Appendix

The complete listing of the SMALLTALK code for PYGMALION follows. At the time of this writing, PYGMALION is the largest existing SMALLTALK program.

```
(GET obset ☞ DO)[4][14][6] ← ☞
  (☞vec ← vec[1 to ☞size ← size + 10].
   vec[☞end ← end + 1] ← input).

to icon x y z : name ix iwd iy iht frame CALLER value shape body runcode displayed fetcher sto
**rer container
  (◁'s ⇒
    (:☞x is vector ⇒
      (☞x ← x eval.
       ◁← ⇒ (⇑x ← :) ⇑x eval)
     ◁← ⇒ (⇑x ← :) ⇑x eval)
   ◁has ⇒
    (:x. :y.
     displayed is false ⇒ (⇑false)
     x < ix ⇒ (⇑false)
     y < iy ⇒ (⇑false)
     x > ix + iwd ⇒ (⇑false)
     y > iy + iht ⇒ (⇑false)
     ⇑true)
   ◁run ⇒ (SELF has mx my ⇒ (runcode eval))
   ◁display ⇒
    (◁name ⇒
      ((◁← ⇒ (☞name ← stringify :)).
       displaymode ⇒
        (SELF display erase.
         write SELF name.
         ☞displayed ← ☞name))
     ◁value ⇒
      ((◁← ⇒ (:value)).
       displaymode ⇒
        (☞displayed ← ☞value.
         value is iconstructure ⇒ (value map(xi display name))
         write SELF value))
     ◁shape ⇒
      ((◁← ⇒ (:shape)).
       displaymode ⇒
        ((null shape ⇒ (SELF display name)
         shape eval).
         ☞displayed ← ☞shape))
     ◁body ⇒ (◁← ⇒ (⇑:body) ⇑body)
     ◁erase ⇒
      (displaymode ⇒
        (eq displayed ☞name ⇒
          (frame fclear.
           frame frame 0.
           ☞displayed ← false)
         eq displayed ☞value ⇒
          ((value is iconstructure ⇒ (value map(xi display erase))).
           frame fclear.
           frame frame 0.
           ☞displayed ← false)
         eq displayed ☞shape ⇒
          ((null shape ⇒ () white shape eval black).
           ☞displayed ← ☞value.
           SELF display erase)))
     ◁delete ⇒
      ((displayed ⇒ (SELF display erase)).
       container's value delete CALLER.
       SELF map ☞('s (☞name ← ☞frame ← ☞CALLER ← ☞value ←
                ☞shape ← ☞body ← ☞runcode ← ☞fetcher ←
                ☞storer ← ☞container ← nil)))
     ◁← ⇒ (⇑:displayed) ⇑displayed)
   ◁fetch ⇒ (:x. ⇑fetcher eval)
   ◁store ⇒ (:x. :y. ⇑storer eval)
   ◁eval ⇒ (⇑body eval World)
   ◁map ⇒
    (:x.
     (value is iconstructure ⇒ (value map(xi map x))).
```

```
      apply SELF to x)
    ◁copy ⇒
     (☞x ← icon name ix iwd iy iht nil quick.
      x's runcode ← runcode.
      x's displayed ← displayed.
      x's fetcher ← fetcher.
      x's storer ← storer.
      ◁quick ⇒ (⇑x)
      x's value ← (value is iconstructure ⇒ (value copy) value).
      x's shape ← (shape is iconcontext ⇒ (shape copy) shape).
      x's body  ← (body is iconcontext ⇒ (body copy) body).
      ◁containerless ⇒ (⇑x)
      x's container ← container.
      ⇑x)
    ◁change ⇒
     ((◁position ⇒
          ((◁to ⇒
             (☞x ← - ix - :ix.
              ☞y ← - iy - :iy)
           ☞ix ← ix + :x.
           ☞iy ← iy + :y).
          value is iconstructure ⇒
            (value map (xi change position x y)))
        ◁size ⇒
          (☞iwd ← max 16 ((:x * iwd) / 100) \ 16.
           ☞iht ← max 16 ((:y * iht) / 100) \ 16.
            (value is iconstructure ⇒
               (value map (xi change size x y))).
           ☞ix ← (basex + (x * ix - basex) / 100) \ 16.
           ☞iy ← (basey + (y * iy - basey) / 100) \ 16)).
      frame param
       (☞winx ← ☞frmx ← ix.
        ☞winy ← ☞frmy ← iy.
        ☞winwd ← ☞frmwd ← iwd.
        ☞winht ← ☞frmht ← iht))
   isnew ⇒
    (☞name ← stringify :.
     ☞frame ← dispframe :ix :iwd :iy :iht :.
     ☞CALLER ← SELF.
     ◁quick ⇒ ()
     (displaymode ⇒
      (frame frame - 1.
       frame fclear.
       write SELF name.
       ☞displayed ← ☞name)).
     ☞runcode ← iconrun.
     ☞fetcher ← iconfetch.
     ☞storer ← iconstore.
     change container absolute SELF ix iy.
     ⇑SELF)
   ◁print ⇒ (disp ← '<icon ' name print disp ← '>'))

to iconcontext j oldx oldy oldWorld : i x y quick code World
  (◁eval ⇒
    ((World's value is vector ⇒
       (☞oldx ← World's value.
        World's value ← iconstructure 10.
        for j ← 1 to oldx length - 1 do
          (World's value push Icontable[oldx[j]]))).
     geticon i ← CALLER.
     quick ⇒ (code eval)
     :oldWorld.
     ☞oldx ← CALLER's ix.
     ☞oldy ← CALLER's iy.
     showicon CALLER x y World.
     code eval.
     SELF delete.
     showicon CALLER oldx oldy oldWorld)
   isnew ⇒
```

```
    (☞World ← Icontable['world'] copy quick.
     👁initially ⇒
      (:i. :x. :y. :quick.
       ☞code ←
         (:j is supervector ⇒ (j) supervector initially j).
       World's value ←
         (:j is vector ⇒ (j) iconstructure initially j vector))
     ☞quick ← (👁quick ⇒ (true) false).
     :oldWorld. :i. :code.
     World's value ← iconstructure initially oldWorld's value vector.
     ☞x ← i's ix.
     ☞y ← i's iy.
     ☞i ← geticon x y index)
   👁's ⇒
    (:☞j.
     👁← ⇒ (⇑j ← :) ⇑j eval)
   👁copy ⇒ (⇑iconcontext initially i x y quick code World's value)
   👁is ⇒ (ISIT eval)
   👁print ⇒
    (disp ← 'iconcontext initially ☞'.
     i print sp.
     x print sp.
     y print sp.
     quick print.
     disp ← ' ☞' code print.
     disp ← ' ☞( '.
     World's value map(xi's name print sp).
     disp ← ')')
   👁delete ⇒
    (☞oldx ← World's value.
     for j ← oldx length to 6 by - 1 do
       (eq oldx[j] CALLER ⇒ () oldx[j] display delete)))

to iconout i j x v c
  (:i.
   disp ← '☞iiii ← icon '.
   i's name print.
   disp ← ' basex+'.
   (i's ix - basex) print.
   sp i's iwd print.
   disp ← ' basey+'.
   (i's iy - basey) print.
   sp i's iht print.
   (null i's frame param(buf) ⇒ (disp ← ' nil.' cr cr)
    sp i's frame print disp ← '.' cr cr).
   (☞c ← i's container.
    c has i's ix i's iy ⇒ ()
    disp ← 'change container absolute iiii basex+'.
    (c's ix - basex) print.
    disp ← ' basey+'.
    (c's iy - basey) print.
    disp ← '.' cr cr).
   ☞v ← [☞shape nil ☞body nil ☞runcode iconrun ☞fetcher iconfetch ☞storer iconstore ☞value ni
**l].
   for j ← 1 to v length - 1 by 2 do
    (☞x ← i's (v[j]).
     eq x v[j+1] ⇒ ()
     x is iconstructure ⇒ (x map (iconout xi))
     disp ← 'iiii's ' v[j] print disp ← ' ← '.
     x is atom or x is vector ⇒
       (disp ← '☞' x print disp ← '.' cr cr)
     x print disp ← '.' cr cr).
   Icontable[i's name] ⇒
    (Icontable[i's name] delete))

to iconstructure ii xx : vv
  (👁is ⇒ (ISIT eval)
   👁copy ⇒
    (☞xx ← vv vector.
      for ii ← 1 to xx length - 1 do
        (xx[ii] ← xx[ii] copy).
      ⇑iconstructure initially xx)
    isnew ⇒
     (👁initially ⇒ (☞vv ← supervector initially :)
      ☞vv ← supervector :)
    👁print ⇒
     (disp ← 'iconstructure initially ['.
      vv map(sp xi print).
      disp ← ' ]')
    eq vv ☞xx ← apply vv ⇒ () ⇑xx)

to abort (disp ← ' ...aborted')

to and (⇑:)

to announce x
   (:x.
    within dispframe 192 320 16 16 string 40
      (disp clear.
      disp ← x.
      disp ← '...'))

to blink
   (disp ← 20.
    do 10().
    disp ← 8.
    do 10())

to box x y wd ht
   (penup. goto :x :y. pendn. up. :wd. :ht.
    do 2
     (right 90. go wd.
      right 90. go ht))

to change i j x y basex basey
   (👁position ⇒
     (☞i ← geticon :☞.
      i display erase.
      i's container's value delete i.
      move i to :x :y)
    👁size ⇒
     (☞i ← geticon :☞.
      i's (☞basex ← ix. ☞basey ← iy).
      i display erase.
      i change size :x :y.
      i display shape)
    👁container ⇒
     ((👁absolute ⇒
        (☞i ← :.
         ☞j ← geticon :x+1 :y+1)
       ☞i ← geticon :☞.
       ☞j ← geticon :☞).
      ☞x ← i's container.
      i's container ← j.
      (null x ⇒ ()
       x's value is iconstructure ⇒ (x's value delete i)).
      (eq j's displayed ☞name ⇒ (j's displayed ← ☞value)).
      eq i j ⇒ ()
      j's value is iconstructure ⇒ (j's value push i)
      eq j's value nil ⇒
        (j's value ← iconstructure 2.
         j's value push i)
      sorry 'container is not an iconic structure: ' + stringify j's value.
      SELF display erase))

to constant
   (Mouse store ☞value :)
```

```
to copy i x y
  (move geticon :☞i copy containerless to :x :y)

to create x wd y ht
  (icon 'icon' :x :wd :y :ht nil)

to delete i
  (☞i ← (👁icon ⇒ (:) geticon :☞).
   i store ☞display ☞delete)

to disk fil basex basey baseicon iiii : : showev
  (👁's ⇒ (⇑:☞x eval)
   :fil.
   👁fetch ⇒
    (:basex. :basey.
     filin fil.
     baseicon display shape.
     Icontable[baseicon's name] ← baseicon copy containerless)
   👁store ⇒
    (☞baseicon ← geticon :☞.
     ☞basex ← baseicon's ix.
     ☞basey ← baseicon's iy.
     filout fil ☞(baseicon)))

to drawline x y
  (goto :x + ix :y + iy)

to Eval
  ((geticon :☞) eval)

to extend x attrib
  (:☞attrib.
   ☞x ← CALLER's (attrib).
   cr disp ← 'need more instructions for ' + CALLER's name.
   (x is iconcontext ⇒ (☞x ← x's code)).
   (eq Remember's value x ⇒ (remember resume)
    remember start with x).
   repeat (eq Remember's value x ⇒ (World run) done))

to fetch i
  (☞i ← geticon :☞.
   Mouse store ☞value i fetch :☞)

to getbutton m n
  (☞n ← 0.
   repeat
    (0 = ☞m ← mouse 7 ⇒ (black ⇑ n)
     ☞mousex ← mouse 8.
     ☞mousey ← mouse 9.
     n = m ⇒ ()
     ☞n ← m.
     (m > 3 ⇒
       (☞m ← m - 4.
        Mouse's (black box ix iy + 12 iwd - 1 17))
      Mouse's (white box ix iy + 12 iwd - 1 17)).
     (m > 1 ⇒
       (☞m ← m - 2.
        Mouse's (black box ix iy + 68 iwd - 1 17))
      Mouse's (white box ix iy + 68 iwd - 1 17)).
     (m > 0 ⇒
       (Mouse's (black box ix iy + 40 iwd - 1 17))
      Mouse's (white box ix iy + 40 iwd - 1 17))))

to geticon x y v i
  (:x is vector ⇒
    (☞i ← World.
     👁← ⇒
      (:v.
       for y ← 1 to x length - 2 do (☞i ← i's value[x[y]]).
```

```
        ⇑i's value[x[x length - 1]] ← v)
      for y ← 1 to x length - 1 do (☞i ← i's value[x[y]]).
      ⇑i)
    :y.
    ⊲top ⇒
     (⊲index ⇒ (⇑[World's value map until (xi has x y) index])
      ⇑World's value map until(xi has x y))
    ⊲index ⇒
     (☞v ← supervector 5.
     getindex x y World's value.
      ⇑v vector)
    ☞i ← get1 x y World's value ⇒ (⇑i) ⇑World)

to getindex x y z i xi
   (:x. :y.
    for i ← 1 to :z length do
     (☞xi ← z[i].
      xi's displayed is false ⇒ (⇑false)
      v ← i.
      (xi's value is iconstructure ⇒
        (getindex x y xi's value ⇒ (⇑true))).
      xi has x y ⇒ (⇑true) v pop).
    ⇑false)

to get1 x y z i j xi
   (:x. :y.
    for i ← 1 to :z length do
     (☞xi ← z[i].
      xi's displayed is false ⇒ (⇑false)
      (xi's value is iconstructure ⇒
        (☞j ← get1 x y xi's value ⇒ (⇑j))).
      xi has x y ⇒ (⇑xi)).
    ⇑false)

to IF i x y
   (Mouse store ☞value 'if'.
    ☞i ← memory fetch :x :y.
    :☞x.
    :☞y.
    i's value[2]'s body ← x.
    i's value[3]'s body ← y.
    ⇑i)

to init
   (PUT USER ☞DO ☞(World run).
    ☞disp ← Smalltalk's frame.
    disp clear.
    World's frame fclear.
    World display shape.)

to makeline x y
   (penup. goto :x :y.
    pendn. goto :x :y)

to max x y
   (:x > :y ⇒ (⇑x) ⇑y)

to memory i x y
   (⊲fetch ⇒
     (:x. :y.
      ☞i ← Icontable[Mouse's value] ⇒ (⇑move i copy to x y)
      sorry '<icon ' + stringify Mouse's value + '> is not in memory')
    ⊲store ⇒
     (☞i ← geticon :☞.
      Icontable[i's name] ← i copy containerless))

to memq x v
   (:x.
    ⇑0 < :v[1 to v length] find x)
```

```
to min x y
  (:x < :y ⇒ (⇑x) ⇑y)

to move i x y
  (:i.
   ◁to.
   change container absolute i :x :y.
   i change position to x y.
   i display shape.
   ⇑i)

to nameout i
  (:i's
    (☞displayed ← ☞name.
     frame frame - 1.
     frame show.
     within dispframe ix iwd iy - 16 16 string 100(disp ← name)))

to neg x
  (:x < 0 ⇒ (⇑[x]) ⇑x)

to opcode op i x y
  (☞i ← icon :☞op :x 176 :y 64 nil.
   icon '' i's ix + 16 48 i's iy + 16 32 nil.
   icon '' i's ix + 112 48 i's iy + 16 32 nil.
   i display ← ☞shape.
   i's shape ← ☞
    (SELF display name.
     value[1] display value.
     value[2] display value).
   ☞x ← ☞(SELF display value ←
     Mouse store ☞value value[1]'s value ☺ value[2]'s value).
   x[15] ← op.
   i's body ← x[1 to x length].
   ⇑i)

to opcode1 op i j x y
  (:☞op.
   ☞i ← icon '' :x 112 :y 64 nil.
   ☞j ← geticon x y index.
   [☞text, op, j, 16, 24] eval.
   icon '' i's ix + 48 48 i's iy + 16 32 nil.
   i display ← ☞shape.
   ☞x ← ☞
    (text name ☺ 16 24.
     value[1] display value).
   x[3] ← j.
   i's shape ← x[1 to x length].
   ☞x ← ☞(SELF display value ←
     Mouse store ☞value © value[1]'s value).
   x[9] ← op.
   i's body ← x[1 to x length].
   ⇑i)

to or
  (:. ⇑true)

to plot
  ((null GET xplot ☞DO ⇒ (filin 'xplot')).
   disp fclear.
   xplot :.
   disp show)

to refresh i
  (☞i ← geticon :☞.
   i display erase.
   i display shape)
```

```
to remember i x CALLER
  (◁start ⇒
    (announce 'remembering'.
     ☞remembermode ← true.
     ◁with ⇒
      (Remember's value ← :x.
       Remember's frame clear.
       within Remember's frame
         (for i ← 1 to x length - 1 do (cr x[i] print)))
     :☞x.
    Remember's value ← supervector initially[[☞extend x]].
    Remember's frame clear)
   ◁stop ⇒
    (remembermode is false ⇒ ()
    Remember's value[end] ← nil.
    announce 'stopped remembering'.
    ☞remembermode ← false.
    Remember's frame clear)
   ◁suspend ⇒ (remembermode ⇒
      (announce 'temporarily stopped remembering'.
       ☞remembermode ← false))
   ◁resume ⇒
    (Remember's value ⇒
      (announce 'remembering'.
       ☞remembermode ← true))
   :x.(remembermode ⇒
      (☞i ← Remember's value pop.
       Remember's value push x.
       Remember's value push i.
       within Remember's frame(cr x print))).
   ◁doit ⇒ (⇑x eval) ⇑x)

to REPEAT i x y
  (Mouse store ☞value 'repeat'.
   ☞i ← memory fetch :x :y.
   :☞x.
   i's value[1]'s body ← x.
   ⇑i)

to setmouse x y z
  (:x. :y. :z.
   within Mouse's frame
    (disp clear.
     cr disp ← x cr.
     cr disp ← y cr.
     cr disp ← z))

to Show i
  (☞i ← geticon :☞.
   i store ☞display :☞)

to showicon i x y
  (:i change position to :x :y.
   within dispframe 112 400 32 432 nil(disp fclear).
   :i's value map from 6(xi display shape))

to sorry
  (cr disp ← 'sorry, ' disp ← :.
   cr disp ← 'last operation aborted'.
   cr disp ← 'read-eval-print loop -- type done to proceed:'.
   ev)

to startline x y
  (penup. goto :x + ix :y + iy. pendn)

to store i x
  (☞i ← geticon :☞.
   i store :☞x Mouse's value)
```

```
to stringify x disp
  (:x is string ⇒ (⇑x)
  ☞disp ← superstring 10.
  x print.
  ⇑disp string)

to superstring xx : vec end
  (◁← ⇒
    (:xx is string ⇒
      (☞vec ← vec[1 to end + xx length].
       vec[end + 1 to ☞end ← end + xx length] ← xx.
       ⇑xx)
     vec length < ☞end ← end + 1 ⇒
      (☞vec ← vec[1 to 2 * end - 1].
       ⇑vec[end] ← xx)
     ⇑vec[end] ← xx)
   ◁string ⇒ (⇑vec[1 to end])
   isnew ⇒
    (☞vec ← string :.
     ☞end ← 0))

to supervector ii xx xi : vec end
  (◁← ⇒
    (vec length > ☞end ← end + 1 ⇒ (⇑vec[end] ← :)
     ☞vec ← vec[1 to 2 * end].
     ⇑vec[end] ← :)
   ◁map ⇒
    (◁until ⇒
      (:☞xx.
       for ii ← 1 to end do
        (☞xi ← vec[ii].
         xx eval ⇒ (◁index ⇒ (⇑ii) ⇑xi)).
       ◁index ⇒ (⇑0) ⇑false)
     ☞xi ← (◁from ⇒ (:) 1).
     :☞xx.
     for ii ← xi to end do
      (☞xi ← vec[ii].
       xx eval))
   ◁push ⇒ (⇑SELF ← :)
   ◁pop ⇒
    (end = 0 ⇒ (⇑nil)
     ☞xx ← vec[end].
     vec[end] ← nil.
     ☞end ← end - 1.
     ⇑xx)
   ◁length ⇒ (⇑end)
   ◁vector ⇒ (⇑vec[1 to end + 1])
   ◁eval ⇒ (⇑vec eval)
   ◁delete ⇒
    (☞ii ← vec[1 to end] find :.
     ii = 0 ⇒ ()
     vec[ii to end - 1] ← vec[ii + 1 to end].
     vec[end] ← nil.
     ☞end ← end - 1)
   ◁is ⇒ (ISIT eval)
   ◁print ⇒ (vec[1 to end + 1] print)
   isnew ⇒
    (◁initially ⇒
      (☞end ← :vec length - 1.
       ⇑SELF)
     ☞vec ← vector :.
     ☞end ← 0.
     ⇑SELF)
   eq vec ☞xx ← apply vec ⇒ () ⇑xx)

to table i x : names values
  (◁[ ⇒
    (:x. ◁].
     ☞i ← names map until (x = xi) index.
```

```
      ◁← ⇒
        (i = 0 ⇒
          (names ← x.
           ⇑values ← :)
         :x.
         ⇑values[i] ← x)
      i = 0 ⇒ (⇑false)
      ◁delete ⇒
    (names delete names[i].
     values delete values[i])
      ⇑values[i])
    isnew ⇒
     (☞names ← supervector :i.
      ☞values ← supervector i)
    ◁print ⇒ (names print))

to text s i x y
   (:s.
    ☞i ← geticon :☞.
    ☞x ← i's ix + :.
    ☞y ← i's iy + :.
    within dispframe x 256 y 256 string 100(disp ← s))

to waitmouse n x y z
   (setmouse :x :y :z.
    ☞topbutton ← ☞midbutton ← ☞botbutton ← false.
    repeat
     (0 = mouse 7 ⇒ ()
      ☞n ← getbutton.
       n = 3 ⇒ (Mouse's frame show. opplot) done).
    Mouse's frame clear.
    n = 4 ⇒ (☞topbutton ← true)
    n = 1 ⇒ (eq y dashes ⇒ (abort) ☞midbutton ← true)
    n = 2 ⇒ (eq z dashes ⇒ (abort) ☞botbutton ← true)
    abort)

to within disp
   (:disp.
    ⇑(:☞) eval)

to write i v x wd y ht
   (:i.
    ☞v ← stringify :.
    ☞wd ← min i's iwd 8 * v length.
    ☞ht ← min i's iht 16 *
     (1 + (v length - 1) / i's iwd / 8).
    ☞x ← i's ix + 4 + (i's iwd - wd) / 2.
    ☞y ← i's iy + 2 + (i's iht - ht) / 2.
    i's frame frame - 1.
    i's frame fclear.
    within dispframe x wd y ht string 100(disp ← v))

to [ v i
   (☞v ← vector 10.
    ☞i ← 0.
    repeat
     (◁] ⇒ (⇑v[1 to i + 1])
      v[☞i ← i + 1] ← :.
      ◁,.
      i = v length ⇒ (☞v ← v[1 to 2 * v length])))

to < i
   (◁icon ⇒ (☞i ← Icontable[:i]. ◁>. ⇑i))

to op1 op
   (:☞op.
    waitmouse '  position' dashes dashes.
    topbutton ⇒
     (remember[☞opcode1, op, mousex \ 16, mousey \ 16] doit.))
```

```
to op2 op
  (:☞op.
   waitmouse '  position' dashes dashes.
   topbutton ⇒
    (remember[☞opcode, op, mousex \ 16, mousey \ 16] doit.))

to opbody i
  (waitmouse ' define body' ' fetch body' ' store body'.
   topbutton ⇒
    (☞i ← geticon mousex mousey.
     sp disp ← i's name.
     i's body is iconcontext ⇒
       (remember start with i's body's code)
     remember start body.
     i's body ← iconcontext World i Remember's value.
     Icontable[i's name] ⇒
       (Icontable[i's name]'s body ← i's body copy)
     cr disp ← 'do you want ' + i's name + ' saved in memory? (y or n)'.
     memq read[1] ☞(y Y) ⇒
       (Icontable[i's name] ← i copy containerless))
   midbutton ⇒
    (remember[☞fetch, geticon mousex mousey index, ☞body] doit)
   botbutton ⇒
    (remember[☞store, geticon mousex mousey index, ☞body] doit))

to opchange i j
  (waitmouse ' change pos' ' change size' ' change cont'.
   topbutton ⇒
    (☞i ← geticon mousex mousey.
     eq i World ⇒ (sorry 'cant change position of world')
     ☞j ← geticon mousex mousey index.
     waitmouse ' upper left' dashes dashes.
     topbutton ⇒
       (remember[☞change, ☞position, j, mousex\16, mousey\16] doit))
   midbutton ⇒
    (☞i ← geticon mousex mousey.
     eq i World ⇒ (sorry 'cant change size of world')
     ☞j ← geticon mousex mousey index.
     waitmouse ' lower right' dashes dashes.
     topbutton ⇒
       (remember[☞change, ☞size, j,
           (100 * mousex - i's ix) / i's iwd,
           (100 * mousey - i's iy) / i's iht] doit))
   botbutton ⇒
    (☞i ← geticon mousex mousey index.
     waitmouse ' container' dashes dashes.
     topbutton ⇒
       (remember[☞change, ☞container, i, geticon mousex mousey index] doit)))

to opconstant
  (cr disp ← 'value? '.
   remember[☞constant, read eval] doit)

to opcopy i
  (waitmouse ' copy icon' dashes dashes.
   topbutton ⇒
    (☞i ← geticon mousex mousey index.
     waitmouse '  position' dashes dashes.
     topbutton ⇒
       (remember[☞copy, i, mousex\16, mousey\16] doit)))

to opcreate n started x wd y ht
  (☞started ← false.
   setmouse ' set point' ' del point' '    quit'.
   repeat
    (☞n ← getbutton.
     n = 4 ⇒
       (Mouse's (white box ix iy + 12 iwd - 1 17 black).
        started is false ⇒
         (☞x ← mousex \ 16.
          ☞y ← mousey \ 16.
          ☞wd ← ☞ht ← 0.
          ☞started ← true).
        mem 65 ← 1 'XOR ink'.
        box x y wd ht.
        mem 65 ← 0.
        ☞wd ← (mousex \ 16) - x.
        ☞ht ← (mousey \ 16) - y.
        (wd < 0 ⇒
          (☞wd ← - wd.
           ☞x ← mousex \ 16)).
        (ht < 0 ⇒
          (☞ht ← - ht.
           ☞y ← mousey \ 16)).
        remember[☞create x wd y ht] doit.
        ☞started ← false)
      n = 1 ⇒
       (Mouse's (white box ix iy + 40 iwd - 1 17 black).
        started ⇒
         (mem 65 ← 1.
          box x y wd ht.
          mem 65 ← 0.
          ☞wd ← ☞ht ← 0.
          ☞started ← false))
      n = 2 ⇒
       (done with Mouse's frame clear)
      started ⇒
       (mem 65 ← 1.
        box x y wd ht.
        box x y ☞wd ← (mx \ 16) - x ☞ht ← (my \ 16) - y.
        mem 65 ← 0)))

to opdelete
  (waitmouse ' delete icon' dashes dashes.
   topbutton ⇒
    (remember[☞delete, geticon mousex mousey index] doit))

to opdisk i j fil
  (waitmouse '   fetch' '   store' dashes.
   topbutton ⇒
    (cr disp ← 'please type a file name: '.
     ☞fil ← read eval.
     fil is string ⇒
      (remember[☞disk, fil, ☞fetch, mousex \ 16, mousey \ 16] doit)
     abort)
   midbutton ⇒
    (☞i ← geticon mousex mousey.
     ☞j ← geticon mousex mousey index.
      (i's name = 'icon' ⇒
        (cr disp ← 'please type a name (a string) for the icon: '.
         i display name ← read eval)).
     cr disp ← 'please type a file name: '.
     ☞fil ← read eval.
     fil is string ⇒
      (sp disp ← i's name.
       remember [☞disk, fil, ☞store, j] doit)
     abort))

to opdisplay
  (waitmouse '   on' '   off' dashes.
   topbutton ⇒ (remember ☞(☞displaymode ← true) doit)
   midbutton ⇒ (remember ☞(☞displaymode ← false) doit))

to opdraw i n started xstart ystart xstop ystop
  (☞started ← false.
   waitmouse ' relative to' dashes dashes.
   topbutton is false ⇒ ()
   ☞i ← geticon mousex mousey.
   setmouse ' start line' ' stop line' '    quit'.
   repeat
     (☞n ← getbutton.
      n = 4 ⇒
        (Mouse's (white box ix iy + 12 iwd - 1 17 black).
         started ⇒
           (remember[☞drawline, xstop - i's ix, neg ystop - i's iy].
            ☞xstart ← xstop.
            ☞ystart ← ystop)
         ☞xstart ← ☞xstop ← mousex.
         ☞ystart ← ☞ystop ← mousey.
         remember[☞startline, xstart - i's ix, neg ystart - i's iy].
         ☞started ← true)
      n = 1 ⇒
        (Mouse's (white box ix iy + 40 iwd - 1 17 black).
         started ⇒
           (remember[☞drawline, xstop - i's ix, neg ystop - i's iy].
            ☞started ← false))
      n = 2 ⇒
        ((started ⇒
           (mem 65 ← 1 'XOR ink'.
            makeline xstart ystart xstop ystop.
            mem 65 ← 0)).
         done with Mouse's frame clear)
      started ⇒
        (mem 65 ← 1.
         makeline xstart ystart xstop ystop.
         makeline xstart ystart ☞xstop ← mx ☞ystop ← my.
         mem 65 ← 0)))

to opeval
  (waitmouse ' eval icon' dashes dashes.
   topbutton ⇒
     (remember[☞Eval, geticon mousex mousey index] doit))

to opexit
  (PUT USER ☞DO sysUSER.
   ☞disp ← sysDISP.
   remember suspend.
   World's frame fclear.
   disp frame - 1.)

to opif
  (waitmouse '  position' dashes dashes.
   topbutton ⇒
     (remember[☞IF, mousex \ 16, mousey \ 16,
        supervector initially ☞((extend body)),
        supervector initially ☞((extend body))] doit))

to opmemory i j
  (waitmouse '    fetch' '    store' ' fetch mouse'.
   topbutton ⇒
     (cr disp ← 'please type a name (a string) for the icon: '.
      remember[☞constant, read eval] doit.
      remember[☞memory, ☞fetch, mousex \ 16, mousey \ 16] doit)
   midbutton ⇒
     (☞i ← geticon mousex mousey.
      ☞j ← geticon mousex mousey index.
       (i's name = 'icon' ⇒
         (cr disp ← 'please type a name (a string) for the icon: '.
          i store ☞name read eval)).
       (Icontable[i's name] ⇒
         (cr i print disp ← ' is already in memory'.
          cr disp ← 'type y to replace: '.
          memq read[1] ☞(y Y) ⇒ () ⇑abort)).
      sp disp ← i's name.
      remember [☞memory, ☞store, j] doit)
   botbutton ⇒
```

```
    (remember[☞memory, ☞fetch, mousex \ 16, mousey \ 16] doit))

to opname (repeat
    (waitmouse ' fetch name' ' store name' '   quit'.
     topbutton ⇒
      (remember[☞fetch, geticon mousex mousey index, ☞name] doit)
     midbutton ⇒
      (remember[☞store, geticon mousex mousey index, ☞name] doit)
     botbutton ⇒ (done)))

to opnext ()

to opplot fil
  (cr disp ← 'please type a file name: '.
   ☞fil ← read eval.
   fil is string ⇒ (remember[☞plot, fil] doit) abort)

to oprefresh (repeat
    (waitmouse 'refresh icon' dashes '   quit'.
     topbutton ⇒
      (remember[☞refresh, geticon mousex mousey index] doit)
     botbutton ⇒ (done)))

to opremember
  (waitmouse '    stop' '   suspend' '   resume'.
   topbutton ⇒ (remember stop)
   midbutton ⇒ (remember suspend)
   botbutton ⇒ (remember resume))

to oprepeat
  (waitmouse '   position' dashes dashes.
   topbutton ⇒
    (remember[☞REPEAT, mousex \ 16, mousey \ 16,
       supervector initially ☞((extend body))] doit))

to opshape i
  (waitmouse 'define shape' ' fetch shape' ' store shape'.
   topbutton ⇒
    (☞i ← geticon mousex mousey.
     sp disp ← i's name.
     i's shape is iconcontext ⇒
       (remember start with i's shape's code)
     remember start shape.
     i's shape ← iconcontext quick World i Remember's value.
     Icontable[i's name] ⇒
       (Icontable[i's name]'s shape ← i's shape copy))
   midbutton ⇒
    (remember[☞fetch, geticon mousex mousey index, ☞shape] doit)
   botbutton ⇒
    (remember[☞store, geticon mousex mousey index, ☞shape] doit))

to opshow i j
  (waitmouse ' show name' ' show value' ' show shape'.
   topbutton ⇒
    (☞i ← geticon mousex mousey.
     ☞j ← geticon mousex mousey index.
      (eq i display ☞name ⇒
        (eq i's container World ⇒ ()
         j[j length - 1] ← nil.
         ☞j ← j[1 to j length - 1])).
     remember[☞Show, j, ☞name] doit)
   midbutton ⇒
    (remember[☞Show, geticon mousex mousey index, ☞value] doit)
   botbutton ⇒
    (☞j ← geticon mousex mousey index.
     remember[☞Show, j, ☞shape] doit))

to optext i j s
  (waitmouse ' relative to' dashes dashes.
   topbutton ⇒
    (☞i ← geticon mousex mousey.
     ☞j ← geticon mousex mousey index.
     cr disp ← 'please type the text (a string) to be displayed: '.
     ☞s ← stringify read eval.
     waitmouse '  position' dashes dashes.
     topbutton ⇒
       (remember[☞text, s, j, mousex - i's ix, mousey - i's iy + 16] doit)))

to optrace
  (waitmouse ' trace icon' 'untrace icon' dashes.
   topbutton ⇒
    (remember[☞trace, geticon mousex mousey index, ☞on] doit)
   midbutton ⇒
    (remember[☞trace, geticon mousex mousey index, ☞off] doit))

to opvalue n
  (setmouse ' fetch value' ' store value' ' ----------'.
   repeat
    (☞n ← getbutton.
     n = 4 ⇒
      (Mouse's frame clear.
       remember[☞fetch, geticon mousex mousey index, ☞value] doit.
       done)
     n = 1 ⇒
      (Mouse's frame clear.
       remember[☞store, geticon mousex mousey index, ☞value] doit.
       done)
     CALLER has mx my ⇒ ()
     done with Mouse's frame clear))

to initall
  (disk's
    (to showev x
      (disp ← '☞baseicon ← '.
       iconout :x eval)).
    (memq ☞\ GET number ☞DO ⇒ ()
     addto number ☞(◁\ ⇒ (⇑:x * (SELF + x / 2) / x)).
     addto dispframe ☞(◁print ⇒ (buf print)).
     (GET dispframe ☞DO)[59][50] ← 0.
     (GET pshow ☞DO)[10][4] ← 6.
     (GET pshow ☞DO)[22][34] ← ☞(: ☞ # ⇑ [ ◁ ⇒ ⧈ ).
     (GET file ☞DO)[75][24][4] ← ☞
      (dp0 evals filesopen map ☞
        (vec[i] evals ☞(☞dirinst ← nil))).
     ☞sysUSER ← GET USER ☞DO.
     ☞sysDISP ← disp).
   ☞dashes ← ' ----------'.
   ☞run ← ☞run.
   ☞Icontable ← table 10.
   Icontable['world'] ← nil.
   ☞iconrun ← ☞(opvalue).
   ☞iconfetch ← ☞([CALLER ☞'s x] eval).
   ☞iconstore ← ☞
    (eq x ☞display ⇒ ([CALLER ☞display y] eval)
    [CALLER ☞display x ☞← ☞☞ y] eval).
   ☞specialstore ← ☞(sorry 'cant change ' + stringify CALLER).
   ☞displaymode ← true.
   ☞remembermode ← false.
   initworld initmenu initmouse initremember initsmalltalk initicons.
   World's frame clear.
   disp frame -1.
   ☞initworld ← ☞initmenu ← ☞initmouse ← ☞initremember ← ☞initsmalltalk ← ☞initicons ← ☞inita
**ll ← nil)

to initicons i baseicon basex basey
  (☞basex ← 192.
   ☞basey ← 192.
```

```
☞baseicon ← ☞i ← icon 'if' basex+0 192 basey+0 112 nil.
i's shape ← ☞(value[1] display name. value[2] display value. value[3] display value. startline
**66 18. drawline 124 18. drawline 114 10. drawline 114 26. drawline 124 18. startline 66 34.
**drawline 104 77. drawline 92 75. drawline 104 65. drawline 104 77).
i's body ← ☞( value [ 1 ] 's value ⇒ ( value [ 2 ] eval ) value [ 3 ] eval ).

☞i ← icon '?' basex+0 64 basey+0 32 ' '.
i's runcode ← ☞( opvalue ).

☞i ← icon 'true branch' basex+128 64 basey+0 32 ' '.
i's shape ← ☞( SELF display value ).
i's value ← true.

☞i ← icon 'false branch' basex+96 64 basey+80 32 ' '.
i's shape ← ☞( SELF display value ).
i's value ← false.

   Icontable['if'] ← baseicon copy containerless.
   baseicon display delete.

☞baseicon ← ☞i ← icon 'repeat' basex+0 48 basey+0 32 nil.
i's shape ← ☞(CALLER display value. startline 23 33. drawline 23 59. drawline -31 60. drawline
**-31 (-38). drawline 23 (-38). drawline 23 (-2). drawline 13 (-13). drawline 33 (-13). drawli
**ne 23 (-2)).
i's value ← ' loop'.

   Icontable['repeat'] ← baseicon copy containerless.
   baseicon display delete)

to initmenu i
  (☞Menu ← Icontable['menu'] ← icon 'menu' 16 80 32 647 string 300.
   Menu's shape ← ☞(nameout SELF).
   Menu's runcode ← ☞
     (setmouse '    doit' dashes dashes.
      repeat
        (getbutton = 4 ⇒
           (Mouse's frame clear.
            ☞x ← (Menu's frame mfindt mousex mousey) / 2.
            0 < x < menuops length + 1 ⇒
              (sp menunames[x] print.
               ☞operation ← menuops[x].
               eq operation ☞undefined ⇒
                 (done with disp ← ' undefined')
               World's (operation eval).
               disp ← ' ok'.
               done))
         SELF has mx my ⇒ ()
         done with Mouse's frame clear)).
   Menu's fetcher ← iconfetch.
   Menu's storer ← specialstore.
   ☞menunames ← supervector 50.
   ☞menuops ← supervector 50.
   within Menu's frame
     (supervector initially ☞
        ((icons undefined create opcreate change opchange delete opdelete copy opcopy refresh
** oprefresh show opshow name opname value opvalue shape opshape body opbody)
         (opcodes undefined
          + (op2 +) - (op2 -) * (op2 *) / (op2 /)
          = (op2 =) < (op2 <) > (op2 >)
          and (op2 and) or  (op2 or) not (op1 not))
         (control undefined if opif repeat oprepeat done opdone eval opeval return opreturn
**)
         (others undefined memory opmemory disk opdisk next opnext display opdisplay rememb
**er opremember draw opdraw text optext trace optrace constant opconstant plot opplot exit ope
**xit))
      map
       (menunames ← xi[1].
        menuops ← xi[2].
        cr xi[1] print cr.
      for i ← 3 to xi length - 1 by 2 do
        (menunames ← xi[i].
         menuops ← xi[i + 1].
         memq xi[i] ☞(+ * <) ⇒ (sp xi[i] print sp)
         sp xi[i] print cr))))

to initmouse
  (☞Mouse ← Icontable['mouse'] ← icon 'mouse' 400 96 480 96 string 50.
   Mouse's shape ← ☞(nameout SELF).
   Mouse's runcode ← nil.
   Mouse's fetcher ← iconfetch.
   Mouse's storer ← ☞
     (eq x ☞display ⇒ (nil)
      within Mousevalue's frame
        (disp clear.
         Mouse's value ← y print)).
   ☞Mousevalue ← Icontable['mouse value'] ← icon 'mouse value' 112 272 480 32 string 100.
   Mousevalue's value ← supervector initially ☞('' '' '').
   Mousevalue's shape ← Mouse's shape.
   Mousevalue's runcode ← nil.
   Mousevalue's fetcher ← iconfetch.
   Mousevalue's storer ← specialstore)

to initremember
  (☞Remember ← Icontable['remembered'] ← icon 'remembered' 112 272 544 32 string 100.
   Remember's value ← false.
   Remember's shape ← ☞(nameout SELF).
   Remember's runcode ← ☞
     (setmouse '   insert' '   delete' '   scroll'.
      repeat
        (☞x ← getbutton.
   x = 4 ⇒ ('...')
         x = 1 ⇒ ('...')
         x = 2 ⇒ ('...')
         SELF has mx my ⇒ ()
         done with Mouse's frame clear)).
   Remember's fetcher ← iconfetch.
   Remember's storer ← specialstore)

to initsmalltalk
  (☞Smalltalk ← Icontable['smalltalk'] ← icon 'smalltalk' 112 384 608 71 string 300.
   Smalltalk's shape ← ☞(nameout SELF).
   Smalltalk's body ← ☞
     (cr disp ← 'input? '.
      Mouse store ☞value read eval).
   Smalltalk's runcode ← ☞(repeat
        (kbck ⇒ (cr read eval print)
         SELF has mx my ⇒ (blink)
         done)).
   Smalltalk's fetcher ← ☞
     (cr x print disp ← '? '.
      read eval).
   Smalltalk's storer ← ☞(Mouse's value print))

to initworld
  (☞World ← Icontable['world'] ← icon 'world' 0 512 0 680 nil quick.
   World's value ← iconstructure 10.
   World's shape ← ☞(value map(xi display shape)).
   World's runcode ← ☞
     (Mouse's frame clear.
      value map(xi run)).
   World's fetcher ← iconfetch.
   World's storer ← specialstore.
   World's container ← World)

( initall )
```