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

Chapter 23

One of those Weeks

Contents

The first type of adventure engine we create is the all-in-one-movie engine. This is your typical Flash movie, with the entire game in a single movie.

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. I used a variation on this having colored lines corresponding to the room the line is from.

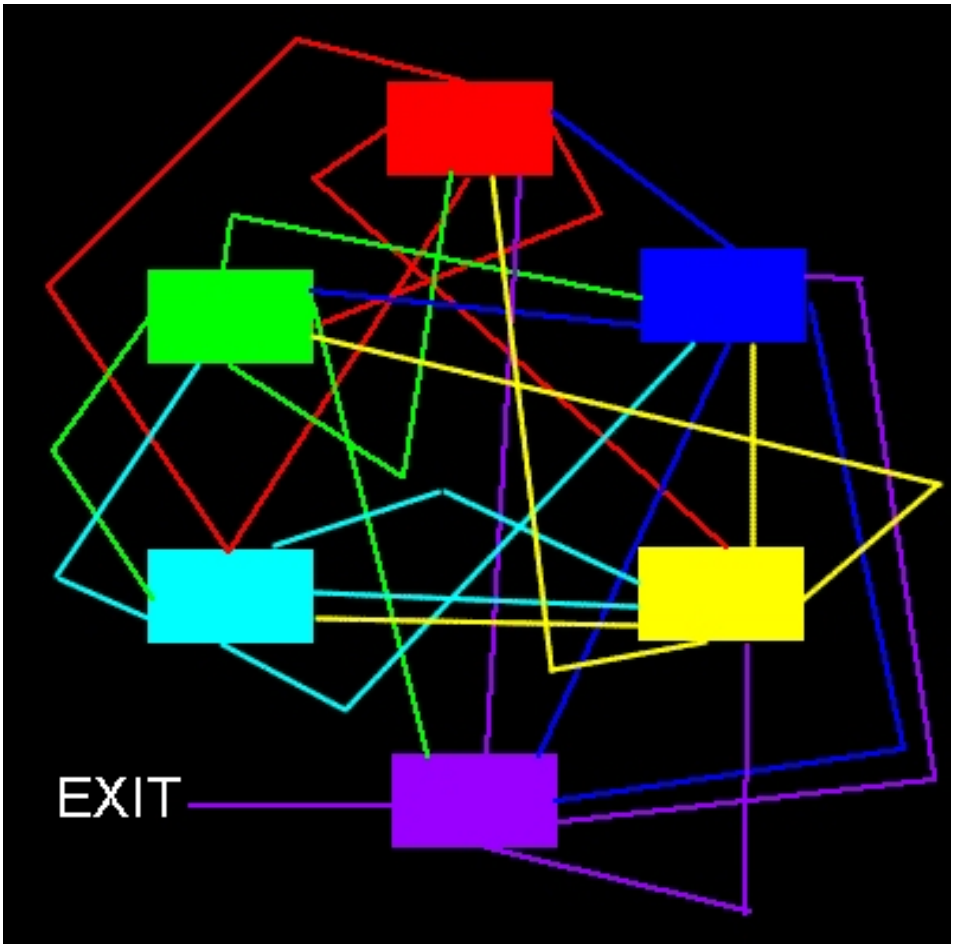

Figure 1: One of those Weeks Episode 1 Map

As you can see by the above map, the links between rooms are fairly complex. However implementing the links is actually very easy. You can also see that there are a lot of ways of reaching the exit, but do to the fact that there is nothing highlighting the exit, a player could end up being stuck in the maze for quite a while.

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).

To make tinting work the best, it is best to create the non-tinted image using grayscale. As with Dragon and the Sword, I am going to keep the game in third person perspective. This means that the room needs to be drawn to look three dimensional. To better add to the three dimensional look, I created a tiled floor. This was done by drawing receding lines, then by drawing lines in the distance and angling them so that there is a larger distance between the closer points than the further points. Had I been really ambitious I could have created a far off point (a vanishing point) and having all the horizontal lines coming from that point. I then filled in the squares and finally removed the lines.

The ceiling I wanted to be stuccoed. This would have required a huge amount of work to accomplish using vectors, so I opted to use a texture for this image. The texture was generated in fireworks, but all high-end paint programs (and many low end ones as well) have texture creation support.

The three doors that are in the scene are actually all the same. I drew the far door and then used the distort command to alter the other two doors. While I could have been picky and tried to get the doors to look perfect, to keep the dream motif, I wanted the doors to be a bit distorted.

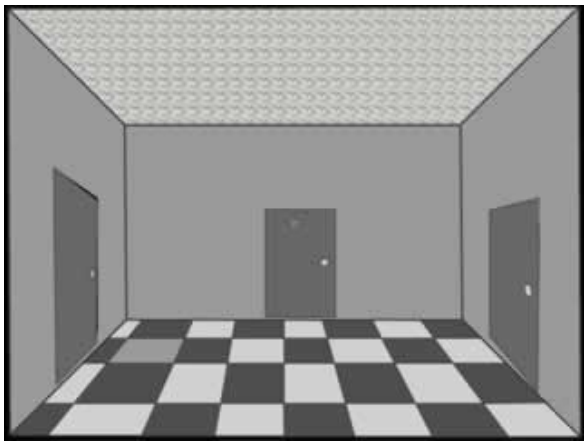

Figure 2: Grayscale room

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 (labelled enterColor, with Color being the color of the room). This sequence shows the room zooming into view. Next is the main room loop (labelled colorLoop, with color being the color of the room). I want to have an additional disorienting effect added to the room. Therefore, I set this block of frames to loop. The room will slowly grow and then shrink back to it's starting size. Finally there is an exit sequence (labelled colorExit, with color being the color of the room). This shows the room shrinking into nothing.

Now we need buttons for the four directions that the player can go. Quite simply, I will label the directions North, South, East and West. The four buttons will be invisible buttons. These are buttons that have a hit box but no image. Actually, when the player is over the buttons, we will have a "Go Direction" message appear. This is done simply by having the over and down frames of the button have the desired text in them.

Every room will have it's own set of four buttons. I use the convention of naming the button instances colorDirection_btn. I also use a global variable named nextRoom to hold the room information and have assigned a variable to each of the rooms. All of this is defined on the starting frame of the game as follows:

```
// startup code
initGame();

function initGame()
{
	if (ootwInitialized != undefined)
		return;

	ootwInitialized = true;
	REDROOM = 1;
	GREENROOM = 2;
	BLUEROOM = 3;
	PURPLEROOM = 4;
	CYANROOM = 5;
	YELLOWROOM = 6;
	BEDROOM = 7;
}
```

Next we create a function that will go to the desired room. This is your typical jump table. In other words, based on the id passed to the function, the movie's play head is adjusted to the desired frame to be played.

```
function enterRoom(id)
{
	switch (id)
	{
		case REDROOM :
			gotoAndPlay("enterRed");
			break;

		case GREENROOM :
			gotoAndPlay("enterGreen");
			break;

		case BLUEROOM :
			gotoAndPlay("enterBlue");
			break;

		case PURPLEROOM :
			gotoAndPlay("enterPurple");
			break;

		case CYANROOM :
			gotoAndPlay("enterCyan");
			break;

		case YELLOWROOM :
			gotoAndPlay("enterYellow");
			break;

		case BEDROOM :
			gotoAndPlay("enterBedroom");
			break;
	}
}
```

The buttons actions are set up on the last frame of the enterColor sequence of frames. This code is similar for each of the rooms, the difference bing the nextRoom variable is assigned the proper room label.

```
purpleNorth_btn.onRelease = function() {
      nextRoom = REDROOM;
      gotoAndPlay("exitPurple");
}

purpleEast_btn.onRelease = function() {
      nextRoom = BLUEROOM;
      gotoAndPlay("exitPurple");
}

purpleSouth_btn.onRelease = function() {
      nextRoom = YELLOWROOM;
      gotoAndPlay("exitPurple");
}

purpleWest_btn.onRelease = function() {
      nextRoom = BEDROOM;
      gotoAndPlay("exitPurple");
}
```

Finally, in the last frame of the exit we have the following line of code:

```
enterRoom(nextRoom);
```

Note that next room is not specified, but is instead the value that was assigned to the variable when a button was clicked.

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. Linux and Macintosh OS X don't have this problem (and are extremely stable operating systems) but most people have windows, and a lot of my programming happens to be for Windows machines.

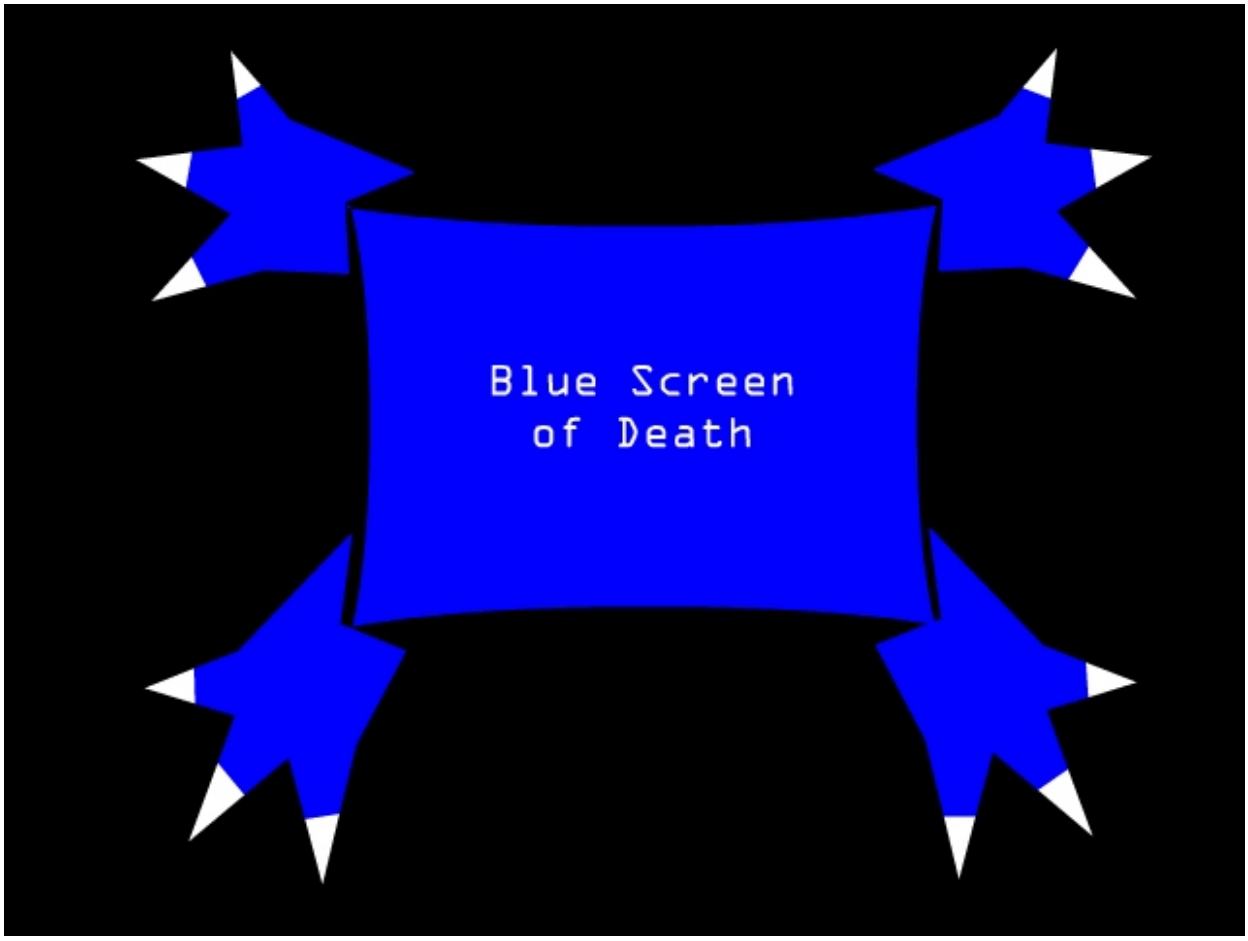

Figure 3: The Blue Screen of Death

In addition to the BSoD image, we also need a movie that controls the BSoD. This movie would consist of nothing for the first few frames (labelled noBsod) with a stop on the second frame. There would also be a block labelled bsodAppearing which would be a two second sequence of the BSoD appearing. For control purposes we would have the following functions:

```
function showBSOD()
{
	gotoAndPlay("bsodAppearing");
}

function hideBSOD()
{
	gotoAndPlay("noBsod");
}
```

As we already have a room loop, that loop can be used as a timer. This, of course, is going to require some variables be initialized. In the starting frame and in the enter room function, add the following lines of code

```
bsodTimer = 0;
bsodAppearing = false;
bsod_movie.hideBSOD();
```

Now at the end of each of the room loops we change the code to the following.

```
if (bsodTest() == false)
	gotoAndPlay("redLoop");
```

This calls the following function which controls the BSoD. Of course, we need to write this function, which we place in frame 2. The function simply keeps track of how many times it has been called. As the function is called every 2 seconds, we simply figure out how long we give the players before the BSoD appears and then start the appearing animation. If another tick happens after the appearing animation has occurred then we know that the player has bit the bullet.

```
function bsodTest()
{
	if (bsodAppearing)
	{
		gotoAndPlay("loseGame");
		return true;
	}
	else
	{
		++bsodTimer;
		if (bsodTimer == 5)
		{
			bsodAppearing = true;
			bsod_movie.showBSOD();
		}
		return false;
	}
}
```

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. This would reflect the feeling you get when you have an hours worth of unsaved work and the real blue screen of death shows up. Not that I've ever had that happen to me as I always save my work frequently. Mind you, the reason I always save my work frequently is because of learning what losing hours of work because of a computer crashing feels like.

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.

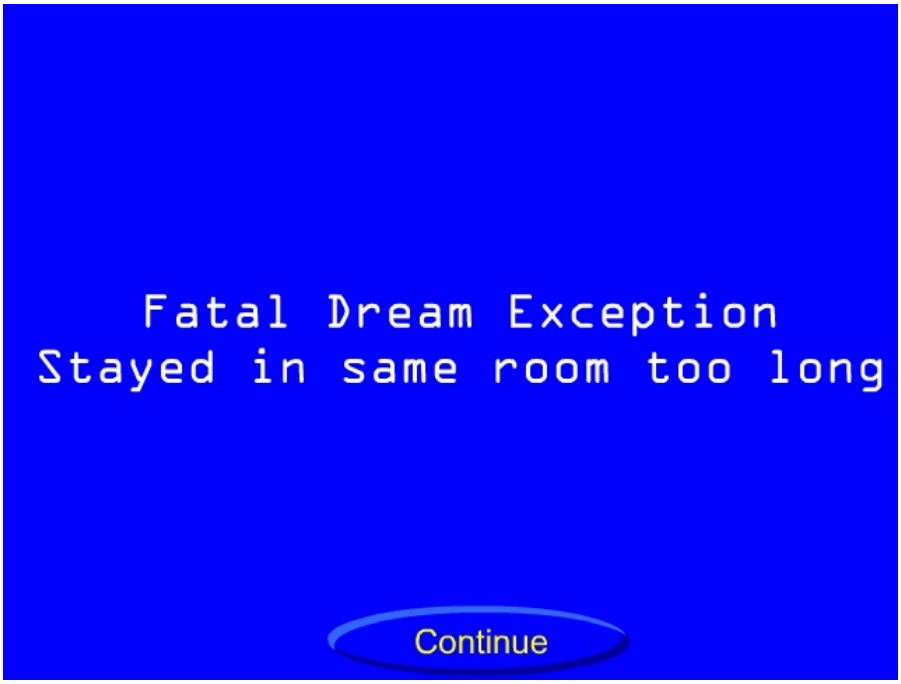

Figure 4 Game Over Sequence

As you can see by the screen shot, this defiantly reflects that the game is over, while also explaining why the game is over. All without taking the player out of the games reality. The punishment for failing is the game restarts. The code for handling this is very simple:

```
con_btn.onRelease = function() { gotoAndPlay("Title", 1); }
stop();
```

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.

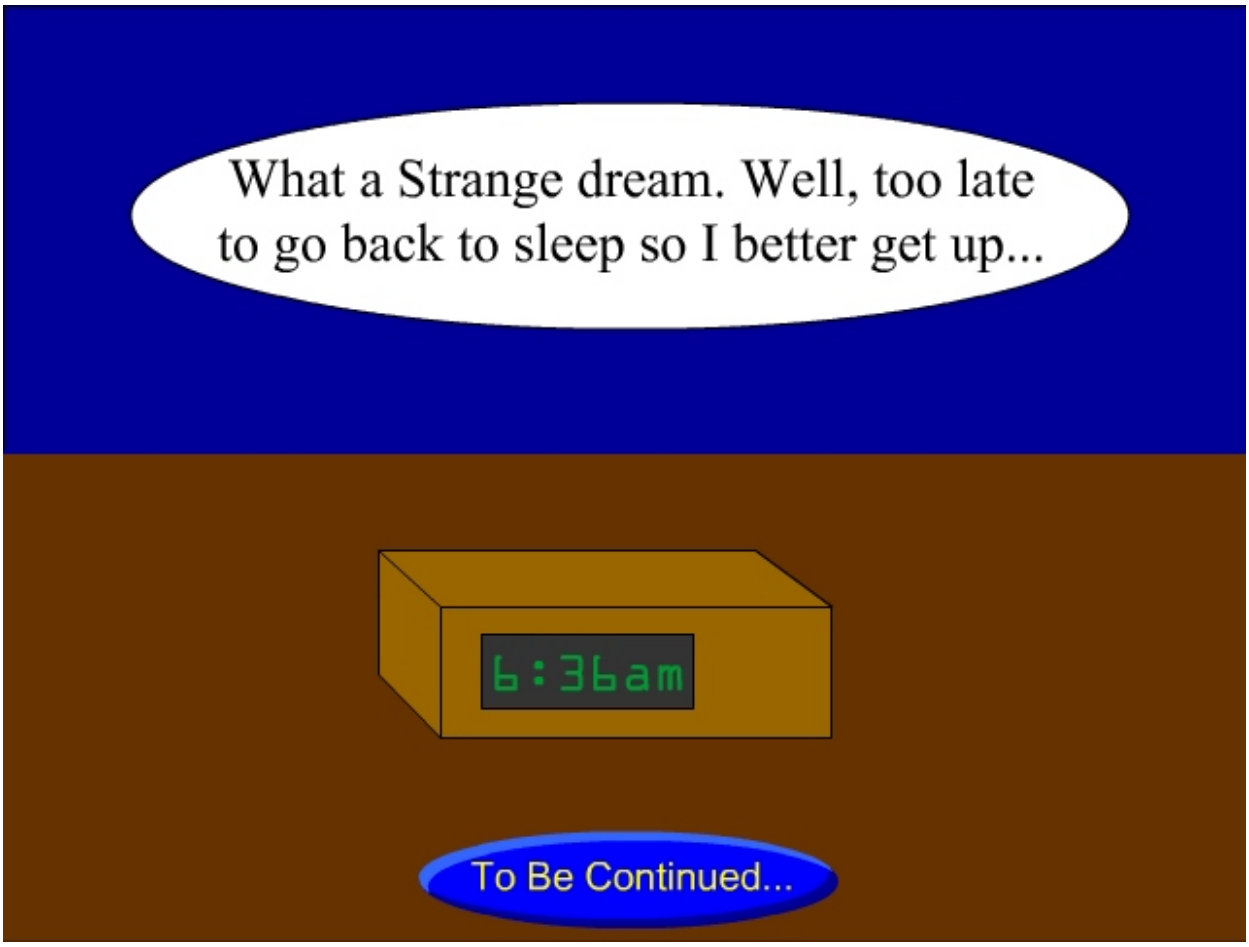

Figure 5: Winning the Game

The code for handling the "To be Continued..." button, as with the lose screen, is very simple:

```
con_btn.onRelease = function() { gotoAndPlay("Title", 1); }
stop();
```

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.

While I would like to think that people read the instruction pages that I put a huge amount of effort into creating. The reality is that a lot people go to the instruction page, rapidly scroll to the bottom of the page and click on the game's link.

One solution would be to have instructions on the game's title screen (which we will be building shortly). The problem with doing this is that the people who skip over the instructions page are likely to not bother clicking on the instructions page. This means that I would end up putting more time into creating the instruction pages - which is more time consuming then writing an instruction page - which few people are going to read. This is obviously not a good solution.

The solution I did is to incorporate the background portion of the instructions into the game by starting the player at an explanation screen. This is simply a looped animation of the text bubble growing then shrinking. The code for handling the button is also very simple.

```
con_btn.onRelease = function() { gotoAndPlay("enterRed"); }
```

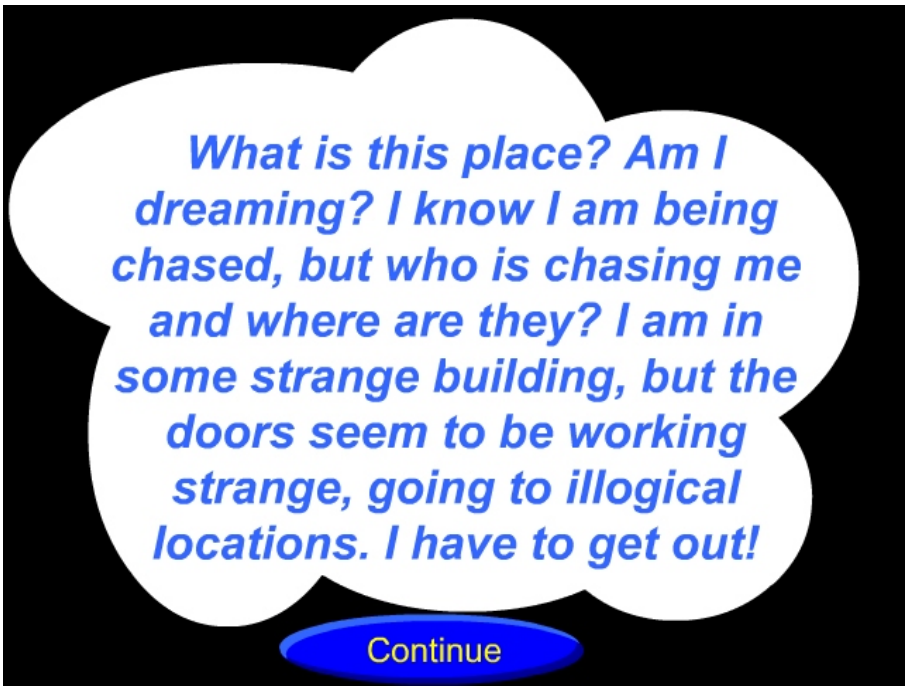

Figure 6: Introduction

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.

The first part of the title screen is the game's logo. I first did a few variations of the logo on scrap paper before coming up with a logo that I liked. This is created as a separate object. The logo is shifted using the skew feature. The logo is then motioned tweened into place. Next is a motion tween of the skew returning to normal orientation. Finally there is a faded in effect as the subtitle appears followed by the button. As always, the code to control the button is very straight forward.

Figure 7: Title Screen