GDD 1200 Project Increment 3

Help Policy

The help policy I have for project increments is significantly more strict than for the programming assignments. Some people describe the policy as an "empty hands" policy; basically, you can talk to people about problems you're having, but you're not allowed to type on your computer or write anything down to take away with you after the discussion ends.

Overview

In this project increment, you're adding asteroid spawning and collisions between the ship and the asteroids to the game.

Getting Started

Open GitHub Desktop and make sure the Current Repository on the upper left is set to gdd-1200-<your GitHub user name>. Click Fetch origin near the upper middle to make sure you have the most recent copy of your repo.

Use File Explorer (Windows) or Finder (Mac) to navigate to your local repo. Navigate into the PI3 folder in your repo.

The folder contains a script for you to use for the assignment.

To start this project increment, you can download my solution to Project Increment 2 (available on Canvas) and extract it into the PI3 folder in your repo or you can copy your Asteroids folder from the PI2 folder in your repo into the PI3 folder in your repo.

Step 1: Make an asteroid prefab

For this step, you're making an asteroid prefab. We need an asteroid prefab so the asteroid spawner can spawn asteroids at runtime.

1. Right click in the Project window and create a new Prefabs folder.
2. Drag the Asteroid game object from the Hierarchy window onto the prefabs folder in the Project window.
3. Delete the Asteroid game object from the Hierarchy window.
4. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to drive the ship around, but there aren't any asteroids in the game.

Step 2: Add Direction enum

For this step, you're adding an enumeration so the asteroid spawner will be able to set the direction that an asteroid moves in when it's spawned.

1. Copy the Direction.cs file from the PI3 folder in your repo to the Scripts folder in your Unity project.
2. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, it should run just like it did before.

Step 3: Add Asteroid Initialize method

For this step, you're adding an `Initialize` method to the `Asteroid` class so the asteroid spawner can set the movement direction for an asteroid when it spawns one.

1. We really need to support two things that the spawner will need to do: place the asteroid just outside one of the screen sides and make sure the asteroid is moving in the correct direction. We're using the `Direction` enumeration to support that second piece, so let's implement that first. We'll handle asteroid positioning in a later step.
2. In the `Asteroid` class, create a public `Initialize` method that has a single `Direction` parameter to tell in what direction the asteroid should move.
3. Cut the code that gets the asteroid moving from the `Asteroid` `Start` method and paste it into the body of the new `Initialize` method.
4. Change all the references to `direction` in that code to `moveDirection` instead.
5. Change the code that picks a random angle between 0 and 2*PI radians to pick an angle based on the provided direction parameter. The angle should still be random in a 30 degree arc, so if the provided direction is `Direction.Up` then the random angle you generate should be between `75 * Mathf.Deg2Rad` and `105 * Mathf.Deg2Rad`. The easiest way to do this is to generate a random angle between 0 and 30 degrees (in radians, of course) then add that random angle to the appropriate "base angle" (75 degrees, in radians, for up, for example).
6. Use GitHub Desktop to commit your changes to your local and remote repos.

You won't be able to test this code until we have a new asteroid spawner, so let's move on to that step now.

Step 4: Add asteroid spawner

For this step, you're adding an asteroid spawner.

1. Create a new AsteroidSpawner script in the scripts folder in the Project window and double click it to open it in Visual Studio.
2. Add a documentation comment for the class.
3. Declare a field called `prefabAsteroid` to hold the asteroid prefab you're going to spawn. Be sure to mark the field with `[SerializeField]` so you can populate it in the Inspector.

4. In the Unity editor, attach the AsteroidSpawner script to the main camera and populate the field with the Asteroid prefab.
5. Add code to the `Start` method to spawn an asteroid going up to make sure you implemented that correctly. Remember that you can get access to a script attached to a game object by using the `GetComponent` method.
6. Try spawning asteroids going left, down, and right as well. Debug as necessary.
7. Use GitHub Desktop to commit your changes to your local and remote repos.

We're getting closer to what we actually getting the asteroid spawner to do what we need it to, but we have one more step to go.

Step 5: Spawn four asteroids on four screen sides

For this step, you're having the asteroid spawner four asteroids, one on each side of the screen, moving into the screen.

1. Change the `Asteroid Initialize` method to take an additional parameter, a `Vector3` for the location of the asteroid. Add code in the method to set the position of the asteroid to the parameter.
2. Add code to the `AsteroidSpawner Start` method to instantiate the asteroid prefab, retrieve and save the radius for its circle collider, and destroy the asteroid.
3. Remove the testing code from the previous step from the `AsteroidSpawner Start` method. Add code to create an asteroid, moving left, just outside the right side of the screen. You should find the `ScreenUtils` properties and the asteroid radius you saved above useful as you figure out the position of the asteroid (make the y location the middle of the screen).
4. Run the code to make sure it works properly.
5. Add code to create the other three asteroids outside the top, left, and bottom edges of the screen.
6. Delete the `AsteroidSpawner Update` method.
7. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should see asteroids coming into the screen from the four edges of the screen.

Step 6: Make it so asteroids don't collide with each other

You probably already saw in your testing that the asteroids can collide with each other in the game. That certainly makes sense from a physics perspective, but in the original Asteroids game asteroids just passed over each other instead of colliding. That's the functionality you'll be implementing in this step.

1. We're going to use Layers in Unity to do this. Select the Asteroid prefab in the prefabs folder in the Project window. In the Inspector, left click the Layer dropdown near the top right (it should say Default) and select Add Layer... Type Asteroid in User Layer 8, Ship in User Layer 9, and Bullet in User Layer 10. Ctrl + S to save.

2. Select the Asteroid prefab again. In the Inspector, left click the Layer dropdown near the top right and select Asteroid.
3. Select Edit > Project Settings > Physics 2D from the top menu bar. In the grid at the bottom right in the Inspector, uncheck the box at the intersection of the Asteroid row and Asteroid column (it will show Asteroid/Asteroid if you hold the mouse over the box). This disables collisions between the asteroids.
4. If you look at the Asteroid row in the grid, you'll see it will collide with game objects in the Bullet and Ship layers, which is correct. Select the Ship game object in the Hierarchy window and set the Layer to Ship in the Inspector.
5. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should see that the asteroids don't collide with each other anymore, and they still collide with the ship.

Step 7: Destroy ship when it collides with asteroid

For this step, you're destroying the ship when it collides with an asteroid.

1. Add an Asteroid tag to the Asteroid prefab.
2. Go to the Unity Scripting Reference and look up the documentation for the `MonoBehaviour` `OnCollisionEnter2D` method. Add an `OnCollisionEnter2D` method with the appropriate return type and parameter list to your `Ship` script. Add a documentation comment above the new method. Add code to the body of the method to destroy the `gameObject` the script is attached to if the game object the ship collided with is an asteroid.
3. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should see the ship disappear when it collides with an asteroid. Yes, you might have to drive the ship into an asteroid on purpose!

That's the end of this assignment.

Solution Requirements

Your solution to this problem must:

- Implement all the functionality described above
- Comply with the GDD Coding Standards

Turning In Your Assignment

Navigate to the PI3 folder in your repo. Compress the Asteroids folder into a zip file and submit it in the appropriate assignment on Canvas.

Use the default Windows compression utility for this; the grader may not own WinZip, 7Zip, or whatever other program you're using to build your zip file. If you use something other than the

Windows compression utility and the grader can't unzip your submission, you'll get a 0 on the assignment.

Late Turn-ins

- Turn-ins are due at the beginning of the scheduled class time on the specified due date.
- No late turn-ins will be accepted.