Sonic Unreal Engine 4 – Instruction Manual

Resume

What it is

HDK , acronym for Hedgehog Development Kit, is a Sonic fan game engine for non-profit purposes that is free to distribute, copy and modificate, it is to be used in with Epic Unreal Engine 4. For being a fangame engine, every game, level or other work made using using copyrighted material, as from Sega, Sonic the Hedgehog can't be used commerciallized, only free distribuition.

The engine idea is to create the Ultimate Sonic Fan-Game Engine, a modular and expansive engine, that any person will be able to contribute to the official builds, with codes, characters or expansion packs – that would contain a certain group of files to develop games, as SoundFX packages that were some of the first to go into the engine, making packs of sound from several games. Other expansion packs could contain modular parts for making certain types of levels, even classic ones, or objects ripped that form a level. With this premisse, the engine can grow into the most versatile, easier to use and powerful Sonic engine.

About the manual

Using HDK engine may be a straight-foward situation, but need some stuff in order to set up levels and objects to work. This document aims to clarify each part of the engine, while offer some basic advice and good habits and tips on using UE4 and avoiding headaches.

Where this manual is going at

What this manual hopes to do is explain everything about the HDK Engine dependencies, and some basics about Unreal Engine 4 that is very tied to the work of the engine. By no means it wants to explain UE4 functionality besides the ones outlined here. Also, if the user wants to know more, the Epic Unreal Engine 4 documentation already provides most, if not all the necessary information to work, and is highly suggested to the user to read and keep it as a bookmark for continoulsy references and higly recommended to the user to make experiments and learn, and always look for the answers and not give up, most things aren't impossible or improbable to an individual to make, if he(she) dedicates on it. Also the UE4 community is one of the best tools for learning and sharing resources, which is wise to make use of it the most.

Project and Level Setup

When creating a project and a new level, some basic setup is in order, and the user must know these. To reduce the setup time, the user also can start by the HDKDefaultMap, which contains the same setup outlined in this topic, and save as a new level.

World Settings

Game Mode

The Game Mode file determinates which control, character and HUD to setup. Use it to be able to control Sonic as you press the Play button on the editor or compile a game project. There are two ways to use the Game Mode file, one is on Project Settings, that will overwrite all levels created within the project to run with that Game Mode, or in the World Settings, that is the local level settings, not affecting other levels created. After set a GameMode file, the HUD, Controler and GameState are automatically defined by the files linked to it, but you can change to custom made ones.

>To enter the Project Settings, click on the Edit Tab on the Main Menu>Project Settings, and on the window that opens go to the Maps and Modes Tab and alterate on Default Modes>Default GameMode>Select a file.
>To enter the World Settings, you can click on the globe icon in the main ribbon, or go to Window>World Settings

GameMode Overrride – this is where the SonicGameMode goes, it set the type of game with it's parameters
Selected GameMode

Default Pawn Class – The main character to use in the level
HUD Class – Defines the HUD to be used in the level, the HUD comunicates with GameState to get the variables to set on screen as Rings, Lives and Timer
Player Controller Class – Defines the controller scheme for the level
GameState Class – the game state defines the parameters for the game as Rings, Timer, Lives and Checkpoints

Damage

The damage sets a KillZ area, KillZ is where in the Z axis the character would take damage, normally a kill point, most used for bottomless pits, but can be used for other types of damage. Also the player don't need to be locked to this option, since the Damage Volumes serve to a much dynamic purpose.

Lightmass

The lightmass tab controls some important controls over the light. Is here that the user can determine if it want to use Baked light or Dynamic Global Illumination (Indirect Illumination) .

World Settings

This is where you set the title of the level, that will be a detail. This is not a necessary set up.

Force No Precomputed Lighting – Determines if wants to use baked light (turn off) or Dynamic GI (Turn On – UE4.3 or higher), note that only video cards with Shader Model 5 can use this light setting, it also doens't need to set anything up like baked light (lightmass). Is just set up put a movable Directional Light with Affect Dynamic Indirect Light turned on and Indirect Lighting Intensity and a Post Processing Volume with settings on indirect illumination. Also it needs the following line of code in a new line on the ConsoleVariables.ini file on the config folder of the engine: "r.LightPropagationVolume=1". *For more details, check Unreal Engine 4 documentation.*

Editor Preferences

The editor preferences give some options that will work in all UE4 projects, here, the most important are outlined. To find the Editor Preferences, is under Edit> Editor Preferences.

Loading and saving

Auto Save – leave auto save as the default configuration is highly recommended. It saves all the content of the content browser and open maps.

Miscellaneous

Developer Tools – Turn On “Show Frame Rate and Memory” is highly recommended, it will have these stats above the main ribbon, not clutching the interface and great to track the game speed and spent memory for control and optimizations.

Region and Language

Internationalization – Change the Editor language, current suported languages in the time of this document update is Japanese and Korean.

Play

Play in New Window – Set window size or common windows sizes for play in a new window

Play in Standalone Game – Set window size or common windows sizes for play in standalone game

Project Settings

The project settings give some options that will work in the current UE4 project. To find the Project Settings, is under Edit> Project Settings.

Description – the user can input some information about the project as publisher, site and contact. Also version, ID and legal (if applies).

Maps & Modes – Set the Default start map and the default Game Mode.

Physics – Set the update per frame of the physics engine, use substep for complex physics calculations, else do not change anything. *See documentation for details.*

Windows – Change the Splash Screen and Game Icon here.

Unreal Engine 4 Tips and Tricks

Know these tricks to save you from some troubles and work better on the engine.

Folder Organization and Redirections - When moving files to folder, leftovers references are left and the folder cannot be deleted. To fix this, right click and “Fix Up Redirectors in Folder”, it will fix the redirections and then delete the folder. After this, the folder may remain in Windows Explorer, but now it will be able to be deleted. Do not delete folders using Windows Explorer without doing the Fixing unless you really want to delete everything that was inside or linked with the original folder.

Classes to Blueprints

The documentation to how to make c++ classes editable by blueprints in depth
https://docs.unrealengine.com/latest/INT/Engine/Blueprints/TechnicalGuide/ExtendingBlueprints/index.html

TERMINOLOGY

Objects – instances of classes that inherit from the UObject class, so, all instances of the engine are Objects.
Class – The coded objects of the engine(or programming language) that can be instanced to create a hierarchy
Actor – Instances of classes that are derived from AActor class, can be placed and spawned in the world, also can be containers that hold special objects called components
AActor – Base class of all gameplay objects that can be placed in the world
Component – Actor Subobjects that can be instanced and swapped across actors, different components can be used to say how the actor will move, render, etc.
UActorComponent – the base component. Can be included as a part of an actor

USceneComponent – a physical thing that can exist in the world. They have transform commands, meaning they can move around and interact. Also join in an attachment hierarchy

UPrimitiveComponent – a component that not only exist, but represent something, like a mesh or particle system, many of the interesting physics and collision settings are here