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

Chapter 24

Rooms Engine

Contents

The other approach that we take towards creating an adventure game is to create a game where every room in the game is a separate movie, with a single master movie being in control of everything.

My First Prototype

As I said earlier, my dragon and the sword game was a result of talking to some contacts of mine about Flash. I had told them that I thought it would be easy to use Flash to put together an adventure game. As a result of that conversation, I quickly put together a prototype of Dragon and the Sword (which we will be creating in the next few chapters).

While it is possible to have the entire adventure game in a single Flash file, if the game is fairly large, and especially if there are going to be a lot of animated sequences in the game, then this may not be a realistic option for a variety of reasons.

The biggest reason being the file sizes. If you had a 100MB Flash movie, there would be lots of problems streaming the file. In fact, there could be problems with the flash player's ability to even handle such a large file. I don't know if Flash keeps the entire movie in memory, but if it does then that alone is a problem. The difficulty working with such a large source file, and the fragility that such a large file have are also big concerns. Remember that the source file is usually much larger than the resulting movie file. Just working with such a huge movie would be a nightmare.

As this is the key problem that has to be solved before Flash adventure games can be developed, I had to come up with a solution. The obvious solution is to break up an adventure into a huge number of smaller episodes. For a linear story, such as my One of those Weeks, this is quite a simple process. For a less-linear adventure, however, this could be a problem. Thankfully, Flash has a solution to this problem.

As you should already know, Flash allows you to have movies within movies. Flash takes this even further. It is possible to have a movie clip be replaced with a new movie, and that movie can be a separate movie file. In other words, if you broke down the game into rooms, each room could be a separately handled Flash file. Even better, the movie that is loaded can actually make function calls to the movie that loaded it.

In other words, it is possible to have one main Flash movie that handles all the games logic and puzzles, while having separate movies for handling each room, possibly with room specific material. In fact, it would be possible to have special animation sequences that can be called up when they are needed. If the animation sequence is designed to be streamable, there wouldn't even have to be any type of wait message, the cut scene would just play naturally.

For any sizable non-linear story this is probably the best route to take. Again, both types of methods for creating adventures are presented in this part of the book so you can take a look at your adventure game's requirements and decide which method is the best one for your game.

The Blank Movie

As you should already know, Flash allows you to have movies within movies. Flash takes this even further. It is possible to have a movie clip be replaced with a new movie, and that movie can be a separate movie file. In other words, if you broke down the game into rooms, each room could be a separately handled flash file. Even better, the movie that is loaded can actually make function calls to the movie that loaded it.

To make use of such a scheme, we will need a main movie, which we will now start to create. First, the main movie is broken into 4 different layers. The layers are: Code, Popup, Inventory, and RoomLayer.

Code is my standard layer where I place all the Action Script code. Popup and Inventory will be for the user interface which will be discussed later in this chapter. The RoomLayer is where the rooms will be placed.

To switch movies, we will need a placeholder movie. This is easily created by creating a 640x400 blank rectangle and add it to the library as a movie clip. Make sure to set the reference to the top left corner as any swf movies that are loaded externally seem to have this as their reference spot. If you want to get a bit fancier with the blank movie, you could have it contain the words, “Please wait, loading”.

Ideas for Improving the Loading of Rooms

For my prototype I was not to worried about delays in swapping the movie out, though for larger projects this could be a concern. One way to deal with this concern would be to have a message box saying "Please wait...loading" appear while the movie is loading.

A better method, though one that requires a bit more work, would be to have transition effects occur while the movie is being loaded. When the player first leaves the room, you would start a sequence where a transition out effect (such as fading or a sweeping effect) takes place. You would then load the movie (possibly with a loading message). Finally you would have a transition in effect to show the new room.

In Flash, because you are able to have movies within movies, any movie clip object has a variable called _parent. This variable is the movie that contains the movie clip. In addition, all objects have access to a global variable named _root which contains the main time-line.

Any movie clips that a movie clip creates are known as children. Children get assigned to their _parent variable a reference to the movie clip that created them. The _root variable, on the other hand, is always going to be the main movie time-line.

Any functions in the parent movie can be called by simply using

```
_parent.functionname(params);
```

Where functionname is the name of the function and params are the parameters (if any) for the function. If the function you are calling does not exist, nothing happens, so make sure you are typing the name of the function correctly. In fact, not typing the function name correctly is a common bug. Flash MX 2004's Action Script 2 is a bit better at detecting such problems but it is still a problem that you should be aware of.

Moving from Room to Room

One important aspect of the game is the moving between rooms. This is directly tied to how movies are loaded so we are able to combine work on both tasks into the same block of work. Of course, we need a room. I place all my room movies in a separate directory called rooms.

While we could create the first room movie at this time, this is not really necessary. Instead, so that we can see our work in action quickly, we will just create a placeholder movie. This movie is 640x400 and will just contain the words “Place real room here”. This movie file can be named whatever we want as long as the exported movie it creates is called “Room1.swf”.

First thing I do, is on frame 2 of the movie I create a label in the code layer called Init. This is the frame where all the game initialization and all the common functions of the game will be placed. Initially there is not much code that needs to be done. We will write initGame and restartGame functions, even though neither function does very much at the moment. As with other projects we have done, additional code will be added to these functions later. We will need a routine to load new rooms, which we will also write at this time.

```
initGame();
restartGame();

function initGame()
{
      // make sure function only executes once
      if (gameInitialized != undefined)
            return;
      gameInitialized = true;
}

function restartGame()
{
      nextRoom = "rooms/Room1.swf";
}


function changeRoom(n)
{
      trace ("Changing room to " + n);
      nextRoom = "rooms/Room" + n + ".swf";
      gotoAndPlay("Loader");
}
```

As you can see, we have a variable called nextRoom in which the name of the file for the next room is contained. The code starts by putting room1 into this variable so that whenever the game starts the player will be in room 1. At the moment, that is the only game initialization that is needed. Once we start scripting the game, additional initialization will be required, but we will cover that when we come to it.

The changeRoom function simply changes the nextRoom variable to the room of the file to load. It then moves the movie to a frame called loader, which is the frame that actually loads the movie. As mentioned in an earlier section, it would be possible to have some type of transitioning animation be called.

The fifth frame is labelled Loader. This frame contains a single line of code

```
room_movie.loadMovie(nextRoom);
```

Yes, that is all that is needed to load a movie! We conclude the main movie file by having frame 10 labelled Main and having stop() called in frame 11.

How Inventory will Work

The inventory system is the most important part of many adventure games. The inventory system allows for interaction between the player and objects in the game. In Dragon and the Sword, the player is able to pick up items, which are then added to the inventory bar. When they are in the proper location, players are able to click on the item in the inventory bar which will result in something happening in the game.

To program the inventory system, every item in the game will have a button created for it. This button will be placed on the inventory bar. All the inventory buttons will be made invisible (by setting the _visible variable to false) until the player has picked up the item associated with the button. This means that the game will need to track which items the player has picked up. This information will also be needed by the individual room movies, as they need to know if there is an object in the room.

Within Dragon and the Sword, there are the following inventory items:

Bar
This object can be bent on a rock to become a hook. The hook is used in conjunction with rope to allow the player to cross a chasm.

Metal Chunk
Used with flint and moss to start a fire. The fire is used to unfreeze the sword from a block of ice.

Rope
Used with hook to cross chasm.

Flint
Used with metal chunk and moss to start a fire. The fire is used to unthaw the sword from a block of ice.

Moss
Used with metal chunk and flint to start a fire. The fire is used to unthaw the sword from a block of ice.

Sword.
Used to win game.

The Map

Perhaps the best way of developing an adventure game is to work out the story and then from the story work out the map and the puzzles. The basic story for Dragon and the Sword is that there is a dragon that is terrorizing the player's village. The player needs to find a magic sword in order to get rid of the dragon. An overly simple story, but this is a very small adventure so that is no big deal.

With the story in hand, we can now work out the map and some puzzles for the player to solve. I drew out my original map on the back of an envelope. I had considered scanning in the envelope, but figured I would spend a bit of time in my paint program and produce a nice looking map. If you are planning on playing my Dragon and the Sword game, do so now as the rest of this section will ruin the game for you.

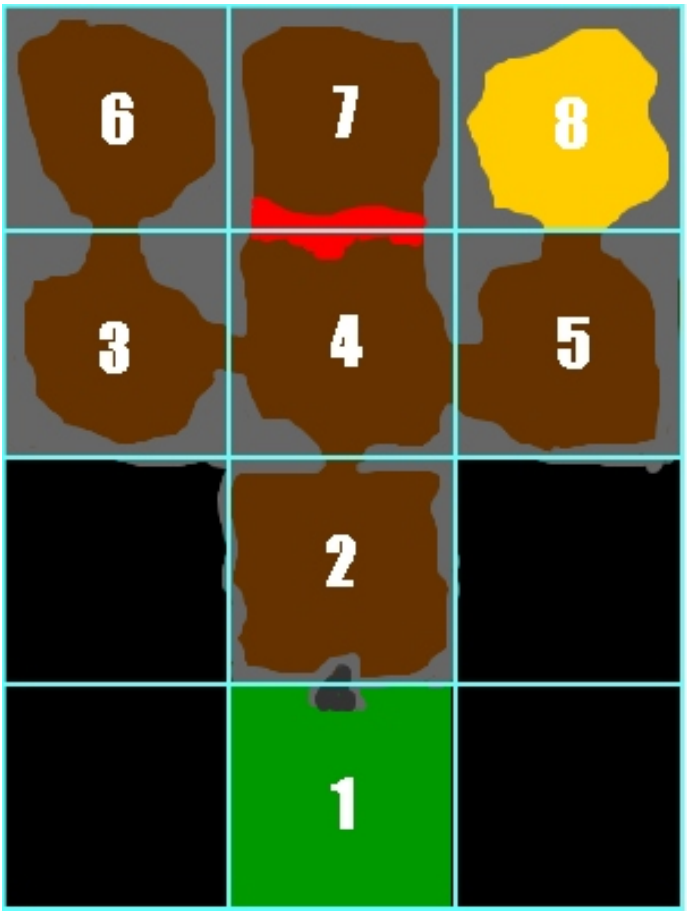

Figure 1: Map

The numbers on the map indicate the rooms. Here is a description of what is in each room.

1. **Outside cave**. Metal bar here, bending bar on rock in room 4 turns bar into hook.
2. **Cave entrance**. Chunk of metal here. Metal one of 3 components needed for fire.
3. **Left Side corridor**. Rope here. Rope, when combined with hook and tied to rock allows player to cross chasm.
4. **Main chamber**. Rock here used to make hook and to tie rope to. Chasm here needs rope/hook to cross.
5. **Right side corridor**. Flint is one of 3 components needed for fire.
6. **Moss room**. Can grab chunk of moss here. Moss is one of three fire components.
7. **Sword area**. Sword is frozen in block of ice. Need to build fire to get it out. Fire is a combination of moss, flint and metal.
8. **Dragon**. If have sword player wins game, otherwise is toast!

The Message Box

One nice feature about Flash is it's support of dynamic text through the textfield object. Dynamic text is simply a block of text in which the text can be changed. To change the text, you simply have to have a line of Action Script like the following

```
textfield.text = "New text";
```

While on the surface having dynamic text may not seem that big of deal, it has a lot of uses. The biggest use would be to have a generic movie symbol in which the text can be changed programmatically. This can be very helpful if the text that is to be displayed will be based on a number of conditional factors. Perhaps the best example of such a situation would be almost any game which gives the player a score.

In the Dragon and the Sword game, it is important to give the user feedback. My first thought was to have a message bar on the screen, perhaps just above the inventory bar. The problem with this idea is that messages would be very easy to ignore. In addition, the relevancy and the timeliness of the message could be jeopardized by using this technique.

A better technique would be to have a message box that pops up with a message for the user. The user would then have to close the box thereby insuring that the user has seen the message. This does not, however, insure that the user has actually read the message, but that's a minor point.

Figure 2 The Message Box

The popup is setup as a movie object. It consists of a decorative background with three components. The caption is a bold dynamic text (textfield) symbol named "caption". The message is a larger multi-lined textfield labelled "message". Finally, the ok button is simply your typical button and is labelled "ok_btn".

Dialog Box Scripts

In order to make use of the message box, the popup movie should be placed on the popup layer in the center of the screen. Because the popup layer is the top layer, it's contents will be displayed above everything else in the game. In order to control the popup, two functions will be needed.

The first function causes the popup to be displayed with the indicated caption and message text. Quite simply, this function simply takes the passed variables and assigns them to the movie's dynamic text variables. It then makes sure that the popup's ok button will call the hidePopup method. Finally, it set's the movie to visible so that the popup will be displayed.

```
function showPopup(sCap, sMes)
{
      popup.caption.text = sCap;
      popup.message.text = sMes;
      popup.ok_btn.onRelease = hidePopup;
      popup._visible = true;
}
```

To hide the popup again, we have a simple function that hides the popup. You will notice that there is a commented out line that calls the updateInventory() function. This is a placeholder for when the function exists. This will insure that modifications to the inventory bar will be processed.

```
function hidePopup()
{
      popup._visible = false;
//    updateInventory();
}
```

Finally, we need to make sure that the popup is initially invisible, so we add a call to hidePopup() to the restartGame() function.

At this point we have the framework for our Dragon and the Sword game.