# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 6.

# Objectives

The objectives for this project are three-fold:

* To implement your own Matrix2D module.
* To calculate transformations for sprite positioning, scaling, and rotation.
* To implement displaying of sprite text.

# Description

For this project, you have been provided with a set of header files (.h) that specify the interface for two new modules. 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 Level1 game state from Project 2 to implement three unique game objects:

* A moving, jumping animated “monkey”
* A moving, bouncing “planet”
* A sprite text object indicating the number of lives remaining

# Files

NOTE: You may not change the public interface of the header files (.h) that were provided in Projects 2 and 3, 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 Animation, 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.

NOTE: The Vector2D and Matrix2D modules must not contain any references to the Alpha Engine.

The following header files specify the public interface for two new 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.

Matrix2D.h

* This header file declares the public interface for creating and manipulating 2D matrices and performing 2D transformations
* The Matrix2D structure is compatible with the AEMtx33 structure and should be used instead of the AEMtx33 structure in all future CS230 projects
* NOTE: The Matrix2D module must not contain any references to the Alpha Engine’s AEMtx33 or AEVec2 modules
* 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 work properly

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

The requirements for the following modules (.c and .h files) are unchanged from Project 2. Any issues identified in the Project 1 and/or Project 2 grading feedback should be addressed for Project 3. Otherwise, there is no need to make any changes to these files for Project 3.

Main.c

Engine.c/.h

GameStateLevel2.c/.h

GameStateManager.c/.h

GameStateStub.c/.h

GameStateTable.c/.h

Physics.c/.h

SpriteSource.c/.h

Stub.c/.h

System.c/.h

Trace.c/.h

Vector2D.c/.h

The following modules were created as part of Project 2 and will need to be modified for Project 3.

GameObject.c/.h

* This header file has been modified to include a new component type, Animation
* You must modify the private Animation structure to include a pointer to an Animation component
* You must implement the new GameObjectAddAnimation() and GameObjectGetAnimation() functions
* You must modify the GameObjectUpdate() function to correctly update any attached Animation component.
* NOTE: It is possible for a game object to contain all or none of the implemented 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 components attached to a given game object are freed correctly when an object is destroyed. Make sure to test your code using the Visual Studio debugger.

Transform.c/.h

* The header file has been modified to include a new function, TransformGetMatrix(), which returns the transformation matrix necessary to convert from the game object’s local (or model) space to the world space (world coordinates)
* You will need to calculate the transformation matrix when a game object’s translation, rotation, and/or scale are changed. To reduce the number of unnecessary calculations, the private Transform structure should be modified to include two new variables, a stored matrix (type: Matrix2D) and an isDirty flag (type: bool)
* The isDirty flag must be set to true when a Transform component is created or when the translation, rotation, or scale values are changed
* If the isDirty flag is true when TransformGetMatrix is called, then the stored transformation matrix must be calculated, as follows:
  + Use the Matrix2DScale, Matrix2DRotRad, and Matrix2DTranslate functions to create separate matrices for the transform’s scale, rotation, and translation values
  + Concatenate the rotation and scale matrices into a result matrix
  + Concatenate the translation and result matrices into a result matrix
    - The final concatenation can be performed directly into the transform’s stored matrix
  + Set the isDirty flag to false
  + NOTE: If your objects appear to be transformed incorrectly, then you are likely concatenating the matrices in the wrong order
* The resulting matrix must be passed into AEGfxSetTransform when drawing sprites

Sprite.c/.h

* The header file has been modified to include a new function, SpriteSetText(), which sets a zero-terminated string that is used to draw sprite text
* The function, SpriteDraw, must be modified to display sprites without using the functions AEGfxSetFullTransform() or AEGfxSetPosition(). The use of either of these two functions will result in a penalty to the project grade
* Now, when drawing a sprite, you must obtain a transformation matrix from the transform component passed into the function. This may be done by calling TransformGetMatrix()
* To support the drawing of both single sprites and sprite text, the code for rendering a mesh has been moved to a new function, MeshRender(). The code for SpriteDraw should now be implemented as follows:
  + Validate the sprite and sprite mesh pointers
  + If the sprite has a valid sprite source,
    - Set the render mode to TEXTURE
  + Else,
    - Set the render mode to COLOR
  + Set the alpha transparency for the sprite
  + Set the blend color for the sprite to (0, 0, 0, 0)
  + If the sprite’s text pointer is NULL,
    - Call MeshRender(), passing the sprite’s mesh, spriteSource, and frameIndex, as well as the Transform component’s transformation matrix
  + Else,
    - Call TransformGetMatrix() to get a local copy of the Transform component’s transformation matrix (“matrix”)
    - Call Matrix2DTranslate() to create an translation matrix (“offset”) with an X value equal to the Transform component’s X scale
    - Assign a local char \* variable equal to the sprite’s text pointer. This variable will be used to “walk” through the string without modifying the sprite’s text pointer
    - While the local text pointer points at a non-zero character,
      * Convert the current character into a zero-based frame index
        + Hint: The font sheet begins at the space character (‘ ‘)
      * Call MeshRender(), passing the sprite’s mesh and spriteSource, the local copy of the transformation matrix, and the frame index that was just calculated
      * Increment the local text pointer by 1
      * Call Matrix2DConcat() to concatenate the translation matrix and transformation matrix
        + matrix = offset \* matrix

Mesh.c/.h

* The header file has been modified to include a new function, MeshRender(), which will be used to render a mesh (either textured or untextured).
* This new function must be called by SpriteDraw to perform the following steps that are required to display graphics:
  + If the sprite source pointer is valid,
    - Call SpriteSourceGetUV() to calculate the UV coordinates of the specified frame index
    - Call AEGfxTextureSet() to set the texture pointer and the UV coordinates
  + Call AEGfxSetTransform(), passing the matrix parameter’s, “m” member
  + Call AEGfxMeshDraw(), passing the mesh parameter

GameStateLevel1.c/.h

* The existing functionality will be repurposed for Project 3.
* You must make the following changes to this file for Project 3:
  + Private Constants
    - Add an enum, MonkeyStates, with the following states:
      * MonkeyInvalid = -1,
      * MonkeyIdle,
      * MonkeyWalk,
      * MonkeyJump
    - Add the following constants:
      * static const float wallDistance = 350.0f;
      * static const float CheckSquareDistance = (75.0f \* 75.0f);
  + Private Variables
    - Add a MonkeyStates variable
      * static enum MonkeyStates monkeyState;
    - Add a buffer for printing the Lives text (“Lives: #”)
      * static char livesBuffer[16] = "";
      * Note: This is an arbitrarily sized buffer that is sufficiently large for its intended purpose
    - In addition to the private variables previously created for the “Planet” game object, you will need to create new private variables for the “Monkey” and “LivesText” objects
  + GameStateLevel1Load:
    - Create a quad mesh with the following parameters:
      * 0.5, 0.5, 1.0f / 3, 1.0f / 3, "Mesh3x3"
    - Create a quad mesh with the following parameters:
      * 0.5, 0.5, 1.0f / 16, 1.0f / 6, "Mesh16x6"
    - Create a texture with the following parameter:
      * "Assets/MonkeyIdle.png"
    - Create a texture with the following parameter:
      * "Assets/MonkeyWalk.png"
    - Create a texture with the following parameter:
      * "Assets/MonkeyJump.png"
    - Create a texture with the following parameter:
      * "Assets/FontSheet.png"
    - Create a sprite source object with a 1x1 sprite sheet:
      * 1, 1, pTextureMonkeyIdle
    - Create a sprite source object with a 3x3 sprite sheet:
      * 3, 3, pTextureMonkeyWalk
    - Create a sprite source object with a 1x1 sprite sheet:
      * 1, 1, pTextureMonkeyJump
    - Create a sprite source object with a 16x6 sprite sheet:
      * 16, 6, pTextureFontSheet
  + GameStateLevel1CreatePlanet:
    - Update the existing *private* function as follows:
      * Set the planet’s translation to { 0, 300 }
      * Set the planet’s velocity to { 150, 0 }
      * Set the planet’s acceleration to gravityNormal
  + GameStateLevel1CreateMonkey:
    - Create a new *private* function for creating a “Monkey” game object.
      * GameObjectPtr GameStateLevel1CreateMonkey(void)
    - Create a game object with the following parameter:
      * "Monkey"
    - Create a transform component with the following parameters:
      * Translation: 0, groundHeight
      * Rotation: 0
      * Scale: 150, 150
    - Create a sprite component with the following parameter:
      * "Monkey Sprite"
    - Create an animation component, specifying the created sprite
    - Create a physics component without any additional changes.
    - Add the transform, sprite, animation, and physics components to the game object
  + GameStateLevel1CreateLivesText:
    - Create a new *private* function for creating a “LivesText” game object.
      * GameObjectPtr GameStateLevel1CreateLivesText(void)
    - Create a game object with the following parameter:
      * "LivesText"
    - Create a transform component with the following parameters:
      * Translation: -350, 250
      * Rotation: 0
      * Scale: 50, 50
    - Create a sprite component with the following parameter:
      * "LivesText Sprite"
      * Set sprite’s mesh (“Mesh16x6”)
      * Set the sprite’s sprite source (“FontSheet”)
    - Use sprint\_s() to write the following text to the livesBuffer:
      * “Lives: %d”, numLives
    - Call SpriteSetText(), passing the livesBuffer
    - Add the transform and sprite components to the game object
  + GameStateLevel1SetMonkeyState:
    - Create a new *private* function for managing the “Monkey” object’s current state and animation
      * void GameStateLevel1SetMonkeyState(GameObjectPtr gameObject, MonkeyStates newState)
    - If (monkeyState != newState)
      * Set monkeyState = newState
      * Get the game objects sprite and animation components
      * Switch(newState)
        + Case MonkeyIdle:

Set the sprite’s mesh to the 1x1 mesh

Set the sprite source to “MonkeyIdle”

Call AnimationPlay with a frame count of 1, a frame delay of 0.0f, and looping = false

* + - * + Case MonkeyWalk:

Set the sprite’s mesh to the 3x3 mesh

Set the sprite source to “MonkeyWalk”

Call AnimationPlay with a frame count of 8, a frame delay of 0.05f, and looping = true

* + - * + Case MonkeyJump:

Set the sprite’s mesh to the 1x1 mesh

Set the sprite source to “MonkeyJump”

Call AnimationPlay with a frame count of 1, a frame delay of 0.0f, and looping = false

* + GameStateLevel1Init:
    - Create a “Planet” game object (this code already exists)
    - Create a “Monkey” game object
    - Initialize the monkeyState variable to “MonkeyInvalid”
    - Call GameStateLevel1SetMonkeyState(gameObjectMonkey, MonkeyIdle);
    - Create a “LivesText” game object
    - Set Alpha Engine’s background color to white (1,1,1)
    - Set Alpha Engine’s blend mode to blend.
  + GameStateLevel1MovementController:
    - This existing *private* function must be modified to support animations. The code for changing the monkey’s state and animation variables can be added to the existing movement code, as follows:
    - If VK\_LEFT is pressed,
      * Set X velocity = - moveVelocity (this code should already exist)
      * If the monkeyState variable is not equal to MonkeyJump,
        + Call GameStateLevel1SetMonkeyState(gameObject, MonkeyWalk)
    - Else if VK\_RIGHT is pressed,
      * Set X velocity = moveVelocity (this code should already exist)
      * If the monkeyState variable is not equal to MonkeyJump,
        + Call GameStateLevel1SetMonkeyState(gameObject, MonkeyWalk)
    - Else
      * Set X velocity = 0 (this code should already exist)
      * If the monkeyState variable is not equal to MonkeyJump,
        + Call GameStateLevel1SetMonkeyState(gameObject, MonkeyIdle)
    - If VK\_UP is “triggered”
      * Call GameStateLevel1SetMonkeyState(gameObject, MonkeyJump)
    - In the existing check for “landing” code:
      * REMOVE the following code:
        + Decrement numLives by 1
        + If numLives <= 0, then set next game state to Level2
      * If Y translation is < groundHeight
        + Call GameStateLevel1SetMonkeyState(gameObject, MonkeyIdle)
  + GameStateLevel1BounceController:
    - Create a new *private* function for “bouncing” the “Planet” game object
      * void GameStateLevel1BounceController(GameObjectPtr)
    - Get the physics and transform components from the game object
    - Verify that the pointers are valid
    - Get the object’s current position and velocity
    - If position.x <= -WallDistance
      * Set position.x = -WallDistance
      * Set velocity.x = -velocity.x
    - If position.x >= WallDistance
      * Set position.x = WallDistance
      * Set velocity.x = -velocity.x
    - If position.y <= groundHeight
      * Set position.y = groundHeight + (groundHeight - position.y)
        + Note: This calculation is necessary to conserve energy
      * Set velocity.y = -velocity.y
    - Set the object’s new position and velocity
  + GameStateLevel1IsColliding:
    - Create a new *private* function for detecting when two game objects are “colliding”
      * bool GameStateLevel1IsColliding(GameObjectPtr objectA), GameObjectPtr objectB)
    - Get the current position of the two game objects
    - Using the function, Vector2DSquareDistance, calculate the distance (squared) between the two objects
    - If the distance (squared) < CheckSquareDistance
      * Return true
    - Else
      * Return false
  + GameStateLevel1Update:
    - Call GameStateLevel1MovementController(), passing the “Monkey” game object *instead* of the “Planet” game object
    - Call GameStateLevel1BounceController(), passing the “Planet” game object
    - Call GameObjectUpdate() to update the “Monkey”, “Planet” and “LivesText” game objects
    - Call GameObjectDraw() to draw the “Monkey”, “Planet” and “LivesText” game objects
    - Call GameStateLevel1IsColliding(), passing the “Monkey” and “Planet” game objects. If this function returns true, then do the following:
      * Decrement numLives by 1
      * If numLives <= 0
        + Change the game state to Level 2
      * Else
        + Restart the current level
    - The following code already exists and should not be modified or removed:
      * If the user presses the ‘1’ key, restart the current level
      * If the user presses the ‘2’ key, change the game state to Level 2
  + GameStateLevel1Shutdown:
    - Free the “Monkey”, “Planet” and “LivesText” game objects
  + GameStateLevel1Unload:
    - Free all sprite sources
    - Free all Alpha Engine mesh and texture objects (using the AE functions!)

# 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
    - Project3.sln
    - Project3.vcxproj
    - Project3.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:
    - CS230S19<section letter>\_<Login ID>\_Project3.zip
      * Example: CS230S19A\_john.doe\_Project3.zip
* Upload the submission .zip file via the Moodle page for your CS230 section (A, B or C)
* 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
  + 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 program

# 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.