# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 1.

# Objectives

The objectives for this Project are three-fold:

* To implement tracing/logging functionality
* To implement game state loading and unloading
* To implement Alpha Engine demo code as a new game state

# 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.

# 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.

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

Engine.c

* *“An engine is the sum of its parts.”*
  + This module combines the individual engine components together. Avoid the urge to dump engine functionality into this module when working on future GAM projects. Instead, move that functionality into separate modules and call the necessary functions from here
* The order of execution of certain function calls can, often, 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 this file for Project 0

System.c

* This module initializes the Windows application
* There is no need to make any changes to this file for Project 0

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)
* The stdafx.h file must be included as the ***first include file in every .c file***. You will likely encounter build errors if you mistakenly place any other header files before this one

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 using the function, AESysPrintf() (include “AEEngine.h”)
      * 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() function
    - Every message must be printed on its own line
    - There must be no blank lines between messages
  + TraceShutdown:
    - Close the file if-and-only-if the file was opened successfully
* Verify that the trace.log file contains the following messages before continuing to the next step:

|  |
| --- |
| Engine: Init  GSM: Init  Engine: Update  GSM: Update  Engine: Update  GSM: Update  Engine: Shutdown  GSM: Shutdown |

GameStateManager.c

* This module manages transitions between game states
* This module is game-independent
* You must make the following changes to this file for Project 0:
  + GameStateManagerUpdate:
    - Add code to correctly handle the “GsRestart” command
    - See the “Engine Flow” lecture notes for additional information

GameStateTable.h

* This header contains the GameStates enum.
* You must make the following changes to this file for Project 0:
  + Add “GsDemo” to the “Normal Game States” section
  + Set “GsInitial” equal to “GsDemo”

GameStateTable.c

* This module provides the GameStateManager with access to the game-specific game state modules without exposing the game implementation
* A data table contains function pointers for every game state
* You must make the following changes to this file for Project 0:
  + Add entries for the “Demo” game state to the table
    - Hint: Use the existing entry for the “Stub” game state as an example.
  + Add TraceMessage calls to each of the GameStateExecute functions (5 total). *The format string must exactly match that provided in the comments.* For example, the following code generates the correct trace message for GameStateExecuteLoad().
    - TraceMessage("%s: Load", GameStateTab[gameState].gameStateName);
* Verify that the trace.log file contains the following messages before continuing to the next step (Note: The “…” represents three repeating “Update” messages):

|  |
| --- |
| Engine: Init  GSM: Init  Engine: Update  GSM: Update  Demo: Load  Demo: Init  Demo: Update  …  Engine: Update  GSM: Update  Demo: Shutdown  Demo: Unload  Engine: Shutdown  GSM: Shutdown |

GameStateDemo.c/.h

* These two files will be used to implement the Alpha Engine demo
* You must make the following changes to this file for Project 0:
  + ***Download the AlphaEngineDemo.zip file from Moodle***
  + Open/edit the demo source file, Main\_CS230.c
  + Locate the code (in Main\_CS230.c) for each of the following sections and copy the code into the indicated location in GameStateDemo.c:
    - Declare Variables
    - Create Meshes
    - Load/Create Textures
    - Set Background Color and Blend Mode
    - Update Spaceship Position
    - Update Alpha
    - Update Camera Position
    - Update Texture Offset
    - Draw Line Strip and Objects
    - Unload All Textures
    - Free All Meshes
  + Add code in GameStateDemoInit to set the following variables to their starting values:
    - obj1X, obj1Y
    - obj1texX, obj1texY
    - alpha
    - camX, camY
      * NOTE: You will need to call AEGfxSetCamPosition to correctly restore the camera’s starting position
  + Add code in GameStateDemoUpdate to restart the level when the ‘0’ key is *triggered* (when the key changes state from not pressed to pressed)
* Verify that all objects and lines are drawn correctly
* Verify that all input updates the objects in the scene correctly
* Verify that the scene is restored to its original state when the ‘0’ key is triggered

GameStateStub.c/.h

* Stub files for easily creating new game state modules and header files
* You should update these files with the correct “Author” information

Stub.c/.h

* Stub files for easily creating new modules and header files.
* You should update these files with the correct “Author” information.

# 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:
    - “AE” folder
    - “Assets” 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 of the following Visual Studio generated folders and files:
    - Folders: “Debug”, “Release”, “ipch”
    - Files (\*.db, \*.sdf, \*.opendb)
  + Rename the resultant .zip file using the following naming convention:
    - CS230S22<section letter>\_<Login ID>\_Project0.zip
      * Example: CS230S22A\_john.doe\_Project0.zip
* Upload the submission .zip file via the Moodle page for your CS230 section (A or B)
* 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 -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 three 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 and ending of the file does not match the following, then points will be deducted from the project grade:

Engine: Init

GSM: Init

Engine: Update

GSM: Update

Demo: Load

Demo: Init

Demo: Update

Engine: Update

GSM: Update

Demo: Update

…

Engine: Update

GSM: Update

Demo: Update

…

Engine: Update

GSM: Update

Demo: Shutdown

Demo: Unload

Engine: Shutdown

GSM: Shutdown