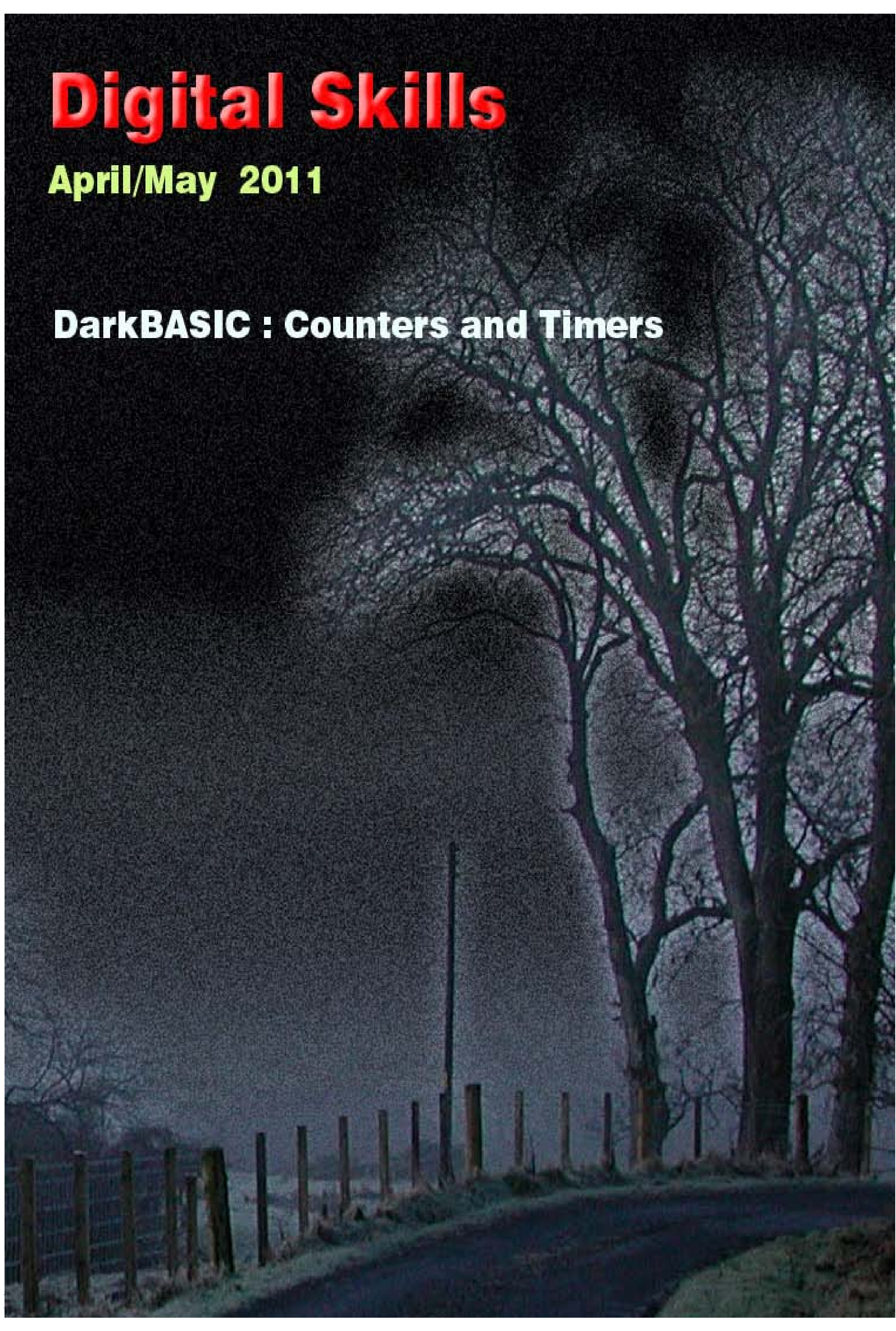

Digital Skills
April/May 2011
DarkBASIC : Counters and Timers

Counters and Timers

Introduction

Many games need to display either counters, or timers, or both.

We need to count how many points you've accumulated during a game, the number of bullets remaining in your weapon, the number of health points your characters still has. Likewise, we need to know how much time is remaining to complete a task or how much time has been taken to achieve some objective.

Of course, counters and timers are fairly easy to achieve if they are never to be displayed or only to be displayed at the end of a game; a simple variable and some type of display statement will achieve this. However, when we want to see a running total or time on screen with real time updates then we need to work a little harder. This article will take you through some of the ways of achieving this.

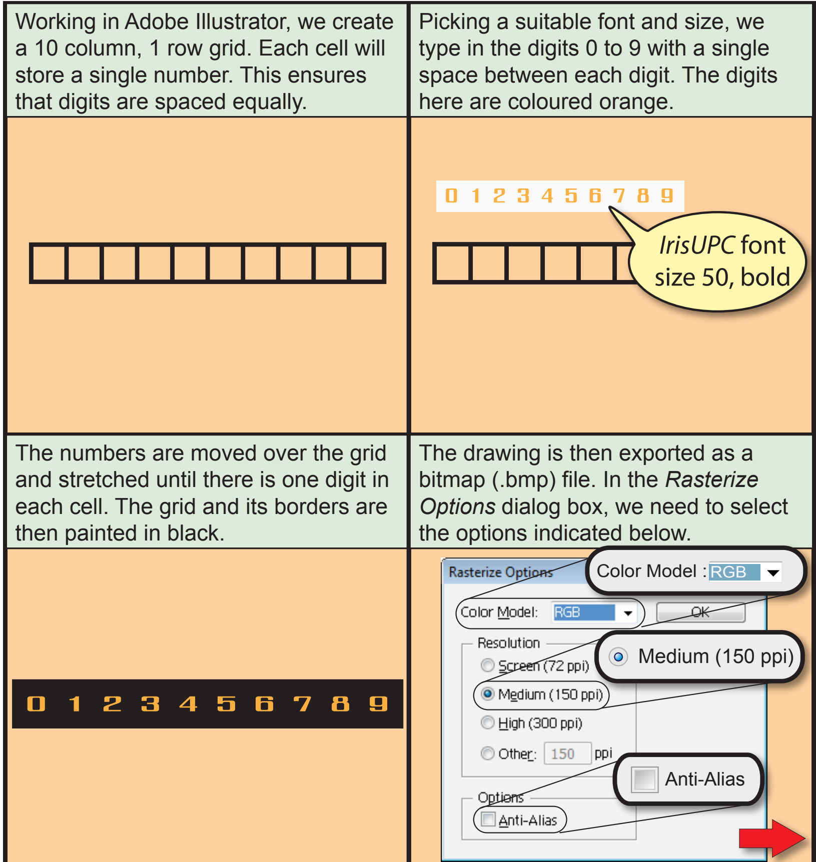
Counters

Sprites

The simplest way to create a text display when working on a game is to create a graphic containing the required display. For our counter we need an image containing the digits 0 to 9. FIG-1 shows one way to go about this.

FIG-1

Creating an Image



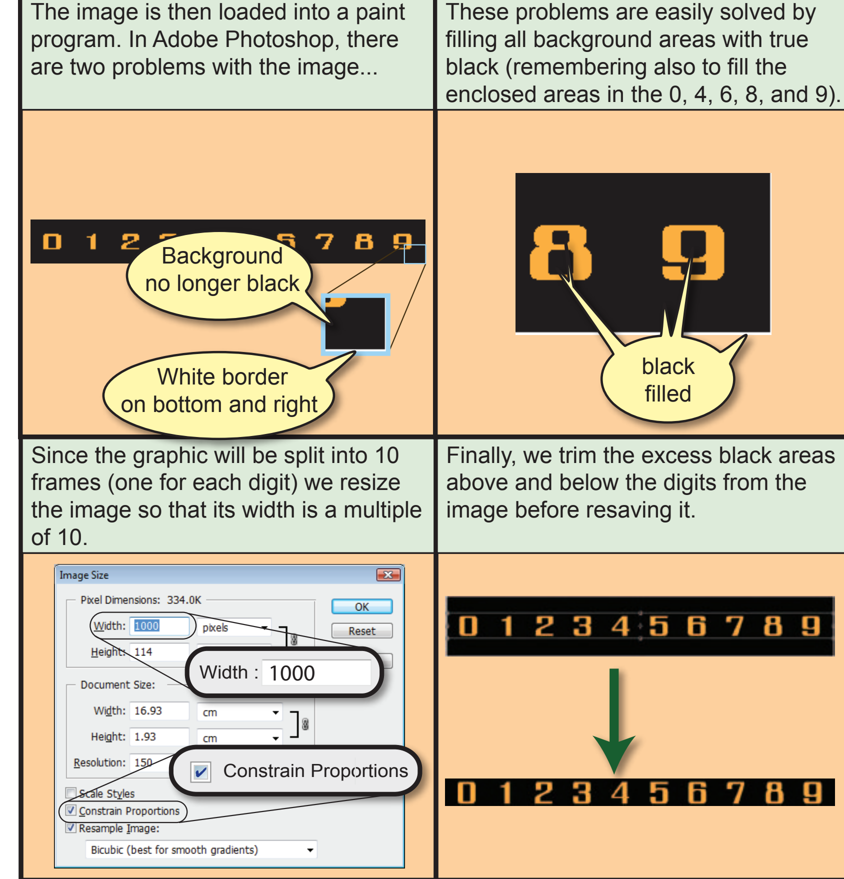
If the *Anti-Alias* option is checked, the resulting image will not display clearly, showing rough pixelation around the edges of the digits.

FIG-1
(continued)

Creating an Image

Although the background may look black, when the eye dropper is moved over the background we can see the the RGB settings are not 0,0,0 (true black).

We need the background to be true black so that it will be transparent within the sprite.



If you do not have the resources to create the necessary file, this image and all others used in this article can be downloaded from our website (**www.digital-skills.co.uk**).

Go to the *Downloads* page in the DarkBASIC section.

Activity 1

Use the software available to you to create an image containing the digits 0 to 9.

Make sure the digits are evenly spaced and that the background colour is black.

If necessary, resize the image so that its size in pixels is exactly divisible by 10.

Trim any excess background above and below the digits.

Save the file as *Digits.bmp*.

Displaying the Digits

To display the individual digits within our image we need to create a 10 framed animated sprite. Frame 1 will contain the digit 0; frame 2 the digit 1; etc.

The program in FIG-2 demonstrates the use of the image. The program has the following logic.

```
Initialise screen
Set up the animated sprite.
Position the sprite
FOR 9 times do
    Wait 500 milliseconds
    Display the next frame
ENDFOR
End the program
```

FIG-2

Testing the Digits

NOTE: You will probably have to modify the SET DISPLAY MODE line to match your own screen's resolution.

```
REM *** Initialise screen          ***
SET DISPLAY MODE 1920,1200,32

REM *** Set up the animated sprite ***
CREATE ANIMATED SPRITE 1,"Digits.bmp",10,1,1

REM *** Position the sprite        ***
SPRITE 1,640,480,1   // *** Defaults to frame 0 ***

REM *** Display each frame in turn  ***
FOR count = 2 TO 10
   WAIT 500
   SET SPRITE FRAME 1,count
NEXT count

REM *** End program                ***
WAIT KEY
END
```

Activity 2

Start up DarkBASIC Pro and create a new project called *Counter.dbpro*.

Enter the code given in FIG-2.

Copy *Digits.bmp* into the project's folder.

Execute your code and check that all 10 digits are correctly displayed.

Save your project.

If we want our count to go above 9, all we need to do is use two or more sprites. Counts up to 99 will need two sprites, 999, three sprites etc.

When we are dealing with two sprites, we need to determine which frame each of these sprites needs to be set to for a given value of a variable called *count*.

If *count* is, say, 37, then the first sprite needs to display frame 4 (which contains the digit 3) and the second sprite frame 8. We can determine the frame settings for any value of count using the formulae:

```
firstframe   = count / 10 + 1
secondframe = count mod 10 + 1
```

The program in FIG-3 displays the values 00 to 99.

FIG-3

Double Digit Counting

```
REM *** Initialise screen                 ***
SET DISPLAY MODE 1920,1200,32

REM *** Create sprites                    ***
CREATE ANIMATED SPRITE 1,"Digits.bmp",10,1,1
CREATE ANIMATED SPRITE 2,"Digits.bmp",10,1,1

REM *** Position sprites ***
SPRITE 1,640,480,1
SPRITE 2,710,480,1

REM *** Set both sprites to frame 1       ***
SET SPRITE FRAME 1,1
SET SPRITE FRAME 2,1

REM *** Display all value up to 99        ***
FOR count = 1 TO 99
   WAIT 500
   frame1 = count / 10 + 1
   frame2 = count mod 10 + 1
   SET SPRITE FRAME 1,frame1
   SET SPRITE FRAME 2,frame2
NEXT count

REM *** End program                      ***
WAIT KEY
END
```

Activity 3

Create a new project named *Counter2.dbpro*.

Enter the code given in FIG-3.

Copy *Digits.bmp* into the new project's folder.

Test your program and then save the project.

For a count which reaches three digits, the frames for each of the sprites would be calculated as follows:

```
frame1   = count / 100 + 1
temp     = count mod 100
frame2   = temp / 10 + 1
frame3   = temp mod 10 + 1
```

Note the use of the variable *temp* to remove the 100s from the count once the first frame value has been calculated.

Activity 4

Modify *Counter2.dbpro* to handle a three digit count.

 Copyright © 2011 A. Stewart

The final program (see FIG-4) makes use of the counter in a simple game. The game displays a circle at a random position on the screen. The player must then click on the circle. The number of circles clicked within a 15 second period is the score achieved.

The program makes use of the following logic:

```
Initialise screen
Load Images used by static sprites
Set up animated sprites
Set up counter sprites
Position counter sprites
Set both counter sprites to frame 1
Set count to zero
Seed random number generator
Set start time
REPEAT
    Randomly position ball
    Wait for ball to be clicked
    Increment count
    Update counter sprites
UNTIL 15 seconds passed
Display result
End program
```

The static images are the ball, a black, single pixel and final text message. The black pixel image is used to track the mouse pointer and help detect when the mouse is over the ball.

FIG-4

Using a Count in a Game

```
REM *** Initialise screen                        ***
SET DISPLAY MODE 1920,1200,32

REM *** Load images used by static sprites  ***
LOAD IMAGE "black.bmp",3
LOAD IMAGE "ball.bmp",4
LOAD IMAGE "YourScoreWas.bmp",5

REM *** Create sprites                           ***
CREATE ANIMATED SPRITE 1,"Digits.bmp",10,1,1
CREATE ANIMATED SPRITE 2,"Digits.bmp",10,1,1

REM *** Position counter sprites                 ***
SPRITE 1,1750,80,1
SPRITE 2,1820,80,1

REM *** Set both counter sprites to frame 1 ***
SET SPRITE FRAME 1,1
SET SPRITE FRAME 2,1

REM *** Set count to zero                        ***
count = 0

REM *** Seed random number generator        ***
RANDOMIZE TIMER()

REM *** Start game                               ***

REM *** Set start time                           ***
starttime = TIMER()
REPEAT
   REM *** Randomly position ball             ***
```

Continued on next page

FIG-4
(continued)

Using a Count in a Game

```
   ballx = RND (1800) + 50
   bally = RND (1100) + 50
   SPRITE 4,ballx,bally,4
   REM *** Wait for ball to be clicked ***
   REPEAT
      SPRITE 3, MOUSEX(), MOUSEY(),3
   UNTIL SPRITE COLLISION(3,4) AND MOUSECLICK() = 1
   REM *** Increment count ***
   INC count
   REM *** Update displayed count ***
   frame1 = count / 10 + 1
   frame2 = count mod 10 + 1
   SET SPRITE FRAME 1,frame1
   SET SPRITE FRAME 2,frame2

UNTIL TIMER() - starttime >= 15000

REM *** Display result           ***
HIDE SPRITE 4
SPRITE 5,380,600,5
SPRITE 1,1000,614,1
SPRITE 2,1070,614,2

REM *** End program ***
WAIT 5000
END
```

Activity 5

If you have not done so before, download the images used from our website (**www.digital-skills.co.uk**). You'll find them in the *Downloads* page of the DarkBASIC section.

Start a new project named *CounterGame.dbpro*.

Enter the code given in FIG-4.

Make sure the images you downloaded are unzipped and placed in the project's folder.

Test out the game.

Save your project.

Timers

In one respect, a typical timer is very similar to a counter: it displays an incrementing number. The main difference is that the number displayed is incremented due to the passage of time rather than to signify an increase in points or a decrease in ammunition.

Typically, a timer will increment every second, so a program which makes use of a timer will employ the following logic:

```
Initialise timer
REPEAT
    Play game for 1 second
    Update timer
UNTIL game complete
```

Digital Timer

We can make use of the methods we used to implement a counter to produce a digital clock (see the code in FIG-5).

FIG-5

Using a Digital Timer

```
SET DISPLAY MODE 1920,1200,32

REM *** Load image used by static sprite    ***
LOAD IMAGE "Colon.bmp",1

REM *** Create sprites                      ***
CREATE ANIMATED SPRITE 1,"Digits.bmp",10,1,2 `Minutes
CREATE ANIMATED SPRITE 2,"Digits.bmp",10,1,2 `Seconds (tens)
CREATE ANIMATED SPRITE 3,"Digits.bmp",10,1,2 `Seconds (units)

REM *** Position sprites                    ***
SPRITE 1,490,80,2
SPRITE 4,560,80,1    `Colon
SPRITE 2,570,80,2
SPRITE 3,640,80,2

REM *** Set sprites to frame 1              ***
SET SPRITE FRAME 1,1
SET SPRITE FRAME 2,1
SET SPRITE FRAME 3,1

REM *** Set timer to zero                   ***
time = 0
REPEAT
   REM *** Wait one second                  ***
   WAIT 1000
   REM *** Update timer                     ***
   time = time + 1
   minutes = time / 60
   seconds = time mod 60
   SET SPRITE FRAME 1,  minutes + 1
   SET SPRITE FRAME 2, seconds / 10 + 1
   SET SPRITE FRAME 3, seconds mod 10 + 1
UNTIL time = 100
END
```

Activity 6

Start a new project and test out the code given in FIG-5.

Remember to copy the Digits.bmp *and* Colon.bmp *files to the new project's folder.*

Of course, we could make the clock count backwards down to zero simply by changing three lines:

First we need to intialise the time variable to the number of seconds allowed, for example:

```
time = 100
```

The second modification is to decrease the time value after a one second wait:

```
time = time - 1
```

And lastly, we need to change the condition in the UNTIL statement to halt the loop when the time is up:

```
UNTIL time = 0
```

Activity 7

Modify your last project so that the timer counts backwards.

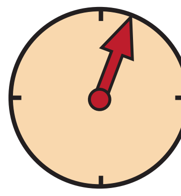
Analogue Timers

An alternative to the digital clock is to use the older analogue-style clock with a moving hand. This is probably best suited to a countdown clock with a maximum of a 60 second time limit.

A typical clock design is shown in FIG-6.

FIG-6

A Countdown Timer

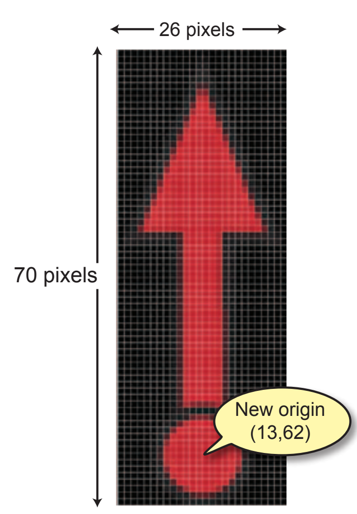


The clock face and hand are created as two separate images. The passing of time is shown by rotating the hand.

The trick is to reset the origin of the hand so that it is at the centre of the small circle at one end of the hand (see FIG-7).

FIG-7

The Clock Hand



 Copyright © 2011 A. Stewart

When a sprite is rotated, it always rotates about its point of origin so this repositioning will ensure that the sprite rotates in an identical way to the hand of a real clock.

The hand sprite itself must be positioned over the face so that its origin is in the exact centre of the face.

In the images supplied the face is exactly 128 x 128 pixels so the hand's new origin needs to be placed at position (64,64) on the face.

Since a circle is 360° and one revolution of the clock face represents 60 seconds, each second which passes requires the hand to rotate by 6° (360 / 60).

The complete program (shown in FIG-8) makes use of the following logic:

```
Load images used
Create sprites
Reposition hand sprite's origin
Position hand sprite
Set time to 60
REPEAT
    Wait 1 second
    Subtract 1 from time
    Rotate hand 6°
UNTIL time = 0
```

FIG-8

Using a Countdown Analogue Timer

```
REM *** Initialise display                  ***
SET DISPLAY MODE 1920, 1200, 32

REM *** Load images used by sprites         ***
LOAD IMAGE "face.bmp", 1
LOAD IMAGE "hand.bmp",2

REM *** Create sprites ***
SPRITE 1,400,400,1
SPRITE 2,0,0,2

REM *** Reposition hand sprites origin      ***
OFFSET SPRITE 2,13,62

REM *** Position hand sprite over face      ***
SPRITE 2,464,464,2

REM *** Set the time to 60 seconds          ***
time = 60

REPEAT
   REM *** Wait 1 second                    ***
   WAIT 1000
   REM *** Update time                      ***
   time = time - 1
   REM *** Rotate hand by 6 degrees         ***
   ROTATE SPRITE 2,time*6
UNTIL time = 0
END
```

Of course, you don't have to time a full minute. For example, if you want to limit the time to, say, 45 seconds, its just a matter of setting the *time* variable to 45 and starting the hand already rotated by 90°.

Activity 8

Start a new DBPro project and enter the code given above.

Copy the required images into the project folder and test your program.

Modify the program to allow only 30 seconds on the clock.

Since the analogue timer doesn't show the exact time, you might want to display the digital time beneath the face as demonstrated in the program in FIG-9.

FIG-9

Analogue and Digital Display

The animated sprites make use of a reduced-size version of the digits image.

```
REM *** Initialise display                *** 
SET DISPLAY MODE 1920, 1200, 32
REM *** Load images used by sprites       ***
LOAD IMAGE "face.bmp", 1
LOAD IMAGE "hand.bmp",2
REM *** Create sprites ***
SPRITE 1,400,400,1
SPRITE 2,0,0,2
REM *** Create animated sprites for       ***
REM *** digital count                     ***
CREATE ANIMATED SPRITE 3,"digitssmall.bmp",10,1,3
CREATE ANIMATED SPRITE 4,"digitssmall.bmp",10,1,3
REM *** Reposition hand sprites origin    ***
OFFSET SPRITE 2,13,62
REM *** Position hand sprite over face    ***
SPRITE 2,464,464,2
REM *** Position digital count            ***
SPRITE 3,420,540,3
SPRITE 4,455,540,3
REM *** Set the time to 60 seconds        ***
time = 60
REM *** Set digits to 60                  ***
SET SPRITE FRAME 3,7
SET SPRITE FRAME 4,1
REPEAT
   REM *** Wait 1 second                  ***
   WAIT 1000
   REM *** Update time                   ***
   time = time - 1
   REM *** Rotate hand by 6 degrees       ***
   ROTATE SPRITE 2,time*6
   REM *** Set digital count              ***
   SET SPRITE FRAME 3, time / 10 + 1
   SET SPRITE FRAME 4, time mod 10 + 1
UNTIL time = 0
END
```

Activity 9

Modify your last program to incorporate a digital readout beneath the clock face.

Copyright © 2011 A. Stewart

Fuse Bomb Timer

If you don't want to specify an exact time to the player of your game but do want to show them that they have a limited time in which to achieve some objective, you could make use of more visually interesting timers such as a cartoon-style, fused bomb (see FIG-10).

FIG-10

A Bomb Timer

This time we need several graphics:

➢ a static graphic for the bomb

➢ an animated graphic showing the shorting fuse

➢ an animated graphic for the burning tip of the fuse

➢ an animated graphic showing the explosion

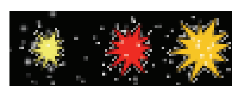
These are shown in FIG-11.

FIG-11

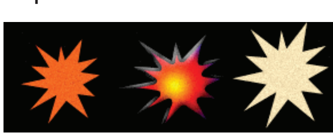
Bomb Timer Graphics

Fuse

The fuse is organised as a 7 x 3 graphic, but the last two frames are unused.



Explosion



The program required is a bit more complicated this time since we need to maintain the flame effect at the end of the fuse. To make things easier, we'll look at the code in sections before giving the complete program.

The first innovation in the program is to position all of the graphics relative to the bomb's position rather than give absolute coordinates for everything. This way, if we want to move the timer to a different position on the screen we only need to modify the coordinates of the bomb graphic, the others will be repositioned automatically.

So the bomb's coordinates are set using two named constants:

```
REM *** Program constants ***
REM *** Position of bomb  ***
#CONSTANT BOMBX = 400
#CONSTANT BOMBY = 400
```

When the burn graphic is shown, it has to move as the fuse grows shorter. The burn's offset from the bomb after each second is stored in two arrays. This requires the following declarations:

The symbol ⇘ is used to signify the continuation of a line.

```
REM *** Burn offsets ***
REM *** x offsets ***
DATA 122,121,119,113,105,   98,90,84,79,78,   78,74,70,63,53,
      ⇘45,38,33,31
REM *** y offsets ***
DATA 0,8,17,25,29,   29,27,21,12,3,   -6,-16,-25,-30,-30,
      ⇘-29,-24,-14,-5
REM  *** Arrays for burn offsets ***
DIM x(19)
DIM y(19)
```

The data is copied into the arrays using a function:

```
FUNCTION InitialiseData()
   REM *** Read x offsets into array        ***
   FOR c = 1 TO 19
      READ x(c)
   NEXT c
   REM *** Read y offsets int array         ***
   FOR c = 1 TO 19
      READ y(c)
   NEXT c
ENDFUNCTION
```

Another function loads all the graphics ready for use:

```
FUNCTION InitialiseGraphics()
   LOAD IMAGE "bomb.bmp",1
   CREATE ANIMATED SPRITE 2,"fuse.bmp",7,3,2
   CREATE ANIMATED SPRITE 3,"flame.bmp",3,1,3
   OFFSET SPRITE 3,16,16
   CREATE ANIMATED SPRITE 4,"blast.bmp",3,1,4
   OFFSET SPRITE 4,50,50
ENDFUNCTION
```

Notice the offset for the burn and explosion sprites (*flame.bmp* and *blast.bmp,* sprites 3 and 4). The offset points are calculated as the centre point of the sprites so that they are correctly positioned later in the program.

Since we'll want to add the sound of the fuse burning and the final explosion, we need a another routine to load the sound files:

```
FUNCTION InitialiseSound()
  LOAD SOUND "Fuse.wav",1
  LOAD SOUND "Bomb.wav",2
ENDFUNCTION
```

One routine positions the bomb and the fuse

```
FUNCTION SetUpBomb()
   REM *** Position bomb                          ***
   SPRITE 1,BOMBX,BOMBY,1
   REM *** Position fuse                          ***
   SPRITE 2,BOMBX+30,BOMBY-32,2
ENDFUNCTION
```

and the main routine runs through the complete animation reducing the length of the fuse and making sure the burn animation is correctly positioned:

```
FUNCTION Play()
   REM *** Set time till explosion                ***
   time = 19
   REPEAT
      REM *** Calculate frame and array subscript ***
      sub = 20 - time
      REM *** Update fuse sprite frame            ***
      SET SPRITE FRAME 2,sub
      REM *** Move burn sprite                    ***
      SPRITE 3,BOMBX+x(sub),BOMBY+y(sub),3
      REM *** Play burn for 1 second              ***
      now = TIMER()
      PLAY SOUND 1
      REPEAT
         PLAY SPRITE 3,1,3,3
      UNTIL TIMER()-now >= 1000
      REM *** Reduce time remaining by one second ***
      time = time - 1
   UNTIL time = 0
   REM *** Explode bomb                           ***
   Explode()
ENDFUNCTION
```

The *Explode()* function plays the explosion animation, increasing its size and making it fade to invisibility:

```
FUNCTION Explode()
   REM *** Delete all existing sprites            ***
   DELETE SPRITE 1
   DELETE SPRITE 2
   DELETE SPRITE 3
   REM *** And burn sound                         ***
   DELETE SOUND 1
   REM *** Position explosion sprite              ***
   SPRITE 4, BOMBX, BOMBY,4
   REM *** Play explosion sound                   ***
   PLAY SOUND 2
   REM *** Make explosion bigger and fainter      ***
   scale = 100
   alpha = 255
   REPEAT
      now = TIMER()
      REPEAT
```

```
            SCALE SPRITE 4, scale
            PLAY SPRITE 4,1,3,3
         UNTIL TIMER()-now >= 100
         scale = scale + 50
         alpha = alpha / 2
         SET SPRITE ALPHA 4,alpha
      UNTIL scale > 400
      REM *** Delete explosion sprite          ***
     DELETE SPRITE 4
  ENDFUNCTION
```

The complete program is shown in FIG-12.

FIG-12

A Bomb Timer

```
REM *** Program constants                              ***
REM *** Position of bomb                               ***
#CONSTANT BOMBX = 400
#CONSTANT BOMBY = 400

REM *** Burn offsets                                   ***
REM *** x offsets                                      ***
DATA 122,121,119,113,105,   98,90,84,79,78,   78,74,70,63,53,
      ↳45,38,33,31

REM *** y offsets                                      ***
DATA 0,8,17,25,29,   29,27,21,12,3,   -6,-16,-25,-30,-30,
      ↳-29,-24,-14,-5

REM *** Arrays for burn offsets                        ***
DIM x(19)
DIM y(19)

REM **********************************************
REM *             Main program                   *
REM **********************************************
InitialiseData()
SET DISPLAY MODE 1920, 1200,32
InitialiseGraphics()
InitialiseSound()
SetUpBomb()
Play()
WAIT KEY
END




REM **********************************************
REM *                Functions                   *
REM **********************************************

FUNCTION InitialiseData()
FOR c = 1 TO 19
   READ x(c)
NEXT c
FOR c = 1 TO 19
   READ y(c)
NEXT c
ENDFUNCTION
```

Continued on next page

FIG-12
(continued)

A Bomb Timer

```
FUNCTION InitialiseGraphics()
   LOAD IMAGE "bomb.bmp",1
   CREATE ANIMATED SPRITE 2,"fuse.bmp",7,3,2
   CREATE ANIMATED SPRITE 3,"flame.bmp",3,1,3
   OFFSET SPRITE 3,16,16
   CREATE ANIMATED SPRITE 4,"blast.bmp",3,1,4
   OFFSET SPRITE 4,50,50
ENDFUNCTION


FUNCTION InitialiseSound()
   LOAD SOUND "Fuse.wav",1
   LOAD SOUND "Bomb.wav",2
ENDFUNCTION


FUNCTION SetUpBomb()
   SPRITE 1,BOMBX,BOMBY,1
   SPRITE 2,BOMBX+30,BOMBY-32,2
ENDFUNCTION


FUNCTION Play()
   REM *** Set time till explosion                ***
   time = 19
   REPEAT
      REM *** Calculate frame and array subscript ***
      sub = 20 - time
      REM *** Update fuse sprite                  ***
      SET SPRITE FRAME 2,sub
      REM *** Move burn sprite                    ***
      SPRITE 3,BOMBX+x(sub),BOMBY+y(sub),3
      REM *** Play burn for 1 second              ***
      now = TIMER()
      PLAY SOUND 1
      REPEAT
         PLAY SPRITE 3,1,3,3
      UNTIL TIMER()-now >= 1000
      REM *** Reduce time remaining by one second ***
      time = time - 1
   UNTIL time = 0
   Explode()
ENDFUNCTION


FUNCTION Explode()
   REM *** Delete all existing sprites            ***
   DELETE SPRITE 1
   DELETE SPRITE 2
   DELETE SPRITE 3
   REM *** Move sprite to correct position        ***
   SPRITE 4, BOMBX, BOMBY,4
   REM *** Play explosion sound                   ***
   PLAY SOUND 2
   REM *** Make explosion bigger and fainter      ***
   scale = 100
   alpha = 255
   REPEAT
      now = TIMER()
      REPEAT
```

Continued on next page

FIG-12
(continued)

A Bomb Timer

```
         SCALE SPRITE 4, scale
         PLAY SPRITE 4,1,3,3
      UNTIL TIMER()-now >= 100
      scale = scale + 50
      alpha = alpha / 2
      SET SPRITE ALPHA 4,alpha
   UNTIL scale > 400
   REM ***  Delete explosion sprite             ***
   DELETE SPRITE 4
ENDFUNCTION
```

For copyright reasons the sound files *fuse.wav* and *bomb.wav* are not included in the downloads available from our website. You will find these files at *www.soundbible.com*.

Activity 10

Type in and test the program given in FIG-12.

Try moving the bomb to a different position on the screen by changing the values assigned to the constants BOMBX and BOMBY.

Smoother Burn Movement

A problem with the burning fuse is that it moves in very obvious steps. It would be better if we could create a smoother movement of the flame.

To do this we need to calculate the distance the burn travels between one step and the next. This can be done using the lines

```
REM *** Calculate the difference between    ***
REM *** this position and the next          ***
diffx# = x(sub+1)-x(sub)
diffy# = y(sub+1)-y(sub)
```

But to handle the final move, we need to add a dummy value to the x and y offsets. This means the following changes:

```
REM *** x offsets ***
DATA 122,121,119,113,105,   98,90,84,79,78,   78,74,70,63,53,
      ↳45,38,33,31,31
REM *** y offsets ***
DATA 0,8,17,25,29,   29,27,21,12,3,   -6,-16,-25,-30,-30,
      ↳-29,-24,-14,-5,-5
REM  *** Arrays for burn offsets ***
DIM x(20)
DIM y(20)
```

And, of course, that means we have an extra item of data to read into each of the arrays:

```
FUNCTION InitialiseData()
   FOR c = 1 TO 20
      READ x(c)
   NEXT c
   FOR c = 1 TO 20
      READ y(c)
   NEXT c
ENDFUNCTION
```

The main change is to the *Play()* function where we need to calculate the distance moved

by the flame from one second to the next and then move the flame in interim steps every, say, 200 milliseconds. The modified code for the routine is shown below:

```
FUNCTION Play()
   REM *** Set time till explosion ***
   time = 19
   REPEAT
      REM *** Calculate frame and array subscript ***
      sub = 20 - time
      REM *** Update fuse sprite                  ***
      SET SPRITE FRAME 2,sub
      REM *** Move burn sprite                    ***
      SPRITE 3,BOMBX+x(sub),BOMBY+y(sub),3
      REM *** Calculate the difference between    ***
      REM *** this position and the next          ***
      diffx# = x(sub+1)-x(sub)
      diffy# = y(sub+1)-y(sub)
      REM *** Interim move counter                ***
      move = 1
      REM *** Play burn for 1 second              ***
      now = TIMER()
      PLAY SOUND 1
      REPEAT
         REM *** IF multiple of 200 msec  THEN    ***
         IF TIMER() - now > 200 * move
            REM *** Move flame slightly           ***
            SPRITE 3, BOMBX+x(sub)+move*(diffx#/5),
                     ↳BOMBY+y(sub)+move*(diffy#/5),3
            REM *** Next move number              ***
            move = move + 1
         ENDIF
         PLAY SPRITE 3,1,3,3
      UNTIL TIMER()-now >= 1000
      REM *** Reduce time remaining by one second ***
      time = time - 1
   UNTIL time = 0
   Explode()
ENDFUNCTION
```

Using a Timer in a Game

So far we've created the code for various timers, but of course, these timers are meant to be used as just one small part of a game. In this section we'll look at how to incorporate the timer code within a game.

Using the Digital Timer

To demonstrate the digital counter, we create a variation on the ball game we used to demonstrate counters earlier in this chapter.

The main difficulty is integrating the game play into the one second time interval before the clock needs to be updated. This is all achieved by one function:

```
FUNCTION Play()
   REM *** No time has passed yet                 ***
   time = 0
   REM *** No balls clicked yet                   ***
   count = 0
   REM *** New ball must be shown                 ***
```

```
      newballneeded = 1
      REM *** Play until 20 balls clicked             ***
      REPEAT
         REM *** Play for one second                  ***
         now = TIMER()
         REPEAT
            REM *** IF new ball needed THEN           ***
            IF newballneeded = 1
               REM *** Randomly position ball         ***
               ballx = RND (1800) + 50
               bally = RND (1100) + 50
               SPRITE 3,ballx,bally,3
               REM *** New ball not required          ***
               newballneeded = 0
               REM *** Ball not clicked               ***
               ballclicked = 0
            ENDIF
            REM *** Wait for ball clicked  or 1 sec   ***
            REPEAT
               SPRITE 2, MOUSEX(), MOUSEY(),2
               REM *** IF ball clicked THEN           ***
               IF (SPRITE COLLISION(2,3) AND MOUSECLICK() = 1)
                  REM *** ball clicked                ***
                  ballclicked = 1
                  REM *** New ball needed             ***
                  newballneeded = 1
                  REM Add 1 to count of balls clicked ***
                  count = count + 1
               ENDIF
            UNTIL  ballclicked = 1 OR TIMER()-now >=1000
         UNTIL TIMER()-now >= 1000
            REM *** Update timer                      ***
            time = time + 1
            minutes = time / 60
            seconds = time mod 60
            SET SPRITE FRAME 6,  minutes + 1
            SET SPRITE FRAME 7, seconds / 10 + 1
            SET SPRITE FRAME 8, seconds mod 10 + 1
      UNTIL count = 20
   ENDFUNCTION
```

The complete program is shown in FIG-13.

FIG-13

Digital Timer Game

```
REM *** Program constants                            ***
REM *** Clock position                               ***
#CONSTANT CLOCKX 1600
#CONSTANT CLOCKY 20

REM **********************************************
REM *             Main program                   *
REM **********************************************
REM *** Initialise screen                            ***
SET DISPLAY MODE 1920,1200,32
InitialiseGraphics()
SetUpClock()
REM *** Seed random number generator                 ***
RANDOMIZE TIMER()
Play()
WAIT 5000
END
```

Continued on next page

 Copyright © 2011 A. Stewart

FIG-13
(continued)

Digital Timer Game

```
REM *********************************************
REM *                 Functions                 *
REM *********************************************

FUNCTION InitialiseGraphics()
   REM *** Load images used by static sprites     ***
   LOAD IMAGE "Colon.bmp",1
   LOAD IMAGE "black.bmp",2
   LOAD IMAGE "ball.bmp",3
   LOAD IMAGE "YourTimeWas.bmp",4
   LOAD IMAGE "ClockBackground.bmp",5
   REM *** Create animated sprites for timer      ***
   CREATE ANIMATED SPRITE 6,"Digits.bmp",10,1,6 `Minutes
   CREATE ANIMATED SPRITE 7,"Digits.bmp",10,1,6 `Seconds (tens)
   CREATE ANIMATED SPRITE 8,"Digits.bmp",10,1,6 `Seconds (units)
   REM *** Ensure digits on top of clock background ***
   SET SPRITE PRIORITY 6,1
   SET SPRITE PRIORITY 7,1
   SET SPRITE PRIORITY 8,1
ENDFUNCTION

FUNCTION SetUpClock()
   REM *** Position clock sprites                 ***
   SPRITE 5,CLOCKX,CLOCKY,5
   SPRITE 6,CLOCKX-10,CLOCKY+35,6
   SPRITE 1,CLOCKX+55,CLOCKY+35,1    `Colon
   SPRITE 7,CLOCKX+60,CLOCKY+35,6
   SPRITE 8,CLOCKX+115,CLOCKY+35,6
ENDFUNCTION

FUNCTION Play()
   REM *** No time has passed yet                 ***
   time = 0
   REM *** No balls clicked yet                   ***
   count = 0
   REM *** New ball must be shown                 ***
   newballneeded = 1
   REM *** Play until 20 balls clicked            ***
   REPEAT
      REM *** Play for one second                 ***
      now = TIMER()
      REPEAT
         REM *** IF new ball needed THEN          ***
         IF newballneeded = 1
            REM *** Randomly position ball        ***
            ballx = RND (1800) + 50
            bally = RND (1100) + 50
            SPRITE 3,ballx,bally,3
            REM *** New ball not required         ***
            newballneeded = 0
            REM *** Ball not clicked              ***
            ballclicked = 0
         ENDIF
         REM *** Wait for ball clicked  or 1 sec  ***
         REPEAT
            SPRITE 2, MOUSEX(), MOUSEY(),2
            REM *** IF ball clicked THEN          ***
            IF (SPRITE COLLISION(2,3) AND MOUSECLICK() = 1)
```

Continued on next page

FIG-13 (continued)

Digital Timer Game

```
                REM *** ball clicked              ***
                ballclicked = 1
                REM *** New ball needed           ***
                newballneeded = 1
                REM Add 1 to count of balls clicked ***
                count = count + 1
             ENDIF
          UNTIL  ballclicked = 1 OR TIMER()-now >=1000
       UNTIL TIMER()-now >= 1000
          REM *** Update timer                    ***
          time = time + 1
          minutes = time / 60
          seconds = time mod 60
          SET SPRITE FRAME 6,  minutes + 1
          SET SPRITE FRAME 7, seconds / 10 + 1
          SET SPRITE FRAME 8, seconds mod 10 + 1
    UNTIL count = 20
ENDFUNCTION
```

Activity 11

Type in and test the program given in FIG-13.

Using the Bomb

Embedding the game play within the bomb timer is slightly more complex since we need to keep the burn effect animated as the game plays. Again, we have made use of the ball game to demonstrate the techniques used. The complete program is shown in FIG-14.

FIG-14

Bomb Timer Game

```
REM *** Program constants                          ***
REM *** Position of bomb                           ***
#CONSTANT BOMBX = 400
#CONSTANT BOMBY = 400

REM *** Burn offsets                               ***
REM *** x offsets                                  ***
DATA 122,121,119,113,105,   98,90,84,79,78,   78,74,70,63,53,
      ↳45,38,33,31,31
REM *** y offsets                                  ***
DATA 0,8,17,25,29,   29,27,21,12,3,   -6,-16,-25,-30,-30,
      ↳-29,-24,-14,-5,-5
REM  *** Arrays for burn offsets                   ***
DIM x(20)
DIM y(20)

REM ***********************************************
REM *             Main program                    *
REM ***********************************************
InitialiseData()
SET DISPLAY MODE 1920, 1200,32
InitialiseGraphics()
InitialiseSound()
SetUpBomb()
Play()
END
```

Continued on next page

 Copyright © 2011 A. Stewart

FIG-14
(continued)

Bomb Timer Game

```
REM *************************************************
REM *                Functions                      *
REM *************************************************

FUNCTION InitialiseData()
FOR c = 1 TO 20
   READ x(c)
NEXT c
FOR c = 1 TO 20
   READ y(c)
NEXT c
ENDFUNCTION


FUNCTION InitialiseGraphics()
   REM *** Load bomb graphics                           ***
   LOAD IMAGE "bomb.bmp",1
   CREATE ANIMATED SPRITE 2,"fuse.bmp",7,3,2
   CREATE ANIMATED SPRITE 3,"flame.bmp",3,1,3
   OFFSET SPRITE 3,16,16
   CREATE ANIMATED SPRITE 4,"blast.bmp",3,1,4
   OFFSET SPRITE 4,50,50
   REM *** Load game graphics                           ***
   LOAD IMAGE "black.bmp",5
   LOAD IMAGE "ball.bmp",6
ENDFUNCTION


FUNCTION InitialiseSound()
   LOAD SOUND "Fuse.wav",1
   LOAD SOUND "Bomb.wav",2
ENDFUNCTION


FUNCTION SetUpBomb()
   REM *** Position bomb                                ***
   SPRITE 1,BOMBX,BOMBY,1
   REM *** Position fuse                                ***
   SPRITE 2,BOMBX+30,BOMBY-32,2
ENDFUNCTION


FUNCTION Play()
   REM *** Set time till explosion                      ***
   time = 19
   REM *** No balls clicked yet                         ***
   count = 0
   REM *** New ball must be shown                       ***
   newballneeded = 1
   REM *** Play until 20 balls clicked or bomb explodes ***
   REPEAT
      REM *** Calculate frame and array subscript ***
      sub = 20 - time
      REPEAT
         REM *** IF new ball needed THEN           ***
         IF newballneeded = 1
            REM *** Randomly position ball         ***
            ballx = RND (1800) + 50
            bally = RND (1100) + 50
```

Continued on next page

FIG-14
(continued)

Bomb Timer Game

```
            SPRITE 6,ballx,bally,6
            REM *** New ball not required            ***
            newballneeded = 0
         ENDIF
         REM *** Update fuse sprite                  ***
         SET SPRITE FRAME 2,sub
         REM *** Move burn sprite                    ***
         SPRITE 3,BOMBX+x(sub),BOMBY+y(sub),3
         REM *** Calculate the difference between    ***
         REM *** this position and the next          ***
         diffx# = x(sub+1)-x(sub)
         diffy# = y(sub+1)-y(sub)
         REM *** Interim move counter                ***
         move = 1
         REM *** Play burn for 1 second              ***
         now = TIMER()
         LOOP SOUND 1
         REPEAT
            REM *** IF multiple of 200 msec  THEN    ***
            IF TIMER() - now > 200 * move
               REM *** Move flame slightly           ***
               SPRITE 3, BOMBX+x(sub)+move*(diffx#/5),
                        ↳BOMBY+y(sub)+move*(diffy#/5),3
               REM *** Next move number              ***
               move = move + 1
            ENDIF
            PLAY SPRITE 3,1,3,1
            SPRITE 5, MOUSEX(), MOUSEY(),5
            REM *** IF mouse clicked over ball THEN  ***
            IF (SPRITE COLLISION(5,6) AND MOUSECLICK() = 1)
               REM *** New ball needed               ***
               newballneeded = 1
               REM Add 1 to count of balls clicked   ***
               count = count + 1
            ENDIF
         UNTIL newballneeded=1 OR TIMER()-now >=1000
      UNTIL TIMER()-now >= 1000
      REM *** Reduce time remaining by one second    ***
      time = time - 1
   UNTIL time = 0 OR count = 20
   REM *** Hide ball                                 ***
   DELETE SPRITE 6
   IF time = 0
      Explode()
   ELSE
      DestroyBomb()
   ENDIF
ENDFUNCTION


FUNCTION DestroyBomb()
   DELETE SPRITE 1
   DELETE SPRITE 2
   DELETE SPRITE 3
   DELETE SOUND 1
ENDFUNCTION
```

Continued on next page

FIG-14
(continued)

Bomb Timer Game

```
FUNCTION Explode()
   REM *** Delete all existing sprites             ***
   DELETE SPRITE 1
   DELETE SPRITE 2
   DELETE SPRITE 3
   REM *** And burn sound                          ***
   DELETE SOUND 1
   REM *** Move sprite to correct position         ***
   SPRITE 4, BOMBX, BOMBY,4
   REM *** Play explosion sound                    ***
   PLAY SOUND 2
   REM *** Make explosion bigger and fainter       ***
   scale = 100
   alpha = 255
   REPEAT
      now = TIMER()
      REPEAT
         SCALE SPRITE 4, scale
         PLAY SPRITE 4,1,3,3
      UNTIL TIMER()-now >= 100
      scale = scale + 50
      alpha = alpha / 2
      SET SPRITE ALPHA 4,alpha
   UNTIL scale > 400
   REM *** Delete explosion sprite                 ***
   DELETE SPRITE 4
ENDFUNCTION
```

Activity 12

Type in and test the program given in FIG-14.