# Due Date

This assignment must be completed and submitted via Moodle before end-of-day on Friday during Week 8.

# Objectives

The objectives for this project are three-fold:

* To implement a simple game object manager.
* To implement cloning of game objects.
* 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.

One of the new modules is a game state that will be used to implement an Asteroids clone across Projects 4 & 5. The two game states created during Projects 2 & 3 should remain in the game and be accessible from the Asteroids game state.

# 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, Transform and GameObjectManager 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 may be declared publicly in the .h file, as it will be used to implement pseudo-inheritance in Project 5.

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, except for ‘sprite’, which should, instead, point at the cloned object’s sprite component.
* 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

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 game objects. For now, the Spaceship and Bullet behaviors will use only the base Behavior structure.
* The function, BehaviorUpdate, should be implemented 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.

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 game object’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.

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.

GameObject.h

* This header file has been updated to include the function, GameObjectClone. This function should perform a shallow copy of the ‘name’ member variable and create cloned copies of any attached components.
  + Exception: If you have implemented a buffer for the ‘name’ member variable, then you must perform a deep copy of this variable.
* This header file has been updated to include the functions, GameObjectIsDestroyed and GameObjectDestroy. These functions should get and set a new member variable that indicates when the game object should be destroyed. For example, this new variable might be defined as:
  + bool isDestroyed
* This header file has been updated to include the function, GameObjectGetName. This function should return a pointer the game object’s ‘name’ member variable.
* This header file has been updated to include the functions, GameObjectGetBehavior and GameObjectAdddBehavior. These functions should get and set a new member variable that stores a behavior component. For example, this new variable might be defined as:
  + BehaviorPtr behavior;

GameObject.c

* GameObjectAddBehavior
  + This function must store the game object pointer in the behavior’s ‘parent’ member variable, as follows:
    - behavior->parent = gameObject;
* GameObjectUpdate
  + This function must be modified to call BehaviorUpdate.

GameObjectManager.h

* This header file declares the public interface for managing lists of game objects. The current implementation will utilize two lists, one containing active game objects and another contain game object archetypes.
* These two lists can be fixed-length arrays (minimum size of 100 entries) or dynamically-sized linked lists.

GameObjectManager.c

* The following private variables will be required:
  + static GameObjectManager gameObjectActiveList;
  + static GameObjectManager gameObjectArchetypes;

Engine.c

* In EngineInit(), add a call to GameObjectManagerInit().
* In EngineUpdate(), add calls to GameObjectManagerUpdate() and GameObjectManagerDraw() *before* SystemDraw().
* In EngineShutdown(), add a call to GameObjectManagerShutdown().

GameStateTable.c/.h

* Modify these files to add a new game state, ‘GsAsteroids’.
* Set the game state’s name to “Asteroids”.
* Set GsInitial = GsAsteroids.

GameStateAsteroids.c/.h

* You will need to create these files and add them to the project.
* Hint: Use the existing GameStateStub.c & .h files as examples.
* You must make the following changes to this file for Project 4:
  + Add the following private variables:
    - static AEGfxVertexList\* pMeshSpaceship = NULL;
    - static AEGfxVertexList\* pMeshBullet = NULL;
  + Add the following private function declarations:
    - static void GameStateAsteroidsCreateMeshes(void);
    - static void GameStateAsteroidsFreeMeshes(void);
    - static void GameStateAsteroidsCreateSpaceship(void);
    - static void GameStateAsteroidsCreateBulletArchetype(void);
  + GameStateAsteroidsLoad:
    - Call GameStateAsteroidsCreateMeshes().
  + GameStateAsteroidsInit:
    - Call GameStateAsteroidsCreateSpaceship().
    - Call GameStateAsteroidsCreateBulletArchetype().
    - Set Alpha Engine’s background color to black (0,0,0).
    - Set Alpha Engine’s blend mode to blend.
  + GameStateAsteroidsUpdate:
    - If the user presses the ‘1’ key, change the game state to Level 1.
    - If the user presses the ‘2’ key, change the game state to Level 2.
    - If the user presses the ‘3’ key, restart the current level.
  + GameStateAsteroidsShutdown:
    - Call GameObjectManagerShutdown()
      * Note: This causes all active game objects and game object archetypes to be freed.
  + GameStateAsteroidsUnload:
    - Call GameStateAsteroidsFreeMeshes().
  + GameStateAsteroidsCreateMeshes:
    - Create a “unit”-sized triangular mesh, as follows:

AEGfxMeshStart();

AEGfxTriAdd(

-0.5f, -0.5f, 0xFFFF0000, 0.0f, 0.0f,

0.5f, 0.0f, 0xFFFFFF00, 0.0f, 0.0f,

-0.5f, 0.5f, 0xFFFF0000, 0.0f, 0.0f);

pMeshSpaceship = AEGfxMeshEnd();

AE\_ASSERT\_MESG(pMeshSpaceship, "Failed to create spaceship mesh!");

* + - Create a “unit”-sized triangular mesh, as follows:

AEGfxMeshStart();

AEGfxTriAdd(

-0.5f, -0.5f, 0xFF808080, 0.0f, 0.0f,

0.5f, 0.0f, 0xFFFF0080, 0.0f, 0.0f,

-0.5f, 0.5f, 0xFF808080, 0.0f, 0.0f);

pMeshBullet = AEGfxMeshEnd();

AE\_ASSERT\_MESG(pMeshBullet, "Failed to create bullet mesh!");

* + GameStateAsteroidsFreeMeshes:
    - Free all created meshes.
  + GameStateAsteroidsCreateSpaceship
    - Create a game object with the following parameter:
      * "Spaceship"
    - Create a transform component with the following parameters:
      * Translation: 0, 0
      * Rotation: 0
      * Scale: 50, 40
    - Create a sprite component with the following parameter:
      * "Spaceship Sprite"
      * Set sprite’s mesh.
    - Create a physics component without any additional changes.
    - Create a behavior component by calling BehaviorSpaceshipCreate().
    - Attach the created components to the game object.
    - Call GameObjectManagerAdd(), passing the created game object.
  + GameStateAsteroidsCreateBulletArchetype
    - Create a game object with the following parameter:
      * "Bullet"
    - Create a transform component with the following parameters:
      * Translation: 0, 0
      * Rotation: 0
      * Scale: 10, 10
    - Create a sprite component with the following parameter:
      * "Bullet Sprite"
      * Set sprite’s mesh.
    - Create a physics component without any additional changes.
    - Create a behavior component by calling BehaviorBulletCreate().
    - Attach the created components to the game object.
    - Call GameObjectManagerAddArchetype(), passing the created game object.

GameStateLevel1.c

* You must make the following changes to this file for Project 4:
  + GameStateLevel1Update:
    - If the user presses the ‘3’ key, change the game state to “GsAsteroids”.

GameStateLevel2.c

* You must make the following changes to this file for Project 4:
  + GameStateLevel2Update:
    - If the user presses the ‘3’ key, change the game state to “GsAsteroids”.

BehaviorBullet.h

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

BehaviorBullet.c

* Create an enum with the following entries:
  + cBulletInvalid
  + cBulletIdle
* Add the following private constants:
  + // Maximum speed of the bullet.
  + static const float bulletSpeedMax = 500.0f;
  + // Maximum lifetime of a bullet (in seconds).
  + static const float bulletLifeTimeMax = 3.0f;
* Add the following private function declaration:

static void BehaviorBulletUpdateLifeTimer(BehaviorPtr behavior, float dt);

* BehaviorBulletCreate
  + Calloc the memory for a Behavior structure.
  + If the memory was allocated successfully,
    - Set stateCurr = cBulletInvalid
    - Set stateNext = cBulletIdle
    - Set the onInit, onUpdate, and onExit functions
    - Set the behavior timer = bulletLifeTimeMax
* BehaviorBulletInit
  + This function may remain empty.
  + Use UNREFERENCE\_PARAMETER() to fix the warning.
* BehaviorBulletUpdate
  + Add a switch statement with the expression, (behavior->stateCurr)
  + Add a case for cBulletIdle
    - Call BulletBehaviorUpdateLifeTimer.
* BehaviorBulletExit
  + This function may remain empty.
  + Use UNREFERENCE\_PARAMETER() to fix the warning.
* BehaviorBulletUpdateLifeTimer()
  + Verify that a valid pointer was specified.
  + If the behavior timer > 0,
    - Decrement the behavior timer by ‘dt’.
    - If the behavior timer <= 0,
      * Call GameObjectDestroy(), passing the behavior’s ‘parent’ pointer.

BehaviorSpaceship.h

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

BehaviorSpaceship.c

* Create an enum with the following entries:
  + cSpaceshipInvalid
  + 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 BehaviorSpaceshipUpdateRotation(BehaviorPtr behavior, float dt);
  + static void BehaviorSpaceshipUpdateVelocity(BehaviorPtr behavior, float dt);
  + static void BehaviorSpaceshipUpdateWeapon(BehaviorPtr behavior, float dt);
  + static void BehaviorSpaceshipSpawnBullet(BehaviorPtr behavior);
* BehaviorSpaceshipCreate:
  + Calloc the memory for a Behavior structure.
  + If the memory was allocated successfully,
    - Set stateCurr = cSpaceshipInvalid
    - Set stateNext = cSpaceshipIdle
    - Set the onInit, onUpdate, and onExit functions
* BehaviorSpaceshipInit:
  + This function may remain empty.
  + Use UNREFERENCE\_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,
      * 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 UNREFERENCE\_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 game object
  + 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 += direction of rotation \* spaceshipAcceleration \* dt
  + Calculate the ‘speed’ of this new velocity
    - Hint: speed = length(new velocity)
  + If the speed > spaceshipSpeedMax, then limit the speed:
    - New 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 GameObjectManagerGetArchetype to get the ‘Bullet’ archetype.
    - NOTE: The ‘name’ parameter is case-sensitive!
  + If the bullet archetype was found
    - Clone the bullet archetype.
    - 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 game object manager’s active list.

# 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
    - 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 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>\_Project4.zip
      * Example: CS230S19A\_john.doe\_Project4.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 an empty folder
  + Open the Visual Studio solution file
  + Clean and rebuild the project
  + Verify that the program runs correctly

# 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.