

CHAPTER 13

VERTICAL SCROLLING ARCADE GAMES

Most arcade games created and distributed to video arcades in the 1980s and 1990s were scrolling shoot-em-up games (also called simply *shooters*). About an equal number of vertical and horizontal shooters were released. This chapter focuses on vertical shooters (such as *Mars Matrix*) and the next chapter deals with the horizontal variety (although it focuses on platform “jumping” games, not shooters). Why focus two whole chapters on the subject of scrolling games? Because this subject is too often ignored. Most aspiring game programmers know what a shooter is but have no real idea how to develop one. That’s where this chapter comes in! This chapter discusses the features and difficulties associated with vertical shooters and explains how to develop a vertical scroller engine, which is used to create a sample game called *Warbirds Pacifica*, a 1942-style arcade game with huge levels and professionally-drawn artwork.

Here is a breakdown of the major topics in this chapter:

- Building a vertical scroller engine
- Writing a vertical scrolling shooter

Building a Vertical Scroller Engine

Scrolling shooters are interesting programming problems for anyone who has never created one before (and who has benefited from an experienced mentor). In the past, you have created a large memory bitmap and blitted the tiles into their appropriate places on that bitmap, which could then be used as a large game world (for instance, in an earlier revision of *Tank War*). A scrolling shooter, on the other hand, has a game world that is far too large for a single bitmap. For that matter, most games have a world that is too large for a single bitmap, and using such a bitmap goes against good design practices. The world is comprised of tiles, after all, so it would make sense to draw only the tiles needed by the current view.

456 Chapter 13 ■ Vertical Scrolling Arcade Games

But for the sake of argument, how big of a world bitmap would you have to use? Mappy (the map editor tool covered in the previous chapter) supports a map of around 30,000 tiles. If you are using a standard 640-pixels-wide screen for a game, that is 20 tiles across, assuming each tile is 32×32. Thirty-thousand tiles divided by 20 tiles across gives you...how many? Fifteen-hundred tiles spread vertically. At 32 pixels each, that is a bitmap image of 640×48,000. That is ridiculously large—so large that I do not need to argue the point any further. Of course, the game world can be much smaller than this, but a good scrolling shooter will have nice, large levels to conquer.

What you need is a vertical scrolling game engine capable of blitting only those tiles needed by the current display. I once wrote a game called *Warbirds* for another book titled *Visual Basic Game Programming with DirectX* (Premier Press, 2002). The game featured a randomly generated vertical scrolling level with warping. This meant that when the scrolling reached the end of the level, it wrapped around to the start of the level and continued scrolling the level without interruption (see Figure 13.1).

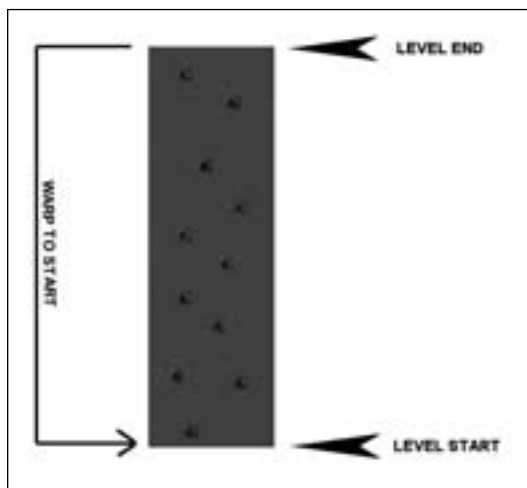


Figure 13.1 Level warping occurs when the end of the level is reached in a scrolling game.

Given that the levels were generated randomly, the game could go on forever without the need for new levels. Unfortunately, as you might have guessed, the levels were quite boring and repetitive. Even with a fairly good warping technique and random map generator, the levels were not very attractive. See Figure 13.2 for a screenshot of *Warbirds*.

If you don't want to use wraparound, or warping, then what happens when the scroller reaches the end? Of course, that's the end of the level. At this point, you want to display the score, congratulate the player, add bonus points, and then proceed to load the next level of the game.

The vertical scroller engine that you'll put together shortly will just sort of stop when it reaches the end of the level; this is a design decision, because I want you to take it from there (load the next level). Then, you can add the custom artwork for a new scrolling shooter, and I'll provide a template by having you build a sample game at the end of this chapter: *Warbirds Pacifica*.

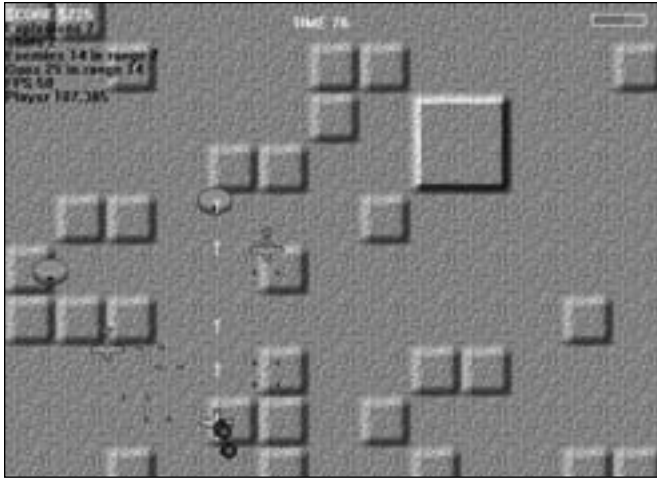


Figure 13.2 *Warbirds* featured a randomly-generated scrolling map.

Creating Levels Using Mappy

The *Warbirds Pacifica* game developed later in this chapter will use high-quality custom levels created with Mappy (which was covered in the previous chapter). Although I suggested using a data array for the maps in simple games, that is not suitable for a game like a scrolling shooter—this game needs variety! To maximize the potential for this game, I'm going to create a huge map file that is 20 tiles wide and 1,500 tiles high! That's equivalent to an image that is 640×48,000 pixels in size. This game *will* be fun; oh yes, it will be!

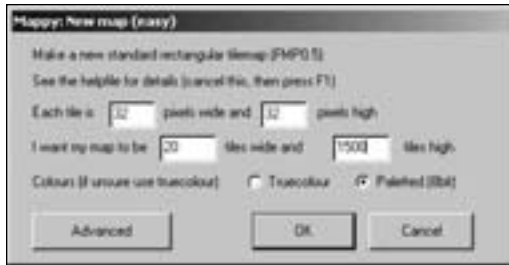
If you read the previous chapter, then you should have Mappy handy. If not, I recommend you go back and read Chapter 12 because familiarity with Mappy is crucial for getting the most out of this chapter and the one that follows.

Assuming you have Mappy fired up, open the File menu and select New Map. First, be sure to select the Paletted (8bit) option. You want to use simple 8-bit tiles when possible to lighten the memory load with MappyAL, although you may use hi-color or true color tiles if you want. (I wouldn't recommend it generally.) You might recall from the last chapter that MappyAL is a public domain source code library for reading and displaying a Mappy level, and that is what you'll use in this chapter to avoid having to create a tile engine from scratch. Next, for the width and height of each tile, enter 32 and 32, respectively. Next, for the map size, enter 20 for the width and 1500 for the height, as shown in Figure 13.3.

tip

Be sure to select Paletted (8bit) for the color depth of a new map in Mappy if you intend to use the MappyAL library in your Allegro games.

458 Chapter 13 ■ Vertical Scrolling Arcade Games



Mappy will create a new map based on your specifications, and then will wait for you to import some tiles (see Figure 13.4).

Figure 13.3 Creating a new map in Mappy for the vertical scroller demo

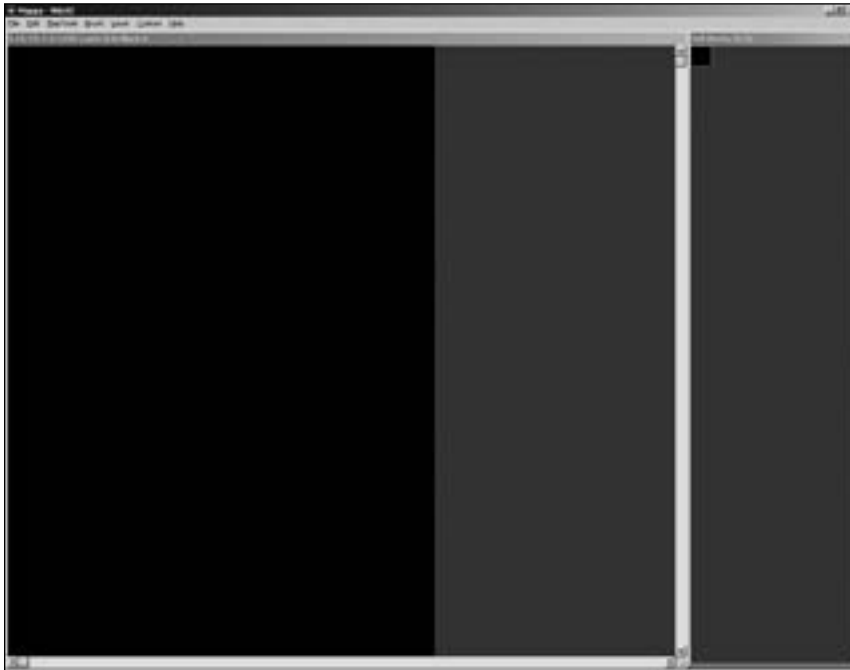


Figure 13.4 Mappy has created the new map and is now waiting for tiles.

Now open the File menu and select Import to bring up the File Open dialog box. This is the part where you have some options. You can use the large collection of tiles I have put together for this chapter or you can create your own tiles and use them. Your results will certainly look different, but if you have your own tiles, by all means use them. Otherwise, I recommend that you copy the `maptiles8.bmp` file from the CD-ROM to a folder on your hard drive. The tile image is located in `\chapter13\VerticalScroller` on the CD-ROM under the sources folder for the environment you are using (Visual C++, KDevelop, or Dev-C++). Select this file using the File Open dialog box, and the 32×32 tiles will be added to the tile palette in Mappy (see Figure 13.5).

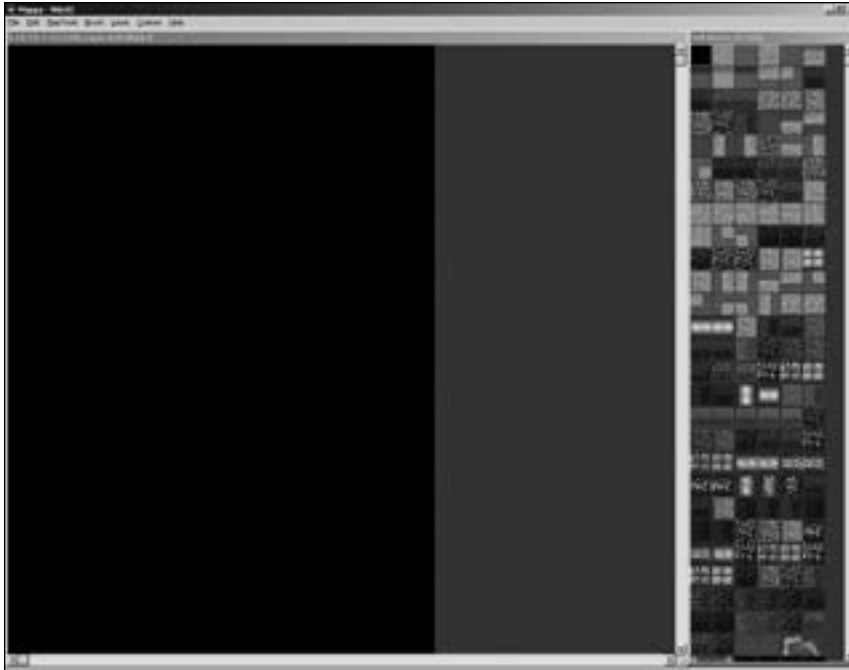


Figure 13.5 The tile palette has been filled with tiles imported from a bitmap file.

If the tiles look familiar, it's because most of them were used in the last chapter. I added new tiles to the `maptiles.bmp` file while working on the *Warbirds Europa* game. Note that when you add new tiles, you must add them to the bottom row of tiles, not to a column on the right. Mappy reads the tiles from left to right, top to bottom. You can add new tiles to the bottom of the `maptiles.bmp` file (which I have called `maptiles8.bmp` to reflect that it is an 8-bit image with 256 colors), and then import the file again into your Mappy map to start using new tiles. Simply select the first tile in the tile palette before you import again, and the existing tiles will be replaced with the new tiles.

Filling in the Tiles

Now that you have a big blank slate for the level, I want to show you how to create a template map file. Because the sample game in this chapter is a World War II shooter based on the arcade game *1942*, you can fill the entire level with a neutral water tile and then save it as a template. At that point, it will be relatively easy to use this template to create a number of levels for the actual game.

460 Chapter 13 ■ Vertical Scrolling Arcade Games

tip

All of the graphics in this game are available in the free SpriteLib GPL at <http://www.arifeldman.com>. Thanks to Ari Feldman for allowing me to use his tiles and sprites in this chapter.

Locate a water tile that is appealing to you. I have added two new water tiles just for this chapter, again from SpriteLib. Again, this was created by Ari Feldman and released into the public domain with his blessing. However, I encourage you to visit Ari's Web site at <http://www.arifeldman.com> to contact him about commissioning custom artwork for your own games. These are high-quality sprite tiles, and I am grateful to Ari for allowing me to use them.

Because this map is so big, it would take a very long time to fill in all the tiles manually. Thankfully, Mappy supports the Lua scripting language. Although it's beyond the scope of this chapter, you can edit Lua scripts and use them in Mappy. One such script is called Solid Rectangle, and it fills a region of the map with the selected tile. Unfortunately, there's a bug in this Lua script so it leaves out the last row and column of tiles. On a map this big, it takes a long time just to fill in a single column or row. I fixed the bug and have included the script on the CD-ROM. If you have just copied Mappy off the CD-ROM, then you should have the fix. If you have downloaded a new version of Mappy, then you'll have to fill in the unfilled tiles manually.

Having selected an appropriate water tile, open the Custom menu and select Solid Rectangle. A dialog box will appear, asking you to enter four numbers separated by commas. Type in these values:

0,0,20,1500

If you have the buggy version of this script, then type in:

0,0,19,1499

Now save the map as `template.fmp` so it can be reused to create each level of the game. By the way, while you have one large ocean level available, why not have some fun playing with Mappy? See what kind of interesting ocean level you can create using the available tiles. The map should look interesting, but it won't be critical to the game because all the action will take place in the skies.

Let's Scroll It

Now that you have a map ready to use, you can write a short program to demonstrate the feasibility of a very large scrolling level. Figure 13.6 shows the output from the *VerticalScroller* program. As was the case in the last chapter, you will need the MappyAL files to run this program. The `mappyal.c` and `mappyal.h` files are located on the CD-ROM under `\chapter13\VerticalScroller`.

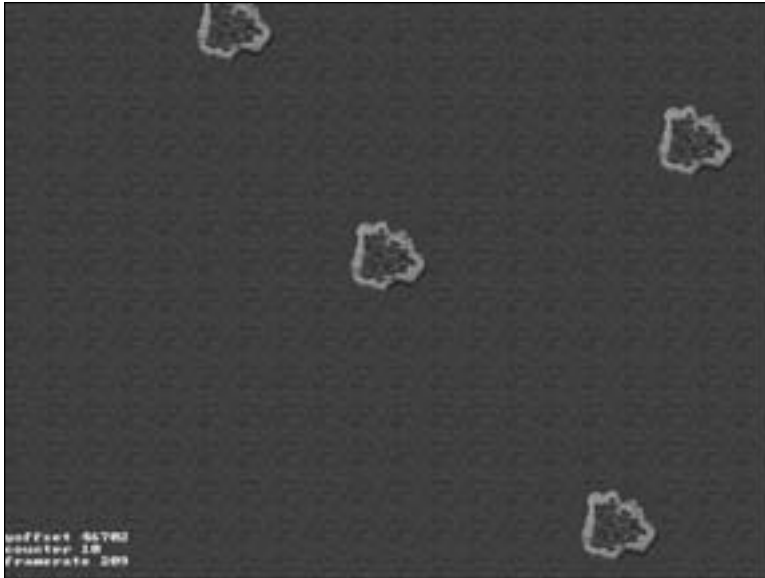


Figure 13.6 The *VerticalScroller* program contains the code for a basic vertical scroller engine.

```
#include "allegro.h"
#include "mappyal.h"

//this must run at 640x480
#define MODE GFX_AUTODETECT_FULLSCREEN
//#define MODE GFX_AUTODETECT_WINDOWED
#define WIDTH 640
#define HEIGHT 480
#define WHITE makecol(255,255,255)

#define BOTTOM 48000 - HEIGHT
//y offset in pixels
int yoffset = BOTTOM;

//timer variables
volatile int counter;
volatile int ticks;
volatile int framerate;

//double buffer
BITMAP *buffer;
```

462 Chapter 13 ■ Vertical Scrolling Arcade Games

```
//calculate framerate every second
void timer1(void)
{
    counter++;
    framerate = ticks;
    ticks=0;
}
END_OF_FUNCTION(timer1)

void main (void)
{
    //initialize program
    allegro_init();
    install_timer();
    install_keyboard();
    // set_color_depth(16);
    set_gfx_mode(MODE, WIDTH, HEIGHT, 0, 0);
    text_mode(-1);

    //create the double buffer and clear it
    buffer = create_bitmap(SCREEN_W, SCREEN_H);
    if (buffer==NULL)
    {
        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
        allegro_message("Error creating double buffer");
        return;
    }
    clear(buffer);

    //load the Mappy file
    if (MapLoad("level1.fmp"))
    {
        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
        allegro_message ("Can't find level1.fmp");
        return;
    }

    //set palette
    MapSetPal8();

    //identify variables used by interrupt function
    LOCK_VARIABLE(counter);
```



```
LOCK_VARIABLE(framerate);
LOCK_VARIABLE(ticks);
LOCK_FUNCTION(timer1);

//create new interrupt handler
install_int(timer1, 1000);

//main loop
while (!key[KEY_ESC])
{
    //check for keyboard input
    if (key[KEY_PGUP]) yoffset-=4;
    if (key[KEY_PGDN]) yoffset+=4;
    if (key[KEY_UP]) yoffset-=1;
    if (key[KEY_DOWN]) yoffset+=1;

    //make sure it doesn't scroll beyond map edge
    if (yoffset < 0) yoffset = 0;
    if (yoffset > BOTTOM) yoffset = BOTTOM;

    //draw map with single layer
    MapDrawBG(buffer, 0, yoffset, 0, 0, SCREEN_W-1, SCREEN_H-1);

    //update ticks
    ticks++;

    //display some status information
    textprintf(buffer,font,0,440,WHITE,"yoffset %d",yoffset);
    textprintf(buffer,font,0,450,WHITE,"counter %d", counter);
    textprintf(buffer,font,0,460,WHITE,"framerate %d", framerate);

    //blit the double buffer
    acquire_screen();
    blit (buffer, screen, 0, 0, 0, 0, SCREEN_W-1, SCREEN_H-1);
    release_screen();
}

//delete double buffer
destroy_bitmap(buffer);

//delete the Mappy level
MapFreeMem();
```

464 Chapter 13 ■ Vertical Scrolling Arcade Games

```
    allegro_exit();  
    return;  
}  
  
END_OF_MAIN()
```

Writing a Vertical Scrolling Shooter

To best demonstrate a vertical scroller, I have created a simple scrolling shooter as a sample game that you can use as a template for your own games of this genre. Simply replace the map file with one of your own design and replace the basic sprites used in the game, and you can adapt this game for any theme—water, land, undersea, or outer space.

Whereas the player's airplane uses local coordinates reflecting the display screen, the enemy planes use world coordinates that range from 0–639 in the horizontal and 0–47,999 in the vertical. Hey, I told you these maps were huge! The key to making this game work is that a test is performed after each sprite is drawn to determine whether it is within the visible range of the screen. Keep in mind that while the enemy fighters are moving toward the player, the map itself is scrolling downward to simulate forward movement.

Describing the Game

I have called this game *Warbirds Pacifica* because it was based on my earlier *Warbirds* game but set in the Pacific campaign of World War II. The game is set over ocean tiles with frequent islands to help improve the sense of motion (see Figure 13.7).

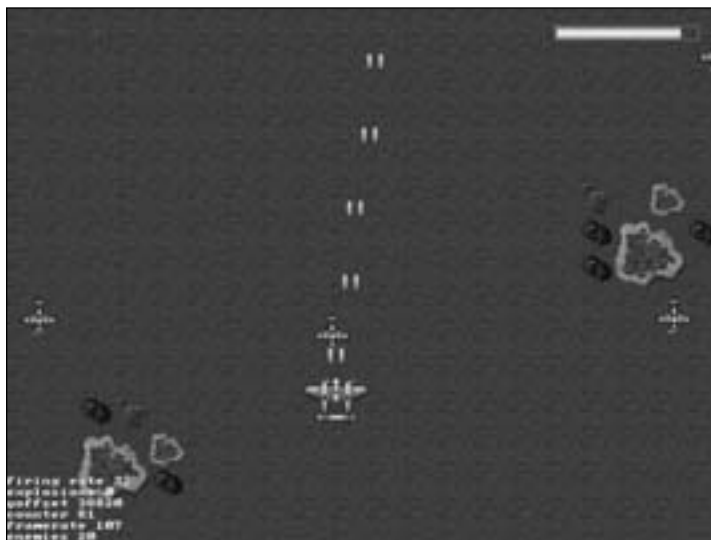


Figure 13.7 *Warbirds Pacifica* is a vertical scrolling shooter.

This is a fast-paced game and even with numerous sprites on the screen, the scrolling engine (provided by MappyAL) doesn't hiccup at all. Take a look at Figure 13.8. The player has a variable firing rate that is improved by picking up power-ups.



Figure 13.8 The firing rate of the player's P-38 fighter plane is improved with power-ups.

Another cool aspect of the game, thanks to Allegro's awesome sprite handling, is that explosions can overlap power-ups and other bullet sprites due to internal transparency within the sprites (see Figure 13.9). Note also the numerous debug-style messages in the bottom-left corner of the screen. While developing a game, it is extremely helpful to see status values that describe what is going on in order to tweak gameplay. I have modified many aspects of the game thanks to these messages.

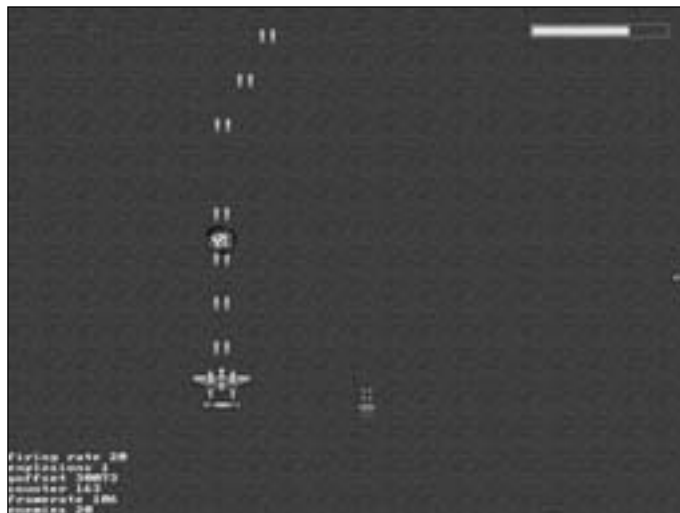


Figure 13.9 Destroying enemy planes releases power-ups that will improve the player's P-38 fighter.

466 Chapter 13 ■ Vertical Scrolling Arcade Games

Of course, what would the game be like without any challenge? Although this very early alpha version of *Warbirds Pacifica* does not have the code to allow enemy planes to fire at the player, it does detect collisions with enemy planes, which cause the player's P-38 to explode. (Although gameplay continues, the life meter at the top drops.) One of the first things you will want to do to enhance the game is add enemy firepower (see Figure 13.10).

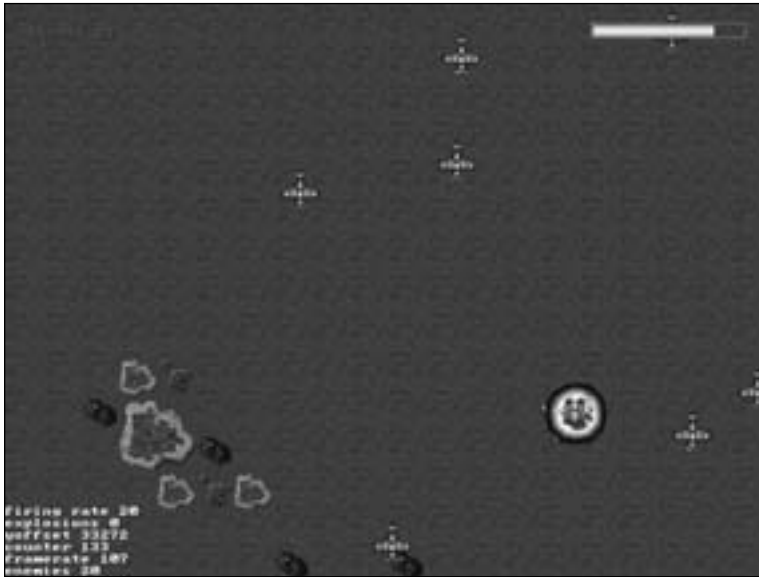


Figure 13.10 The enemy planes might not have much firepower, but they are still capable of Kamikaze attacks!

The Game's Artwork

This game is absolutely loaded with potential! There is so much that could be done with it that I really had to hold myself back when putting the game together as a technology demo for this chapter. It was so much fun adding just a single power-up that I came very close to adding all the rest of the power-ups to the game, including multi-shots! Why such enthusiasm? Because the artwork is already available for building an entire game, thanks to the generosity of Ari Feldman. The artwork featured in this game is a significant part of Ari's SpriteLib.

Let me show you some examples of the additional sprites available that you could use to quickly enhance this game. Figure 13.11 shows a set of enemy bomber sprites. The next image, Figure 13.12, shows a collection of enemy fighter planes that could be used in the game. Notice the different angles. Most shooters will launch squadrons of enemies at the player in formation, which is how these sprites might be used.

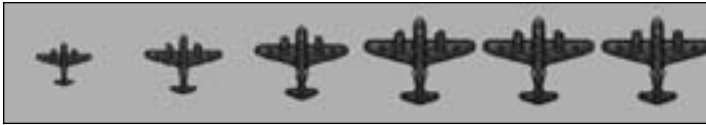


Figure 13.11 A set of enemy bomber sprites. Courtesy of Ari Feldman.

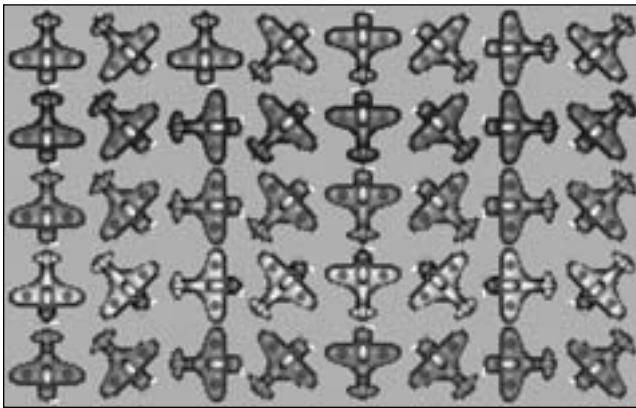


Figure 13.12 A collection of enemy fighter planes. Courtesy of Ari Feldman.

The next image, Figure 13.13, is an animated enemy submarine that comes up out of the water to shoot at the player. This would be a great addition to the game!

Yet another source of sprites for this game is shown in Figure 13.14—an enemy battleship with rotating gun turrets! The next image, Figure 13.15, shows a number of high-quality power-up sprites and bullet sprites. I used the shot power-up in the game as an example so that you can add more power-ups to the game.

Of course, a high-quality arcade game needs a high-quality font that looks really great on the screen. The default font with Allegro looks terrible and should not be used in a game like *Warbirds Pacifica*. Take a look at Figure 13.16 for a sample of the font available for the game with SpriteLib. You can use the existing menus and messages or construct your own using the provided alphabet.

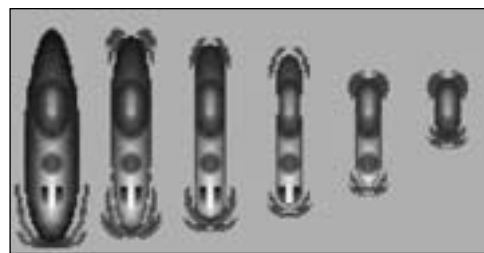


Figure 13.13 An enemy submarine sprite. Courtesy of Ari Feldman.

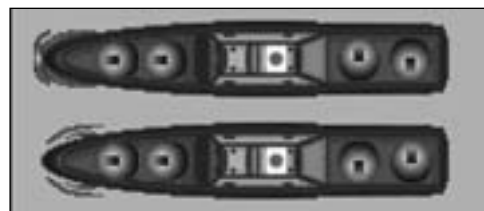


Figure 13.14 An enemy battleship with rotating gun turrets. Courtesy of Ari Feldman.

468 Chapter 13 ■ Vertical Scrolling Arcade Games

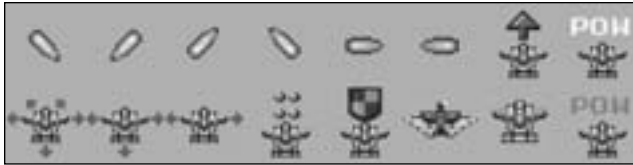


Figure 13.15 A collection of high-quality power-ups and bullets. Courtesy of Ari Feldman.



Figure 13.16 A high-quality font suitable for a scrolling shooter, such as *Warbirds Pacifica*. Courtesy of Ari Feldman.

Writing the Source Code

The source code for *Warbirds Pacifica* is designed to be easy to enhance because my intent was to provide you with a template, something to which you can apply your imagination to complete. The game has all the basic functionality and just needs to be well-rounded and, well, finished.

I recommend you use the *VerticalScroller* program as a basis because it already includes the two support files from the MappyAL library (*mappyal.c* and *mappyal.h*). If you are creating a new project from scratch, simply copy these two files to your new project folder and add them to the project by right-clicking on the project name and selecting Add Files to Project.

All the artwork for this game is located on the CD-ROM under \chapter13\Warbirds. You can open the project directly if you are not inclined to type in the source code; however, the more code you type in, the better programmer you will become. In my experience, just the act of typing in a game from a source code listing is a great learning experience. I see aspects of the game—and how it was coded—that are not apparent from simply paging

through the code listing. It helps you to become more intimate and familiar with the source code. This is an absolute must if you intend to learn how the game works in order to enhance or finish it.

warbirds.h

All of the struct and variable definitions are located in the warbirds.h file. You should add a new file to the project (File, New, C/C++ Header File) and give it this name.

```
#ifndef _WARBIRDS_H
#define _WARBIRDS_H

#include "allegro.h"
#include "mappyal.h"

//this must run at 640x480
//#define MODE GFX_AUTODETECT_FULLSCREEN
#define MODE GFX_AUTODETECT_WINDOWED
#define WIDTH 640
#define HEIGHT 480

#define WHITE makecol(255,255,255)
#define GRAY makecol(60,60,60)
#define RED makecol(200,0,0)

#define MAX_ENEMIES 20
#define MAX_BULLETS 20
#define MAX_EXPLOSIONS 10
#define BOTTOM 48000 - HEIGHT

//define the sprite structure
typedef struct SPRITE
{
    int dir, alive;
    int x,y;
    int width,height;
    int xspeed,yspeed;
    int xdelay,ydelay;
    int xcount,ycount;
    int curframe,maxframe,animdir;
    int framecount,framedelay;
}SPRITE;

//y offset in pixels
```

470 Chapter 13 ■ Vertical Scrolling Arcade Games

```

int yoffset = BOTTOM;

//player variables
int firecount = 0;
int firedelay = 60;
int health = 25;
int score = 0;

//timer variables
volatile int counter;
volatile int ticks;
volatile int framerate;

//bitmaps and sprites
BITMAP *buffer;
BITMAP *temp;
BITMAP *explosion_images[6];
SPRITE *explosions[MAX_EXPLOSIONS];
BITMAP *bigexp_images[7];
SPRITE *bigexp;
BITMAP *player_images[3];
SPRITE *player;
BITMAP *bullet_images[2];
SPRITE *bullets[MAX_BULLETS];
BITMAP *enemy_plane_images[3];
SPRITE *enemy_planes[MAX_ENEMIES];
BITMAP *progress, *bar;
BITMAP *bonus_shot_image;
SPRITE *bonus_shot;

#endif

```

main.c

Now for the main source code file. The `main.c` file will contain all of the source code for the *Warbirds Pacifica* template game. Remember, this game is not 100-percent functional for a reason—it was not designed to be a polished, complete game; rather, it was designed to be a template. To make this a complete game, you will want to create additional levels with Mappy; add some code to handle the loading of a new level when the player reaches the end of the first level; and add the additional enemy planes, ships, and so on, as described earlier. Then this game will rock! Furthermore, you will learn how to add sound effects to the game in Chapter 15, “Mastering the Audible Realm: Allegro’s Sound Support,” which will truly round out this game!


```
#include "warbirds.h"

//reuse our friendly tile grabber from chapter 9
BITMAP *grabframe(BITMAP *source,
                  int width, int height,
                  int startx, int starty,
                  int columns, int frame)
{
    BITMAP *temp = create_bitmap(width,height);

    int x = startx + (frame % columns) * width;
    int y = starty + (frame / columns) * height;

    blit(source,temp,x,y,0,0,width,height);

    return temp;
}

void loadsprites(void)
{
    int n;

    //load progress bar
    temp = load_bitmap("progress.bmp", NULL);
    progress = grabframe(temp,130,14,0,0,1,0);
    bar = grabframe(temp,6,10,130,2,1,0);
    destroy_bitmap(temp);

    //load bonus shot
    bonus_shot_image = load_bitmap("bonusshot.bmp", NULL);
    bonus_shot = malloc(sizeof(SPRITE));
    bonus_shot->alive=0;
    bonus_shot->x = 0;
    bonus_shot->y = 0;
    bonus_shot->width = bonus_shot_image->w;
    bonus_shot->height = bonus_shot_image->h;
    bonus_shot->xdelay = 0;
    bonus_shot->ydelay = 2;
    bonus_shot->xcount = 0;
    bonus_shot->ycount = 0;
    bonus_shot->xspeed = 0;
    bonus_shot->yspeed = 1;
    bonus_shot->curframe = 0;
```

472 Chapter 13 ■ Vertical Scrolling Arcade Games

```
bonus_shot->maxframe = 0;
bonus_shot->framecount = 0;
bonus_shot->framedelay = 0;

//load player airplane sprite
temp = load_bitmap("p38.bmp", NULL);
for (n=0; n<3; n++)
    player_images[n] = grabframe(temp,64,64,0,0,3,n);
destroy_bitmap(temp);

//initialize the player's sprite
player = malloc(sizeof(SPRITE));
player->x = 320-32;
player->y = 400;
player->width = player_images[0]->w;
player->height = player_images[0]->h;
player->xdelay = 1;
player->ydelay = 0;
player->xcount = 0;
player->ycount = 0;
player->xspeed = 0;
player->yspeed = 0;
player->curframe = 0;
player->maxframe = 2;
player->framecount = 0;
player->framedelay = 10;
player->animdir = 1;

//load bullet images
bullet_images[0] = load_bitmap("bullets.bmp", NULL);

//initialize the bullet sprites
for (n=0; n<MAX_BULLETS; n++)
{
    bullets[n] = malloc(sizeof(SPRITE));
    bullets[n]->alive = 0;
    bullets[n]->x = 0;
    bullets[n]->y = 0;
    bullets[n]->width = bullet_images[0]->w;
    bullets[n]->height = bullet_images[0]->h;
    bullets[n]->xdelay = 0;
    bullets[n]->ydelay = 0;
```

```
    bullets[n]->xcount = 0;
    bullets[n]->ycount = 0;
    bullets[n]->xspeed = 0;
    bullets[n]->yspeed = -2;
    bullets[n]->curframe = 0;
    bullets[n]->maxframe = 0;
    bullets[n]->framecount = 0;
    bullets[n]->framedelay = 0;
    bullets[n]->animdir = 0;
}

//load enemy plane sprites
temp = load_bitmap("enemyplane1.bmp", NULL);
for (n=0; n<3; n++)
    enemy_plane_images[n] = grabframe(temp,32,32,0,0,3,n);
destroy_bitmap(temp);

//initialize the enemy planes
for (n=0; n<MAX_ENEMIES; n++)
{
    enemy_planes[n] = malloc(sizeof(SPRITE));
    enemy_planes[n]->alive = 0;
    enemy_planes[n]->x = rand() % 100 + 50;
    enemy_planes[n]->y = 0;
    enemy_planes[n]->width = enemy_plane_images[0]->w;
    enemy_planes[n]->height = enemy_plane_images[0]->h;
    enemy_planes[n]->xdelay = 4;
    enemy_planes[n]->ydelay = 4;
    enemy_planes[n]->xcount = 0;
    enemy_planes[n]->ycount = 0;
    enemy_planes[n]->xspeed = (rand() % 2 - 3);
    enemy_planes[n]->yspeed = 1;
    enemy_planes[n]->curframe = 0;
    enemy_planes[n]->maxframe = 2;
    enemy_planes[n]->framecount = 0;
    enemy_planes[n]->framedelay = 10;
    enemy_planes[n]->animdir = 1;
}

//load explosion sprites
temp = load_bitmap("explosion.bmp", NULL);
for (n=0; n<6; n++)
    explosion_images[n] = grabframe(temp,32,32,0,0,6,n);
```

474 Chapter 13 ■ Vertical Scrolling Arcade Games

```
destroy_bitmap(temp);

//initialize the sprites
for (n=0; n<MAX_EXPLOSIONS; n++)
{
    explosions[n] = malloc(sizeof(SPRITE));
    explosions[n]->alive = 0;
    explosions[n]->x = 0;
    explosions[n]->y = 0;
    explosions[n]->width = explosion_images[0]->w;
    explosions[n]->height = explosion_images[0]->h;
    explosions[n]->xdelay = 0;
    explosions[n]->ydelay = 8;
    explosions[n]->xcount = 0;
    explosions[n]->ycount = 0;
    explosions[n]->xspeed = 0;
    explosions[n]->yspeed = -1;
    explosions[n]->curframe = 0;
    explosions[n]->maxframe = 5;
    explosions[n]->framecount = 0;
    explosions[n]->framedelay = 15;
    explosions[n]->animdir = 1;
}

//load explosion sprites
temp = load_bitmap("bigexplosion.bmp", NULL);
for (n=0; n<8; n++)
    bigexp_images[n] = grabframe(temp,64,64,0,0,7,n);
destroy_bitmap(temp);

//initialize the sprites
bigexp = malloc(sizeof(SPRITE));
bigexp->alive = 0;
bigexp->x = 0;
bigexp->y = 0;
bigexp->width = bigexp_images[0]->w;
bigexp->height = bigexp_images[0]->h;
bigexp->xdelay = 0;
bigexp->ydelay = 8;
bigexp->xcount = 0;
bigexp->ycount = 0;
bigexp->xspeed = 0;
bigexp->yspeed = -1;
```

```
bigexp->curframe = 0;
bigexp->maxframe = 6;
bigexp->framecount = 0;
bigexp->framedelay = 10;
bigexp->animdir = 1;

}

int inside(int x,int y,int left,int top,int right,int bottom)
{
    if (x > left && x < right && y > top && y < bottom)
        return 1;
    else
        return 0;
}

void updatesprite(SPRITE *spr)
{
    //update x position
    if (++spr->xcount > spr->xdelay)
    {
        spr->xcount = 0;
        spr->x += spr->xspeed;
    }

    //update y position
    if (++spr->ycount > spr->ydelay)
    {
        spr->ycount = 0;
        spr->y += spr->yspeed;
    }

    //update frame based on animdir
    if (++spr->framecount > spr->framedelay)
    {
        spr->framecount = 0;
        if (spr->animdir == -1)
        {
            if (--spr->curframe < 0)
                spr->curframe = spr->maxframe;
        }
        else if (spr->animdir == 1)
        {

```

476 Chapter 13 ■ Vertical Scrolling Arcade Games

```
        if (++spr->curframe > spr->maxframe)
            spr->curframe = 0;
    }
}

void startexplosion(int x, int y)
{
    int n;
    for (n=0; n<MAX_EXPLOSIONS; n++)
    {
        if (!explosions[n]->alive)
        {
            explosions[n]->alive++;
            explosions[n]->x = x;
            explosions[n]->y = y;
            break;
        }
    }

    //launch bonus shot if ready
    if (!bonus_shot->alive)
    {
        bonus_shot->alive++;
        bonus_shot->x = x;
        bonus_shot->y = y;
    }
}

void updateexplosions()
{
    int n, c=0;

    for (n=0; n<MAX_EXPLOSIONS; n++)
    {
        if (explosions[n]->alive)
        {
            c++;
            updatesprite(explosions[n]);
            draw_sprite(buffer, explosion_images[explosions[n]->curframe],
                explosions[n]->x, explosions[n]->y);
        }
    }
}
```

```
        if (explosions[n]->curframe >= explosions[n]->maxframe)
        {
            explosions[n]->curframe=0;
            explosions[n]->alive=0;
        }
    }
    textprintf(buffer,font,0,430,WHITE,"explosions %d", c);

    //update the big "player" explosion if needed
    if (bigexp->alive)
    {
        updatesprite(bigexp);
        draw_sprite(buffer, bigexp_images[bigexp->curframe],
            bigexp->x, bigexp->y);
        if (bigexp->curframe >= bigexp->maxframe)
        {
            bigexp->curframe=0;
            bigexp->alive=0;
        }
    }
}

void updatebonuses()
{
    int x,y,x1,y1,x2,y2;

    //add more bonuses here

    //update bonus shot if alive
    if (bonus_shot->alive)
    {
        updatesprite(bonus_shot);
        draw_sprite(buffer, bonus_shot_image, bonus_shot->x, bonus_shot->y);
        if (bonus_shot->y > HEIGHT)
            bonus_shot->alive=0;

        //see if player got the bonus
        x = bonus_shot->x + bonus_shot->width/2;
        y = bonus_shot->y + bonus_shot->height/2;
        x1 = player->x;
        y1 = player->y;
```

478 Chapter 13 ■ Vertical Scrolling Arcade Games

```
x2 = x1 + player->width;
y2 = y1 + player->height;

if (inside(x,y,x1,y1,x2,y2))
{
    //increase firing rate
    if (firedelay>20) firedelay-=2;

    bonus_shot->alive=0;
}

}

void updatebullet(SPRITE *spr)
{
    int n,x,y;
    int x1,y1,x2,y2;

    //move the bullet
    updatesprite(spr);

    //check bounds
    if (spr->y < 0)
    {
        spr->alive = 0;
        return;
    }

    for (n=0; n<MAX_ENEMIES; n++)
    {
        if (enemy_planes[n]->alive)
        {
            //find center of bullet
            x = spr->x + spr->width/2;
            y = spr->y + spr->height/2;

            //get enemy plane bounding rectangle
            x1 = enemy_planes[n]->x;
            y1 = enemy_planes[n]->y - yoffset;
            x2 = x1 + enemy_planes[n]->width;
            y2 = y1 + enemy_planes[n]->height;
```



```
//check for collisions
if (inside(x, y, x1, y1, x2, y2))
{
    enemy_planes[n]->alive=0;
    spr->alive=0;
    startexplosion(spr->x+16, spr->y);
    score+=2;
    break;
}
}
}

void updatebullets()
{
    int n;
    //update/draw bullets
    for (n=0; n<MAX_BULLETS; n++)
        if (bullets[n]->alive)
        {
            updatebullet(bullets[n]);
            draw_sprite(buffer,bullet_images[0], bullets[n]->x, bullets[n]->y);
        }
}

void bouncex_warpy(SPRITE *spr)
{
    //bounces x off bounds
    if (spr->x < 0 - spr->width)
    {
        spr->x = 0 - spr->width + 1;
        spr->xspeed *= -1;
    }

    else if (spr->x > SCREEN_W)
    {
        spr->x = SCREEN_W - spr->xspeed;
        spr->xspeed *= -1;
    }

    //warps y if plane has passed the player
    if (spr->y > yoffset + 2000)
```

480 Chapter 13 ■ Vertical Scrolling Arcade Games

```
{
    //respawn enemy plane
    spr->y = yoffset - 1000 - rand() % 1000;
    spr->alive++;
    spr->x = rand() % WIDTH;
}

//warps y from bottom to top of level
if (spr->y < 0)
{
    spr->y = 0;
}

else if (spr->y > 48000)
{
    spr->y = 0;
}
}

void fireatenemy()
{
    int n;
    for (n=0; n<MAX_BULLETS; n++)
    {
        if (!bullets[n]->alive)
        {
            bullets[n]->alive++;
            bullets[n]->x = player->x;
            bullets[n]->y = player->y;
            return;
        }
    }
}

void displayprogress(int life)
{
    int n;
    draw_sprite(buffer,progress,490,15);

    for (n=0; n<life; n++)
        draw_sprite(buffer,bar,492+n*5,17);
}
```

```
void updateenemyplanes()
{
    int n, c=0;

    //update/draw enemy planes
    for (n=0; n<MAX_ENEMIES; n++)
    {
        if (enemy_planes[n]->alive)
        {
            c++;
            updatesprite(enemy_planes[n]);
            bouncex_warpy(enemy_planes[n]);

            //is plane visible on screen?
            if (enemy_planes[n]->y > yoffset-32 && enemy_planes[n]->y <
                yoffset + HEIGHT+32)
            {
                //draw enemy plane
                draw_sprite(buffer, enemy_plane_images[enemy_planes[n]->curframe],
                    enemy_planes[n]->x, enemy_planes[n]->y - yoffset);
            }
        }
        //reset plane
        else
        {
            enemy_planes[n]->alive++;
            enemy_planes[n]->x = rand() % 100 + 50;
            enemy_planes[n]->y = yoffset - 2000 + rand() % 2000;
        }
    }
    textprintf(buffer,font,0,470,WHITE,"enemies %d", c);
}

void updatescroller()
{
    //make sure it doesn't scroll beyond map edge
    if (yoffset < 5)
    {
        //level is over
        yoffset = 5;
        textout_centre(buffer, font, "END OF LEVEL", SCREEN_W/2,
            SCREEN_H/2, WHITE);
    }
}
```

482 Chapter 13 ■ Vertical Scrolling Arcade Games

```

    if (yoffset > BOTTOM) yoffset = BOTTOM;

    //scroll map up 1 pixel
    yoffset-=1;

    //draw map with single layer
    MapDrawBG(buffer, 0, yoffset, 0, 0, SCREEN_W-1, SCREEN_H-1);
}

void updateplayer()
{
    int n,x,y,x1,y1,x2,y2;

    //update/draw player sprite
    updatesprite(player);
    draw_sprite(buffer, player_images[player->curframe],
        player->x, player->y);

    //check for collision with enemy planes
    x = player->x + player->width/2;
    y = player->y + player->height/2;
    for (n=0; n<MAX_ENEMIES; n++)
    {
        if (enemy_planes[n]->alive)
        {
            x1 = enemy_planes[n]->x;
            y1 = enemy_planes[n]->y - yoffset;
            x2 = x1 + enemy_planes[n]->width;
            y2 = y1 + enemy_planes[n]->height;
            if (inside(x,y,x1,y1,x2,y2))
            {
                enemy_planes[n]->alive=0;
                if (health > 0) health--;
                bigexp->alive++;
                bigexp->x = player->x;
                bigexp->y = player->y;
                score++;
            }
        }
    }
}

```

```
void displaystats()
{
    //display some status information
    textprintf(buffer,font,0,420,WHITE,"firing rate %d", firedelay);
    textprintf(buffer,font,0,440,WHITE,"yoffset %d",yoffset);
    textprintf(buffer,font,0,450,WHITE,"counter %d", counter);
    textprintf(buffer,font,0,460,WHITE,"framerate %d", framerate);

    //display score
    textprintf(buffer,font,22,22,GRAY,"SCORE: %d", score);
    textprintf(buffer,font,20,20,RED,"SCORE: %d", score);
}

void checkinput()
{
    //check for keyboard input
    if (key[KEY_UP])
    {
        player->y -= 1;
        if (player->y < 100)
            player->y = 100;
    }
    if (key[KEY_DOWN])
    {
        player->y += 1;
        if (player->y > HEIGHT-65)
            player->y = HEIGHT-65;
    }
    if (key[KEY_LEFT])
    {
        player->x -= 1;
        if (player->x < 0)
            player->x = 0;
    }
    if (key[KEY_RIGHT])
    {
        player->x += 1;
        if (player->x > WIDTH-65)
            player->x = WIDTH-65;
    }

    if (key[KEY_SPACE])
    {

```

484 Chapter 13 ■ Vertical Scrolling Arcade Games

```
        if (firecount > firedelay)
        {
            firecount = 0;
            fireatenemy();
        }
    }

//calculate framerate every second
void timer1(void)
{
    counter++;
    framerate = ticks;
    ticks=0;
    rest(2);
}
END_OF_FUNCTION(timer1)

void initialize()
{
    //initialize program
    allegro_init();
    install_timer();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode(MODE, WIDTH, HEIGHT, 0, 0);
    text_mode(-1);
    srand(time(NULL));

    //create the double buffer and clear it
    buffer = create_bitmap(SCREEN_W, SCREEN_H);
    if (buffer==NULL)
    {
        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
        allegro_message("Error creating double buffer");
        return;
    }
    clear(buffer);

    //load the Mappy file
    if (MapLoad("level1.fmp"))
    {
```

```
        set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
        allegro_message ("Can't find level1.fmp");
        return;
    }

    //set palette
    MapSetPal8();

    //identify variables used by interrupt function
    LOCK_VARIABLE(counter);
    LOCK_VARIABLE(framerate);
    LOCK_VARIABLE(ticks);
    LOCK_FUNCTION(timer1);

    //create new interrupt handler
    install_int(timer1, 1000);
}

void main (void)
{
    int n;

    //init game
    initialize();
    loadsprites();

    //main loop
    while (!key[KEY_ESC])
    {
        checkinput();

        updatescroller();

        updateplayer();
        updateenemyplanes();

        updatebullets();
        updateexplosions();
        updatebonuses();

        displayprogress(health);
        displaystats();
    }
}
```

486 Chapter 13 ■ Vertical Scrolling Arcade Games

```
//blit the double buffer
acquire_screen();
    blit (buffer, screen, 0, 0, 0, 0, SCREEN_W-1, SCREEN_H-1);
release_screen();

ticks++;
firecount++;
}

//delete the Mappy level
MapFreeMem();

//delete bitmaps
destroy_bitmap(buffer);
destroy_bitmap(progress);
destroy_bitmap(bar);

for (n=0; n<6; n++)
    destroy_bitmap(explosion_images[n]);

for (n=0; n<3; n++)
{
    destroy_bitmap(player_images[n]);
    destroy_bitmap(bullet_images[n]);
    destroy_bitmap(enemy_plane_images[n]);
}

//delete sprites
free(player);
for (n=0; n<MAX_EXPLOSIONS; n++)
    free(explosions[n]);
for (n=0; n<MAX_BULLETS; n++)
    free(bullets[n]);
for (n=0; n<MAX_ENEMIES; n++)
    free(enemy_planes[n]);

    allegro_exit();
    return;
}

END_OF_MAIN()
```


Summary

Vertical scrolling shooters were once the mainstay of the 1980s and 1990s video arcade, but have not been as prevalent in recent years due to the invasion of 3D, so to speak. Still, the scrolling shooter as a genre has a large and loyal fan following, so it will continue to be popular for years to come. This chapter explored the techniques involved in creating vertical scrollers and produced a sample template game called *Warbirds Pacifica* using the vertical scroller engine (which is really powered by the MappyAL library). I hope you enjoyed this chapter because this is not the end of the scroller! The next chapter takes a turn—a 90-degree turn, as a matter of fact—and covers the horizontal scroller.

Chapter Quiz

You can find the answers to this chapter quiz in Appendix A, “Chapter Quiz Answers.”

1. In which game genre does the vertical shooter belong?
 - A. Shoot-em-up
 - B. Platform
 - C. Fighting
 - D. Real-time strategy
2. What is the name of the support library used as the vertical scroller engine?
 - A. ScrollerEngine
 - B. VerticalScroller
 - C. MappyAL
 - D. AllegroScroller
3. What are the virtual pixel dimensions of the levels in *Warbirds Pacifica*?
 - A. 640×480
 - B. 48,000×640
 - C. 20×1500
 - D. 640×48,000
4. What is the name of the level-editing program used to create the first level of *Warbirds Pacifica*?
 - A. Happy
 - B. Mappy
 - C. Snappy
 - D. Frappy

488 Chapter 13 ■ Vertical Scrolling Arcade Games

5. How many tiles comprise a level in *Warbirds Pacifica*?
 - A. 30,000
 - B. 1,500
 - C. 48,000
 - D. 32,768
6. Which of the following games is a vertical scrolling shooter?
 - A. *R-Type*
 - B. *Mars Matrix*
 - C. *Contra*
 - D. *Castlevania*
7. Who created the artwork featured in this chapter?
 - A. Ray Kurzweil
 - B. Clifford Stoll
 - C. Ari Feldman
 - D. Nicholas Negroponte
8. Which MappyAL function loads a map file?
 - A. LoadMap
 - B. MapLoad
 - C. LoadMappy
 - D. ReadLevel
9. Which MappyAL function removes a map from memory?
 - A. destroy_map
 - B. free_mappy
 - C. DeleteMap
 - D. MapFreeMem
10. Which classic arcade game inspired *Warbirds Pacifica*?
 - A. *Pac-Man*
 - B. *Mars Matrix*
 - C. *1942*
 - D. *Street Fighter II*