Author: Hong Luu
2017-05-27

- **Added instructions on download, and create project**

2017-06-25

- **Added instructions collision boxes**
- **Added how to connect C++ class to blueprints**
- **Started on save/load text files**

This document explains how to install and use Unreal Engine and includes problems along with possible solutions.

List of acronyms within this document	
OS	**Operating System**

Recommended Tutorial

- https://www.udemy.com/unrealcourse/

To download Unreal Engine

- Go to https://www.unrealengine.com/download
- Select your OS

To Open/Create Unreal Project

- Load Unreal Engine Launcher
- On left side, click “Launch”
- On middle of screen, there should be a list Unreal Engine versions installed (typically just one). Click “Launch” on the version you want to run.
- After Editor finishes loading, there is a choice to load existing project or create new one
- To create new project (C++):
    - Click New Project tab on top
    - Choose between blue print or C++

        - For the sake of this tutorial, select C++
    - Select a template under C++ tab
    - Select the following options:
        - Desktop/Console
        - Maximum Quality
        - With Starter Kit
    - Choose your folder for your project and give it a name
    - Click Create Project
- To open an existing project:
    - Click Projects tab on top
    - If project is not listed in middle section, then click “Browse” on lower right
    - Locate project folder, and within the project there should be a file with extension *.uproject

To attach C++ code to an Object

- Select the gameobject from the perspective view
    - There should be a window titled Details that displays the instance of the gameobject
- Go to Content Browser
    - Click C++ Classes
    - Traverse through folders to find the code
    - Drag and drop code object to gameobject in the Details window
- You may need to adjust the Detail window to view the code object under the gameobject
    - Suggestion: Hover below the instance of object in the Details window and move mouse slowly downward until an arrow icon appears and let you adjust window pane sizes
-

Have collision box to print string onto the screen in blueprints

- Create blueprint as a class
    - On toolbar on top, click Blueprints -> create new blueprint class
    - Pick parent class to inherit properties
        - In this example, we’ll use actor
    - Enter new name for blueprint class and select folder to store blueprint class
- Adding a prop to blueprint
    - In the blueprint window, click on Viewport
    - For this example, we’ll use a chair from the starter content
    - On explorer window, go to Starter Content > Props > SM_Chair
    - Click and drag chair onto the blueprint edit window, on Components (upper left) section
        - Drop chair under default scene root
    - Now you can replace default scene root with the chair

        - drag the chair from under default scene root and move it on the default scene root
- Add collision box
    - In Components section, click Add Components
    - Type in “Box Collision”
    - Rename box collision
    - Resize collision box as needed
    - Right click collision box and mouse over Add Event > Add OnComponentBeginOverlap
    - On the white arrow pointing right, click and drag a line out
    - Enter “Print String” to produce a print function
    - Enter text in String field
    - Click save and compile on the toolbar on top of window
- Testing
    - Go to your level editor
    - On Content browser, locate the newly made blueprint class
    - drag and drop blueprint class object to viewport
    - Click play
    - Maneuver player to chair
    - Text should appear on console and in screen upper left

Getting blueprint to run C++ function

- The assumption is your project is in C++ mode
    - If your project is in Blueprint mode, you will need to make modification to convert to c++ project
- Reference
    - https://www.youtube.com/watch?v=4CgCJsThsWU
- Add c++ function to your project
    - In content Browser window, C++ Classes > your_project_name
    - Right click on your project name and select New C++ class
    - Select parent type
        - For my example, I used GameStateBase (to save games)
        - Give it a name
    - In your .h file
        - Have this code:
            - GENERATED_BODY()
            - UFUNCTION(BlueprintCallable, Category="SomeName")
            - static void someFunction();
        - Example:
            - GENERATED_BODY()
            - UFUNCTION(BlueprintCallable, Category="MySaveGame")
            - static void runSaveGame();

    - In your .cpp file
        - have this code:
            - void AMyGameStateBase::runSaveGame() {
            - UE_LOG(LogTemp, Warning, TEXT("runSaveGame!!"));
            - }
    - Rebuild and Compile project
        - You may need to reload your entire project
- Create collision box
    - Follow steps above to generate collision box component to your blueprint class upto part where you created the box collider and renamed it
- Adding event to trigger c++ function
    - Right click on collision box
    - Mouse to Add Event > Click on AddComponentBeginOverlap
    - On Event graph window, from OnComponentBeginOverlap
        - click and hold from white arrow
        - drag to right and release to draw line
        - Enter your BlueprintCallable category (from your .h file)
            - in my case, it is “My Save Game”
- Result
    - Now everytime I step on trigger box, the function “runSaveGame” will be executed

Save/Load from text files

- Reference
    - Text files
        - https://www.youtube.com/watch?v=wemG-YI2iag
    - Read/Write to files via C++
        - https://www.youtube.com/watch?v=Iho2EdJgusQ
        - https://wiki.unrealengine.com/File_Management,_Create_Folders,_Delete_Files,_and_More#Read.2FWrite_to_Files
- Issue: Linking Buttons pressed to run save function doesn’t work
    - Other solutions: Using a trigger box around blueprint game object to run save function works fine
        - Instructions on generating collision boxes can be found above
- Creating directory to save your files using Unreal C++
    - Assumption: This C++ save class is accessed properly, meaning your project is calling this class to save and this class will do the rest.
    - In our example, we’ll start at project root directory and create a SaveData folder
        - FString saveDirectory = FString(FPaths::GameDir()) + FString("/SaveData");
    - Create a filename. In our example, we’ll create a filename with a number to modify filenames dynamically.
        - int fileNum = 0;

        - FString fileName = FString("MySaveFile") +
        - FString::FormatAsNumber(fileNssssssssssum) +
        - FString(".txt");
    - Using platform independent
        - IPlatformFile& platformFile = FPlatformFileManager::Get().GetPlatformFile();
    - Create directory
        - if (platformFile.CreateDirectoryTree(*saveDirectory))
        - {
        - // Get absolute file path
        - FString absoluteFilePath = saveDirectory + "/" + fileName;
        -
        - // file doesn't already exist
        - if (!platformFile.FileExists(*absoluteFilePath))
        - {
        - FFileHelper::SaveStringToFile(<your_text>, *absoluteFilePath);
        - }
        - }

Creating Save Button

- Assumption: There exists a C++ class that saves data to a file
    - For our example, we'll say there is a function call runSaveGame in that class
- Reference: https://www.youtube.com/watch?v=dOCNKMTBL-A
-

List of problems using Unreal

- Loading Level (or Maps) results in a blank screen on the editor
    - Solution:
- Loading Project Editor does not load the map you want to work on
    - Solution:
        - In Unreal Project, Edit > Project Settings
        - In Project Settings, under Project, click Maps & Modes
        - In Default Maps, select the map you want to start up
-