Written by Billy D. Spelchan for www.BlazingGames.com

Chapter 33

Arcade Game Summary

Contents

This is a simple summary of what was learned in this part of the book and some suggestions on how you can applied what was learned here towards your own projects.

Summary of Chapter 28: Arcade Games

What are Arcade Games
The term comes from the generic term for coin-operated video games. These games tend to be more reflex related. Early arcade games were rather primitive, but it was playing these games that got me interested in game development.

What's Required for Writing Arcade Games
While arcade games seem to be fairly simple (and in fact, in many cases they are) creating an arcade game in Flash is going to require a knowledge of Action Script. Arcade games tend to require a lot of interaction with the game's environment. Since this type of interaction can happen at any time and is in the player's control, things can get complicated really quickly. Graphics are another area that Video Games require, but these can be built into Flash. Sound Effects are another requirement. This one will pose a bit of a problem. Flash can easily import sound files.

String Along Game Design
The first arcade game that we are going to create is going to be called String Along. This game is a variant of the common tap-worm or snake games. Essentially the player controls a sphere and needs to collect energy. However, when the player collects energy, the player grows. If the player runs into an obstacle or himself, the player loses a life.

Designing Lights Out
We are then going to create a game that could be considered an extended version of pong. This game, which I am calling "Lights Out" consists of a playfield filled with colored balls. you have an instigator ball. This ball is the only one that you can touch and you have to keep it from going past you.

Summary of Chapter 29: Game Concepts

Lists
The list starts with an object within the list. This object points to the next object in the list. To traverse the list you simply need to go to the first object on the list and use it's next object to get to the next object. The next object has a reference to the next object after it. You keep going to the next object (doing whatever it is that you are doing with these objects) until the next object is null. A doubly linked list is very similar, but also points to the previous object. Flash makes it possible to have the flexibility of linked lists without there complexity by having a very robust Array class.

Sprites
Sprites and Movie Clips are pretty much the same thing. Movie clips should be thought of as very sophisticated sprites. While this may be a bit of a letdown, quite often techniques and algorithms have complex sounding names simply because people need to call them something. In fact, many advanced graphic algorithms are simply named after the person who came up with the technique.

Tile sets
Essentially, the game's playable area (playfield) is broken into a grid. Each grid holds a tile id value, so a large area can be represented with a very small amount of information.

Scrolling
Tile sets can be of even greater use for larger worlds. If you wanted to, you could create a playfield that was 32x32, even though the viewable portion of the map is 20x15. By doing so, only a portion of the playfield, commonly called the viewport, would be visible. As the player moved, the other portions of the playfield would be shown. This is what is known as scrolling.

Summary of Chapter 30: String Along

Building the Tile set
In this game, there are three types of tiles. There is the empty tile, which has nothing in it and makes up the majority of tiles. There is the obstacle, which the player has to avoid. Finally, there is the energy tile, which the player is trying to pick up. We could also have the player as part of the playfield, but I have chosen to have the player in a separate layer. Why? Quite simply, I want the player to move smoothly. Having the player as part of the playfield would either greatly complicate the playfield movie or would require that the player only be able to move a whole tile at a time.

The Basic Playfield
The basic playfield will be 40x26. This results in an actual playfield size of 640x416. It is useful to create the playfield movie by simply drawing a hollow rectangle that is 640x416 and then turning that hollow rectangle into a movie clip.

Playfield Control
Having a playfield that can vary certainly adds to the game, but we need a way of telling Flash what the playfield looks like. When you think about it, all we have to do is draw blocks of obstacle tiles. What if we sent the playfield an array that contains a list of rectangles. If the array was null, then the playfield would create an empty playfield, otherwise it would loop through the elements using groups of four elements as the bounds of blocks to draw.

Player Layer
The player layer is also a movie clip. The main part of the player movie is the string of balls that represent the player. We will use a linked list to handle the string. While we could just use an array (as Flash's arrays have the ability to grow and shrink), we learned about linked lists last chapter so now's our chance to play around with them.

Moving Around
While we have created some of the movement code that will be needed, before any movement can happen two things are needed. First, we need to be able to control the animation on a per-frame basis. This can be done by overriding the onEnterFrame function in the player movie and providing our own playerTick() function. The other function that is needed is the ability to handle the keyboard. Flash has an onKeyDown function that handles key presses. The information about what is happening with the keyboard is stored in the global Key class. Likewise, the Key class has a bunch of constants defined for keys such as the cursor keys.

Collision Detection

There are three types of collisions that we have to worry about. One is with yourself. Two are with the playfield. The first of the playfield collisions is with a wall. The second is with food. Collision with oneself is easy to check, as you only have to worry about the head coliding with something. Our idea of a collision is when the player attempts to go through a part of ones body. This requires the head go through another location. A simple method can do the check. The same principle can be applied to the playfield, though we will need a more complex return value. -1 is a wall, 0 is empty, 1 is food.

Feeding Frenzy

In order for the player to eat objects, they need to be placed on the playfield. The way we will do this is with a simple playfield function which we will call addFood. This will work by randomly selecting a location on the playfield. If the location on the playfield is empty, the location will have it's tile type changed to food otherwise another location will be selected. When a player eats food, a new node is added to the players' body. This is set up across two frames, to make sure that the newly cloned objects initiate properly (though if we were using a Flash MX 2004 movie, we could get by this problem by having a proper class tied to the string node symbol).

Scoring

We are ready to add scoring to the game. At the beginning of the main movie we add the code that sets all the variables to the appropriate values. In addition, in order to keep a high score, we add a high score variable which we will define in the games initialization so that it only gets reset the first time the game is played. Score text is dynamic text which is placed in it's own layer, with a function to update all text created. The eat function also is updated so it properly tracks the score.

Title

Now we want to do a title sequence. When you think about it, The game is called String Along, so the title appears as a string of letters following the leader until they reached their desired locations. The last thing we have to do is create buttons for the five different speeds that we have.

Summary of Chapter 32: Lights Out

Designing the Lights and Racket
Originally I started with your traditional pong paddle. I rounded off the corners but that still didn't give me the look I wanted. I then went with an oval shape. This gave me the idea of turning the player's paddle into something that actually looked like a racket. This was done by adding string and then colouring the oval to make it look more like wood. The instigator ball also serves as a template for the colored lights. This is because the colored lights are essentially larger versions of the instigator ball. I stared with a flat ball. To that I switched to a radial fill to give me more of a three-dimensional look for the ball. Finally I increased the size of the glow spot by adding another white and a lighter gray area in the gradient.

Coding for Color
Obviously we are going to need a way to change the color of the ball. At the same time we are going to want some easy to remember color constants. The constants can be defined in an initialization phase. To set the color of the ball we simply write a function which will change the frame to the appropriate color.

Some Paddling Action
At this point we want to get the racket to be tied with movement of the mouse. For this, we move the racket to the bottom of the window. We then write the onMouseMove function in the newly created code layer of the main movie.

A Fancy Layout Sequence
First, we need to add a bit to the initialization of the game. This code is added to the initialize function.Now that we are starting the level, we want the lights to appear on the screen. By creating a bit of a loop we can have the lights appear one at a time.

Dropping the Ball
We are ready to add the instigator ball into the game. The speed of the ball will be based on the level the player is on, so we are going to have to add support for tracking the current level. Next we need to set up some constants for the instigator. Now, we want to add a new label to our timeline, which we will call "DropInstigator". This will have the code for resetting the ball when it is lost. When the ball finally does appear, we want to give the player a chance to re-orient themselves to the game. To do this we give the ball a little hover time before we get to the main game play loop, which is labelled "MainLoop". In this loop we adjust the position of the instigator.

Bouncing

Right now the instigator ball just drops off the screen. What we need is some bouncing. First, let's worry about bouncing off the four walls (even the bottom wall for now, though later this will result in loss of a ball). This is going to need some new constants which we will add to the game initialization function. The computer doesn't know how to bounce an object around so we will add a bounce function. Finally we replace the code on the frame right after the MainLoop label with the following code. This is what determines if the instigator has hit a wall.

Bouncing off Rackets and Lights

We already have a racket that the player can control, so now we need code to handle bouncing the instigator ball off the player's racket. To give the player a bit more control of the ball, the angle will be dependent on where the ball hits the racket. Now comes the dreaded hitting the lights part of the game. The first thing we are going to need is a function for calculating an angle between two points. The next step is to actually see if the instigator has hit one of the balls.

Smashing Lights

The instigator ball now bounces off lights, but we want the lights to break. More particularly, if the light is a non-prime color, we want one of the colors to be knocked off of the light. To handle this, we add a new function to the light movie that will change the color for us and returns true if the light has been destroyed. Finally we add the code for handling the lights splitting or breaking. This code is placed in the true portion of the collision detection code we wrote.

Clearing Levels

At this point we have quite a nice game. What we need to do now is handle the clearing of a level. When you think about it, a level is clear when the dead light list has 140 lights in it.

Scoring

Now we are ready to add scoring to the game. There are 4 elements of the score line that we are going to track. The level, lives, high score, and the score. We need to create a layer to hold the four display variables. These are just dynamic text blocks. To actually implement the scoring I add a bit of code to the block of code for handling balls being hit.

Ending the Game

The game will now run forever, as there currently is no way of losing the game. In order for the player to lose, they must lose all the instigator balls. Right now that is not possible as the ball bounces when it hits the bottom of the screen. It is a fairly simple to change this so that. Now, we need a game over sequence. This could be just a simple text message, and in fact, for the most part that is all that it is. I just feel that there needs to be some type of feeling of finality. To achieve that, I am going to make the letters appear slowly.

The Title Screen

The title screen needed to reflect the game. To do this, I thought of having a growing spotlight with LIGHTS being in the various light colors and OUT being black. Designing this image was very simple. I started with a circle and then wrote the word LIGHTS in a big font. Broke it apart twice to turn the letters into objects. I colored the letters and placed them in the appropriate locations in the circle. I did the same for OUT, except in that case it was colored black.

Fine Tuning

First, all arcade games need decent sound. Importing sound files is simple enough. Next, let us adjust the starting location of the lights and paddle and then speed up the ball. This is just a mater of adjusting the constant values we set up in the initialization section of the code. The game is a bit nicer but it is kind of hard to figure out when the ball is going to bounce of a wall. The solution, draw a box around the screen. Finally, the start button could be a bit better.

Projects

There are lots of different arcade games out there, so ideas for specific games are probably not needed. For those of you who are interested in practising what you have learned before starting work on your own games, here are a few suggestions. Other than additional level sets, I have not actually done any of the projects in this section, so don't bother looking on the CD for these.

First, I would recommend creating your own set of String Along levels. This is not too difficult to do and will help you get use to working with levels. Extending the code to String Along to add new tiles to the game may also be possible. Also, if you wanted to get really elaborate, you could create a two player version of the game. This could be interesting game play as players would have to race for food while also avoiding each other.

Lights Out was originally going to have falling lights in it. The player would have to avoid these lights or be paralysed for a few seconds after touching the falling light. The game proved long enough on it's own without adding a lot of complexity to incorporate this feature, so this idea was dropped from the game.

One thing I would recommend to anyone who is thinking about creating arcade games in Flash would be to use Flash MX 2004 or later, as Action Script 2 is far nicer to work with when it comes to creation of classes. As arcade games are going to require code, having a better programming environment to begin with will make the creation of the game much smoother.