# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 12 (Spring Semester) or on Wednesday during Week 9 (Summer Semester).

# Objectives

The objectives for this project are three-fold:

* Implement a new scene (“Omega”).
* Implement separate colliders for circles and lines.
* Implement Point-Line collision detection and reflection.

# Description

For this project, you have been provided with a set of header files (.h) that specify the interface for three 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.

The scene (“Asteroids”) created in Projects 4 and 5 will be duplicated in order to implement a new scene in Project 6. All scenes created during the previous projects should remain in the game and be accessible from the Asteroids and Omega scenes.

# Suggested Steps

1. Migrate your source code (.c files) from Project 5 to Project 6.
2. Commit the project to your version control solution (SVN, Git, etc.).
3. Implement any stub functions required to get the project to compile. Use temporary return values (0 or NULL) and UNREFERENCED\_PARAMETER() macros to solve any compiler warnings.
4. Commit the project to your version control solution (SVN, Git, etc.).
5. Implement the ColliderCircle functionality.
   * Verify that the Asteroids scene still works properly.
6. Create a new Omega scene by duplicating the Asteroids scene and changing the default scene in SceneSystemInit.
   * Verify that the Omega scene works exactly like the Asteroids scene.
7. Add the Arena entity to the Omega scene.
   * Verify that the inner rectangle is displayed correctly.
8. Switch the HUDText entities from the “Asteroids” versions to the “Omega” versions.
   * Verify that the HUD text now appears within the inner region of the Omega scene.
9. Implement the ColliderLine functionality and update the Omega scene to use this new functionality.
   * Verify that entities correctly bounce off the arena walls.

# Files

NOTE: You may not change the public interface of the header files (.h) that have been provided, 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, ColliderCircle, ColliderLine, Entity, EntityController, Mesh, MeshLibrary, Physics, Sprite, SpriteSource, SpriteSourceLibrary, 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.

The Behavior and Collider structures are declared publicly in the .h file, as they will be used to implement pseudo-inheritance in this project.

## Entity.c

The following changes should be made to your existing Entity.c file:

* EntityRead:
  + Add code to handle the new ColliderCircle and ColliderLine components.
  + The code to handle the original Collider component should be removed or rewritten, as appropriate.

## Collider.h

This header file has been updated as follows:

* Public Structures:
  + A ColliderType enum has been added.
  + The Collider structure has been modified as follows:
    - The structure has been moved from the .c file to the .h file to facilitate the creation of new “derived” Collider components.
    - A ColliderType variable has been added to differentiate between the ColliderCircle and ColliderLine derived structures.
    - A memorySize variable has been added to allow the ColliderClone function to allocate and copy the correct amount of memory.
      * (Hint: This was also done with the Behavior components.)
* Public Functions:
  + ColliderCreate
    - This function has been replaced by the Create functions in the two new Collider components.
      * [In C++, this would be implemented using a pure virtual constructor and derived constructors.]
  + ColliderRead
    - This function has been replaced by the Read functions in the two new Collider components.
  + ColliderCheck
    - The parameters in the declaration have been changed to const. The function definition in the .c file should be changed to match.

## Collider.c

The following changes must be made to your existing Collider.c file:

* Public Structures:
  + Remove the Collider structure.
* Public Functions:
  + ColliderCreate
    - Remove this function.
  + ColliderRead
    - Remove this function.
  + ColliderClone:
    - Modify the call to calloc() to use the value in memorySize, rather than sizeof(Collider) to ensure that the correct amount of memory is allocated
    - Replace the code to copy the contents of “other” with the following:
      * memcpy(clone, other, other->memorySize);
    - Set the clone’s parent to NULL.
  + ColliderCheck:
    - Change the two parameters to match the function declaration in the header file.
    - Replace your implementation of the collision detection with a call to a new private function:
      * bool ColliderIsColliding(const Collider\* collider, const Collider\* other)
      * If the function returns true, then invoke the collision handlers, as per the instructions in Project 5.
* Private Functions:
  + ColliderIsColliding:
    - This new, *private* function is responsible for calling the correct collision check function, as determined by the ColliderType of the two colliders.
      * If both colliders are circle colliders,
        + Return the result from ColliderCircleIsCollidingWithCircle.
      * If one collider is a circle and the other collider is a line,
        + Return the result from ColliderLineIsCollidingWithCircle.
        + Make sure to pass the two colliders in the correct order.
      * If both colliders are line colliders,
        + Return false.
        + [***Hint:*** This condition should never be true, as there is only one line collider in the Omega scene (the Arena entity).]

## ColliderCircle.h

* This header file declares the public interface for detecting collisions between two circular objects. Refer to the function headers in this file for instructions on the implementation of the .c file.
  + ***Important Note:*** The ColliderCircle structure is “derived” from the Collider structure. Make sure to set the memorySize member variable correctly to avoid severe memory bugs.
  + There is no need to make any changes to this file for Project 6.

## ScoreSystem.c/.h

## This module will be reused to implement a score system for the Omega scene.

## There is no need to make any changes to these files for Project 6.

## SceneSystem.c

* You must make the following changes to this file for Project 6:
  + SceneSystemInit:
    - Change the starting (Default) scene from "Asteroids" to "Omega".

## CheatSystem.c

* You must make the following changes to this file for Project 6:
  + Update the key bindings to handle the following:
    - If the user *triggers* the ‘4’ key, change the scene to Omega.

## OmegaScene.c/.h

* Create these files by duplicating AsteroidsScene.c/.h.
  + Rename the public and private function names, replacing “AsteroidsScene” with “OmegaScene”.
* You must make the following changes to these files for Project 6:
  + OmegaSceneInit
    - Create an “Arena” Entity and add it to the scene.
    - Replace the “Spaceship” Entity with the “SpaceshipOmega” Entity.
    - Replace the existing HUD Text entities with the following:
      * OmegaScore, OmegaHighScore, OmegaWave

## Data/MeshArena.txt

* This data file specifies the mesh data for the rectangular inner arena wall sprite.
  + ***Requirement:*** This file must be created by you.
* The mesh may be created using a quad mesh or a list of vertices.
  + ***Requirement:*** The vertices must fit within a unit-sized area centered around the origin (-0.5f to 0.5f).
  + ***Requirement:*** The mesh color must clearly contrast with the background color and the sprite text color. If the Arena sprite is difficult to see or if the HUD text is difficult to read because of poor color choice, then a penalty may be applied.

## Data/MeshAsteroid.txt

* This data file must be copied over from your Project 5 implementation.

## ColliderLine.h

* This header file declares the public interface for detecting collisions between a Line Collider and a Circle Collider. Refer to the function headers in this file for instructions on the implementation of the .c file.
  + There is no need to make any changes to this file for Project 6.

## ColliderLine.c

* ***Important Note:*** The ColliderLine structure is “derived” from the Collider structure. Make sure to set the memorySize member variable correctly to avoid severe memory bugs.
* Implement Point-Line collision detection, as discussed during the Week 10 lectures (Week 8 during the summer semester). Refer to the lecture slides, available on Moodle, for implementation details.
* Implement Reflection, as discussed during the Week 12 lectures (Week 7 during the summer semester). Refer to the series of “Collisions” lecture slides, available on Moodle and Teams, for implementation details.
  + Each object’s position, rotation, and velocity should be updated correctly following the collision/reflection.
  + ***Important Note:*** You are strongly encouraged NOT to implement circle-line collision detection (radius > 0) for this Project. Only point-line collision is required.

# 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:
    - “Assets” folder
    - “Data” folder
    - “DGL” folder
    - “Source” folder
    - Project6.sln
    - Project6.vcxproj
    - Project6.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:
    - CS230S25<section letter>\_<Login ID>\_Project6.zip
      * Example: CS230S25A\_john.doe\_Project6.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. However, no project submissions will be accepted after end-of-day Friday of Finals Week, without prior written approval of the instructor.
* A -25% penalty will be applied to any submissions that utilize the project materials provided in a previous 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.