UO-LISP NEWSLETTER

| April 1985 | Vol. 2 No. 2 |

Please accept our apologies for the lateness of this issue. Your July issue should be on time. We hope you like the new print of the newsletter, as we have just purchased a Hewlett Packard Laserjet printer with which we will be augmenting the print quality of all our documentation.

Version 3.1 Announced

Northwest Computer Algorithms takes pleasure in announcing Version 3.1 of UO-LISP for the IBM-PC computer and compatibles. Version 3.1 is a major enhancement of Version 3.0 featuring larger data spaces and additional functionality. This includes:

- 16k free pairs (8k in V3.0)
- 16k bytes of string space (8k in V3.0)
- Small integers in the range -8192 and 8191
- The Little Meta Translator Writing System
- A compiler option for open coding commonly used functions such as the CAR and CDR composites, small integer arithmetic, and some predicates.
- The MAPOBL function scans the identifier table applying a user supplied function to each symbol.

UO-LISP Version 3.1 requires a minimum of 256k bytes of main storage and PC-DOS 1.1 and higher, or MS-DOS 2.0 and later. The system is source code compatible with Version 3.0 and Version 1.16 (the CP/M version).

Simulation Using the Simple Objects Package

In last months news letter, we presented a simple object based programming system. In this issue we add additional functionality to the objects package and demonstrate its use with a simple simulation of a number of moving interacting objects.

From the last issue, recall that there are a number of built-in behaviors: GET!-YOUR, SET!-YOUR and so on. We now add a behavior to destroy an instance of an object. This function simply removes any properties associated with the object system.

```
(METHOD ROOT KILLYOURSELF ()
% Remove all object properties associated with SELF.
  (REMPROP SELF 'SUPERCLASSES)
  (REMPROP SELF 'VARIABLES)
  (REMPROP SELF 'CLASS))
```

During the course of a long simulation hundreds of objects may be created. KILLYOURSELF frees up storage allocated to an object when it is no longer in use. Since KILLYOURSELF resides at the ROOT of the hierarchy, the behavior can remove any object.

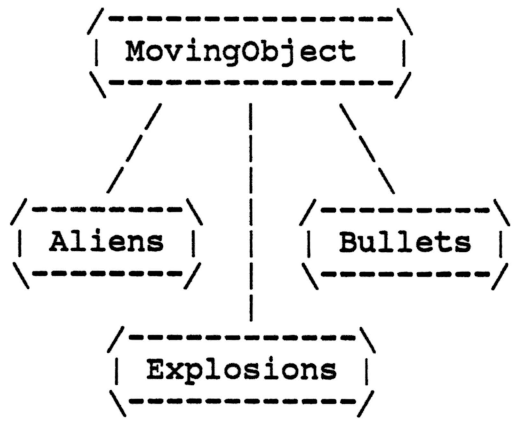
Most simulations require dynamic object creation. The INSTANCE declaration is ill suited to this purpose, hence we introduce an analogous construct, MAKE, for constructing objects at run time. Like INSTANCE, MAKE constructs an object instance but with the added twist that the object's class and name can be computed at run time. The INSTANCE declaration automatically quotes the objects name and class.

```
(DEFMACRO MAKE (cname iname . vars)
% Construct an object instance. This function has the exact
% same arguments as INSTANCE, but cname and iname are
% computed at run time.
  '(PROG (tmp)
     (SETQ tmp ,iname)
     (PUT tmp 'VARIABLES
       (COLLECTVARS (LIST ,cname)
             (FOR (IN x ',vars)
                  (COLLECT (CONS (CAR x) (EVAL (CDR x)))))))
     (PUT tmp 'CLASS (LIST ,cname))
     (RETURN tmp)))
```

You should add these two procedures to the objects package listed in the previous issue.

The following program simulates interactions among some moving objects of different types. So that you can understand

the interactions better, I've called these Aliens and Bullets. Each of these is a type of MovingObject the topmost class is our hierarchy. Basically:



An Explosion is another moving object. Though its behavior is different than that of Aliens and Bullets we attach it to MovingObjects for convenience. This class hierarchy is coded as follows:

```
(CLASS MovingObjects (ROOT)
% MovingObjects have a location and velocity.
   (Xloc . 0)              % Initial X location * 10.
   (Yloc . 0)              % Initial Y location * 10.
   (Xvel . 0)              % Initial X velocity * 10.
   (Yvel . 0) )            % Initial Y velocity * 10.

(CLASS Explosion (MovingObjects)
% An explosion has a location, state (changes with time)
% and picture.
   (Xloc . 0)              % Y Location * 10.
   (Yloc . 0)              % Y Location * 10.
   (State . 0)             % Explosion state.
   (Picture . NIL))        % Explosion picture.

(CLASS Aliens (MovingObjects)
% An alien is a MovingObject with a 4 character picture.
   (Picture . ((-1 0 <) (0 0 *) (1 0 >) (0 1 ^))))

(CLASS Bullets (MovingObjects)
% A bullet is a moving @ sign.
   (Picture . ((0 0 !@))))
```

MovingObjects have only one common procedure. The object erases its previous screen image by writing blanks at all its locations and then gets drawn at a new location. As you can see from

above, each object has a picture associated with it. The picture is a list of 3 element lists. The first two elements of each picture element are x and y coordinates relative to the center of the object at which to display the third element of the list; a character. The CLIPW routine displays a single character at coordinates x and y provided that x and y lie within the screen boundary.

```
% Load the terminal package if not present.
(COND ((NOT (GETD 'CURSOR)) (FLOAD "TERMINAL")))

(GLOBAL '(TERM!-MAXX TERM!-MAXY))

(DE CLIPW (x y c)
% Write a character c at location x y unless it's off
% the screen.
  (IF (NOT (OR (MINUSP x) (GREATERP x TERM!-MAXX)
               (MINUSP y) (GREATERP y TERM!-MAXY))) THEN
      (CURSOR x y)
      (PRIN2 c)))
```

We now provide two routines: one to erase an object by drawing blanks over it, and the other to display it in a new position.

```
(METHOD MovingObjects EraseYourself ()
% Clear an objects screen representation.
  (PROG (lx ly)
     (SETQ lx (QUOTIENT (SEND SELF 'GET!-YOUR 'Xloc) 10))
     (SETQ ly (QUOTIENT (SEND SELF 'GET!-YOUR 'Yloc) 10))
     (FOR (IN pos (SEND SELF 'GET!-YOUR 'Picture))
          (DO (CLIPW (PLUS lx (CAR pos))
                     (PLUS ly (CADR pos)) " "))))))

(METHOD MovngObjects DrawYourself ()
% Draw an object on the screen.
  (PROG (lx ly)
     (SETQ lx (QUOTIENT (SEND SELF 'GET!-YOUR 'Xloc) 10))
     (SETQ ly (QUOTIENT (SEND SELF 'GET!-YOUR 'Yloc) 10))
     (FOR (IN pos (SEND SELF 'GET!-YOUR 'Picture))
          (DO (CLIPW (PLUS lx (CAR pos))
                     (PLUS ly (CADR pos))
                     (CADDR pos))))))
```

The controlling routine, MoveYourself, first erases the object currently on the screen, then updates its position based on its current velocity, and then draws the object at its new location. The key fact to notice is that the MoveYourself routine can be used by all moving objects: in this program, both Aliens and Bullets both use this routine to update their positions.

```
(METHOD MovingObjects MoveYourself ()
% Move an object by clearing it from the screen,
% updating its position, and then redrawing it.
  (SEND SELF 'EraseYourself)
  (SEND SELF 'SET!-YOUR 'Xloc
        (PLUS (SEND SELF 'GET!-YOUR 'Xloc)
              (SEND SELF 'GET!-YOUR 'Xvel)))
  (SEND SELF 'SET!-YOUR 'Yloc
        (PLUS (SEND SELF 'GET!-YOUR 'Yloc)
              (SEND SELF 'GET!-YOUR 'Yvel)))
  (SEND SELF 'DrawYourself))
```

While both Bullets and Aliens move around on the screen, they have slightly different behaviors. A bullet disappears when it goes off the screen. We let the Alien decide what happens when it's hit by a bullet rather than the other way around. To keep track of what bullets are active, we keep a list of active object names for each type of object. Thus, when a bullet goes off the screen, it both deletes its instance data structure (using the KILLYOURSELF behavior) and removes the instance name from the list of active bullets.

```
% Lists of currently active objects.
(GLOBAL '(Bullets Aliens Explosions))

(METHOD Bullets ChangeYourself ()
% If the bullet goes off the screen, erase its object from
% the list of active bullets.
  (PROG (lx ly)
     (SETQ lx (QUOTIENT (SEND SELF 'GET!-YOUR 'Xloc) 10))
     (SETQ ly (QUOTIENT (SEND SELF 'GET!-YOUR 'Yloc) 10))
     (IF (OR (MINUSP lx) (GREATERP lx TERM!-MAXX)
             (MINUSP ly) (GREATERP ly TERM!-MAXY)) THEN
         (SEND SELF 'EraseYourself)
         (SEND SELF 'KILLYOURSELF)
         (SETQ Bullets (DELETE SELF Bullets)) )))
```

In addition to running off the screen and disappearing, Aliens are destroyed by bullets. The collission results in an explosion and disappearance of both objects. In the following code segment, if the alien being examined is sufficiently close to any bullet on the screen, it destroys both objects and signals that an explosion should take place at the point of intersection.

```
(METHOD Aliens ChangeYourself ()
% If an alien runs off the screen, then remove it. If it
% runs into a bullet then start an explosion.
  (PROG (lx ly blowup)
     (SETQ lx (QUOTIENT (SEND SELF 'GET!-YOUR 'Xloc) 10))
     (SETQ ly (QUOTIENT (SEND SELF 'GET!-YOUR 'Yloc) 10))
     (IF (OR (MINUSP lx) (GREATERP lx TERM!-MAXX)
             (MINUSP ly) (GREATERP ly TERM!-MAXY)) THEN
          (SEND SELF 'EraseYourself)
          (SEND SELF 'KILLYOURSELF)
          (RETURN (SETQ Aliens (DELETE SELF Aliens))))
     (FOR (IN b Bullets)
          (WHEN
            (AND
             (EQUAL lx
               (QUOTIENT (SEND b 'GET!-YOUR 'Xloc) 10))
             (EQUAL ly
               (QUOTIENT (SEND b 'GET!-YOUR 'Yloc) 10))))
          (DO (SEND b 'EraseYourself)
              (SEND b 'KILLYOURSELF)
              (SETQ Bullets (DELETE b Bullets))
              (SETQ blowup T)))
     (IF blowup THEN
          (SEND SELF 'EraseYourself)
          (SETQ Explosions
                (CONS
                  (MAKE 'Explosion (SETQ blowup (GENSYM)))
                  Explosions))
          (SEND blowup 'SET!-YOUR 'Xloc
               (SEND SELF 'GET!-YOUR 'Xloc))
          (SEND blowup 'SET!-YOUR 'Yloc
               (SEND SELF 'GET!-YOUR 'Yloc))
          (SEND SELF 'KILLYOURSELF)
          (SETQ Aliens (DELETE SELF Aliens))) ))
```

An explosion is scheduled by the Alien ChangeYourself behavior. An explosion is a sequence of two different pictures that are done during each simulation step. The object variable State associated with each explosion tells which of the two pictures to display and when to terminate the explosion.

```
(METHOD Explosion NextState ()
% Change an explosion to the next state.
  (PROG (state)
     (SETQ state (SEND SELF 'GET!-YOUR 'State))
     (IF (ZEROP state) THEN
        (SEND SELF 'SET!-YOUR 'Picture
           '((-1 0 -) (-1 1 !.) (0 1 |) (1 1 !.) (1 0 -)
             (1 -1 !.) (0 -1 |) (-1 -1 !.)))
        (SEND SELF 'SET!-YOUR 'State 1)
        (SEND SELF 'DrawYourself)
      ELSEIF (ONEP state) THEN
        (SEND SELF 'EraseYourself)
        (SEND SELF 'SET!-YOUR 'Picture
           '((-2 0 -) (0 2 |) (2 0 -) (0 -2 |)))
        (SEND SELF 'SET!-YOUR 'State 2)
        (SEND SELF 'DrawYourself)
      ELSEIF (EQUAL state 2) THEN
        (SETQ Explosions (DELETE SELF Explosions))
        (SEND SELF 'EraseYourself)
        (SEND SELF 'KILLYOURSELF))))
```

The top level driver completes the simulation. It clears the screen and runs the simulation until no more objects are active and then stops.

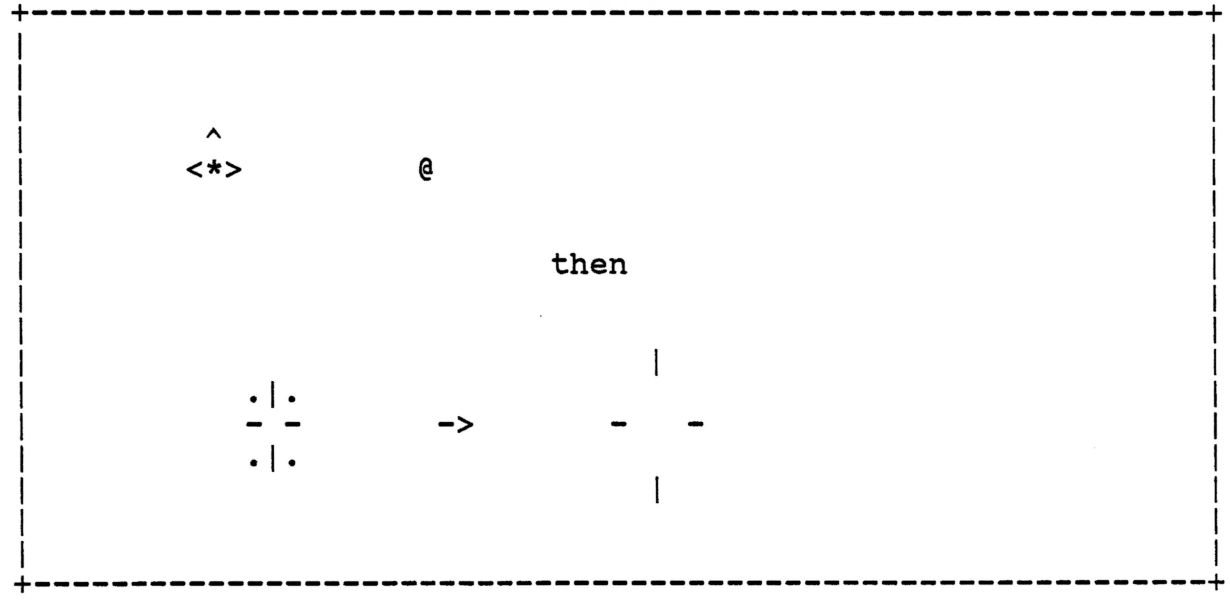
```
(DE RunTheScreen ()
% Run the screen until no more objects are visible.
  (LINELENGTH 0)
  (CLEAR)
  (WHILE (OR Aliens Bullets Explosions)
         (DO (FOR (IN o (APPEND Bullets Aliens))
                  (DO (SEND o 'MoveYourself)
                      (SEND o 'ChangeYourself)))
             (FOR (IN o Explosions)
                  (DO (SEND o 'NextState))))))
```

To run the simulation we create an instance of an alien and a bullet headed at each other. We put the names of these instances on the appropriate lists so the top level loop changes their states at the appropriate times.

```
% Put two objects on the screen and let them go at it.
(SETQ Aliens
  (LIST (INSTANCE Aliens A1
            (Xloc . 300)
            (Yloc . 120)
            (Xvel . 5)
            (Yvel . 0))))
(SETQ Bullets
  (LIST (INSTANCE Bullets B1
            (Xloc . 400)
            (Yloc . 120)
            (Xvel . -6)
            (Yvel . 0))))

(RunTheScreen)
```

The initial screen configuration should resemble the following. The two objects "rush" at each other and disappear in an explosion, only, if you haven't compiled the program, they won't be in any great hurry.



Finally, we demonstrate a more complex example, one with four initial objects aimed at each other but with anhilation scheduled at different times.

```
% Now Put 4 objects on the screen.
(SETQ Aliens (LIST
  (INSTANCE Aliens A1
     (Xloc . 100) (Yloc . 100) (Xvel . 5) (Yvel . 0))
  (INSTANCE Aliens A2
     (Xloc . 200) (Yloc . 200) (Xvel . 6) (Yvel . -6))))
(SETQ Bullets (LIST
  (INSTANCE Bullets B1
     (Xloc . 205) (Yloc . 100) (Xvel . -5) (Yvel . 0))
  (INSTANCE Bullets B2
     (Xloc . 250) (Yloc . 150) (Xvel . -5) (Yvel . 5))))
(RunTheScreen)
```

Institute of Artificial Intelligence

The Institute of Artificial Intelligence is sponsoring a summer training program for workers in the field of Artificial Intelligence. Their brochure states:

> "The Institute of Artificial Intelligence is a permanent, fully independent repository of AI experience, learning and research. Its primary goal is in-depth education and training of functional AI practitioners, such as knowledge engineers, project managers, and AI system programmers. The Institute is affiliated with Harvey Mudd College, the prestigious engineering and science school of the Claremont Colleges."

The Institute can be reached at (213)-201-0106 or

The Institute of Artificial Intelligence
1888 Century Park East, Suite 1207
Los Angeles, California
90067-1716

Classes commence June 24, 1985.

UO-LISP Version Number Changes

Northwest Computer Algorithms has renumbered the versions of UO-LISP to simplify ordering and communication with our users. UO-LISP Version 1 is for the older TRS-80 systems, Version 2 for the Z80 CP/M systems, and Version 3 for the 8086 family. Each version is futher specified by a release number, and, for the Version 3 group, still further by a modification number. The following table gives the current versions and numbers:

Version Name	Current Release	CPU Family	Operating System(s)
V1	V1.5B	Z80	TRSDOS
V2	V2.16	Z80	CP/M 2.2
V3	V3.0.03	8086	PC-DOS, MS-DOS

The old version CP/M system V1.16 is now called V2.16. NOTE: The (BLS.1) style configuration is no longer sold. However, purchasers of this configuration can buy the new Version 2 at less than the total price of the add-ons of the previous system.

New Product Configurations

To streamline mail-order distribution, we have simplified UO-LISP packaging and ordering. The following package pricing applies as of July 1, 1985:

UO-LISP V1	$80.00
Little Meta V1	$40.00
UO-LISP V2	$125.00
Learn Lisp V2	$85.00
Little Meta V2	$80.00
UO-LISP V2 with Lisp Tutorial Support	$160.00
UO-LISP V3	$150.00
Learn Lisp V3	$85.00
Little Meta V3	$80.00
UO-LISP V3 with Lisp Tutorial Support	$185.00
Reference Manual V3	$30.00
Reference Manual V2	$30.00
Reference Manual V1	$20.00
Tutorial Guide	$15.00
Newsletter 1 Year	$12.00
Back issues	$3.00