`start_explosion(x,y)`	Begin explosion at location given by x, y coordinates
`swap_screens()`	Swap video screens.
`unpackbits(srcptr, destptr, count)`	Interface to Macintosh ROM routine, not in Megamax library.
`updatewind(wp)`	Update primary screen with expanded Macpaint picture.

(concluded)

/* Main program segment */

```
/*  Megaroids for the Macintosh is supplied courtesy of
    Megamax, Inc. It was written by Mike Bunnell and Mitch
    Bunnell, 1985.  */
#include <qd.h>
#include <qdvars.h>
#include <win.h>
#include <menu.h>
#include <event.h>
#include <dialog.h>
#define LASTMENU 6
#define APPLEMENU 1
#define FILEMENU 256
#define STARTMENU 2
#define INFOMENU 3
#define HISCOREMENU 4
#define SCOREMENU 5
#define CURPAGEOPTION 0x936
#define NULL 0L
#define TRUE 1
#define FALSE 0
#define SCORE_DIGITS 6
extern int hiscore[SCORE_DIGITS];
char score_string[SCORE_DIGITS+1];
menuhandle mymenus[LASTMENU+1];
rect screenrect, dragrect, prect;
boolean doneflag, temp;
eventrecord myevent;
```

EXHIBIT 11-2 megaroids.c: arcade game *(continues)*

```
int code, refnum;
windowrecord wrecord;
windowptr mywindow;
windowptr whichwindow;
int themenu, theitem;
char *progname = "megaroids";
#define SCREEN 0x7a700
#define SCREEN2 0x72700
#define O_RDONLY 1
#define O_RDWR 3
#define O_BINARY 8192
#define MAXSIZE 10000
char *free_info[] = {
     "This program is free.",
     " ",
     "You may make as many copies as you like.",
     " ",
     "It is not for resale",
     NULL
    };
char *Megamax_info[] = {
     "The Megamax C compiler is full K & R + common
      extensions",
     "Its features include: ",
     "  full floating point, one pass compilation, optimized
         for",
     "  68000, smart linker (only loads library routines
         called)",
     "  librarian, dynamic overlays, creates stand alone MAC",
     "  applications, in-line assembly language, batch
         facility,",
     "  nothing left to buy, no license fees.",
     " ",
     "All for only 299.95 (ed. and quantity discounts
      available)",
     "Megamax, Inc., P. O. Box 851521, Richardson, TX
      75085-1521",
     "Phone: (214) 987-4931   Dealer inquiries welcome.",
     NULL
    };
```

EXHIBIT 11-2 *(continued)*

```
char *source_info[] = {
    " ",
    "A commented source code listing (bound hardcopy
     listing)",
    "identical to this chapter is also available for $25
     from: ",
    " ",
    "                              Megamax Inc.",
    "                              P. O. Box 31294",
    "                              Dallas, TX  75231-0294",
    NULL
   };
/*  This routine was not in the library. This is the
    interface to the Macintosh Toolbox ROM routine.  */
```

unpackbits (srcptr, destptr, count)

```
char **srcptr, **destptr;
int count;
{
    asm {
            move.l   srcptr(A6), -(A7)
            move.l   destptr(A6), -(A7)
            move.l   count(A6), -(A7)
            dc.w     0xa8d0
    }
}
char *inbuf;
char destbuf[72];
bit srcbitmap;
#define START 116
#define LEFT 5
/*  This routine reads in the high score and a Macpaint
    formatted picture of the title screen out of the data
    fork of the program. It stores the high score in
    hiscore[] and stores the picture in the unused memory
    between the two graphic screens. It does this to make use
    of the space that is lost when using the second screen
    (The screen takes up about 22k but if you use this, you
    lose an even 32k). */
```

EXHIBIT 11-2 *(continued)*

getpaintdoc (filename)

```
char *filename;
{
     int f;
     int i;
     inbuf = (char *) (SCREEN2 + 21888);
      /* space between screens */
     f = open(filename, O_RDONLY | O_BINARY);
     if (f != -1) {
          read(f, score_string, SCORE_DIGITS);
          score_string[SCORE_DIGITS] = 0;
          for (i = 0; i < SCORE_DIGITS; i++) {
             hiscore[i] = score_string[SCORE_DIGITS - 1 - i]
              - '0';
          }
          read(f, inbuf, 512 - (SCORE_DIGITS));
           /* read past header info */
          read(f, inbuf, MAXSIZE);
           /* read in the rest of the file */
          close (f);
/*   second screen - the area that the later expansion will go
     in */
          srcbitmap.baseaddr = (qdptr) SCREEN2;
          srcbitmap.rowbytes = 64;
          srcbitmap.bounds.a.top = 0;
          srcbitmap.bounds.a.left = 0;
          srcbitmap.bounds.a.bottom = 342;
          srcbitmap.bounds.a.right = 512;
     }
}
/*   Expand Macpaint document format from the buffer between
     the screens onto the alternate screen.  */
```

expandpic()

```
{
     char *screenptr, *dptr, *sptr;
     int i, j;
     sptr = inbuf;
```

EXHIBIT 11-2 ***(continued)***

```
    screenptr = (char *) SCREEN2;
    for (i=0; i < 342 + START; i++) {
        dptr = destbuf;
        unpackbits (&sptr, &dptr, 72);
        if (i >= START)
           for (j = 0; j < 64; j++) {
                *screenptr++ = destbuf[j + LEFT];
            }
    }
}
/*  This routine writes out the selected array of text onto
    the dialog window. (Used when reading info)  */
```

put_text (which)

```
int which;
{
    char **p;
    int y;
    if (which == 1)
        p = free_info;
    else if (which == 2)
        p = Megamax_info;
    else
        p = source_info;
    y = 16;
    while (*p) {
        moveto (2, y);
        drawstring (*p);
        y += 16;
        p++;
    }
}
/*  Copy the title screen, which has been expanded onto the
    alternate screen, to the primary screen.  */
```

updatewind (wp)

```
windowptr wp;
{
```

EXHIBIT 11-2 *(continued)*

```
    copybits(&srcbitmap, &wp->portbits, &wp->portrect,
                    &wp->portrect, srccopy, 0L);
}
/*  Put all the information into the menu bar.
    Set up an array so that the information can be redrawn
    easily.  */
```

setupmenus ()

```
{
    int i;
    char appletitle[2];
    appletitle[0] = applesymbol; appletitle[1] = 0;
    mymenus[1] = newmenu(APPLEMENU, appletitle);
    addresmenu(mymenus[1], "DRVR");
    mymenus[2] = newmenu(FILEMENU, "File");
    appendmenu(mymenus[2], "Quit");
    mymenus[3] = newmenu(STARTMENU, "Start");
    appendmenu[mymenus[3], "1 Game;3 Games");
    mymenus[4] = newmenu(INFOMENU, "Info");
    appendmenu(mymenus[4], "Free;Megamax C;Source Code");
    mymenus[5] = newmenu(HISCOREMENU, "High Score");
    appendmenu(mymenus[5], "Clear High Score");
    mymenus[6] = newmenu(SCOREMENU, score_string);
    for (i=1; i <=LASTMENU; i++)
     /* this actually draws the menu bar */
        insertmenu(mymenus[i], 0);
    drawmenubar();
}
/*  Handle response to selecting an item from the menu */
```

docommand (themenu, theitem)

```
int themenu, theitem;
{
    char name [256];
    int i;
    int item = 0;
    dialogptr  dp;
```

EXHIBIT 11-2 *(continued)*

```
    switch (thememu)  {
    case APPLEMENU:
        getitem(mymenus[1], theitem, name);
        refnum = opendeskacc (name);
        break;
    case FILEMENU:
            doneflag = 1;
        break;
    case STARTMENU:
        switch (theitem)  {
            case 1:  doneflag = 2; break;
            case 2:  doneflag = 3; break;
            }
        break;
    case INFOMENU:
        dp = getnewdialog(100, NULL, -1L);
        setport (dp);
        put_text (theitem);
        do {
                systemtask();
                getnextevent(everyevent, &myevent);
                if (myevent.what == updateevt)
                    put_text(theitem);
                modaldialog(NULL, &item);
        }       while ( item != 1);
        disposdialog (dp);
        break;
    case HISCOREMENU:
        for (i=0; i < SCORE_DIGITS; i++) {
            hiscore[i] = 0;
            score_string[i] = '0';
        }
        deletemenu (SCOREMENU);
        disposemenu (mymenus[6]);
        mymenus[6] = newmenu(SCOREMENU, score_string);
        insertmenu(mymenus[6], 0);
        drawmenubar();
        break;
    }
    hilitemenu(0);
}
```

EXHIBIT 11-2 *(continued)*

dowindowstuff()

```
{
    static int first = TRUE;
    grafptr wmgrport;
    eventrecord tempevent;
    getwmgrport(&wmgrport);
     /* draw background wiped out by game */
    setport(wmgrport);
    fillrgn(*(handle *(0x9eeL, 0xa3cL);
     /* grayrgn & desk pattern */
    flushevents(everyevent, 0);
    setupmenus();
    initcursor();
    setrect(&screenrect, 4, 24, 508, 338);
    doneflag = 0;
    mywindow=newwindow(&wrecord, &screenrect, "Megaroids", 1,
     3, (long)-1, 1, (long)0);
     /* name not used for this window */
    setport (mywindow);
    expandpic();
    update(mywindow);
    if (first) /* done only at first invocation */
        initgamestuff();
        unloadseg (initgamestuff);
    }
    first = FALSE;
    do {            /* main event loop */
            systemtask();
            temp = getnextevent(everyevent, &myevent);
            switch (myevent.what)   {
                    case mousedown:
                    code=findwindow(&myevent.where,
                     &whichwindow);
                    switch (code)  {
                        case inmenubar:
                            docommand
                             (menuselect(&myevent.where));
                             break;
                        case insyswindow:
                            systemclick(&myevent,
                             whichwindow); break;
```

EXHIBIT 11-2 ***(continued)***

```
                    }
                    break;
                    case updateevt:
                        setport(mywindow);
                        beginupdate (mywindow);
                        updatewind (mywindow);
                        endupdate (mywindow);
                    break;
        }
    } while (doneflag == 0);
    return (doneflag);
}

main()

{
    int retval, i;
    int f;
    long stackbase;
    struct {
        char *pgname;
        int pageoption;
    } launchrec;
    launchrec.pgname = progname;
     /*program launches itself if it */
    launchrec.pageoption = -1;
     /* doesn't have both screens */
    if (*((int *) CURPAGEOPTION) >= 0)
        launch (&launchrec);
    asm {
     /* set stack to 3k (normally 8k) */
            move.l  2312, stackbase (A6)
    }
    setappllimit (stackbase - 3096);
    initgraf (&theport);   /* initialization */
    initfonts();
    initwindows ();
    initmenus();
    teinit();
    initdialogs (NULL);
    getpaintdoc (progname);
    do {
```

EXHIBIT 11-2 *(continued)*

```
        retval = dowindowstuff();
         /* this returns if game is asked for */
        closewindow (mywindow);
         /* get rid of all memory allocated */
        clearmenubar()              /* will be added later */
        for (i=1; i <= LASTMENU; i++)
            disposemenu(mymenus[i]);
        if (retval != 1)  {
            if (retval == 2)
                game();
            else
                for (i = 0; i < 3; i++)
                        game();
            for (i = 0; i <= SCORE_DIGITS; i++)  {
                    score_string[SCORE_DIGITS-1-1] =
                     hiscore[i] + '0';
            }
            unloadseg(game);
        }  while (retval != 1);
        f = open (progname, O_RDWR | O_BINARY);
          /* save high score */
        if (f != -1)  {
            lseek (f, 0L, 0);
            write (f, score_string, SCORE_DIGITS);
            close (f);
        }
        disposptr (inbuf);
}
```

/* Game overlay segment */

```
/*  This is the main part of the game itself.
    All of the game is in a different code segment called
    game so that on a 128k Mac there is room if a person
    wants to use a desk accessory. There is not that much
/*  room left on a 128k Mac when using both screens.  */
```

overlay "game"

```
#include <event.h>
```

EXHIBIT 11-2 *(continued)*

```
/*  Locations of the video primary and secondary screens.
    Note that on a 128k Mac these are actually aliases
    for 0x1a700 and 0x12700  */
#define SCREEN1 0x7a700
#define SCREEN2 0x72700
#define SHEIGHT 29 /* height of player's ship in pixels */
#define SCENTER 14
 /* x and y coordinates of center of ship */
#define NUMSHOTS 5 /* total number of shots allowed */
#define NUMDOTS 5  /* number of dots in an explosion */
#define NUMEXPLOSIONS 4
 /* max number of simultaneous explosions */
#define NUMFRAMES 4 /* number of frames in an explosion */
#define SHOT_LIFE 320/6  /* shot duration in 1/60 sec */
#define SMALLHEIGHT 11
 /* height of small asteroid in pixels */
#define MEDHEIGHT 22 /* height of medium asteroid */
#define BIGHEIGHT 41 /* height of large asteroid */
#define N 30
 /* maximum number of asteroid records */
#define TRUE 1
#define FALSE 0
#define NULL 0L
#define SCORE_DIGITS 6 /* number of digits in the score */
/* Note:  all time measurements in 1/60th of a second
   increments*/
#define DONE_TIME 300
 /* pause after last ship before returning */
#define HYPER_TIME 60
 /* time before returning from hyperspace */
#define DEAD_TIME 180 /* time before starting a new ship */
#define START_TIME 120 /* time before doing anything */
#define START_BIGS 4 /* initial number of large asteroids */
#define MAX_BIGS 6    /* maximum number of large asteroids */
#define MIN_ROOM 60
 /* minimum clearance at middle of screen */
#define START_SHIPS 3  /*initial number of ships */
#define CENTERX  (512/2 - SCENTER)
 /* initial location of ship */
#define CENTERY  (342/2 - SCENTER)
#define ALIEN_TIME  (20*60) /* time till next alien time */
```

EXHIBIT 11-2 ***(continued)***

```
#define FAST_BUMP (10*60)          /* time in 1/3rd second */
#define ASHEIGHT   18 /* height of big alien ship */
#define ASWIDTH  30    /* width of big alien ship */
#define LASHEIGHT  10 /* height of little alien ship */
#define LASWIDTH 14    /* width of little alient ship */
/*  These are the external declarations of the figure
    definition arrays used in both drawing & initialization
    routines  */
extern int ships[16] [16*4];
extern int f1ships [16] [16*4];
extern int f2ships [16] [16*4];
extern int small_meteor[];
extern int medium_meteor[];
extern int big_meteor[];
extern int small_shadowmask[];
extern int medium_shadowmask[];
extern int big_shadowmask[];
extern short digits[];
extern long alien[];
extern long small_alien[];
/*  external variables used in sound routines */
extern int wsound;      /* current sound */
extern int running;
 /* TRUE if a sound is currently playing */
int startsound;
 /* if TRUE call sound routine this cycle */
/*  These are the definitions for the different sounds that
    are available from the sound routines. The higher the
    number, the higher the priority. In other words,
    BSAUCER_SOUND (big saucer sound) cannot be interrupted
    by the explosion sound but it can interrupt the
    explosion sound and the shot sound */
#define LBUMP_SOUND  0
#define HBUMP_SOUND  1
#define EXPL_SOUND  3
#define BSAUCER_SOUND 4
#define LSAUCER_SOUND 5
#define SHOT_SOUND 2
#define NEWSHIP_SOUND 6
/*  chscrn contains the address of the VIA port used to
    change between the primary and secondary screens  */
```

EXHIBIT 11-2 ***(continued)***

```
char *chscrn = (char *) 0xeffffe;
/*  front_screen will contain the address of the graphic
    screen currently visible. The back_screen will contain
    the other screen. Note: information is always drawn on
    the back screen.  */
char *front_screen, *back_screen;
int primary_screen;
 /* used to toggle between screens */
long shipx, shipy;          /* location of ship *256 */
int intshipx, intshipy;     /* location of ship in pixels */
int oldshipx, oldshipy;     /* last location in pixels */
int  rot_pos;
 /* current rotated position of ship (0-15) */
int shipxv, shipyv;         /* x and y velocity of ship */
int score[SCORE_DIGITS];
 /* each digit occupies one word */
int hiscore[SCORE_DIGITS];
int  hyper_count;
 /* time until return from hyperspace */
int done_count;          /* time until return from game */
int start_count;         /* time until start of game */
int num_ships;           /* number of ships left */
int hx, hy;              /* location to hyperspace to */
int dead;                /* true if ship just destroyed */
int ashipx, ashipy;      /* location of alien ship */
int oldashipx, oldashipy; /* last location of alien ship */
int ashipxv;              /* velocity of alien ship */
int a_count;
 /* amount of time until next alien ship */
int aship_dead;
 /* set true if alien ship gets hit */
int safe;
 /* ship allowed to start out safely */
int little_ship;
 /* if true little alien ship appears; */
 /* if false a large alien ship appears */
int aship_counter;          /* y location of alien ship */
/* x (and y off 90 degrees) acceleration provided by
 thrusting */
int addx[16] = {0, 6, 11, 15, 16, 15, 11, 6, 0,
                -6, -11, -15, -16, -15, -11, -6};
```

EXHIBIT 11-2 *(continued)*

```
/* x velocities (and y off 90 degrees) given to shots
 when fired */
int shotaddx[16] = {0, 2, 3, 5, 5, 5, 3, 2, 0,
                    -2, -3, -5, -5, -5, -3, -2};
/*  Location relative to top left-hand corner of the ship
    of the three corners of the ship. These points are used
    to see if the ship has run into something.  */
int check1x[16] = {0+15, 2+15, 5+15, 8+15,9+15,8+15, 5+15,
                   2+15, 0+15, -2+15, -5+15, -8+15,-9+15,
                   -8+15, -5+15, -2+15};
int check2x[16] = {-3+15, -6+15, -5+15, -5+15,-3+15,-1+15,
                   0+15, 3+15, 3+15, 6+15, 5+15,5+15,
                   3+15, 1+15, 0+15, -3+15};
int check3x[16] = {3+15, 3+15, 0+15, -1+15,-3+15,-5+15,
                   -5+15, -6+15, -3+15, -3+15, 0+15, 1+15,
                   3+15, 5+15, 5+15, 6+15};
/*  Definition of little ship figure that is beside the
    number of ships left digits (top left of screen.  */
short small_ship[] = {0xf7, 0xf7, 0xe3, 0xeb, 0xc9, 0xc9,
                      0x88,0x88, 0x88, 0xc9};
/* blank space used instead of leading zeros on number of
   ships */
    int blank[] = {0xffff, 0xffff, 0xffff, 0xffff,
     0xffff};
/* structure used in link list of asteroids */
        struct mrec {
        int     x, y;
        int     life_count;
        int     oldx, oldy;
        int     addx, addy;
        int     height;
        struct mrec *next;
    }   meteor [N];
typedef struct mrec *metptr;
metptr first, freeptr;
 /* first points to start of asteroid list */
 /* freeptr is used to allocate asteroid structs */
/* shifted definitions of 3 types of asteroids */
long small_mdefs[(4*SMALLHEIGHT*4)*2];
long medium_mdefs[(6(MEDHEIGHT*4)*2];
long big_mdefs[(8*BIGHEIGHT*4)*2];
```

EXHIBIT 11-2 ***(continued)***

```
/* This is an array of dot locations for the explosions */
struct dotrect {
    int x;
    int y;
} dots [NUMDOTS*(NUMFRAMES+1)] = {
       {0,-5}, {5,0}, {3,4}, {-5,6}, {-6,-3},
       {0,-10},{10,0},{6,8}, {-10,12},{-12,-6},
       {0,-15},{15,0},{9,12},{-15,18},{-18,-9},
       {0,-20},{20,0},{12,16},{-20,24},{-24,-12},
       {0,-25},{25,0},{15,20},{-25,30},{-30,-15},
    };
struct shotrec {  /* array of shot structures */
    int x, y;
    int life_count;
    ont oldx, oldy;
    int addx, addy;
} shot[NUMSHOTS+1];
struct exprec {  /* array of explosion structures */
    int x, y;
    int life_count;
    struct  dotrec *fig, *last_fig;
} explosion[NUMEXPLOSIONS];
typedef struct exprec *expptr;
```

init_meteors(num_bigs)

```
{
    register metptr mptr;
    metptr temp;
    register int i;
    while (first)  {
     /* get rid of any meteors still in list */
        temp = first -> next;
        first->next = freeptr;
        freeptr = first;
        first = temp;
    }
    for (i = 0; i <= num_bigs; i++)  {
        mptr = freeptr;
         /* take meteor off freeptr list */
```

EXHIBIT 11-2 *(continued)*

```
        freeptr = mptr->next; /* and add it to active list */
        mptr->next=first;
        first = mptr;
        if (random() & 8) {
         /* initials the number of large */
        mptr->x=random() & 511;
         /* asteroids required around */
        if (random() & 4)
         /* the edge of the screen */
           mptr->y = 341;
        else
           mptr->y = -BIGHEIGHT + 1;
     }
     else {
        mptr->y=random() % 342;
        if (random() & 4)
           mptr->x = 511;
        else
           mptr->x = -BIGHEIGHT + 1;
     }
     mptr->addx=(random() & 1) - 1;
     mptr->addx += mptr->addx >= 0;
     mptr->addy = (random() & 1) - 1;
     mptr->addy += mptr->addy >= 0;
     mptr->life_count = 5;
     mptr->height = BIGHEIGHT;
    }
}

clear_screen() /* clears both screens */

{
     register long *screenloc1, *screenloc2;
     register int counter;
     screenloc1 = (long *) SCREEN1;
     screenloc2 = (long *) SCREEN2;
     counter = 21888/4;
     do {
        *screenloc2++ = *screenloc1++ = ~0;
     } while (--counter);
}
```

EXHIBIT 11-2 ***(continued)***

```
swap_screens()

{
     primary_screen = !primary_screen
     if (primary_screen) {
          front_screen = (char *) SCREEN1;
          back_screen = (char *) SCREEN2;
          *chscrn |= 64;
     }
     else {
          front_screen = (char *) SCREEN2;
          back_screen = (char *) SCREEN1;
          *chscrn & = ~64;
     }
}
/* random number routine - does not require Quickdraw to be
   used */

int random()

{
     static long seed = 5671378192;
     long tickcount();
     seed = (seed << 3) ^ (seed >> 2) ^ tickcount();
     return seed;
}

erase_meteors()

{
     register struct mrec *mptr;
     mptr = first;
     while (mptr) {
          if (mptr->life_count <= 3) {
               if (mptr->height == BIGHEIGHT
                    bigerase_fig(mptr->oldx, mptr->oldy);
               else if (mptr->height == SMALLHEIGHT)
                    smallerase_fig(mptr->oldx, mptr->oldy);
               else
                    erase_fig(mptr->oldx, mptr->oldy,
                     mptr->height);
```

EXHIBIT 11-2 *(continued)*

```
        }
        mptr = mptr->next;
    }
}
```

add_meteor(mptr, second)

```
register struct mrec *mptr;
{
    register struct mrec *mptr2;
    static int ax, ay;
    int temp;
    mptr2 = freeptr;
     /* get a record of the freeptr list */
    freeptr = mptr2->next;
    mptr2->next = mptr->next;
     /* add the record to the active list */
    mptr->next = mptr2;
    if (mptr->height == MEDHEIGHT)
        mptr2->height = SMALLHEIGHT);
    else
        mptr2->height = MEDHEIGHT;
    if (second && ax != ay) {
        temp = ax;
        ax = ay;
        ay = temp;
    }
    else {
        ax = ((random() & 32767 % 3)  -1;
        ay = ((random() & 32767 % 3)  -1;
    }
    mptr2->addx=mptr->addx + ax;
    if (mptr2->addx == 0)
     /* make sure velocity is not zero */
        mptr2->addx = mptr->addx;
         /* in either x or y direction */
    mptr2->addy=mptr->addy + ay;
    if (mptr2->addy == 0)
        mptr2->addy = mptr->addy;
    mptr2->x = mptr->x + mptr2->height +
     (mptr -> addx <<(1 + second));
```

EXHIBIT 11-2 ***(continued)***

```
    mptr2->y = mptr->y + mptr2->height +
     (mptr -> addy <<(2 - second));
    mptr2->life_count = 5;
}
/* Check to see if the shot at location x,y has hit
   anything. If alien_shot == TRUE then check player's
   ship also. */
```

int hit(x, y, alien_shot)

```
register int x, y;
int alien_shot;
{
    register metptr mptr;
    mptr = first; /* check asteroids */
    while (mptr) {
        if (mptr->x <= x && mptr->x + mptr->height >= x &&
            mptr->y <= y && mptr->y + mptr->height >= y) {
                mptr->life_count=2;
                return TRUE;
        }
        mptr = mptr->next;
    }
    if (alien_shot) {
        if (intshipx + (SCENTER-10) <= x && intshipx
             +(SCENTER+10) >= x &&
            intshipy + (SCENTER-10) <= y && intshipy
             +(SCENTER+10) >= y) {
            dead = TRUE;
            return TRUE;
        }
    }
    else {
            if (little_ship) {
                if (!a_count && ashipx <= x && ashipx +
                     LASWIDTH >= x &&
                    ashipy <= y && ashipy + LASHEIGHT >= y) {
                    aship_dead = TRUE;
                    add_to_score (5, 2);
                    add_to_score (1, 3);
                    return TRUE;
```

EXHIBIT 11-2 ***(continued)***

```
                }
            }
            else {
                if (!a_count && ashipx <= x && ashipx +
                     ASWIDTH >= x &&
                    ashipy <= y && ashipy + ASHEIGHT >= y) {
                    aship_dead = TRUE;
                    add_to_score (5, 2);
                    return TRUE;
                }
            }
    }
    return FALSE;
}

int abs(x) /* absolute value */

int x;
{
x > 0 ? x : -x;
}
/*
    return TRUE if there is an asteroid less than MIN_ROOM
    from where the ship is going to appear.
*/

int no_room()

{
    register metptr mptr;
    mptr = first;
    while (mptr) {
        if (abs (mptr->x - hx) < MIN_ROOM &&
             abs(mptr->y - hy) < MIN_ROOM)
            return TRUE;
        mptr = mptr->next;
    }
    return FALSE;
}
/*  This routine draws the asteroids, moves them and
    manages their presence on the screen. This routine also
```

EXHIBIT 11-2 *(continued)*

```
    draws their explosions and has the task of starting new
    asteroids.  */

draw_meteors()

{
    register metptr mptr;
    metptr last_mptr;
    register int j;
    int farx, fary;
    int half_height;
    mptr = first;
    while (mptr) {
        mptr->oldx=mptr->x;
        mptr->oldy=mptr->y;
        if (!a_count) {
             /* check to see if alien runs into asteroid */
            if (little_ship) {
                if (mptr->x - ashipx < LASWIDTH
                    && ashipx - mptr->x < mptr->height
                    && mptr->y - ashipy < LASHEIGHT
                    && ashipy - mptr->y < mptr->height) {
                    aship_dead = TRUE;
                    mptr->life_count = 2;
                     /* will die in two cycles */
                }
            }
            else {
                if (mptr->x - ashipx <ASWIDTH
                    && ashipx - mptr->x < mptr->height
                    && mptr->y - ashipy < ASHEIGHT
                    && ashipy - mptr->y < mptr->height) {
                    aship_dead = TRUE;
                    mptr->life_count = 2;
                     /* will die in two cycles */
                }
            }
        }
        mptr->life_count--;
        if (mptr->life_count == 0) { /* asteroid is dead */
            if (mptr->height != SMALLHEIGHT) {
```

EXHIBIT 11-2 *(continued)*

```
            if (mptr->height !=BIGHEIGHT) {
                start_explosion (mptr->x + BIGHEIGHT/2,
                                 mptr->y + BIGHEIGHT/2);
                add_to_score(2,1); /* add 20 to score */
            else {
                start_explosion (mptr->x + MEDHEIGHT/2,
                                 mptr->y + MEDHEIGHT/2);
                add_to_score (6,1);
                 /* add 60 to score */
            }
            add_meteor (mptr, 0);
            add_meteor (mptr, 1);
        }
        else {
            start_explosion (mptr->x + SMALLHEIGHT/2,
                             mptr->y + SMALLHEIGHT/2);
            add_to_score (2,2); /* add 200 to score */
        }
        if (mptr == first) {  /* remove meteor record */
            first = mptr->next;
            mptr->next = freeptr;
            freeptr = mptr;
            mptr = NULL;
        }
        else {
            last_mptr->next = mptr->next;
            mptr->next = freeptr;
            freeptr = mptr;
            mptr = last_mptr;
        }
    }
    else if (mptr->life_count > 1) {
        if (mptr->life_count == 2)
             /*don't let asteroid go away */
            mptr->life_count++;
             /*unless it was killed */
        mptr->x += mptr->addx;
         /* move asteroid to next location */
        mptr->y += mptr->addy;
        if (mptr->x >= 512)
            mptr->x 1-mptr->height;
        else if (mptr->x <= -mptr->height)
```

EXHIBIT 11-2 ***(continued)***

```
                mptr->x = 511;
            if (mptr->y >= 342)
                mptr->y = 1-mptr->height;
            else if (mptr->y <= -mptr->height)
                mptr->x = 341;
            if (mptr->height == BIGHEIGHT)
                 /*draw asteroid on screen */
                bigdraw_fast(mptr->x, mptr->y, big_mdefs);
            else if (mptr->height == SMALLHEIGHT)
                smalldraw_fast(mptr->x, mptr->y,
                 small_mdefs);
            else
                draw_fast(mptr->x, mptr->y, medium_mdefs);
        }
        last_mptr = mptr;
         /* get last_mptr for deletion purposes */
        if (mptr)                       /* go to next asteroid */
            mptr = mptr->next;
        else
            mptr = first;
    } ;
}
```

erase_shots()

```
{
    register struct shortrec *sptr;
    register int i;
    i = NUMSHOTS + 1;
    sptr = shot;
    do {
        if (sptr->life_count)
            erase_dot(sptr->oldx, sptr->oldy);
        sptr++;
    } while (--i);
}
```

int erase_explosions()

```
{
    register expptr, eptr;
```

EXHIBIT 11-2 *(continued)*

```
    register struct dotrec *dptr;
    register int i, j;
    i = NUMEXPLOSIONS;
    eptr = explosion;
    do {
        if (eptr->life_count){
            dptr = eptr->last_fig;
            j = NUMDOTS;
            do {
                erase_dot(eptr->x + dptr->x, eptr->y +
                 dptr->y);
                dptr++;
            } while (--j);
        }
        eptr++;
    } while (--i);
}
```

draw_shots() /* draw shots and move them */

```
{
    register struct shortrec *sptr;
    register int i;
    sptr=shot;
    i = NUMSHOTS+1;
    do {
        sptr->oldx = sptr->x;
        sptr->oldy = sptr->y;
        if (sptr->life_count && --sptr->life_count > 1) {
            sptr->x += sptr->addx;
            sptr->x &= 511;
            sptr->y += sptr->addy;
            if (sptr->y >= 342)
                sptr->y = 0;
            else if (sptr->y < 0)
                sptr->y = 340;
            /* if shot hit something then the shot goes
               away too */
            /* shot is drawn by check_dot too */
            /* (check_dot(sptr->x, sptr->y, TRUE) != ~0) {
                if (hit (sptr->x, sptr->y, i == 1))
```

EXHIBIT 11-2 *(continued)*

```
                    sptr->life_count = 2;
                }
            }
            sptr++;
        } while (--i);
}
```

draw_explosions()

```
{
    register expptr, eptr;
    register struct dotrec *dptr;
    register int i, j;
    static odd;
    i = NUMEXPLOSIONS;
    eptr = explosion;
    odd++;
    do {
        if (eptr->life_count) {
            eptr->last_fig = eptr->fig;
            if (odd & 1) {
                eptr->life_count--;
                eptr->fig += NUMDOTS;
            }
            if (eptr->life_count > 1) {
                dptr = eptr->fig;
                j = NUMDOTS;
                do {
                    draw_dot (eptr->x + dptr->x, eptr->y +
                     dptr->y);
                } while (--j);
            }
        }
        eptr++
    } while (--i);
}
```

start_explosion (x, y);

```
int x, y;
{
```

EXHIBIT 11-2 *(continued)*

```
    register int i;
    register expptr eptr;
    setasound(EXPL_SOUND);
    if (hyper_count)        /* your ship is dead */
        i = NUMEXPLOSIONS;
         /* all possible explosions can be used */
    else
        i = 2;
         /* only two simultaneous explosions otherwise */
    eptr = explosion;
     /* because not enough time for more */
    while (eptr->life_count && --i) {
        eptr++;
    }
    if (!eptr->life_count) {
        eptr->x = x;
        eptr->y = y;
        eptr->life_count = NUMFRAMES + 2;
        eptr->last_fig = eptr->fig = dots;
    }
}
/*  This routine draws the player's ship in the correct
    rotation and with the correct characteristics (e.g.
    flame). It also checks the keyboard and changes ship
    behavior accordingly and moves the ship too. It also
    handles the condition of the ship running into an
    asteroid. All in all, a busy little C function!  */

draw_ship()

{
    register struct shortrec *sptr;
    register int i, j;
    int *ship_fig;
    static int flamecount;
    long key;
    static int slow_rotate;
    static int slow_flame;
    static int key_down;
    int rot_posy;
    oldshipx = intshipx;  /* used for erasing purposes */
```

EXHIBIT 11-2 ***(continued)***

```
oldshipy = intshipy;
key = *((long *) 370);
if (key & 0xc000) {
    slow_rotate++;
    if (key & 0x4000) {       /* rotate left (the z key) */
        if (!(slow_rotate & 3))
            rot_pos = (rot_pos-1) & 15;
    }
    else
     /* rotate right (the x key) */
        if (!(slow_rotate & 3))
            rot_pos = (rot_pos+1) & 15;
}
else
    slow_rotate = 0;
rot_posy = (rot_pos - 4) & 15;
key = *((long *) 374);
if (key & 0x10)  {          /* thrust (the ? key) */
    shipxv += addx[rot_pos];
    shipyv += addx[rot_pos];
    if (slow_flame < 2)
        ship_fig = f1ships[rot_pos];
    else
        ship_fig = f2ships[rot_pos];
    if (slow_flame++ >= 4)
        slow_flame = 0;
}
else {
     ship_fig = ships[rot_pos];
     slow_flame =0;
}
shipxv -= shipxv>>7;  /* friction effects */
shipyv -= shipyv>>7;
shipx += shipxv;      /* move ship */
shipy += shipyv;
i = shipx>>8;
 /* i and j contain ship position in screen */
j = shipy>>8;             /* coordinates */
if ( i >= 512) {
    shipx = (1 - SHEIGHT)<<8) | 255;
    i = 1 - SHEIGHT;
```

EXHIBIT 11-2 *(continued)*

```
}
else if ( i <= -SHEIGHT) {
     shipx = 511<<8;
     i = 511;
}
if ( j >= 342) {
     shipy = (1 - SHEIGHT)<<8) | 255;
     j= 1 - SHEIGHT;
}
else if (j <= -SHEIGHT) {
     shipx = 341<<8;
     j = 341;
}
intshipx = i;
intshipy = j;
/* check all corners of the ship to see if it ran
   into something */
if (~check_dot(check1x[rot_pos]+i, check1x[rot_posy]+j,
      FALSE))
     if (hit(check1x[rot_pos]+i, check1x[rot_posy]+j,
           FALSE))
          dead = TRUE;
if (~check_dot(check2x[rot_pos]+i, check2x[rot_posy]+j,
      FALSE))
     if (hit(check2x[rot_pos]+i, check2x[rot_posy]+j,
           FALSE))
          dead = TRUE;
if (~check_dot(check3x[rot_pos]+i, check3x[rot_posy]+j,
      FALSE))
     if (hit(check3x[rot_pos]+i, check3x[rot_posy]+j,
           FALSE))
          dead = TRUE;
if (dead)
      /* note that dead can also be set by check_dot */
     hyper_count = DEAD_TIME;
     safe = TRUE;
     hx = CENTERX;
     hy = CENETRY;
     rot_pos = 0;
     start_explosion(i + SCENTER, j + (SCENTER - 7));
     start_explosion(i + SCENTER+6), j + (SCENTER + 4));
     start_explosion(i + SCENTER-6), j + (SCENTER + 4));
```

EXHIBIT 11-2 ***(continued)***

```
        /* subtract 1 from num_ships (one byte used for
           each digit) */
        if (num_ships & 0xff)
            num_ships--;
        else
            num_ships += 9 - 256;
        if (a_count)
            a_count = ALIEN_TIME;
    }
    draw_fig(i, j, ship_fig, SHEIGHT);
    key = *((long *) 378);
    if (key & 0x10000) {          /* fire button (shift key) */
        if (key_down) {
            /* only good if key goes from up to down */
            sptr = shot;
            for (i = 0; sptr->life_count && i < NUMSHOTS;
                 i++, sptr++);
                 /* if not all shots are out already */
            if (i < NUMSHOTS) {
                setasound(SHOT_SOUND);
                sptr->addx = shotaddx[rot_pos] +
                 (shipxv>>8);
                sptr->addy = shotaddx[rot_posy] +
                 (shipyv>>8);
                sptr->x = (shotaddx[rot_pos]<<1) +
                 intshipx + SCENTER;
                sptr->y = (shotaddx[rot_posy]<<1) + j +
                 SCENTER;
                sptr->life_count = SHOT_LIFE;
            }
        }
        key_down = FALSE
    }
    else key_down = TRUE;
    if (key & 0x2000000) {
         /* if spacebar down then hyperspace */
        hx = ((unsigned) random()) % (512-32);
        hy = ((unsigned) random()) % (342-32);
        hyper_count = HYPER_TIME;
        safe = FALSE;
    }
}
```

EXHIBIT 11-2 *(continued)*

```
/*  Return the angle (0-16) where x,y lies in respect to 0,0
    This routine is used by the small alien ship to aim at
    player.  */

int shot_angle (x, y)

int x, y;
{
    int angle;
    int absx, absy;
    absx = x > 0 ? x : -x;
    absy = y > 0 ? y : -y;
    if (absy > absx<<2)
        angle = 0;
    else if (absy >= absx<<1)
        angle = 1;
    else if (absx > absy<<2)
        angle = 4;
    else if (absx >= absy<<1)
        angle = 3;
    else
        angle = 2;
    if (x > 0) {
        if (y > 0)
            return 8 - angle;
        else
            return angle;
    }
    else {
        if (y > 0)
            return 8 + angle;
        else
            return (16 - angle) & 15;
    }
}
/*  draw the large alien ship */

draw_aship()

{
    register struct shotrec *sptr;
```

EXHIBIT 11-2 ***(continued)***

```
int rand;
static long *afig_def = alien;
static move_count;
static ashipyv;
setbsound(BSAUCER_SOUND);
oldashipx = ashipx;  /* for erasing purposes */
oldashipy = ashipy;
rand = random() & 3;
 /* change vertical direction every 32 dots */
if (!(ashipx & 31)) {
    if (ashipyv) {
        if (rand & 1)
            ashipyv = 0;
    }
    else {
        if (rand == 0)
            ashipyv = 1;
        else if (rand == 1)
            ashipyv = -1;
    }
}
if (ashipx < 64 || ashipx > (512-64))
    ashipyv = 0;
ashipy += ashipyv;          /* move ship */
ashipx += ashipxv;
if (ashipy <= -ASHEIGHT)
     /* ship wraps around edges of the screen */
    ashipy = 341;
else if (ashipy >= 342)
    ashipy = 1 - ASHEIGHT;
if (aship_dead || ashipx >= 512 || ashipx <= -32) {
    a_count = ALIEN_TIME;
    if (aship_dead) {
        start_explosion(ashipx + (ASWIDTH/2), ashipy +
         (ASHEIGHT/2));
    }
    return;
}
else
    draw_fig(ashipx, ashipy, afig_def, ASHEIGHT);
sptr = &shot[NUMSHOTS];
```

EXHIBIT 11-2 *(continued)*

```
     /* alien ship uses last shot in shot list */
    if (!sptr->life_count) {
     /* if shot not already shot */
        setasound(SHOT_SOUND);  /* then begin */
        rand = random() & 15;
        sptr->addx = shotaddx[rand]; /* shoot randomly */
        sptr->addy = shotaddx[(rand - 4) & 15];
        sptr->x = (sptr->addx<<1) + ashipx + (ASHEIGHT/2);
        sptr->y = (sptr->addy<<1) + ashipy + (ASHEIGHT/2);
        sptr->life_count = SHOT_LIFE;
    }
    move_count++;
     /* used for changing which big ship figure */
    if (move_count & 1) { /* to draw (for rotation) */
        if (shipxv == 1) {
            afig_def += ASHEIGHT;
            if (afig_def == alien + (ASHEIGHT*3))
                afig_def += -ASHEIGHT*3;  /* -= */
        }
        else {
            if (afig_def == alien)
                afig_def += ASHEIGHT*3;
            afig_def += -ASHEIGHT;   /* -= */
        }
    }
}
/*  This is the same as draw_aship(), but it draws the
    little alien ship. Instead of aiming shots randomly, it
    aims shots at the player's ship.  */
```

draw_little_ship()

```
{
    register struct shotrec *sptr;
    int rand;
    static long *afig_def = small_alien;
    static move_count;
    static ashipyv;
    setbsound(LSAUCER_SOUND);
    oldashipx = ashipx;  /*for erasing purposes */
```

EXHIBIT 11-2 *(continued)*

```
oldashipy = ashipy;
rand = random() & 3;
 /* change vertical direction every 32 dots */
if (!(ashipx & 31)) {
    if (ashipyv) {
        if (rand & 1)
            ashipyv = 0;
    }
    else {
        if (rand == 0)
            ashipyv = 1;
        else if (rand == 1)
            ashipyv = -1;
    }
}
if (ashipx < 64 || ashipx > (512-64))
    ashipyv = 0;
ashipy += ashipyv; /* move ship */
ashipx += ashipxv;
if (ashipy <= -LASHEIGHT)
     /* ship wraps around edges of the screen */
    ashipy = 341;
else if (ashipy >= 342)
    ashipy = 1 - LASHEIGHT;
if (aship_dead || ashipx >= 512 || ashipx <= -32) {
    a_count = ALIEN_TIME;
    if (aship_dead) {
        start_explosion(ashipx+(LASWIDTH/2), ashipy +
         (LASHEIGHT/2));
    }
    return;
}
else
    draw_fig(ashipx, ashipy, afig_def, LASHEIGHT);
sptr = &shot[NUMSHOTS];
 /* alien ship uses last shot in shot list */
if (!sptr->life_count) {
 /* if shot not already shot */
    setasound(SHOT_SOUND); /* then begin */
    rand = shot_angle(intshipx-ashipx, intshipy -
     ashipy);
```

EXHIBIT 11-2 *(continued)*

```
        /* note that here rand is not random, */
        /* it is in the direction of the player's ship */
        sptr->addx = shotaddx[rand];
        sptr->addy = shotaddx[(rand - 4) & 15];
        sptr->x = (sptr->addx<<1) + ashipx + (LASHEIGHT/2);
        sptr->y = (sptr->addy<<1) + ashipy + (LASHEIGHT/2);
        sptr->life_count = SHOT_LIFE;
    }
    move_count++;
     /* used for changing which ship figure */
    if (move_count & 1) { /* to draw (for rotation) */
        if (shipxv == 1) {
            afig_def += LASHEIGHT;
            if (afig_def == small_alien + (LASHEIGHT*3))
                afig_def += -LASHEIGHT*3; /* -= */
        }
        else {
            if (afig_def == small_alien)
                afig_def += LASHEIGHT*3;
            afig_def += -LASHEIGHT;  /* -= */
        }
    }
}
/*  Start a sound if the current sound playing (or the next
    selected sound) has lower or equal priority.  */
```

setasound(sound)

```
int sound;
{
    if (!running || sound >= wsound) {
        wsound = sound;
        startsound = TRUE;
    }
}
/*  Start a sound if the current sound playing (or the next
    selected sound) has lower priority.  */
```

setbsound(sound)

```
int sound;
{
```

EXHIBIT 11-2 *(continued)*

```
    if (!running || sound > wsound) {
        wsound = sound;
        startsound = TRUE;
    }
}
```

draw_score() /* draw score and number of ships remaining */

```
{
    register int *digptr;
    digptr = &score[SCORE_DIGITS];
    asm {
                move.l  back_screen(A4), A0
                adda    #4*64, A0               ; 4 lines down
                move    #SCORE_DIGITS-1, D1     ; D1 is counter
        loop:
                addq    #1, A0
                move    -(digptr), D0     ; get next digit
                lea     digits(A4), A1    ;
                 get next address of digits array
                adda    D0, A1            ;
                 add D0*9 to A1 to get address
                asl     #3, D0            ; of digit definition
                adda    D0, A1
                jsr     drawd
                dbf     D1, loop
                adda    #14*64-3, A0      ;
                 move down 14 lines and left
                lea     blank (A4), A1    ; a few bytes
                move.b num_ships(A4), D0 ;
                 draw number of ships with
                ext.w   D0                ;
                 leading blank if 1st digit
                beq     is0               ; is a zero
                lea     digits(A4), A1
                adda    D0, A1            ; A1 += D0*9
                asl     #3, D0            ;
                 point A1 in digit definition
                adda    D0, A1
        is0:
                jsr     drawd
```

EXHIBIT 11-2 *(continued)*

```
                addq    #1, A0              ;
                 move right for next digit
                move.b num_ships+1(A4), D0 ;
                 do the same as last time
                ext.w   D0
                lea     digits(A4), A1
                adda    D0, A1
                asl     #3, D0
                adda    D0, A1
                jsr     drawd
                suba    #63, A0             ;
                 move left 1 byte and down 1
                lea     small_ship(A4), A1  ;
                 draw the ship nest to the
                move.b (A1)+, (A0)          ;
                 number of ships left
                adda    #64, A0
                jsr     drawd
                bra     done
        drawd:
                move.b (A1)+, (A0)          ;
                 draw digit defined at A1
                move.b (A1)+, 64(A0)        ;
                 to address contained in A0
                move.b (A1)+, 64*2(A0)
                move.b (A1)+, 64*3(A0)
                move.b (A1)+, 64*4(A0)
                move.b (A1)+, 64*5(A0)
                move.b (A1)+, 64*6(A0)
                move.b (A1)+, 64*7(A0)
                move.b (A1)+, 64*8(A0)
                rts
            done:               ; drop through normal return
        }
}
/*  val contains the value to be added;
    pos tells which digit  */

add_to_score(val, pos)

int val, pos
{
```

EXHIBIT 11-2 *(continued)*

```
    int save;
    save = score[4];
    score[pos] += val;
    while (score[pos] >= 10) {
        score[pos] -= 10;
        if (pos < SCORE_DIGITS-1)
            score[++pos]++;
    }
    if (save != score[4]) {
        num_ships++;
        setasound(NEWSHIP_SOUND);
        if (num_ships & 0xff) > 9)
            num_ships += 256 - 10;
    }
}
/*  The asteroid definitions are expanded so there is a
    different definition for each rotated position. This
    saves time when drawing the asteroids. Also, the ship
    is expanded but in a different way. The 16 rotated
    positions of the ship are made up of 3 ship different
    rotated positions. This is done for the ship with no
    flame, the ship with a small flame, and the ship with
    the big flame.  */
```

initgamestuff()

```
{
    expand_smalldef(small_meteor, small_mdefs, SMALLHEIGHT,
     ~0);
    expand_smalldef(small_shadowmask,
     &small_mdefs[SMALLHEIGHT*16],
           SMALLHEIGHT, 0);
    expand_def(medium_meteor, medium_mdefs, MEDHEIGHT, ~0);
    expand_def(medium_shadowmask,
     &medium_mdefs[MEDHEIGHT*24],
           MEDHEIGHT, 0);
    expand_bigdef(big_meteor, big_mdefs, BIGHEIGHT, ~0);
    expand_bigdef(big_shadowmask, &big_mdefs[BIGHEIGHT*32),
           BIGHEIGHT, 0);
    expand_ships();
}
```

EXHIBIT 11-2 *(continued)*

game()

```
{
     int i;
     long key;
     int k;
     int bump_count;
     static which_bump;
     int big[100];
     eventrecord the_event;
     hidecursor();
     /* (eject 0L, 1) */
     front_screen = (char *) SCREEN1;
     back_screen = (char *) SCREEN2;
     *chscrn |= 64;
      /* make sure primary screen is up */
     primary_screen = ~0;
     first = NULL;
     freeptr = NULL;
     for (i = 0; i < N; i++)  {     /* make up freeptr list */
          meteor[i].next = freeptr;
          freeptr = &meteor[i];
     }
     for (i = 0; i < SCORE_DIGITS; i++) {
          /* initialize score to 0 */
          score[i] = 0;
     }
     for (i = 0; i < NUMSHOTS+1; i++) {
          /* no shots initially */
          shot[i].life_count = 0;
     }
     shipx = CENTERX<<8;
      /*put ship in center of screen */
     shipy = CENTERY<<8;
     intshipx = CENTERX;
     intshipy = CENTERY;
     oldshipx = 0;
     oldshipy = 0;
     shipxv = 0;
     shipxy = 0;
     dead = FALSE;
```

EXHIBIT 11-2 *(continued)*

```
ashipx = 0;
ashipy = 0;
open_sound();
clear_screen();
hyper_count = 0;        /* set up all th counters */
aship_counter = 0;
k = 0;
num_ships = START_SHIPS;
done_count = DONE_TIME;
start_count = START_TIME;
a_count = ALIEN_TIME;
do {
    if (!first && a_count)
         /* if there are asteroids and no alien */
        start_count--;
    if (!start_count)    /* if time to start */
        k++;
        if (k <= 2)
            init_meteors(START_BIGS);
        else if (k <= 4)
            init_meteors(START_BIGS+1);
        else
            init_meteors(START_BIGS+2);
        start_count = START_TIME;
        a_count = ALIEN_TIME;
        bump_count = ALIEN_TIME + FAST_BUMP;
        which_bump = 0;
    }
    startsound = FALSE;
     /* will be set later if sound to be changed */
    erase_meteors();
    erase_fig(oldshipx, oldshipy, SHEIGHT);
    erase_fig(oldashipx, oldashipy, ASHEIGHT);
    erase_shots();
    erase_explosions();
    draw_explosions();
    draw_score();
    draw_meteors();
    if (a_count) {  /*if alien ship not on screen */
        if (!--a_count) {
             /* if time for alien to come out */
```

EXHIBIT 11-2 *(continued)*

```
            if (++aship_counter <= 2)
                 /* big ship first two times */
                little_ship = FALSE;
            else
                little_ship - random() & 1;
                 /* later 50% chance */
            ashipx = -31;
            ashipy = (random() & 225) + 40;
            aship_dead = FALSE;
            if (random() & 1) {
                 /* randomly start left or right */
                ashipx = -31;
                ashipxv = 1;
            }
            else {
                ashipx = 511;
                ashipxv = -1;
            }
        }
    }                   /* if there is a ship then draw it */
    else {
        if (little_ship)
            draw_little_ship();
        else
            draw_aship();
    }
    if (hyper_count) {
         /* in hyperspace; ship not showing */
        if (num_ships) {
            if (a_count || !safe)
                 /* don't give new ship while alien */
                hyper_count--;      /* is on the screen */
            if (!hyper_count) {
                if (no_room() && safe)
                     /* if new ship and there */
                    hyper_count = 1;
                     /* isn't room then wait */
                else {
                    shipx = (long) hx << 8;
                    shipy = (long) hy << 8;
```

EXHIBIT 11-2 ***(continued)***

```
                        intshipx = hx;
                        intshipy = hy;
                        shipxv = 0;
                        shipyv = 0;
                        dead = FALSE;
                    }
                }
            }
        }
        else
            draw_ship();
        draw_shots();
        bump_count -= 20;
         /* handle the background sound */
        if (bump_count <= 0) {
            which_bump++;
            if (which_bump & 1)
                setasound (LBUMP_SOUND);
            else
                setasound (HBUMP_SOUND);
            bump_count = a_count + FAST_BUMP;
        }
        pause();
         /*wait for vertical blanking interrupt */
        swap_screens();
        if (num_ships == 0)
             /*handle pause before returning to title */
            done_count--;
             /*screen after the game is over */
        if (startsound)  /* if needed start a new sound */
            init_sound();
        do_sound();
         /* continue current sound (if present) */
        while (*(long *) 378 & 0x20000)
         /* wait if pause is down */
            getnextevent(0, &the_event);
    } while (done_count);
    *chrscrn |= 64;     /* display first graphics page */
    close_sound();
    i = SCORE_DIGITS;
```

EXHIBIT 11-2 *(continued)*

```
    while ( i-- && score[i] == hiscore[i])
     /* see if score > hiscore */
        ;
    if (score[i] > hiscore[i]) {
         /* if so change high score */
        for (i = 0; i < SCORE_DIGITS; i++)
            hiscore[i] = score[i];
    }
    flushevents (-1, 0);
     /* get rid of any events queued up */
}
overlay "game"
#include "figdefs.h"
#define SMALLHEIGHT 11
#define MEDHEIGHT 22
#define BIGHEIGHT 41
extern char *back_screen;
long dot_fig[16] = {0x3fffffff, 0x9fffffff, 0xcfffffff,
                    0xe7ffffff, 0xf3ffffff, 0xf9ffffff,
                    0xfcffffff, 0xfe7fffff, 0xff3fffff,
                    0xff9fffff, 0xffcfffff, 0xffe7ffff,
                    0xfff3ffff, 0xfff9ffff, 0xfffcffff,
                    0xfffe7fff };
/*  This routine makes sure that any figure at x, y
    32 bits wide by height tall is erased.  */
```

erase_fig (x, y, height)

```
register int x, y;
{
    int count;
    if (y >= 342 || y <= -height || x >= 512 || x <= -32)
        return;
    count = (342 - 1) - y;
    if (count >= height)
        count = height - 1;
    if (y < 0) {
        count += y;
        y = 0;
    }
    if (x >= 512 - (32+16))
```

EXHIBIT 11-2 *(continued)*

```
        x = 512 - (32+16);
    else if (x < 0)
        x = 0;
    asm {
            asl         #6, y               ; y *= 8
            movea.l  back_screen(A4), A0;
             set A0 to first position on
            adda        y, A0               ; the screen
            move        x, D0
            lsr         #4, D0
            asl         #1, D0
            adda        D0, A0
            move    count(A6), y
            move.l  #~0, D1
            move    #~0, D2
            move    #64-4, D0
        lp:
            move.l  D1, (A0)+
            move.w  D2, (A0)
            adda.w  D0, A0
            dbf     y, lp
        }
}
/*  This routine does the same as erase_fig, but the height
    is fixed to the height of a small asteroid and the
    maximum width is 16.  */
```

smallerase_fig (x, y)

```
register int x, y;
{
    int count;
    count = 342 - y - 1;
    if (count > SMALLHEIGHT - 1)
        count = SMALLHEIGHT - 1;
    if (y < 0) {
        count += y;
        y = 0;
    }
    if (x >= 512 - 32)
        x = 512 - 32;
```

EXHIBIT 11-2 *(continued)*

```
    else if (x < 0)
        x = 0;
    asm {
            asl         #6, y               ; y *= 8
            movea.l  back_screen(A4), A0;
             set A0 to first position on
            adda        y, A0               ; the screen
            move        x, D0
            lsr         #4, D0
            asl         #1, D0
            adda        D0, A0
            move     count(A6), y
            move.l   #~0, D1
            move     #64-4, D0
        lp:
            move.l   D1, (A0)+
            adda.w   D0, A0
            dbf      y, lp
        }
}
/*  This routine does the same as erase_fig, but the height
    is fixed at BIGHEIGHT and the maximum width is 48.  */
```

bigerase_fig (x, y, height)

```
register int x, y;
{
    int count;
    count = 342 - y - 1;
    if (count > BIGHEIGHT - 1)
        count = BIGHEIGHT - 1;
    if (y < 0) {
        count += y;
        y = 0;
    }
    if (x >= 512 - 64)
        x = 512 - 64;
    else if (x < 0)
        x = 0;
    asm {
            asl         #6, y               ; y *= 64
```

EXHIBIT 11-2 *(continued)*

```
            movea.l  back_screen(A4), A0;
             set A0 to first position on
            adda        y, A0              ; the screen
            move        x, D0
            lsr         #4, D0
            asl         #1, D0
            adda        D0, A0
            move    count(A6), y
            move.l  #~0, D1
            move    #64-4, D0
        lp:
            move.l  D1, (A0)+
            move.l  D1, (A0)
            adda.w  D0, A0
            dbf     y, lp
        }
}
/*  This routine draws a medium-sized asteroid. It
    saves time by using a pre-shifted version of the
    asteroid. The array of pre-shifted versions begins at
    def. It handles clipping to the screen and masking the
    background so asteroids can go over each other without
    appearing transparent.  */

draw_fast(x, y, def)

register int x, y;
register char * def;
{
    int count;
    count = 342 - y - 1;
    if (count >= MEDHEIGHT)
        count = MEDHEIGHT - 1;
    if (y < 0) {
        def += (-y - y - y)<<1;
        count += y;
        y = 0;
    }
    asm {
            move    x, D0
            and     #15, D0
```

EXHIBIT 11-2 *(continued)*

```
            asl     #2, D0                      ;
             def += MEDHEIGHT*6
            adda    D0, def                     ; (132)
            asl     #5,D0
            adda    D0, def
            lea     MEDHEIGHT*24*4(def), A1 ;
             A1 gets mask address
            asl     #6, y                       ; y *= 64
            movea.l back_sreeen(A4), A0         ;
             Set A0 to first position on
            adda    y, A0
            move    count(A6), y
            cmp     #-16, x
             ;there is different drawing
            blt     clip1
             ;code for each clipping
            tst     x                           ;possibility
            blt     clip2
            move    x, D0
            lsr     #4, D0
            asl     #1, D0
            adda    D0, A0
            cmp     #512-16, x
            bge     clip3
            cmp     #512-32, x
            bge     clip4
            move    #64-4,  D3                   ;no need to clip
lp0;        move.l  (A0), D0
            or.l    (A1)+, D0
            and.l   (def)+, D0
            move.l  D0, (A0)+
            move    (A0), D0
            or      (A1)+, D0
            and     (def)+, D0
            move    D0, (A0)
            adda    D3, A0
            dbf     y, lp0
            bra     done
clip1:
            addq    #4, def
            addq    #4, A1
```

EXHIBIT 11-2 *(continued)*

```
            move    #64, D3
    |p1:    move    (A0), D0
            or      (A1), D0
            and     (def), D0
            move    D0, (A0)
            addq    #6, A1
            addq    #6, def
            adda.w  D3, A0
            dbf     y, |p1
            bra     done
clip2:
            addq    #2, def
            addq    #2, A1
            move    #64, D3
    |p2:    move.l   (A0), D0
            or.l    (A1), D0
            and.l   (def), D0
            move.l  D0, (A0)
            addq    #6, A1
            addq    #6, def
            adda.w  D3, A0
            dbf     y, |p2
            bra     done
clip3:
            move    #64, D3
    |p3:    move    (A0), D0
            or      (A1), D0
            and     (def), D0
            move    D0, (A0)
            addq    #6, A1
            addq    #6, def
            adda.w  D3, A0
            dbf     y, |p3
            bra     done
clip4:
            move    #64, D3
    |p4:    move.l  (A0), D0
            or.l    (A1), D0
            and.l   (def), D0
            move.l  D0, (A0)
            addq    #6, A1
```

EXHIBIT 11-2 ***(continued)***

```
            addq    #6, def
            adda.w  D3, A0
            dbf     y, |p4
    done:
    }
}
/*  This routine does the same thing as draw_fast, but with
    objects the size of a small asteroid  */
```

smalldraw_fast(x, y, def)

```
register int x, y;
register     char *def;
{
    int count;
    count = 342 - y - 1;
    if (count >=SMALLHEIGHT)
        count = SMALLHEIGHT - 1;
    if (y < 0)  {
    def += -y << 2;
    count += y;
    y = 0
    }
asm {
            move    x, D0
            and     #15, D0
            asl     #2, D0                  ;
             def += 44*(x &15)
            adda    D0, def
            asl     #1, D0
            adda    D0, def
            asl     #2, D0
            adda    D0, def
            lea     SMALLHEIGHT*16*4(def), A1
             ;A1 gets mask address
            asl     #6, y                   ; y *= 64
            movea.l back_screen(A4), A0     ;
             Set A0 to first position on
            adda    y, A0
            move    count(A6), y
            move    x, D0
```

EXHIBIT 11-2 ***(continued)***

```
            blt     clip1
            lsr     #4, D0
            asl     #1, D0
            adda    D0, A0
            cmp     #512-16, x
            bge     clip2
            move    #64, D3                 ;no need to clip
    lp0:    move.l  (A0), D0
            or.l    (A1)+, D0
            and.l   (def)+, D0
            move.l  D0, (A0)
            adda    D3, A0
            dbf     y, lp0
            bra     done
    clip1:
            addq    #2, def
            addq    #2, A1
    clip2:
            move    #64, D3
    lp1:    move     (A0), D0
            or      (A1), D0
            and     (def), D0
            move    D0, (A0)
            addq    #4, A1
            addq    #4, def
            adda.w  D3, A0
            dbf     y, lp1
            bra     done
    done:
    }
}
/*  This routine does the same thing as draw_fast() and
    smalldraw_fast(), but it handles objects the size of
    the big asteroid.  */

bigdraw_fast(x, y, def)

register int x, y;
register     char *def;
{
    int count;
```

EXHIBIT 11-2 *(continued)*

```
count = (342 - 1) - y;
if (count >= BIGHEIGHT )
    count = BIGHEIGHT - 1;
if (y < 0)  {
    def -= y<<3;
    count += y;
    y = 0
}
asm {
        move    x, D0
        and     #15, D0
        asl     #3, D0
         def += BIGHEIGHT*8
        adda    D0, def
        asl     #3, D0
        adda    D0, def
        asl     #2, D0
        adda    D0, def
        lea     BIGHEIGHT*8*16(def), A1
         ;A1 gets mask address
        asl     #6, y                       ; y *= 64
        movea.l back_screen(A4), A0         ;
         Set A0 to first position on
        adda    y, A0
        move    count(A6), y
        cmp     #-32, x
        blt     clip1
        cmp     #-16, x
        blt     clip2
        tst     x
        blt     clip3
        move    x, D0
        lsr     #4, D0
        asl     #1, D0
        adda    D0, A0
        cmp     #512-16, x
        bge     clip4
        cmp     #512-32, x
        bge     clip5
        cmp     #512-48, x
        bge     clip6
        move    #64-4, D3                   ;no need to clip
```

EXHIBIT 11-2 ***(continued)***

```
|p0:     move.l    (A0), D0
       or.l      (A1)+, D0
       and.l     (def)+, D0
       move.l    D0, (A0)+
       move      (A0), D0
       or.l      (A1)+, D0
       and.l     (def)+, D0
       move.l    D0, (A0)
       adda.w    D3, A0
       dbf       y, |p0
       bra       done
clip1:
       addq      #6, def
       addq      #6, A1
       move      #64, D3
|p1:    move      (A0), D0
       or.w      (A1), D0
       and.w     (def), D0
       move      D0, (A0)
       addq      #8, def
       addq      #8, A1
       adda.w    D3, A0
       dbf       y, |p1
       bra       done
clip2:
       addq      #4, def
       addq      #4, A1
       move      #64, D3
|p2:    move.l    (A0), D0
       or.l      (A1), D0
       and.l     (def), D0
       move.l    D0, (A0)
       addq      #8, A1
       addq      #8, def
       adda.w    D3, A0
       dbf       y, |p2
       bra       done
clip3:
       addq      #2, def
       addq      #2, A1
       move      #64, D3
|p3:    move.l    (A0), D0
```

EXHIBIT 11-2 *(continued)*

```
            or.l    (A1)+, D0
            and.l   (def)+, D0
            move.l  D0, (A0)+
            move    (A0), D0
            or      (A1), D0
            and     (def), D0
            move    D0, (A0)
            addq    #4, A1
            addq    #4, def
            adda.w  D3, A0
            dbf     y, |p3
            bra     done
clip4:
            move    #64, D3
|p4:       move    (A0), D0
            or      (A1), D0
            and     (def), D0
            move    D0, (A0)
            addq    #8, A1
            addq    #8, def
            adda.w  D3, A0
            dbf     y, |p4
            bra     done
clip5:
            move    #64, D3
|p5:       move.l  (A0), D0
            or.l    (A1), D0
            and.l   (def), D0
            move.l  D0, (A0)
            addq    #8, A1
            addq    #8, def
            adda.w  D3, A0
            dbf     y, |p5
            bra     done
clip6:
            move    #64-4, D3
|p6:       move.l   (A0), D0
            or.l    (A1)+, D0
            and.l   (def)+, D0
            move.l  D0, (A0)+
            move    (A0),D0
```

EXHIBIT 11-2 ***(continued)***

```
            or      (A1),D0
            and     (def), D0
            move    D0, (A0)
            addq    #4, A1
            addq    #4, def
            adda.w  D3, A0
            dbf     y, |p6
    done:
    }
}
/*  This routine is like draw_fast(), but it takes the
    definition of a figure, not an array of rotated
    positions. It is a little slower because it must
    rotate the data before storing it to the screen, but
    less memory is needed for the figure definitions.
    This routine does not mask the images; it only ANDs
    it to the screen.  */
```

draw_fig(x, y, def, height)

```
register int x, y;
register long *def;
int height;
{
    int count;
    count + 342 - y - 1;
     /* This is for clipping in the y direction */
    if (count >= height)
        count = height - 1;
    if (y < 0) {
        def += -y;
        count += y;
        y = 0;
    }
    asm {
            clr.1 D3        ;D3 and D4 are used like a mask to
            clr   D4
             ;clip to the screen in the x direction
            cmp   #-16, x   ;(left and right)
            bge   L1
```

EXHIBIT 11-2 *(continued)*

```
        not.l D3
        bra   cont
L1:
        tst    x
        bge    L2
        move.l #0xffff0000, D3
        bra    cont
L2:
        cmp    #512-16, x
        bit    L3
        not.w  D3
        not.w  D4
        bra    cont
L3:
        cmp    #512-32, x
        bit    cont
        not.w  D4
cont:                              ;D3 and D4 are now set.
        asl    #6, y               ; y *= 64
        movea.l  back_screen(A4), A0  ;
         Set A0 to first position on
        adda     y, A0             ;the screen
        move     x, D0
        bpl      noch
        add      #32, D0
         ;if x < 9 then set A0 and D0
        move     D0, x
         ;to make up for that
        subq     #4, A0
noch:
        lsr      #4, D0
        asl      #1, D0
        adda     D0, A0
        and      #15, x
        move     #16, D0
        sub      x, D0
        move     count(A6), y
         ;y is now used as a counter
lp:     move.l   (def)+, D1
         ;clip and put on screen
        move.w   D1, D2
```

EXHIBIT 11-2 ***(continued)***

```
            lsr.l    x, D1
            asl.w    D0, D2
            not.l    D1
            not.w    D2
            or.l     D3, D1
            or.w     D4, D2
            and.l    D1, (A0)+
            and.w    D2, (A0)
            adda.w   #64-4, A0
            dbf      y, lp
    }
}
/*  This routine draws (if draw is TRUE) a "dot" at the
    location requested and returns  0 if there is
    nothing already at that location. It does not clip so
    one must be careful.  */

int check_dot(x, y, draw)

register int x, y;
int draw;
{
    if (x > 510 || x < 0 || y < 0 || y > 340)
        return;
    asm {
            movea.l  back_screen(A4), A0 ;Get screen address
            asl      #6, y
             ;calculate screen coordinate using
            adda     y, A0                ;x and y
            move     x, D0
            lsr      #4, D0
            asl      #1, D0
            adda     D0, A0
            cmp      #512-32, x
            bge      word
            and      #15, x
            lea      dot_fig(A4), A1
             ;Get address of figure array in A1
            asl      #2, x
             ;calculate figure address
```

EXHIBIT 11-2 *(continued)*

```
            adda    x, A1
            move    (A1), D0
            or      (A0), D0            ;D0 is return result
            tst     draw(A6)
            beq     done
            move.l  (A1), D1
            and.l   D1, (A0)
            and.l   D1, 64(A0)
            bra     done
    word:
     ;This part of the routine is used
            and   #15, x
             ;if the point is closer than 32
            lea   dot_fig(A4), A1
             ;dots from the right of the screen
            asl   #2, x
            adda  x, A1
            move  (A1), D1
            move  (A0), D0
            or    D1, D0
            tst   draw(A6)
            beq   done
            or    D1, D0
            and   D1, (A0)
            and   D1, 64(A0)
    done:
    }
}
/*  This is the same as check_dot, but the dot is always
    drawn and it does not return a value.  */
```

draw_dot(x, y)

```
register int x, y;
{
    if (x > 510 || x < 0 || y < 0 || y > 340)
        return;
    asm {
            movea.l back_screen(A4), A0
            asl     #6, y
            adda    y, A0
```

EXHIBIT 11-2 *(continued)*

```
            move    x, D0
            lsr     #4, D0
            asl     #1, D0
            adda    D0, A0
            cmp     #512-32, x
            bge     word
            and     #15, x
            lea     dot_fig(A4), A1
            asl     #2, x
            adda    x, A1
            move.l  (A1), D0
            and.l   D1, (A0)
            and.l   D1, 64(A0)
            bra     done
    word:
            and   #15, x
            lea   dot_fig(A4), A1
            asl   #2, x
            adda  x, A1
            move  (A1), D1
            move  (A0), D0
            and   D1, (A0)
            and   D1, 64(A0)
    done:
    }
}
/* erase an area at least the size of a "dot" on the
   screen */

erase_dot(x, y)

register int x, y;
{
    if (x > 510 || x < 0 || y < 0 || y > 340)
        return;
    asm {
            movea.l back_screen(A4), A0
            asl     #6, y
            adda    y, A0
            move    x, D0
            lsr     #4, D0
```

EXHIBIT 11-2 *(continued)*

```
            asl     #1, D0
            adda    D0, A0
            move.l  #~0, D1
            cmp     #512-32, x
            bge     word
            move.l  D1, (A0)
            move.l  D1, 64(A0)
            bra     done
    word:
            move    D1, (A0)
            move    D1, 64(A0)
    done:
    }
}
/*  This routine shifts the figure defined at orig and
    makes an array of shifted figures at new. Shiftin is
    the bit value (1 or 0) that is shifted in from the
    left. On the Mac it should be 1 for the figure and
    0 for a mask. This routine works for original
    figure definitions that are <= 32 dots wide.  */
```

expand_def(orig, new, height, shiftin)

```
register char *origin, *new;
{
    int i, j;
    register long first;
    register int second
    for (j = 0; j < height; j++) {
        asm {
                move.l  (orig)+, first
                move.w  shiftin(A6), second
        }
        for (i = 0; i < 16; i++) {
             /* rotates can't be done in C */
            asm {
                    move.l  first, (new)
                    move.w  second, 4(new)
                    move    shiftin(A6), D0
                    roxr    #1, D0
                     ; set or clear the x bit
                    roxr.l  #1, first
```

EXHIBIT 11-2 ***(continued)***

```
                    roxr.w  #1, second
                }
                new += height *6;
            }
            new -= height*6*16 - 6;
        }
}
/*  This is the same as expand_def(). It works here for
    figure definitions that are <= 16 bits wide.  */
```

expand_smalldef(orig, new, height, shiftin)

```
register char *origin, *new;
{
    int i, j;
    register long first;
    for (j = 0; j < height; j++) {
        asm {
                move.l  (orig)+, first
        }
        for (i = 0; i < 16; i++) {
             /* rotates can't be done in C */
            asm {
                    move.l  first, (new)
                    move    shiftin(A6), D0
                    roxr    #1, D0
                     ; set or clear the x bit
                    roxr.l  #1, first
            }
            new += height *4;
        }
        new -= height*4*16 - 4;
    }
}
/*  This routine does the same thing as expand_def() and
    expand_smalldef() but it is for figures that are <= 48
    bits wide.  */
```

expand_bigdef(orig, new, height, shiftin)

```
register char *origin, *new;
{
```

EXHIBIT 11-2 *(continued)*

```
    int i, j;
    register long first;
    register int second;
    for (j = 0; j < height; j++) {
        asm {
                move.l   (orig)+, first
                move.w   (orig)+, second
                swap     second
                move.w   shiftin(A6), second
        }
        for (i = 0; i < 16; i++) {
             /* rotates can't be done in C */
            asm {
                    move.l   first, (new)
                    move.l   second, 4(new)
                    move     shiftin(A6), D0
                    roxr     #1, D0
                     ; set or clear the x bit
                    roxr.l   #1, first
                    roxr.l   #1, second
            }
            new += height *8;
        }
        new -= height*8*16 - 8;
    }
}
/*  This routine gets the value of a bit out of a list
    of long words. y tells which long word. x contains
    which bit.  */
```

get_bit(x, y, matrix)

```
char *matrix;
{
    int val;
    matrix += ((y<<2) + (x>>3));
    val = *matrix & (128>>(x&7));
    return(val !=0);
}
/*  This routine sets a bit in an array of long words.
    y tells which long word and x tells which bit.  */
```

EXHIBIT 11-2 *(continued)*

set_bit(x, y, val, matrix)

```
char *matrix;
{
    matrix += ((y<<2) + (x>>3));
    *matrix |= val << (7 - x&7);
}
/*  This routine makes 16 rotated positions out of 3
    definitions of the player's ship. It does it one
    bit at a time.
    It takes about 3 seconds.  */
```

expand_ships()

```
{
    register int i, j;
    int k;
    for (j = 0; j < 3; j++)          /* invert ships */
        for (i = 0; i <16*4; i++) {
            ships[j][i] = ~ ships[j][i]
            f1ships[j][i] = ~ f1ships[j][i];
            f2ships[j][i] = ~ f2ships[j][i];
        }
    for (k = 0; k <= 1; k++) {
      for (i = 0; i < 29; i++)
          for (j = 0; j < 29; j++) {
            set_bit(i, j, get_bit(28-j, 28-i, ships[k]),
             ships[4-k]);
            set_bit(i, j, get_bit(28-j, 28-i, f1ships[k]),
             f1ships[4-k]);
            set_bit(i, j, get_bit(28-j, 28-i, f2ships[k]),
             f2ships[4-k]);
            }
    }
    for (k = 0; k <= 3; k++) {
        for (i = 0; i < 29; i++)
            for (j = 0; j <29; j++) {
                set_bit(i, j, get_bit(i, 28-j, ships[k]),
                 ships[-k]);
                set_bit(i, j, get_bit(i, 28-j, f1ships[k]),
                 f1ships[8-k]);
```

EXHIBIT 11-2 *(continued)*

```
                set_bit(i, j, get_bit(i, 28-j, f2ships[k]),
                 f2ships[8-k]);
            }
    }
    for (k = 1; k <= 7; k++) {
        for (i = 0; i < 29; i++)
            for (j = 0; j < 29; j++) {
                set_bit(i, j, get_bit(28-i, j, ships[k]),
                 ships[16-k]);
                set_bit(i, j, get_bit(28-i, j, f1ships[k]),
                 f1ships[16-k]);
                set_bit(i, j, get_bit(28-i, j, f2ships[k]),
                 f2ships[16-k]);
        }
    }
}
/* wait for the vertical blanking period at tickcount
change */

pause()

{
    static int x;
    while (tickcount() == x);
        x = tickcount();
}
```

/* Figure definitions—figdefs.h */

```
/*  Definition of the first three rotated positions of
    the player's ship with no flame.  */
int ships[16][16*4] = {
    {   0xFFFF, 0xFFFF, 0xFFFD, 0xFFFF, 0xFFFD, 0xFFFF,
        0xFFF8, 0xFFFF, 0xFFF8, 0xFFFF, 0xFFF2, 0x7FFF,
        0xFFF2, 0x7FFF, 0xFFF2, 0x7FFF, 0xFFE2, 0x3FFF,
        0xFFE2, 0x3FFF, 0xFFC2, 0x1FFF, 0xFFC2, 0x1FFF,
        0xFFC2, 0x1FFF, 0xFF82, 0x0FFF, 0xFF82, 0x0FFF,
        0xFF02, 0x07FF, 0xFF02, 0x07FF, 0xFF02, 0x07FF,
        0xFE02, 0x03FF, 0xFE02, 0x03FF, 0xFF02, 0x07FF,
```

EXHIBIT 11-2 ***(continued)***

```
        0xFFC2, 0x1FFF, 0xFFF2, 0x7FFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF},
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xEFFF,
        0xFFFF, 0xCFFF, 0xFFFF, 0x8FFF, 0xFFFF, 0x2FFF,
        0xFFFE, 0x2FFF, 0xFFFC, 0x4FFF, 0xFFF8, 0x4FFF,
        0xFFF0, 0x8FFF, 0xFFE0, 0x8FFF, 0xFFC1, 0x0FFF,
        0xFF81, 0x0FFF, 0xFF01, 0x0FFF, 0xFE02, 0x0FFF,
        0xFC02, 0x0FFF, 0xFC04, 0x0FFF, 0xFC04, 0x0FFF,
        0xFE04, 0x0FFF, 0xFF08, 0x0FFF, 0xFF88, 0x0FFF,
        0xFFD0, 0x0FFF, 0xFFF0, 0x1FFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF},
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xF9FF, 0xFFFF, 0xE1FF, 0xFFFF, 0x0BFF,
        0xFFFC, 0x13FF, 0xFFE0, 0x27FF, 0xFF80, 0x47FF,
        0xFC00, 0x87FF, 0xF801, 0x0FFF, 0xF802, 0x0FFF,
        0xFC04, 0x1FFF, 0xFC08, 0x1FFF, 0xFC10, 0x1FFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFE20, 0x3FFF,
        0xFE40, 0x3FFF, 0xFF80, 0x7FFF, 0xFF80, 0x7FFF,
        0xFFE0, 0x7FFF, 0xFFFC, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF},
    } ;
/*  First three rotated positions of the ship with a small
    flame  */
int f1ships[16][16*4] = {
    {   0xFFFF, 0xFFFF, 0xFFFD, 0xFFFF, 0xFFFD, 0xFFFF,
        0xFFF8, 0xFFFF, 0xFFF8, 0xFFFF, 0xFFF2, 0x7FFF,
        0xFFF2, 0x7FFF, 0xFFF2, 0x7FFF, 0xFFE2, 0x3FFF,
        0xFFE2, 0x3FFF, 0xFFC2, 0x1FFF, 0xFFC2, 0x1FFF,
        0xFFC2, 0x1FFF, 0xFF82, 0x0FFF, 0xFF82, 0x0FFF,
        0xFF02, 0x07FF, 0xFF02, 0x07FF, 0xFF02, 0x07FF,
        0xFE02, 0x03FF, 0xFE02, 0x03FF, 0xFF02, 0x07FF,
        0xFFC2, 0x1FFF, 0xFFF2, 0x7FFF, 0xFFFF, 0xFFFF,
        0xFFEF, 0xBFFF, 0xFFF2, 0x7FFF, 0xFFF8, 0xFFFF,
        0xFFFD, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF},
```

EXHIBIT 11-2 *(continued)*

```
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xEFFF,
        0xFFFF, 0xCFFF, 0xFFFF, 0x8FFF, 0xFFFF, 0x2FFF,
        0xFFFE, 0x2FFF, 0xFFFC, 0x4FFF, 0xFFF8, 0x4FFF,
        0xFFF0, 0x8FFF, 0xFFE0, 0x8FFF, 0xFFC1, 0x0FFF,
        0xFF81, 0x0FFF, 0xFF01, 0x0FFF, 0xFE02, 0x0FFF,
        0xFC02, 0x0FFF, 0xFC04, 0x0FFF, 0xFC04, 0x0FFF,
        0xFE04, 0x0FFF, 0xFF08, 0x0FFF, 0xFF88, 0x0FFF,
        0xFED0, 0x0FFF, 0xFF70, 0x1FFF, 0xFF3F, 0xFFFF,
        0xFF81, 0xFFFF, 0xFF9F, 0xFFFF, 0xFFBF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xF9FF, 0xFFFF, 0xE1FF, 0xFFFF, 0x0BFF,
        0xFFFC, 0x13FF, 0xFFE0, 0x27FF, 0xFF80, 0x47FF,
        0xFC00, 0x87FF, 0xF801, 0x0FFF, 0xF802, 0x0FFF,
        0xFC04, 0x1FFF, 0xFC08, 0x1FFF, 0xFC10, 0x1FFF,
        0xFE20, 0x3FFF, 0xFFFF, 0xFFFF, 0xFE20, 0x3FFF,
        0xFA40, 0x3FFF, 0xFB80, 0x7FFF, 0xFB80, 0x7FFF,
        0xF9E0, 0x7FFF, 0xF83C, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF},
    } ;
/*  First three rotated positions of the ship with a large
    flame  */
int f2ships[16][16*4] = {
    {   0xFFFF, 0xFFFF, 0xFFFD, 0xFFFF, 0xFFFD, 0xFFFF,
        0xFFF8, 0xFFFF, 0xFFF8, 0xFFFF, 0xFFF2, 0x7FFF,
        0xFFF2, 0x7FFF, 0xFFF2, 0x7FFF, 0xFFE2, 0x3FFF,
        0xFFE2, 0x3FFF, 0xFFC2, 0x1FFF, 0xFFC2, 0x1FFF,
        0xFFC2, 0x1FFF, 0xFF82, 0x0FFF, 0xFF82, 0x0FFF,
        0xFF02, 0x07FF, 0xFF02, 0x07FF, 0xFF02, 0x07FF,
        0xFE02, 0x03FF, 0xFE02, 0x03FF, 0xFF02, 0x07FF,
        0xFFC2, 0x1FFF, 0xFFF2, 0x7FFF, 0xFFBF, 0xEFFF,
        0xFF9F, 0xCFFF, 0xFFCF, 0x9FFF, 0xFFC2, 0x1FFF,
        0xFFE0, 0x3FFF, 0xFFF0, 0x7FFF, 0xFFF8, 0xFFFF,
        0xFFFD, 0xFFFF, 0xFFFF, 0xFFFF},
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xEFFF,
        0xFFFF, 0xCFFF, 0xFFFF, 0x8FFF, 0xFFFF, 0x2FFF,
        0xFFFE, 0x2FFF, 0xFFFC, 0x4FFF, 0xFFF8, 0x4FFF,
        0xFFF0, 0x8FFF, 0xFFE0, 0x8FFF, 0xFFC1, 0x0FFF,
```

EXHIBIT 11-2 *(continued)*

```
        0xFF81, 0x0FFF, 0xFF01, 0x0FFF, 0xFE02, 0x0FFF,
        0xFC02, 0x0FFF, 0xFC04, 0x0FFF, 0xFC04, 0x0FFF,
        0xFE04, 0x0FFF, 0xFF08, 0x0FFF, 0xFD88, 0x0FFF,
        0xFCD0, 0x0FFF, 0xFCF0, 0x1FFF, 0xFCFF, 0xFFFF,
        0xFC7E, 0x7FFF, 0xFC60, 0xFFFF, 0xFE07, 0xFFFF,
        0xFE0F, 0xFFFF, 0xFE3F, 0xFFFF, 0xFEFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
    {   0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xF9FF, 0xFFFF, 0xE1FF, 0xFFFF, 0x0BFF,
        0xFFFC, 0x13FF, 0xFFE0, 0x27FF, 0xFF80, 0x47FF,
        0xFC00, 0x87FF, 0xF801, 0x0FFF, 0xF802, 0x0FFF,
        0xFC04, 0x1FFF, 0xFC08, 0x1FFF, 0xF410, 0x1FFF,
        0xF620, 0x3FFF, 0xE640, 0x3FFF, 0xE780, 0x7FFF,
        0xE780, 0x7FFF, 0xE7E0, 0x7FFF, 0xC3FC, 0xFFFF,
        0xC00F, 0xFFFF, 0xC03F, 0xFFFF, 0xC3FF, 0xFFFF,
        0xBFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
    } ;
int small_meteor [] = {
        0xF1FF, 0xFFFF, 0xE07F, 0xFFFF, 0xD03F, 0xFFFF,
        0xA01F, 0xFFFF, 0xF41F, 0xFFFF, 0xA81F, 0xFFFF,
        0xD01F, 0xFFFF, 0xAA3F, 0xFFFF, 0xDDBF, 0xFFFF,
        0xF57F, 0xFFFF, 0xFAFF, 0xFFFF
    };
int medium_meteor[] = {
        0xFC0F, 0xFFFF, 0xFA04, 0x1FFF, 0xF502, 0x0FFF,
        0xF404, 0x07FF, 0xFA00, 0x03FF, 0xD540, 0x03FF,
        0xBAA0, 0x03FF, 0xFE40, 0x03FF, 0xBE20, 0x27FF,
        0xDD00, 0x4FFF, 0xFE80, 0x27FF, 0xF740, 0x47FF,
        0xFEA0, 0x87FF, 0xFD48, 0x07FF, 0xEFA8, 0x0FFF,
        0xDD50, 0x1FFF, 0xEFA8, 0x3FFF, 0xFFF5, 0x7FFF,
        0xFAEA, 0xFFFF, 0xFF5F, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF
    } ;
int big_meteor[] = {
        0xffff, 0x83ff, 0xffff, 0xffff, 0x003f, 0xffff,
        0xfffc, 0x000f, 0xffff, 0xfff0, 0x0003, 0xffff,
        0xffc0, 0x0001, 0xffff, 0xff00, 0x0000, 0x0fff,
        0xfe80, 0x0000, 0x03ff, 0xfd40, 0x0000, 0x01ff,
        0xfa80, 0x0000, 0x00ff, 0xfd00, 0x0005, 0x00ff,
```

EXHIBIT 11-2 *(continued)*

```
        0xfa80, 0x0008, 0x007f, 0xf404, 0x0014, 0x007f,
        0xea04, 0x0008, 0x007f, 0xd508, 0x0010, 0x007f,
        0xbaa0, 0x0000, 0x007f, 0x5540, 0x0000, 0x013f,
        0xe880, 0x0000, 0x021f, 0x7440, 0x0000, 0x051f,
        0xba80, 0xa000, 0x029f, 0x7741, 0x0000, 0x055f,
        0xafaa, 0x0000, 0x00bf, 0x7544, 0x0010, 0x005f,
        0xeeaa, 0x08a8, 0x009f, 0x7575, 0x5140, 0x011f,
        0xeefa, 0xa0a0, 0x0a3f, 0x7d54, 0x5500, 0x147f,
        0xeaea, 0x2a80, 0x2a7f, 0x7d54, 0x0540, 0x1dff,
        0xfaea, 0x00a0, 0x3b7f, 0xfbda, 0x0055, 0x7f7f,
        0xffb5, 0x680a, 0xaaff, 0xffda, 0xb415, 0x75ff,
        0xffad, 0x6aaa, 0xabff, 0xf7d7, 0xd5d5, 0x57ff,
        0xfaee, 0xaab7, 0xafff, 0xfddd, 0x5e57, 0x7fff,
        0xfefe, 0xaebb, 0xffff, 0xffff, 0xd7ff, 0xffff,
        0xffab, 0xeeeb, 0xffff, 0xffd7, 0x7f57, 0xffff,
        0xfffe, 0xfeff, 0xffff, 0xffff, 0xffff, 0xffff,
        0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff,
        0xffff, 0xffff, 0xffff
    } ;
int small_shadowmask[] = {  /* small asteroid shadow mask */
        0x0e00, 0, 0x1f80, 0, 0x3fc0, 0, 0x7fe0, 0,
        0xffe0, 0, 0xffe0, 0, 0xffe0, 0, 0xffc0, 0,
        0xffc0, 0, 0x7f80, 0, 0x3f00, 0
    } ;
int medium_shadowmask[] = {
        0x03f0, 0x0000, 0x07ff, 0xe000, 0x0fff, 0xf000,
        0x0fff, 0xf800, 0x1fff, 0xfc00, 0x3fff, 0xfc00,
        0x7fff, 0xfc00, 0xffff, 0xfc00, 0xffff, 0xf800,
        0xffff, 0xf000, 0xffff, 0xfc00, 0xffff, 0xfc00,
        0x7fff, 0xfc00, 0x7fff, 0xfc00, 0x7fff, 0xf800,
        0x3fff, 0xf000, 0x3fff, 0xe000, 0x3fff, 0xc000,
        0x3fff, 0x8000, 0x1fff, 0x0000, 0x03e0, 0x0000,
        0x0000, 0x0000
    } ;
int big_shadowmask [] = {
        0x0000, 0x7c00, 0x0000, 0x0000, 0xffc0, 0x0000,
        0x0003, 0xfff0, 0x0000, 0x000f, 0xfffc, 0x0000,
        0x003f, 0xfffe, 0x0000, 0x00ff, 0xffff, 0xf000,
        0x01ff, 0xffff, 0xfc00, 0x03ff, 0xffff, 0xfe00,
        0x07ff, 0xffff, 0xff00, 0x07ff, 0xffff, 0xff00,
        0x0fff, 0xffff, 0xff80, 0x0fff, 0xffff, 0xff80,
```

EXHIBIT 11-2 ***(continued)***

```
        0x1fff, 0xffff, 0xff80, 0x3fff, 0xffff, 0xff80,
        0x7fff, 0xffff, 0xff80, 0xffff, 0xffff, 0xffc0,
        0xffff, 0xffff, 0xffe0, 0xffff, 0xffff, 0xffe0,
        0xffff, 0xffff, 0xffe0, 0xffff, 0xffff, 0xffe0,
        0xffff, 0xffff, 0xffe0, 0xffff, 0xffff, 0xffe0,
        0xffff, 0xffff, 0xffe0, 0xffff, 0xffff, 0xffe0,
        0xffff, 0xffff, 0xffc0, 0xffff, 0xffff, 0xff80,
        0xffff, 0xffff, 0xff80, 0xffff, 0xffff, 0xff80,
        0x7fff, 0xffff, 0xff80, 0x7fff, 0xffff, 0xff80,
        0x7fff, 0xffff, 0xff00, 0x3fff, 0xffff, 0xfe00,
        0x1fff, 0xffff, 0xfc00, 0x0fff, 0xffff, 0xf800,
        0x07ff, 0xffff, 0xf000, 0x03ff, 0xffff, 0xe000,
        0x01ff, 0xffff, 0x8000, 0x00ff, 0xffff, 0x0000,
        0x007f, 0xffff, 0x0000, 0x003f, 0xfffe, 0x0000,
        0x000f, 0xfff8, 0x0000
    } ;
short digits [] = {
        0xe7, 0xdb, 0xbd, 0xbd, 0xbd, 0xbd, 0xbd, 0xbd, 0xe7,
         /* 0 */
        0xef, 0xcf, 0xef, 0xef, 0xef, 0xef, 0xef, 0xef, 0xef,
         /* 1 */
        0xe3, 0xdd, 0xbd, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0x81,
        0x81, 0xfb, 0xf7, 0xe7, 0xfb, 0xfd, 0xfd, 0xbb, 0xc7,
        0xfd, 0xf9, 0xf5, 0xed, 0xdd, 0xbd, 0x80, 0xfd, 0xfd,
        0x81, 0xbf, 0xbf, 0x83, 0xfd, 0xfd, 0xfd, 0xbd, 0xc3,
        0xe3, 0xdf, 0xbf, 0x83, 0xbd, 0xbd, 0xbd, 0xbd, 0xc3,
        0x81, 0xfd, 0xfb, 0xfb, 0xf7, 0xf7, 0xef, 0xef, 0xef,
        0xc3, 0xbd, 0xbd, 0xc3, 0xbd, 0xbd, 0xbd, 0xbd, 0xc3,
        0xc3, 0xbd, 0xbd, 0xbd, 0xbd, 0xc1, 0xfd, 0xfb, 0xc7
         /* 9 */
    } ;
long alien[] = {
        0x00078000, 0x001fe000, 0x002ff000, 0x005ff800,
        0x002ff800, 0x0057f800, 0x00000000, 0x02d7ff00,
        0x05abff80, 0x0ad7fec0, 0x15efff60, 0x2bd7ffb0,
        0x55efff98, 0xabd7fffc, 0x00000000,
        0xcf3cf3cc, 0xcf3cf3cc, 0x4fccf3c8,
        0x00078000, 0x001fe000, 0x002ff000, 0x005ff800,
        0x002ff800, 0x0057f800, 0x00000000, 0x02d7ff00,
        0x05abff80, 0x0ad7fec0, 0x15efff60, 0x2bd7ffb0,
        0x55efff98, 0xabd7fffc, 0x00000000,
```

EXHIBIT 11-2 *(continued)*

```
        0xf3cf3cf0, 0xf3cf3cf0, 0x73cf3cf0,
        0x00078000, 0x001fe000, 0x002ff000, 0x005ff800,
        0x002ff800, 0x0057f800, 0x00000000, 0x02d7ff00,
        0x05abff80, 0x0ad7fec0, 0x15efff60, 0x2bd7ffb0,
        0x55efff98, 0xabd7fffc, 0x00000000,
        0x3cf3cf3c, 0x3cf3cf3c, 0x3cf3cf38
    } ;
long small_alien[] = {
        0x07800000, 0x0fc00000, 0x0fc00000, 0x00000000,
        0x3ff00000, 0x5b680000,   /* windows */
        0xfffc0000, 0x00000000, 0x7ff80000, 0x1fe00000,
        0x07800000, 0x0fc00000, 0x0fc00000, 0x00000000,
        0x3ff00000, 0x6db00000,   /* windows */
        0xfffc0000, 0x00000000, 0x7ff80000, 0x1fe00000,
        0x07800000, 0x0fc00000, 0x0fc00000, 0x00000000,
        0x3ff00000, 0x36d80000,   /* windows */
        0xfffc0000, 0x00000000, 0x7ff80000, 0x1fe00000
    } ;
```

/* Sound routines */

```
overlay "game"
/* Sound routines for Megaroids by Mitch Bunnell */
#define VBase (char *)0xefe1fe
#define SoundLow 0x7fd00
#define SndBufWLen 0x0172
extern long tickcount();
long old_tick;
int wsound;          /*which sound is being played */
/* sound info for explosion sound */
char pamp1[] = { 20,128,  20,112,  20,96,  20,80,  20,64,
                 20,48, 30,32, 30,24, 30,16, 30,12, 30,8,
                 10,0, 0,0  } ;
char pl1[] = { 36,36,36,36,36,36,36,36,36,36,36,36,36,
               36,36,36,36,36,36,36,36,36,36,36,36,
               36,36,36,36,36,36,36,36,36,36,36,36,36
               };
char sound_list{420};
/* fire sound */
char pamp2[] = { 85,64, 70,128, 58,64, 54,32, 48,32, 43,16,
                 39,8,  36,4, 30,2, 30,0, 0,0  };
```

EXHIBIT 11-2 *(continued)*

```
char pl2[] = { 8,9,10,11,12,13,14,15,16,17,18,19,20,21,
               22,23,24,25,26,27,28,29,30};
char sound2_list[495];
/* big saucer sound */
char pl3[] = { 35,34,33,32,31,30,31,32,33,34,35};
char pamp3[] = { 125,200, 0,0 };
char sound3_list[125];
/* little saucer sound */
char pl4[] = { 15,14,13,12,11,10,11,12,13,14,15};
char pamp4[] = { 250,200, 70,200, 0,0 };
char sound4_list[320];
/* new ship sound */
char pl5[] = { 10,10,10,10,10,10,10,10,10,10,
               148,148,148,148,148,148,148,148,148,148,
              10,10,10,10,10,10,10,10,10,10,
               148,148,148,148,148,148,148,148,148,148,
              10,10,10,10,10,10,10,10,10,10 };
char pamp5[] = {170,255, 190,255, 35,0, 170,255, 190,255,
               35,0, 170,255,200,255, 0,0 };
char sound5_list[1160];
/* low bomp sound */
char pamp6[] = { 6,128, 6,120, 6,112, 6,104, 6,96, 6,88,
                 6,80, 6,72, 6,64, 6,56, 6,48, 6,40, 6,32,
                 6,24, 6,16, 6,8, 0,0 };
char pl6[] = { 62,62,62,62,62,62,62,62,62,62,62,62,62,62,
              62,62 };
char sound6_list[230];
/* high bomp sound */
char pamp7[] = { 6,128, 6,120, 6,112, 6,104, 6,96, 6,88,
                 6,80, 6,72, 6,64, 6,56, 6,48, 6,40, 6,32,
                 6,24, 6,16, 6,8, 0,0 };
char pl7[] = { 60,60,60,60,60,60,60,60,60,60,60,60,60,60,
              60,60  };
char sound7_list[210];
/* sound list tables */
unsigned char *tblsound[] = { sound6_list, sound7_list,
 sound2_list, sound1_list, sound3_list, sound4_list,
 sound5_list };
unsigned char *tblpl[] = { pl6, pl7, pl2, pl1, pl3, pl4,
 pl5 };
int tblsize[] = { sizeof(pl6), sizeof(pl7), sizeof(pl2),
 sizeof(pl1), sizeof(pl3), sizeof(pl4), sizeof(pl5)};
```

EXHIBIT 11-2 *(continued)*

```
unsigned char *tblpamp[] = { pamp6, pamp7, pamp2, pamp1,
 pamp3, pamp4, pamp5 };
unsigned tblpolyl[] = { 1,1,8, 0x4000,1,1,1 };
int period, pcount, oldv, running;
unsigned char *save_psound, *save_per;
unsigned randat;

init_sound()    /* initialize sound */

{
    int i;
    i = wsound;                 /* rotate through sounds */
    save_per = tblpl[i];
    period = *save_per;
    pcount = period;
    save_psound = tblsound[i] + 1;
    oldv = *tblsound[i];
    running = tblesize[i];
}

rand(pcl)    /* random number routine */

unsigned pcl;
{
    int t1, t2;
    if (pcl > 1) {
        t1 = randat;
        t2 = ((randat >>= 1) >> 1);
        if (!((t1 ^ t2) & 1)) randat |= pcl;
        else randat &= ~pcl:
        return (randat & 0xff);
    }
    else return (0xff);
}

open_sound()    /* open sound routine */

{
    int i,j, temp;
    unsigned char *pamp, *psound_list;
```

EXHIBIT 11-2 *(continued)*

```
    wsound = 0;
    for (i=0; i < sizeof(tblpamp)/4 ; i++) {
        pamp = tblpamp[i];
        psound_list = tblsound[i];
        randat = 87367;
        temp = 0:
        do {
            j = *pamp++;
            do {
                if (rand(tblpoly1[i]) & 1)
                    *psound_list++ =
                        (((temp =  ~temp) & 0xff)*(int)
                         *pamp)/0x100;
                else *psound_list++ = 0;
            } while (--j);
            pamp++;
        } while (*pamp);
    }
    running = 0;
    do_sound();
     /* clear sound list (because running is 0) */
    *VBase &= 0x7f;      /* turn sound on */
}


do_sound()    /* do the sound (call 1/60 sec) */

{
    if (running)  {    /* fill the sound buffer */
        period = *save_per++;
        asm {
            MOVE.L  #SoundLow,A0
             ; point to sound buffer
            MOVE    #SndBufWLen-1, D0      ; get buffer length
            MOVE.L  save_psound(A4),A1
             ; get pointer in sound list
            MOVE    period(A4),D1          ; get period
            MOVE    pcount(A4),D2          ; get period count
            MOVE    oldv(A4),D3            ; get old volume
    LOOP:
            MOVE.B  D3(A0)              ; repeat old amplitude
```

EXHIBIT 11-2 *(continued)*

```
            ADDQ    #2,A0
             ; point to next byte in sound buffer
            SUBQ    #1, D2
             ; decrement buffer counter
            DBEQ    D0,LOOP
             ; do for full buffer or end period
            BNE     DONE
             ; if not end of period then done
              MOVE  D1,D2
               ; count else (period count=0)
                                    ; get next period
              MOVE.B  (A1)+,D3
               ; get save new amplitude as oldv
              DBF     D0,LOOP
               ; do till end of buffer
    DONE:
              MOVE.L A1,save_psound(A4)
               ; save spot in sound list
              MOVE    D2,pcount(A4)          ; save period count
              MOVE    D3,oldv(A4)            ; save old volume
              }
             --running;
    }             /* else init_sound(); */
    else {
        asm {
            move.l     #SoundLow,A0
             ; point to sound buffer
            move       #370-1, D0
        |p:  clr.b     (A0)
            addq       #2, A0
            dbf        D0, |p
        }
    }
}

close_sound()

{
    #VBase |= ~0x7f;
}
```

EXHIBIT 11-2 *(continued)*

Resource file for megaroids

```
!megaroids
APPLMEGR
TYPE DLOG
     ,100
dummy
75 45 277 467
visible
1
0
100
TYPE DITL
     ,100
1
button
172 372 192 412
OK
TYPE MEGR = STR
     ,0
Test string
TYPE FREF
     ,128
APPL 0
TYPE BNDL
     ,128
MEGR 0
ICN# 0
0 128
FREF 0
0 128
TYPE ICN# = GNRL
     ,128
.H
0000 0000 0000 0000 0003 E000 0004 1C00
0018 0200 00E0 0100 0300 00C0 0400 0020
0400 0010 0800 0008 0A0A 0004 1530 0002
1A20 0002 3A00 0A22 2A30 0402 3DC0 0822
2EEA 0044 3F50 4004 2AA1 8004 3D51 3454
2FB0 C8A4 1755 7148 1FAA 86A8 17DD 1D50
0AEA ABF0 0FDD DD50 06FB 57A0 01F7 EEC0
```

EXHIBIT 11-2 *(continued)*

```
003D 7500 0007 FE00 0000 0000 0000 0000
0000 0000 0000 0000 0003 E000 0007 FC00
001F FE00 00FF FF00 03FF FFC0 07FF FFE0
07FF FFF0 0FFF FFF8 0FFF FFFC 1FFF FFFE
1EFF FFFE 3FFF FFFE 3FFF FFFE 3FFF FFFE
3FFF FFFC 3FFF FFFC 3FFF FFFC 3FFF FFFC
3FFF FFFC 1FFF FFF8 1FFF FFF8 1FFF FFF0
0FFF FFF0 0FFF FFF0 08FF FFE0 01FF FFC0
003F FF00 0008 FE00 0000 0000 0000 0000
```

EXHIBIT 11-2 ***(concluded)***