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

Chapter 25

Dragon and the Sword

Contents

With the concepts of creating a movie that uses external movies understood, it is time to actually create a game using this technique.

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.

As this is outside, we need a view of the mountains. I want to see the sky as that is a good way of indicating to the player that they are outside. Likewise, the grass is useful for providing that information.

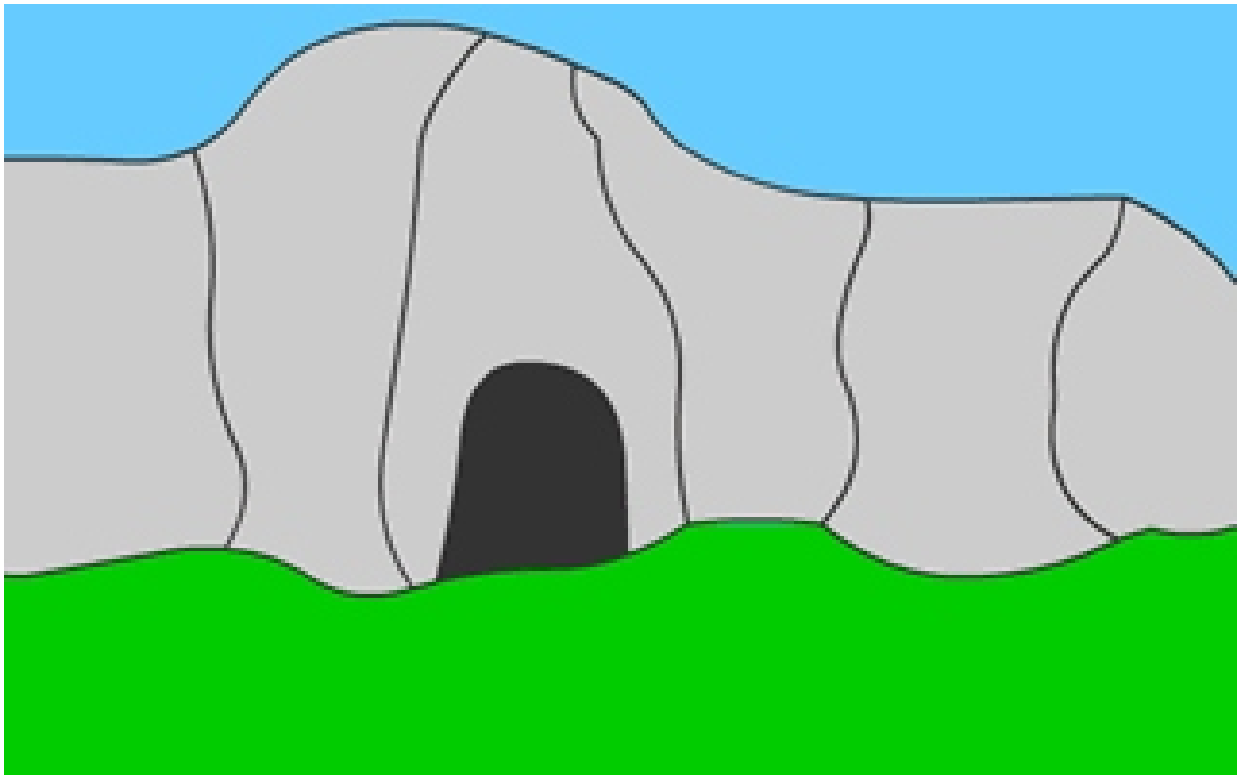

Figure 1 Room 1

In this first room of the adventure, the player is outside of the cave. At this point the player's only movement option is to enter the cave. Lying on the ground near the entrance of the cave is a bar. The player can pick up this bar, which will be used later in the game to make a hook.

To enter the cave, we need a button that the player can click on. This button would be the shape of the cave entrance. In fact, if the cave entrance is a different color from the surroundings, which it is, you can simply select the shape that makes up the cave. Converting this shape into a button simple is simply a matter of right clicking on the shape and choosing the "convert to symbol..."

menu option.

We want the player to know that the mouse is over something special. One way of doing this would be to do some type of highlighting. Another way, which also happens to provide the player with more information about what they are selecting, is to provide a message. The work to do either of these things is very minuscule as Flash's button already has the necessary functionality.

Remember that buttons in flash have three visible frames. The first frame is the normal appearance. The second frame is the mouse over appearance. The final frame is the mouse down appearance. Here are the three frames that were used in this button:

Figure 2 Cave buttons

The code to actually make the button do something is placed in the second frame, and is as follows:

```
cave_btn.onRelease = function()
{
      trace ("Room 1 cave button clicked");
      _parent.changeRoom(2);
}
```

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. There are two types of initialization that have to take place. The first type of initialization is the actual program initialization, which should only happen once. It is contained in a fuction called initGame(). This function simply sets up any constants that are used in the game. At the moment, the function is mostly empty (remember that we created it last chapter). Now we are going to add some constants to the initialization. By constant, I mean a variable who's value shouldn't change. Flash, however, doesn't have any keyword that enforces the constant state. At the moment the only constants that are needed are for the states of the bar object. More constants will be added to the initGame function later, however.

```
	// set up my "constant" variables
	BAR_NONE = 0;
	BAR_HAVE = 1;
	BAR_HOOK = 2;
	BAR_USED = 3;
```

The second function that is needed actually initializes the game to it's starting state. This function should run every time the game is restarted. Quite simply, it takes all the variables in the game and sets them to their starting state. At the moment, we only need to worry about adding the bar status. Other variables will be added when we come to them.

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

	inv_bar = BAR_NONE;

	hidePopup();
	updateInventory();
}
```

You will notice that I am prefacing all the inventory state variables with the inv_ preface. This helps me remember that the variable is used for tracking the state of an inventory item. In addition to setting the state of the bar inventory item, the inventory bar button for handling that item is also assigned a function to call when the button has been released. We will be writing that method shortly.

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.

To start with, we create a bar image. While we are at it, we might as well create the hook image at the same time. The bar image will need to be used in a button that will be used in the first room movie. Sadly, using the bar image in both movies results in duplication of symbols, though that could be avoided by using a common library and have both the main movie and the room movie reference that library. This project is too small to worry about such things, however.

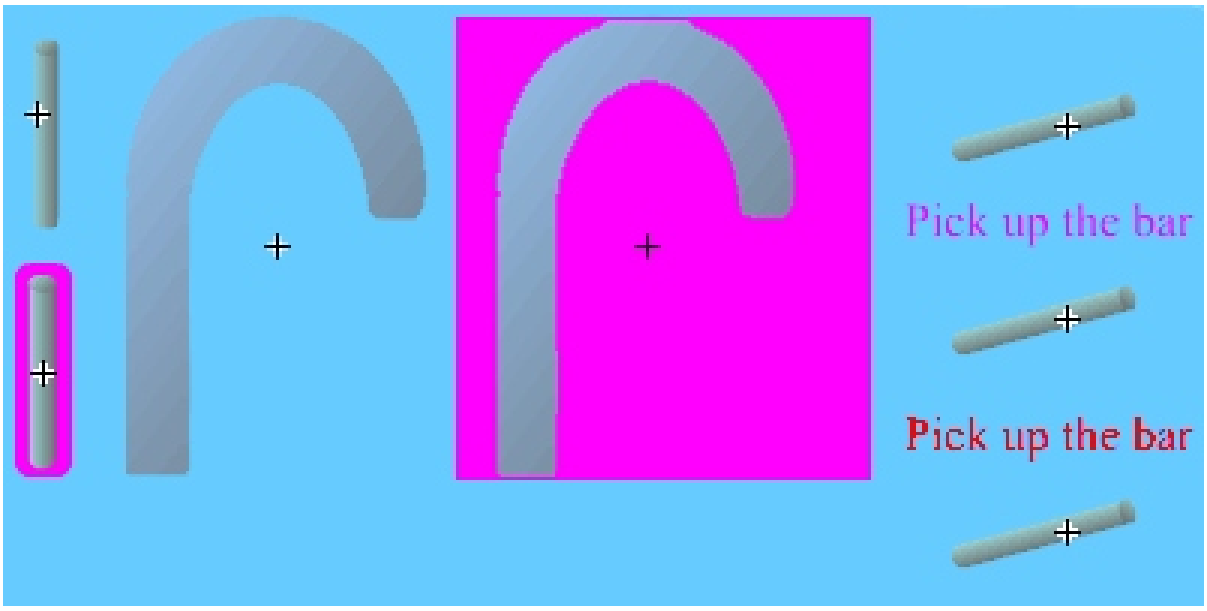

Figure 3 Bar and Hook

Here is the code for handling the bar that is in the room1 movie. Like usual, I place the code on frame 2 of the code layer. You should notice that the code takes advantage of the parent movie, using the parent as a placeholder for global game information. The movie simply looks at the state of the inv_bar variable to see if the player has picked up the bar or not. If the bar is not in the inventory then it must be in this location so it is shown.

```
bar_btn.onRelease = _parent.getBar;

if (_parent.inv_bar == _parent.BAR_NONE)
      bar_btn._visible = true;
else
      bar_btn._visible = false;
stop();
```

Within the main movie, we are going to have to have a method for picking up the bar. This simply updates the bar state then updates the main inventory bar.

```
function getBar()
{
      trace("Picking up bar");
      inv_bar = BAR_HAVE;
      updateInventory();
}
```

Updating the Inventory

The first thing we have to do to get the inventory to work is create a backdrop for the inventory bar. As this is a simple game, we are simply going to go with a solid color. I have chosen a light brown. If you wanted to get fancy, you could certainly have some type of texture created. If your inventory is limited and you don't mind if players know what items are in the game you could even create a background that looks like wood or stone with the basic outline of the objects in the game carved into the wood or stone.

At this point in time, we only have two inventory items, or more accurately, 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.

Now we need to actually get the inventory to be updated. The updateInventory will handle this problem. As the only inventory item we are dealing with now is the bar, we will only have code for dealing with the bar. This, however, is more complicated code than other inventory items due to the fact that the bar has more than one state. We first have to see if the bar is in the inventory. If it is, we decide if we should be showing the bar or showing the hook.

```
function updateInventory()
{
	if  ( (inv_bar == BAR_NONE) || (inv_bar == BAR_USED) )
	{
		invBar_btn._visible = false;
		invHook_btn._visible = false;
	}
	else if (inv_bar == BAR_HAVE)
	{
		invBar_btn._visible = true;
		invHook_btn._visible = false;
	}
	else
	{
		invBar_btn._visible = false;
		invHook_btn._visible = true;
	}

	room_movie.gotoAndPlay(1);
}
```

One interesting piece of code is the last line of the function. This tells the room_movie object (which will be whatever movie has been loaded) to restart. The reason for doing this is to make sure that the movie will update the state of any buttons it is using.

Now, 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. Right now we will just default to the "Not usable at this time" message.

```
function useBar()
{
	if (inv_bar == BAR_HAVE)
	{
		showPopup("Use Bar", "You see nothing to use the bar on here!");
	}
}
```

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.

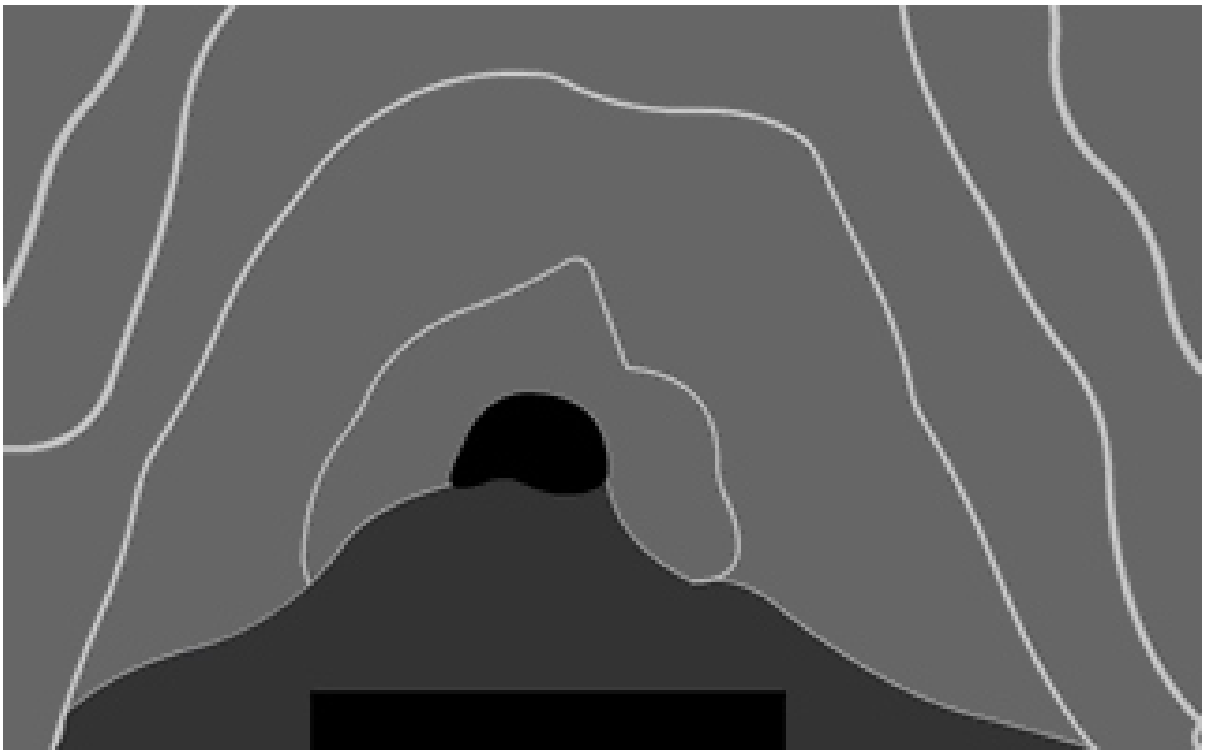

Figure 4 Room 2

Room 2 is the cave entrance. It has two exits, the first leading back outside (to room 1) and the second leading to the main chamber of the cave (room 4). In addition, there is a piece of metal lying on the cave floor that the player can pick up. When this metal is used with the flint and with moss it can be used to light a fire that will unthaw the sword.

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. Figure 11-3 shows the buttons used for this room.

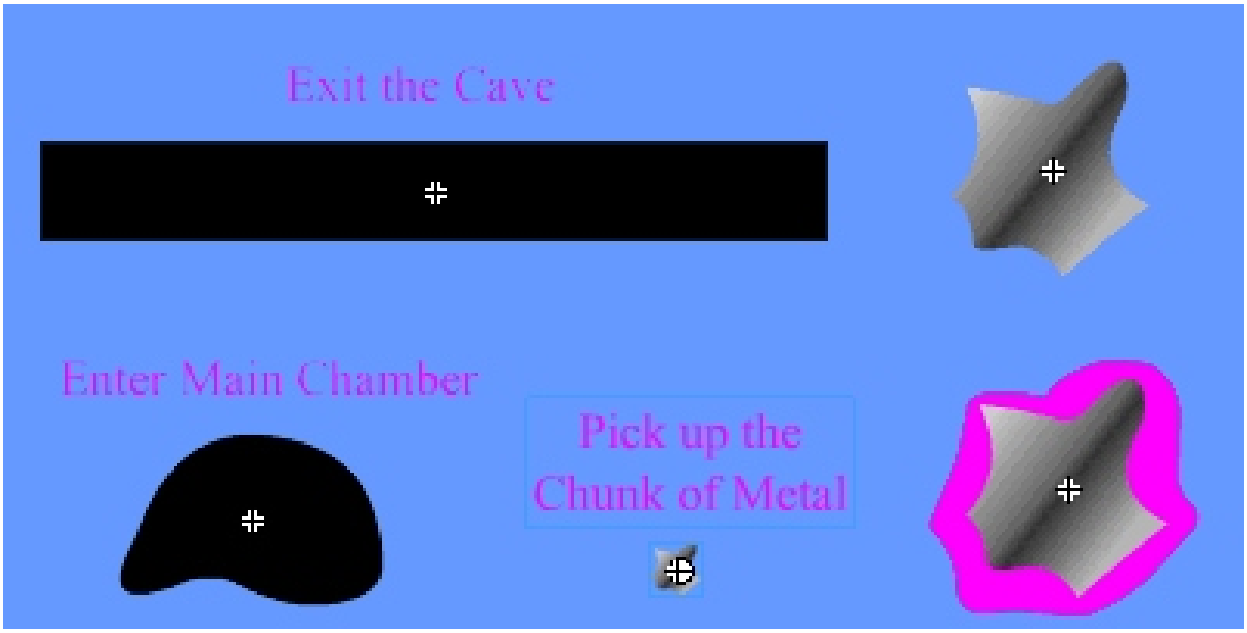

Figure 5 Room 2 exits and metal chunk

Within the room, the following code is needed to activate the buttons.

```
toMain_btn.onRelease = function() {
      trace ("toMain button clicked");
      _parent.changeRoom(4);
}

exit_btn.onRelease = function() {
      trace ("exit button clicked");
      _parent.changeRoom(1);
}

metal_btn.onRelease = function() {
      _parent.getMetal();
}

if (_parent.inv_metal == false)
      metal_btn._visible = true;
else
      metal_btn._visible = false;

stop();
```

The following lines need to be added to the main movie's restartGame function.

```
      inv_metal = false;
      invMetal_btn.onRelease = useFire;
```

Likewise, the main movie's updateInventory needs some lines added.

```
      invMetal_btn._visible = inv_metal;
```

Finally, the method for handling the acquisition of the metal and the placeholder for the useFire function needs to be written.

```
function getMetal()
{
      trace("Picking up chunk of metal");
      inv_metal = true;
      updateInventory();
}
```

```
function useFire()
{
      showPopup("Use Flint and Steel", "You are missing something!");
}
```

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.

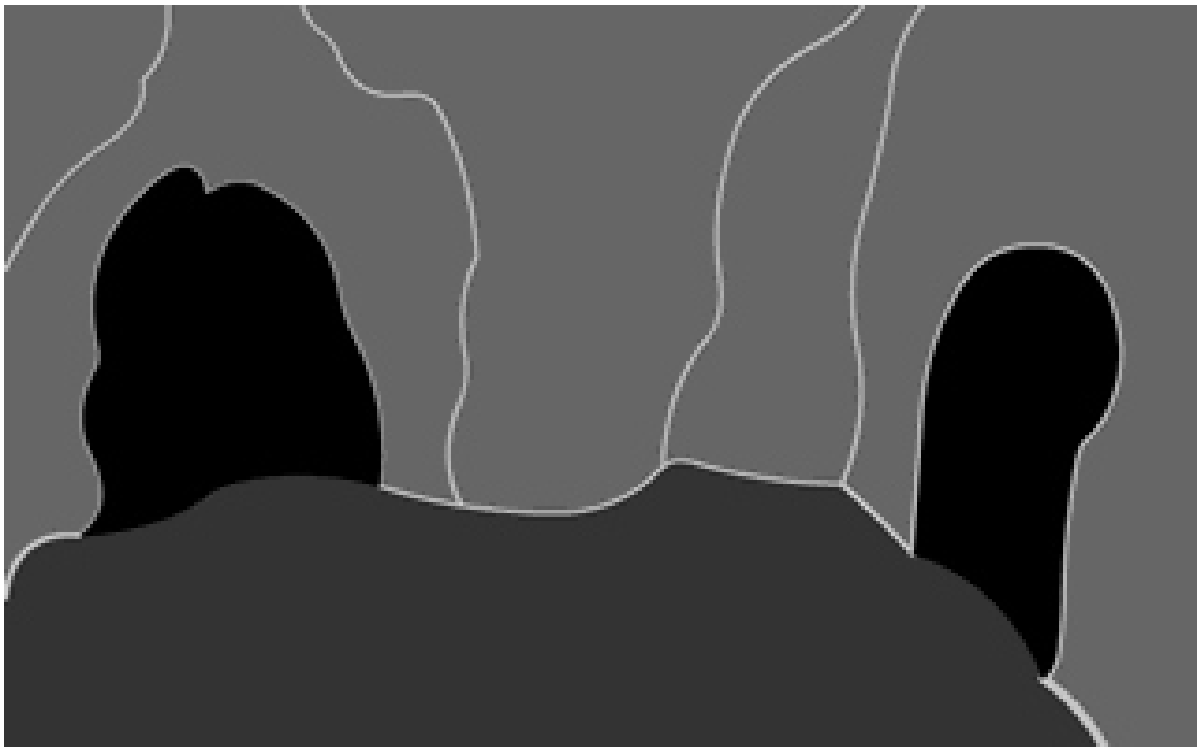

Figure 6 Room #3

This room is a side connector room. This room has two exits, the first leading back to the main chamber (room 4) and the second leading to the moss room (room 6). The rope is contained in this room. The rope is used with the hook in order to cross the chasm.

Quite simply, Three buttons will be needed for this room. The two exits and the rope button. A second rope button will be needed for the inventory. Figure 7 shows the buttons in their over states.

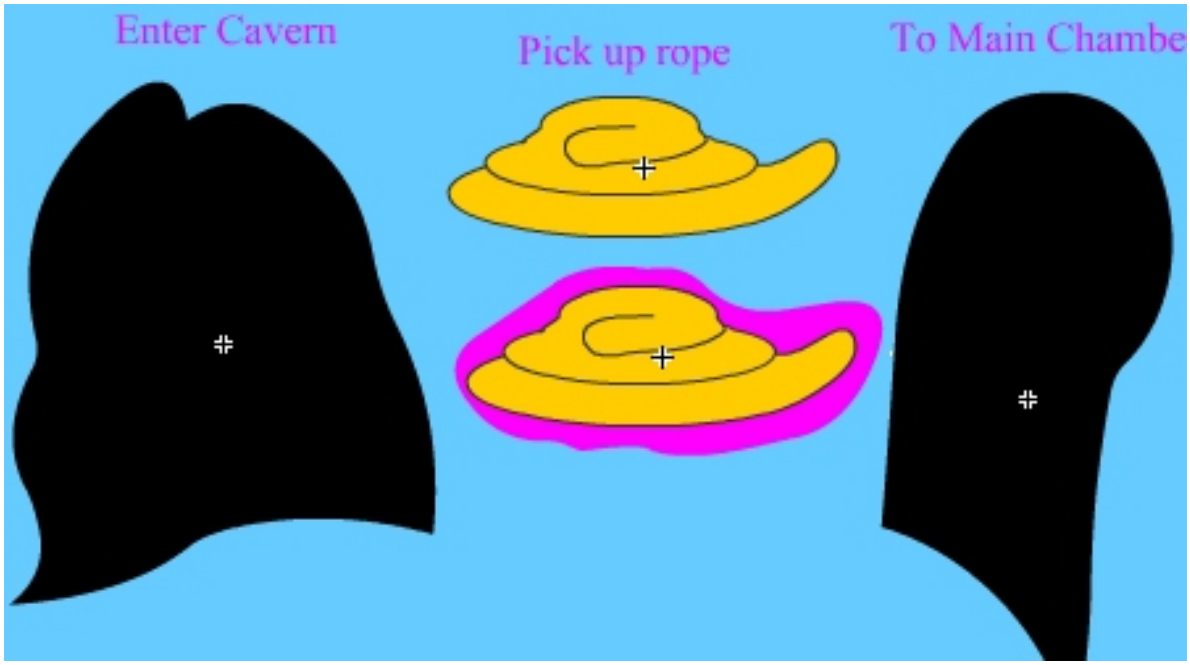

Figure 7 Exits and Rope

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 following code is therefore added to the initGame function

```
	ROPE_NONE = 0;
	ROPE_HAVE = 1;
	ROPE_USED = 2;
```

Within the restartGame function, the following lines need to be added.

```
	inv_rope = ROPE_NONE;
	invRope_btn.onRelease = useRope;
```

And obviously we are going to have to update the updateInventory function. Notice that the rope only appears on the inventory bar if we have it, not if it is a greater state than ROPE_NONE. This is because when the rope has been used on the rock it is no longer available.

```
	if (inv_rope == ROPE_HAVE)
		invRope_btn._visible = true;
```

```
      else
            invRope_btn._visible = false;
```

We need to add the code to handle room 3. This of course goes in frame 2 of the room3 movie.

```
to6_btn.onRelease = function() {
      _parent.changeRoom(6);
}

to4_btn.onRelease = function() {
      _parent.changeRoom(4);
}

rope_btn.onRelease = _parent.getRope;

if (_parent.inv_rope == _parent.ROPE_NONE)
      rope_btn._visible = true;
else
      rope_btn._visible = false;

stop();
```

As you should have noticed, we need a function for getting the rope, and a placeholder function for using the rope.

```
function getRope()
{
      trace("Picking up rope");
      inv_rope = ROPE_HAVE;
      updateInventory();
}

function useRope()
{
      showPopup("Use Rope", "There is no reason to use the rope here!");
}
```

The problem with scripting the rooms in order is that not all the rooms are linked in order. In fact, to get to room 3 you need to go through room 4 which has yet to be scripted. This problem is easy to handle, as all that needs to be done is to make some minor changes to the restartGame function. Just change the line

```
nextRoom = "rooms/Room1.swf";
```

to

```
nextRoom = "rooms/Room3.swf";
```

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.

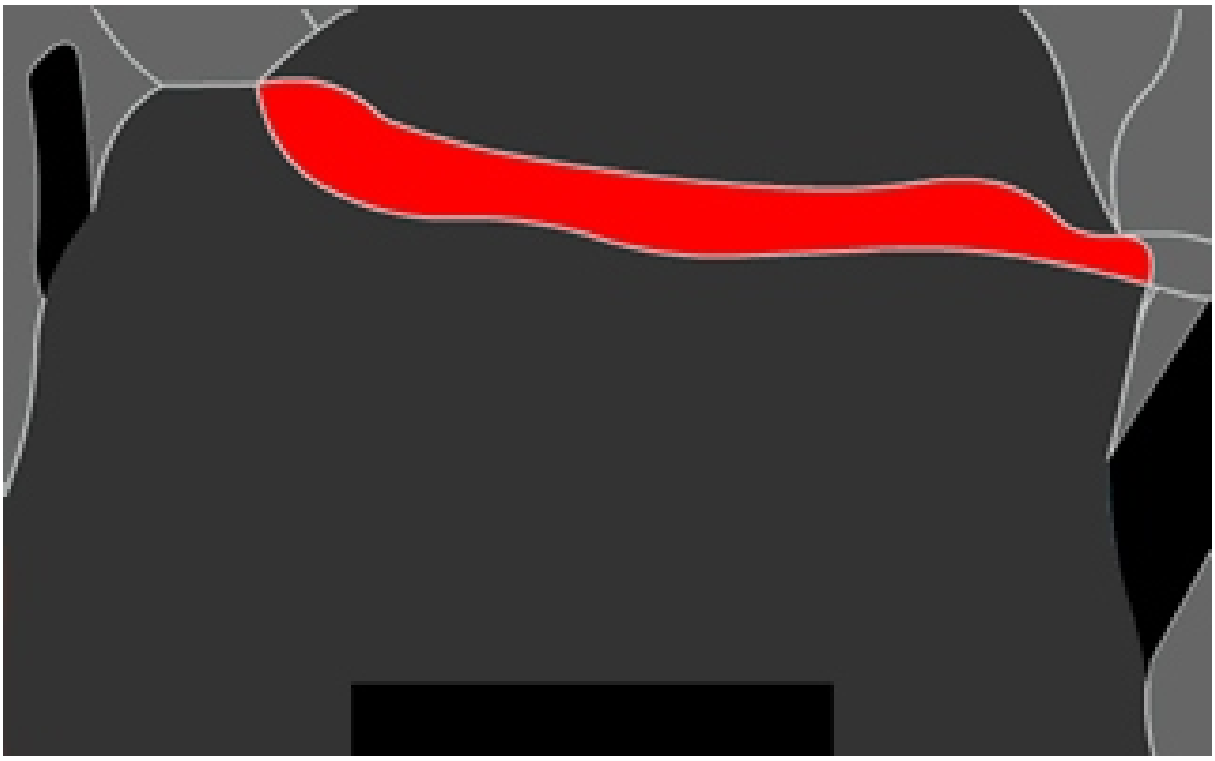

Figure 8 Room #4 - The main chamber

This is the main room in the game. This room leads back to the entrance, two two different side chambers, and accross the chasm is another chamber. This is the room where the first major puzzle of the game needs to be solved. Simply put, the puzzle is how do you cross the chasm.

A large rock (which is a separate movie so not seen in the screen shot) is near the edge of the chasm. This rock is vital to solving the game. There are three things that need to be done with the rock. First, the player needs to bend the bar on the rock. This transforms the rock into a hook. Next the player needs to tie a rope to the rock. Finally the player needs to tie the hook to the rope and throw it across the chasm.

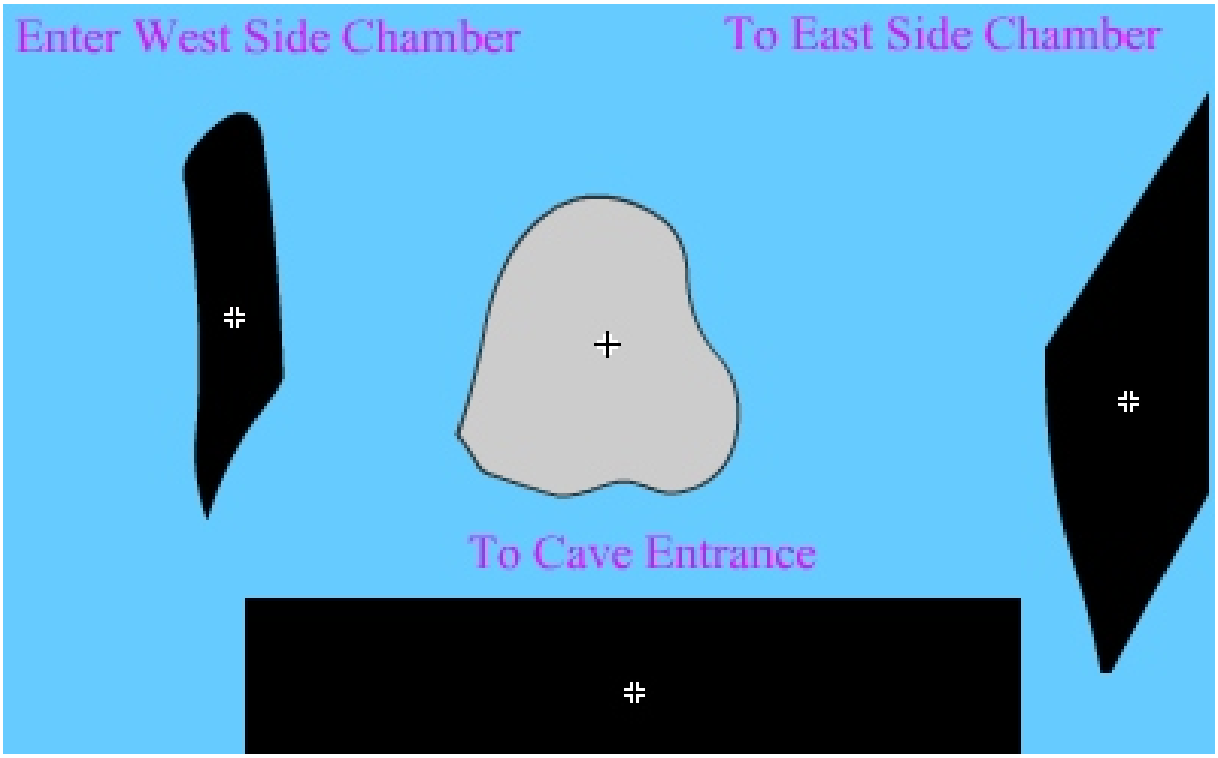

Figure 9 Buttons and basic rock movie

The code for handling the room buttons is simple enough. As usual this code goes in the second frame of the room 4 movie.

```
to2_btn.onRelease = function() {
      _parent.changeRoom(2);
}

to3_btn.onRelease = function() {
      _parent.changeRoom(3);
}

to5_btn.onRelease = function() {
      _parent.changeRoom(5);
}
stop();
```

Now, before we can start actually working on the scripting, we are going to have to make a few changes to our main code to make some of the scripting work easier. In particular, many of the inventory item actions are room specific. There are at least two ways that this problem can be solved.

One route would be for the inventory handlers to call the room movie telling the movie that an item was used. This can be accomplished by having each movie have a function (possibly called useItem(id)) that takes some type of item id variable. The movie would then either do something or return a value that indicated that it doesn't know what to do with the item. This has the added benefit of making the rooms more independent of the main movie.

The other route, which for a small game such as this is easier and faster, is to track the room numbers and have the main movie inventory handlers just use the room that the player is currently in to decide what should be done. This has the added advantage of making the game easier to convert to a single movie version, which is something we are going to discus in a later chapter.

To get the room numbers, we simply have to make a minor addition to the changeRoom() function by adding this line to the beginning.

```
currentRoom = n;
```

and to make sure the room initializes properly, to the restartGame function, add this line.

```
currentRoom = 1;
```

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. Currently the game is designed to automatically use the inventory items for what they are suppose to be used for if the player is in the proper room. A more complex adventure game may actually use a dragging type interface where the player would have to drag the item they want to use over the object they want to use the item on.

Essentially, the useBar code makes sure the player is in room 4. If they are, then it will allow the bar to become a hook (by simply changing the bar's state variable). If the bar is a hook, it sees if the rope has been tied to the rock. If the rope has then The bar's state changes to the used state which will tell room 4 that there is a rope crossing the chasm. Here is the new useBar function.

```
function useBar()
{
	if (inv_bar == BAR_HAVE)
	{
		if (currentRoom == 4)
		{
			showPopup("Bend Bar", "You use the rock to bend the bar and
turn it into a hook");
			inv_bar = BAR_HOOK;
		}
		else
		{

			showPopup("Use Bar", "You see nothing to use the bar on
here!");
		}
	}
	else if (inv_bar == BAR_HOOK)
	{
		if (currentRoom == 4)
		{
			if (inv_rope == ROPE_USED)
			{
				showPopup("Use Hook", "You tie the hook to the rope
and throw it across the chasm");
				inv_bar = BAR_USED;
			}
			else
			{
				showPopup("Use Hook", "You need something to attach
the hook to");
			}
		}
		else
		{
			showPopup("Use Hook", "You see nothing to use the hook on
here!");
		}
	}
```

```
}
```

The astute reader will have noticed that hook actions are implemented in this method, but right now there is no release function assigned to the inventory's bar or hook button. This is easily fixed by adding the following lines to the updateInventory function.

```
      invBar_btn.onRelease = useBar;
      invHook_btn.onRelease = useBar;
```

The rope has to know that it can be tied to a rock. This is done by replacing the useRope function with this enchanced version.

```
function useRope()
{
      if (currentRoom == 4)
      {
            showPopup("Use Rope", "You tie the rope to the rock!\rYou still
need to find some way of hooking the rope to the other side, though.");
            inv_rope = ROPE_USED;
      }
      else
      {
            showPopup("Use Rope", "There is no reason to use the rope here!");
      }
}
```

Now, one problem that we have is that inventory item states are changed without calling the updateInventory function. Instead of having a large number of calls to updateInventory, we will cheat by placing a call to updateInventory in the hidePopup function (which conveniently, I already have placed there in a commented out form.

Finally we need to actually create some images for the rope and the rope around the rock. We create a rope button wich we place where the rock is. Remember that this will be invisible until the hook has been used with the rope. Next we create a one frame movie of the rock with a rope tied around it. This is placed right on top of the rock image, but it won't be visible until the rope has been tied to the rock.

We will need a bit of code for handling this, which is placed in frame 1of the room 4 movie. What's that? Frame 1? Yes. While I prefer placing things in frame 2 if we did that with these two hidden object, there would be a brief flicker which would give away the solution to the problem!

```
to7_btn.onRelease = function() {
      _parent.changeRoom(7);
}
to7_btn._visible = (_parent.inv_bar == _parent.BAR_USED);

ropedRock_movie._visible = (_parent.inv_rope == _parent.ROPE_USED);
```