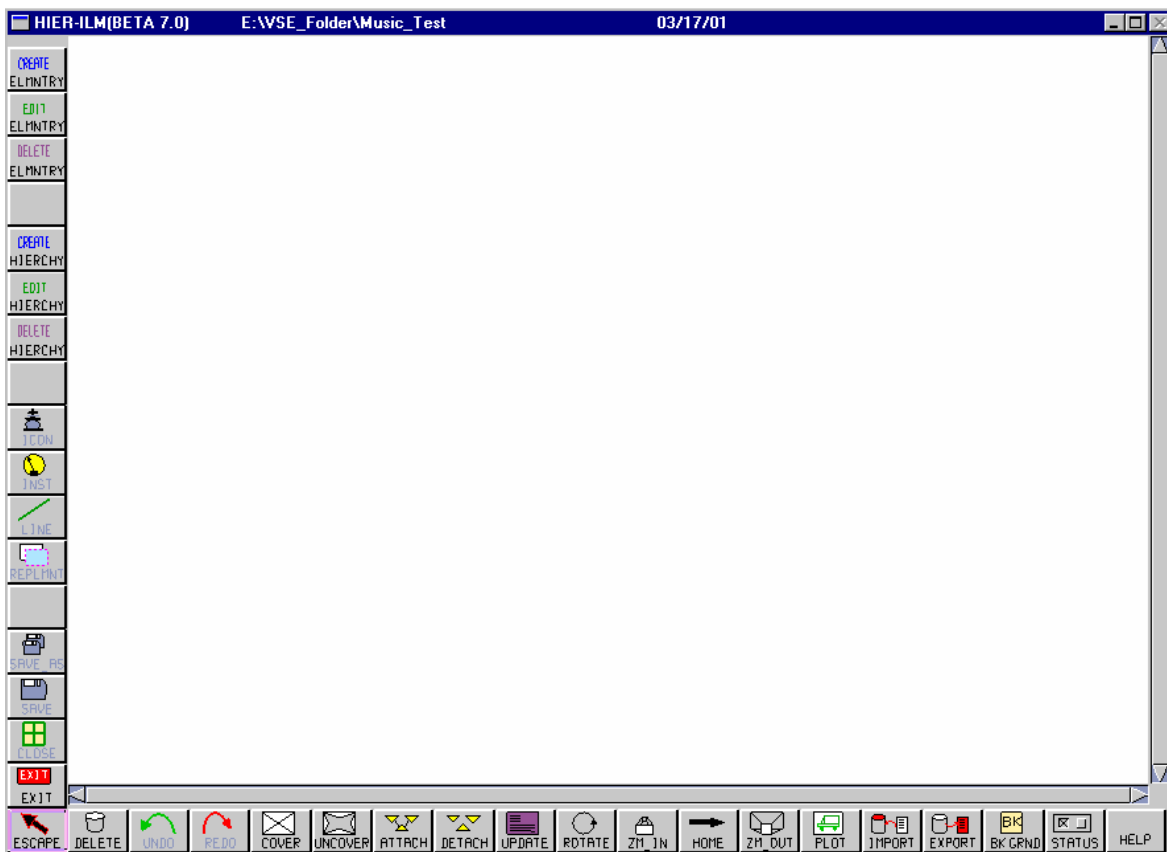


CHAPTER 6: RUN-TIME GRAPHICS - BASICS

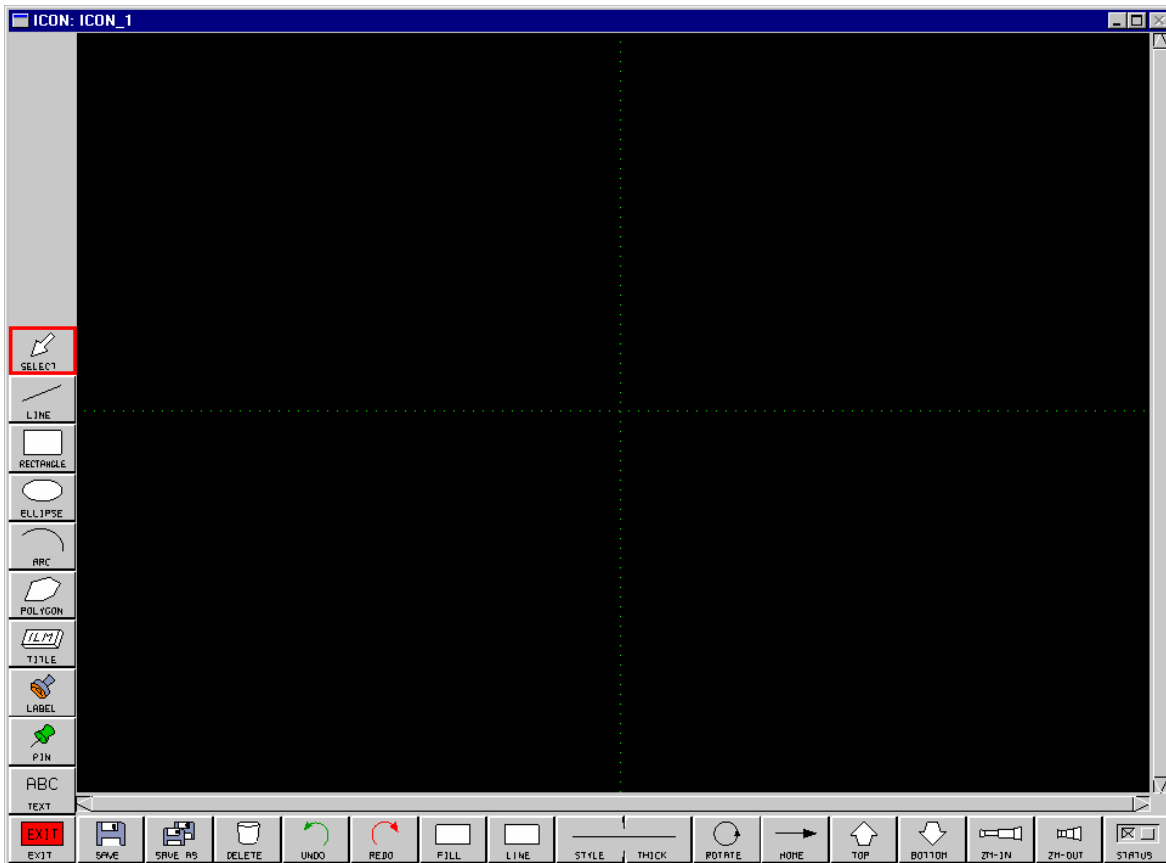
Now that we have some background in VSE, we can discuss the fun part of it - the graphics section. VSE has two main graphical tools - the ILM and the PLM. The ILM is used to create icons that can be used to provide a layer of abstraction, while the PLM is used to create graphical user interfaces (GUIs). In the next two chapters, we will discuss the ILM.

For starters, ILM stands for Icon Library Management. You can put together some cool pictures and make them move. The ILM can be accessed by clicking the ILM button on the lower right of the VSE window. This is what it looks like at first glance:



Very similar interface to that of VSE, isn't it? It's just as easy to get the hang of it as well. Just wait and watch.

The ILM provides us with the means to create our own user-defined icons. These are created within the Basic Drawing Space (BDS). This is brought up by clicking on the first button of the ILM interface that says 'CREATE ELEMENTARY ICON'. It looks something like this:



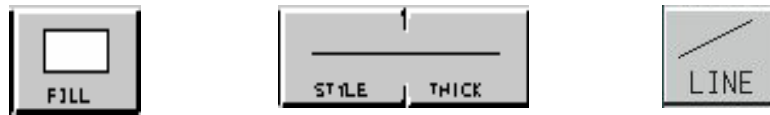
CREATING ICONS

Along the left side of the drawing space, you will notice buttons for drawing shapes like ellipses, rectangles, straight lines, arcs, polygons etc. Experiment with them and see what you can come up with. Tips on using the buttons are displayed at the bottom of the workspace as you click on them.

For example, an arc is drawn by first clicking on the arc button and then drawing the outline of an ellipse on the drawing space. Next, the left mouse button is clicked on a point to the left of the ellipse and a clockwise or counterclockwise path is traced along it till it is met at the opposite end. On releasing the mouse button, the required shape of the ellipse is obtained.

The bottom edge of the workspace is lined with another set of buttons, quite similar to those on the VDE interface. The usual 'SAVE', 'SAVE AS', 'DELETE', 'UNDO' and 'REDO' buttons are present.

In addition to those we've already seen, there are some buttons that deserve special mention:



FILL: It fills the selected object with the chosen color.

STYLE/ THICK: This tool is used to choose a certain style for the selected line e.g. dotted, dashed etc. or to vary the thickness of the selected line.

LINE: It is used to select a particular color for the selected line or the outline of the selected object.

THE SMILEY FACE EXAMPLE

Let us experiment with a few of these options by creating a smiley.

Step 1: Making the Face.

You will need to draw a big circle first. Click on the Ellipse tool and draw a circular shape. Then, click on Fill and select a bright yellow color.

Draw two smaller circles for the eyes. Click on Fill and select the same color as before. Then, click on Line and select black (or any dark color). This will make the outlines of the circles stand out on the face. Next, click on Select. Click on one of the smaller circles and drop it in position on to the bigger circle. Do the same with the other smaller circle.

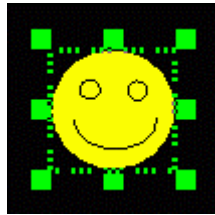


All our smiley needs now is a smile! Go on and create an arc in the shape of a mouth. Once again, choose the Fill and Line options as before and drag the smile onto the face.



Step 2: Making the Face Bigger.

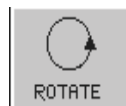
Do you want it to look a little bigger? Click Select and drag a rectangle around the smiley so that it is completely encircled. A bounding rectangle will appear:



Now, just click on any of the little pins that appear to stretch the size of the smiley. After you are satisfied with the shape, click on SAVE AS and name it as FACE1 for use in the next few examples.

Try and create a few more basic shapes. The reason why this interface does not have the immense capabilities of a full-fledged graphics editor like Paint Shop is that the icons are secondary to the task at hand - they are just an abstract representation of the complex modules underneath them - in keeping with our philosophy of separating instructions from data.

There are some other buttons that you can now play with. You can test each of these on the smiley we have just created.



ROTATE: Rotates the selected shape through a certain angle in either the clockwise or anti-clockwise direction. The angle through which it is rotated can be varied through the status bar.

Select the entire icon as discussed above, and click on rotate. The smiley will rotate with a slight angle. This angle can be varied by selecting it in the status bar.



TOP: This tool is used to place a shape on top of another shape.

BOTTOM: This tool is used to position a shape below a previously created shape.

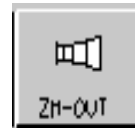
To demonstrate the use of these tools, let's make some further additions to the smiley. Using the Line tool, draw a pipe and drag it to the smiley's mouth.



After this, select only the pipe and click on BOTTOM. You will notice that a part of the pipe has disappeared behind the smiley.



Now, select the pipe again and click on TOP. It will come to the fore again. Quite simply, the TOP and BOTTOM buttons are used to hide and show certain shapes relative to others. This is useful in the creation of complex icons.

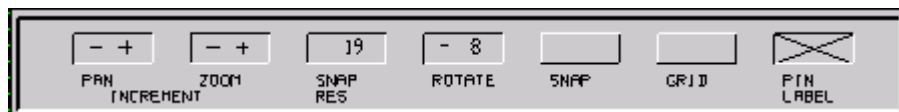


The function of these buttons can again be observed by selecting the smiley and clicking on each button separately. They help in magnifying and diminishing the view respectively.



STATUS: The Status button reveals the status bar, which can be used to vary certain settings.

On clicking the status button, we get the status bar that has many options. Here, we can change the angle of rotation, and the resolution of the grid and zoom-in and zoom - out options.



You can now create a few more basic shapes and experiment with all the buttons that we have learned thus far.

RTG SYNTAX

Now that we know how to create icons, we can move on to the syntax that is used to manipulate them.

There will be a few differences in the code from what we have done so far. For example, remember when we discussed the Control Specification? The format stays pretty much the same, except that in addition to the previous sections, namely - the control section and the module section, we will leave it in the graphics section. And follow it by the statement - activate. This is followed by a list of the icons that we have created. These will be accessed from the icon library and assigned to variables, which are then used in our program.

Let's try and make that a little more lucid. The additional code will be:

```
`GRAPHICS SECTION
  ACTIVATE'

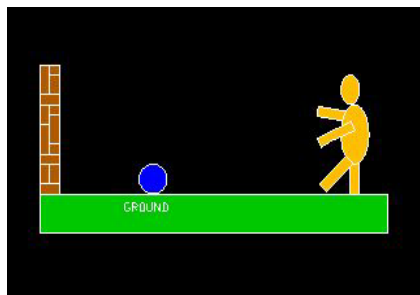
`ICON NAME_ASSIGNED = NAME_ON_CREATION'
```

“NAME_ON_CREATION” is the name you save an icon as after having creating it in the basic drawing space. “NAME_ASSIGNED” is the name of the variable to which the icon is assigned. This variable is then used to manipulate the icon within the program. There can be any number of such declarations – one for each of the icons used in the program.

The two main commands that are used to manipulate icons are INSERT and UPDATE. Now, let's write some code and learn more about them.

THE FOOTBALL EXAMPLE

In this example, we will create a small animation of a man hitting a ball against a brick wall. This is what it will look like:



Step 1: Creating Icons.

You will first need to create 4 different icons in the ILM, namely, the man, the ball, the wall and the ground. These icons will be passed into variables declared in the resource.

Step 2: Creating the Task.

Now, create a task containing one resource and one process. The code for the control specification, resource and process is detailed below followed by an explanation of some of the statements that are different from what we have studied so far.

***Data for the Control Specification

```
CONTROL SECTION
  TITLE, FOOTBALL

GRAPHICS SECTION
  ACTIVATE
  ICON MY_MAN = MAN
  ICON MY_BALL = BALL
  ICON WALL = WALL
  ICON GROUND = GROUND

MODEL SECTION
  GAME_PRO

END
```

In the Control Specification, we assign each icon to an icon variable. Thus, icons MAN, BALL, WALL and GROUND are assigned to the variables 'MY_MAN', 'MY_BALL', 'WALL' and 'GROUND' respectively.

***Data for the Resource: GAME_DATA

```
GAME_DATA
  1 I          INTEGER
  1 X          INTEGER
  1 Y          INTEGER
  1 MY_MAN     ICON
  1 MY_BALL    ICON
  1 WALL       ICON
  1 GROUND     ICON
```

The variables "MY_MAN", "MY_BALL", "WALL" and "GROUND" are declared to be icons, which are now manipulated in the program.

***Data for the Process: GAME_PRO

```
GAME_PRO
  I = 1
  EXECUTE INSERT_ICONS
  EXECUTE GAME UNTIL I GREATER THAN 1000.

INSERT_ICONS
  INSERT GROUND AT 400, 560
  INSERT WALL AT 400, 600
  INSERT MY_MAN AT 700, 600
  INSERT MY_BALL AT 655, 600
  X = 655
  Y = 600.

GAME
  EXECUTE MOVE_MAN
  EXECUTE PLAY
  INCREMENT I.

MOVE_MAN
  UPDATE MY_MAN TO 698, Y.
  UPDATE MY_MAN TO 700, Y.

PLAY
  EXECUTE MOVE_BALL UNTIL X LESS THAN 425
  EXECUTE BACK_BALL UNTIL X GREATER THAN 655

MOVE_BALL
  X = X - 10
  UPDATE MY_BALL TO X, Y.

BACK_BALL
  X = X + 10
  UPDATE MY_BALL TO X, Y.
```

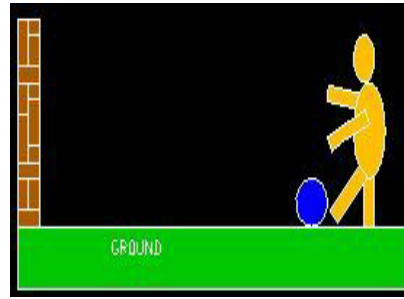
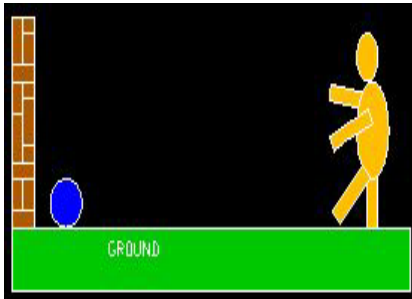
In the process GAME_PRO, we insert the icons at specific positions defined by the x-y coordinates (INSERT_ICONS). To create the moving effect (GAME), we use the UPDATE command, which erases the previous image from the screen and inserts it at the newly specified location.

Let us summarize the working of this program. First, the icons are inserted at various locations of the screen in the “INSERT_ICONS” rule. Next, the rule “GAME” is executed, which call various other rules that create the animation effect. There are different rules for moving the man and moving the ball to the left and then to the right.

Step 3: Running the Program.

Go ahead and PREPARE and RUN the program and see for yourself.

The output during run-time will vary between the following two screenshots:



INPUT PROMPTS

You can also write programs to query the user for input and act on it using user prompts. The syntax for this is:

```
'DISPLAY PROMPT  
ACCEPT INPUT VARIABLE_NAME'
```

For a demonstration of this, let us use our previous smiley icon. Create a simulation containing one resource and one process. Then, use the following code:

***Data for the Control Specification

```
CONTROL SECTION  
    TITLE, FACE  
  
GRAPHICS SECTION  
    ACTIVATE  
  
ICON FACE = FACE1  
  
MODEL SECTION  
    FACE_PROC  
  
END
```

***Data for the Resource: GAME_DATA

```
FACE    ICON  
I        INTEGER  
J        INTEGER  
POS     INTEGER
```

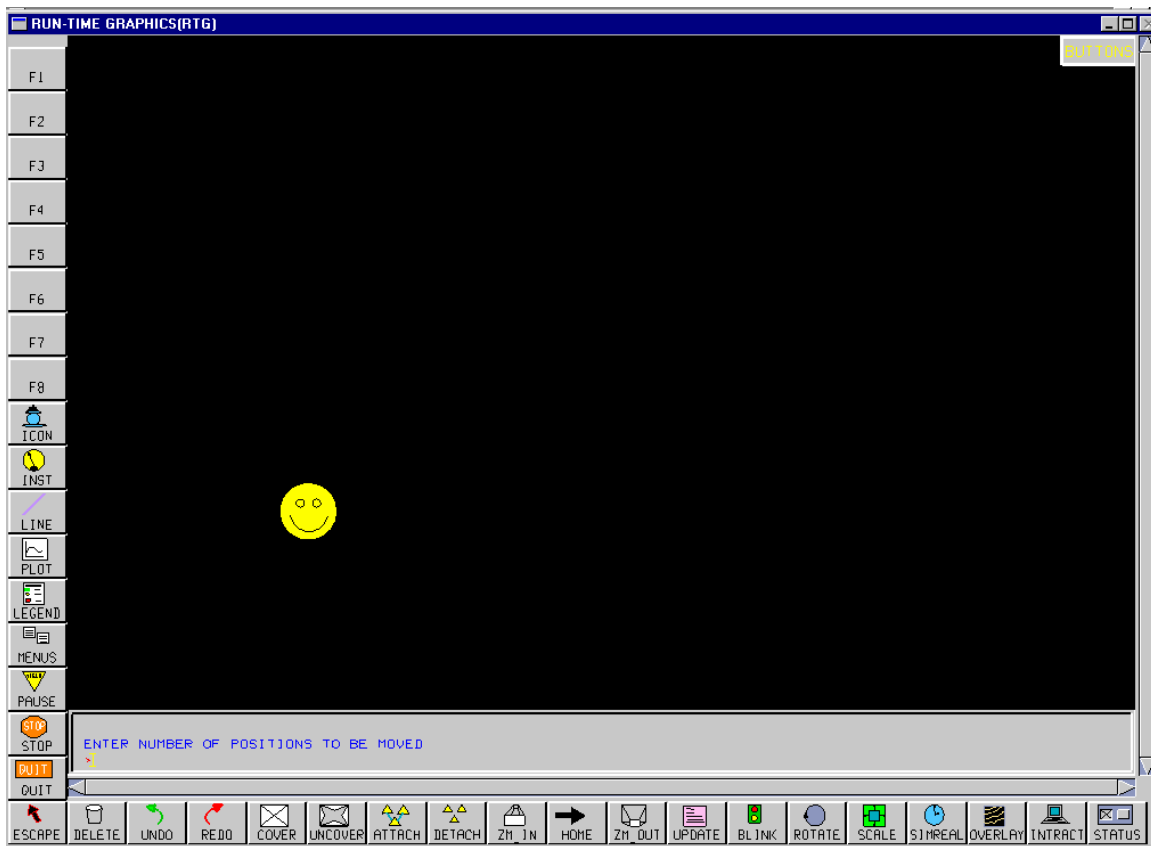
***Data for the Process: FACE_PROC

```
FACE_MAIN
  INSERT FACE AT 500,500
  EXECUTE USER_INPUT INCREMENTING I FROM 1 BY 1 TO 3

USER_INPUT
  DISPLAY PROMPT 'ENTER NUMBER OF POSITIONS TO BE MOVED'
  ACCEPT INPUT POS
  EXECUTE MOVE_FACE

MOVE_FACE
  J = POS + 500
  UPDATE FACE TO J,500
```

On running the program, your output will look something like this:



The program will ask the user for input three times, and then it will shift the smiley as indicated each time.