SOIL
Simple Object Interaction Language

Language Proposal

COMS W4115: Programming Languages and Translators
Professor Stephen A. Edwards

Richard Zieminski
rez2107@columbia.edu

Objective:
To create a language which could be used to teach the concepts of object interactions to young children. While this is the specified audience, this language could easily be adapted and used for any number of simulations. To help with understanding, the system will output the results to the screen in a 2D animation format.

Motivation:
While watching my one year old son try to put a square peg into a round hole I realized that this action, while learnable could be taught better. If I could somehow show him how the basic characteristics, such as shape and hardness, affect how objects interact, I thought I could help accelerate his learning curve. Of course I don't expect him to be able to use it directly, but this could be an invaluable aid to the parents as the child develops.

Features:
- The small size (command set) of the language makes it easier to learn.
- Concepts and commands are presented in a more human friendly way.

Language Overview:
As described, this language is meant to bring simplified learning and experimental abilities to the average user. A person can create simple objects which are composed of basic shapes, and then assign basic properties which characterize them. The four basic shapes are squares, circles, lines, and triangles and can be combined to create more complex entities. Objects can them be assigned as static or movable, and then put in to motion to see how they interact with each other. Depending on their preset properties and described rule sets, the objects will experience any number of changes when interacting physically with each other. These changes might be result in distortion or destruction of an object, or an object moving through another through compression. The objects will be represented by different colors and the shapes (or combinations of) as defined.

User (non-reserved) identifiers can start with any alphabet character, but not numbers. All functions parameters are passed by value, not reference. The language is intended to abstract low level concepts, such as variable types away from the user as much as possible. To do this simple, everyday terms are being used as much as possible. To further simply things, loops will be understood, not implemented, so a simple go/stop command will control movement. While functionality will be supported but only to control how long motion can take place. While will have the notion of seconds and minutes of execution only.

When building objects, combinations of two or more objects will results in combined characteristics along with the assumption that 'weaker of the overlapping properties will propagate forward. (i.e. hardness, elasticity, etc.) Adding/subtracting objects will increase/decrease properties linearly.

Functions will be defined through the use of the whatif keyword. Function parameters will be accessible via. the '.' dot operator as with other languages.

Language Attributes:

Blank spaces are used as separators for keywords, operators, and code.

Reserved keywords:

if, then, else (elseif has been left out to simplify things). All rules will have only two possible outcomes. Rules than required more outcomes should extend from other rules. Since rules contain there own if/then behavior, this allows for a more clear definition of the interaction outcomes.
stop, go, turn
left, right, turnaround
c*reate, destroy*
combine
compare - (returns =, >, <) based on defined characteristic property definitions. Weighting of properties will in the end allow for a win/loss decision as to which object dominates the outcome of the interaction.
touches/bumps
inform (similar to printf)
whatif
or
and
world – (the 'playing field')
while
end
true
false
hardness
elasticity
direction
position

qualifiers (these define/qualify characteristics in common terms. Their default values can be overridden as design time. They give a quick way to describe characteristics without defining value ranges and let interactions between objects experience relative outcomes.

low/medium/high
min/max/middle
rigid, pliable

// comments (inline only supported)

Data Types:

Text (strings)
Number (only whole numbers are supported, +/-)
Boolean

Operators:

+ (addition)
- (subtraction)
/
*
>
<
=
>=
<=

Sample Interaction/Program:
This program creates two objects, one static, one mobile. Upon creation the mobile object is then set in motion.

create(shape, color, width, height, max_speed) *max_speed 0 = static object*

```
Shape1 = create (circle, RED, 50, 0, 0)
Shape1a = create (square, RED, 50, 50, 0)
Shape1a.rotateLeft

ShapeFinal = combine Shape1 and Shape1a

ShapeFinal.position = random
ShapeFinal.hardness = max
ShapeFinal.elasticity= rigid

Shape2 = create (circle, BLUE, 10, 0, 10)
Shape2.position = center
Shape2.speed = 3
```

```
Shape2.direction = left
Shape2.hardness = medium
Shape2.elasticity = rigid

// Parameter based rule – only applied when shapes are passed into it
create whatif Collision (shape_1, shape_2)
    If shape1 touches shape_2 then
        loser = compare shape_1, shape_2
        destroy loser
    End

// Parameter based rule – only applied when shapes are passed into it
create whatif Lost (shape_1, shape_2)
    If shape_2.position - shape_1.position > 100 then
        destroy shape_2
    Else
        inform "I'm lost"
    End

// Assign generic rules to specific objects - the
Collision (Shape1, Shape2)

// Run the simulation
go
    while (time < 5)
    end
stop
```

If a shape interacts with the 'world' (room constraints) it will interact as with any other object as the world has a predefined notion of properties (it's a shape also made up of lines, which properties can be overridden/modified)