Geo Language Proposal

Group Members:

PM (make sure conductible, write-ups):

Language Guru (design language syntax):

System Architect (based on syntax, build compiler):

Tester (test test test test):

Geo is a language that enables physicists, architects, mathematicians, as well as other geometry-interested professionals to solve spatial problems efficiently. It involves functionalities that computes relationships among geometric figures, including lines, circles, and rectangles, etc. Users are able to define figures, set their moving patterns, and perform various analysis of the interacting figures. If desired, a user can also change the moving pattern or shape of objects.

The first part of the language is the declaration of various settings, including panel, mode choices, and coordinate system.

|  |
| --- |
| @Panel panelDemo /\* Define panel \*/  @Mode Console /\* Mode choices: Console & Panel \*/  //@Co Cart /\* Default: Cartesian coordinate system \*/ |

The second part of the language defines the figure elements in the panel, and figures can be declared anywhere in the program. Figure elements include dot, line, rectangle, and circle. A dot is defined by its x and y coordinates, a line by its slope and intersect or two coordinates, a rectangle by a coordinate and its width and height, and a square by its center and radius.

|  |
| --- |
| /\* Dot definition \*/  dot1 = [2,3];  dot2 = [3,4];  /\* Line definition \*/  1ine1 = Line(3,4); //Line(a,b) denotes the function y=ax+b  line2 = Line([0,0],[5,10]); //  /\* Rectangle definition \*/  rect1 = Rect([2,5],3,5);  /\* Circle definition \*/  circle1 = Circle([3,5],5); |

The third part includes the built-in functions (Not sure about the name build-in functions here…...) for each figure element, including intersect, distance calculation, and print commands, etc.

|  |
| --- |
| /\* Dot.Distance(Dot):Float or Dot.Distance(Line):Float \*/  /\* Line.Intersect(Circle):Float \*/  /\* Line.Expression:string Circle.Expression \*/  /\* Console mode ONLY \*/  /\* Println(String/Float/Int/Dot)\*/  Println("Helloworld!");  Println(line1.Expression);  Println(circle1.Expression);  Println(dot1.Distance(dot2)); |

Another part is setting up modes for specific attributes of figures (Not sure here…...), or relationships between figures. For example, one can set the distance between a dot and the center of a circle static, and the length between a segment and circle dynamic.

|  |
| --- |
| /\* Panel Mode ONLY\*/  Info.add("Distance1",dot1.Distance(circle1.Center),static);  Info.add("Length1",Length(line2,circle1),dynamic); |

Further, timers are set up and added to figure elements, and dynamic movements are performed interactively. (Not sure what the Run(20,line1,circle1); means?)

|  |
| --- |
| t1 = Timer(0.5);  t2 = Timer(1);  line1.AddTimer(t1,a);//  circle.AddTimer(t2,r);//(x-a)^2+(y-b)^2=r^2  /\* Run line1 & circle1 20 times. Note that line1 & circle must both have a timer \*/  Run(20,line1,circle1); |

Finally, a standard library is built in for I/O performance, including various print commands.

|  |
| --- |
| Println(line1.Expression);  Println(circle1.Expression);  /\* Println function allows different data type in one function, divided by comma. The following example prints a string and float\*/  Println("Intersection length: ",line1.Intersect(circle1));  End  If(line1.Intersect(circle1) > 0):  Println("Intersection");  Elif(line1.Intersect(circle1) == 0):  Println("Tangent");  Else:  Println("Neither intersection nor tangent");  End  @End Panel |