moveIt

Language Reference Manual

Chengchen Sun (cs2890)

Benjamin Kornacki (blk2129)

Thomas Rantasa (tr2286)

Introduction

This is the Language Reference Manual of the language: **moveIt**. The default file extension for source code is **.tbc**.

This language is devised by Thomas Rantasa (tr2286), Benjamin Kornacki (blk2129) and Chengchen Sun (cs2890) as a project in Programming Languages and Translators by Professor Stephen A. Edwards.

Basic Types:

Tokens:

There're six classes of tokens: Identifiers, Keywords, Constants, String literals, Operators, Separators.

Comments:

Paired comment symbols are `(:` and `:)` which indicates the start and end of comments. Comments do not nest.

Identifiers:

An identifier of a single entity (not an array) starts with a dollar sign `'$'`, followed by a sequence of letters and digits. Upper and lower case letters are different. Identifier length must be within 31 characters.

Arrays will be identified with the `'%'` symbol. However, individual elements of the array are identified as single entities and therefore use the dollar sign.

Identifiers can refer to variable types, graphic object types and bindings.

Keywords:

The following keywords are reserved and cannot therefore be used as an identifier:

variable type declaration: `int, float, char, struct`
graphic object type declaration: `dot, line, triangle, circle`
movement manipulation: `time, in`

object binding: `bind`

program control: `if, else, while, for, do`

global control setting: `define`

Constants & String Literals:

Constants can also be defined with both variable types, graphic object types and bindings, like identifiers. String Literals will be stored in array of char constants.

Separators:

Normally, separators include whitespace, tab indentation, and new line. They are thought to be separating two tokens.

Type Cast

Still undefined. Which types can be cast each other and what about default type cast behaving like?

Operators:

The following operators are accepted by system:

- Arithmetic Operators:
  - Assignment: `a = b`
  - Addition: `a + b`
  - Subtraction: `a - b`
  - Multiplication: `a * b`
  - Division: `a / b`
  - Modulo: `a % b`
- Comparison Operators:
  - Equal to: `a == b`
  - Not Equal to: `a != b`
  - Greater than: `a > b`
  - Less than: `a < b`
  - Not Greater than: `a <= b`
  - Not Less than: `a >= b`
- Logical Operators:
  - Logical NOT: `!a`

- Logical AND: `a && b`
    - Logical OR: `a || b`
- Member Operators:
    - Array Index: `a[b]`
    - Member Index: `a.b`: Used in bind.
- 2D Object Shift Operators:
    - Shift Toward, Movement: `a->(X, Y)`
    - Serialized Movement: `a -> A in 10 :: b -> B in 5`
    - Parallelized Movement: `a -> A in 10 ^^ b -> B in 5`
- Other Operators:
    - Function call: `f(a, varain...)`
    - End of Line or Sentence: `;`
    - Comma: `,`

Operator Priority

Undefined yet.

Operations on Graphic Objects

Creating Basic Objects

Definition of triangle, circle, etc

The following words are also reserved for 2D object manipulation: `dot, line, triangle, circle` Meanwhile, a new type of time is also defined. Time is basically of int type, but this could only be used to represent time intervals. To define a time variable, use keyword `time`, e.g. `time a`

Also, these four keywords are reserved for functions to initialize 2D objects, respectively:

```
dot(float $XAxis, float $YAxis) ;
line(dot $A, dot $B) ;
triangle(dot $A, dot $B, dot $C);
circle(dot $A, float $Radius) ;
```

Display & Movement:

On a global scale, a variable called MINIMUM_INTERVAL is defined as the minimum time interval for calculating movement. All objects' movement is calculated based on current position and MINIMUM_INTERVAL time later's position. The default MINIMUM_INTERVAL is 0.001 with unit second.

Movement is defined using symbol ->. A screen consists of X by Y pixels and to move a dot to a new position, say to move dot $A to dot $B, simply using $A ->$B.

Resize, Shape Change:

Have not been defined yet. It's also possible to leave this for users to implement their own resize/shaping policy.

Timing

To introduce the concept of speed, movement and morphing must be carried out in a specific given time. the interval is introduced by affixing in <time> after moving or morphing operations. For example:

```
time $t = 10000;
dot $A, $B, $C;
triangle $triA = triangle($A, $B, $C);
dot $D;
circle $cirA = circle($D, 10.0);
$triA => $cirA in $t;
```

means to complete this morphing in 10 seconds (since the default MINIMUM_INTERVAL is set to 0.001 second. If without in <time> affix, the default value for this operation would be 1000 times MINIMUM_INTERVAL.

Binding & Creating Advanced Objects

The real world consists of more complicated shapes. In order to represent them, binding is defined. Generally binding is to use a structure to store all geometric shapes together, and to manipulate on this structure. For example,

```
dot $A = dot(1,0);
dot $B = dot(2,0);
dot $C = dot(2,1);
```

```
triangle $triA = triangle(A, B, C);
dot $D = dot(3,0);
circle $cirA = circle(D, 1.0);
bind $bindA = bind(triA, cirA);
triangle $triB = triangle(A, C, D);
bind $bindB = bind(bindA, triB);
```

A new type of `bind` is introduced using `bind` keyword. the bind type variable can move, morph, just as previously defined. Manipulation to a specific element of this binding is also available. This will be talked in the manipulation in next paragraph.

Modifying Objects

Objects' parameters can be modified. By default, a dot's X-axis value is referenced by `.X`. A triangle/circle's parameter can be modified similarly. For example:

```
dot $A = (1,0); (: This defines a new dot. :)
$A.X = 2; (: This modifies A into (2,0) :)
dot $B = (1,1);
dot $C = (2,2);
triangle $triA = triangle(A, B, C);
$triA.A = (2,3); (: This changes triA's first vertex into
(2,3), while not changing dot A's value. :)
float $R = 10.0;
circle $cirA = circle(A, R);
$cirA.R = 5.0; (: This changes cirA's radius into 5.0. :)
```

Advanced parameters of Objects

Visibility

For visual effect, objects can be defined or modified to be seen or unseen. Objects' color can also be modified. Visibility can be used as `Object_Name.seen()`. To make it unseen, simply call `Object_Name.unseen()`. Objects can be line, circle, triangle, or bind.

Physical Property

When an object consists of triangle, circle or bind of them, physical characters can be assigned to it. To convert such an object to physical object, use `PhyInit()` function to initialize a new object.

The new object will have characters like mass, color, collision behavior, etc

Loading Images

Objects can also be painted by loading a pre-existing image inside. All triangles, circles and binds can load an image on it by calling `Object_Name.loadimg(imgname)`. If `imgname` is too large, crop down from top left. Some common picture format will be supported and this part still needs consideration.

Movement Control

After all the movement has been defined, call the `run()` function to enable all movement. Movement will begin from the first ->.

Movement chains can be defined with symbols `::` and ^^. `::` and ^^ must concatenate movement sentences. `$A :: $B` means $B starts to move once $A finishes. `$A ^^ $B` means $A and $B start simultaneously.

Without `::` or ^^, movement just happens one after another. For example, `$A; $B` is the same as `$A :: $B` but this allows other operations to be done between movement operations (like assignment or modification to objects).

When program runs to `run()` function, it starts to display and all the following movement sentences will be processed, until it encounters a `stop()` function. `stop()` cleans all movement while `pause()` can temporarily pause all movements. By calling `resume()`, previously paused movements will be resumed until they end. After `stop()`, all movement sentences will not be processed until a new `run()` is called.

Sample Code

Finally, Here's a sample hello world program:

```
{
dot[10] %A, %B, %C, %D;
triangle[10] %triA, %triB;
bind[10] %bindA;
int i, j;
for (i = 0; i < 10; i = i + 1) (* Construct object for
each letter. *)
{
  $A[i].X = 20 * i;
  $A[i].Y = 40;
  $B[i].X = 20 * i + 10;
  $B[i].Y = 40;
  $C[i].X = A[i].X;
  $C[i].Y = 60;
  $D[i].X = B[i].X;
  $D[i].Y = 60;
  $triA[i] = triangle($A[i], $B[i], $C[i]);
  $triB[i] = triangle($B[i], $C[i], $D[i]);
  $bindA[i] = bind($triA[i], $triB[i]);
  $bindA[i].seen();
} (* Every letter will be in a 10 x 20 rectangle composed
by two triangles. *)

(* The following bmp files must exist together with the
program executable. *)
$bindA[0].loadimg("h.bmp");
$bindA[1].loadimg("e.bmp");
$bindA[2].loadimg("l.bmp");
$bindA[3].loadimg("l.bmp");
$bindA[4].loadimg("o.bmp");
$bindA[5].loadimg("w.bmp");
$bindA[6].loadimg("o.bmp");
$bindA[7].loadimg("r.bmp");
$bindA[8].loadimg("l.bmp");
$bindA[9].loadimg("d.bmp");

run();
}
```