Scripter's Guide

Intro

This guide will serve as a knowledge base for what to do and not do when scripting a map. It will go through what is important to think about and will include lists with various tips and advice on how to approach this.

Set-up

At least one time, you need to do the following:

- Make sure that you have read the Design Document You must feel at home with the design principles we want to use, the general outline of the player's journey and with the mood/themes of the game.
- Play all three Penumbra Games and Amnesia (do not forget Justine expansion) so you get a feel for what kind of activities are possible.

Every time you start on a new map, you need to:

- Make sure to read the design carefully and make a list of the things you need to implement. It is good to figure out some kind of priority order and what will be the hardest tasks from the get go.
- Have chat with Thomas and talk through the design of the level. Make sure there is nothing that you are unsure about. Remember that Thomas might sometimes forget these meetings, so nag at him. You do not want to start before having done a walkthrough.
- The most important part of the design for level is what it is meant to achieve in terms of narrative and mood. Make sure you get this.

Programming

General

When you need to implement something in script there are a few things you need to first consider:

- Whatever you make will probably go through several iterations.
- There will be people besides you that will mess with the code.
- Making something functional according to design is only the first and simplest of your problems.

What this means is that you should always try and do something as fast as possible by using available solution. Also make sure that whatever you do is open to manipulation and not only working for a very specific behavior. The best way to go about is to always use some entity type that already exist and change it's properties to fit your needs. If what you are doing with require some extra features, then either make it specific to this map (in the maps' script) or add it to the entity's code. When adding it to the entity make sure that it is something that can be useful elsewhere and also think carefully so

what you add a generic version of your wanted feature. That is: do not make up something extremely specific.

Remember KISS!
This lecture is a must view: http://the-witness.net/news/2011/06/how-to-program-independent-games/

Also important is that you structure your code in a way so that other people can easily see what does what. It is very common that other people will want to tweak something or just add some extra effect. It must be easy to see where your different events occurs. The engine supports TODO message pop-ups (`cLux_AddTodoMessage(..)`), so use that in all places where some effect/sound/whatever is required to be added. Despite adding a todo, you should still have your own temp assets when possible. For instance, sometimes sounds are vital to see if an event works or not.

Philosophy for level scripting

A level script should not be thought of as file for programing. A level script should be written and read as though it is a story (or a sequence of events if you wish).

In the level script you are crafting the experience for the user and the script should be read as the manuscript of that experience.

If the level script file contains rows of code that does not match the above description, then the rows of code does not belong in the level script, they belong in a helper file or even deeper down in the hierarchy of script files (or even in the c++ code!).

This primarily means that when working to solve a problem of how to implement an event (activity, puzzle, challenge, character interaction etc) in a level, the first approach is not that of a programmer. The correct approach to take is:

- Begin by examining and configuring the resources (entities, sounds, graphics) that is used in the event. This to make best use of what is already available (in terms of prop types, configurations and so on).
- Move on and make sure that the area in the level used for the event is created in a way that works. If not, you tweak it to be better. This can be as simple as renaming entities in a way that simplifies the scripting of them. You also make sure you to add in what ever new things you need (such as script areas).
- Finally you can begin scripting. This you do by making use of all the existing helper functions to create a short and easy to read solution for the event implementation.

Only when or if the above will not accomplish the task at hand you continue and create helper functions, new states or other changes and additions to the lower level scripts.

However, You should also consider the functions and entities that are at hand. Do they really solve your problem in the easiest way possible? If not, then you should consider adding or extending the functionality of them.

When doing the actual level editing the focus must never be on a technical level. It should not be complicated to add something, rather it should be easy. The hard part should be making the gameplay and atmosphere good, it should not be making the game do what you want. If this is not true, then you probably need to add some more functionality yourself or contact a friendly programmer to do so for you.

Beware though, we do not want to have thousands of object types and functions, we still want it all to feel comprehensible. Make your additions and changes as generic as possible, meaning that it does not only fit your very specific need, but can be used for a multitude of purposes.

Work-flow

The game has been made to make feedback when scripting as frequent as possible. First of all, with the basic settings the script will be reloaded every time you task switch from the editing program (usually CodeLite). So when making smaller changes in code that are constantly updated, then simply task switch to see your new code in action. Inside the F1-menu you can also turn on “Update script constantly” which checks for script updates as soon as you save the code. So if you have two monitors (or one really large) you can have the game running next to the code and every time you save you see updates and see if there is any compile errors.

If there is any updates that require the map to restart, meaning any init function or properties in the map, then you of course need to reload. Do this by pressing F5 or the reload button in the F1-menu. It is only very seldom that you actually need to turn off the application

Here are some other tips:

- Press F3 to speed up the update rate in the game. This is especially useful to quickly playing through conversations, long events, etc when testing.
- Press F7 to go into spectator mode. This allows you to easily move around in the map. Hold down shift to go extra fast.
- Press F2 to pauses the game. This is nice to use with Spectator mode and you can closely inspect something to look for bugs and make sure it plays out like it should.

Code Structure

Since a lot of people will be working on the same piece of code it is important that the same structure is maintained. This way it makes it easier for people to find what they need to do to change something.

Note: In the code below, spaces are used to indent, do NOT use this, use tab instead!

```
//------------------------------------------------

/*Place any global values here. These must be const variables as they will
not be saved*/
/*This is also the place for enums and classes, but these should be avoided
whenever possible*/

//------------------------------------------------

class cScrMap : iScrMap
{
    //-----------------------------------------
```

```
////////////////////////////////////////////////////////////////////////
//////////////
    // ===============
    // MAIN CALLBACKS
    // ===============
//{////////////////////////////////////////////////////////////////////
//////////////

    //-------------------------------------------------------

    /*OnStart, OnEnter, OnLeave, Update (avoid this), OnAction (debug only),
OnPlayerDead, etc are here.*/

    //-------------------------------------------------------

    //} END MAIN CALLBACKS


////////////////////////////////////////////////////////////////////////
//////////////
    // ===============
    // MAIN FUNCTIONS
    // ===============
//{////////////////////////////////////////////////////////////////////
//////////////

    //-------------------------------------------------------

    /*Put any variables that are used in more than one scene here.*/

    //-------------------------------------------------------

    /*Put any functions that are used in more than one scene here.*/

    //-------------------------------------------------------

    //} END MAIN FUNCTIONS

////////////////////////////////////////////////////////////////////////
//////////////
    // ===============
    // SCENE X *NAME OF SCENE*
    // ===============
//{////////////////////////////////////////////////////////////////////
//////////////

        //-------------------------------------------------------

        /*Put any variables that are used by many events in Scene X here.*/
```

```
        //-------------------------------------------------------

        /*Put any functions that are used in more than one event in Scene X
here.*/

         //-------------------------------------------------------


         ////////////////////////////////////////
         // Scene X Event X *Name Of Event*
         //{////////////////////////////////////

         //-------------------------------------------------------

         /*Put any variables that are only used in Scene X, Event X here.*/

         //-------------------------------------------------------

         /*Put any functionsthat are only used in Scene X, Event X here.*/

         //-------------------------------------------------------

         //} END Scene X Event X

     //} END SCENE X


     ////////////////////////////////////////
     // ==============
     // TERMINALS
     // ==============
     //{////////////////////////////////////

         //-------------------------------------------------------

         ////////////////////////////////////////
         // Terminal *Name Of Terminal*
         //{////////////////////////////////////

         //-------------------------------------------------------

         /*Put any variables that are only used Terminal here.*/

         //-------------------------------------------------------

         /*Put any functions that are only used Terminal here.*/

         //-------------------------------------------------------

         //} END Terminal *Name Of Terminal*
```

```
    //} END TERMINALS

}
```

Note1:
It is OK if terminals use variables that are scene or event specific!

Note2:
Have snuck in some { and } to make code folding for sections possible.

Note3:
Remember to always separate functions with a comment line, like this:

```
void Func1()
{

}

//------------------------------------------------

void Func2()
{

}
```

Naming Conventions

A few naming conventions for map scripting and level building:

Areas

Name trigger areas in the level Trigger_xxxx to make them easier to find.

Sequences

Name sequence functions in your map script Seq_xxxx

Design

Intro

Remember, when you script maps you are a designer. Your job is not just simply to replicate the descriptions in the design doc. Your job is to use that as a base and create an engaging experience in each level. Sometimes this means just making slight alterations. Sometimes it requires huge changes

to the first design. You need to have in mind what we want to achieve with the level you are working on, and then make sure that is the end experience.

A good mantra is: *If you simply copy the design, the game will suck.*

Tour

So lets have a little tour of DO's and DON'T's of design. Start this video and then check the time-stamps below: http://www.youtube.com/watch?v=6sEyzilwfAE

0.30
Notice the fly particles systems and stuff like that? Try and add stuff like that where you think it works in order to increase the mood. This of course not your first priority but I want to mention it because stuff like this is part of your work.

0.36
Never, ever take away the player's control like this! The player must always be able to move and feel as if they are in control of the actions. Even if you move the camera for something like sitting down on chair, the player must still be able to move their heads. Keep the immersion-loop of interaction going at all times!
If the player needs to spot something, use other tricks than taking the control away. This is cheap.

0.46
Another bad thing here. We never want to tell the player outright what to do next. The player should feel free to do what they please and the design should lure them into going in the right direction.

1.06
Here you use the same basic controls for doing a new kind of activity. This is a good thing! If you can think up things like this to have in the level; add it! Keeping balance on a log might not be good in first person though ;)

2.04
Never ever show the player what controls they need to use. We will have some kind of (optional) help popups at the very beginning (chapter 0 basically) and after that there is none of that!

2.30
The metal breaks as you climb it, this is good stuff for the engagement and also make the world feel more alive. Feel free to add unexpected (but realistic) events to the player's interaction. The plane falling down afterwards is also good. But if you do it in cut scene, you are doing it wrong!

2.50
Laura holds her hand against the wall. To change the player's movement/posture/etc has they move through specific environments is really good stuff. See if you can fit something like that in. Might be as simple as more heavy breathing or shorter footsteps in some section.

3.20
If the player finds a bag or similar, try and not just have a “click to open”-interaction. Instead have some form of analog input to do this instead. Always best if the player can use mouse movements to perform an action.

6.10

Birds that fly when you approach. This kinda stuff is nice. If you add it make sure to make it generic so we can use it many times and for different things!

7.05
Introducing new controls like this is not good! This does not come intuitively from the basics and thus not something we should have.

7.30
When the game opens up like this it is important that there is not only a single hotspot for the player to find, but that one can complete the task or find interesting info in many places. I think this is done nicely here with the deers showing up wherever you are.

Foundational Rules

No Puzzles
Do not think of our puzzles as such. Think of them as *Activities* in stead. This is very important to keep in mind as they are not meant to pose a challenge but to make the player feel more part of the world. The player might still be required to think, but we do not want to end up in a “guess the designer” type of gameplay. It should feel natural and the player should “solve” any activity by simply just trying enough. For instance in Amnesia you can break down a wall by throwing a rock at it or simply clicking on it a lot. In either case the player has made a proper attempt at breaking down the wall and needs to be rewarded. Being streamlined, intuitive and coherent comes way ahead of being a satisfying problem to solve.

Always Interactive
Whenever possible, the objects in the environment needs to be interactive. For instance, a locked door should be able to move so the player can notice, by interacting, that it cannot be opened. Another examples is that the player might be able to grab and swing lamps that hang down from the ceiling. We want the player to feel part of a real living world and the interaction is the way we do this. Never waste an object that can be made interactive.

No Trial and Error
Never have challenges where failure means restart. Examples of this is just about any puzzle in Limbo. We want the player to a smooth narrative experience that always moves forward. We do not want to to be stuck repeating the same actions over and over.

Physics When Possible
Do not “hard-code” behaviors, but try and use physics when ever possible. This will also allow the player to experiment more and to allow a great set of possible solutions to activities. Beware that physics can be very unruly and make sure that there are restraints on what can be done. When using physics you really need to test a lot.

Specific Guidelines

Readable items must have consistent placement
When placing things that can be read, such as notes, books, newspapers, pamphlets, etc, it is very important to follow strict rules on this. We want the player to figure out by simply looking, what is possible to read and what is not. The player should not need to go pixel hunting. Here are the two basic rules for this:

1) Readables lying by themselves can always be interacted with.
2) Readables that can not be read, bust be in groups; piles, stacks, rows, etc.
Also make sure that the colors on the readables that are slightly more saturated than the group ones. The only exception for this is scraps of paper, which are possible to have lying by themselves. But these can not have any longer text, but only pictures, diagrams, etc with some small written notation. They must also feel crumbled and fairly useless.

Longer must be confined to single lang entry
Any longer text for notes, descriptions, books, etc must be inside a single lang entry. They cannot be spread out over several entries for any reason. When having a single lang entry, it makes it a lot easier to maintain and translation gets easier.

Books

For books, we can show the front, the back, or an interesting page in the middle (swapping the model lets us show an 'open' version).

- If the back, then we show text of blurb
- If the front, then we just show title, author & whatever
- If somewhere in the middle, then we show an extract from the book.

The key thing to remember here is: “If the protagonist picked up this book, which bit would he be most interested in?”

i.e. it's unlikely we'd show an extract from the middle, unless it's critical to the protagonist's current plans e.g. the instruction manual for a device he's interested in or something.

Further information

What follows is a collection of articles that are good to read:

http://frictionalgames.blogspot.se/2013/02/puzzles-what-are-good-for.html

http://frictionalgames.blogspot.se/2013/03/puzzles-and-causal-histories.html

http://frictionalgames.blogspot.se/2013/01/goals-and-storytelling.html

http://frictionalgames.blogspot.se/2012/12/introduction-i-recently-started-to-play.html