# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 2.

# Objective

The objective of this assignment is to create a game engine that can switch between two levels and restart a level, while displaying the correct order of state function calls (Load, Init, Update, Shutdown, and Unload).

A secondary objective is to implement a module for handling tracing/logging. This is a simple debugging feature that you should consider incorporating into your future GAM projects.

# Description

In this assignment, you will implement the following features:

* Basic engine flow
* Tracing/logging functionality
* Two simple game states

Level 1

Level 2

# Game States

Two levels should be implemented in this assignment

* Each level will have its own state functions to load, initialize, update, shutdown, and unload its data
* The functionality required in each of these functions are detailed below.

Progress through each of the game states must be recorded to a trace log file. For example:

* Loading level 1 must append “Level1: Load”
* Initializing level 1 must append “Level1: Init”
* Updating level 1 must append “Level1: Update”
* Shutting down level 1 must append “Level1: Shutdown”
* Unloading level 1 must append “Level1: Unload”
* Loading level 2 must append “Level2: Load”
* Initializing level 2 must append “Level2: Init”
* Updating level 2 must append “Level2: Update”
* Shutting down level 2 must append “Level2: Shutdown”
* Unloading level 2 must append “Level2: Unload”

# Files

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 1.

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. Instead, move that functionality into separate modules and call the necessary functions from here.
* The order of execution of certain function calls can sometimes 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 1.

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 1:
  + Private Variables:
    - Declare a private variable for storing a file handle
      * static FILE \* traceFile;
  + TraceInit:
    - 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
      * The program must continue to run properly, without any fatal exceptions in subsequent calls to TraceMessage or TraceShutdown.
      * HINT: You should test your code by setting traceFile = NULL
  + TraceMessage:
    - Print the given message to the file
    - Also print the message to the console window
    - 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

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 1:
  + GameStateManagerUpdate:
    - Add code to correctly handle the “GsRestart” command
    - See the “Engine Flow II” lecture notes for additional information

GameStateTable.h

* This header contains the GameStates enum.
* You must make the following changes to this file for Project 1:
  + Add “GsLevel1” and “GsLevel2” to the “Normal Game States” section
  + Remove “GsStub” from the “Normal Game States” section
  + Set “GsInitial” equal to “GsLevel1”

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 1:
  + Add entries for the “Level1” and “Level2” game states to the table
    - Hint: Use the existing entry for the “Stub” game state as an example.
  + Remove the entry for the “Stub” game state from the table
  + 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);

Stream.h

* This header file declares the public interface for reading data from a file. You are responsible for creating the associated source file (.c) and implementing the required functionality, as outlined in the header file and the lecture notes.

GameStateStub.c/.h

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

GameStateLevel1.c/.h

* You will need to create these files and add them to the project
  + Hint: Use copies of the existing GameStateStub.c & .h files to get started
* You must make the following changes to this file for Project 1:
  + Private Variables:
    - Create a private variable, of type int, called numLives1
      * static int numLives1 = 0;
  + GameStateLevel1Load:
    - Read the initial value of “numLives1” from a file named “Level1\_Lives.txt” (provided in the “Data” folder).
  + GameStateLevel1Update:
    - Decrement “numLives1” by 1
    - When “numLives1” reaches 0, the game should switch to Level2

GameStateLevel2.c/.h

* You will need to create these two files and add them to the project.
  + Hint: Use the existing GameStateStub.c & .h files to get started
* You must make the following changes to this file for Project 1:
  + Private Variables:
    - Create a private variable, of type int, called numLives
      * static int numLives = 0;
    - Create a private variable, of type int, called numHealth
      * static int numHealth = 0;
  + GameStateLevel2Load:
    - Read the initial value of “numLives” from a file named “Level2\_Lives.txt” (provided in the “Data” folder).
  + GameStateLevel2Init:
    - Read the initial value of “numHealth” from a file named “Level2\_Health.txt” (provided in the “Data” folder).
  + GameStateLevel2Update:
    - Decrement “numHealth” by 1
    - When “numHealth” reaches 0
      * Decrement “numLives” by 1
      * When “numLives” reaches 0, the game should quit (GsQuit).
      * Otherwise, restart the level (GsRestart).

# 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:
    - “Data” folder
    - “Source” folder
    - Project1.sln
    - Project1.vcxproj
    - Project1.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:
    - CS230SU19<section letter>\_<Login ID>\_Project1.zip
      * Example: CS230SU19A\_john.doe\_Project1.zip
* Upload the submission .zip file via the Moodle page for your CS230 section (A or B)
* It is your responsibility to ensure that the project was submitted properly. Once the 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 home 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 an empty folder
  + Open up the Visual Studio solution file
  + Clean and rebuild the project
  + Test the executable

# 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 1 Testing

Below is the output that you should find in your “Trace.log” file after running the application, assuming:

* Level1\_Lives.txt contains 3
* Level2\_Health.txt contains 2
* Level2\_Lives.txt contains 2

If your output does not match the following, then points will be deducted from the project grade:

Engine: Init

GSM: Init

Engine: Update

GSM: Update

Level1: Load

Level1: Init

Level1: Update

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Shutdown

Level1: Unload

Level2: Load

Level2: Init

Level2: Update

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Shutdown

Level2: Init

Level2: Update

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Shutdown

Level2: Unload

Engine: Shutdown

GSM: Shutdown