# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 4.

# Objectives

The objectives for this Project are three-fold:

* To integrate your code from Project 1 with the Alpha Engine.
* To implement modules for displaying sprites and animations.
* To update the Trace module to handle messages with optional parameters.

# Description

For this project, you have been provided with a set of header files (.h) that specify the public interface for a sprite and animation system. You are responsible for creating the associated source files (.c) and implementing the functionality, as outlined in the header files and the lecture notes.

Instructions have been provided below on how to repurpose the two game states from Project 1 to display animated sprites using two common approaches:

* Simple animation (displays frames 0 to frame count)
* Animation sequences (can display any frame, in any order, for any duration)

Additionally, instructions have been provided below on how to modify the Trace module to handle messages with optional parameters.

# Game States

In Project 1, two levels were implemented as simple game states. These existing game states will be repurposed for Project 2.

As in Project 1, each of the games state functions must append a message to a “Trace.log” file

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

# Files

main.c

* This module contains the minimal amount of code necessary for the Main Loop.
* This module has been modified to incorporate changes for Project 2.
* Note that ‘dt’ is currently hardcoded to 0.0167f.
* There is no need to make any additional changes to this file for Project 2.

Engine.c

* This module combines the individual engine components together.
* This module has been modified to incorporate changes for Project 2.
* There is no need to make any additional changes to this file for Project 2.

Trace.c

* In Project 1, this module was created to open a text file for writing, append messages to the file, and close the file when the engine was shut down.
* In Project 2, you must rewrite 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, ...)
* In Project 2, trace messages need to be written only to the “trace.log” file.
* Writing messages to a file can be done using the following code:
  + va\_list arg\_ptr;
  + va\_start(arg\_ptr, formatString);
  + vfprintf\_s(traceStream, formatString, arg\_ptr);
  + fprintf\_s(traceStream, "\n");
  + va\_end(arg\_ptr);

Animation.h

* This header file declares the public interface for an animation object.
* The following typedef is used to store references to animation objects.
  + typedef struct Animation \* AnimationPtr;
* The contents of the Animation structure may not be accessed directly anywhere outside of Animation.c. The public interface provides everything necessary for this project.
* There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Animation.c. You are free to change the contents of this structure as long as you do not change the public interface.

AnimationFrame.h

* This header file declares a public structure used for storing information on a single frame of animation. This structure is public, instead of private, because it is used for what is referred to as “*plain old data*” (POD). In C++ programming, this is also referred to as a C-Style Structure.
* There is no need to make any changes to this file for Project 2.

AnimationSequence.h

* This header file declares the public interface for an animation sequence object.
* The following typedef is used to store references to animation sequence objects.
  + typedef struct AnimationSequence \* AnimationSequencePtr;
* The contents of the AnimationSequence structure may not be accessed directly anywhere outside of AnimationSequence.c. The public interface provides everything necessary for this project.
* There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into AnimationSequence.c. You are free to change the contents of this structure as long as you do not change the public interface.

Mesh.h

* This header file declares the public interface for creating a mesh object using the Alpha Engine.
* There is no need to make any changes to this file for Project 2.

Sprite.h

* This header file declares the public interface for a sprite object.
* The following typedef is used to store references to sprite objects.
  + typedef struct Sprite \* SpritePtr;
* The contents of the Sprite structure may not be accessed directly anywhere outside of Sprite.c. The public interface provides everything necessary for this project.
* There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into Sprite.c. You are free to change the contents of this structure as long as you do not change the public interface.

SpriteSource.h

* This header file declares the public interface for a sprite source object.
* The following typedef is used to store references to sprite source objects.
  + typedef struct SpriteSource \* SpriteSourcePtr;
* The contents of the SpriteSource structure may not be accessed directly anywhere outside of SpriteSource.c. The public interface provides everything necessary for this project.
* There is no need to make any changes to this file for Project 2. However, there is a sample structure that should be incorporated into SpriteSource.c. You are free to change the contents of this structure as long as you do not change the public interface.

GameStateManager.c

* In Project 1, this module was created to game state transitions, including “restart” functionality. This functionality is required for Project 2.
* There is no need to make any additional changes to this file for Project 2.

GameStateTable.c/.h

* In Project 1, these files were updated to include two new game states: “GsLevel1” and “GsLevel2”. These game states are also required for Project 2.
* There is no need to make any additional changes to these files for Project 2.

GameStateStub.c/.h

* Stub files for easily creating new game state modules and header files.
* Some minor improvements have been made to these files for Project 2.
* There is no need to make any changes to these files for Project 2.

GameStateLevel1.c/.h

* In Project 1, these files were created to create a simple game state. The existing functionality will be repurposed for Project 2.
* You must make the following changes to this file for Project 2:
  + Private Variables:
    - Add new private variables of the following types:
      * static AEGfxTexture \*
      * static AEGfxVertexList \*
      * static SpritePtr
      * static SpriteSourcePtr
      * static AnimationPtr
  + GameStateLevel1Load:
    - Read the initial value of “numLives” from a file named “Level1\_Lives.txt” (provided).
    - Create a quad mesh with the following parameters:
      * 300, 300, 0.25f, 0.25f, "Mesh4x4"
    - Create a texture with the following parameter:
      * "Assets\\Hexidecimal.png"
  + GameStateLevel1Init:
    - Create a sprite source object with a 4x4 sprite sheet:
      * 4, 4, pTexture
    - Create a sprite object with the following parameter:
      * "Level1 Sprite"
    - Set sprite’s mesh and sprite source.
    - Create an animation object, specifying the created sprite.
    - Initiate a simple looping animation with only 3 frames and a frame delay of 0.25f seconds, as follows:
      * AnimationPlay(animation, 3, 0.25f, true);
    - Set Alpha Engine’s background color to white (1,1,1).
    - Set Alpha Engine’s blend mode to blend.
  + GameStateLevel1Update:
    - Update the animation object.
    - Draw the sprite object at position (0, 0).
    - Each time the animation is done (AnimationIsDone()),
      * Decrement “numLives” by 1
      * When “numLives” reaches 0, the game should switch to Level2
  + GameStateLevel1Shutdown:
    - Free the Sprite, SpriteSource, and Animation objects.
  + GameStateLevel1Unload:
    - Free the Alpha Engine mesh and texture objects.

GameStateLevel2.c/.h

* In Project 1, these files were created to create a simple game state. The existing functionality will be repurposed for Project 2.
* The ‘numHealth’ variable is no longer used and can be removed.
* You must make the following changes to this file for Project 2:
  + Private Variables:
    - Add new private variables of the following types:
      * static AEGfxTexture \*
      * static AEGfxVertexList \*
      * static SpritePtr
      * static SpriteSourcePtr
      * static AnimationPtr
      * static AnimationSequencePtr
      * static AnimationFrame array with the following records:
        + { 4, 0.32f }, { 2, 0.32f }, { 5, 0.5f }
  + GameStateLevel2Load:
    - Read the initial value of “numLives” from a file named “Level2\_Lives.txt” (provided).
    - Create a quad mesh with the following parameters:
      * 300, 300, 0.25f, 0.25f, "Mesh4x4"
    - Create a texture with the following parameter:
      * "Assets\\Hexidecimal.png"
  + GameStateLevel2Init:
    - Create a sprite source object with a 4x4 sprite sheet:
      * 4, 4, pTexture
    - Create a sprite object with the following parameter:
      * "Level2 Sprite"
    - Set sprite’s mesh and sprite source.
    - Create an animation object, specifying the created sprite.
    - Create a *non-looping* animation sequence using the AnimationFrame array created in the private variables section.
    - Initiate an animation using the animation sequence.
      * AnimationPlaySequence()
    - Set Alpha Engine’s background color to white (1,1,1).
    - Set Alpha Engine’s blend mode to blend.
  + GameStateLevel2Update:
    - Update the animation object.
    - Draw the sprite object at position (0, 0).
    - Each time the animation is done (AnimationIsDone()),
      * Decrement “numLives” by 1
      * When “numLives” reaches 1, initiate the animation sequence again using AnimationPlaySequence()
      * When “numLives” reaches 0, the game should quit (GsQuit).
  + GameStateLevel2Shutdown:
    - Free the Sprite, SpriteSource, Animation, and AnimationSequence objects.
  + GameStateLevel2Unload:
    - Free the Alpha Engine mesh and texture objects.

Stub.c/.h

* Stub files for easily creating new modules and header files.
* Some minor improvements have been made to these files for Project 2.
* There is no need to make any changes to these files for Project 2.

# 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:
    - “.vs” folder
    - “AE” folder
    - “Assets” folder
    - “Data” folder
    - “Source” folder
    - Project2.sln
    - Project2.vcxproj
    - Project2.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:
    - CS230S18<section letter>\_<Login ID>\_Project2.zip
      * Example: CS230S18A\_john.doe\_Project2.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 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 your project 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 2 Testing

Here's sample output that you would find in your “Trace.log” file after running the application, assuming the following conditions (***NOTE: The last two conditions differ from the requirements in the assignment instructions, above, in order to improve the readability of the output***):

* Level1\_Lives.txt contains 3
* Level2\_Lives.txt contains 2
* EngineUpdate() is called with a dt = 0.0167f.
* GameStateLevel1 initiates the animation as follows:
  + AnimationPlay(animationData, 3, 0.032f, true);
* GameStateLevel2 initiates the animation sequence using the following frame data:
  + { 4, 0.032f }, { 2, 0.032f }, { 5, 0.05f }

Engine: Init

GSM: Init

Engine: Update

GSM: Update

Level1: Load

Level1: Init

SpriteSetFrame: Level1 Sprite frame index = 0

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 1

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 2

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 0

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 1

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 2

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 0

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 1

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 2

Engine: Update

GSM: Update

Level1: Update

Engine: Update

GSM: Update

Level1: Update

SpriteSetFrame: Level1 Sprite frame index = 0

Engine: Update

GSM: Update

Level1: Shutdown

Level1: Unload

Level2: Load

Level2: Init

SpriteSetFrame: Level2 Sprite frame index = 4

Level2: Update

Engine: Update

GSM: Update

Level2: Update

SpriteSetFrame: Level2 Sprite frame index = 2

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

SpriteSetFrame: Level2 Sprite frame index = 5

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

SpriteSetFrame: Level2 Sprite frame index = 4

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

SpriteSetFrame: Level2 Sprite frame index = 2

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

SpriteSetFrame: Level2 Sprite frame index = 5

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Update

Engine: Update

GSM: Update

Level2: Shutdown

Level2: Unload

Engine: Shutdown

GSM: Shutdown