# 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 create a game engine that can switch between three levels and restart a level, while executing the correct order of state function calls (Load, Init, Update, Shutdown, and Unload).
* To implement a module containing a Vector2D structure and specific mathematical operations associated with this structure. These mathematical operations will be used in the third level.

# Description

In this assignment, you will implement the following features:

* Basic engine flow
* Three simple game states
* Vector2D math

Level 1

Level 2

Sandbox

# Game States

Three 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

NOTE: You may not change the public interface of the header files (.h) that are provided in Project 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: Engine.c/.h, GameStateDemo.c/.h, GameStateManager.c/.h, GameStateStub.c/.h, GameStateTable.c/.h, Main.c, stdafx.c/.h, Stub.c/.h, targetver.h, and Trace.c/.h.

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

* This module combines the individual engine components together
* There is no need to make any changes to this file for Project 1

System.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

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 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

GameStateTable.h

* In Project 0, this file was modified to add a “Demo” game state
  + Do NOT remove “GsDemo”
* You must make the following changes to this file for Project 1:
  + Add “GsLevel1”, “GsLevel2”, and “GsSandbox to the “Normal Game States” section
  + Set “GsInitial” equal to “GsLevel1”

GameStateTable.c

* In Project 0, this file was modified to add a “Demo” game state
  + Do NOT remove the entry for the “Demo” game state from the table
* You must make the following changes to this file for Project 1:
  + Add entries for the “Level1”, “Level2”, and “Sandbox” game states to the table
    - Hint: Use the existing entry for the “Stub” game state as an example

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.

**HERE**

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)
      * 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()
  + 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)
      * Hint: See the steps listed under GameStateLevel1Load, above
  + GameStateLevel2Init:
    - Read the initial value of “numHealth” from a file named “Level2\_Health.txt” (provided in the “Data” folder)
      * Hint: See the steps listed under GameStateLevel1Load, above
  + GameStateLevel2Update:
    - Decrement “numHealth” by 1
    - When “numHealth” reaches 0
      * Decrement “numLives” by 1
      * When “numLives” reaches 0, the game should switch to the Sandbox level (GsSandbox)
      * Otherwise, restart the level (GsRestart)

Vector2D.h

* This header file declares the public interface for creating and manipulating 2D vectors.
* The Vector2D structure is compatible with the AEVec2 structure and should be used instead of the AEVec2 structure in all future CS230 projects
* NOTE: The Vector2D module must not contain any references to the Alpha Engine’s AEVec2 module
* NOTE: All functions in this module will be tested during the grading process. It is your responsibility to make sure that all functions have been implemented and tested properly

GameStateSandbox.c/.h

* In this new game state 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 following these instructions properly and implemented the Vector2D functions correctly, then the contents of Trace.log should match the output is listed in the “Project 1 Testing” section, below
* 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:
  + GameStateSandboxTraceFloat:
    - Create a new *private* function for writing float values to the trace file
      * static void GameStateSandboxTraceFloat(const char\* text, float f);
    - The trace message should be formatted as "Vector Test: %s = %f"
  + GameStateSandboxTraceVector:
    - Create a new *private* function for writing Vector2D values to the trace file
      * static void GameStateSandboxTraceVector(const char\* text, const Vector2D\* v);
    - The trace message should be formatted as "Vector Test: %s = [%f, %f]"
  + GameStateSandboxUpdate:
    - Open the file, “Data/VectorTests.txt”, using StreamOpen()
    - If the stream was opened successfully,
      * Create a *single* Vector2D variable for use in the following tests. After each step you must write out either a Vector2D 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 Vector2D variables for use in the following tests. After each step you must write out either a Vector2D 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* Vector2D variable for use in the following tests. Optionally, you may use the first Vector2D variable created in this function. After each step you must write out either a Vector2D 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()
    - Set the next game state to “GsQuit”

# 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
    - “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:
    - CS230SU22<section letter>\_<Login ID>\_Project1.zip
      * Example: CS230SU22A\_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 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

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

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: Update

GSM: Update

Sandbox: Shutdown

Sandbox: Unload

Engine: Shutdown

GSM: Shutdown