Sequences

Many of the events that happen throughout SOMA are triggered sequences - a sound plays, then the player's FoV changes, then a light starts flashing etc. etc. We control all of those through a set of wrappers we call Sequences, which hide a bunch of timers away and make things easier to read.

For each sequence you need a map property to store the state - a `cSequenceStatesData` property e.g.

```
cSequenceStatesData mSequenceAlert;
```

Then you create a sequence function. This will be repeatedly called until the whole sequence is over. It looks something like this:

```
void Sequence_Alert(const tString& in asName)
{
    Sequence_Begin("Sequence_Alert", mSequenceAlert);
    if(Sequence_DoStepAndWait(1.0f))  // Do this step and then wait for 1
second
    {
        MakeALoudNoise();
    }
    else if (Sequence_DoStepAndWait(2.5f)) // Do this and then wait for 2.5
seconds
    {
        FlashABrightLight();
    }
    else if (Sequence_DoStepAndPause()) // Do this and then pause until told
otherwise
    {
        SaySomethingAndCallBack("OnSayingSomethingComplete");
    }
    else if (Sequence_DoStepAndWait(10.0f)) // Do this and then wait for 10s
    {
        CrushPlayerLikeAnAnt();
    }
    else if (Sequence_DoStepAndContinue()) // Do this and go on to the next
step (in this case there isn't one)
    {
        ApologiseToPlayer();
    }
    Sequence_End();
}

void OnSayingSomethingComplete()
{
    // Saying something is now complete - poke the sequence to continue
processing
    SequenceStates_Resume("Sequence_Alert");
```

```
}
```

As you can see, `Sequence_DoStepAndPause()` in there actually pauses the whole sequence until some external event - in this case the callback from the voice playing code - calls `SequenceStates_Resume()` and asks it to continue.

To start the sequence, you just call the sequence function **once** with an empty argument when you want it to trigger e.g.

```
Sequence_Alert("");
```

no need to call it every frame or anything! Once started, timers will automatically make sure that the sequence steps get followed when they need to be.

We use this a lot, all the way through SOMA, sometimes running multiple sequences in parallel, as they're totally independent of each other. (Which is perfectly possible, but can get very confusing - we really wouldn't recommend it, it more grew out of level complexity than anything else!)

Important Functions

Sequence_Begin

Mark the start of a sequence block.

Sequence_End

Mark the end of the current sequence block.

Sequence_Stop

Stop the current sequence immediately (sort of like an abort).

Sequence_DoStepAndWait

Do the step within the following brackets and then wait for the specified time.

Sequence_DoStepWaitAndRepeat

Do the step within the following brackets and then wait for the specified time; repeat for a number of iterations.

Sequence_DoStepAndContinue

Do the step within the following brackets and then immediately carry on to the next step.

Sequence_DoStepAndPause

Do the step within the following brackets and then pause until Sequence_Resume is called.

Sequence_Wait

Just wait for a set period of time (no step in brackets).

Sequence_Pause

Pause the sequence until Sequence_Resume is called.

Sequence_SkipNextSteps

Skip the specified number of sequence steps.

Sequence_SkipNextStep

Skip the next sequence step.

SequenceStates_Pause

Pause a specified sequence.

SequenceStates_Resume

Resume the specified sequence.

SequenceStates_Stop

Stop the specified sequence.

SequenceStates_IsActive

Returns true if a particular sequence is active.