HPL3 Reference Guide

This is a guide for anyone coming from HPL2 to help describe common actions done in Amnesia modding. In this guide, I will show you how to accomplish how to achieve the same effect (or at least as similar as you can get) in HPL3.

Key Pick-Up

In HPL3, keys, puzzle objects, or any other item that can be picked up is all grouped together under the term “Tool”. When an entity with the type “Tool” is placed in the level, the player can pick it up just by interacting with it - no scripting required.

[Image Placeholder - Player Picks Up Chip]

Of course, there are times when you want to know if the player has picked up a tool. In this case, you can add a function to the tool's “OnPlayerPickup” callback. This callback gets fired when the player (appropriately enough) picks up the tool.

[Image Placeholder - Editor Screenshot of Callback]

[Image Placeholder - Codelite Screenshot of Callback]

Forces and Impulses

Force vs Impulse

Force changes the speed of an object over time. It is used for things that have a cumulative effect, like gravity or acceleration.

Impulse changes the speed of an object instantaneously. It is used for things like jumps and explosions.

Simple Way

There are four convenience functions that will quickly allow you to add force or impulse to an entity:

- Entity_AddForce
- Entity_AddForceFromEntity
- Entity_AddImpulse
- Entity_AddImpulseFromEntity

The names of the functions offer a straightforward explanation of what the functions do:

The **Entity_AddForce** and **Entity_AddImpulse** functions enable you to apply force/impulse on an entity with a direction and magnitude specified by a given cVector3f parameter.

The **Entity_AddForceFromEntity** and **Entity_AddImpulseFromEntity** functions enable you to add force/impulse on an entity originating from the position of another entity with a magnitude specified by a given **float** parameter.

All four functions support wildsards (*) for affecting multiple entities with a single function call.

Usage example:

```
Entity_AddForce(
	"entity_name",			// The name of the entity
	cVector3f(0.0, 5.0, 0.0),	// The direction and magnitude of the force (in this case, straight upward)
	false,				// If true, the force is applied relative to the entity's local rotation
	false);				// If true, the force is applied to the entity's main physics body only
```

Advanced Way

Physics in HPL3 is a bit more complicated than in HPL2. Every physics-enabled entity in HPL3 has attached to it an iPhysicsBody which can be retrieved in script. (In order for an entity to be physics-enabled, it must have body objects attached to it in the ModelViewer.)

To get the iPhysicsBody object of an entity, you do the following:

```
// Retrieve the entity object from the map
iLuxEntity @entity = Map_GetEntity("entity_name");

// Retrieve the iPhysicsBody from the entity object
iPhysicsBody @body = entity.GetMainBody();
```

From here, the functions related to force and impulse are:

- **AddForce** (const cVector3f &in avForce)
- **AddForceAtPosition** (const cVector3f &in avForce, const cVector3f &in avPos)
- **AddImpulse** (const cVector3f &in avImpulse)
- **AddImpulseAtPosition** (const cVector3f &in avImpulse, const cVector3f &in avPos)

These functions behave similarly to their convenience function counterparts, with **AddForceAtPosition** and **AddImpulseAtPosition** applying their effects from an arbitrary point in the world rather than the position of another entity.

Usage example:

```
// Add a vertical force to the entity with a magnitude of 5.0
body.AddForce(cVector3f(0.0, 5.0, 0.0));
```

To-Do List

- Monster Spawn
- Wake-Up Sequence