# Due Date

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

# Objective

The objectives for this Project are two-fold:

* To implement a game engine flow that can switch between three scenes and restart a scene, while executing the correct order of state function calls (Load, Init, Update, Render, Exit, and Unload).
* To implement a module for performing mathematical operations on the DGL\_Vec2 structure. These mathematical operations will be used in the third scene.

# Description

In this assignment, you will implement the following features:

* Basic engine flow
* Three simple scenes (AKA “Levels”)
* Vector2D math

Level 1

Level 2

Sandbox

# Scenes

Three scenes should be implemented in this assignment

* Each scene will have its own state functions to load, initialize, update, render, exit, and unload its data.
* The functionality required in each of these functions is detailed below.

Progress through each of the scenes 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”
* Rendering Level 1 must append “Level1: Render”
* Exiting Level 1 must append “Level1: Exit”
* Unloading Level 1 must append “Level1: Unload”

# Files

NOTE: You may not change the public interface of the header files (.h) that are provided in Projects 0 & 1, 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.

For Project 1 you will need to copy certain files from Project 0. To do so, simply drag-and-drop the files from Project 0 into the source folder for Project 1. However, *do NOT overwrite any existing files in the Project 1 source folder!*

The files to be copied include: DemoScene.c/.h, BaseSystem.h, Engine.c/.h, Main.c, PlatformSystem.h, Scene.c/.h, SceneSystem.c/.h, stdafx.c/.h, Stub.c/.h, StubScene.c/.h, StubSystem.c/.h, targetver.h, and Trace.c/.h.

## stdafx.h

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

## PlatformSystem.c

* This module has been modified to display a new window title:
  + AESysSetWindowTitle("CS230 Project 1 – Engine Flow");
* There is no need to make any changes to this file for Project 1.

## Stream.h

* This header file declares the public interface for reading data from a file.
* You must create the associated source file (.c) and implement the required functionality, as outlined in the header file and the lecture slides.

## Level1Scene.c/.h

* You must create these two files and add them to the project.
  + Hint: Use copies of the existing StubScene.c/.h files to get started.
* You must make the following changes to the .c file for Project 1:
  + Private Structures:
    - Add a new variable to the Level1Scene structure:
      * int numLives;
  + Instance Variable:
    - Initialize the new variable to 0 in the structure initializer.
  + Level1SceneLoad:
    - Read the initial value of “numLives” from a file named “Level1\_Lives.txt” (located in the “Data” folder).
      * Open the file using StreamOpen().
      * If the stream was opened successfully,
        + Read the first integer from the file using StreamReadInt().
        + Close the file using StreamClose().
  + Level1SceneUpdate:
    - Decrement “numLives” by 1.
    - If “numLives1” <= 0
      * Switch the Scene System to “Level2”.

## SceneSystem.c

* This module implements the Scene System, which manages transitions between scenes.
* You must make the following changes to this file for Project 1:
  + Change the starting (Default) scene from "Demo" to "Level1":
    - SceneSystemSetNext(DemoSceneGetInstance());

**Level2Scene.c/.h**

* You must create these two files and add them to the project.
  + Hint: Use copies of the existing StubScene.c/.h files to get started.
* You must make the following changes to the .c file for Project 1:
  + Private Structures:
    - Add two new variables to the Level2Scene structure:
      * int numLives;
      * int numHealth;
  + Instance Variable:
    - Initialize the new variables to 0 in the structure initializer.
  + Level2SceneLoad:
    - Read the initial value of “numLives” from a file named “Level2\_Lives.txt” (located in the “Data” folder).
      * Hint: See the steps listed under Level1SceneLoad, above.
  + Level2SceneInit:
    - Read the initial value of “numHealth” from a file named “Level2\_Health.txt” (provided in the “Data” folder).
      * Hint: See the steps listed under Level1SceneLoad, above.
  + Level2SceneUpdate:
    - Decrement “numHealth” by 1.
    - If “numHealth” <= 0:
      * Decrement “numLives” by 1.
      * If “numLives” > 0
        + Restart the scene.
      * Else
        + Switch the Scene System to “Sandbox”.

**Vector2D.h**

* This header file declares the public interface for creating and manipulating 2D vectors.
* You must create the associated source file (.c) and implement the required functionality, as outlined in the header file and the lecture slides.

**SandboxScene.c/.h**

* In this new scene, you must call each of the Vector2D functions and write the resultant values to the trace file using one of the two new private functions listed below. If you’ve followed these instructions properly and implemented the Vector2D functions correctly, then the contents of Trace.log should match the output listed in the “Project 1 Testing” section, below.
* You must create these two files and add them to the project.
  + Hint: Use copies of the existing StubScene.c/.h files to get started.
* You must make the following changes to the .c file for Project 1:
  + SandboxSceneTraceFloat:
    - Create a new *private* function for writing float values to the trace file:
      * static void SandboxSceneTraceFloat(const char\* text, float f);
    - The trace message should be formatted as "Vector Test: %s = %f"
  + SandboxSceneTraceVector:
    - Create a new *private* function for writing DGL\_Vec2 values to the trace file:
      * static void SandboxSceneTraceVector(const char\* text, const DGL\_Vec2\* v);
    - The trace message should be formatted as "Vector Test: %s = [%f, %f]"
  + SandboxSceneUpdate:
    - Open the file, “Data/VectorTests.txt”, using StreamOpen()
    - If the stream was opened successfully,
      * Create a *single* DGL\_Vec2 variable for use in the following tests. After each step you must write out either a DGL\_Vec2 or float value to the trace file, as appropriate
        + Call Vector2DZero
        + Call Vector2DSet with the parameters 1.0f and 1.5f
        + Call Vector2DNeg
        + Call Vector2DAdd
        + Call Vector2DSub
        + Call StreamReadVector2D, passing the vector
        + Call Vector2DNormalize
        + Call StreamReadFloat to initialize a “scale” variable
        + Call Vector2DScale
        + Call Vector2DScaleAdd
        + Call Vector2DScaleSub
        + Call Vector2DLength
        + Call Vector2DSquareLength
      * Create two DGL\_Vec2 variables for use in the following tests. After each step you must write out either a DGL\_Vec2 or float value to the trace file, as appropriate
        + Call StreamReadVector2D, passing the 1st vector
        + Call StreamReadVector2D, passing the 2nd vector
        + Call Vector2DDistance
        + Call Vector2DSquareDistance
        + Call Vector2DDotProduct
      * Create a *single* DGL\_Vec2 variable for use in the following tests. Optionally, you may reuse one of the DGL\_Vec2 variables created earlier. After each step you must write out either a DGL\_Vec2 or float value to the trace file, as appropriate
        + Call StreamReadFloat to read an angle, in degrees
        + Call Vector2DFromAngleDeg
        + Call StreamReadFloat to read an angle, in radians
        + Call Vector2DFromAngleRad
        + Call Vector2DToAngleRad, passing the result from the previous test
      * Close the file using StreamClose()
    - Tell the Scene System to terminate the program by setting the next scene to NULL

# 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
    - “Data” folder
    - “DGL” 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:
    - Folders: .vs, “Debug”, “Release”, “x64”
  + Rename the resultant .zip file using the following naming convention:
    - CS230S23<section letter>\_<Login ID>\_Project1.zip
      * Example: CS230S23A\_john.doe\_Project1.zip
* Upload the submission .zip file via the Moodle page for your CS230 section (A or B)
* ***Important Note:*** It is your responsibility to ensure that the project was submitted properly. 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 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 the Visual Studio solution file
  + Clean and rebuild the project
  + Verify that the program runs correctly (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 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

PlatformSystem: Init

SceneSystem: Init

Engine: Update

Level1: Load

Level1: Init

Level1: Update

Engine: Render

Level1: Render

Engine: Update

Level1: Update

Engine: Render

Level1: Render

Engine: Update

Level1: Update

Engine: Render

Level1: Render

Engine: Update

Level1: Exit

Level1: Unload

Level2: Load

Level2: Init

Level2: Update

Engine: Render

Level2: Render

Engine: Update

Level2: Update

Engine: Render

Level2: Render

Engine: Update

Level2: Exit

Level2: Init

Level2: Update

Engine: Render

Level2: Render

Engine: Update

Level2: Update

Engine: Render

Level2: Render

Engine: Update

Level2: Exit

Level2: Unload

Sandbox: Load

Sandbox: Init

Sandbox: Update

Vector Test: Vector2DZero = [0.000000, 0.000000]

Vector Test: Vector2DSet = [1.000000, 1.500000]

Vector Test: Vector2DNeg = [-1.000000, -1.500000]

Vector Test: Vector2DAdd = [-2.000000, -3.000000]

Vector Test: Vector2DSub = [0.000000, 0.000000]

Vector Test: StreamReadVector2D = [3.000000, 4.000000]

Vector Test: Vector2DNormalize = [0.600000, 0.800000]

Vector Test: StreamReadFloat = 5.000000

Vector Test: Vector2DScale = [3.000000, 4.000000]

Vector Test: Vector2DScaleAdd = [18.000000, 24.000000]

Vector Test: Vector2DScaleSub = [72.000000, 96.000000]

Vector Test: Vector2DLength = 120.000000

Vector Test: Vector2DSquareLength = 14400.000000

Vector Test: StreamReadVector2D = [1.000000, 2.000000]

Vector Test: StreamReadVector2D = [5.000000, -1.000000]

Vector Test: Vector2DDistance = 5.000000

Vector Test: Vector2DSquareDistance = 25.000000

Vector Test: Vector2DDotProduct = 3.000000

Vector Test: StreamReadFloat = 135.000000

Vector Test: Vector2DFromAngleDeg = [-0.707107, 0.707107]

Vector Test: StreamReadFloat = -0.785398

Vector Test: Vector2DFromAngleRad = [0.707107, -0.707107]

Vector Test: Vector2DToAngleRad = -0.785398

Engine: Render

Sandbox: Render

Engine: Update

Sandbox: Exit

Sandbox: Unload

Engine: Render

Engine: Exit

SceneSystem: Exit

PlatformSystem: Exit