Long Way Down

Your main engines have malfunctioned, and you're falling towards your final resting place at the heart of a cavernous asteroid. You've still got thruster control, but from the look of things that's not going to be enough. Still, an adventurist space pilot like yourself has taken defeat so easily. It may just be prolonging the inevitable but you grab your thruster controls and brace yourself for your last ride. It's a Long Way Down, so enjoy!

Long Way Down is an excellent beginners program to learn allegro. It's relatively simple, but covers elements such as timing, keyboard input, fullscreen/windowed modes, text output and particles. The game will even load external wav files if you have them and add sound to the game. All you need to do is choose wav files called hum.wav that will grow louder as you get closer to the edge, jet.wav for the sound of your thrusters, and explode.wav for when you die and place them in the same directory as the executable.

Long Way Down is written by Paul Pridham aka Madgarden.

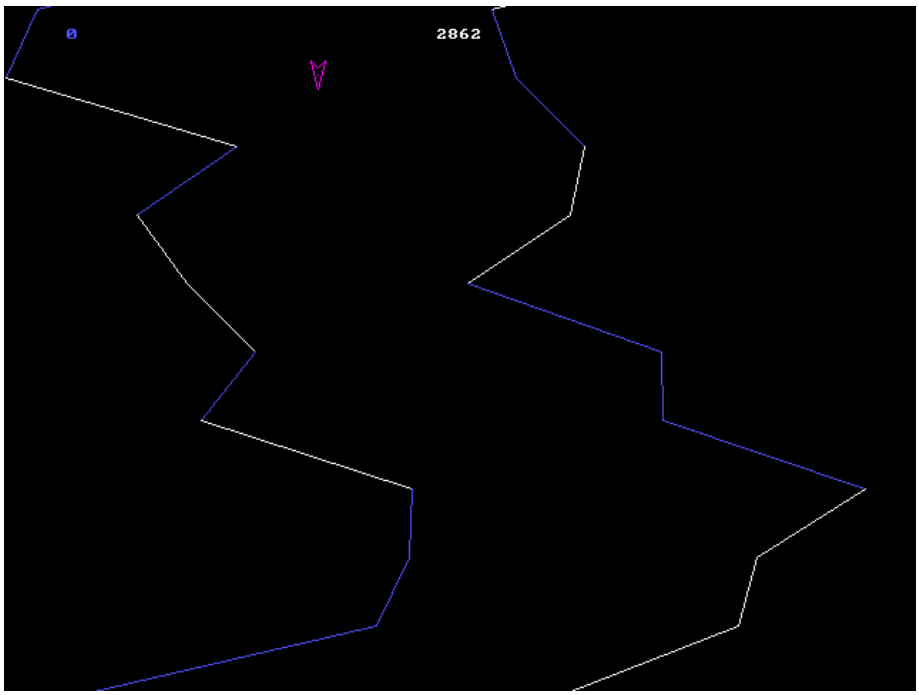
/* lwd.c listing begins: */

```
#include <math.h>
#include <time.h>
#include <allegro.h>

#define LOGIC_RATE    100

#ifdef _DEBUG
#define WINDOWED_MODE  GFX_GDI
#else
#define WINDOWED_MODE  GFX_AUTODETECT_WINDOWED
#endif

#define NUM_SECTIONS    12
#define START_SPEED      0.5
#define SHIP_ACCEL_X    0.17
#define SHIP_ACCEL_Y    0.0500000
#define MICRO_THRUST_SCALE  4
#define LEVEL_ACCEL      0.0005
#define HUM_PROXIMITY    5000

#define NUM_PARTICLES    750
#define SCREEN_MARGIN    48
#ifndef PI
#define PI 3.14159265
#endif

#define frand(n) (((float)(rand()%1000)/1000)*n)

#define LEFT  0x01
#define RIGHT  0x02
#define UP    0x04
#define DOWN  0x08

typedef struct
{
  float x;
  float y;
}Vec2d;

typedef struct
{
  Vec2d pos;
  Vec2d v;
  int colour;
}Particle;

enum{ATTRACT=1, DEAD=2};

volatile int tick;

void ticker(void)
{
  tick++;
}
END_OF_FUNCTION(ticker)

int main()
{
  BITMAP *backBuffer;
  int score, highscore, quit=FALSE, isReady=FALSE;
  int i, j, index, gameOver, shipIndex, bottomIndex;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
  int black, white, mauve, green, blue, orange;
  int leftWall[NUM_SECTIONS], rightWall[NUM_SECTIONS];
  int width, toggle=0;
  float cross, center, proximity;
  float levelSpeed, swing, yOff;
  Particle ship, particle[NUM_PARTICLES];
  Vec2d vShip, vWall;
  SAMPLE *hum, *jet, *explode;

  hum = jet = explode = NULL;
  allegro_init();
  install_keyboard();
  install_timer();
  install_sound(DIGI_AUTODETECT, MIDI_NONE, NULL);

  hum=load_sample("hum.wav");
  jet=load_sample("jet.wav");
  explode=load_sample("explode.wav");

  if (hum) {
    play_sample(hum, 0, 128, 1000, TRUE);
    adjust_sample(hum, 0, 128, 1000, TRUE);
  }
  if (jet) {
    play_sample(jet, 0, 128, 2000, TRUE);
    adjust_sample(jet, 0, 128, 2000, TRUE);
  }

  tick=0;
  LOCK_FUNCTION(ticker);
  LOCK_VARIABLE(tick);
  install_int_ex(ticker, BPS_TO_TIMER(LOGIC_RATE));

  set_color_depth(8);

  set_gfx_mode(WINDOWED_MODE, 640, 480, 0, 0);

  black=makecol(0, 0, 0);
  white=makecol(255, 255, 255);
  mauve=makecol(255, 0, 255);
  green=makecol(0, 255, 0);
  blue=makecol(64, 64, 255);
  orange=makecol(128, 64, 0);

  srand(time(NULL));

  backBuffer=create_bitmap(640, 480);

  clear_to_color(backBuffer, black);

  highscore=0;

  gameOver=ATTRACT;

  while(!quit)
  {
    int counter=0;
    int restart=FALSE;

    // Initialize walls
    for(i=0; i<NUM_SECTIONS; i++)
    {
      leftWall[i]=i*10;
```



/* Listing continued on next page…*/

/* Listing continued from previous page */

```
      rightWall[i]=SCREEN_W-i*10;
    }

    // Initialialize ship
    ship.pos.x=SCREEN_W/2;
    ship.pos.y=SCREEN_H/2;
    ship.v.x=0;
    ship.v.y=0;
    ship.colour=mauve;

    levelSpeed=START_SPEED;
    center=SCREEN_W/2;
    width=300;

    yOff=0;
    index=0;

    score=0;
    swing=6;

    while(!quit)
    {
      while(tick)
      {
        tick--;
        counter++;

        if(!gameOver&&!restart)
        {
          int adjust=FALSE;

          // TODO: Add to ship velocity
          if(key[KEY_LEFT])
          {
            if(key_shifts&KB_SHIFT_FLAG)
            {
              ship.v.x-=(float)SHIP_ACCEL_X/(float)MICRO_THRUST_SCALE;
            }
            else
            {
              ship.v.x-=SHIP_ACCEL_X;
            }

            adjust=TRUE;
          }
          if(key[KEY_RIGHT])
          {
            if(key_shifts&KB_SHIFT_FLAG)
            {
              ship.v.x+=(float)SHIP_ACCEL_X/(float)MICRO_THRUST_SCALE;
            }
            else
            {
              ship.v.x+=SHIP_ACCEL_X;
            }

            adjust=TRUE;
          }
          if(key[KEY_UP])
          {
            if(key_shifts&KB_SHIFT_FLAG)
            {
              ship.v.y-=(float)SHIP_ACCEL_Y/(float)MICRO_THRUST_SCALE;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
    }
    else
    {
      ship.v.y-=SHIP_ACCEL_Y;
    }

    adjust=TRUE;
  }
  if(key[KEY_DOWN])
  {
    if(key_shifts&KB_SHIFT_FLAG)
    {
      ship.v.y+=(float)SHIP_ACCEL_Y/(float)MICRO_THRUST_SCALE;
    }
    else
    {
      ship.v.y+=SHIP_ACCEL_Y;
    }

    adjust=TRUE;
  }

  if (jet) {
    if(adjust)
    {
      adjust_sample(jet, 64, 128, 2000, TRUE);
    }
    else
    {
      adjust_sample(jet, 0, 128, 2000, TRUE);
    }
  }

  ship.pos.x+=ship.v.x;
  ship.pos.y+=ship.v.y;

  if(ship.pos.y<SCREEN_MARGIN)
  {
    ship.pos.y=SCREEN_MARGIN;
    ship.v.y=0;
  }
  else if(ship.pos.y>SCREEN_H-SCREEN_MARGIN)
  {
    ship.pos.y=SCREEN_H-SCREEN_MARGIN;
    ship.v.y=0;
  }

  yOff+=levelSpeed;
  levelSpeed+=LEVEL_ACCEL;

  if(!(rand()%30))
    toggle=!toggle;

  if(toggle&&center<SCREEN_W-width*0.75)
  {
    center+=swing;
  }
  else if(!toggle&&center>width*0.75)
  {
    center-=swing;
  }

  if(yOff>48)
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
        {
          yOff=48-yOff+1;

          {
            leftWall[index]=center-(width>>2)-rand()%(width>>1);
            rightWall[index]=center+(width>>2)+rand()%(width>>1);
          }

          index++;

          if(index>=NUM_SECTIONS)
          {
            index=0;

            if(score<10000)
            {
              if(width>50)
                width-=5;

              if(swing>0)
                swing-=0.15;
              else
                swing=0;
            }
            else if(score<12000)
            {
              swing=0.05;
            }
            else if(score<13000)
            {
              swing=2;
              width=320;
            }
            else if(score<14000)
            {
              swing=0.5;
              width=100;
            }
            else if(score<14500)
            {
              swing=0.25;
              width=75;
            }
            else if(score<15000)
            {
              swing=4;
              width=300;
            }
            else // End of the road
            {
              swing=0.1;
              width=10;
            }

          }
        }


        // Get the index of the ship into the wall array
        shipIndex=(int)((ship.pos.y+yOff)/48)+index;
        if(shipIndex>=NUM_SECTIONS)
          shipIndex-=NUM_SECTIONS;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
            // Check for a collision with either wall

            bottomIndex=shipIndex+1;
            if(bottomIndex>=NUM_SECTIONS)
              bottomIndex=0;

            // Left
            vWall.x=(leftWall[bottomIndex]-leftWall[shipIndex]);
            vWall.y=48;

            vShip.x=(ship.pos.x-leftWall[shipIndex]);
            vShip.y=((int)(ship.pos.y+yOff))%48;

            cross=(vShip.x*vWall.y)-(vShip.y*vWall.x);

            gameOver=cross<0?DEAD:FALSE;

            proximity=0;
            if(cross<HUM_PROXIMITY)
            {
              proximity=255-255*cross/HUM_PROXIMITY;
            }

            if(!gameOver)
            {
              // Right
              vWall.x=(rightWall[bottomIndex]-rightWall[shipIndex]);
              vShip.x=(ship.pos.x-rightWall[shipIndex]);

              cross=(vShip.x*vWall.y)-(vShip.y*vWall.x);

              gameOver=cross>0?DEAD:FALSE;
            }

            if(cross>-HUM_PROXIMITY)
            {
              int proxtemp;

              proxtemp=255-255*(-cross/HUM_PROXIMITY);

              if(proxtemp>proximity)
                proximity=proxtemp;
            }

            if (hum && proximity)
              adjust_sample(hum, proximity, 128, 1000+(int)(levelSpeed*40), TRUE);

            score++;

            if(gameOver==DEAD)
            {
              for(i=0; i<NUM_PARTICLES; i++)
              {
                float xs, ys, speed;
                float angle;

                angle=rand()%360;

                particle[i].colour=green;
                speed=frand(5);

                xs = sin((angle / 180) * PI) * speed;
                ys = -cos((angle / 180) * PI) * speed;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
            particle[i].pos.x=ship.pos.x;
            particle[i].pos.y=ship.pos.y;
            particle[i].v.x=xs;
            particle[i].v.y=ys;
          }

          if (explode) play_sample(explode, 255, 128, 1000, FALSE);
          if (jet) adjust_sample(jet, 0, 128, 2000, TRUE);
          if (hum) adjust_sample(hum, 0, 128, 2000, TRUE);
        }
      }
      // Update particles
      else
      {
        if(gameOver==DEAD)
        {
          for(i=0; i<NUM_PARTICLES; i++)
          {
            particle[i].pos.x+=particle[i].v.x;
            particle[i].pos.y+=particle[i].v.y;
            particle[i].v.y+=LEVEL_ACCEL;
            particle[i].v.x*=(1-LEVEL_ACCEL);
          }
        }

        if(key[KEY_ENTER])
        {
          gameOver=FALSE;
          restart=TRUE;
          break;
        }

        if(key[KEY_F])
        {
          set_gfx_mode(GFX_AUTODETECT_FULLSCREEN, 640, 480, 0, 0);
        }

        if(key[KEY_W])
        {
          set_gfx_mode(WINDOWED_MODE, 640, 480, 0, 0);
        }
      }

      if(key[KEY_ESC])
      {
        quit=TRUE;
        break;
      }

      isReady=TRUE;
    }


    if(isReady)
    {
      isReady=FALSE;

      // Draw stuff
      j=index;

      // Draw walls
      for(i=0; i<NUM_SECTIONS; i++)
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
        {
          int i1, i2, colour=white;

          i1=j;

          j++;
          if(j>=NUM_SECTIONS)
            j=0;

          i2=j;

          if(leftWall[i1]>leftWall[i2])
            colour=blue;
          else
            colour=white;

          line(backBuffer, leftWall[i1], i*48-yOff, leftWall[i2],
            (i+1)*48-yOff, colour);

          if(rightWall[i1]<rightWall[i2])
            colour=blue;
          else
            colour=white;

          line(backBuffer, rightWall[i1], i*48-yOff, rightWall[i2],
            (i+1)*48-yOff, colour);
        }

        if(restart)
          break;

        // Draw ship
        if(!gameOver)
        {
          line(backBuffer, ship.pos.x, ship.pos.y+10, ship.pos.x-5,
            ship.pos.y-10, ship.colour);
          line(backBuffer, ship.pos.x, ship.pos.y+10, ship.pos.x+5,
            ship.pos.y-10, ship.colour);
          line(backBuffer, ship.pos.x-5, ship.pos.y-10, ship.pos.x,
            ship.pos.y-5, ship.colour);
          line(backBuffer, ship.pos.x, ship.pos.y-5, ship.pos.x+5,
            ship.pos.y-10, ship.colour);
          putpixel(backBuffer, ship.pos.x, ship.pos.y, ship.colour);

          if(counter%10<5)
          {
            if(key[KEY_LEFT])
              putpixel(backBuffer, ship.pos.x+5, ship.pos.y, green);
            if(key[KEY_RIGHT])
              putpixel(backBuffer, ship.pos.x-5, ship.pos.y, green);
            if(key[KEY_UP])
              putpixel(backBuffer, ship.pos.x, ship.pos.y+12, green);
            if(key[KEY_DOWN])
              putpixel(backBuffer, ship.pos.x, ship.pos.y-9, green);
          }
        }
        else
        {
          for(i=0; i<NUM_PARTICLES; i++)
          {
            putpixel(backBuffer, particle[i].pos.x, particle[i].pos.y,
              particle[i].colour);
          }
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```
          if(score>highscore)
          {
            int colour;

            colour=counter%20<10?mauve:green;
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-SCREEN_MARGIN*3, colour, "NEW HIGH SCORE!");
          }

          if(counter%1000<500)
          {
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-80, blue, "ARROW keys to move");
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-64, blue, "Hold SHIFT for micro thrust");
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-48, blue, "F for fullscreen, W for windowed");
          }
          else
          {
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-64, blue, "Long Way Down... by Paul Pridham");
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-48, blue, "Built with Allegro %s",
              ALLEGRO_VERSION_STR);
          }

          if(counter%40<20)
          {
            textprintf_centre(backBuffer, font, SCREEN_W/2,
              SCREEN_H-16, white, "Press ENTER to play");
          }
        }

        textprintf_centre(backBuffer, font, SCREEN_W/2, 16, white, "%d", score);
        textprintf_centre(backBuffer, font, 48, 16, blue, "%d", highscore);

        blit(backBuffer, screen, 0, 0, 0, 0, SCREEN_W, SCREEN_H);
        clear_to_color(backBuffer, black);
      }

      yield_timeslice();
    }

    if(score>highscore)
    {
      highscore=score;
    }
  }

  destroy_bitmap(backBuffer);
  if (hum) destroy_sample(hum);
  if (jet) destroy_sample(jet);
  if (explode) destroy_sample(explode);

  return 0;
}
END_OF_MAIN();
```