

Drag and Drop

Introduction

A common requirement in a GUI environment and in many games is the ability to use the mouse to drag items about the screen. In this section we'll see how this can be done in DarkBASIC Pro.

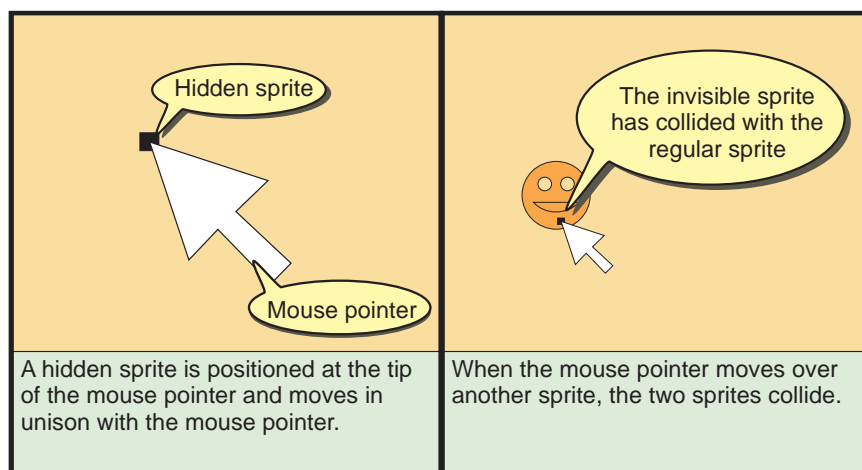
The examples shown below use sprites, but there is no reason why other items such as images and pictures cannot be dragged using the techniques given.

Detecting When the Mouse is Over a Sprite

As we saw in Hands On DarkBASIC Pro volume 1, a simple way of detecting when the mouse is positioned over a sprite is to create an invisible one-pixel sprite which is positioned at the mouse pointer tip and then check for the invisible sprite hitting the other sprite (see FIG-Sup08).

Fig-Sup08

Using an Invisible Sprite to Detect the Mouse-Over Event



We start by creating a 1x1 black pixel image (*black.bmp*) and loading this into an image:

```
#CONSTANT BlackImg 1
LOAD IMAGE "black.bmp",BlackImg
```

The image is then linked to the invisible sprite:

```
#CONSTANT InvisibleSpr 1
SPRITE InvisibleSpr,0,0,BlackImg
```

To make sure that the sprite is always positioned at the mouse tip, the line

```
SPRITE InvisibleSpr, MOUSEX(), MOUSEY(), BlackImg
```

must be repeatedly executed.

These lines are brought together in the example program given in LISTING-Sup008.

LISTING-Sup008

Linking a Sprite to the Mouse Pointer

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg      1
REM *** Sprites ***
#CONSTANT InvisibleSpr  1

REM *** Load image ***
LOAD IMAGE "black.bmp",BlackImg

REM *** Load sprite ***
SPRITE InvisibleSpr,0,0,BlackImg

REM *** Main loop ***
DO
  SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),BlackImg
LOOP

REM *** End program ***
END
```

Activity Sup019

Type in the program given above.

To check that the code actually works, change the image used from *black.bmp* to *red.bmp* which is both visible and larger.

Once you have seen that the technique works, return to using the *black.bmp* image.

We can check for the mouse moving over a regular sprite by adding a new sprite to our program:

```
#CONSTANT SmileyImg      2
#CONSTANT SmileySpr      2
LOAD IMAGE "smiley.bmp",SmileyImg
SPRITE SmileySpr,400,300,SmileyImg
```

and changing the main loop to check for a collision between the sprites:

```
DO
  SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),BlackImg
  IF SPRITE HIT(InvisibleSpr, SmileySpr)
    PRINT "HIT"
  ENDIF
LOOP
```

Activity Sup020

Modify your existing program to display the word "HIT" when the mouse pointer moves over the smiley sprite.

Dragging the Sprite

Moving the regular sprite is little different from moving the invisible sprite. The main difference being that the regular sprite should only move when the mouse pointer is over the sprite and the mouse button is being pressed.

A sprite's origin is normally at its top-left corner; this can cause problems when dragging a sprite, so it's best if we move the origin to the centre of the sprite with the line:

```
OFFSET SPRITE SmileySpr, SPRITE WIDTH(SmileySpr)/2,  
↳SPRITE HEIGHT(SmileySpr)/2
```

Now all we need to do is change the IF statement in the main loop to read:

```
IF SPRITE HIT(InvisibleSpr, SmileySpr) AND (MOUSECLICK() = 1)  
  SPRITE SmileySpr, MOUSEX(), MOUSEY(),SmileySpr  
ENDIF
```

Activity Sup021

Make the changes described above and run the program to check that the sprite can now be dragged.

The Jump Problem

Although the code works, there is a slight problem when we first click on the sprite to be dragged.

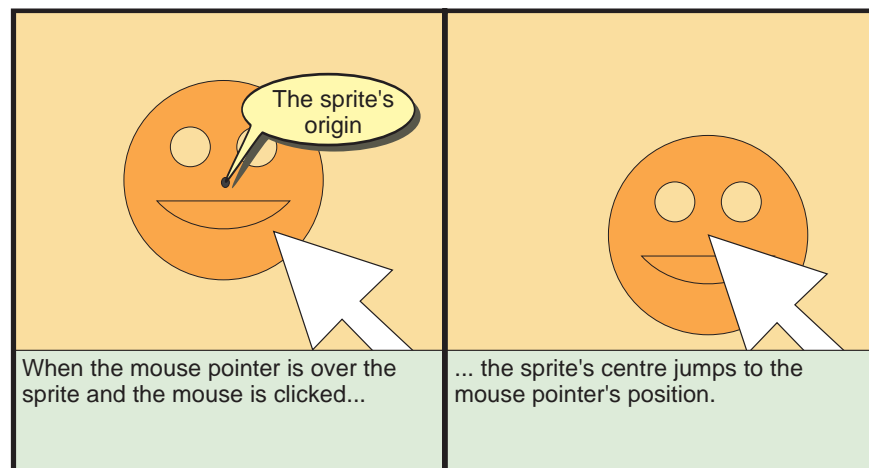
Activity Sup022

Place the mouse pointer in the lower right area of the smiley sprite and click the mouse button without any further movement of the mouse pointer.

What happens to the sprite when the mouse click occurs?

What happens when we first click the mouse within the sprite's area is shown in FIG-Sup09.

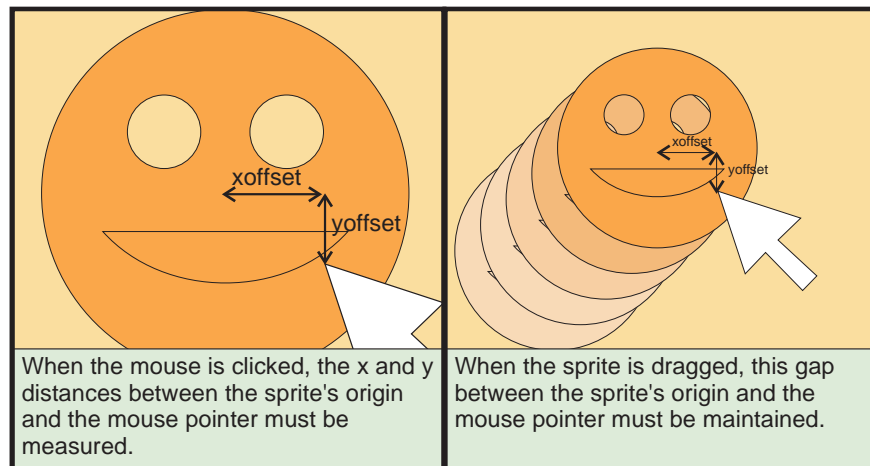
FIG-Sup09
The Sprite Jump



We don't want this jump to happen; the sprite should only move once the mouse pointer itself is moved. We can solve this problem by discovering how far the mouse pointer is from the centre of the sprite when the mouse is first clicked, and then maintain this difference as the sprite is dragged (see FIG-Sup10).

FIG-Sup10

Maintaining the Offset



The code required to record the x and y offsets from the sprite origin to the mouse pointer is shown below:

```
IF SPRITE HIT(InvisibleSpr, SmileySpr) AND (MOUSECLICK()=1)
  xoffset = SPRITE X(SmileySpr) - MOUSEX()
  yoffset = SPRITE Y(SmileySpr) - MOUSEY()
  WHILE MOUSECLICK() = 1
    SPRITE SmileySpr, MOUSEX()+xoffset, MOUSEY()+yoffset,
    ↵SmileySpr
  ENDWHILE
ENDIF
```

Activity Sup023

Modify your code to incorporate these changes to the main loop and check that the program worked correctly.

Handling Several Sprites

It is quite likely that your screen will contain several sprites, so when we perform dragging we first need to detect which sprite has been selected with a line such as:

```
spriteselectd = SPRITE HIT(InvisibleSpr, 0)
```

The variable *spriteselectd* is then used in all subsequent sprite-related statements:

```
IF (spriteselectd <> 0) AND (MOUSECLICK()=1)
  xoffset = SPRITE X(spriteselectd) - MOUSEX()
  yoffset = SPRITE Y(spriteselectd) - MOUSEY()
  WHILE MOUSECLICK() = 1
    SPRITE spriteselectd, MOUSEX()+xoffset, MOUSEY()+
    ↵yoffset,spriteselectd
  ENDWHILE
ENDIF
```

This ensures that the correct sprite is being dragged.

Activity Sup024

Modify your last program so that it contains three sprites (*SmileySpr*, *BoltSpr*, and *StarSpr*) which use the images *smiley.bmp*, *bolt.bmp* and *star.bmp*.

Change the main loop so that any of the three sprites may be dragged.

Disabling Dragging

In a complete game it is unlikely that every sprite on the screen will be draggable by the player. Sprites used to show buttons or messages will usually be stationary.

Controlling a Single Sprite

We can control the druggability of a sprite by associating a Boolean variable with each sprite. This variable can be set to *true* if the sprite can be dragged and *false* when the sprite is to maintain a fixed position on the screen.

Starting with a single sprite, the necessary code for setting up the variable is:

```
REM *** Boolean Values ***
#CONSTANT FALSE      0
#CONSTANT TRUE       1

REM *** Set drag control variable ***
draggable
draggable = TRUE
```

To make use of the variable we need to change the IF statement which checks if the mouse is attempting to drag a sprite from

```
IF (spritesselected<> 0) AND (MOUSECLICK())=1)
```

to

```
IF (spritesselected<> 0) AND (MOUSECLICK())=1) AND (draggable)
```

A complete program which tests this code is given in LISTING-Sup009.

LISTING-Sup009

Disabling Dragging

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg    1
#CONSTANT SmileyImg   2
REM *** Sprites ***
#CONSTANT InvisibleSpr 1
#CONSTANT SmileySpr    2
REM *** Boolean Values ***
#CONSTANT FALSE      0
#CONSTANT TRUE       1

REM *** Set drag control variable ***
draggable = FALSE

REM *** Load images ***
LOAD IMAGE "black.bmp",BlackImg
LOAD IMAGE "smiley.bmp",SmileyImg
```

Continued on next page

LISTING-Sup009
(continued)

Disabling Dragging

```

REM *** Load mouse tip sprite ***
SPRITE InvisibleSpr,0,0,BlackImg

REM *** Load regular sprites ***
SPRITE SmileySpr,200,300,SmileyImg
OFFSET SPRITE SmileySpr, SPRITE WIDTH(SmileySpr)/2,
↳SPRITE HEIGHT(SmileySpr)/2

REM *** Main loop ***
DO
  SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),BlackImg
  spritesselected = SPRITE HIT(InvisibleSpr, 0)
  IF (spritesselected<> 0) AND (MOUSECLICK()=1) AND (draggable)
    xoffset = SPRITE X(spritesselected) - MOUSEX()
    yoffset = SPRITE Y(spritesselected) - MOUSEY()
    WHILE MOUSECLICK() = 1
      SPRITE spritesselected, MOUSEX()+xoffset,
↳MOUSEY()+yoffset,spritesselected
    ENDWHILE
  ENDIF
LOOP

REM *** End program ***
END

```

Activity Sup025

Type in and test the program given in LISTING-Sup009 (*drag02.dbp*).

Can the smiley sprite be dragged?

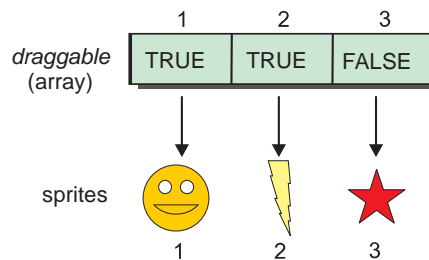
Change the value of *draggable* to TRUE. Does this allow the sprite to be dragged?

Controlling Multiple Sprites

If several sprites are being used, then we need a draggable variable for each sprite. The simplest way to do this is to use an array, with one element for each sprite (see FIG-Sup11).

FIG-Sup11

Representing the Drag Characteristic of three Sprites



In the program, we'll start by setting up image and sprite constants:

```

REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg      1
#CONSTANT SmileyImg     2
#CONSTANT BoltImg       3
#CONSTANT StarImg       4
REM *** Sprites ***
#CONSTANT InvisibleSpr  1
#CONSTANT SmileySpr    2
#CONSTANT BoltSpr      3
#CONSTANT StarSpr      4

```

Now we need to set up an array to control the draggability of the sprites:

```
REM *** Sprite Control Array ***  
DIM draggable(4)
```

We'll use element 2 to set the draggability of sprite 2, element 3 to control sprite 3, etc. Note that elements 0 and 1 of the array are not used.

To make sprites 2 and 3 draggable and sprite 4 stationary we would set the array with the lines:

```
REM *** Enable dragging for first two sprites ***  
draggable(2) = TRUE  
draggable(3) = TRUE  
draggable(4) = FALSE
```

The code only allows a sprite to be dragged if the following condition is true:

```
draggable(spriteselectd) = TRUE
```

The complete program is given in LISTING-Sup010.

LISTING-Sup010

Controlling the
Draggability of Three
Sprites

```
REM *** Constants ***  
REM *** Images ***  
#CONSTANT BlackImg      1  
#CONSTANT SmileyImg     2  
#CONSTANT BoltImg       3  
#CONSTANT StarImg       4  
  
REM *** Sprites ***  
#CONSTANT InvisibleSpr  1  
#CONSTANT SmileySpr     2  
#CONSTANT BoltSpr       3  
#CONSTANT StarSpr       4  
  
REM *** Boolean Values ***  
#CONSTANT FALSE         0  
#CONSTANT TRUE          1  
  
REM *** Sprite Control Array ***  
DIM draggable(4)  
  
REM *** Load images ***  
LOAD IMAGE "black.bmp",BlackImg  
LOAD IMAGE "smiley.bmp",SmileyImg  
LOAD IMAGE "bolt.bmp",BoltImg  
LOAD IMAGE "star.bmp",StarImg  
  
REM *** Load mouse tip sprite ***  
SPRITE InvisibleSpr,0,0,BlackImg  
  
REM *** Load regular sprites ***  
SPRITE SmileySpr,200,300,SmileyImg  
SPRITE BoltSpr,300,300,BoltImg  
SPRITE StarSpr,400,300,StarImg  
OFFSET SPRITE SmileySpr, SPRITE WIDTH(SmileySpr)/2,  
↳SPRITE HEIGHT(SmileySpr)/2  
OFFSET SPRITE BoltSpr, SPRITE WIDTH(BoltSpr)/2,  
↳SPRITE HEIGHT(BoltSpr)/2  
OFFSET SPRITE StarSpr, SPRITE WIDTH(StarSpr)/2,  
↳SPRITE HEIGHT(StarSpr)/2
```

continued on next page

LISTING-Sup010

(continued)

Controlling the
Draggability of Three
Sprites

```
REM *** Enable dragging for first two sprites ***
draggable(2) = TRUE
draggable(3) = TRUE
draggable(4) = FALSE

REM *** Main loop ***
DO
  SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),BlackImg
  spriteselcted = SPRITE HIT(InvisibleSpr, 0)
  IF (spriteselcted<> 0) AND (MOUSECLICK())=1)
    IF draggable(spriteselcted) = TRUE
      xoffset = SPRITE X(spriteselcted) - MOUSEX()
      yoffset = SPRITE Y(spriteselcted) - MOUSEY()
      WHILE MOUSECLICK() = 1
        SPRITE spriteselcted, MOUSEX()+xoffset,
          MOUSEY()+yoffset,spriteselcted
      ENDWHILE
    ENDIF
  ENDIF
LOOP

REM *** End program ***
END
```

Activity Sup026

Type in and test the program given in LISTING-Sup010 (*drag03.dbp*).

Creating a Jigsaw Game

Introduction

In this section we are going to create a jigsaw-type game by taking an existing picture and splitting it into a set of smaller squares, with each square being displayed on a sprite. The squares can then be dragged into position to recreate the original picture.

To help the player get started, the four corner pieces will be correctly positioned at the start of the game and cannot be moved.

We can describe the overall logic of the game as follows:

```
Load up the image used and split it into pieces
Place the four corner pieces in position and scatter other pieces about the screen
REPEAT
    Allow user to move a piece
UNTIL all pieces are in place
```

We can see a screen dump from a partially completed jigsaw in FIG-Sup12.

FIG-Sup12

Jigsaw Screen Shot



Game Constants and Screen Setup

Initially the game needs two pictures: the black image used by the mouse pointer sprite and the bitmap used in the jigsaw. The black image will, as usual, be used to create a sprite. IDs for these components can be set up using the lines:

```
REM *** Constants ***
REM *** Bitmaps ***
#CONSTANT wholeBmp      1
REM *** Images ***
#CONSTANT BlackImg     1
REM *** Sprites ***
#CONSTANT BlackSpr     1
```

The small function used to set up the screen resolution can also seed the random number generator - since we'll be using random numbers to position the pieces of the jigsaw on the screen - and load the image used at the mouse pointer tip. The code for the routine is:

```

FUNCTION GameSetUp
  REM *** Set screen resolution ***
  SET DISPLAY MODE 1280,1024,32
  BACKDROP ON
  REM *** Set random number generator ***
  RANDOMIZE TIMER()
  REM *** Load black image ***
  LOAD IMAGE "black.bmp",BlackImg
ENDFUNCTION

```

Splitting Up the Main Picture

We need to load the image used for the jigsaw and then split it into the jigsaw pieces. Unlike a physical jigsaw, the pieces here will be simple rectangles and also be correctly oriented - no upside-down or back-to-front pieces.

We have to decide on the size of the pieces and, since we may want to vary this, it would be best to define the dimensions using named constants. Here we'll use 64 x 64 pieces:

```

REM *** Numeric Values ***
#CONSTANT piecewidth 64
#CONSTANT pieceheight 64

```

Creating the pieces will be performed by a function which first loads the jigsaw image into bitmap area 1 (where the image will not be seen by the player). Once loaded the size of the image is examined to determine the number of rows and columns the image can be split into. For example using a 640 x 320 pixel image, the image would be split into 10 rows by 5 columns giving us a 50 piece jigsaw. The routine uses the following logic:

```

Load the jigsaw image into bitmap area 1
Determine the number of rows and columns
Start sprite number at 2
FOR each row DO
  FOR each column DO
    Get section of image
    IF it's not a corner piece THEN
      Calculate random position
      Create sprite
      Reposition sprite origin at its centre
    ELSE
      Create sprite at corner position
    ENDIF
    Make sprite opaque
    Increment sprite number
  ENDFOR
ENDFOR
Reset output area to screen

```

Notice that the corner pieces are placed in their correct positions. This gives the player a help in getting started since they will not have seen the completed image.

The code for this function is:

```

FUNCTION CreatePieces(filename$)
  REM *** Load the jigsaw image ***

```

```

REM *** into bitmap area 1 ***
LOAD BITMAP filename$,1
REM *** Calculate number of rows and columns ***
rows = BITMAP HEIGHT(1)/pieceheight
cols = BITMAP WIDTH(1)/piecewidth
REM *** Split image into pieces ***
REM *** creating a sprite for each ***
spriteno = 2
FOR r = 0 TO rows-1
  FOR c = 0 TO cols-1
    REM *** Capture a section of the bitmap ***
    GET IMAGE spriteno,c*piecewidth,r*pieceheight,
    ↵c*piecewidth+piecewidth,r*pieceheight+pieceheight,1
    REM *** IF it's not a corner piece ***
    IF NOT ((r = 0 AND c = 0) OR (r = 0 AND c=cols-1)
    ↵OR(r = rows-1 AND c=0)OR(r = rows-1 AND c=cols-1))
      REM *** Generate a random position for piece ***
      y = RND(900)
      IF y < rows*pieceheight+pieceheight/2
        x = cols*piecewidth+piecewidth/2+
        ↵RND(1100-(cols*piecewidth))
      ELSE
        x = RND(1200)
      ENDIF
      REM *** Create a sprite from bitmap section ***
      SPRITE spriteno,x,y,spriteno
      REM *** Position sprite origin at its centre ***
      OFFSET SPRITE spriteno, piecewidth/2,piecewidth/2
    ELSE
      REM *** Position corner piece ***
      SPRITE spriteno,c*piecewidth,r*pieceheight,
      ↵spriteno
    ENDIF
    REM *** No transparency ***
    SET SPRITE spriteno,1,0
    REM *** Increment sprite number used ***
    INC spriteno
  NEXT c
NEXT r
REM *** Restore screen as output device ***
SET CURRENT BITMAP 0
ENDFUNCTION

```

Activity Sup027

Create a program (*jigsaw.dbp*) using the code already given, which initialises the screen and displays the pieces of the jigsaw.

Use the image *costumes.jpg*, making sure that you have copied it into the project directory.

Dragging Pieces

Dragging a piece of the jigsaw about the screen involves the logic covered earlier. The only refinement is that the piece must drop into an exact position when played within the area of the screen used to construct the jigsaw. This is done by modifying the dragged sprite's position after it has been released using the lines:

```

REM *** IF the piece is dropped within jigsaw area ***
IF MOUSEX(<)(cols)*piecewidth AND MOUSEY(<)(rows)*pieceheight
  REM *** Position it exactly ***
  SPRITE spriteselectd,
  ↵SPRITE X(spriteselectd)/piecewidth*piecewidth

```

```

    ↵+piecewidth/2,
    ↵SPRITE Y(spriteselected)/pieceheight*pieceheight
    ↵+pieceheight/2,spriteselected
ENDIF

```

In the code, *spriteselected* is the ID of the sprite being dragged. We'll make use of this code in the function that follows.

Dragging a piece about the screen will be handled by a single function, *HandlePieceMovement()*, which makes use of the following logic:

```

Move mouse sprite
Check for collision with mouse sprite
IF there's been a collision and mouse button pressed THEN
    Calculate mouse offset from sprite centre
    WHILE mouse button pressed DO
        Move sprite
    ENDWHILE
    IF sprite within jigsaw area THEN
        Position it exactly
    ENDIF
ENDIF
ENDIF

```

and is coded as:

```

FUNCTION HandlePieceMovement()
    REM *** Move mouse sprite ***
    SPRITE BlackSpr,MOUSEX(),MOUSEY(),BlackImg
    REM *** Check for sprite collision ***
    spriteselected = SPRITE HIT(BlackSpr,0)
    REM *** IF collision & mouse button pressed THEN ***
    IF (spriteselected<> 0) AND (MOUSECLICK()=1)
        REM *** Calculate the sprite/mouse offsets ***
        xoffset = SPRITE X(spriteselected) - MOUSEX()
        yoffset = SPRITE Y(spriteselected) - MOUSEY()
        REM *** Drag sprite until mouse button released ***
        WHILE MOUSECLICK() = 1
            SPRITE spriteselected, MOUSEX()+xoffset, MOUSEY()+
            ↵yoffset,spriteselected
        ENDWHILE
        REM *** IF sprite in jigsaw area, position exactly ***
        IF MOUSEX()<cols*piecewidth AND MOUSEY()<rows*pieceheight
            SPRITE spriteselected,
            ↵SPRITE X(spriteselected)/piecewidth*piecewidth+
            ↵piecewidth/2,
            SPRITE Y(spriteselected)/pieceheight*pieceheight
            ↵+pieceheight/2,spriteselected
        ENDIF
    ENDIF
ENDIF
ENDFUNCTION

```

Since the function makes use of the *rows* and *cols* variables calculated in the *CreatePieces()* function, we'll need to make those variables global:

```

REM *** Global Variables ***
GLOBAL rows
GLOBAL cols

```

Activity Sup028

Modify your previous program adding the *HandlePieceMovement()* function and the global variable declarations.

Change the main section in the program to the following logic:

```
GameSetUp()  
CreatePieces("costumes.jpg")  
DO  
    HandlePieceMovement()  
LOOP  
END
```

What problem occurs when you try to move a corner piece of the jigsaw?

Fixing the Corner Pieces

By creating an array to record which pieces can be dragged, we can make sure the corner pieces cannot be moved. The array will have an element for each sprite similar to that used in LISTING-Sup010.

The size of the array depends on the number of pieces in the array (*rows * cols*), but since we don't know these values at the start of the program, we'll content ourselves by creating an array which is large enough for any jigsaw:

```
DIM pieces(400)
```

A piece that can be moved will have its entry in the pieces array set to 1 and those that cannot be moved (the corner pieces) will have their array entry set to zero. We can create a function - *SetDraggable()* - to perform the initialisation of the pieces array:

```
FUNCTION SetDraggable()  
    REM *** Set every pieces as draggable ...***  
    FOR c = 3 TO rows*cols+1  
        pieces(c) = 1  
    NEXT c  
    REM *** ... except the four corner pieces ***  
    pieces(2) = 0  
    pieces(cols+1) = 0  
    pieces((rows-1)*cols+2) = 0  
    pieces(rows*cols+1) = 0  
ENDFUNCTION
```

Notice that, since sprite ID 1 is used for the sprite at the mouse pointer, the first piece in the jigsaw uses ID 2.

We also need to modify the IF statement in *HandlePieceMovement()*, changing it from

```
IF (spritesselected<> 0) AND (MOUSECLICK())=1)
```

to

```
IF (spritesselected<> 0) AND (MOUSECLICK())=1)  
AND pieces(spritesselected) > 0
```

Finally, we just need a call to the new function in the main section of the program placed immediately after the call to *CreatePieces()* and the updated version is ready to test.

Activity Sup029

Update your last program to incorporate the changes discussed above.

Sprite Priority

When you click the mouse on a group of overlapping sprites, the sprite with the lowest ID will be selected. For example, if sprites with IDs 7, 20 and 87, appear on the same area of the screen, clicking within this group will select sprite 7. However, it is the sprite with the largest ID which appears to be on top of the pile - sprite 87 in our example - so you won't see which sprite you are dragging until it is dragged clear of the others within its area.

Activity Sup030

Run your program and click on a group of overlapping sprites and drag the selected sprite out from the group. Note that it is not the top sprite that is moved.

Although we can't easily do anything about which sprite is selected, we can at least make the selected sprite appear on the top of the pile by telling DarkBASIC Pro to draw the selected sprite last, thereby making it appear at the top of the pile. This is done using the statement

```
REM *** Draw selected sprite on top ***  
SET SPRITE PRIORITY spritesselected,1
```

when the sprite is first selected.

Once the sprite is dropped, the priority should be returned to the default zero value with the line:

```
REM *** Default sprite order ***  
SET SPRITE PRIORITY spritesselected,0
```

Activity Sup031

Insert the new lines at the appropriate points in the function *HandlePieceMovement()*.

Test the effect of the changes.

Losing Pieces

It is possible to "lose" a piece of the jigsaw by placing it in the exact same position as an existing piece within the jigsaw area. If the new piece has a lower ID than the piece already at that position, the newly positioned piece will be lost under the existing piece. But the worst case scenario is when a piece with a higher ID number is placed under a corner piece!

Activity Sup032

Run your program and try placing a piece of the jigsaw on the bottom-right corner piece. Can you retrieve the piece?

Place a piece on the top-left corner of the jigsaw. Can the piece be retrieved?

To handle these situations we need to record which positions within the jigsaw area are already occupied and prevent a second piece being placed at the same position.

To do this we'll create an array representing the board:

```
DIM board(400)
```

Each element in this array represents a position within the jigsaw. We'll store the ID of a piece placed within the jigsaw area in the corresponding element of the array. If the piece is placed in its correct position then the ID and element number will match. For example, piece 2 (the top-left corner piece) will have its ID (2) stored in element 2 of the array. Where no piece is held at a given position, the corresponding element will be set to zero.

At the start of the game, the *board* array will have all elements set to zero except for those containing the corner pieces. This task is performed by the *InitialiseBoard()* function which is coded as:

```
FUNCTION InitialiseBoard()  
  REM *** Board is empty ... ***  
  FOR c = 2 TO rows*cols+1  
    board(c)=0  
  NEXT c  
  REM *** ... except for corners ***  
  board(2)=2  
  board(cols+1)=cols+1  
  board((rows-1)*cols+2) = (rows-1)*cols+2  
  board(rows*cols+1) = rows*cols+1  
ENDFUNCTION
```

Now, when a piece is dragged onto the board, it will only remain where it is dropped if that position is empty. This means removing from *HandlePieceMovement()* the code which places the dropped piece:

```
REM *** IF sprite in jigsaw area, position exactly ***  
IF MOUSEX()<cols*piecewidth AND MOUSEY()<rows*pieceheight  
  SPRITE spriteselectd,  
  ↵SPRITE X(spriteselectd)/piecewidth*piecewidth+  
  ↵piecewidth/2,  
  SPRITE Y(spriteselectd)/pieceheight*pieceheight  
  ↵+pieceheight/2,spriteselectd  
ENDIF
```

and replacing it with a call to a new function which positions the piece or returns it to its original position.

The routine, *DropPiece()*, uses the following logic:

```
IF piece placed outside jigsaw area THEN  
  Exit routine  
ENDIF  
Position sprite at exact position within jigsaw  
Calculate piece's position in board array  
IF that position is empty THEN
```

```

        Store piece's ID at that element in board array
ELSE
    Return piece to its original position
ENDIF

```

To perform this logic, the function needs three items of data: the ID of the piece being dropped, and the original coordinates of the piece before it was moved. These are supplied to the routine in the way of parameters

```

FUNCTION DropPiece(spriteselected, oldscreenx, oldscreeny)
    REM *** IF piece not in build area, return ***
    IF MOUSEX() >= (cols)*piecewidth OR
    MOUSEY() >= (rows)*pieceheight
        EXITFUNCTION
    ENDIF
    REM *** Position piece ***
    SPRITE spriteselected,
    ↵SPRITE X(spriteselected)/piecewidth*piecewidth+piecewidth/2,
    ↵SPRITE Y(spriteselected)/pieceheight*pieceheight
    ↵+pieceheight/2, spriteselected
    REM *** Calculate column(x) and row(y) position of piece ***
    x = (SPRITE X(spriteselected) - piecewidth/2)/piecewidth
    y = (SPRITE Y(spriteselected) - pieceheight/2)/pieceheight
    positiononboard = y*cols+x+2
    REM *** IF jigsaw position empty THEN ***
    IF board(positiononboard)=0
        REM *** Record sprite's ID in board array ***
        board(positiononboard) = spriteselected
    ELSE
        REM *** ELSE return piece to its original position ***
        SPRITE spriteselected, oldscreenx, oldscreeny,
        ↵spriteselected
    ENDIF
ENDFUNCTION

```

Notice that the function requires the original coordinates of the sprite to be passed as parameters and so *HandlePieceMovement()* needs to record this position before it begins moving a piece. This can be done with the lines

```

    REM *** Store sprite's start position ***
    oldx = SPRITE X(spriteselected)
    oldy = SPRITE Y(spriteselected)

```

and the call to *DropPiece()* will be:

```

    DropPiece(spriteselected, oldx, oldy)

```

Activity Sup033

Update your program to include these new routines (*InitialiseBoard()* and *DropPiece()*).

Modify the main logic to call *InitialiseBoard()* immediately after the *SetDraggable()* call.

Modify *HandlePieceMovement()* to store the original coordinates of the piece being moved and call *DropPiece()*.

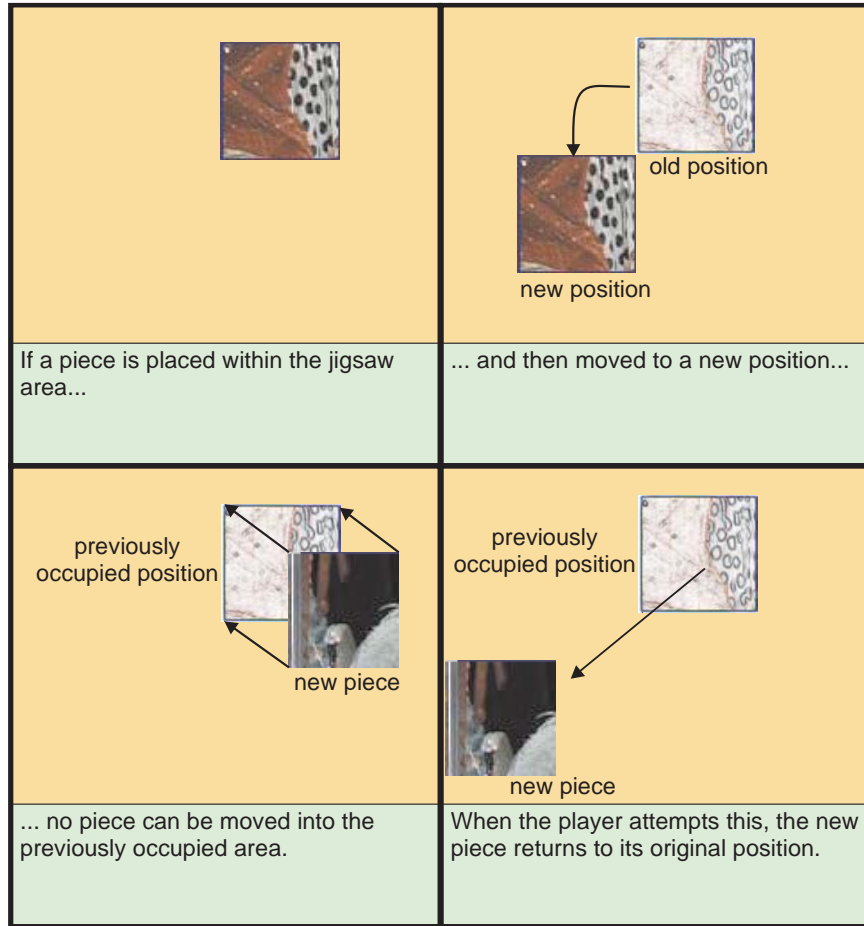
What happens when you attempt to place a second piece over an existing piece within the jigsaw area?

A Problem with Vacated Areas

Our latest code has a problem which might not be immediately obvious; when a position within the jigsaw receives a piece and that piece is then moved to a new position, the piece's original place cannot receive a new piece (see FIG-Sup13).

FIG-Sup13

Trying to Store a Piece in a Previously Occupied Position



Activity Sup034

Run your program again and check for the problem described above.

To allow a position within the jigsaw to accept a new piece, we'll need to make a few modifications. Firstly, we must use the *pieces* array to record the position within the board at which a piece is stored. This means adding the following lines to the *DropPiece()* function.

```
REM *** board position in pieces array ***
pieces(spriteselected)=positiononboard
```

Now, an element of the *pieces* array can contain one of the following values:

Value	Meaning
0	Piece cannot be moved
1	Piece is outside the jigsaw area
>1	Piece is inside the jigsaw area

Next we need to modify the *HandlePieceMovement()* routine to include the code

```
REM *** IF piece in jigsaw area THEN ***
IF pieces(spritesselected)> 1
  REM *** Remove it from board array ***
  board(pieces(spritesselected)) = 0
  REM *** and mark as outside jigsaw area ***
  pieces(spritesselected)=1
ENDIF
```

Activity Sup035

Modify your functions *DropPiece()* and *HandlePieceMovement()* using the code given above.

Run the program and check that you have eliminated the problem highlighted in the last Activity.

Restoring Position

The *DropPiece()* routine returns a piece to its original position when the player attempts to move it to an illegal position. However, the jump from new to old position is instantaneous; a better option would be to have the piece slide back to its original position. To do this we introduce a new function, *SlideBack()*, which slides a piece back to its original position using 20 separate stopping positions to create an animated slide back to the starting position. The function takes as parameters the ID of the piece to be returned and the coordinates of the position to which it is to be returned. The function uses the following logic:

```
Determine the current position of the piece
Calculate the step size along the X-axis
Calculate the step size along the Y-axis
FOR 20 times DO
  Subtract step sizes from current position
  Reposition piece at recalculated position
  Wait 10 milliseconds
ENDFOR
Position piece at its original position
```

The code for the routine is:

```
FUNCTION SlideBack(spritesselected, oldscreenx,oldscreeny)
  REM *** Determine current position of piece ***
  xnow# = SPRITE X(spritesselected)
  ynow# = SPRITE Y(spritesselected)
  REM *** Calculate X and Y step sizes ***
  xstep# = (xnow# - oldscreenx)/20.0
  ystep# = (ynow# - oldscreeny)/20.0
  REM FOR 20 times DO ***
  FOR c = 1 TO 20
    REM *** Calculate next position ***
    xnow# = xnow# - xstep#
    ynow# = ynow# - ystep#
    REM *** Reposition piece ***
    SPRITE spritesselected, xnow#, ynow#,spritesselected
    WAIT 10
  NEXT c
  REM *** Position piece at its original position ***
  SPRITE spritesselected, oldscreenx,oldscreeny,spritesselected
ENDFUNCTION
```

If we think about it, we might expect the piece to be in its old position after the 20 moves, but because of rounding errors when dealing with real numbers, the position of the piece will almost certainly be a little out at the end of the 20 moves. Although this doesn't matter too much if the piece has returned to a position outside the jigsaw area, we need to ensure exact positioning within the jigsaw area and hence we include a final sprite moving line to place it exactly at its original position.

Activity Sup036

Modify your program to include the new routine and include a call to it at the appropriate point in your program. Remove existing lines as required.

Detecting Completion

At the moment the only way to stop the program is to press the ESCAPE key, but obviously, it would be better if the program could detect when the jigsaw has been successfully completed.

The first criteria for completion is that the jigsaw area contains every piece of the jigsaw and the second is that all pieces are in the correct position.

We can check for the first by creating another global variable, *piecesplaced*, which starts with a value of 4 (the corner pieces being in place at the start of the game):

```
GLOBAL piecesplaced = 4
```

Now each time a piece is dropped within the jigsaw area we must add 1 to *piecesplaced*, and each time a piece already within the jigsaw area is lifted, we must subtract 1 from *piecesplaced*.

Once *piecesplaced* contains a value equal to the number of pieces in the jigsaw (*rows*cols*), then we must test to see if each piece is in the correct position. To do this all we need to do is check that every element of board contains the correct piece ID number. All that is required for this is that position *c* within the array contains the value *c*.

By adding a routine, *IsComplete()*, which returns 1 if every piece of the jigsaw is in place and 0 if they are not, we can replace the DO LOOP structure in the main section of the program to a REPEAT..UNTIL structure.

The code for *IsComplete()* is::

```
FUNCTION IsComplete()  
  REM *** IF all pieces not in place, return 0 ***  
  IF piecesplaced <> rows*cols  
    EXITFUNCTION 0  
  ENDF  
  REM *** IF any piece not in correct position, return 0 ***  
  FOR c = 2 TO rows*cols+1  
    IF board(c) <> c  
      EXITFUNCTION 0  
    ENDF  
  NEXT c  
  REM *** IF both tests above are OK, return 1 ***  
ENDFUNCTION 1
```

Activity Sup037

Add the global variable *piecesplaced* to your code, initialising it to the value 4.

Modify *HandlePieceMovement()* so that *piecesplaced* is decremented when a piece from within the jigsaw area is lifted.

Modify *DropPiece()* so that *piecesplaced* is incremented when a piece is dropped in the jigsaw area.

Add *IsComplete()* to your program.

Modify the code of the main logic so that the loop terminates when the puzzle is complete.

At last, we have the completed version of the program! The complete code is given in LISTING-Sup011.

LISTING-Sup011

The Complete Jigsaw
Program

```
REM *** Constants ***
REM *** Bitmaps ***
#CONSTANT wholeBmp      1
REM *** Images ***
#CONSTANT BlackImg     1
REM *** Sprites ***
#CONSTANT BlackSpr     1
REM *** Numeric Values ***
#CONSTANT piecewidth   64
#CONSTANT pieceheight  64
REM *** Global Variables ***
GLOBAL rows
GLOBAL cols
GLOBAL piecesplaced = 4

DIM pieces(400)
DIM board(400)

GameSetUp()
CreatePieces("costumes.jpg")
SetDraggable()
InitialiseBoard()
REPEAT
  HandlePieceMovement()
UNTIL IsComplete()
WAIT KEY
END

FUNCTION GameSetUp()
  REM *** Set screen resolution ***
  SET DISPLAY MODE 1280,1024,32
  BACKDROP ON
  REM *** Set random number generator ***
  RANDOMIZE TIMER()
  REM *** Load black image ***
  LOAD IMAGE "black.bmp",BlackImg
ENDFUNCTION

FUNCTION InitialiseBoard()
  REM *** Board is empty ... ***
  FOR c = 2 TO rows*cols+1
    board(c)=0
  NEXT c
```

continued on next page

LISTING-Sup011

(continued)

The Complete Jigsaw
Program

```

REM *** ... except for corners ***
board(2)=2
board(cols+1)=cols+1
board((rows-1)*cols+2) = (rows-1)*cols+2
board(rows*cols+1) = rows*cols+1
ENDFUNCTION

FUNCTION CreatePieces(filename$)
REM *** Load the jigsaw image ***
REM *** into bitmap area 1 ***
LOAD BITMAP filename$,1
REM *** Calculate number of rows and columns ***
rows = BITMAP HEIGHT(1)/pieceheight
cols = BITMAP WIDTH(1)/piecewidth
REM *** Split image into pieces ***
REM *** creating a sprite for each ***
spriteno = 2
FOR r = 0 TO rows-1
FOR c = 0 TO cols-1
REM *** Capture a section of the bitmap ***
GET IMAGE spriteno,c*piecewidth,r*pieceheight,
↵c*piecewidth+piecewidth,r*pieceheight+pieceheight,1
REM *** IF it's not a corner piece ***
IF NOT ((r = 0 AND c = 0) OR (r = 0 AND c = cols-1)
↵OR (r = rows-1 AND c=0) OR (r = rows-1 AND c=cols-1))
REM *** Generate a random position for piece ***
y = RND(900)
IF y < rows*pieceheight+pieceheight/2
x = cols*piecewidth+piecewidth/2+
↵RND(1100-(cols*piecewidth))
ELSE
x = RND(1200)
ENDIF
REM *** Create a sprite from bitmap section ***
SPRITE spriteno,x,y,spriteno
REM *** Position sprite origin at its centre ***
OFFSET SPRITE spriteno, piecewidth/2,piecewidth/2
ELSE
REM *** Position corner piece ***
SPRITE spriteno,c*piecewidth,r*pieceheight,spriteno
ENDIF
REM *** No transparency ***
SET SPRITE spriteno,1,0
REM *** Increment sprite number used ***
INC spriteno
NEXT c
NEXT r
REM *** Restore screen as output device ***
SET CURRENT BITMAP 0
ENDFUNCTION

FUNCTION HandlePieceMovement()
REM *** Move mouse sprite ***
SPRITE BlackSpr,MOUSEX(),MOUSEY(),BlackImg
REM *** Check for sprite collision ***
spriteselcted = SPRITE HIT(BlackSpr,0)
REM *** IF collision & mouse button pressed THEN ***
IF (spriteselcted<> 0) AND (MOUSECLICK())=1) AND
↵pieces(spriteselcted) > 0
REM *** Calculate the sprite/mouse offsets ***
xoffset = SPRITE X(spriteselcted) - MOUSEX()
yoffset = SPRITE Y(spriteselcted) - MOUSEY()
REM *** Store sprite's start position ***
oldx = SPRITE X(spriteselcted)

```

continued on next page

LISTING-Sup011

(continued)

The Complete Jigsaw
Program

```
oldy = SPRITE Y(spriteselectd)
REM *** Draw selected sprite on top ***
SET SPRITE PRIORITY spriteselectd,1
REM *** IF piece in jigsaw area THEN ***
IF pieces(spriteselectd)> 1
    REM *** Remove it from board array ***
    board(pieces(spriteselectd)) = 0
    REM *** and mark as outside jigsaw area ***
    pieces(spriteselectd)=1
    REM *** Reduce no. of pieces in area ***
    DEC piecesplacd
ENDIF
REM *** Drag sprite until mouse button released ***
WHILE MOUSECLICK() = 1
    SPRITE spriteselectd, MOUSEX()+xoffset,
    ↵MOUSEY()+yoffset,spriteselectd
ENDWHILE
DropPieced(spriteselectd,oldx,oldy)
REM *** Return to default drawing order ***
SET SPRITE PRIORITY spriteselectd,0
ENDIF
ENDFUNCTION

FUNCTION SetDraggabled()
    REM *** Set every pieces as draggable ...***
    FOR c = 3 TO rows*cols+1
        pieces(c) = 1
    NEXT c
    REM *** ... except the four corner pieces ***
    pieces(2) = 0
    pieces(cols+1) = 0
    pieces((rows-1)*cols+2) = 0
    pieces(rows*cols+1) = 0
ENDFUNCTION

FUNCTION DropPieced(spriteselectd, oldscreenx, oldscreeny)
    REM *** IF piece not in build area, return ***
    IF MOUSEX()>=(cols)*piecewidth OR MOUSEY()>=(rows)*piecheight
        EXITFUNCTION
    ENDIF
    REM *** Position piece ***
    SPRITE spriteselectd,SPRITE X(spriteselectd)/piecewidth
    ↵*piecewidth+piecewidth/2,SPRITE Y(spriteselectd)/piecheight
    ↵*piecheight+piecheight/2,spriteselectd
    REM *** Calculate column(x) and row(y) position of piece ***
    x = (SPRITE X(spriteselectd)- piecewidth/2)/piecewidth
    y = (SPRITE Y(spriteselectd)- piecheight/2)/piecheight
    positiononboard = y*cols+x+2
    REM *** IF jigsaw position empty THEN ***
    IF board(positiononboard)=0
        REM *** Record sprite's ID in board array ***
        board(positiononboard) = spriteselectd
        REM *** board position in pieces array ***
        pieces(spriteselectd)=positiononboard
        REM *** Increment no. of pieces in area
        INC piecesplacd
    ELSE
        REM *** ELSE return piece to its original position ***
        slideBack(spriteselectd,oldscreenx, oldscreeny)
    ENDIF
ENDFUNCTION
```

continued on next page

LISTING-Sup011 (continued)

The Complete Jigsaw
Program

```
FUNCTION SlideBack(spriteselected,oldscreenx,oldscreeny)
  REM *** Determine current position of piece ***
  xnow# = SPRITE X(spriteselected)
  ynow# = SPRITE Y(spriteselected)
  REM *** Calculate X and Y step sizes ***
  xstep# = (xnow# - oldscreenx)/20.0
  ystep# = (ynow# - oldscreeny)/20.0
  REM FOR 20 times DO ***
  FOR c = 1 TO 20
    REM *** Calculate next position ***
    xnow# = xnow# - xstep#
    ynow# = ynow# - ystep#
    REM *** Reposition piece ***
    SPRITE spriteselected, xnow#, ynow#,spriteselected
    WAIT 10
  NEXT c
  REM *** Position piece at its original position ***
  SPRITE spriteselected, oldscreenx,oldscreeny,spriteselected
ENDFUNCTION

FUNCTION IsComplete()
  REM *** IF all pieces not in place, return 0 ***
  IF piecesplaced <> rows*cols
    EXITFUNCTION 0
  ENDIF
  REM *** IF any piece not in correct position, return 0 ***
  FOR c = 2 TO rows*cols+1
    IF board(c) <> c
      EXITFUNCTION 0
    ENDIF
  NEXT c
  REM *** IF both tests above are OK, return 1 ***
ENDFUNCTION 1
```

Creating Your Own Images

Of course, you will want to use your own images in the jigsaw. Just make sure that you've resized them to be some multiple of 64 in both width and height.

Solutions

Activity Sup019

No solution required.

Activity Sup020

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg 1
#CONSTANT SmileyImg 2
REM *** Sprites ***
#CONSTANT InvisibleSpr 1
#CONSTANT SmileySpr 2
REM *** Load image ***
LOAD IMAGE "black.bmp",BlackImg
LOAD IMAGE "smiley.bmp",SmileyImg
REM *** Load sprite ***
SPRITE InvisibleSpr,0,0,BlackImg
SPRITE SmileySpr,400,300,SmileyImg
REM *** Main loop ***
DO
    SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),
    ↵BlackImg
    IF SPRITE HIT(InvisibleSpr, SmileySpr)
        PRINT "HIT"
    ENDIF
LOOP
REM *** End program ***
END
```

Activity Sup021

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg 1
#CONSTANT SmileyImg 2
REM *** Sprites ***
#CONSTANT InvisibleSpr 1
#CONSTANT SmileySpr 2
REM *** Load image ***
LOAD IMAGE "black.bmp",BlackImg
LOAD IMAGE "smiley.bmp",SmileyImg
REM *** Load sprite ***
SPRITE InvisibleSpr,0,0,BlackImg
SPRITE SmileySpr,400,300,SmileyImg
REM *** Move sprite's origin to centre ***
OFFSET SPRITE SmileySpr,
↵SPRITE WIDTH(SmileySpr)/2,
↵SPRITE HEIGHT(SmileySpr)/2
REM *** Main loop ***
DO
    SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),
    ↵BlackImg
    IF (SPRITE HIT(InvisibleSpr, SmileySpr))
        ↵AND (MOUSECLICK() = 1)
        SPRITE SmileySpr, MOUSEX(), MOUSEY(),
        ↵SmileySpr
    ENDIF
LOOP
REM *** End program ***
END
```

Activity Sup022

The sprite jumps to a new position such that its origin (centre) is now at the tip of the mouse pointer.

Activity Sup023

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg 1
#CONSTANT SmileyImg 2
REM *** Sprites ***
#CONSTANT InvisibleSpr 1
#CONSTANT SmileySpr 2
REM *** Load images ***
LOAD IMAGE "black.bmp",BlackImg
LOAD IMAGE "smiley.bmp",SmileyImg
REM *** Load mouse tip sprite ***
SPRITE InvisibleSpr,0,0,BlackImg
REM *** Load regular sprite ***
SPRITE SmileySpr,400,300,SmileyImg
OFFSET SPRITE SmileySpr,
↵SPRITE WIDTH(SmileySpr)/2,
↵SPRITE HEIGHT(SmileySpr)/2
REM *** Main loop ***
DO
    SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),
    ↵BlackImg
    IF SPRITE HIT(InvisibleSpr,SmileySpr)
        ↵AND (MOUSECLICK()=1)
        xoffset = SPRITE X(SmileySpr)-MOUSEX()
        yoffset = SPRITE Y(SmileySpr)-MOUSEY()
        WHILE MOUSECLICK() = 1
            SPRITE SmileySpr, MOUSEX()+xoffset,
            ↵MOUSEY()+yoffset,SmileySpr
        ENDWHILE
    ENDIF
LOOP
REM *** End program ***
END
```

Activity Sup024

```
REM *** Constants ***
REM *** Images ***
#CONSTANT BlackImg 1
#CONSTANT SmileyImg 2
#CONSTANT BoltImg 3
#CONSTANT StarImg 4
REM *** Sprites ***
#CONSTANT InvisibleSpr 1
#CONSTANT SmileySpr 2
#CONSTANT BoltSpr 3
#CONSTANT StarsSpr 4
REM *** Load images ***
LOAD IMAGE "black.bmp",BlackImg
LOAD IMAGE "smiley.bmp",SmileyImg
LOAD IMAGE "bolt.bmp",BoltImg
LOAD IMAGE "star.bmp",StarImg
REM *** Load mouse tip sprite ***
SPRITE InvisibleSpr,0,0,BlackImg
REM *** Load regular sprites ***
SPRITE SmileySpr,200,300,SmileyImg
OFFSET SPRITE SmileySpr, SPRITE
↵WIDTH(SmileySpr)/2, SPRITE
↵HEIGHT(SmileySpr)/2
SPRITE BoltSpr,300,300,BoltImg
OFFSET SPRITE BoltSpr,
↵SPRITE WIDTH(BoltSpr)/2,
↵SPRITE HEIGHT(BoltSpr)/2
SPRITE StarsSpr,400,300,StarImg
OFFSET SPRITE StarsSpr,
↵SPRITE WIDTH(StarSpr)/2,
↵SPRITE HEIGHT(StarSpr)/2
REM *** Main loop ***
DO
    SPRITE InvisibleSpr,MOUSEX(),MOUSEY(),
    ↵BlackImg
```

```

spritesselected =
↳SPRITE HIT(InvisibleSpr,0)
IF (spritesselected<> 0) AND
↳(MOUSECLICK(=1)
  xoffset = SPRITEX(spritesselected)
  ↳-MOUSEX()
  yoffset = SPRITE Y(spritesselected)
  ↳-MOUSEY()
  WHILE MOUSECLICK() = 1
    SPRITE spritesselected,
    ↳MOUSEX()+xoffset,
    ↳MOUSEY()+yoffset,spritesselected
  ENDWHILE
ENDIF
LOOP
REM *** End program ***
END

```

Activity Sup025

The *smiley* sprite cannot be dragged.

To update the program change the line

```
draggable = FALSE
```

to

```
draggable = TRUE
```

The *smiley* sprite can now be dragged.

Activity Sup026

No solution required.

Activity Sup027

```

REM *** Constants ***
REM *** Bitmaps ***
#CONSTANT wholeBmp 1
REM *** Images ***
#CONSTANT BlackImg 1
REM *** Sprites ***
#CONSTANT BlackSpr 1
REM *** Numeric Values ***
#CONSTANT piecewidth 64
#CONSTANT pieceheight 64

REM *** Main Logic ***
GameSetUp()
CreatePieces("costumes.jpg")
WAIT KEY
END

FUNCTION GameSetUp
  REM *** Set screen resolution ***
  SET DISPLAY MODE 1280,1024,32
  BACKDROP ON
  REM *** Set random number generator ***
  RANDOMIZE TIMER()
  REM *** Load black image ***
  LOAD IMAGE "black.bmp",BlackImg
ENDFUNCTION

FUNCTION CreatePieces(filename$)

  REM *** Load the jigsaw image ***
  REM *** into bitmap area 1 ***
  LOAD BITMAP filename$,1
  REM *** Calc no. of rows & columns ***

```

```

rows = BITMAP HEIGHT(1)/pieceheight
cols = BITMAP WIDTH(1)/piecewidth
REM *** Split image into pieces ***
REM *** creating a sprite for each ***
spriteno = 2
FOR r = 0 TO rows-1
  FOR c = 0 TO cols-1
    REM *** Get section of bitmap ***
    GET IMAGE spriteno,c*piecewidth,
    ↳r*pieceheight,c*piecewidth+
    ↳piecewidth,r*pieceheight+
    ↳pieceheight,1
    REM *** IF not a corner piece ***
    IF NOT ((r = 0 AND c = 0)
    ↳OR (r = 0 AND c = cols-1)
    ↳OR (r = rows-1 AND c=0)
    ↳OR (r = rows-1 AND c=cols-1))
      REM *** Random place for piece
      y = RND(900)
      IF y < rows*pieceheight+
        ↳pieceheight/2
        x = cols*piecewidth+
        ↳piecewidth/2+ RND(1100-
        ↳(cols*piecewidth))
      ELSE
        x = RND(1200)
      ENDIF
      REM *** Create sprite from
      REM *** bitmap section ***
      SPRITE spriteno,x,y,spriteno
      REM *** Sprite origin at its
      REM *** centre ***
      OFFSET SPRITE spriteno,
        ↳piecewidth/2,piecewidth/2
    ELSE
      REM *** Place corner piece ***
      SPRITE spriteno,c*piecewidth,
        ↳r*pieceheight,spriteno
    ENDIF
    REM *** No transparency ***
    SET SPRITE spriteno,1,0
    REM *** Incr sprite no. used ***
    INC spriteno
  NEXT c
NEXT r
REM *** Screen as output device ***
SET CURRENT BITMAP 0
ENDFUNCTION

```

Activity Sup028

The corner pieces did not have their origins adjusted and therefore do not position correctly when moved.

Activity Sup029

The updated version of the program is shown below:

```

REM *** Constants ***
REM *** Bitmaps ***
#CONSTANT wholeBmp 1
REM *** Images ***
#CONSTANT BlackImg 1
REM *** Sprites ***
#CONSTANT BlackSpr 1
REM *** Numeric Values ***
#CONSTANT piecewidth 64
#CONSTANT pieceheight 64
REM *** Global Variables ***
GLOBAL rows
GLOBAL cols

DIM pieces(400)

```

```

REM *** Main Logic ***
GameSetUp()
CreatePieces("costumes.jpg")
SetDraggable()
DO
  HandlePieceMovement()
LOOP
END

FUNCTION GameSetUp
  REM *** Set screen resolution ***
  SET DISPLAY MODE 1280,1024,32
  BACKDROP ON
  REM *** Set random number generator ***
  RANDOMIZE TIMER()
  REM *** Load black image ***
  LOAD IMAGE "black.bmp",BlackImg
ENDFUNCTION

FUNCTION CreatePieces(filename$)
  REM *** Load the jigsaw image ***
  REM *** into bitmap area 1 ***
  LOAD BITMAP filename$,1
  REM *** Calc no. of rows & columns ***
  rows = BITMAP HEIGHT(1)/pieceheight
  cols = BITMAP WIDTH(1)/piecewidth
  REM *** Split image into pieces ***
  REM *** creating a sprite for each ***
  spriteno = 2
  FOR r = 0 TO rows-1
    FOR c = 0 TO cols-1
      REM *** Get section of bitmap ***
      GET IMAGE spriteno,c*piecewidth,
        ↵r*pieceheight,c*piecewidth+
        ↵piecewidth,r*pieceheight+
        ↵pieceheight,1
      REM *** IF not a corner piece ***
      IF NOT ((r = 0 AND c = 0)
        ↵OR (r = 0 AND c = cols-1)
        ↵OR (r = rows-1 AND c=0)
        ↵OR (r = rows-1 AND c=cols-1))
        REM *** Random place for piece
        y = RND(900)
        IF y < rows*pieceheight+
          ↵pieceheight/2
          x = cols*piecewidth+
            ↵piecewidth/2+ RND(1100-
            ↵(cols*piecewidth))
        ELSE
          x = RND(1200)
        ENDIF
      REM *** Create sprite from
      REM *** bitmap section ***
      SPRITE spriteno,x,y,spriteno
      REM *** Sprite origin at its
      REM *** centre ***
      OFFSET SPRITE spriteno,
        ↵piecewidth/2,piecewidth/2
    ELSE
      REM *** Place corner piece ***
      SPRITE spriteno,c*piecewidth,
        ↵r*pieceheight,spriteno
    ENDIF
    REM *** No transparency ***
    SET SPRITE spriteno,1,0
    REM *** Incr sprite no. used ***
    INC spriteno
  NEXT c
NEXT r
REM *** Screen as output device ***
SET CURRENT BITMAP 0
ENDFUNCTION

```

```

FUNCTION HandlePieceMovement()
  REM *** Move mouse sprite ***
  SPRITE BlackSpr,MOUSEX(),MOUSEY(),
  ↵BlackImg
  REM *** Check for sprite collision ***
  spriteselcted = SPRITE HIT(BlackSpr,0)
  REM *** IF collision & mouse pressed
  REM *** THEN ***
  IF (spriteselcted<> 0) AND
  ↵(MOUSECLICK()=1) AND
  ↵pieces(spriteselcted) > 0
    REM *** Calc sprite/mouse offsets ***
    xoffset = SPRITE X(spriteselcted)
    ↵- MOUSEX()
    yoffset = SPRITE Y(spriteselcted)
    ↵- MOUSEY()
    REM *** Drag sprite until mouse
    REM *** button released ***
    WHILE MOUSECLICK() = 1
      SPRITE spriteselcted, MOUSEX()
        ↵+xoffset, MOUSEY()+yoffset,
        ↵spriteselcted
    ENDWHILE
    REM *** IF sprite in jigsaw area,
    REM *** position exactly ***
    IF MOUSEX()<cols*piecewidth
      ↵AND MOUSEY()<rows*pieceheight
      SPRITE spriteselcted,
        ↵SPRITE X(spriteselcted)
        ↵/piecewidth*piecewidth
        ↵+piecewidth/2,
        ↵SPRITE Y(spriteselcted)
        ↵/pieceheight*pieceheight
        ↵+pieceheight/2,spriteselcted
    ENDIF
  ENDIF
ENDFUNCTION

FUNCTION SetDraggable()
  REM *** Set every pieces draggable
  ...***
  FOR c = 3 TO rows*cols+1
    pieces(c) = 1
  NEXT c
  REM *** ... except four corner pieces
  ***
  pieces(2) = 0
  pieces(cols+1) = 0
  pieces((rows-1)*cols+2) = 0
  pieces(rows*cols+1) = 0
ENDFUNCTION

```

Activity Sup030

No solution required.

Activity Sup031

The new code for *HandlePieceMovement()* is:

```

FUNCTION HandlePieceMovement()
  REM *** Move mouse sprite ***
  SPRITE BlackSpr,MOUSEX(),MOUSEY(),
  ↵BlackImg
  REM *** Check for sprite collision ***
  spriteselcted = SPRITE HIT(BlackSpr,0)
  REM *** IF collision & mouse pressed
  REM *** THEN ***
  IF (spriteselcted<> 0) AND
  ↵(MOUSECLICK()=1) AND
  ↵pieces(spriteselcted) > 0
    REM *** Calc sprite/mouse offsets ***
    xoffset = SPRITE X(spriteselcted)
    ↵- MOUSEX()
    yoffset = SPRITE Y(spriteselcted)

```

```

↳- MOUSEY()
REM *** Draw selected sprite on top ***

SET SPRITE PRIORITY spriteselcted,1
REM *** Drag sprite until mouse
REM *** button released ***
WHILE MOUSECLICK() = 1
    SPRITE spriteselcted, MOUSEX()
    ↳+xoffset, MOUSEY()+yoffset,
    ↳spriteselcted
ENDWHILE
REM *** IF sprite in jigsaw area,
REM *** position exactly ***
IF MOUSEX()<cols*piecewidth
↳AND MOUSEY()<rows*pieceheight
    SPRITE spriteselcted,
    ↳SPRITE X(spriteselcted)
    ↳/piecewidth*piecewidth
    ↳+piecewidth/2,
    ↳SPRITE Y(spriteselcted)
    ↳/pieceheight*pieceheight
    ↳+pieceheight/2,spriteselcted
ENDIF
REM *** Default sprite order ***
SET SPRITE PRIORITY spriteselcted,0
ENDIF
ENDFUNCTION

```

Activity Sup032

The piece placed at the bottom-right corner disappears under the corner piece but can be retrieved by dragging.

The piece placed at the top-left corner appears on top of the corner piece and cannot be moved.

Activity Sup033

The new version of the main logic should be:

```

GameSetUp()
CreatePieces("costumes.jpg")
SetDraggable()
InitialiseBoard()
DO
    HandlePieceMovement()
LOOP
END

```

The new version of *HandlePieceMovement()* is:

```

FUNCTION HandlePieceMovement()
REM *** Move mouse sprite ***
SPRITE BlackSpr,MOUSEX(),MOUSEY(),
↳BlackImg
REM *** Check for sprite collision ***
spriteselcted = SPRITE HIT(BlackSpr,0)
REM *** IF collision & mouse pressed
REM *** THEN ***
IF (spriteselcted<> 0) AND
↳(MOUSECLICK()=1) AND
↳pieces(spriteselcted) > 0
    REM *** Calc sprite/mouse offsets ***
    xoffset = SPRITE X(spriteselcted)
    ↳- MOUSEX()
    yoffset = SPRITE Y(spriteselcted)
    ↳- MOUSEY()
    REM *** Store sprite's start post ***
    oldx = SPRITE X(spriteselcted)
    oldy = SPRITE Y(spriteselcted)
    SET SPRITE PRIORITY spriteselcted,1
    REM *** Drag sprite until mouse

```

```

REM *** button released ***
WHILE MOUSECLICK() = 1
    SPRITE spriteselcted, MOUSEX()
    ↳+xoffset, MOUSEY()+yoffset,
    ↳spriteselcted
ENDWHILE
DropPiece()
REM *** Default sprite order ***
SET SPRITE PRIORITY spriteselcted,0
ENDIF
ENDFUNCTION

```

The piece you are attempting to place jumps back instantly to its old position.

Activity Sup034

No solution required.

Activity Sup035

The updated functions are listed below:

```

FUNCTION DropPiece(spriteselcted,
↳oldscreenx, oldscreeny)
REM *** IF piece not in build area,
REM *** return ***
IF MOUSEX()>=(cols)*piecewidth
↳OR MOUSEY()>=(rows)*pieceheight
    EXITFUNCTION
ENDIF
REM *** Position piece ***
SPRITE spriteselcted,
↳SPRITE X(spriteselcted)/piecewidth
↳*piecewidth+piecewidth/2,
↳SPRITE Y(spriteselcted)/pieceheight
↳*pieceheight+pieceheight/2,
↳spriteselcted
REM *** Calculate column(x) and row(y)
REM *** position of piece ***
x = (SPRITE X(spriteselcted)-
↳piecewidth/2)/piecewidth
y = (SPRITE Y(spriteselcted)-
↳pieceheight/2)/pieceheight
positiononboard = y*cols+x+2
REM *** IF jigsaw post empty THEN ***
IF board(positiononboard)=0
    REM *** Record ID in board array ***
    board(positiononboard) =
    ↳spriteselcted
    REM *** Store board post in pieces
    ↳array ***
    pieces(spriteselcted)=
    ↳positiononboard
ELSE
    REM *** ELSE return piece to its
    REM *** original position ***
    SPRITE spriteselcted,oldscreenx,
    ↳oldscreeny,spriteselcted
ENDIF
ENDFUNCTION

```

```

FUNCTION HandlePieceMovement()
REM *** Move mouse sprite ***
SPRITE BlackSpr,MOUSEX(),MOUSEY(),
↳BlackImg
REM *** Check for sprite collision ***
spriteselcted = SPRITE HIT(BlackSpr,0)
REM *** IF collision & mouse pressed
REM *** THEN ***
IF (spriteselcted<> 0) AND
↳(MOUSECLICK()=1) AND

```

```

    ↵pieces(spritesselected) > 0
    REM *** Calc sprite/mouse offsets ***
    xoffset = SPRITE X(spritesselected)
    ↵- MOUSEX()
    yoffset = SPRITE Y(spritesselected)
    ↵- MOUSEY()
    REM *** Store sprite's start post ***
    oldx = SPRITE X(spritesselected)
    oldy = SPRITE Y(spritesselected)
    SET SPRITE PRIORITY spritesselected,1
    REM *** IF piece in jigsaw area ***
    IF pieces(spritesselected)> 1
        REM *** Remove from board array ***
        board(pieces(spritesselected)) = 0
        REM *** and mark as outside area ***
        pieces(spritesselected)=1
    ENDIF
    REM *** Drag sprite until mouse
    REM *** button released ***
    WHILE MOUSECLICK() = 1
        SPRITE spritesselected, MOUSEX()
        ↵+xoffset, MOUSEY()+yoffset,
        ↵spritesselected
    ENDWHILE
    DropPiece()
    REM *** Default sprite order ***
    SET SPRITE PRIORITY spritesselected,0
ENDIF
ENDFUNCTION

```

```

    DEC piecesplaced
ENDIF

```

In *DropPiece()*, change the last IF statement to begin with:

```

IF board(positiononboard) = 0
    REM *** Record sprite's ID in board ***
    board(positiononboard) = spritesselected
    REM *** board post in pieces array ***
    pieces(spritesselected) = positiononboard
    REM *** Increment no. of pieces in area
        INC piecesplaced
ELSE

```

Activity Sup036

In *DropPiece()* the lines

```

ELSE
    REM *** ELSE return piece to its
    ↵original position ***
    SPRITE spritesselected,oldscreenx,
    ↵oldscreeny,spritesselected

```

should be changed to

```

ELSE
    REM *** ELSE return piece to its
    ↵original position ***
    SlideBack(spritesselected,oldscreenx,
    ↵oldscreeny)

```

Activity Sup037

In the main section, add the line

```

GLOBAL piecesplaced = 4

```

and change the main loop to:

```

GameSetUp()
CreatePieces("costumes.jpg")
SetDraggable()
InitialiseBoard()
REPEAT
    HandlePieceMovement()
UNTIL IsComplete()
WAIT KEY
END

```

In *HandlePieceMovement()*, change the last IF structure to:

```

IF pieces(spritesselected)> 1
    REM *** Remove it from board array ***
    board(pieces(spritesselected)) = 0
    REM *** and mark as outside area ***
    pieces(spritesselected)=1
    REM *** Reduce no. of pieces in area ***

```