NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Mar 10, 2011**

Write a C subroutine **moveBall()** that moves a ball along a fixed-length string. The ball has position *x* in metres and velocity *vx* in metres per second (e.g., 3m/s). Once per second, main() calls **moveBall()**. Thus, to move the ball the right amount, simply add *vx* to *x*. However, the string is tied at both ends to a solid wall. The ball to bounces off the wall, causing the velocity to change sign (to –*vx*) and forcing the ball to stay in the region between the two walls. Assume this confines the ball to the region from *min_x* to *max_x*. The subroutine must change both *x* and *vx*.

```
#include "259macros.h"

_________ moveBall( ____________________________________________________ )
{
```

```
// used in main() as follows...
int main(...)
{
      int x=0, vx=3, min_x=0, max_x=20;
      unsigned short BLACK = makeColour( 0,  0, 0 );
      unsigned short GREEN = makeColour( 0, 63, 0 );
      initScreen();
      fillScreen( BLACK );

      while(1) {
            drawPixel( x, y, BLACK ); // erase current ball position
            ____________ moveBall( ____________________________________________ );
            drawPixel( x, y, GREEN ); // draw new ball position

            timedDelay( ONE_MS * 1000 ); // one second delay
      }

}
```

NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Mar 10, 2011**

Write a C subroutine **moveBall()** that moves a ball along a fixed-length string. The ball has position *x* in metres and velocity *vx* in metres per second (e.g., 3m/s). Once per second, main() calls **moveBall()**. Thus, to move the ball the right amount, simply add *vx* to *x*. However, the string is tied at both ends to a solid wall. The ball to bounces off the wall, causing the velocity to change sign (to –*vx*) and forcing the ball to stay in the region between the two walls. Assume this confines the ball to the region from *min_x* to *max_x*. The subroutine must change both *x* and *vx*.

```
#include "259macros.h"

// solution 1, return new x and modify one parameter
int moveBall1( int x, int *pVX, int min_x, int max_x )
{
      x += *pVX;                        /* move ball constant amount each time */

      if( x < min_x || x > max_x ) {    /* check if ball is out of bounds */
            x -= *pVX;                  /* return ball to previous position */
            *pVX = -*pVX;               /* change velocity to opposite direction */
      }

      return x;
}

// solution 2, return nothing and modify two parameters
void moveBall2( int *pX, int *pVX, int min_x, int max_x )
{
      *pX += *pVX;

      if( *pX < min_x || *pX > max_x ) {
            *pX -= *pVX;
            *pVX = -*pVX;
      }
}

// Suppose the ball travels at 3m/s and is 2m from the wall. In "real life",
// the ball should travel 2m to the wall, then 1m backwards after 1 second.
// To keep the code simple, the solutions above ignore "real life" and use
// "simplified physics": when it detects that the ball will go out of bounds,
// it simply keeps the ball in the same position. How can you fix the code?




// used in main() as follows...
int main(...)
{
      int x=0, vx=3, min_x=0, max_x=20;
      unsigned short BLACK = makeColour( 0,  0, 0 );
      unsigned short GREEN = makeColour( 0, 63, 0 );
      initScreen();
      fillScreen( BLACK );

      while(1) {
            drawPixel( x, y, BLACK ); // erase current ball position
            x = moveBall1( x, &vx, min_x, max_x );
            //moveBall2( &x, &vx, min_x, max_x );
            drawPixel( x, y, GREEN ); // draw new ball position

            timedDelay( ONE_MS * 1000 ); // one second delay
      }

}
```

```
/* Bouncing pixels demo */

/* moves one ball along the x axis */

#include "259macros.h"


int moveBall( int x, int *pvx, int min_x, int max_x )
{
        x += *pvx;

        /* if pixel moves off screen */
        if(   x < min_x   ||   x > max_x   ) {
                x -= *pvx;              /* return pixel to previous location */
                *pvx = -*pvx;           /* change sign of velocity */
        }

        return x;
}


int main( int argc, char *argv[] )
{

        int x=0, y=30, vx=7;

        unsigned short GREEN = makeColour(  0, 63,  0 );        // bright green
        unsigned short BLACK = makeColour(  0,  0,  0 );        // black

        initScreen();
        fillScreen( BLACK );

        while(1) {
                drawPixel( x, y, BLACK );
                x = moveBall( x, &vx, 0, MAX_X_PIXELS-1 );
                drawPixel( x, y, GREEN );

                timedDelay( ONE_MS * 50 );
        }
}
```

```
/* Bouncing pixels demo */

/* moves one ball along both the x axis and y axis at different speeds */

#include "259macros.h"

int moveBall( int position, int *pVelocity, int min_pos, int max_pos )
{
        position += *pVelocity;

        /* if pixel moves off screen */
        if( position < min_pos || position > max_pos ) {
                *pVelocity = -*pVelocity; /* change sign of velocity */
                position += *pVelocity;   /* return pixel to previous location */
        }

        return position;
}


int main( int argc, char *argv[] )
{

        int x=0, y=0, vx=7, vy=3;

        unsigned short GREEN = makeColour(  0, 63,  0 );        // bright green
        unsigned short BLACK  = makeColour(  0,  0,  0 );       // black

        initScreen();
        fillScreen( BLACK );

        while(1) {
                drawPixel( x, y, BLACK );
                x = moveBall( x, &vx, 0, MAX_X_PIXELS-1 );
                y = moveBall( y, &vy, 0, MAX_Y_PIXELS-1 );
                drawPixel( x, y, GREEN );

                timedDelay( ONE_MS * 50 );
        }
}
```

```
/* Bouncing pixels demo */


/* moves one ball along both the x axis and y axis at different speeds */

/* this program uses a larger virtual grid for the ball that is SCALE=11
 * times bigger than the real graphics grid. the ball moves 7 pixels in
 * the virtual grid along the x axis, which is only 7/11ths of a pixel in
 * the real graphics grid. since the ball moves less than one pixel each
 * iteration, it appears to move more smoothly. the time delay is shorter.
 */

#include "259macros.h"



int moveBall( int position, int *pVelocity, int min_pos, int max_pos, int SCALE )
{
        position += *pVelocity;

        /* if pixel moves off screen */
        if( position < min_pos*SCALE || position >= max_pos*SCALE ) {
                *pVelocity = -*pVelocity; /* change sign of velocity */
                position += *pVelocity;   /* return pixel to previous location */
        }

        return position;
}



int main( int argc, char *argv[] )
{
        int SCALE = 11;

        int x=0, y=0, vx=7, vy=3;

        unsigned short GREEN = makeColour(  0, 63,  0 );        // bright green
        unsigned short BLACK = makeColour(  0,  0,  0 );        // black

        initScreen();
        fillScreen( BLACK );

        while(1) {
                drawPixel( x/SCALE, y/SCALE, BLACK );
                x = moveBall( x, &vx, 0, MAX_X_PIXELS-1, SCALE );
                y = moveBall( y, &vy, 0, MAX_Y_PIXELS-1, SCALE );
                drawPixel( x/SCALE, y/SCALE, GREEN );

                timedDelay( 3 * ONE_MS );
        }
}
```

```
/* Bouncing pixels demo */

/* moves two balls along both the x axis and y axis at different speeds */
/* flashes red when the two balls occupy the same pixel location */

/* this program uses a larger virtual grid for the ball that is SCALE=11
 * times bigger than the real graphics grid. the ball moves 7 pixels in
 * the virtual grid along the x axis, which is only 7/11ths of a pixel in
 * the real graphics grid. since the ball moves less than one pixel each
 * iteration, it appears to move more smoothly. the time delay is shorter.
 */

#include "259macros.h"

int moveBall( int A, int *pAv, int MAX_PIXELS, int SCALE )
{
        A += *pAv;
        if( A < 0 || A >= SCALE * MAX_PIXELS ) {
                *pAv = -*pAv;
                A += *pAv;
        }
        return A;
}

int main( int argc, char *argv[] )
{
        int ASCALE = 11, BSCALE=17;
        int Ax=0, Ay=0, Avx=7, Avy=3;
        int Bx=40, By=40, Bvx=5, Bvy=-3;

        unsigned short Acolour = makeColour(  0, 63,  0 );      // bright green
        unsigned short Bcolour = makeColour( 63,  0, 63 );      // bright magenta
        unsigned short RED     = makeColour( 63,  0,  0 );      // bright red
        unsigned short BLACK   = makeColour(  0,  0,  0 );      // black

        initScreen();
        fillScreen( BLACK );

        while(1) {
                drawPixel( Ax/ASCALE, Ay/ASCALE, BLACK );
                drawPixel( Bx/BSCALE, By/BSCALE, BLACK );

                Ax = moveBall( Ax, &Avx, MAX_X_PIXELS, ASCALE );
                Ay = moveBall( Ay, &Avy, MAX_Y_PIXELS, ASCALE );
                Bx = moveBall( Bx, &Bvx, MAX_X_PIXELS, BSCALE );
                By = moveBall( By, &Bvy, MAX_Y_PIXELS, BSCALE );

                if( (Ax/ASCALE == Bx/BSCALE) && (Ay/ASCALE == By/BSCALE ) ) {
                        // collision !
                        fillScreen( RED );
                        drawPixel( Ax/ASCALE, Ay/ASCALE, Acolour );
                        drawPixel( Bx/BSCALE, By/BSCALE, Bcolour );
                        timedDelay( 5 * ONE_MS );
                        fillScreen( BLACK );
                }

                drawPixel( Ax/ASCALE, Ay/ASCALE, Acolour );
                drawPixel( Bx/BSCALE, By/BSCALE, Bcolour );

                timedDelay( 3*ONE_MS );
        }
}
```

```
/* Bouncing pixels demo */

/* moves two balls along both the x axis and y axis at different speeds */
/* flashes red when the two balls occupy the same pixel location */
/* adds ball speed control using SWITCHES */

/* this program uses a larger virtual grid for the ball that is SCALE=11
 * times bigger than the real graphics grid. the ball moves 7 pixels in
 * the virtual grid along the x axis, which is only 7/11ths of a pixel in
 * the real graphics grid. since the ball moves less than one pixel each
 * iteration, it appears to move more smoothly. the time delay is shorter.
 */

#include "259macros.h"


int moveBall( int A, int *pAv, int MAX_PIXELS, int SCALE )
{
        A += *pAv;
        if( A < 0 || A >= SCALE * MAX_PIXELS ) {
                *pAv = -*pAv;
                A += *pAv;
        }
        if( A >= SCALE * MAX_PIXELS )
                A = SCALE * MAX_PIXELS-1;
        return A;
}



int main( int argc, char *argv[] )
{
        int ASCALE = 11, BSCALE=17;
        int speed = 1, old_speed = 1;
        int Ax=0, Ay=0, Avx=7, Avy=3;
        int Bx=40, By=40, Bvx=5, Bvy=-3;

        unsigned short Acolour = makeColour(  0, 63,  0 );      // bright green
        unsigned short Bcolour = makeColour( 63,  0, 63 );      // bright magenta
        unsigned short RED     = makeColour( 63,  0,  0 );      // bright red
        unsigned short BLACK   = makeColour(  0,  0,  0 );      // black

        initScreen();
        fillScreen( BLACK );

        while(1) {
                drawPixel( Ax/ASCALE, Ay/ASCALE, BLACK );
                drawPixel( Bx/BSCALE, By/BSCALE, BLACK );

                speed = (*pSWITCH); if (speed==0) speed=1;
                if( speed != old_speed ) {
                        ASCALE = 11*speed;
                        BSCALE = 17*speed;
                        Ax = Ax * speed / old_speed;
                        Ay = Ay * speed / old_speed;
                        Bx = Bx * speed / old_speed;
                        By = By * speed / old_speed;
                        old_speed = speed;
                }

                Ax = moveBall( Ax, &Avx, MAX_X_PIXELS, ASCALE );
                Ay = moveBall( Ay, &Avy, MAX_Y_PIXELS, ASCALE );
                Bx = moveBall( Bx, &Bvx, MAX_X_PIXELS, BSCALE );
                By = moveBall( By, &Bvy, MAX_Y_PIXELS, BSCALE );

                if( (Ax/ASCALE == Bx/BSCALE) && (Ay/ASCALE == By/BSCALE ) ) {
                        // collision !
                        fillScreen( RED );
                        drawPixel( Ax/ASCALE, Ay/ASCALE, Acolour );
                        drawPixel( Bx/BSCALE, By/BSCALE, Bcolour );
                        timedDelay( 5 * ONE_MS );
                        fillScreen( BLACK );
                }

                drawPixel( Ax/ASCALE, Ay/ASCALE, Acolour );
                drawPixel( Bx/BSCALE, By/BSCALE, Bcolour );

                timedDelay( 2*ONE_MS );
        }
}
```

```
/* simple colour graphics demo */


#include "259macros.h"


void delay( int c )
{
        //while(c--);
        timedDelay(c);
}




int main( int argc, char *argv[] )
{
        int x, y, z;
        unsigned int r=0,g=0,b=0;
        unsigned short rgb=0;
        volatile unsigned int *pPixel;

        initScreen();

        do {
                // draw some colour bars
                for( z=0; z < 100; z++ ) {
                        for( y=0; y < MAX_Y_PIXELS; y++ ) {
                                for( x=0; x < MAX_X_PIXELS; x++ ) {
                                        drawPixel( x, y, rgb );
                                        rgb++;
                                        if( rgb > 0xffff ) rgb = 0;
                                }
                        }
                        delay(1000000);
                        rgb++;
                }


                // flash full-colour screens
                for( z=0; z < 5; z++ ) {

                        for( r=g=b=0; r < 64; r+=2 )
                        { rgb = makeColour(r,g,b); fillScreen( rgb ); delay(500000); }

                        for( r=g=b=0; g < 64; g+=2 )
                        { rgb = makeColour(r,g,b); fillScreen( rgb ); delay(500000); }

                        for( r=g=b=0; b < 64; b+=2 )
                        { rgb = makeColour(r,g,b); fillScreen( rgb ); delay(500000); }

                        for( r=g=b=0; r < 64; r+=2 )
                        { rgb = makeColour(r,r,r); fillScreen( rgb ); delay(500000); }

                }
        } while(1);

}
```

```
/* mouse paint demo */

#include "259macros.h"



int main( int argc, char *argv[] )
{
        int x, y, dx, dy;

        int mouse_status, mouse_button;

        unsigned short dragcolour;

        unsigned short BLACK, WHITE, RED, GREEN, BLUE;

        char *p;
        char *pstr   = "\nhello, world\n";
        char *pmouse = "found mouse\n";
        char *pkb    = "found keyboard\n";
        char *perr   = "protocol error\n";
        char *perr2  = "unknown error\n";

        printstringJTAG( pstr );


        mouse_status = resetPS2();
        if( mouse_status == PS2_MOUSE )
                p = ( pmouse );
        else if( mouse_status == PS2_KB )
                p = ( pkb );
        else if( mouse_status == PS2_ERROR )
                p = ( perr );
        else
                p = ( perr2 );
        printstringJTAG( p );


        while( mouse_status == PS2_KB ) {
                x = getcharPS2();
                printbyteJTAG( x );
        }

        x = MAX_X_PIXELS/2;
        y = MAX_Y_PIXELS/2;

        BLACK = makeColour( 0, 0, 0 );  /* use gray to see the borders better */
        WHITE = makeColour(63,63,63 );
        RED   = makeColour(63, 0, 0 );
        GREEN = makeColour( 0,63, 0 );
        BLUE  = makeColour( 0, 0,63 );

        dragcolour = BLACK;

        initScreen();
        fillScreen( BLACK );
        drawPixel( x, y, RED );

        while( mouse_status == PS2_MOUSE ) {

                if( getMouseMotion( &dx, &dy, &mouse_button ) ) {

                        if( mouse_button & MOUSE_BUTTON1 )
                                dragcolour = WHITE;
                        else
                                dragcolour = BLACK;

                        if( mouse_button & MOUSE_BUTTON2 )
                                fillScreen( BLACK );

                        if( mouse_button & MOUSE_BUTTON3 )
                                fillScreen( GREEN );

                        drawPixel( x, y, dragcolour );

                        x += dx;
                        y -= dy;
                        x = max( 0, min( x, MAX_X_PIXELS-1 ) );
                        y = max( 0, min( y, MAX_Y_PIXELS-1 ) );

                        drawPixel( x, y, RED );
                }

        }

}
```

```
/* Game of Life demo */

#include "259macros.h"



void paintLife( unsigned short life[MAX_X_PIXELS][MAX_Y_PIXELS] )
{
        int x, y, colour;

        for( y=0; y < MAX_Y_PIXELS; y++ ) {
                for( x=0; x < MAX_X_PIXELS; x++ ) {
                        colour = (unsigned short)life[x][y];
                        drawPixel( x, y, colour );
                }
        }
}



int countNeighbours( unsigned short life[MAX_X_PIXELS][MAX_Y_PIXELS], int x, int y )
{
#define MX (MAX_X_PIXELS-1)
#define MY (MAX_Y_PIXELS-1)
        int count=0;
        if( x>0 && y>0 &&                 life[x-1][y-1] ) count++;
        if( x>0 &&                        life[x-1][y  ] ) count++;
        if( x>0 &&                y<MY && life[x-1][y+1] ) count++;
        if(        y>0 &&                 life[x  ][y-1] ) count++;

        if(                       y<MY && life[x  ][y+1] ) count++;
        if(        y>0 && x<MX &&         life[x+1][y-1] ) count++;
        if(               x<MX &&         life[x+1][y  ] ) count++;
        if(               x<MX && y<MY && life[x+1][y+1] ) count++;
        return count;
}



void life()
{
        int x, y, neighbours;
        unsigned short *pPixel;

        unsigned short BIRTH, DEATH, ALIVE;

        unsigned short prevLife[MAX_X_PIXELS][MAX_Y_PIXELS];    // cells:  dead = 0, alive > 0
        unsigned short nextLife[MAX_X_PIXELS][MAX_Y_PIXELS];    // next generation of cells

        BIRTH = makeColour(48, 0,63 );
        DEATH = makeColour( 0, 0, 0 );
        ALIVE = makeColour( 0,63, 0 );

#if 0
        // some chaotic initial pattern
        for( y=0; y < MAX_Y_PIXELS; y++ ) {
                for( x=0; x < MAX_X_PIXELS; x++ ) {
                        prevLife[x][y] = (x+y)&9 ? ALIVE : DEATH;
                }
        }
#else
        // read initial pattern from current VGA buffer
        for( y=0; y < MAX_Y_PIXELS; y++ ) {
                for( x=0; x < MAX_X_PIXELS; x++ ) {
                        pPixel = getPixelAddr(x,y);
                        prevLife[x][y] = *pPixel;
```

```
                }
        }

#endif


        while(1) {

                timedDelay( 300 * ONE_MS );

                paintLife( prevLife );

                // Compute Rules of Life
                // Next generation depends on previous generation
                //   1) cells with 0 or 1 neighbours die from loneliness
                //   2) cells with >= 4 neighbours die by overpopulation
                //   3) cells with 2 or 3 neighbours survive
                //   4) births occur at empty cells with exactly 3 neighbours

                for( y=0; y < MAX_Y_PIXELS; y++ ) {
                        for( x=0; x < MAX_X_PIXELS; x++ ) {
                                neighbours = countNeighbours( prevLife, x, y );
                                if( prevLife[x][y] == DEATH ) {
                                        // was previously dead. comes alive
                                        // if only 3 neighbours alive
                                        if( neighbours == 3 )
                                                nextLife[x][y] = BIRTH;
                                        else
                                                nextLife[x][y] = DEATH;
                                } else { // cell is alive
                                        if( neighbours < 2 )
                                                nextLife[x][y] = 0;
                                        else if( neighbours >= 4 )
                                                nextLife[x][y] = DEATH;
                                        else
                                                nextLife[x][y] = ALIVE;
                                }
                        }
                }

                // copy next generation to previous generation
                for( y=0; y < MAX_Y_PIXELS; y++ ) {
                        for( x=0; x < MAX_X_PIXELS; x++ ) {
                                prevLife[x][y] = nextLife[x][y];
                        }
                }

        }

}



int main( int argc, char *argv[] )
{
        // do not use initScreen() if you want to use old screen contents as
        // the initial values for the game of life
        // initScreen();

        life();
}
```

```
/* Pong demo */

#include "259macros.h"



#define PADDLE_LENGTH 25

unsigned short BLACK, WHITE, RED, GREEN, BLUE, MGNTA;




int moveBall( int A, int *pAv, int MAX_PIXELS, int SCALE )
{
        A += *pAv;
        if( A < 0 || A >= SCALE * MAX_PIXELS ) {
                *pAv = -*pAv;
                A += *pAv;
        }
        return A;
}


int drawPaddle( int px, int len, unsigned short colour )
{
        int x;
        for( x=1; x < MAX_X_PIXELS-1; x++ ) {
                if( x < px || x >= (px + len) )
                        drawPixel( x, MAX_Y_PIXELS-1, BLACK );
                else
                        drawPixel( x, MAX_Y_PIXELS-1, colour );
        }
}


void drawPixelScale( int x, int y, int colour, int scale )
{
        drawPixel( x/scale, y/scale, colour );
}


int collision( int Ax, int Ay, int Ascale, int Bx, int By, int Bscale )
{
        Ax = Ax/Ascale; Ay = Ay/Ascale;
        Bx = Bx/Bscale; By = By/Bscale;
        if( Ax==Bx && Ay==By ) return 1;
        return 0;
}


void flashRedScreen()
{
        fillScreen( RED );
        timedDelay( 17 * ONE_MS );
        fillScreen( BLACK );
        timedDelay( 17 * ONE_MS);
        fillScreen( RED );
        timedDelay( 17 * ONE_MS);
        fillScreen( BLACK );
}



int main( int argc, char *argv[] )
{
        unsigned int i,j,k;

        int pause = 8 * ONE_MS, tpause ;
        int ASCALE = 11, BSCALE=17, PSCALE=2;
        int Ax=0, Ay=0, Avx=7, Avy=3;
        int Bx=40, By=40, Bvx=5, Bvy=-3;
        int px, paddleX = 0, movePaddle = 0;
```

```
        int oldAx, oldAy, oldBx, oldBy;

        unsigned short Acolour;
        unsigned short Bcolour;

        int drawPaddleFlag=0;


        i = 0xfffffffc;
        j = 0xffffffff;


        BLACK = makeColour( 0, 0, 0 );  /* use gray to see the borders better */
        WHITE = makeColour(63,63,63 );
        RED   = makeColour(63, 0, 0 );
        GREEN = makeColour( 0,63, 0 );
        MGNTA = makeColour(63, 0,63 );
        BLUE  = makeColour( 0, 0,63 );

        Acolour = GREEN;
        Bcolour = MGNTA;


        initScreen();

    while(1) {

        timedDelay( 1000 * ONE_MS );


        Ax=0, Ay=0, Avx=7, Avy=3;
        Bx=40, By=40, Bvx=5, Bvy=-3;


        fillScreen( BLACK );

        drawPaddle( paddleX/PSCALE, PADDLE_LENGTH, WHITE );

        while(1) {

                k = j>>31;
                j = (j<<1) | (i>>31);
                i = (i<<1) | k;

                *pHEX7SEGA = i;
                //*pHEX7SEGA = (DIGIT0<<24)|(DIGIT2<<16)|(DIGIT5<<8)|DIGIT9;

                *pHEX7SEGB = j;
                //*pHEX7SEGB = (DIGITE<<24)|(DIGITE<<16)|(DIGITC<<8)|DIGITE;



                // draw border outline on 3 sides
                for( px=0; px<MAX_X_PIXELS; px++ ) {
                        drawPixel( px, 0, WHITE );
                }
                for( px=0; px<MAX_Y_PIXELS; px++ ) {
                        drawPixel( 0, px, WHITE );
                        drawPixel( MAX_X_PIXELS-1, px, WHITE );
                }

    //          drawPixelScale( Ax, Ay, BLACK, ASCALE );
    //          drawPixelScale( Bx, By, BLACK, BSCALE );
                oldAx = Ax/ASCALE; oldAy = Ay/ASCALE;
                oldBx = Bx/BSCALE; oldBy = By/BSCALE;


                Ax = moveBall( Ax, &Avx, MAX_X_PIXELS, ASCALE );
                Ay = moveBall( Ay, &Avy, MAX_Y_PIXELS, ASCALE );
                Bx = moveBall( Bx, &Bvx, MAX_X_PIXELS, BSCALE );
                By = moveBall( By, &Bvy, MAX_Y_PIXELS, BSCALE );

                if( collision(Ax,Ay,ASCALE,Bx,By,BSCALE) ) {
                        flashRedScreen();
                        drawPaddleFlag = 1;
                }
```

```
                if( (Ay/ASCALE == MAX_Y_PIXELS-1) &&
                ( paddleX/PSCALE <= Ax/ASCALE && Ax/ASCALE <= paddleX/PSCALE + PADDLE_LENGTH-1 ) ) {
                        drawPaddleFlag=1;
                } else if( (Ay/ASCALE == MAX_Y_PIXELS-1) && Avy > 0 ) {
                        Avy = 0; Avx = 0;               // Dead ball !
                        flashRedScreen();
                        drawPaddleFlag=1;
                }

                if( (By/BSCALE == MAX_Y_PIXELS-1) &&
                ( paddleX/PSCALE <= Bx/BSCALE && Bx/BSCALE <= paddleX/PSCALE + PADDLE_LENGTH-1 ) ) {
                        drawPaddleFlag=1;
                } else if( (By/BSCALE == MAX_Y_PIXELS-1) && Bvy > 0 ) {
                        Bvy = 0; Bvx = 0;               // Dead ball !
                        flashRedScreen();
                        drawPaddleFlag=1;
                }

                // sometimes the ball leaves a ghost image. erase it.
                if( ((oldAx!=Ax/ASCALE)||(oldAy!=Ay/ASCALE))&&(oldAy!=MAX_Y_PIXELS-1) ) {
                        drawPixel( oldAx, oldAy, BLACK );
                        //drawPixelScale( oldAx, oldAy, BLACK, ASCALE );
                        drawPaddleFlag=1;
                }
                if( ((oldBx!=Bx/BSCALE)||(oldBy!=By/BSCALE))&&(oldBy!=MAX_Y_PIXELS-1) ) {
                        drawPixel( oldBx, oldBy, BLACK );
                        //drawPixelScale( oldBx, oldBy, BLACK, BSCALE );
                        drawPaddleFlag=1;
                }

                // draw paddle after screen updates
                if( drawPaddleFlag ) {
                        drawPaddle( paddleX/PSCALE, PADDLE_LENGTH, WHITE );
                        drawPaddleFlag=0;
                }

                // draw balls after paddle
                drawPixelScale( Ax, Ay, Acolour, ASCALE );
                drawPixelScale( Bx, By, Bcolour, BSCALE );

                // time delay to keep gameplay realistic
                tpause = pause;
                if( (*pKEY&2) ) tpause = 40 * ONE_MS;   // super slowdown key
                timedDelay( tpause );
                pause = max( pause-100, 1 * ONE_MS );           // game play speeds up as time goes on

                // check if we should move the paddle
                movePaddle = 0;
                if( (*pKEY&8) && paddleX>0 )
                        movePaddle = -1;
                if( (*pKEY&4) && paddleX/PSCALE<MAX_X_PIXELS-PADDLE_LENGTH-1 )
                        movePaddle = +1;

                // move the paddle by 0, +1, or -1 pixels.
                // do it efficiently: erase 1 pixel on one side, grow a pixel on other side
                if( (paddleX+movePaddle)/PSCALE != (paddleX/PSCALE) ) {
                        if( movePaddle < 0 ) {
                                drawPixel( (paddleX-1)/PSCALE, MAX_Y_PIXELS-1, WHITE );
                                drawPixel( paddleX/PSCALE+PADDLE_LENGTH-1, MAX_Y_PIXELS-1, BLACK );
                        } else if( movePaddle > 0 ) {
                                drawPixel( paddleX/PSCALE, MAX_Y_PIXELS-1, BLACK );
                                drawPixel( paddleX/PSCALE+PADDLE_LENGTH, MAX_Y_PIXELS-1, WHITE );
                        }
                }
                paddleX += movePaddle;


                if( !Avx && !Avy && !Bvx && !Bvy )
                        break;  // end game
        }

} // restart game

}
```