# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday of Week 1 (Spring semester).

# Objectives

The objectives for this Project are three-fold:

* To implement tracing/logging functionality
* To implement scene loading and unloading
* To implement the DigiPen Graphics Library demo code as a new scene

# Description

For this project, you have been provided with a Visual Studio project and a set of code (.c) and header (.h) files that comprise a bare-bones game engine framework. Your job is to add the code necessary to implement the three features identified in the project objectives, above.

Instructions have been provided below on how to complete each of the three steps required for this project. However, you should refer to the lecture materials for specific implementation details.

# Prior Work

Students repeating CS230 are ***required*** to use the project materials provided during the current semester. They are also ***encouraged*** to complete the projects “from scratch”, but are allowed to reference their prior work when implementing the projects.

# Files

NOTE: You may not change the public interface of the header files (.h) that are provided in Project 0, except as expressly directed in the instructions below. Should you modify these header files in any way, exercise extreme caution, as adding, removing, or modifying the public interface will result in a penalty to your project grade.

## stdafx.c/.h

* These files are used to generate a “pre-compiled header”, the use of which can significantly improve project build times by pre-compiling commonly used header files.
  + You are not allowed to change these files in CS230. However, do consider using pre-compiled header files in all future coding projects.
* The .c file is used to “create” the pre-compiled header (“/Yc” compiler option). All other .c files “use” the pre-compiled header (“/Yu” compiler option).
* ***Important Note:*** The stdafx.h file must be included as the ***first include file in every .c file***. You may encounter build errors if you mistakenly place any other header files before this one.

## Main.c

* This module contains the minimal amount of code necessary for the Main Loop.
* There is no need to make any changes to this file for Project 0.

NOTE: Every source file that you create for this course must include a “file header” containing correct Author and DigiPen Copyright information. You are encouraged to use the following “stub” files when creating new .c or .h files. However, if you chose not to do so, then it is your responsibility to ensure that each source file contains the required header information.

## Stub.c/.h

* Stub files for easily creating new source code and header files.
* You must make the following changes to these files for Project 0:
  + Replace the instructor’s “Author” information with your name and student ID:
    - // Author(s): Doug Schilling (dschilling)

## StubScene.c/.h

* Stub files for easily creating new Scene source code and header files.
* You must make the following changes to these files for Project 0:
  + Replace the instructor’s “Author” information with your name and student ID:
    - // Author(s): Doug Schilling (dschilling)

## StubSystem.c/.h

* Stub files for easily creating new System source code and header files.
* You must make the following changes to these files for Project 0:
  + Replace the instructor’s “Author” information with your name and student ID:
    - // Author(s): Doug Schilling (dschilling)

## Engine.c/.h

* *“An engine is the sum of its parts.”*
  + This module combines the individual engine systems together.
  + Avoid the urge to dump engine functionality into this module when working on future GAM projects. Instead, move that functionality into separate systems and add the systems to the engine as demonstrated in Main.c.
* The order of execution of certain function calls can be very important. For example, many engine components have a dependency upon Tracing and Memory Management modules. As a result, these two modules should be initialized first and shutdown last.
* There is no need to make any changes to these files for Project 0.

## BaseSystem.h

* This header file declares a common, abstract interface for use by all systems.
* The BaseSystem structure is declared publicly because it must be included first in every “derived” System structure. This usage will be discussed in future lectures.
* There is no need to make any changes to this file for Project 0.

## PlatformSystem.c/.h

* This module initializes the Windows application and the DigiPen Graphics Library.
* There is no need to make any changes to these files for Project 0.

## Trace.h

* This header file declares the public interface for Tracing/Logging messages to a file.
* There is no need to make any changes to this file for Project 0.

## Trace.c

* This module must open a text file for writing, append messages to the file, and close the file when the engine shuts down.
* You must write TraceMessage() as a *variadic* function for writing trace messages with optional parameters. The declaration for this function must be as follows:
  + void TraceMessage(const char \* formatString, ...)
* You must make the following changes to this file for Project 0:
  + Private Variables:
    - Declare a private variable for storing a file handle, for example:
      * static FILE \* traceFile;
  + TraceInit:
    - Using fopen\_s(), open the file “Trace.log” for writing in text mode (“wt”).
    - If the file failed to open, then you must perform error handling. The exact implementation is up to the student. However:
      * Some form of error message must be written to the console.
        + NOTE: The function, printf(), can be used for this purpose.
      * The program must continue to run properly, without any fatal exceptions in subsequent calls to TraceMessage or TraceShutdown.
        + HINT: You can test this by temporarily setting traceFile = NULL.
  + TraceMessage:
    - Print the given message to the file using the variadic vfprintf\_s() function.
    - Every message must be printed on its own line. Use fprintf\_s() to append a new line after the message.
    - There must be no blank lines between messages written to the file.
  + TraceShutdown:
    - Close the file if-and-only-if the file was opened successfully.
  + ***Pro Tip:*** Verify that the trace.log file contains the following messages before continuing to the next step:

|  |
| --- |
| Engine: Init  PlatformSystem: Init  SceneSystem: Init  Engine: Update  Engine: Render  Engine: Update  Engine: Render  Engine: Exit  SceneSystem: Exit  PlatformSystem: Exit |

## Scene.h

* This header file declares a common, abstract interface for use by all “derived” scenes.
* The Scene structure is declared publicly because it must be included first in every “derived” Scene structure. This usage will be discussed in future lectures.
* There is no need to make any changes to this file for Project 0.

## Scene.c

* This module implements a common, abstract interface for use by all “derived” scenes.
* You must make the following changes to this file for Project 0:
  + Add TraceMessage calls to each of the functions that execute the scene-specific State functions (6 total). *The format string must exactly match that provided in the comments.* For example, the following code generates the correct trace message for SceneLoad():
    - TraceMessage("%s: Load", scene->name);
  + ***Pro Tip:*** Verify that the trace.log file contains the following messages before continuing to the next step:

|  |
| --- |
| Engine: Init  PlatformSystem: Init  SceneSystem: Init  Engine: Update  Stub: Init  Stub: Update  Engine: Render  Stub: Render  Engine: Update  Stub: Exit  Engine: Render  Engine: Exit  SceneSystem: Exit  PlatformSystem: Exit |

## SceneSystem.c/.h

* This module implements the Scene System, which manages transitions between scenes.
* You must make the following changes to this file for Project 0:
  + SceneSystemUpdate:
    - Add code to correctly handle the “isRestarting” condition.
    - See the “Engine Flow” lecture notes for implementation instructions.
  + ***Pro Tip:*** Verify that the trace.log file contains the following messages before continuing to the next step:

|  |
| --- |
| Engine: Init  PlatformSystem: Init  SceneSystem: Init  Engine: Update  Stub: Load  Stub: Init  Stub: Update  Engine: Render  Stub: Render  Engine: Update  Stub: Exit  Stub: Unload  Engine: Render  Engine: Exit  SceneSystem: Exit  PlatformSystem: Exit |

* + Change the starting (Default) scene from "Stub" to "Demo":
    - SceneSystemSetNext(StubSceneGetInstance());
  + ***Pro Tip:*** Verify that the trace.log file contains the following messages before continuing to the next step (Note: The “…” represents four repeating “Update” and “Render” messages):

Engine: Init

PlatformSystem: Init

SceneSystem: Init

Engine: Update

Demo: Load

Demo: Init

Demo: Update

Engine: Render

Demo: Render

…

Engine: Update

Demo: Exit

Demo: Unload

Engine: Render

Engine: Exit

SceneSystem: Exit

PlatformSystem: Exit

## DemoScene.c/.h

* These two files will be used to implement the DigiPen Graphics Library demo.
* You must make the following changes to this file for Project 0:
  + ***Download the DigiPen Graphics Library (DGL) Demo .zip file from Moodle or Teams***
  + Open/edit the demo source file, Main.c.
  + Locate the code (in Main.c) for each of the following sections and copy the code into the indicated location in DemoScene.c:
    - Private Constants:
      * Declare Constants
    - Private Variables:
      * Declare Variables
    - DemoSceneLoad:
      * Create Meshes
      * Load/Create Textures
    - DemoSceneInit:
      * Set the background color and blend mode.
    - DemoSceneUpdate:
      * Update the camera's position.
      * Update the triangular ("Spaceship") mesh's position.
      * Update the triangular ("Spaceship") mesh's alpha value.
      * Update the Ball texture's UV offsets.
      * Update the rotation value.
    - DemoSceneRender:
      * Update the camera's position every frame.
      * Display a simple, colored mesh.
      * Draw a simple, colored mesh with a red (50%) tint.
      * Draw a simple, textured mesh using the Ball texture.
      * Draw a simple, textured mesh using the 2x2 texture.
      * Draw a triangular, colored mesh.
      * Draw a 4-segment line strip.
    - DemoSceneExit:
      * No changes are required here.
    - DemoSceneUnload:
      * Free All Meshes
      * Free All Textures
  + ***Pro Tip:***
    - Verify that all objects and lines are drawn correctly.
    - Verify that all inputs update the objects in the scene correctly.
    - ***Important Note:*** There should be no DGL\_System\* functions copied into this file.
  + Make the following, additional changes:
    - DemoSceneInit:
      * Add code to set the following variables to their starting values:
        + posCamera
        + posSpaceship
        + uvOffset
        + alpha
        + rotation
    - DemoSceneUpdate:
      * Add code to restart the level when the ‘0’ key is *triggered* (when the key changes state from not pressed to pressed).
  + ***Pro Tip:***
    - Verify that the scene is restored to its original state when the ‘0’ key is triggered.

# Submission Requirements

* The project must build cleanly, with no errors or warnings.
* Once the assignment has been completed, create a submission .zip file by performing the following steps:
  + Select the following files and folders:
    - “Assets” folder
    - “DGL” folder
    - “Source” folder
    - Project0.sln
    - Project0.vcxproj
    - Project0.vcxproj.filters
  + Right-click on one of these files and select the option:
    - “Send to” -> “Compressed (zipped) folder”
  + The resultant .zip file **must not** include any extraneous files or folders, including but not limited to the following Visual Studio folders:
    - Folders: .vs, “Debug”, “Release”, “x64”
  + Rename the resultant .zip file using the following naming convention:
    - CS230S25<section letter>\_<Login ID>\_Project0.zip
      * Example: CS230S25A\_john.doe\_Project0.zip
* Upload the submission .zip file via the Moodle page for your CS230 section (A or B)
* ***Important Note:*** Once your submission has been uploaded, it is ***highly recommended*** that you verify that the submission process was completed successfully, by performing the following steps:
  + Return to the Moodle page for your section (A or B).
  + Click on the assignment submission link.
  + Download the .zip file to your computer.
  + Unzip the contents of the .zip file into your project folder.
  + Open up the Visual Studio solution file.
  + Clean and rebuild the project.
  + Test the executable (within Visual Studio is fine).

# Assignment Grading Guidelines

* A -25% penalty will be applied for each week or portion of a week that the project is submitted late.
* A -25% penalty will be applied to any submissions that utilize the project materials provided in a previous semester.
* A -10% penalty will be applied to any submissions that are performed incorrectly (e.g. incorrect .zip format, submitting extraneous files, etc.).
* A -10% penalty will be applied to any submissions that do not conform to the naming convention specified in the Submission Requirements section.

# Project 0 Testing

Below is the output that you should find in your “Trace.log” file after running the completed project. The four lines between the ellipses are repeated every game loop so expect the output file to grow in length very quickly.

If the output at the beginning, middle, and ending of the file does not match the following, then points will be deducted from the project grade:

Engine: Init

PlatformSystem: Init

SceneSystem: Init

Engine: Update

Demo: Load

Demo: Init

Demo: Update

Engine: Render

Demo: Render

…

Engine: Update

Demo: Update

Engine: Render

Demo: Render

…

Engine: Update

Demo: Exit

Demo: Unload

Engine: Render

Engine: Exit

SceneSystem: Exit

PlatformSystem: Exit