# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 10 (Spring Semester) or end-of-day on Wednesday during Week 8 (Summer Semester).

# Objectives

The objectives for this project are four-fold:

* To implement simple collision detection between entities.
* To implement more complex behaviors using pseudo-inheritance in C.
* To implement a simple “wave” system for asteroids.
* To implement a reusable entity for displaying HUD text.

# Description

For this project, you have been provided with a set of header files (.h) that specify the interface for six new modules. You are responsible for creating the associated source files (.c) and implementing the functionality, as outlined in the header files and the instructions below.

The Asteroids Scene created in Project 4 will be used to implement an Asteroids clone in Project 5. The scenes created during Projects 0 – 3 should remain in the game and must be accessible from the Asteroids scene.

# 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, Collider, 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 structure is declared publicly in the .h file, as it will be used to implement pseudo-inheritance in this project. The BehaviorAsteroid and BehaviorHudText structures must be declared privately in their associated .c file.

## Teleporter.h

* This header file declares the public interface for monitoring the movement of entities and “teleporting” them to the opposite side of the screen when they attempt to leave the viewable area. See the Project 5 Demo for proper functionality.
* The position of the screen edges can be obtained as follows:
  + Call DGL\_Window\_GetSize() to get the X/Y dimensions of the window.
  + Call Vector2DScale to scale the dimensions by 0.5f to get the window half size.
  + The bottom-left corner of the window is at –halfSize.
  + The top-right corner of the window is at +halfSize.
* The actual implementation of this function is left up to the student. However, it is recommended that you use an entity’s velocity to determine which edge of the screen to test against, as shown in the Project 5 slide deck.

## Random.h

* This header file declares the public interface for generating random numbers, either int’s or float’s, within specified ranges.

## Engine.c

* The following changes must be made to this module:
  + EngineInit:
    - Add a call to the function, RandomInit. The placement is not critical, but it is probably best to call this function before initializing the engine systems.

## Mesh.c

* The following changes must be made to this module:
  + MeshRead:
    - If the first token in the file is “Quad” (instead of “Mesh”), then:
      * Read a Vector2D representing the half-size values of the mesh.
      * Read two integers representing the number of columns and rows, respectively, of the associated sprite sheet.
      * Read a token representing the name of the mesh to be created.
      * Call MeshBuildQuad.

## SpriteSource.h

* This header file has been modified to include two new functions that must be implemented in the associated .c file:
  + SpriteSourceRead
  + SpriteSourceIsNamed

## SpriteSource.c

* A new “name” variable must be added to the SpriteSource structure. An example of this variable, along with supporting comments, can be found in the header file.

## SpriteSourceLibrary.h

* This header file declares the public interface for creating and managing SpriteSource objects. See the header file and the information below for detailed instructions on the implementation of the .c file.

## SpriteSourceLibrary.c

* NOTE: The functions in this module are very similar to those in MeshLibrary.c. You may be able to save time by adapting the code from the existing module.
  + SpriteSourceLibraryBuild
    - See the function header for detailed implementation instructions.
  + SpriteSourceLibraryAdd
    - This private function should accept a SpriteSource and insert it at the first available location in the SpriteSource list.
  + SpriteSourceLibraryFind:
    - This private function should accept a “const char \*” representing a name and search through the SpriteSource list for a SpriteSource with a matching name. If a matching name is found, then this function should return a pointer to that SpriteSource.

## Sprite.c

* The function, SpriteRead, must be modified to handle sprite sources. Add the following code after the Mesh has been added to the Sprite:
  + Read a token that represents the name of a SpriteSource.
  + Call SpriteSourceLibraryBuild(), passing the name of the SpriteSource.
  + Call SpriteSetSpriteSource(), passing the created SpriteSource.

## Scene.h

* This header file has been updated to include the function, SceneFindEntityByName.
  + This function is simply a helper function that calls EntityContainerFindByName.

## Scene.c

* SceneLoad:
  + Add call to SpriteSourceLibraryInit.
    - NOTE: This must be done before executing the Load function.
* SceneUpdate:
  + Add call to EntityContainerCheckCollisions.
    - NOTE: The *correct* order for the operations in this function are:
      * Execute the Update function.
      * Update all Entities within the Scene.
      * Check for collisions between Entities.
* SceneUnload:
  + Add call to SpriteSourceLibraryFreeAll.

## Collider.h

* This header file declares the public interface for managing collider components, which are used to detect and resolve collisions between two objects.
* A collider component has a pointer to its “parent” entity so that it can access the entity’s transform component. Care must be taken to ensure that a cloned collider component points at the newly cloned entity, rather than the archetype entity.
* Implement a simple Circle-Circle collision check using the entity’s scale (from the transform) to approximate a radius. For example:
  + float radius = TransformGetScale(transform)->x / 2.0f;
* When a collision is detected, check the two colliders for CollisionEventHandlers. *For each handler found*, call the handler, passing pointers to the two parent entities. Note, the ordering of the two pointers varies, depending upon which handler is being called:
  + collider->handler(collider->parent, other->parent);
  + other->handler(other->parent, collider->parent);

## Entity.h

* This header file has been updated to include the Collider component. This includes the new functions:
  + EntityAddCollider()
  + EntityGetCollider()

## Entity.c

* EntityClone:
  + Add code to clone the Collider component.
* EntityFree:
  + Add code to free the Collider component.
* EntityRead:
  + Add code to correctly construct the new behavior and collider components.
    - BehaviorAsteroid, BehaviorHudText, and Collider
    - NOTE: The BehaviorHudText component has a unique function for reading data from the file. Make sure to call BehaviorHudTextRead, instead of BehaviorRead, for this component.

## EntityContainer.h

* This header file has been updated to include the function, EntityContainerCheckCollisions.
  + This function should search through the active entity list, looking for any entities with an attached Collider component. When a Collider component is found, then search through the ***remainder*** of the active entity list, again looking for Collider components. For each *unique* pair of entities with colliders, call the function, ColliderCheck.

## Behavior.h

* This header file has been updated to include a new variable:
  + unsigned int memorySize;
    - This variable is used to allocate the correct amount of memory when cloning a behavior component.
    - ***Warning! Cloning of "derived" behaviors will result in severe crash bugs if insufficient memory is allocated.***

## Behavior.c

* BehaviorClone:
  + Modify the call to calloc() to use the value in memorySize, instead of sizeof(Behavior), 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);

## BehaviorBullet.c

* BehaviorBulletCreate:
  + Set the behavior’s “memorySize” variable to sizeof(Behavior).
* BehaviorBulletCollisionHandler(Entity\* entity1, Entity\* entity2);
  + This is a new, private function for resolving a collision between two objects.
  + If the two pointers are valid,
    - If entity2’s name is “Asteroid”,
      * Call EntityDestroy, passing entity1.
* BehaviorBulletInit:
  + If “stateCurr” is equal to cBulletIdle,
    - Get the parent Entity’s Collider component.
    - If the Collider component exists,
      * Set the Collider’s collision handler to the new private function.
* BehaviorBulletUpdate
  + Call TeleporterUpdateEntity outside of the switch statement.

## BehaviorSpaceship.c

* Add the following to the spaceship’s behavior state enum:
  + cSpaceshipDead
* Reduce the weapon cooldown timer from 0.032f to 0.25f:
  + static const float spaceshipWeaponCooldownTime = 0.25f;
* Set the spaceship’s “death” duration to 3 seconds:
  + static const float spaceshipDeathDuration = 3.0f;
* BehaviorSpaceshipCreate:
  + Set the behavior’s “memorySize” variable to sizeof(Behavior).
* BehaviorSpaceshipCollisionHandler (Entity\* entity1, Entity\* entity2);
  + This is a new, private function for resolving a collision between two objects.
  + If the two pointers are valid,
    - If entity2’s name is “Asteroid”,
      * Set entity1’s “stateNext” behavior variable to cSpaceshipDead
* BehaviorSpaceshipInit:
  + If “stateCurr” is equal to cSpaceshipIdle,
    - Get the parent entity’s collider component.
    - If the collider component exists,
      * Set the collider’s collision handler to the new private function.
  + If “stateCurr” is equal to cSpaceshipDead,
    - Set the behavior timer equal to spaceshipDeathDuration.
    - Implement a “death” effect that lasts for the specified duration. The implementation details are left up to the student, but the effect should be, at least, somewhat interesting.
      * NOTE: At a minimum, this means two simple effects involving a few lines of code or one effect that is somewhat technically challenging.
* BehaviorSpaceshipUpdate
  + Call TeleporterUpdateEntity outside of the switch statement.
  + If “stateCurr” is equal to cSpaceshipDead,
    - Decrement the behavior timer by dt.
    - If the behavior timer < 0,
      * Restart the scene.
    - Implement the “death” effect, as mentioned above.

## Data/MeshAsteroid.txt

* This data file specifies the mesh data for asteroid objects.
* This file must be created by you and must reside in the Data subdirectory.
  + Hint: Use a copy of the MeshBullet.txt file as a starting point.
  + *Warning*: Using MeshAsteroid.txt from the DEMO project, or any variation of the file, may result in a penalty to your grade.
* The asteroid mesh must be constructed using a ***minimum*** of 3 triangles and must fit entirely ***within a 1-by-1 unit area***.

## BehaviorAsteroid.h

* This header file declares the public interface for creating and updating behaviors associated with asteroid entities. See the information below for detailed instructions on the implementation of the .c file.

## BehaviorAsteroid.c

* Create an enum with the following entries:
  + cAsteroidInvalid = -1
  + cAsteroidIdle = 0
* Add the following private constants:
  + // Speed range of the asteroids.
  + static const float asteroidSpeedMin = 50.0f;
  + static const float asteroidSpeedMax = 100.0f;
* Add the following private function declarations:
  + static void BehaviorAsteroidInit(Behavior\*);
  + static void BehaviorAsteroidUpdate(Behavior\*, float dt);
  + static void BehaviorAsteroidExit(Behavior\*);
  + static void BehaviorAsteroidSetPosition(BehaviorAsteroid\*);
  + static void BehaviorAsteroidSetVelocity(BehaviorAsteroid\*);
  + static void BehaviorAsteroidCollisionHandler(Entity\*, const Entity\*);
* BehaviorAsteroidCreate:
  + Allocate memory for the BehaviorAsteroid.
  + Initialize the base behavior variables.
    - Hint: Make sure to set the memorySize correctly.
  + Cast the BehaviorAsteroid pointer to a Behavior pointer and return the result.
* BehaviorAsteroidInit:
  + Cast the Behavior\* to a BehaviorAsteroid\*.
  + If “stateCurr” is equal to cAsteroidIdle,
    - Set the asteroid behavior’s “origin” variable to a random number between 0 and 3 (inclusive).
    - Call the function, BehaviorAsteroidSetPosition().
    - Call the function, BehaviorAsteroidSetVelocity().
    - Get the parent Entity’s Collider component.
    - If the Collider component exists,
      * Set the Collider’s collision handler to the new private function.
* BehaviorAsteroidUpdate:
  + Call TeleporterUpdateEntity outside of the switch statement.
* BehaviorAsteroidCollisionHandler (Entity\* entity1, Entity\* entity2);
  + This is a new, private function for resolving a collision between two objects.
  + If the two pointers are valid,
    - If entity2’s name is either “Bullet” or “Spaceship”,
      * Pass the value 20 to ScoreSystemIncreaseScore.
      * Call EntityDestroy with entity1
* BehaviorAsteroidSetPosition:
  + Set the asteroid’s position to one of the four corners of the screen, depending upon its “origin” variable.
    - Hint: Using a switch statement for this is recommended. Make sure to put a breakpoint in each case-statement and verify that the asteroids are positioned in the correct corner.
* BehaviorAsteroidSetVelocity:
  + Determine the asteroid’s starting direction,
    - If “origin” is top-left corner,
      * Generate a random angle between -10 and -80 degrees.
    - If “origin” is top-right corner,
      * Generate a random angle between -100 and -170 degrees.
    - If “origin” is bottom-left corner,
      * Generate a random angle between 10 and 80 degrees.
    - If “origin” is bottom-right corner,
      * Generate a random angle between 100 and 170 degrees.
  + Set the asteroid’s velocity in the direction of the random angle, with a random speed between asteroidSpeedMin and asteroidSpeedMax.

## ScoreSystem.h

* This header file declares the public interface for tracking the score, high score, and wave count information for the Asteroids scene and, later, a new scene in Project 6.
* Use a copy of StubSystem.c to create the associated .c file.

## ScoreSystem.c

* The default Init, Update, Render, and Exit functions do not require any code.
* Private Structures:
  + Add the following variables to the ScoreSystem struct:
    - unsigned score;
    - unsigned highScore;
    - unsigned waveCount;
* Public Functions:
  + Implement the public functions declared in the header file according to the instructions provided in the function headers.

## Main.c

* Add the ScoreSystem to the engine immediately after the SceneSystem.

## BehaviorHudText.h

* This header file declares the public interface for creating and updating behaviors associated with HUD Text entities. See the information below for detailed instructions on the implementation of the .c file.

## BehaviorHudText.c

* Create a HudTextStates enum (see example in BehaviorHudText.h).
* Create a BehaviorHudText struct (see example in BehaviorHudText.h).
* Add the following private function declarations:
  + static void BehaviorHudTextInit(Behavior\*);
  + static void BehaviorHudTextUpdate(Behavior\*, float dt);
  + static void BehaviorHudTextExit(Behavior\*);
  + static void BehaviorHudTextUpdateText(BehaviorHudText\*);
* BehaviorHudTextCreate:
  + Allocate memory for the BehaviorHudText.
  + Initialize the base behavior variables.
    - Hint: Make sure to set the memorySize correctly.
  + Initialize scoreSystemId to SsiInvalid.
  + Cast the BehaviorHudText pointer to a Behavior pointer and return the result.
* BehaviorHudTextRead:
  + Read the base Behavior values using BehaviorRead.
  + Read the formatString using StreamReadToken.
  + Read the scoreSystemId using StreamReadInt.
* BehaviorHudTextInit:
  + Call BehaviorHudTextUpdateText().
  + Assign the BehaviorHudText component’s displayString variable to the parent’s Sprite component, using the SpriteSetText function.
* BehaviorHudTextUpdate:
  + Call ScoreSystemGetValue to get the current value associated with this Behavior.
  + If the value is displayValue is different than the retrieved value,
    - then call BehaviorHudTextUpdateText.
* BehaviorHudTextExit:
  + Do nothing.
* BehaviorHudTextUpdateText:
  + If the value in scoreSystemId is not SsiInvalid:
    - Call ScoreSystemGetValue to get the current value associated with this Behavior. Store this value in displayValue.
    - Call sprintf\_s(), using displayString, formatString, and displayValue as parameters.

## AsteroidsScene.c

* You must make the following changes to this file for Project 5:
  + Add the following private constants:
    - static const unsigned cAsteroidSpawnInitial = 8;
    - static const unsigned cAsteroidSpawnMaximum = 20;
  + Add the following to the AsteroidScene structure:
    - unsigned asteroidSpawnCount;
  + Add the following private function declarations:
    - static void AsteroidsSceneSpawnAsteroidWave(void);
    - static void AsteroidsSceneSpawnAsteroid(void);
  + AsteroidsSceneLoad:
    - Call ScoreSystemClear.
  + AsteroidsSceneInit:
    - After creating the spaceship entity:
      * Build the "AsteroidsScore" Entity and add it to the Scene.
      * Build the "AsteroidsHighScore" Entity and add it to the Scene.
      * Build the "AsteroidsWave" Entity and add it to the Scene.
      * Call ScoreSystemReset.
      * Initialize the spawn count to cAsteroidSpawnInitial.
      * Call AsteroidsSceneSpawnAsteroidWave().
  + AsteroidsSceneUpdate:
    - Check for any existing “Asteroid” entities by calling SceneFindEntityByName.
    - If there are no asteroids in active list,
      * Call AsteroidsSceneSpawnAsteroidWave.
  + AsteroidsSceneSpawnAsteroidWave:
    - Call ScoreSystemIncreaseWave to increase the wave count by 1.
    - Call AsteroidsSceneSpawnAsteroid a number of times equal to asteroidSpawnCount.
    - Increment asteroidSpawnCount by 1, to a maximum of cAsteroidSpawnMaximum.
  + AsteroidsSceneSpawnAsteroid:
    - Build an “Asteroid” entity and add it to the Scene.

# Optional Features

If you wish, you may implement one or both of the following features. There is no grade bonus associated with these optional features. Please be careful not to gain grade penalties by breaking required functionality.

* Add a reverse thruster state (e.g. cSpaceshipReverse) to the BehaviorSpaceship module. This should apply a negative acceleration based upon the spaceship’s rotation. This state should otherwise behave identically to the cSpaceshipThrust state.
* Implement the drag effect (velocity \* 0.99f) discussed during the “Project 4 - Spaceship" lecture. This drag effect should be applied only to the spaceship. The declaration for a PhysicsSetDrag() function has been added to Physics.h for those wishing to implement this feature.

# 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
    - Project5.sln
    - Project5.vcxproj
    - Project5.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:
    - CS230SU24<section letter>\_<Login ID>\_Project5.zip
      * Example: CS230SU24A\_john.doe\_Project5.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 for each week or portion of a week that the project is submitted late.
* 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.