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

Chapter 27

Adventure 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.

Chapter 21 Adventure Games

What is an Adventure Game
By an adventure game, I am referring to the classic computer gaming definition. The classic adventure games were simply puzzle based stories. Traditional adventure games have largely vanished from North America, though more arcade like adventure games have started to become more common. Likewise, there are a few good adventure games that do get released each year. What I would like to see more of, and with my upcoming One of those Weeks adventure games, is to see a game that is much more story based.

What is a RPG?
RPG stands for Role Playing Game. Like adventure games, RPGs are story based. Many even have puzzle elements like adventure games. The key things that distinguish RPGs from Adventure Games are Character Development, Combat, and World Scope.

A Brief History of Adventure
In the early days of adventure games, the games were text only. Adventure games parsers and vocabularies slowly improved. As computers became more powerful, the graphical abilities of computers became more powerful. Some adventure games actually started to have pictures added to the game. Eventually, the parser was eventually removed by menu based commands and point and click controls.

Why There are so few new Adventure Games
Adventure games use to be a staple of the computer game industry, yet today there are few new adventure games being created. At least in North America. Other part's of the world still seem to have new adventure games being regularly created. Why is this the case? I have found three core reasons for this. More graphics lead to higher production values. Publishers focussed on pleasing hard-core fans instead of the vastly larger general audience. Finally, general public misunderstanding of what adventure games are.

Flash Ideal for Adventure Games
When you think about it you will probably come to the same conclusion that I have. Flash already is an adventure game engine. The goal of a game engine is to handle the game mechanics. With an adventure game, most of the mechanics are graphical or animation related, which Flash already does. The scripting of the game would have to be done anyway, so the only work that really needs to be done is the inventory system. This could be considered an engine, but the amount of scripting code will probably be a far more significant portion of the Action Script required for the game.

Chapter 22 Single versus Multiple Movies

Designing an Adventure Game
The adventure game is fairly similar to any other game that is designed. The one big difference is that the adventure game should have more initial planning as making drastic changes to the game, especially in later stages, can more easily break the game. There are three things that should be worked out before work on the game should begin. These are the story, the map, and the puzzles. While all three things can be considered separate entities, when it comes to adventure games they are actually closely intertwined.

The Most Common Puzzles
There are many types of puzzles within adventure games. Some types of puzzles seem to be used far more frequently than other puzzles. The three most common puzzles that are used are the obstacle, maze, and the fetch quest.

One of those Weeks
Now it is time to create an adventure game that has all of it's locations in a single movie. This game is actually not the complete adventure. Instead what I am doing is breaking the game up into a series of smaller mini-adventures. Now, obviously there is not enough time to cover the creation of the entire game in this book. For that reason we are only going to cover the creation of the first episode of this game. The first chapter of the game is actually a dream sequence. The player is in a maze of rooms and must find their way out. To build tension, if the player stays in the same location for too long, a creature will appear and...well, you get the picture.

Dragon and the Sword
This game was a quickly done prototype to demonstrate that adventure games could be done in Flash for one of my potential clients back when I was running Spelchan Software. The interface is overly simplified, and there isn't as much animation that could be done, but the time it took me to put this game together clearly showed the viability of Flash for creating adventure games. The basic story for Dragon and the Sword is fairly simple. A dragon has been terrorizing your village for the last few years. You have recently heard about a magic sword that is lethal to dragons while protecting the wielder from fire. If you were able to find this sword you would be able to rescue your village.

Chapter 23 One of those Weeks

Planning the adventure
Mazes are fairly easy to implement. This maze, however, is not a linear maze. With a linear maze, you can use graph paper to plan out the maze. This is obviously not the case with a non-linear maze. Instead, a different type of graph is used for creating a non-linear maze. This graph consists of the rooms drawn as boxes with arrowed lines showing how the rooms connect.

Building the rooms
The idea behind this episode of the game is that the player is dreaming that they are in a maze of rooms being chased by something. The rooms are connected in a non-linear fashion, but the rooms each have their own color. Thinking about this, all the rooms in the game will look the same except for their color. This means that all we have to do is create a single room and then duplicate it five times changing the colors in the duplicates! There, however, is an even better way of handling the creation of the rooms. Tinting! Those of you who read through the section on board games are probably already familiar with this technique (and maybe even getting sick of it).

Linking the rooms
Now comes the task of assembling and linking the rooms. I create a block of frames for each of the six rooms with each room being broken into three animated sequences. First is the Enter sequence. This sequence shows the room zooming into view. Next is the main room loop. Finally there is an exit sequence that shows the room shrinking into nothing. Every room will have it's own set of four buttons. I also use a global variable named nextRoom to hold the room information and have assigned a variable to each of the rooms which is set up in the initGame function. The enterRoom function takes the player to the desired room. Each room sequence also has a similar bit of code for handling the door buttons.

Enter the Nasty
To add a bit more pressure on the player, I have a villain that pops up after the player has stayed in the room too long. Being a programmer, the first villain that came to mind is the dreaded Blue Screen of Death that occurs on Windows machines. In addition to the BSoD image, we also need a movie that controls the BSoD. For control purposes we would have the functions for hiding and showing the BSoD and some code to handling the timing requirements of the BSoD appearance.

Losing the game
This leads to the problem of what to do when the player loses the game. My first thought was to have the BSoD claw the player to death. Having someone being clawed to death is not really appropriate for the BlazingGames.com website. Instead, I decided to have a bit of fun and generate a nice error message that should look vaguely familiar to Windows users.

Winning the Game
Now that it is possible to lose the game, perhaps it would be nice to allow the player to win the game. The winning scene was designed with two purposes in mind. First, as a distinct way of showing the player that they have completed the game. Second, to preview the next episode of the game.

An Introduction
One problem that I had with the game at this point was the fact that someone who started playing the game without reading the instructions would be totally confused by what was going on. While confusion may actually add to the story, it can also turn a lot of people off the game. As this is the first episode of a series, one thing I don't want is to turn people off the game. One solution would be to have instructions on the game's title screen but would people bother reading that version of the instructions? The solution is to incorporate the background portion of the instructions into the game by starting the player at an explanation screen.

The Title Screen
As is quite often the case with my games, the last part of the game that is created is the title screen.

Chapter 24 Rooms engine

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. 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. As you should already know, Flash allows you to have movies within movies. You can load movies on demand and the movie that is loaded can actually make function calls to the movie that loaded it.

The Blank Movie
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. 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.

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.

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. 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. We will write initGame and restartGame functions, even though neither function does very much at the moment. We will need a routine to load new rooms, which we will also write at this time. The fifth frame is labelled Loader. This frame contains a single line of code that loads the movie.

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.

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. With the story in hand, we can now work out the map and some puzzles for the player to solve.

The Message Box
One nice feature about Flash is it's support of dynamic text through the textfield object. In the Dragon and the Sword game, it is important to give the user feedback. To make sure the user is aware of the message 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.

Dialog Box Scripts
To actually make the Dialog box appear and disappear requires a bit of code. The showPopup function will cause the Dialog Box to be shown while the hidePopup function will hide it.

Chapter 25 Dragon and the Sword

Room # 1
The first room in the game is outside the cave. This is the location where the game starts. In the game the metal bar will be here. The bar is used later in the game. By bending bar on rock in room 4, the bar turns into a hook. The hook can be used in conjunction with the rope to cross the chasm. To enter the cave, we need a button that the player can click on. This button would be the shape of the cave entrance. The code to actually make the button do something is placed in the second frame.

Starting the Game
Now that we are at the point where we are going to need to keep track of inventory items, we are going to have to initialize the game better by adding some constants for our bar object and making sure the game knows that the player does not have the bar when the game starts.

The Bar and the Hook
Our first inventory object that we have to worry about is the bar. As it worked out, the bar is not a typical inventory object. The bar is special because it can be bent on the rock in room 4 to form a hook. A bit of code for handling the bar that is in the room1 movie. Within the main movie, we are going to have to have a method for picking up the bar called getBar.

Updating the Inventory
The first thing we have to do to get the inventory to work is create a backdrop for the inventory bar. At this point in time, we only have two inventory items, or more accuratly, two states of a single object. Seeing that the objects are essentially the same object, we place both objects in the same location on the inventory bar. While this may seem strange, remember that at the most, only one of the two buttons will ever be visible at any given time. To make sure the inventory bar displays correctly, we write an updateInventory function. We are going to write a placeholder function for dealing with the inventory bar being clicked. This function will be re-written when we get to the room where the bar is manipulated.

Room # 2
The second room is the cave entrance. This leads to the main chamber or back outside. While at the moment we are focussing on the scene, in the game a chunk of metal will be found here. The metal will be used in conjunction with the moss and the flint to create a fire. The goal behind this scene was to have the entrance seem fairly long and leading somewhere. The buttons needed to implement this room consist of two exit buttons, the metal item on the cave floor, and the inventory bar version of this chunk of metal. Information about the metal has to be added to the restartGame function and the updateInventory function and new functions getMetal and startFire have to be written.

Room # 3
This room is the left side corridor and is used to act as buffer between the main room (room 4) and the moss room (room 6). The rope will be located here. Rope, when combined with hook and tied to rock allows player to cross chasm in the main room. The rope object is a bit more complex than most the other objects as it actually alters the state of room 4. For that reason, within initGame we define a set of new variables that hold state values. The restartGame and updateInventory functions need updating as well as the functions getRope and useRope need to be written.

Room # 4
Room 4 is the main chamber and could be considered the hub of the game. There are four directions the player can move to from here. They can go back to the cave entrance, into the left corridor, into the right corridor, or after solving a puzzle cross the chasm. There is going to be a rock in this area. The bar can be bent on this rock, and the rope can be tied to this rock. The screen shot below shows this room.

Special Scripting
The bar needs to be able to transform into the hook and then the hook has to be able to be tied to the rope. Final versions for useBar and useRope need to be written.

Chapter 26 Finishing the Dragon

Room # 5
This is the right side chamber which leads to the dragon's chamber. There are two exits from this room. One leads to the main chamber (room 4) while the other one leads to the dragon chamber (room 8). Located in this room is the flint. Flint, when used with the steel and the moss is used to melt the ice surrounding the sword. Also added to this room is a sign warning the player about the dragon. This was to make sure that the player had ample warning not to enter the final room. The restartGame and updateInventory are updated and a new getFlint function is written.

Room # 6
Room 6 is a side chamber and as that only has a single exit, which leads to the rope room (room 3). This room is vital to the game as it has moss growing on the wall. This moss can be used with the flint and steel to start a fire, which is required to get the sword. This room is easier to code because I felt it was acceptable to allow for an infinite supply of moss. The restartGame and updateInventory functions have additions made to them as well as getMoss and useMoss funtions are created.

Room # 7
Across the chasm is the sword that the player needs in order to win the game. The sword is frozen in a block of ice. The room is only reachable after the player has tied the rope to the rock and the hook and threw it across the chasm. This means that the only exit is a rope that crosses the chasm. The sword, being frozen in a block of ice, needs to be melted before the player can acquire it. This is done by having the player place moss around the block of ice and then start a fire using the flint and steel. Likewise, some new variables for handling the inventory need to be added to the main movie. The restartGame and updateInventory functions are modified and new getSword and useSword functions are written.

Starting a Fire
In order to get the sword, the player needs to start a fire. While we could have this done automatically if the player is in the right room and has all the items, I am requiring that the player first lay down the moss and then attempt to start the fire. We need to replace the useMoss function with an enhanced version. The useFire function is also going to have to be replaced.

Room 8 Win Movie
Room 8 is the final room of the game. This is the location where the dragon is. As such, only two things can happen with this room. The player can win the game or they can lose the game. This is decided by one simple factor. Does the player have the sword? If they do they win. If they don't they lose. The code to handle this decision is in room 8's first frame and is simply a comparison statement. Now all that needs to be done is the winning animation.

Room 8 Lose Movie
We have the winning movie finished, but we need something for the losing movie. This means that we are going to have to bite the bullet and create a dragon. Thankfully, we don't really need to animate the dragon, we just need the dragon in a pose that suggests it is shooting a fireball at the player. We then need a flame object. The flame object expands from the dragon's mouth until it fills the screen. At which point a lose message is displayed.

Title Screen
We now have a fully playable game, so it is time to finish things by creating the title screen. I wanted to have some type of animated title sequence, featuring the dragon breathing the logo at the player. That didn't quite work, so instead, I had the dragon shoot a fireball which fades into the title sequence. Finally, the start game button appears.

Project

Obviously, you should now have a good enough understanding to create your own adventure game. If you want a simpler project to begin with I would suggest playing around with Dragon and the Sword and creating an all in one movie version of the game. Lets briefly take a look at what would be involved in converting the already existing Dragon and the Sword game into a single movie file.

The title sequence for both versions of the movie is the same, so the only work that would have to be done here is copying the movies over to the new movie is simply a matter of copying. In fact, the original prototype that I developed never had a title sequence. I added that to the existing version out of a sake of completeness. Also of interest is the fact that the title sequence in the single movie version is different from the multi-movie version we just developed.

Why is it different? Well, when I decided to release a version of Dragon and the Sword on my web site, I realized that I needed a title sequence. As is usually the case, the title sequence is the last thing that gets created. Because I had the first room as a symbol, I was able to simply use a small version of the first room as the starting point for creating the title sequence. Obviously it doesn't make sense to have a duplicate version of the first movie in the multi-movie version, so instead, the multi-movie version has original artwork designed to look like the first movie.

Now that we have the title, we are ready to start creating the actual game. The movie will have five layers. These are "code", "winlose", "Popup", "inventory", and "roommovie". Four of the five layers are the same as in the multiple move version of the game. In fact, the multiple move version only had four layers. The winlose layer is something new.

In the multi-movie version, every room was it's own movie. This isn't the case with the single-movie version. If you recall, room 8, which is the winning or losing room, has an extra layer to handle animation. As we are going to need to do this animation, a separate layer is needed.

Room movies would be ported over to the main movie to form labelled sequences of the movie. The movement function would have to reflect this. The scripting would also have to be changed and some variables renamed so that all variables have original names. When I ported the movies over, I just used an extended variable labelling system that added the room number to the variables and buttons.