# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 8 (Spring Semester) or on Wednesday during Week 6 (Summer Semester).

# Objectives

The objectives for this project are four-fold:

* To implement a container for storing entities.
* To implement a simple mesh “library”.
* To implement cloning of entities.
* To implement two unique behaviors.

# Description

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

Additionally, a new Scene will be added to implement an Asteroids clone across Projects 4 & 5. The four scenes created during the previous projects should remain in the project 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, Entity, EntityController, Mesh, MeshLibrary, 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.

The Behavior structure has been declared publicly in the .h file, as it will be used to implement pseudo-inheritance in Project 5.

## SceneSystem.c

* You must make the following changes to this file for Project 4:
  + Change the starting (Default) scene from " Level1" to "Asteroids":
    - SceneSystemSetNext(AsteroidsSceneGetInstance());

## CheatSystem.c

* You must make the following changes to this file for Project 4:
  + Update the key bindings to handle the following:
    - If the user *triggers* the ‘3’ key, change the scene to Asteroids.

## AsteroidsScene.c/.h

* You will need to create these files and add them to the project.
  + Hint: Use the existing StubScene.c & .h files as examples.
* You must make the following changes to this file for Project 4:
  + AsteroidsSceneInit:
    - Set the DGL background color to black (0,0,0,0).
    - Set the DGL blend mode to blend.
  + NOTE: Additional changes will be made to this file later in the instructions.

## Scene.h

* This header file has been updated to include the function, SceneAddEntity.
* There is no need to make any changes to this file for Project 4.

## Scene.c

* You must make the following changes to this file for Project 4:
  + In the Private Variables section, add:
    - static EntityContainer\* entities = NULL;
  + SceneLoad:
    - Initialize the entities variable by calling EntityContainerCreate.
      * Hint: This should be done ***before*** executing the “load” function.
  + SceneUpdate:
    - Update any existing entities by calling EntityContainerUpdateAll.
  + SceneRender
    - Render all entities within the scene by calling EntityContainerRenderAll.
  + SceneExit:
    - Free any existing entities by calling EntityContainerFreeAll and EntityFactoryFreeAll.
      * Hint: This should be done ***after*** executing the “free” function.
  + SceneUnload:
    - Free the EntityContainer by calling EntityContainerFree.
      * Hint: This should be done ***after*** executing the “unload” function.
  + SceneAddEntity:
    - This new function should add an Entity to the EntityContainer’s list.

## Entity.h

* The header file has been updated to include the function, EntityIsNamed.
  + Note: This new function should be used in EntityContainer.c.
* This header file has been updated to include the functions, EntityIsDestroyed and EntityDestroy. These functions should get and set a new member variable that indicates when the Entity should be destroyed. For example, this new variable might be defined as:
  + bool isDestroyed
* This header file has been updated to include the functions, EntityGetBehavior and EntityAddBehavior. These functions should get and set a new member variable that stores a behavior component. For example, this new variable might be defined as:
  + Behavior\* behavior;
* This header file has been updated to include the function, EntityClone. This function should first perform a shallow copy of the data, followed by a deep copy of any attached components.
  + Hint: Use the EntityAdd\* functions to attach the cloned components to the newly cloned Entity.

## EntityContainer.h

* This header file declares the public interface for creating and maintaining a collection of Entity objects. This container object will be used to store active entities within each Scene and archetype entities within the EntityFactory.
* There is no need to make any changes to this file for Project 4. However, there is a sample structure that should be incorporated into EntityContainer.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 EntityContainer structure may not be accessed directly anywhere outside of EntityContainer.c. The public interface provides everything necessary for this project.

## AsteroidsScene.c/.h

* You must make the following changes to this file for Project 4:
  + AsteroidsSceneInit:
    - Create a “Spaceship” Entity by calling EntityFactoryBuild() with the parameter, "./Data/Spaceship.txt".
    - If the Entity was created successfully,
      * Call SceneAddEntity(), passing the created Entity.

## EntityFactory.h

* This header file has been updated to change the parameter of the function, EntityFactoryBuild. We will now be passing the name of an Entity, instead of a filename.
* This header file has been updated to add the function EntityFactoryFreeAll.
* There is no need to make any changes to this file for Project 4.

## EntityFactory.c

* You must make the following changes to this file for Project 4:
  + In the Private Variables section, add:
    - static EntityContainer\* archetypes = NULL;
  + EntityFactoryBuild:
    - The function must be changed to accept the name of an entity, rather than the path to the data file. Before opening the data file, the Entity’s name must be combined with the file path information, as follows:
      * char pathName[FILENAME\_MAX] = "";
      * sprintf\_s(pathName, \_countof(pathName), "Data/%s.txt", entityName);
    - ***NOTE:*** After making the preceding change, search for all calls to EntityFactoryBuild and change the parameter from a file path to an entity name.
      * Hint:Change"./Data/Spaceship.txt" to "Spaceship".
    - The function must also be changed to make use of an Archetype Entity list. The code should now behave as follows:
      * If entityName is NULL,
        + Return NULL,
      * If the “archetypes” variable is NULL, then initialize the variable by calling EntityContainerCreate.
      * Call EntityContainerFind to see if an archetype of the requested Entity already exists.
      * If the archetype does not exist,
        + Construct the file pathname as shown above.
        + Open the data file for streaming.
        + If the stream was opened successfully,

Read a token.

If the token == “Entity”.

Call EntityCreate to create a new archetype.

Call EntityRead.

Call EntityContainerAddEntity, passing the new archetype.

Close the stream.

* + - * If the archetype existed or was created successfully,
        + Clone the archetype Entity.
        + Return the ***cloned*** Entity.
    - Hint: You may want to wait and implement the cloning functionality after BehaviorSpaceship has been implemented but before spawning bullets.
  + EntityFactoryFreeAll:
    - If the "archetypes" container exists, then call EntityContainerFreeAll.

## Level1Scene.c

* The following changes should have been made while implementing the changes described in the EntityFactoryBuild section, above. If not, then make these changes now.
  + Level1SceneInit:
    - Change the strings passed to EntityFactoryBuild() to:
      * "PlanetBounce"
      * "Monkey"
      * "MonkeyLivesText"

## Level2Scene.c

* The following changes should have been made while implementing the changes described in the EntityFactoryBuild section, above. If not, then make these changes now.
  + Level2SceneInit:
    - Change the string passed to EntityFactoryBuild () to:
      * "SpaceshipHoming"

## Stream.h

* This header file has been updated to include the function, StreamReadColor. This function should read the RGBA color components from the stream as four floats.

## Mesh.h

* This header file has been updated to include the functions, MeshCreate and MeshRead.
* Implementation details can be found in the function headers.

## MeshLibrary.h

* This header file declares the public interface for creating and freeing meshes. See the information below and in the header file comments for additional, detailed instructions on the implementation of the .c file.
* There is no need to make any changes to this file for Project 4. However, there is a sample structure that should be incorporated into MeshLibrary.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 MeshLibrary structure may not be accessed directly anywhere outside of MeshLibrary.c. The public interface provides everything necessary for this project.

## MeshLibrary.c

* You must make the following changes to this file for Project 4:
  + In the Private Variables section, add:
    - static MeshLibrary meshes;
  + In the Private Function Declarations section, add:
    - static void MeshLibraryAdd(const Mesh\* mesh);
    - static const Mesh\* MeshLibraryFind(const char\* meshName);
  + MeshLibraryInit
    - This function should initialize the MeshLibrary variable.
    - This function will be called multiple times.
  + MeshLibraryBuild
    - This function attempts to find an existing Mesh with the specified name.
    - If a matching Mesh is not found, then a new Mesh must be created by reading data from a file and creating the corresponding DGL\_Mesh resource.
    - After the Mesh is constructed it must be stored in the Mesh list.
    - HINT: See the function header for detailed implementation instructions.
  + MeshLibraryFreeAll
    - All created Mesh objects must be freed by calling MeshFree.
  + MeshLibraryAdd.
    - This private function should insert a new Mesh into the Mesh list.
  + MeshLibraryFind
    - This private function should accept a “const char \*” representing a name and search through the Mesh list for a Mesh with a matching name. If a matching name is found, then this function should return a pointer to that Mesh.
      * HINT: The MeshIsNamed function has been added for this purpose.

## Scene.c

* You must make the following additional changes to this file for Project 4:
  + SceneLoad:
    - Call MeshLibraryInit().
      * Hint: This should be done ***before*** executing the “load” function.
  + SceneUnload:
    - Call MeshLibraryFreeAll().
      * Hint: This should be done ***after*** executing the “unload” function.

## Transform.h

* This header file has been updated to include the function, TransformClone. This function should perform a shallow copy of all member variables of the original object.

## Physics.h

* This header file has been updated to include the function, PhysicsClone. This function should perform a shallow copy of all member variables of the original object.
* This header file has been updated to include the functions, PhysicsGetRotationalVelocity & PhysicsSetRotationalVelocity. These functions should get and set a new rotational velocity variable that you must add to the Physics structure. For example, this new variable might be defined as:
  + float rotationalVelocity;

## Physics.c

* The function, PhysicsUpdate, must be modified to add the rotational velocity to a Entity’s Transform ‘rotation’ every game loop. For example, (in pseudocode):
  + rotation += rotationalVelocity \* dt

## Sprite.h

* This header file has been updated to include the function, SpriteClone. This function should perform a shallow copy of all member variables of the original object.

## Sprite.c

* The function, SpriteRead, must be modified to handle meshes, as follows:
  + After reading frameIndex and alpha from the stream, read a token that represents the name of a mesh.
  + Call MeshLibraryBuild(), passing the name of the mesh.
  + Call SpriteSetMesh(), passing the created mesh.

## Animation.h

* This header file has been updated to include the function, AnimationClone. This function should perform a shallow copy of all member variables of the original object.
  + Hint: It is possible to copy all members of a structure from one variable to another, in the following manner (this is a shallow copy):
    - \*animation = \*other
  + After an Animation component is cloned, its ‘parent’ pointer must be updated to point at the newly cloned Entity. This step should be performed by calling EntityAddAnimation.

## Behavior.h

* This header file declares the public interface for creating and updating behaviors.
* In Project 5, the base Behavior structure will be used to create “derived” behavior structures for specific entities. For now, the Spaceship and Bullet behaviors will use only the base Behavior structure.
* The function, BehaviorUpdate, should implement a finite-state machine (FSM), as follows:
  + Validate the behavior pointer.
  + If the behavior state is changing (stateCurr != stateNext),
    - Call the onExit() function, *iff* (“if and only if”) it exists.
    - Set stateCurr = stateNext
    - Call the onInit() function, *iff* it exists.
  + Call the onUpdate function, *iff* it exists.

## Entity.c

* EntityFree
  + This function must be modified to call BehaviorFree.
* EntityAddBehavior
  + This function must also store the Entity pointer in the behavior’s ‘parent’ member variable, as follows:
    - BehaviorSetParent(behavior, entity);
* EntityRead
  + This function must be updated to create behaviors and add them to the Entity.
    - If the token “BehaviorSpaceship” is encountered,
      1. Call BehaviorSpaceshipCreate()
      2. Call BehaviorRead(), passing the created behavior
      3. Call EntityAddBehavior(), passing the created behavior
    - If the token “BehaviorBullet” is encountered,
      * Repeat steps 1 – 3, above, calling BehaviorBulletCreate() instead
* EntityUpdate
  + This function must be modified to call BehaviorUpdate.
    - Note: The order of the update calls will become important in Project 6. Give some thought to the order that you update each of the components. This will be discussed further during future lectures.

## BehaviorSpaceship.h

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

## BehaviorSpaceship.c

* Create an enum with the following entries:
  + cSpaceshipInvalid = -1
  + cSpaceshipIdle
  + cSpaceshipThrust
* Add the following private constants:
  + static const float spaceshipAcceleration = 150.0f;
  + static const float spaceshipSpeedMax = 500.0f;
  + static const float spaceshipTurnRateMax = PI / 1.5f;
  + static const float spaceshipWeaponCooldownTime = 0.034f;
  + static const float spaceshipWeaponBulletSpeed = 750.0f;
* Add the following private function declarations:
  + static void BehaviorSpaceshipOnInit(Behavior\* behavior);
  + static void BehaviorSpaceshipOnUpdate(Behavior\* behavior, float dt);
  + static void BehaviorSpaceshipOnExit(Behavior\* behavior);
  + static void BehaviorSpaceshipUpdateRotation(Behavior\* behavior, float dt);
  + static void BehaviorSpaceshipUpdateVelocity(Behavior\* behavior, float dt);
  + static void BehaviorSpaceshipUpdateWeapon(Behavior\* behavior, float dt);
  + static void BehaviorSpaceshipSpawnBullet(Behavior\* behavior);
* BehaviorSpaceshipCreate:
  + Calloc the memory for a Behavior structure.
  + If the memory was allocated successfully,
    - Set stateCurr = cSpaceshipInvalid
    - Set stateNext = cSpaceshipInvalid
    - Set the onInit, onUpdate, and onExit variables in the base Behavior structure equal to the corresponding “On” private functions declared above.
* BehaviorSpaceshipInit:
  + This function may remain empty.
  + Use UNREFERENCED\_PARAMETER() to fix the warning.
* BehaviorSpaceshipUpdate:
  + Add a switch statement with the expression, (behavior->stateCurr)
  + Add a case for cSpaceshipIdle
    - Call BehaviorSpaceshipUpdateRotation.
    - Call BehaviorSpaceshipUpdateWeapon
    - If ‘VK\_UP’ is *pressed* (being held down),
      * Set next behavior state = cSpaceshipThrust
  + Add a case for cSpaceshipThrust
    - Call BehaviorSpaceshipUpdateRotation.
    - Call BehaviorSpaceshipUpdateVelocity.
    - Call BehaviorSpaceshipUpdateWeapon
    - If ‘VK\_UP’ is NOT *pressed*,
      * Set next behavior state = cSpaceshipIdle
* BehaviorSpaceshipExit:
  + This function may remain empty.
  + Use UNREFERENCED\_PARAMETER() to fix the warning.
* BehaviorSpaceshipUpdateRotation:
  + If ‘VK\_LEFT’ is *pressed*,
    - Set the physics component’s rotation velocity = spaceshipTurnRateMax
  + Else If ‘VK\_RIGHT’ is *pressed*,
    - Set the physics component’s rotation velocity = -spaceshipTurnRateMax
  + Else
    - Set the physics component’s rotation velocity = 0
* BehaviorSpaceshipUpdateVelocity:
  + Get the transform and physics components from the behavior’s parent Entity.
  + Verify that the pointers are valid.
  + Get the transform component’s ‘rotation’.
  + Get a unit vector in direction of ‘rotation’.
    - Hint: There is a Vector2D function for this
  + Get the physics component’s ‘velocity’.
  + Calculate the new velocity:
    - velocity = velocity + direction of rotation \* spaceshipAcceleration \* dt
    - Note: Try using the Vector2DScaleAdd function for this
  + Calculate the ‘speed’ of this new velocity.
    - Hint: speed = length(new velocity)
  + If the speed > spaceshipSpeedMax, then limit the speed:
    - velocity = velocity \* (spaceshipMaxSpeed / speed)
  + Set the physics component’s new velocity.
* BehaviorSpaceshipUpdateWeapon:
  + If the behavior timer > 0,
    - Decrement the behavior timer by ‘dt’.
    - If the behavior timer < 0,
      * Set the behavior timer = 0
  + If spacebar (‘ ‘) is *pressed*,
    - If behavior timer <= 0
      * Call BehaviorSpaceshipSpawnBullet
      * Set behavior timer = spaceshipWeaponCooldownTime
* BehaviorSpaceshipSpawnBullet:
  + Call EntityFactoryBuild to build a new ‘Bullet’ Entity.
    - NOTE: The ‘entityName’ parameter is case-sensitive!
  + If the Bullet was cloned successfully
    - Get the spaceship’s position and rotation.
    - Set the *cloned* bullet’s position and rotation.
    - Get a unit vector in direction of the spaceship’s ‘rotation’.
    - Set the bullet’s velocity = direction \* spaceshipWeaponBulletSpeed
    - Add the cloned bullet to the Entity manager’s active list.

## BehaviorBullet.h

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

## BehaviorBullet.c

* Create an enum with the following entries:
  + cBulletInvalid = -1
  + cBulletIdle
* Add the following private function declarations:
  + static void BehaviorBulletOnInit(Behavior\* behavior);
  + static void BehaviorBulletOnUpdate(Behavior\* behavior, float dt);
  + static void BehaviorBulletOnExit(Behavior\* behavior);
  + static void BehaviorBulletUpdateLifeTimer(Behavior\* behavior, float dt);
* BehaviorBulletCreate
  + Calloc the memory for a Behavior structure.
  + If the memory was allocated successfully,
    - Set stateCurr = cBulletInvalid.
    - Set stateNext = cBulletInvalid.
    - Set the onInit, onUpdate, and onExit variables in the base Behavior structure equal to the corresponding “On” private functions declared above.
* BehaviorBulletInit
  + This function may remain empty.
  + Use UNREFERENCED\_PARAMETER() to fix the warning.
* BehaviorBulletUpdate
  + Add a switch statement with the expression, (behavior->stateCurr)
  + Add a case for cBulletIdle.
    - Call BehaviorBulletUpdateLifeTimer.
* BehaviorBulletExit
  + This function may remain empty.
  + Use UNREFERENCED\_PARAMETER() to fix the warning.
* BehaviorBulletUpdateLifeTimer()
  + If the behavior timer > 0,
    - Decrement the behavior timer by ‘dt’.
    - If the behavior timer <= 0,
      * Call EntityDestroy(), passing the behavior’s ‘parent’ pointer.

# 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
    - Project4.sln
    - Project4.vcxproj
    - Project4.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>\_Project4.zip
      * Example: CS230SU24A\_john.doe\_Project4.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 -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.