
| do{ |
| x= x+1; |
| } |
| while(x<5) |

After the execution of this code block the new value of x is 6.

|  |
| --- |
| x=5; |
| do{ |
| x= x+1; |
| } |
| while(x<8) |

After the execution of this code block the new value of x is s 8.

***For Loop***

for(expression1, statement, expression2){...}

For loop enables you to create a expression exclusive for itself(expression1) and loop over the condition of the statement. Second expression(expression2) is the changes has to be made after each iteration of the code block.

|  |
| --- |
| x = 8; |
| for(y=3 ; y<x ; y = y+1) { |
| x = x-1; |
| } |

After the execution of this code block the new value of x is 5.

## Conditional Statements

if(statement){...}else{...}

The code block in the first pair of curly braces is going to be executed only if the statement is true. If the statement is not true code block in the second pair of braces is going to be executed.

|  |
| --- |
| x=5; |
| if(x<7){ |
| x=x-1; |
| } |
| else{ |
| x=x+1; |
| } |

After the execution of this code block the new value of x is 4.