Scripting Sequences

A “Sequence” is a series of events. In Amnesia, a sequence will describe a chain of events that happen in your story! This is useful for any sort of long cutscene, or some sort of intro sequence.

The problem

Our first problem that we encounter when thinking about this is, how do we keep track of what part of the sequence we're on? Our second problem would be how to go about structuring it.

Keeping Track of What Part of the Sequence We're in

When we think about it, all we really need is a unique identifier that would denote each section of a sequence. With a unique identifier, and the availability of variables, we can quickly see that either an integer or string can be used. For example, for some intro sequence, I could use “introWakeup”, “introWalkToDoor”, “introOpenDoor”, “introSoilSelf” as unique identifiers to denote what part we're in. This is a user friendly way, but it's quite expensive in memory (compared to the other method). The other method we can utilize (and the one I will be using in this tutorial) is using integers. Like strings, we can store integers. The only real advantage is memory. Integers will not take up more space than a string of characters in memory, so if you're concious about that stuff, then great!

In addition to being low on memory, integers have some other advantages. For instance, if your sequence is 10 parts long, using an int will be able to tell us where we are, percentage wise. If we're at part 5 (in this example), then we know we are 50% done. This is useful for allowing the players to skip a sequence after a certain percentage of viewing (as in, the first part of an intro may be important to view, while the rest isn't).

Using integers also allows you to change what part of the sequence the player is in easily, and with minimal effort. For example, say the player triggered something that would omit a certain part of a sequence. Using ints, we can simply do part+=2, and this will skip to the part after the next.

Structuring it all

We do not want to go ahead and create a function for each part of a sequence, right? (well, unless you're a masochist) So we need to find a way to keep it all nice and tidy. Having timers, and the ability for a function to call itself, we can do this quite easily. What we can do is have one sole function that will be called repeatedly by itself to progress through the parts. We will use a switch statement to only show the current part of the sequence. Switch statements are quick and effecient ways to branch out code.

Digging into the Code

To start, get your hps file up and ready to edit. Once that is done, we will start by creating the

“introSequence” function.

```
<code c>
void introSequence(string &in asTimer)
{
}
```

</code>

Simple enough, now moving on. Remember our first requirement. We need a way to track the part of the sequence we're on. Let's use a local variable for this. (remember, I'm using integers, you can easily edit it to use strings)

```
void OnEnter()
{
     SetLocalVarInt("iIntroPart", 0); // Initialize the sequence to null
}

void OnStart()
{
     AddTimer("tmrIntro", 1.0f, "introSequence"); // Launch the introduction
sequence 1 second in
}

void introSequence(string &in asTimer)
{
     AddLocalVarInt("iIntroPart", 1); // Increment the part so we move onto
the next (first run through will make this 1, or the first part)
     float partSpeed = 1.5f; // This denotes the DEFAULT value for how long
parts last in seconds
}
```

Great! We have our part tracker in place. I also added another key variable, “partSpeed”. This will simply tell our function how long each part is in seconds. The default is 1.5 seconds if you do not specify it yourself. Now let's go ahead and fill out the meaty and interesting part, the sequence! This will be a simple “wake up” sequence.

```
void OnEnter()
{
    SetLocalVarInt("iIntroPart", 0); // Initialize the sequence to null
}

void OnStart()
{
    AddTimer("tmrIntro", 1.0f, "introSequence"); // Launch the introduction
sequence 1 second in
}
```

```
void introSequence(string &in asTimer)
{
    AddLocalVarInt("iIntroPart", 1); // Increment the part so we move onto
the next (first run through will make this 1, or the first part)
    float partSpeed = 1.5f; // This denotes the DEFAULT value for how long
parts last in seconds
    switch (GetLocalVarInt("iIntroPart"))
    {
        case 1: // First part
            // Actions such as FadeIn, PlaySoundAtEntity, whatever you like.
This will be a simple wake up sequence
            partSpeed = 21.0f; // I want this part to last 21 seconds
            FadeOut(0.0f); // Instantly turn screen to black
            FadePlayerRollTo(50, 220, 220); // Simulate being on the floor
            FadeIn(20.0f); // Fade the screen in from black, 20 secs long
        break;
        case 2: // Second part
            partSpeed = 10.0f; // I want this part to last 10 seconds
            FadePlayerRollTo(0.0f, 33.0f, 33.0f); // Make the player's view
normal

            // Player is woken up!
        break;
    }
}
```

Awesome, we're almost done here!

Now we have two more problems to deal with, making the function repeat itself and determining when the sequence ends. Determining when the sequence ends will be as easy as checking the current part. If it's above a certain number (in this case 1) then we're at the final part. In this example, the function runs once, and the part is set to 1. The first case of the switch statement is executed. We will add code for the function to call itself, and through the second execution of this function, the part is set to 2, which is the final part. If we stick a simple if conditional at the end (if (part > 1) in this example) then we can stop calling the function when we reach the end! May sound confusing (actually, I'm just rambling on like a mad man here), but it's quite simple. As for calling the function again, that will be shown in the code :)

```
void OnEnter()
{
    SetLocalVarInt("iIntroPart", 0); // Initialize the sequence to null
}

void OnStart()
{
    AddTimer("tmrIntro", 1.0f, "introSequence"); // Launch the introduction
sequence 1 second in
}

void introSequence(string &in asTimer)
{
```

```
    AddLocalVarInt("iIntroPart", 1); // Increment the part so we move onto
the next (first run through will make this 1, or the first part)
    float partSpeed = 1.5f; // This denotes the DEFAULT value for how long
parts last in seconds
    switch (GetLocalVarInt("iIntroPart"))
    {
        case 1: // First part
            // Actions such as FadeIn, PlaySoundAtEntity, whatever you like.
This will be a simple wake up sequence
            partSpeed = 21.0f; // I want this part to last 21 seconds
            FadeOut(0.0f); // Instantly turn screen to black
            FadePlayerRollTo(50, 220, 220); // Simulate being on the floor
            FadeIn(20.0f); // Fade the screen in from black, 20 secs long
        break;
        case 2: // Second part
            partSpeed = 10.0f; // I want this part to last 10 seconds
            FadePlayerRollTo(0.0f, 33.0f, 33.0f); // Make the player's view
normal

            // Player is woken up!
        break;
    }
    if (GetLocalVarInt("iIntroPart") <2) // If the current part is less than
the total length of the sequence (in this case 2)
    {
        // Then we want to call the same function again to continue the
sequence!
        AddTimer("tmrIntro", partSpeed, "introSequence"); // Notice that we
use "partSpeed" as the delay, which will make the current part last for how
ever long we specified :)
    }
}
```

Conclusion

That's it! Wasn't too hard now was it? Fun fact, this is how the Frictional scripted their sequences. You can expand the “introSequence” function to as many parts as you wish. Just make sure to change the length in the conditional at the end (if (GetLocalVarInt(“iIntroPart”) < 2)) to reflect the length of the sequence.

Happy Coding!

Questions? Ask on the forums.