CS230 Project 6 – Asteroids

Due Date

This assignment must be completed and submitted via Canvas before end-of-day on the date listed.

Objectives

The objectives for this project are three-fold:

- To implement simple collision detection between game objects.
- To implement a simple "wave" system for asteroids.
- To implement a more complex behavior using pseudo-inheritance.

Description

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

The game state (Asteroids) created in Project 5 will be used to implement an Asteroids clone in Project 6. The two game states created during Project 4 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 3 through 6, 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 that the Behavior class has been modified to allow for the implementation of pseudo-inheritance in this project. The interactions between this class and the "subclasses" BehaviorSpaceship, BehaviorBullet and the new BehaviorAsteroid inform most of the changes to the existing implementation files.

CS230 Project 6 – Asteroids

Teleporter.h

- This header file declares the public interface for monitoring the movement of game objects and "teleport" them to the other side of the screen when they attempt to leave the viewable area.
- The actual implementation of this function is left up to the student. However, it is recommended that you use an object's velocity to determine which edge of the screen to test against. Additionally, the position of the screen edges can be obtained from the AEGfxGetWinMinX, MaxX, MinY, and MaxY functions.

Random.h

- This header file declares the public interface for generating random numbers, either ints or floats, within specified ranges.

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" game object so that it can access the game object's transform component. Care must be taken to ensure that a cloned collider component points at the newly cloned game object, rather than the archetype game object.
- The actual implementation of collision detection is left up to the student. However, it is recommended that you use a Circle-Circle collision check using the game object's scale (from the transform) to determine a radius. For example:
    - float radius = transform→GetScale().X() / 2.0f;
- When a collision is detected, check the two colliders for CollisionEventHandlers. For each handler found, call the handler, passing references to the two parent game objects. Note, the ordering of the two pointers varies, depending upon which handler is being called:
    - handler(parent, other.parent);
    - other.handler(other.parent, parent);

GameObject.h

- This header file has been updated to include the Collider component. This includes the new functions:
    - SetCollider()
    - GetCollider()
    - IsNamed()

GameObject.cpp

- GameObject (constructor):
    - Initialize Collider component to **nullptr**. (Make sure you're initializing your other components to **nullptr** as well if you aren't already!)
- GameObject (copy constructor):

    - Add code to copy the Collider component.
    - Replace the code that calls the copy constructor for Behavior with a call to the Clone function on the other object's Behavior component:
        - `behavior = other.behavior->Clone(*this);`
- ~GameObject (destructor):
    - Add code to delete the Collider component. (Make sure you're deleting your other components as well if you aren't already!)
    - Replace the code that deletes the Behavior component with a call to the static Destroy function on the Behavior class:
        - `Behavior::Destroy(behavior);`

GameObjectManager.h

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

Engine.cpp

- Init:
    - Add call RandomInit().
- Update:
    - Add call GameObjectManager::GetInstance().CheckCollisions().

Behavior.h

- Note that most of the data in this class is now private. The only classes allowed to access the Behavior's data are itself and its subclasses, BehaviorSpaceship, BehaviorBullet, etc. Those classes are marked as friend classes to allow the appropriate level of access.
- This header file has been updated to include two new variables and two new functions:
    - BehaviorFunctionPtrGo clone;
    - This variable points to the Clone function for the appropriate Behavior subclass.
    - BehaviorFunctionPtr destroy;
    - This variable points to the Destroy function for the appropriate Behavior subclass.
- Clone:
    - The purpose of this function is to allocate a new Behavior object using new.
    - If the Behavior's clone function pointer is valid, simply call that function and return the result.
    - Otherwise, return a new Behavior using the pseudo-copy constructor (constructor that takes a Behavior and GameObject).
- Destroy:

    - The purpose of this function is to destroy a behavior using either the Destroy function for the Behavior subclass or delete.
    - If the Behavior pointer passed into the function is valid and its destroy function pointer is valid, simply call its destroy function. Otherwise, simply delete the Behavior parameter.
- Behavior (constructor, takes a Behavior and Game Object):
    - Modify this function so that it copies the *clone* and *destroy* function pointers from the other Behavior object.

BehaviorBullet.cpp

- This is now a class!
- CollisionHandler(GameObject&, GameObject&);
    - This is a new, private function for resolving a collision between two objects.
    - If gameObject2's name is "Asteroid",
        - Call Destroy on gameObject1
- BehaviorBullet (default constructor, replaces BehaviorBulletCreate):
    - You no longer need to use new to allocate a Behavior since you are in a constructor. Simply set all of the appropriate member variables for the object instance that called the function (this).
    - Make sure to set the *clone* and *destroy* function pointers to the corresponding functions in the BehaviorBullet class.
        - If you're wondering why we don't simply assign the pointers to BehaviorBullet's constructor and destructor, it's because we can't. You can't assign non-static member functions to function pointers and constructors and destructors can't be static by definition.
- BehaviorBullet (non-default constructor, takes Behavior, GameObject)
    - This is a new function for copying existing BehaviorBullet objects. Initialize the base Behavior object using the provided parameters, then perform a shallow copy of any remaining member variables for the BehaviorBullet class.
- Clone:
    - This is a new function for creating clones of existing BehaviorBullet objects. It uses *new* to allocate a new BehaviorBullet via the copy constructor.
    - You'll need to cast the result of the call to the BehaviorBullet constructor to a Behavior pointer before returning it.
- Destroy:
    - This is a new function for destroying BehaviorBullet objects. It uses *delete* to deallocate a BehaviorBullet object.
    - You'll need to cast the Behavior pointer to a BehaviorBullet pointer before deleting it to ensure the correct destructor is called.
- Init:
    - If "stateCurr" is equal to cBulletIdle,
        - Get the parent game object's collider component.
        - If the collider component exists,

        - Set the collider's collision handler to the new private function.
- Update
    - Call TeleporterUpdateObject outside of the switch statement.

BehaviorSpaceship.cpp

- This is now a class!
- Add the following to the spaceship's behavior state enum:
    - cSpaceshipDead
- Reduce the weapon cooldown timer from 0.032f to 0.25f:
    - `const float BehaviorSpaceship::spaceshipWeaponCooldownTime = 0.25f;`
- Set the spaceship's "death" duration to 3 seconds:
    - `const float BehaviorSpaceship::spaceshipDeathDuration = 3.0f;`
- CollisionHandler (GameObject&, GameObject&);
    - This is a new, private function for resolving a collision between two objects.
    - If gameObject2's name is "Asteroid",
        - Set gameObject1's "stateNext" behavior variable to cSpaceshipDead
- BehaviorSpaceship (default constructor, replaces BehaviorSpaceshipCreate):
    - You no longer need to use new to allocate a Behavior since you are in a constructor. Simply set all of the appropriate member variables for the object instance that called the function (this).
    - Make sure to set the *clone* and *destroy* function pointers to the corresponding functions in the BehaviorSpaceship class.
        - If you're wondering why we don't simply assign the pointers to BehaviorSpaceship's constructor and destructor, it's because we can't. You can't assign non-static member functions to function pointers and constructors and destructors can't be static by definition.
- BehaviorSpaceship (non-default constructor, takes Behavior, GameObject)
    - This is a new function for copying existing BehaviorSpaceship objects. Initialize the base Behavior object using the provided parameters, then perform a shallow copy of any remaining member variables for the BehaviorSpaceship class.
- Clone:
    - This is a new function for creating clones of existing BehaviorSpaceship objects. It uses *new* to allocate a new BehaviorSpaceship via the copy constructor.
    - You'll need to cast the result of the call to the BehaviorSpaceship constructor to a Behavior pointer before returning it.
- Destroy:
    - This is a new function for destroying BehaviorSpaceship objects. It uses *delete* to deallocate a BehaviorSpaceship object.
    - You'll need to cast the Behavior pointer to a BehaviorSpaceship pointer before deleting it to ensure the correct destructor is called.
- Init:
    - If "stateCurr" is equal to cSpaceshipIdle,

        - Get the parent game object'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 can be completed with the death duration. The implementation details are left up to the student but the effect should be, at least, somewhat interesting.
- Update
    - Call TeleporterUpdateObject outside of the switch statement.
    - If "stateCurr" is equal to cSpaceshipDead,
        - Decrement the behavior timer by dt.
        - If the behavior timer < 0,
            - Set the next game state = GsRestart
        - Implement the "death" effect, as mentioned above.

BehaviorAsteroid.h

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

BehaviorAsteroid.cpp

- Create an enum with the following entries:
    - cAsteroidInvalid
    - cAsteroidIdle
- 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 SetPosition(BehaviorAsteroid*);`
    - `static void SetVelocity(BehaviorAsteroid*);`
    - `static void CollisionHandler(GameObject*, GameObject*);`
- BehaviorAsteroid (default constructor):
    - Initialize the base behavior variables.
    - Initialize the asteroid behavior's "size" variable to cAsteroidLarge.
    - Refer to your implementations of the other Behavior subclasses (Bullet, Spaceship) for the rest of the implementation.
- BehaviorAsteroid (non-default constructor, takes Behavior, GameObject):
    - Refer to your implementations of the other Behavior subclasses (Bullet, Spaceship) for this implementation.
- Clone:

    - Refer to your implementations of the other Behavior subclasses (Bullet, Spaceship) for this implementation.
- Destroy:
    - Refer to your implementations of the other Behavior subclasses (Bullet, Spaceship) for this implementation.
- Init:
    - Cast the `Behavior*` to a `BehaviorAsteroid*`. This allows you to access members of BehaviorAsteroid.
    - If "stateCurr" is equal to cAsteroidIdle,
        - Set the asteroid behavior's "origin" variable to a random number between 0 and 3.
        - Call the function SetPosition().
        - Call the function SetVelocity().
        - Get the parent game object's collider component.
        - If the collider component exists,
            - Set the collider's collision handler to the new private function.
- Update:
    - Call TeleporterUpdateObject outside of the switch statement.
- CollisionHandler (GameObject*, GameObject*);
    - This is a new, private function for resolving a collision between two objects.
    - If the two pointers are valid,
        - If gameObject2's name is either "Bullet" or "Spaceship",
            - Call the function, GameStateAsteroids::IncreaseScore(20)
            - Call Destroy on gameObject1
- SetPosition:
    - If the asteroid's "size" is cAsteroidLarge,
        - Set the asteroid's position to one of the four corners of the screen, depending upon its "origin" variable.
- SetVelocity:
    - If the asteroid's "size" is cAsteroidLarge,
        - 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.
    - If the asteroid's "size" is cAsteroidMedium or cAsteroidSmall,
        - Generate a random angle between 0 and 359 degrees.
    - Set the asteroid's velocity in the direction of the random angle, with a random speed between asteroidSpeedMin and asteroidSpeedMax.

CS230 Project 6 – Asteroids

GameStateAsteroids.h

- This header file has been modified to include the function, IncreaseScore.

GameStateAsteroids.cpp

- You must make the following changes to this file for Project 5:
    - Add the following private constants:
        - `static const int cAsteroidSpawnInitial = 8;`
        - `static const int cAsteroidSpawnMaximum = 20;`
    - Add the following private variables:
        - `static AEGfxVertexList* pMeshAsteroid = NULL;`
        - `static int asteroidScore = 0;`
        - `static int asteroidHighScore = 0;`
        - `static int asteroidSpawnCount;`
        - `static int asteroidWaveCount;`
    - Add the following private function declarations:
        - `static void CreateAsteroidArchetype(void);`
        - `static void SpawnAsteroidWave (void);`
        - `static void SpawnAsteroid (void);`
        - `static void UpdateScore (void);`
    - Init:
        - Call CreateAsteroidArchetype().
        - Update the high score if the new score is greater.
        - Initialize the score to 0.
        - Initialize the wave count to 0.
        - Initialize the spawn count to cAsteroidSpawnInitial.
        - Call SpawnAsteroidWave().
    - Update:
        - If there are no asteroids in the game object manager's active list,
            - Call SpawnAsteroidWave().
    - IncreaseScore:
        - Increase the current score by "scoreValue".
        - Call UpdateScore().
    - CreateMeshes:
        - Create an asteroid mesh with a minimum of 5 sides.
    - FreeMeshes:
        - Free the asteroid mesh.
    - CreateSpaceship:
        - Add a Collider component to the spaceship game object.
    - CreateBulletArchetype:
        - Add a Collider component to the bullet archetype game object.
    - CreateAsteroidArchetype:
        - Create a game object with the following parameter:
            - `"Asteroid"`

        - Create a transform component with the following parameters:
            - `Translation: 0, 0`
            - `Rotation: 0`
            - `Scale: 40, 40`
        - Create a sprite component with the following parameter:
            - `"Asteroid Sprite"`
            - Set sprite's mesh.
        - Create a physics component with the following parameter.
            - Rotational velocity = (float)M_PI/4.0f
        - Create a behavior component by using the *new* keyword with the BehaviorAsteroid constructor.
        - Create a Collider component.
        - Attach the created components to the game object.
        - Call GameObjectManager::GetInstance().AddArchetype(), passing the created game object.
    - SpawnAsteroidWave:
        - Increment asteroidWaveCount by 1.
        - Call UpdateScore().
        - Call SpawnAsteroid() a number of times equal to asteroidSpawnCount.
        - Increment asteroidSpawnCount by 1, to a maximum of cAsteroidSpawnMaximum
    - SpawnAsteroid:
        - Get the "Asteroid" archetype game object.
        - Clone the asteroid archetype.
        - Add the cloned asteroid to the active game object list.
    - UpdateScore:
        - Build a string with the following parameters:
            - "CS230 Project 6 - Asteroids, Wave = %d, Score = %d, High Score = %d"
            - asteroidWaveCount, asteroidScore, asteroidHighScore
        - Change the window tile using AESysSetWindowTitle().

Optional Features

Implementing one or more of the following features may result in a bonus to the Project grade in the range of +1% to +5%, if implemented correctly.

- Add a reverse thruster state (e.g. `cSpaceshipReverse`) to the BehaviorSpaceship module. This should apply a negative acceleration based upon the spaceship's rotation.
- Implement the drag effect (velocity * 0.99f) discussed during the Week 8 lecture. This drag effect should be applied only to the spaceship.
- When a large- or medium-sized asteroid is destroyed, spawn two new, smaller asteroids in the original asteroid's location. Each new asteroid should be given a new direction

and velocity, as mentioned in the instructions above. [*For grading purposes, avoid making the asteroids so small that the grader will have difficulty completing a wave.*]

Submission Requirements

- The project must build cleanly, with no errors or warnings.
- The Visual Studio project files that you submit must be named Project6*.*. If you are using a previous set of project files, then please rename them by changing the solution and project properties in Visual Studio.
- 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
        - 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 of the following Visual Studio generated folders and files:
        - Folders: "Debug", "Release", "ipch", ".vs" (hidden)
        - Files (*.db, *.sdf, *.opendb)
    - Rename the resultant .zip file using the following naming convention:
        - CS230_<Login ID>_Project6.zip
            - Example: CS230_john.doe_Project6.zip
- Upload the submission .zip file via the Canvas page for your VGP2 section.
- 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 Canvas page for your section of VGP2.
    - Click on the assignment submission link
    - Download the .zip file to your computer
    - Unzip the contents of the .zip file into your project folder
    - Open up the Visual Studio solution file
    - Clean and rebuild the project
    - Test the executable

Assignment Grading Guidelines

- A -25% penalty will be applied to any late submissions. Any submissions that are more than one week late will automatically result in a grade of 0% for the assignment.

- 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.

