Quick Start

Introduction

Amnesia allows you to associate a script file with your map; from within this script file, you can control various aspects of the gameplay for your level. To connect the level with the appropriate script file, he engine relies on a naming convention: the script file is expected to have the same name as the map file, but the exstension should be hps.

For example:

- Map File: `MyLevel.map`
- Script File: `MyLevel.hps`

The script file is simply a plain text file, and it contains code that is compiled and executed by the script engine. As such, it can be opened, viewed, written and edited in any plain text editor, such as Notepad on Windows. However, it is better to write your scripts in a specialized code editor, since these editors provide additional features like syntax highlighting, word completion, etc., which can make a world of difference when it comes to scripting. Check out Notepad++ and Geany.

Note: You should not use office-like applications or similar text editors when writing your scripts, because these don't save the file as plain text; such editors use specialized file formats, and the documents are saved as binary files, or more often, as markup, where additional information might be inserted in the file, alongside your text. This extra information is used to save the formatting of the text and pages, font properties and colors, graphics, etc...

Your First Script

It is not required for the script file to exist, but if you create one, it must not remain empty – otherwise, the game engine will not accept it. The file should contain at least one line of valid code. When the level is loaded, the game will look into the script, and it will call the OnStart() and OnEnter() functions, if they are defined in the file. OnStart() is called once, when the level is first loaded, while OnEnter() is called every time the Player enters the level.
So, we can start by defining the OnEnter() function. Type in this code in sandboxMap.hps (☒ ~~sandboxMap coming soon~~), or in a script file for your own map:

```
void OnEnter()
{
}
```

Now quick-reload the map from the debug menu. If the script is well-formed, there should be no error messages. When the map was quick-reloaded, the OnEnter() function was called, but so far this function does nothing. Since that is not very interesting, let's add a few more lines of code.

```
void OnEnter()
{
    FadeOut(0.0f);  // Turns the screen black instantly
```

```
    // Now that the screen is black, do a 5 second fade in
    // for dramatic effect.
    FadeIn(5.0f);
}
```

This code first makes the screen start as black, and then fades in the picture during a 5 second period. Quick-reload to try it out. Now, this code demonstrates the basics of Amnesia scripting, so let's go over it.

This short script contains a single function definition – the definition of OnEnter(). It also makes two calls to two other predefined functions, FadeOut() and FadeIn(). (Note, I'll always append '()' to function names when talking about them, to make them easily distinguishable from other script-related things, like variables, parameters, etc.)

The first line
`void OnEnter()`
declares the OnEnter() function. The token in front of it denotes its return type; in this case, it is `void`, which you can think of to mean “nothing” or “empty” – it simply states that this function does not return any value (some functions you'll write *will* return a value, e.g., a result of some calculation). OnEnter() just does something (starts the fade in effect), but nothing is returned from it (you'll learn more about return values in the section on functions).
Everything that follows after that line is the *body* of the OnEnter() function. It is the function's body that defines what the function does. It is enclosed by `{` and `}`. You can think of these symbols as of boundaries – they bound the space of the function body. Everything in between them belongs to the function.
If a region of code is bounded by `{` and `}`, it is also referred to as a *code block*.

The text behind `//` represents scriptwriter's comments. These are completely ignored by the script engine, as if they weren't there at all. They are intended for the human reader – you can use them to explain what a certain peace of code does. You can delete them, and the script's functionality will remain exactly the same.

```
void OnEnter()
{
    FadeOut(0.0f);
    FadeIn(5.0f);
}
```

Inside the OnEnter() function, a call is first made to the FadeOut() function, which results in a black screen. The FadeOut() function comes predefined with the game. You'll notice that there's a number `0.0f` inside the parentheses that follow it. Forget for the 'f' suffix for the moment. This number, (0.0 – zero) is a *parameter* to the FadeOut() function. It represents the duration of the screen fade out effect, in seconds. Since it's zero here, the screen will instantly turn black.

The next line is a call to a function which is also predefined by the engine, and it also takes a single parameter, which likewise determines the duration of the effect. The result is that the screen fades in during a 5 second period, starting from the moment the call to FadeIn() was made. Since this call was made inside OnEnter(), which is invoked by the game at the very start of the level, the Player experiences a nice fade in effect on gameplay start.

That's it for this quick start guide. You might also want to read the Execution Flow section - it explains at what points in time execution jumps from the game engine to the script engine, and vice versa.