# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 14 (Spring Semester) or Week 11 (Summer Semester).

# Objectives

The objectives for this project are as follows:

* Implement a component-based entity architecture using inheritance in C++.
* Refactor the existing scenes to use this new architecture, as necessary.

# Description

For this project, you will update the previously implemented scenes (Level 1, Level 2, Asteroids & Omega) by switching the entity composition to a component-based class using C++’s inheritance and polymorphism, where the entity serves as a container of abstract components.

For this project, you are encouraged to remove the Demo and Sandbox scenes. These scenes are not required for the project and removing them can reduce the work necessary to complete the project.

For this project, you may elect to exclude the Asteroids and Omega scenes for reduced credit, as outlined in the following Rubric section. Any scenes that you do implement must be accessible from all other scenes (using the ‘1’, ‘2’, ‘3’, and ‘4’ keys). The default scene must be set to “Level 1”. Any scenes that cannot be accessed from Level 1, for any reason, will be assumed not to exist.

# Rubric

* Level 1 & 2 (80%)
  + Successfully implement the Level 1 (Monkey) and Level 2 (Homing Spaceship) functionality after converting the following from C-style structs to C++ classes:
    - Entity
    - Animation, Sprite, Physics, Transform
* Asteroids (10%)
  + Successfully implement the Asteroids scene functionality after converting the following from C-style structs to C++ classes:
    - Behavior, BehaviorSpaceship, BehaviorBullet, BehaviorHudText
  + NOTE: You may omit the asteroids and collision functionality from this scene.
* Omega (10%)
  + Successfully implement the Omega scene functionality after converting the following from C-style structs to C++ classes:
    - BehaviorAsteroid
    - Collider, ColliderCircle, ColliderLine

# Suggested Steps

1. Migrate your source code (.c files) and project files from Project 6 to Project 7.
   * Note: This can be accomplished using the branching feature in your version control solution (SVN, Git, etc.).
   * Rename the “Project6.\*” files to “Project7.\*”.
   * The .sln file will still be referencing the “Project6.vcxproj file. To fix this issue:
     1. Delete the Project7.sln file.
     2. Double-click the Project7.vcxproj file to open it in Visual Studio 2022.
     3. Click the “Save All” button (or “Ctrl+Shift+S”) to create a new Project7.sln file.
2. Rename all .c files to .cpp.
   * This can be done within Visual Studio, although this can be a slow, tedious process.
   * Alternatively, you can remove all .c files from the VS project, rename the files, and add the new .cpp files to the VS project. However, you will need to restore the precompiled header settings correctly, if you use this approach (see Step 4, below).
3. Commit the project to your version control solution (SVN, Git, etc.).
   * WARNING: Do not commit files/folders that are automatically generated by Visual Studio, such as the “.vs” and “x64” folders. It is a good idea to add these folders to the “ignore list” in SVN or Git.
4. Change the project to be compiled as C++.
   * Right click on the project in Visual Studio.
   * Click on “Properties”.
   * Next to “Configuration”, select “All Configurations”.
   * Next to “Platform”, select “All Platforms”.
   * On the left side, expand the “C/C++” option.
   * Click on “Advanced”.
   * Next to “Compile As”, select “Compile as C++ Code (/TP)”.
   * Click on “Language”.
   * Next to “Conformance Mode”, select “No (/permissive)”.
   * Click “OK”.
5. If necessary, update the project settings for the precompiled header.
   * Right click on “stdafx.cpp”.
   * Click on “Properties”.
   * Next to “Configuration”, select “All Configurations”.
   * Next to “Platform”, select “All Platforms”.
   * On the left side, expand the “C/C++” option.
   * Click on “Precompiled Headers”.
   * Next to “Precompiled Header”, select “Create (/Yc)”.
   * Click “OK”.
6. Click on “Build/Rebuild Solution”.
7. Clean up all compiler warnings and errors.
8. Verify that all scenes in the project still work correctly.
9. Commit the project to your version control solution (SVN, Git, etc.).
10. Begin working on the changes listed below.
    * ***Pro Tip:*** It is highly recommended that you work in small, incremental steps. For example, instead of converting all components to C++ at the same time, pick one component, Transform for example, and convert it to C++. Once you’ve successfully converted it to C++ and worked out all the problems, then convert a second component, and so on.

# Files

NOTE: For this project, you may freely add/remove/edit the code and header files. However, you should maintain good programming practices and avoid making radical changes to the project that might make it difficult for the instructor or TA’s to grade your project.

When migrating header files from C to C++ you will need to remove the following code from every header (.h) file. If you do not complete this step, then you will encounter linker errors that are difficult to understand.

#ifdef \_\_cplusplus

extern "C" {

/\* Assume C declarations for C++ \*/

#endif

#ifdef \_\_cplusplus

} /\* End of extern "C" { \*/

#endif

## Component.cpp/.h

The project must have an abstract base component class called “Component”. This class must have:

* A public enum (for example, *TypeEnum*) with IDs for each of the component types:
  + Animation, Behavior, Collider, Physics, Sprite, Transform
* A pure virtual function: virtual Component\* Clone(void) const = 0;
  + Every derived class should implement an overload for the Clone function. Incorrectly cloning components can result in memory-related bugs.
  + **Hint:** Each clone function should invoke the copy constructor for the associated class.
* A virtual function for updating a component:
  + virtual void Update(float dt)
* A virtual function for rendering a component:
  + virtual void Render() const
* A private variable of type, TypeEnum, which is used to specify the type of the component (Transform, Sprite, etc…).
  + This variable must be set only when a component is constructed.
    - *WARNING: This variable must be “private” and there must be no accessor function for* setting *the component type.*
  + Implement an accessor function for getting the component type.
* A private variable of type, Entity\*, which is used to specify the “parent” entity of the component.
  + Implement accessor functions for getting and setting the parent pointer.
  + This variable must replace the “parent” variables present in the original Animation, Behavior and Collider structures.

## Entity.cpp/.h

The project must have a class called “Entity”. This class must:

* Implement a container for attaching components. This container can be an array, a linked-list, or a standard-template library class containing variables of type, Component\*.
  + *Pro-Tip:* You are encouraged to use std::vector for this container. However, this is not a requirement.
* Implement a function to add components to the entity. This function should accept a parameter of type, Component\*.
  + *Hint:* Make sure to set the component’s “parent” pointer at this time.
  + *Note:* This Add function has the responsibility of setting the parent-child relationship of the Entity and Component, in a single location.
* Implement a function to iterate through the attached components and return a pointer to a component that matches the specified type. Refer to the “Components in C++” lecture slides for suggestions on possible implementation.
* Modify the EntityUpdate function to *iterate through all attached components* and call the component’s Update function.
* Modify the EntityRender function to *iterate through all attached components* and call the component’s Render function.
* Modify the EntityFree function to *iterate through all attached components* and delete each one.

# Submission Requirements

* The project must build cleanly, with no errors or warnings.
* The Visual Studio project files that you submit must be named Project7\*.\*. If you are using a previous set of project files, then please rename them by changing the solution and project properties in Visual Studio. Make sure that the resultant .exe filename is also changed to Project7.
* 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
    - Project7.sln
    - Project7.vcxproj
    - Project7.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 extraneous files or folders, including but not limited to the following Visual Studio folders:
    - Folders: .vs, “Debug”, “Release”, “x64”
  + Rename the resultant .zip file using the following naming convention:
    - CS230S24<section letter>\_<Login ID>\_Project7.zip
      * Example: CS230S24A\_john.doe\_Project7.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 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 to any late submissions. ***No submissions will be accepted after Friday of Finals Week***; Week 15 (Spring Semester) or Week 12 (Summer Semester).
* 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.