# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 4 (Spring Semester) or Week 3 (Summer Semester).

# Objectives

The objectives for this Project are four-fold:

* To implement game objects using component-based design
* To implement game object construction using deserialization
* To implement functionality for positioning and moving game objects.
* To implement functionality for displaying game objects.

# Description

For this project, you have been provided with a set of header files (.h) that specify the public interface for creating game objects and attaching components. 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 moving sprites using two common approaches:

* Textured sprites
* Colored sprites using colored vertices

# Game States

In Projects 0 and 1, four levels were implemented as simple game states. Two of these existing game states will be repurposed for Project 2. The other two will remain in the project and be made accessible from the other game states

As in Project 1, each of the game state execute 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

For Project 2 you will need to copy certain files from Project 1. To do so, simply drag-and-drop the files from Project 1 into the source folder for Project 2. However, *do NOT overwrite any existing files in the Project 2 source folder!*

The files to be copied include: GameStateDemo.c/.h, GameStateLevel1.c/.h, GameStateLevel2.c/.h, GameStateManager.c, GameStateSandbox.c/.h, GameStateStub.c/.h, GameStateTable.c/.h, Stream.c, Stub.c/.h, Trace.c, and Vector2D.c.

NOTE: You may not change the public interface of the header files (.h) that are provided in Project 2, 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.

NOTE: The GameObject, Physics, Sprite, SpriteSource, and Transform structures must all be declared in their associated .c files, not the .h files. Exposing the internal implementation of these modules by declaring the structures in the .h files 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 additional changes to this file for Project 2

Engine.c

* This module combines the individual engine components together
* There is no need to make any changes to this file for Project 2

System.c

* This module has been modified to display a new window title:
  + AESysSetWindowTitle("CS230 Project 2 - Graphics and Physics");
* There is no need to make any changes to this file for Project 2

GameStateTable.c/.h

* In Project 1, these files were updated to include three new game states: “GsLevel1”, “GsLevel2”, and “GsSandbox”. These game states are also required for Project 2
* You should copy these two files over from Project 1

The following header files specify the public interface for a new set of modules. You are responsible for creating the associated source files (.c) and implementing the required functionality, as outlined in the header files and the lecture notes.

Mesh.h

* This header file declares the public interface for creating a mesh object using the Alpha Engine

Transform.h

* This header file declares the public interface for storing a game object’s position, orientation, and scale within the world space
* 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 Transform.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface
* The contents of the Transform structure may not be accessed directly anywhere outside of Transform.c. The public interface provides everything necessary for this project

Sprite.h

* This header file declares the public interface for a sprite object
* 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 within the .c file as long as you do not change the public interface
* 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
* To successfully draw a sprite, you will need to perform the following steps:
  + If the sprite has a sprite source,
    - Set the render mode to TEXTURE
    - Call AEGfxTextureSet, passing the texture and UV offsets from the sprite source
  + If the sprite does not have a sprite source,
    - Set the render mode to COLOR
  + Call AEGfxSetFullTransform passing the translation, scale, and rotation values from the transform.
    - NOTE: The rotation value must be converted from Radians to Degrees.
    - Hint: Use AERadToDeg
  + Call AEGfxSetTransparency, passing the “alpha” value from the sprite
  + Call AEGfxSetBlendColor, passing all 0.0f values
  + Call AEGfxMeshDraw, passing the mesh pointer from the sprite

SpriteSource.h

* This header file declares the public interface for a sprite source object
* 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 within the .c file as long as you do not change the public interface
* 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

Physics.h

* This header file declares the public interface for handling the motion of an object
* 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 Physics.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface
* The contents of the Physics structure may not be accessed directly anywhere outside of Physics.c. The public interface provides everything necessary for this project
* You will use the Semi-Implicit Euler integrator to update an object’s velocity and position every game loop, as follows:
  + Get the translation from the transform component.
  + Store the translation (as oldTranslation) in the physics component.
  + Use the Vector2DScaleAdd function to perform the following calculation:
    - velocity = acceleration \* dt + velocity
  + Use the Vector2DScaleAdd function to perform the following calculation:
    - translation = velocity \* dt + translation
  + Store the new translation in the transform component.

GameObject.h

* This header file declares the public interface for a simple container storing the individual components associated with a single game object. Each game object may contain one each of the following components (more will be added in future projects):
  + Transform
  + Physics
  + Sprite
* 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 GameObject.c. You are free to change the contents of this structure within the .c file as long as you do not change the public interface.
* NOTE: It is possible for a game object to contain all or none of the specified components. Your code must perform sufficient error checking to ensure that game objects missing one or more components are handled properly (i.e. no crashes, no unexpected side-effects).
* NOTE: It is your responsibility to ensure that all memory allocated for a given game object is freed when an object is destroyed. This includes all components currently attached to the game object. *Make sure to test your code using the Visual Studio debugger.*
* The GameObjectRead() function should work as follows:
  + If the gameObject and stream pointers are not NULL,
    - Read a token from the stream
    - Use the token to set the game object’s name
    - While (true)
      * Read a token from the stream
      * If “token” contains “Transform”,
        1. Create a new transform component using TransformCreate()
        2. Call TransformRead(), passing the created transform
        3. Add the transform to the game object
      * Else if “token” contains “Physics”,
        + Repeat steps 1-3 above, replacing “Transform” with “Physics”
      * Else if “token” contains “Sprite”,
        + Repeat steps 1-3 above, replacing “Transform” with “Sprite”
      * Else if “token” is empty (zero-length string),
        + Break out of the while-loop

GameObjectFactory.h

* This header file declares the public interface for building a new game object using data that is read from a file
* The GameObjectFactoryBuild() function should work as follows:
  + If the filename pointer is not NULL,
    - Open the file using StreamOpen()
    - If the stream was opened successfully,
      * Read the first token from the file using StreamReadToken()
      * Verify that the first token is “GameObject” using strncmp()
        + Hint: Use \_countof(“GameObject”) for \_MaxCount
      * If the first token is “GameObject”,
        + Create a new game object using GameObjectCreate()
        + Call GameObjectRead(), passing the created game object
        + Close the file using StreamClose()
        + Return the created game object
      * Close the file using StreamClose()
  + Return NULL

The following modules were created as part of Project 1 and will need to be modified for Project 2:

Stream.c/.h

* In Project 1, this module was created to read data from a serial stream (AKA “deserialization”). This module will be further expanded, as per the new function declaration in the Stream.h file and with the addition of a single private variable:
  + Add a new private array for storing “tokens” (single words).
    - static char tokenBuffer[1024];
    - This buffer has an arbitrary length that is more than sufficient for the needs of any CS230 project. The correct use of fscanf\_s() will help ensure that no buffer overruns occur should the source data ever become corrupted

GameStateDemo.c/.h

* In Project 0, these files were modified to incorporate the CS230 Demo functionality. In Project 2, this game state will be made accessible from the other three game states
* You must make the following changes to this file for Project 2:
  + GameStateDemoUpdate:
    - If the user triggers the ‘1’ key, change the game state to GsLevel1
      * HINT: Use '1', not VK\_NUMPAD1!
    - If the user triggers the ‘2’ key, change the game state to GsLevel2
    - If the user triggers the ‘9’ key, change the game state to GsSandbox
    - If the user triggers the ‘0’ key, restart the current level

GameStateSandbox.c/.h

* In Project 1, these files were created to test the Vector2D functionality. In Project 2, this game state will be made accessible from the other three game states
* You must make the following changes to this file for Project 2:
  + GameStateDemoUpdate:
    - If the user triggers the ‘1’ key, change the game state to GsLevel1
    - If the user triggers the ‘2’ key, change the game state to GsLevel2
    - If the user triggers the ‘9’ key, restart the current level
    - If the user triggers the ‘0’ key, change the game state to GsDemo

GameStateLevel1.c/.h

* In Project 1, these files were created to implement 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 Constants
    - Add the following constants:
      * static const float groundHeight = -150.0f;
      * static const float moveVelocity = 500.0f;
      * static const float jumpVelocity = 1000.0f;
      * static const Vector2D gravityNormal = { 0.0f, -1500.0f };
      * static const Vector2D gravityNone = { 0.0f, 0.0f };
  + Private Variables:
    - Add new private variables of the following types:
      * static AEGfxVertexList \*
      * static AEGfxTexture \*
      * static SpriteSourcePtr
      * static GameObjectPtr
  + GameStateLevel1Load:
    - Read the initial value of “numLives” from a file named “Data/Level1\_Lives.txt” (provided)
    - Create a quad mesh with the following parameters:
      * 0.5f, 0.5f, 1.0f, 1.0f, "Mesh1x1"
    - Load a texture with the following parameter:
      * " Assets/PlanetTexture.png"
    - Create a sprite source object with a 1x1 sprite sheet:
      * 1, 1, pTexture
  + GameStateLevel1Init:
    - Create a “Planet” game object by calling GameObjectFactoryBuild() with the parameter, "./Data/PlanetJump.txt"
    - If the game object was created successfully,
      * Get the game object’s sprite
      * Set the sprite’s mesh and sprite source
      * Set the sprite’s frame index to 0. While this call is not strictly necessary, it does allow you to test whether or not the trace message is written properly.
    - Set Alpha Engine’s background color to white (1,1,1)
    - Set Alpha Engine’s blend mode to blend
  + GameStateLevel1MovementController:
    - Create a new *private* function for moving the “Planet” game object.
      * void GameStateLevel1MovementController(GameObjectPtr gameObject)
    - Get the physics and transform components from the game object
    - Verify that the pointers are valid
    - Get the current velocity from the physics component and store it in a local variable. (Hint: you will need to dereference the return value)
    - Check for VK\_LEFT and VK\_RIGHT key presses, as follows:
      * If VK\_LEFT is pressed, set velocity.x = - moveVelocity
      * If VK\_RIGHT is pressed, set velocity.x = moveVelocity
      * If neither is pressed, set velocity.x = 0
    - If VK\_UP is “*triggered*”
      * Set velocity.y = jumpVelocity
      * Set the physics acceleration = gravityNormal
    - Check for “landing”, as follows:
      * Get the transform component’s current translation
      * If Y translation is < groundHeight
        + Set Y translation = groundHeight
        + Set velocity.y = 0
        + Set the physics acceleration = gravityNone
        + Decrement numLives by 1
        + If numLives <= 0, then set next game state to Level2
    - Set the physics component’s new velocity
  + GameStateLevel1Update:
    - Remove any existing code and replace it with the following
    - Update and display the Planet game object:
      * Call GameStateLevel1MovementController()
      * Call GameObjectUpdate()
      * Call GameObjectDraw()
    - If the user triggers the ‘1’ key, restart the current level
    - If the user triggers the ‘2’ key, change the game state to GsLevel2
    - If the user triggers the ‘9’ key, change the game state to GsSandbox
    - If the user triggers the ‘0’ key, change the game state to GsDemo
  + GameStateLevel1Shutdown:
    - Free the Planet game object
  + GameStateLevel1Unload:
    - Free the sprite source
    - Free the Alpha Engine mesh and texture objects (using the AE functions!)

GameStateLevel2.c/.h

* In Project 1, these files were created to implement 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:
  + The ‘numLives’ and ‘numHealth’ variables are no longer used and should be removed.
  + Private Constants
    - Add the following constants:
      * static const float spaceshipSpeed = 500.0f;
  + Private Variables:
    - Add new private variables of the following types:
      * static AEGfxVertexList \*
      * static GameObjectPtr
  + GameStateLevel2Load:
    - Create a “unit”-sized triangular mesh, as follows:

// Create a "unit"-sized triangular mesh.

AEGfxMeshStart();

AEGfxTriAdd(

-0.5f, -0.5f, 0xFFFF0000, 0.0f, 0.0f,

0.5f, 0.0f, 0xFFFFFF00, 0.0f, 0.0f,

-0.5f, 0.5f, 0xFFFF0000, 0.0f, 0.0f);

pMeshSpaceship = AEGfxMeshEnd();

AE\_ASSERT\_MESG(pMeshSpaceship, "Failed to create spaceship mesh!");

* + GameStateLevel2Init:
    - Create a “Spaceship” game object by calling GameObjectFactoryBuild() with the parameter, "./Data/SpaceshipHoming.txt"
    - If the game object was created successfully,
      * Get the game object’s sprite
      * Set the sprite’s mesh
    - Set Alpha Engine’s background color to black (0,0,0)
    - Set Alpha Engine’s blend mode to blend
  + GameStateLevel2MovementController:
    - Create a new *private* function for moving the “Spaceship” game object.
      * void GameStateLevel2MovementController(GameObjectPtr gameObject)
    - Get the physics and transform components from the game object
    - Verify that the pointers are valid
    - Get the mouse cursor position (in screen coordinates):
      * AEInputGetCursorPosition()
    - Convert the screen coordinates to world coordinates:
      * AEGfxConvertScreenCoordinatesToWorld()
    - Get the spaceship’s current translation
    - Calculate a direction vector from the spaceship to the mouse position
      * Normalize the direction vector using Vector2DNormalize()
    - Set the transform’s rotation, using Vector2DToAngleRad() to convert the direction vector into an angle (in radians)
    - Set the physics component’s velocity = direction vector \* spaceshipSpeed
  + GameStateLevel2Update:
    - Remove any existing code and replace it with the following
    - If ‘Z’ is pressed, set Spaceship sprite’s alpha value = 0.5f
    - If ‘X’ is pressed, set Spaceship sprite’s alpha value = 1.0f
    - Update and display the Spaceship game object:
      * Call GameStateLevel2MovementController()
      * Call GameObjectUpdate()
      * Call GameObjectDraw()
    - If the user triggers the ‘1’ key, change the game state to GsLevel1
    - If the user triggers the ‘2’ key, restart the current level
    - If the user triggers the ‘9’ key, change the game state to GsSandbox
    - If the user triggers the ‘0’ key, change the game state to GsDemo
  + GameStateLevel2Shutdown:
    - Free the Spaceship game object
  + GameStateLevel2Unload:
    - Free the Alpha Engine mesh object (using the AE function!)

# 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
    - 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:
    - CS230S22<section letter>\_<Login ID>\_Project2.zip
      * Example: CS230S22A\_john.doe\_Project2.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.