

```

/*****
*
*                               S 3 2 2 0 C . C
*
**-----**
* Task                        : Demonstrates sprites in 320x200 VGA graphic
*                             mode, using 256 colors and four screen pages.
*                             This program requires the assembly language
*                             modules V3220CA.ASM and S3220CA.ASM.
**-----**
* Author                      : Michael Tischer
* Developed on                : 09/09/90
* Last update                 : 02/17/92
**-----**
* Memory model                : SMALL
**-----**
* (MICROSOFT C)
* Compilation                  : CL /AS /c /W0 s3220c.c
*                               LINK s3220c v3220ca s3220ca;
**-----**
* (BORLAND TURBO C)
* Compilation                  : Create a project file containing the following:
*                               s3220c.c
*                               v3220ca.asm
*                               s3220ca.asm
**-----**
* Call                        : s3220c
*****/

```

```

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

```

```

#include <conio.h>

/*-- Compiler-dependent declarations -----*/

#ifdef __TURBOC__
    #include <alloc.h>
#else
    #include <malloc.h>
    #define random(x) ( rand() % (x+1) )          /* Random function */
#endif

/*-- Type declarations -----*/

typedef unsigned char BYTE;

typedef struct {
    BYTE twidth,           /* Sprite design */
        theight,          /* Total width */
        ppage,             /* Height in pixel lines */
        *bmskp,            /* Placed in page ... */
        msklen;            /* Pointer to bit mask */
    int pxlin;             /* Entry length */
} SPOOLK;                 /* Pixel lines for sprite */
                           /* in its page */

typedef struct {
    BYTE bkgpage;          /* Sprite descriptor (ID) */
    int x[2], y[2],        /* Background page */
        bkx, bky;         /* Coordinates: pp. 0 & 1 */
    SPOOLK * splookp;      /* Background buffer */
} SPID;                   /* Pointer to sprite design */

/*-- External references to assembler routines -----*/

```

```

extern void init320200( void );
extern void setpix( int x, int y, unsigned char pcolor);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );
extern void waitvsync( void );
extern void blockmove( BYTE frompage, int fromx, int fromy,
                      BYTE topage, int tox, int toy,
                      BYTE pwidth, BYTE pheight, BYTE *bmskp );

/*-- Constants -----*/

#define NOBITMASK (BYTE *) 0

#define MAXX 319                /* Maximum X- and Y-coordinates */
#define MAXY 199

#define OUT_LEFT  1  /* For collision documentation in SpriteMove() */
#define OUT_TOP   2
#define OUT_RIGHT  4
#define OUT_BOTTOM 8
#define OUT_NO     0                /* None */

/*****
*   IsVga: Determines whether a VGA card is installed.
*   -----
*   Input      : None
*   Output     : 0 if no VGA exists, otherwise <0
*****/

```

```

BYTE IsVga( void )
{
    union REGS Regs;          /* Processor registers for interrupt call */

    Regs.x.ax = 0x1a00;        /* Function 1AH applies to VGA only */
    int86( 0x10, &Regs, &Regs );
    return ( Regs.h.al == 0x1a ); /* Is this function available? */
}

/*****
*   Line: Draws a line based on the Bresenham algorithm.
*   -----
*   Input   : X1, Y1 = Starting coordinates (0 - ...)
*            X2, Y2 = Ending coordinates
*            PCOLOR = Color of line pixels
*****/

/*-- Function for swapping two integer variables -----*/

void SwapInt( int *i1, int *i2 )
{
    int dummy;

    dummy = *i2;
    *i2   = *i1;
    *i1   = dummy;
}

/*-- Main part of function -----*/

void Line( int x1, int y1, int x2, int y2, BYTE pcolor )
{

```

```

int d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y;

if ( abs(x2-x1) < abs(y2-y1) )          /* X- or Y-axis overflow? */
{                                       /* Check Y-axis */
    if ( y1 > y2 )                    /* y1 > y2? */
    {
        SwapInt( &x1, &x2 );          /* Yes --> Swap X1 with X2 */
        SwapInt( &y1, &y2 );          /* and Y1 with Y2 */
    }

    xincr = ( x2 > x1 ) ? 1 : -1;      /* Set X-axis increment */

    dy = y2 - y1;
    dx = abs( x2-x1 );
    d = 2 * dx - dy;
    aincr = 2 * (dx - dy);
    bincr = 2 * dx;
    x = x1;
    y = y1;

    setpix( x, y, pcolor );
    for (y=y1+1; y<= y2; ++y )
    {
        if ( d >= 0 )
        {
            x += xincr;
            d += aincr;
        }
        else

```

```

        d += bincr;
        setpix(x, y, pcolor);
    }
}
else
{
    /* Check X-axis */
    if ( x1 > x2 )
    {
        /* x1 > x2? */
        SwapInt( &x1, &x2 );
        SwapInt( &y1, &y2 );
        /* Yes --> Swap X1 with X2 */
        /* and Y1 with Y2 */
    }

    yincr = ( y2 > y1 ) ? 1 : -1;
    /* Set Y-axis increment */

    dx = x2 - x1;
    dy = abs( y2-y1 );
    d = 2 * dy - dx;
    aincr = 2 * (dy - dx);
    bincr = 2 * dy;
    x = x1;
    y = y1;

    setpix(x, y, pcolor);
    for (x=x1+1; x<=x2; ++x )
    {
        /* Set first pixel */
        /* Execute line on X-axes */
        if ( d >= 0 )
        {
            y += yincr;
            d += aincr;
        }
        else
            d += bincr;
    }
}

```

```

        setpix(x, y, pcolor);
    }
}
}

```

```

/*****
*   PrintChar : Writes a character to the screen while in graphic mode.*
**-----**
*   Input      :  THECHAR  = Character to be written                      *
*                 X, Y     = X- and Y-coordinates of upper-left corner    *
*                 FG       = Foreground color                             *
*                 BK       = Background color                             *
*   Info       :  Character is created in an 8x8 matrix, based on the     *
*                 8x8 ROM font.                                           *
*****/

```

```

void PrintChar( char thechar, int x, int y, BYTE fg, BYTE bk )
{
    typedef BYTE FDEF[256][8];
    typedef FDEF far *TPTR;
                                /* Font array */
                                /* Pointer to font */

    BYTE          i, k,
                BMask;
                                /* Loop counter */
                                /* Bit mask for character design */

    static TPTR fptr = (TPTR) 0;
                                /* Pointer to font in ROM */

    if ( fptr == (TPTR) 0 )
        fptr = getfontptr(); /* Pointer to font already set? */
                                /* No --> Use the assembler function to load */

    /*- Generate character pixel by pixel -----*/

    if ( bk == 255 )
        /* Drawing transparent characters? */

```

```

for ( i = 0; i < 8; ++i )      /* Yes --> Set foreground pixels only */
{
    BMask = (*fptr)[thechar][i];      /* Get bit pattern for one line */
    for ( k = 0; k < 8; ++k, BMask <= 1 )      /* Execute column */
        if ( BMask & 128 )      /* Pixel set? */
            setpix( x+k, y+i, fg );      /* Yes */
}
else      /* No --> Consider background as well */
for ( i = 0; i < 8; ++i )      /* Execute lines */
{
    BMask = (*fptr)[thechar][i];      /* Get bit pattern for one line */
    for ( k = 0; k < 8; ++k, BMask <= 1 )      /* Execute columns */
        setpix( x+k, y+i, ( BMask & 128 ) ? fg : bk );
}
}

```

```

/*****
*   GrfxPrintf: Displays a formatted string on the graphic screen.      *
*   -----*
*   Input      : X, Y      = Starting coordinates (0 - ...)      *
*                FG        = Foreground color      *
*                BK        = Background color (255 = transparent)      *
*                STRING     = String with format information      *
*                ...        = Arguments similar to printf      *
*****/

```

```

void GrfxPrintf( int x, int y, BYTE fg, BYTE bk, char * string, ... )
{
    va_list parameter;      /* Parameter list for VA_... macros */
    char stngbuf[255],      /* Buffer for formatted string */
        *cp;

```



```

va_start( parameter, string );           /* Convert parameter */
vsprintf( stngbuf, string, parameter );  /* Format */
for ( cp = stngbuf; *cp; ++cp, x+= 8 )   /* Formatted string */
    PrintChar( *cp, x, y, fg, bk );      /* Display using PrintChar */
}

```

```

/*****
*   CreateSprite: Creates a sprite based on a user-defined      *
*                   pixel pattern.                               *
**-----**
*   Input   : SPLOOKP = Pointer the data structure from CompileSprite() *
*             BKGPAGE = Screen page in which sprite background should *
*                   be stored                                           *
*             BKKX,   = bkgpage coordinates at which sprite background *
*             BKKY    is stored                                         *
*   Output  : Pointer to created sprite structure                  *
*   Info    : The sprite background requires two areas the same size *
*             as the corresponding sprite.                           *
*****/

```

```

SPID *CreateSprite( SPLOOK *splookp, BYTE bkgpage, int bkkx, int bkky )
{
    SPID *spidp;           /* Pointer to created sprite structure */

    spidp = (SPID *) malloc( sizeof(SPID) ); /* Allocate sprite struc. */
    spidp->splookp = splookp;                /* Pass data to the */
    spidp->bkgpage = bkgpage;                /* sprite structure */
    spidp->bkkx = bkkx;
    spidp->bkky = bkky;

    return spidp;          /* Return pointer to the sprite structure */
}

```

```

/*****
*   CompileSprite: Creates a sprite's pixel and bit patterns, based on *
*                   the sprite's definition at runtime.                *
**-----**
*   Input   : BUFP    = Pointer to array contains string pointers    *
*               controlling sprite's pattern                        *
*               SHEIGHT = Sprite height (and number of strings needed) *
*               GPAGE   = Graphic page for sprite design             *
*               Y        = Pixel lines needed for sprite             *
*               FB       = ASCII cahracters for the smallest color    *
*               FGCOLOR  = First color code for FB                   *
*   Info    : Sprite structure of pixel lines starts at left margin. *
*****/

```

```

SPLOOK *CompileSprite( char **bufp, BYTE sheight, BYTE gpage, BYTE y,
                      char fb, BYTE fgcolor )
{
    BYTE    swidth,                      /* String width */
           c,                          /* Get character from c sprite array */
           i, k, l,                    /* Loop variables */
           pixc,                      /* Pixel counter for creating the bit mask */
           pixm,                      /* Pixel mask */
           *lspb;                     /* Floating pointer in sprite buffer */
    int     spacing, /* Spacing from start of sprite to start of sprite */
           lx, ly;   /* Floating coordinates */
    SPLOOK *splookp; /* Pointer to created sprite structure */

    /*-- Create SpriteLook structure and fill with data -----*/

    splookp = (SPLOOK *) malloc( sizeof(SPLOOK) );
    swidth = strlen( *bufp );

```

```

splookp->twidht = spacing = ( ( swidth + 3 + 3 ) / 4 ) * 4;
splookp->bmskp = (BYTE *) malloc( (spacing*sheight+7)/8*4 );
splookp->theight = sheight;
splookp->pxlin = y;
splookp->ppage = gpage;
splookp->msklen = (spacing*sheight+7)/8;

/*-- Fill sprite background in home page with -----*/
/*-- codes for transparent character background -----*/

setpage( gpage ); /* Set page for drawing sprite */
for ( ly = y + sheight-1, lx = 4 * spacing - 1; ly >= (int) y; --ly)
    Line( 0, ly, lx, ly, 255 );

/*-- Draw four matching sprites in the home page -----*/

for ( l = 0, lx = 0; l < 4; ++l, lx+=spacing+1 )
    for ( i = 0; i < sheight; ++i )
        for ( k = 0; k < swidth; ++k )
            setpix( lx+k, y+i, ( c = (*(bufp+i)+k) ) == ' ' ? 255
                    : fgcolor+(c-fb));

/*-- Execute the four sprites and create bit masks -----*/
/*-- for copying the sprites into the bitplanes -----*/

for ( l = pixm = pixc = 0, lx = 0; l < 4; ++l, lx+=spacing )
{
    lspb = splookp->bmskp + splookp->msklen * l;
    for ( i = 0; i < sheight; ++i )
        for ( k = 0; k < spacing; ++k )
        {
            pixm >>= 1; /* Shift pixel mask 1 bit right */

```

```

    if ( getpix( lx+k, y+i ) != 255 )          /* Background pixel? */
        pixm |= 128;                          /* No --> Set bit for mask */
    if ( ++pixc == 8 )                          /* Eight pixels already handled? */
    {
        *lspb++ = pixm;                      /* Yes --> Place bit mask in sprite buffer */
        pixc = pixm = 0;                      /* Set pixel counter and mask to 0 */
    }
}
if ( pixc )                                    /* Last nibble still not in buffer? */
{
    *lspb = pixm >> 4;                      /* No --> Move high nibble to */
    pixc = pixm = 0;                          /* low nibble and store them */
}
}
return splookp;                                /* Return pointer to sprite buffer */
}

```

```

/*****
*  PrintSprite : Displays sprite in a specified page.
**-----**
*  Input   : SPIDP   = Pointer to the sprite structure
*           SPRPAGE = Page in which sprite should be drawn (0 or 1)
*****/

```

```

void PrintSprite( register SPID *spidp, BYTE sprpage )
{
    BYTE twidth;                                /* Total width of sprite */
    int  x;                                    /* X-coordinate of sprite in its page */
    SPLOOK *splookp;                            /* Pointer to sprite's appearance */

    twidth = (splookp = spidp->splookp)->twidth;
    x = spidp->x[sprpage];
}

```

```

    blockmove( splookp->ppage, twidth * (x % 4), splookp->pxlin, sprpage,
               x & ~3, spidp->y[sprpage], twidth, splookp->theight,
               splookp->bmskp + (x % 4) * splookp->msklen );
}

/*****
*   GetSpriteBg: Gets a sprite background and specifies the position.  *
**-----**
*   Input      : SPIDP    = Pointer to the sprite structure          *
*                SPRPAGE  = Page from which the background should be taken *
*                (0 or 1)                                           *
*****/

void GetSpriteBg( register SPID *spidp, BYTE sprpage )
{
    SPOOLK *splookp;                                /* Pointer to sprite graphic */

    splookp = spidp->splookp;
    blockmove( sprpage, spidp->x[sprpage], spidp->y[sprpage],
               spidp->bkgpage, spidp->bkx + ( splookp->twidth * sprpage ),
               spidp->bky, splookp->twidth, splookp->theight, NOBITMASK );
}

/*****
*   RestoreSpriteBg: Restores sprite background from original graphic *
*                   page.                                              *
**-----**
*   Input      : SPIDP    = Pointer to the sprite structure          *
*                SPRPAGE  = Page from which the background should be copied *
*                (0 or 1)                                           *
*****/

```

```

void RestoreSpriteBg( register SPID *spidp, BYTE sprpage )
{
    SPLOOK *splookp;                                /* Pointer to sprite graphic */

    splookp = spidp->splookp;
    blockmove( spidp->bkgpage, spidp->bkx + ( splookp->twidth * sprpage ),
               spidp->bky, sprpage, spidp->x[sprpage], spidp->y[sprpage],
               splookp->twidth, splookp->theight, NOBITMASK );
}

```

```

/*****
* MoveSprite: Copy sprite within background to original graphic page.*
**-----**
* Input   : SPIDP   = Pointer to the sprite structure                *
*           SPRPAGE = Page to which the background should be copied *
*           (0 or 1)                                                *
*           DELTAX  = Movement counter in X-                        *
*           DELTAY  and Y-directions                               *
* Output  : Collision marker (see OUT_ constants)                  *
*****/

```

```

BYTE MoveSprite( SPID *spidp, BYTE sprpage, int deltax, int deltax )
{
    int newx, newy;                                /* New sprite coordinates */
    BYTE out;                                       /* Display collision with border */

    /*-- Move X-coordinates and test for border collision -----*/

    if ( ( newx = spidp->x[sprpage] + deltax ) < 0 )
    {
        newx = 0 - deltax - spidp->x[sprpage];
        out = OUT_LEFT;
    }
}

```

```

    }
else
    if ( newx > 319 - spidp->splookp->twidht )
    {
        newx = 640-newx-2*(spidp->splookp->twidht);
        out = OUT_RIGHT;
    }
    else
        out = OUT_NO;

/*-- Move Y-coordinates and test for border collision -----*/
if ( ( newy = spidp->y[sprpage] + deltay ) < 0 )      /* Top border? */
{
    /* Yes --> Deltay must be negative */
    newy = 0 - deltay - spidp->y[sprpage];
    out |= OUT_TOP;
}
else
    if ( newy + spidp->splookp->theight > 199+1 )    /* Bottom border? */
    {
        /* Yes --> Deltay must be positive */
        newy = 400-newy-2*(spidp->splookp->theight);
        out |= OUT_BOTTOM;
    }

/*-- Set new position only if different from old position -----*/
if ( newx != spidp->x[sprpage] || newy != spidp->y[sprpage] )
{
    /* If there's a new position */
    RestoreSpriteBg( spidp, sprpage ); /* then reset background and */
    spidp->x[sprpage] = newx;           /* store new coordinates      */
    spidp->y[sprpage] = newy;
    GetSpriteBg( spidp, sprpage );     /* Get new background */
}

```

```

    PrintSprite( spidp, sprpage );    /* Draw sprite in specified page */
}
return out;
}

```

```

/*****
*   SetSprite: Sets sprite at a specific position.
*   *****/
*-----*
*   Input    : SPIDP = Pointer to the sprite structure
*               x0, y0 = Sprite coordinates for page 0
*               x1, y1 = Sprite coordinates for page 1
*   Info     : This function call should be made the first time that
*               MoveSprite() is called.
*   *****/

```

```

void SetSprite( SPID *spidp, int x0, int y0, int x1, int y1 )
{
    spidp->x[0] = x0;           /* Store coordinates in sprite structure */
    spidp->x[1] = x1;
    spidp->y[0] = y0;
    spidp->y[1] = y1;

    GetSpriteBg( spidp, 0 );    /* Get sprite backgrounds */
    GetSpriteBg( spidp, 1 );    /* in pages 0 and 1 */
    PrintSprite( spidp, 0 );    /* Draw sprite in */
    PrintSprite( spidp, 1 );    /* pages 0 and 1 */
}

```

```

/*****
*   RemoveSprite: Removes a sprite from its current position and makes
*                   it invisible.
*   *****/

```



```

*   Input   : SPIDP = Pointer to the sprite structure          *
*   Info    : After calling this function the SetSprite() function *
*              must be called before the sprite can be moved using the *
*              MoveSprite() function.                            *
*****/

```

```

void RemoveSprite( SPID *spidp )
{
    RestoreSpriteBg( spidp, 0 );          /* Reset sprite backgrounds */
    RestoreSpriteBg( spidp, 1 );          /* in pages 0 and 1      */
}

```

```

/*****
*   Demo: Demonstrates these functions.                            *
*****/

```

```

void Demo( void )
{
    static char *StarShipUp[20] =
    {
        "          AA          ",
        "          AAAA          ",
        "          AAAA          ",
        "          AA          ",
        "          GBBGG          ",
        "          GBBCCBBG          ",
        "          GBBBCCBBG          ",
        "          GBBBBBBBBBG          ",
        "          GBBBBBBBBBG          ",
        " G          GBBBBBBBBBBBBG          G ",
        "GCG          GGDBBBBBBBBBBDGG          GCG",
        "GCG          GGBBDBBB          BBDBBBGG          GCG",
        "GCBGGGBBBBBDBB          BBDBBBBGGGBCG"
    }
}

```

```

"GCBBBBBBBBBDB      BBBBBBBBBBBBCG" ,
"BBBBBBBBBBBBBDB BB BBBBBBBBBBBBBB" ,
"GGCBBBBBBBDBBBBBBBBBBDBBBBBBBBCG " ,
"   GGCCBBDDDDDDDDDDDDDBBBCCG " ,
"   GGBBDDDDDGGGGGDDDDDBBG " ,
"       GDDDDGGG   GGGDDDDG " ,
"       DDDD       DDDD " } ;

```

```

static char *StarShipDown[20] =
{
"       DDDD       DDDD " ,
"       GDDDDGGG   GGGDDDDG " ,
"       GGBBDDDDDGGGGGDDDDDBBG " ,
"       GGCCBBDDDDDDDDDDDBBBCCG " ,
"GCBBBBBBBBBDBBBBBBBBBBBDBBBBBBBBCG " ,
"BBBBBBBBBBBBBDB BB BBBBBBBBBBBBBB" ,
"GCBBBBBBBBBDB      BBBBBBBBBBBBCG" ,
"GCBGGGBBBBBBDB      BBDBBBBGGGBCG" ,
"GCG   GGBBDBBB   BBBDBBBGG   GCG" ,
"GCG       GGBBBBBBBBBBDGG       GCG" ,
"  G       GBBBBBBBBBBBG       G " ,
"       GBBBBBBBBBBG " ,
"       GBBBBBBBBBBG " ,
"       GBBCCBBBG " ,
"       GBBCCBBG " ,
"       GBBGG " ,
"       AA " ,
"       AAAA " ,
"       AAAA " ,
"       AA " } ;

```

```

#define SPRNUM 6                                /* Number of sprites */
#define CWIDTH 38                               /* Width of copyright message in characters */
#define CHEIGHT 6                               /* Message height in rows */
#define SX (MAXX-(CWIDTH*8)) / 2               /* Starting X-coordinate */
#define SY (MAXY-(CHEIGHT*8)) / 2              /* Starting Y-coordinate */

struct {
    SPID *spidp;                                /* For sprite management */
    int  deltax[2],                             /* Pointer to sprite ID */
        deltax[2];                             /* X-movement for pages 0 and 1 */
    int  deltay[2];                             /* Y-movement for pages 0 and 1 */
} sprites[ SPRNUM ];

BYTE    page,                                  /* Current page */
out;                                           /* Get flags for page collision */
int      x, y, i,                             /* Loop counter */
dx, dy;                                       /* Movement value */
char     lc;
SPLOOK  *starshipupp, *starshipdnp;          /* Sprite pointer */

srand( *(long far *) 0x0040006c );            /* Initialize random */
                                              /* number generator */

/*-- Fill the first two graphic pages with characters -----*/
for ( page = 0; page < 2; ++ page )
{
    setpage( page );
    for ( lc = 0, y = 0; y < 200-8; y += 12 )
        for ( x = 0; x < 320-8; x += 8 )
            GrfxPrintf( x, y, lc % 255, 255, "%c", lc++ & 127 );
}

/*-- Display copyright message -----*/

```

```

Line( SX-1, SY-1, SX+CWIDTH*8, SY-1, 15 );
Line( SX+CWIDTH*8, SY-1, SX+CWIDTH*8, SY+HEIGHT*8,15 );
Line( SX+CWIDTH*8, SY+HEIGHT*8, SX-1, SY+HEIGHT*8, 15 );
Line( SX-1, SY+HEIGHT*8, SX-1, SY-1, 15 );
GrfxPrintf( SX, SY,      15, 4,
            "                                     " );

GrfxPrintf( SX, SY+8,  15, 4,
            " S3220C.C (c) 1992 by Michael Tischer " );
GrfxPrintf( SX, SY+16, 15, 4,
            "                                     " );

GrfxPrintf( SX, SY+24, 15, 4,
            "      Sprite demo for 320x200 mode      " );
GrfxPrintf( SX, SY+32, 15, 4,
            "                                     " );
GrfxPrintf( SX, SY+40, 15, 4,
            "                                     " );
            "                                     " );
}

/*-- Create patterns for the different sprites -----*/

starshipupp = CompileSprite( StarShipUp,  20, 2, 0, 'A', 1 );
starshipdnp = CompileSprite( StarShipDown, 20, 2, 40, 'A', 1 );

/*-- Create different sprites -----*/

for ( i = 0; i < SPRNUM ; ++ i )
{
    sprites[ i ].spidp = CreateSprite( starshipupp, 3, ( i % 3 ) * 100,
                                      ( i / 3 ) * 30 );
    do
        /* Select movement value for sprites */
    {
        dx = 0;

```

```

    dy = random(8) - 4;
}
while ( dx==0  &&  dy==0 );

sprites[ i ].deltax[0] = sprites[ i ].deltax[1] = dx * 2;
sprites[ i ].deltay[0] = sprites[ i ].deltay[1] = dy * 2;

x = ( 320 / SPRNUM * i ) + (320 / SPRNUM - 40) / 2 ;
y = random( 200 - 40 );
SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
}

/*-- Move sprites and bounce them off the page borders -----*/

page = 1;                                     /* Start with page 1 */
while ( !kbhit() )                           /* Press a key to end the loop */
{
    showpage( 1 - page );                    /* Display other page */

    for ( i = 0; i < SPRNUM; ++ i )          /* Execute sprites */
    {
        /* Move sprite and check for page collision */
        out = MoveSprite( sprites[i].spidp, page, sprites[i].deltax[page],
                           sprites[i].deltay[page] );
        if ( out & OUT_TOP   ||  out & OUT_BOTTOM )      /* Top/bottom */
                                                                /* collision? */
        {
            /* Yes --> Change direction of movement and sprite graphic */
            sprites[i].deltay[page] = 0 - sprites[i].deltay[page];
            sprites[i].spidp->splookp = ( out & OUT_TOP ) ? starshipdnp
                                                         : starshipupp;
        }
        if ( out & OUT_LEFT   ||  out & OUT_RIGHT )
            sprites[i].deltax[page] = 0 - sprites[i].deltax[page];
    }
}

```

```

    }
    page = (page+1) & 1;                               /* Toggle between 1 and 0 */
}

/*****
**                               M A I N   P R O G R A M                               **
*****/

void main( void )
{
    union REGS regs;

    if ( IsVga() )                                       /* VGA card installed? */
    {
        init320200();                                   /* Yes --> Go ahead */
        Demo();                                         /* Initialize graphic mode */
        getch();                                       /* Wait for a key */
        regs.x.ax = 0x0003;                             /* Shift into text mode */
        int86( 0x10, &regs, &regs );
    }
    else
        printf( "S3220C.C - (c) 1992 by Michael Tischer\n\n"\
               "This program requires a VGA card\n\n" );
}

```

ÿ•