Scripting Basics for Amnesia Mapping Beginners

So you want to map some stuff for Amnesia. Great! The world needs more custom Amnesia content. The first thing you will note when you try to pick up Amnesia and whip out really cool maps for it is one thing. The whole Amnesia map idea is intricately woven in script. Now some of you may know scripting, and the lucky of you even know the programming language this particular script is based on, C++.

This tutorial is designed to give complete program and script newbies a crash course, and might be helpful to some others, too.

Some Helpful Tips

Here are a set of tips for anyone who is going to try mapping and scripting for this game. First tip: Do not take this whole thing here as a first step! You're going to want to do some other things first. Grab the tools, set up a dev environment, set up a custom story, set up a small map, you know, the usual pre-script things. These tips are, of course, based on personal experience, so may be entirely bunk in some cases. Take with a grain of salt.

Addendum To Dev Setup

If you're mapping for Amnesia, likelihood is that you are mapping under a custom story, in the custom story folder in the Amnesia directory. You followed the custom story setup and the dev environment setup word for word, and that means there's a few mixups.

When you follow the dev environment setup word for word, it seems to cancel out the custom story effort. Loading something that you are working on in the custom story folder through the debug menu alone does not appear to load the custom story content, such as the extra language file and any entities in the custom stories folder. This forces you to work in the main game folders, and pack up everything you need when everything is complete. Or, you have to test frequently outside of dev mode. That is inefficient and prone to mistakes for me, so I found a quick fix for it, so the game will read the custom story content despite debug mode.

Towards the start of the development environment setup, you set some things in main_settings.cfg, in your user settings directory (My Documents for us Windows folk). Open that file up again. We need to make just one change, I have found, to make custom story content load perfectly fine. Right after the start, of the file, you have a setting called 'ShowMenu'. If it says 'ShowMenu="false"', change it to say 'ShowMenu="true"'. That's it.

The main menu will pop up as if on a standard game. Pick the dev user profile, and check the custom story button. If your story is not in there, something went awry, but you caught it early. Check your story settings configuration file, and your language file, and make sure both don't have any issues. If your story is in there, you will load it up and be in debug mode, and all the custom story content will

also load properly, so there is much less fidgeting amongst directories.

I haven't found anything to show that this is a bad idea, but if it is, feel free to edit a note in about it.

Use Something Better than Notepad

This may sound like a silly suggestion, but trust me, it is worth it. When a map picks up in size and complexity, a script file can lead into having 800, 1000, maybe even up to 2000 lines of script. Even the most sane programmer in the world would quickly lose his mind trying to work out 1000 lines of code in standard notepad.

The features of Notepad++ will literally save you sanity. In the least, download it, open your script files, and hit the 'Language' tab and set it to C++. It is worth the extra download.

End of story.

Open the Map in Amnesia BEFORE you Edit Script!

There is one universal about scripting and programming. You will make mistakes. You WILL make mistakes.

This tip makes it such that mistakes in scripting, rather than having the entire game come to its knees and crash, merely bark an error at you so you can fix it and try again really quick. The title of this tip? It says it all. If you change a script and it is wrong, two things can happen:

One, you did not have Amnesia open on the map, and you try to load the map. You will get a popup error which complains of some specific form of issue, often semi-descriptive as to where you messed your script up. Hitting the 'OK' on that prompt closes Amnesia. Obviously, this will drive you insane if you have numerous errors, because no one likes repeatedly starting and shutting down the game.

Two, you were in the map and made the change in script, and hit the 'Recompile Language and Script' button in the debug menu. It gives you a more polite little message with the same semi-descriptive issue as before. The difference here is this time, you can close that little message and stay right where you are, script away, and just hit the compile button again. Incredibly useful.

Scripting Syntax for Beginners

This section focuses on the syntax and flow of the scripting, for those who don't want to break into learning C++ before trying to script for Amnesia. If you are skilled in scripting and/or programming, and especially if you know enough C++ to know how functions look and work under it, you can likely skip the rest of this tutorial. This is to give beginners a fighting chance of actually understanding what they are doing, and becoming self-sufficient at the craft.

Lets jump right into this! Lets say you have a simple map already made... if you don't, head on back and check out the level editor tutorial, and whip up a nice small map, just a single room, and save it somewhere you can load it up easily in-game. Once you have that, open up your text editor of choice, and save a file in the same folder as your map, called "<YOURMAPNAMEHERE>.hps" (of course, without the quotes or <>'s. Make SURE that you don't save it as a 'text file', but that the file type saves as .hps). This is now a script file, and it is blank, and it does nothing. The following is what you might expect such an empty map's script file to look like, so feel free to copy and paste it in there:

Basic Example File

```
//===========================================
// Starter's Script File!
//===========================================


//===========================================
// This runs when the map first starts
void OnStart()
{
    if(ScriptDebugOn())
    {
        GiveItemFromFile("lantern", "lantern.ent");
        SetPlayerLampOil(100.0f);

        for(int i = ;i < 10;i++)
        {
            GiveItemFromFile("tinderbox", "tinderbox.ent");
        }
    }
}


//===========================================
// This runs when the player enters the map
void OnEnter()
{
}


//===========================================
// This runs when the player leaves the map
void OnLeave()
{
}
```

If you are new to scripting, you will likely catch wind of what some of that means, but other things be totally lost on. Lets take a look at it piece by piece.

Comment

First thing I have there, with the slashes, is a comment. That is just there to be readable to you, the

writer of the script. It is not needed, it doesn't perform a function, it is completely ignored by the game.

OnStart

Going on down the line, we have 'void OnStart()' as the next meaningful line. This here is the 'definition' of a function, a special one. This one is called whenever the game sees the map and script file are first loaded. What shows us that it is a function? First, we have a data type, 'void', which basically is fancy program talk for 'nothing', or rather, 'when the function is used, don't expect anything back' (in programming C++, functions often return something, like a number; void does not do that). Next we have 'OnStart', which is just the name of the function, so nothing fancy there. Then we have '()', which is the argument list. There is nothing in it, because OnStart doesn't need anything! I'll outline that in a bit.

{ and }

Most importantly, we then have the opening bracket, {. If you look down below, lined up with it by indent is a paired }. Whenever OnStart is called, it skips right there to the opening bracket, and does every single line in it from top to bottom, until it hits that closing bracket! That is important to know. If you have something sitting out on its own in the script file, not in any bracket pair and doesn't have its own bracket pair, it likely does nothing whatsoever.

The if

Next we have an 'if'. This is just as it sounds! The if statement there has its own bracket pair, just like OnStart. If the condition in the parentheses for the if is true (like, if you have 'if(5 == 5)', which will always be true), it will do all the stuff in the brackets, top to bottom. If it is NOT true (say, you try (5 == 4)), it will do none of it, and will just skip to the if's end bracket and be on its merry way. In our case, we are checking if we are in debug mode, which we likely are. Notice in my comparisons there between 5 and whatever, I use two 'equals' signs! That is because one would not work, it would try to actually set 5 to 4, which is a bit impossible. You'll get an error if you use just one in an if!

Valid comparison operators are:

```
== Equal to
< Less than
> Greater than
<= or >= Less than/Greater than or equal to
!= Not equal to
&& And
|| Or
! Not
```

The 'And' and 'Or' are a way to combine requirements. && Compares two checks, and if they are both true, it says true. If one is true but the other is not, or if both are not true, it says not true. ||

Compares two checks as well. If ANY of the two are true, it says true, otherwise it says false.

The 'Not' is a special case. When put before a boolean (true/false), it signifies the opposite.

```
    // This is an if with an and.  5 == 5, so that would be true.  5 is NOT
equal to 4, so that would be false.
    // Due to the 'and', that means the if would not run.
    if((5 == 5) && (5 == 4))
    {
    }

    // This is the same if with an or.  It, in turn, would run, because one
of the two are true
    if((5 == 5) || (5 == 4))
    {
    }

    // Now this is that and with a NOT!  since 5 is not equal to 4, it
comes up false. The Not sees that and makes
    // it return as true for the if statement, meaning both are true for
the And, and it will run.
    // Do note that I could have just used the Not Equal To, !=, but I
wanted to show the not alone, as it can
    // be useful for functions which return a true or false, like
ScriptDebugOn()
    // (Say you have a scripted intro to your map that you don't want
playing every time you load up in debug,
    // you can simply put it in an if with the condition
if(!ScriptDebugOn()).)
    if((5 == 5) && !(5 == 4))
    {
    }
```

Functions in Play

This is what a function looks like when it is being called, and not being defined. There is a whole page of VITAL scripting functions for Amnesia that I advise you to bookmark if you will be scripting often. If you find the script we use here on that page, 'GiveItemFromFile', you see that unlike OnStart, it takes two arguments, which are separated by comma, the name of the item you are giving, and the entity file of that item. In this case, its the lantern. The next line is a function which sets how much oil the player has, in percent form... in this case i put '100.0f'. The 'f' at the end was because I gave it in float form (floating point... that is, it has a decimal).

Important to note is that, when calling a function to actually perform like this here, you fill out the variables you are shown on the script definition (everything on that vital functions page). If the script definition has a 'string' which is called 'EntityName', assume it means the entity you want to affect with the function, for example.

Very important to note, whenever you have a separate line in script like this, semicolons must be used! The semicolon basically says 'this line has ended'.

The 'for' loop

This next part is not something that comes up in Amnesia scripting often, but knowing it can make things incredibly convenient for you. For example, in my starting script there, I wanted to give the player 10 tinderboxes, just in case they want to run around joyously lighting up all the torches in debug mode like they are celebrating Christmas. Now I could write the give item function 10 times in a row to do this, or I could use a loop.

'for' is one type of loop, useful for things like this here, because it takes a variable and keeps track of it, uses it in a comparison, and then changes it. After the 'for' is the set of arguments, which tells us how to do those 3 things. It thus takes 3, separated by semicolons. The first sets up the variable we will use to count, in this case I create an int (integer/whole number) named i, and start it out at 0.

The second tells us what criteria to continue the loop under. In this case, as long as i is less than 10, we keep going.

The third tells us what to do with i whenever the loop finishes an iteration. In our situation, it says i++, meaning 'add 1 to i'.

The loop will perform whatever is in its brackets for as long as that second portion is true. Every time it hits the closing bracket, it goes to the third argument and does that. If you follow the loop in your mind, it will go from 0 to 9, meaning it will give the player 10 tinderboxes. When i is equal to 10, it is of course no longer less than 10, and the loop does not go again.

Data Types

I am adding this session in here for much needed clarification. If you look at the list of Amnesia script functions, you'll see that they take all sorts of types. Here I will clarify what those mean. When a function on that list has an argument like that, what that means is when you call the function to run it, you put an appropriate value in for it, like how we put "lantern" in for GiveItemFromFile, or 100.0f in the oil function below.

```
    int //(Integer, or whole number.  Examples include -1023, 0, 6)
    float //(Floating point, or decimal number.  -16.235f, 0.0f, 7.2f,
5.0f)
    string //(String.  Basically, a 'string' of letters and numbers.  Often
used as entity names
           // like "mansion_door_1".  Pretty much always in double quotes.)
    bool //(Boolean value.  true or false.)
```

Syntax Conclusion

Those are the basics you need to know about syntax to really understand what is going on, and if you are good with this kind of stuff, you blew through that entirely with a mindset of 'makes sense' or 'too simple…'. If you're not good with this stuff, and still find yourself confused, best thing to do is open Amnesia and toy around with the scripting, see what happens. Another ideal thing is to open up an

actual map in Amnesia and see what the scripting is like!

And remember, just like I have the script here, indenting will save you mind power in figuring out! You'll see what belongs to what more easily, and you will be able to follow the script more easily. So remember, use that tab key!

Amnesia Scripting

I am unfortunately not going to cover this here, because there are a good number of other tutorials already on here which cover that! I will just outline a few very important functions.

If you did read the syntax blurb up there, you know that anything that is to be done must be in a function of some sort. The trick with scripting is knowing how to work these chunks of scripts together such that they flow.

Note that when functions are defined, the parameters remain a data type. When you are writing out the 'void Function(...){...}', whatever is in parentheses should remain a variable, depending on what it needs, such as 'string &in asEntity'. If you're not sure what it needs and you didn't produce the function yourself, seek out if you can find more information on the function (Amnesia default maps are always a good resort to checking script).

Interacts

So you want something to happen when you try to interact with an object, such as a door. In the level editor, most interactive objects have an option in entity settings for a player interact callback. Just put a function name in there, and it is then a valid function to use in script!

It looks like this in there (note, this is not a functional example, just showing what the function looks like):

```
    // asEntity becomes the name of the entity which referenced it, simple
as that.
    // When the variable is referenced in an interact function called by a
door named
    // "door", asEntity would literally read out as a string saying "door"
    void FunctionNameHere(string &in asEntity)
    {
        [STUFF YOU WANT DONE GOES IN HERE]
    }
```

Easy as that, that is run when the player tries to 'use' the entity. The argument 'asEntity' becomes the name of the object used.

Collides

So you want something to happen when the player unwittingly comes into contact with something, be it an object or an invisible, intangible area. For this, you can't just set it up in the editor. You need to

'add' the collide callback in another function, such as OnStart.

(Same as before, but showing the necessary adding of the callback)

```
    void OnStart()
    {
        // Creates a collide check for if player enters 'ScriptArea1',
that runs the function 'Collide_Area'.
        // It deletes the callback when it is run, and only checks for if
the player enters the area.
        AddEntityCollideCallback("Player", "ScriptArea1", "Collide_Area",
true, 1);
    }

    // The actual function.  asParent becomes the 'parent' of the added
callback ("Player"),
    // asChild becomes the 'Child' ("ScriptArea1"), alState becomes the
state (1 if player entered,
    // or -1 if player exited. Can only be 1 in this case).  This all
happens when the script actually
    // runs in-game, do NOT do it in script itself!  You can, however,
reference the variables asParent,
    // asChild, and alState in the function itself, and they will work,
though.
    // This is important to note because, yes, more than one callback can
point to the same function!
    // That can be used as a method of differentiating them.
    void Collide_Area(string &in asParent, string &in asChild, int alState)
    {
        [STUFF YOU WANT DONE HERE...]
    }
```

Timers

Here I will cover timers, which allow you to time out events and occurrences, and if you think hard
enough, even allow you to create a timed version of the for loop! If you have done any video game
mapping and modding before, you know that it is true that timers rule the world.

A timer is called in a function, like any other (as with the past three):

```
    void OnStart()
    {
        // Creates a timer.  The timer is named "Test", triggers 5 seconds
after map starts, and runs the function "Timer_Trigger"
        AddTimer("Test", 5.0f, "Timer_Trigger");
    }

    // asTimer becomes timer name.  In this case, if you would reference
```

```
    // asTimer in the below function, it would contain the string "Test".
    // This means you could do something like "if(asTimer == "Test")", and
    // have multiple timers pointing to the SAME function, but doing wildly
    // different things.
    void Timer_Trigger(string &in asTimer)
    {
        [PUT STUFF HERE TO DO...]
    }
```

Local Variables

One thing you may notice is that, to do some neat stuff, you need more than just your basic variables and such. You need some way for them to communicate. If you have a timer triggering, triggering itself again repeatedly for 10 seconds, how do you tell it to not trigger itself again after that 10 seconds? This is what I implied by 'timed version of the for loop', by the way.

The local variables are called as simply as the following:

```
    SetLocalVar<Type>("<Name>", <Value>);
        <Type> can be Int, Float, or String
        <Name> can be... anything
        <Value> depends on <Type>.  Int can be any whole number, Float any
decimal number, String, anything between quotes
    That set takes the local variable of the given name, and makes it the
value.  If there is no set to a given
    variable before use, it will default to 0, 0.0f, or "" (presumably, have
not tried strings)
    AddLocalVar<Type>("<Name>", <Value>);
        All is same as before, except this time we are not setting the
variable to the value, but adding the value to what
        is already there.
    GetLocalVar<Type>("<Name>");
        This returns whatever is in the variable.
```

For example, we can have this, which gives the player 10 tinderboxes over 10 seconds. This is the only example of this section which is functional, because its harder to understand something like this in description only.

```
    void OnStart()
    {
        // Adds a timer to start us off.  Triggers in one hundredth of a
second... or 'really really quick, you don't even know!'
        AddTimer("", 0.01f, "Amazing_Delayed_Tinderboxes");
    }

    void Amazing_Delayed_Tinderboxes(string &in asTimer)
    {
        // If the local variable 'TinderCount' is less than 10...
        if(GetLocalVarInt("TinderCount") < 10)
        {
```

```
            // Give a tinderbox.  Add 1 to the count.
            GiveItemFromFile("tinderbox", "tinderbox.ent");
            AddLocalVarInt("TinderCount", 1);

            // Do it all again!
            AddTimer("", 1.0f, "Amazing_Delayed_Tinderboxes");
        }
    }
```

From:
https://wiki.frictionalgames.com/ - **Frictional Game Wiki**

Permanent link:
https://wiki.frictionalgames.com/hpl2/tutorials/script/disable_gravity_tutorial?rev=1288882024

Last update: **2010/11/04 14:47**

// Give a tinderbox. Add 1 to the count.
AddLocalVarInt("TinderCount", 1);