GDD 1200 Project Increment 4

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 bullets 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 PI4 folder in your repo.

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

To start this project increment, you can download my solution to Project Increment 3 (available on Canvas) and extract it into the PI4 folder in your repo or you can copy your Asteroids folder from the PI3 folder in your repo into the PI4 folder in your repo.

Step 1: Slow down the asteroids

For this step, you're slowing down the asteroids (if they're going too fast as they are in my solution to the previous programming assignment). You can slow the asteroids down by reducing the range for the impulse force that's applied to get the asteroid moving.

Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, the asteroids should be moving at a more reasonable speed.

Step 2: Make bullet prefab

For this step, you're making a bullet prefab. We need a bullet prefab so the ship can fire bullets at run time.

1. Add a sprite for your bullet to the Sprites folder and drag it from the Sprites folder onto the Hierarchy window. Rename the game object in the Hierarchy window Bullet.

2. Add a Circle Collider 2D component to the bullet and edit the collider as necessary
3. Drag the Bullet game object from the Hierarchy window onto the Prefabs folder in the Project window.
4. Delete the Bullet game object from the Hierarchy window.
5. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, it should work just like it did at the end of the previous step.

Step 3: Make ship fire bullets

For this step, you're making it so the ship can fire bullets.

1. Open up the `Ship` script in Visual Studio and declare a field called `prefabBullet` to hold the bullet prefab you're going to instantiate when the ship shoots. Be sure to mark the field with `[SerializeField]` so you can populate it in the Inspector.
2. Add an if statement at the end of the `Update` method that uses the `Input GetKeyDown` method to see if the player pressed the `KeyCode.LeftControl` key in this frame. We're not using an input axis for this input because we want to make the player release the key and press it again to shoot another bullet. Add code to the if body to instantiate the `bulletPrefab` with the ship's `transform` as the bullet's `transform`.
3. In the Unity editor, select the Ship game object in the Hierarchy window and populate the Bullet Prefab field in the script with the Bullet prefab.
4. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, when you shoot the bullets they just sit there. That's because we haven't made the bullets move yet; we'll do that soon. You may not realize it, but we're actually detecting collisions between the ship and the bullets. Although that doesn't have any effect on the ship or the bullets, it's actually inefficient. We'll fix that in the next step.

Step 4: Set ship and bullets to not collide

For this step, you're making it so the ship and bullets don't collide with each other. Although we could punish the player for shooting themselves, we won't do that in this case.

1. 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 Ship row and Bullet column (it will show Ship/Bullet if you hold the mouse over the box). This disables collisions between the ship and the bullets.
2. Select the Bullet prefab in the prefabs folder in the Project window and set its Layer to Bullet.
3. This doesn't look quite right because the bullet appears on top of the ship. Select Edit > Project Settings > Tags and Layers from the top menu bar. Expand the Sorting Layers section in the Inspector if necessary. Add a layer named Bullet and another layer named Ship (in that order).
4. Select the Ship in the Hierarchy window and set the Sorting Layer in the Sprite Renderer component to Ship.

5. Select the Bullet prefab in the Project window and set the Sorting Layer in the Sprite Renderer component to Bullet.
6. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to shoot a bullet without destroying the ship. You can drive the ship forward to see the bullet you instantiated. In fact, you could drive the ship around leaving a trail of bullets behind you at this point ...

Step 5: Make it so asteroids are destroyed when they collide with a bullet

In this step, you're making it so asteroids are destroyed when they collide with a bullet. This won't be the final game behavior, because larger asteroids will actually split into smaller asteroids in the final project increment, but it's a step toward that final functionality.

1. Select the Bullet prefab in the prefabs folder in the Project window, add a new Bullet tag to the list of available tags, and set the tag to Bullet. We need this tag because asteroids are only destroyed when they collide with a bullet, not when they collide with the ship.
2. Add an `OnCollisionEnter2D` method to the `Asteroid` script. Add an if statement that checks the tag of the game object that collided with the asteroid is "Bullet" (you should use the `coll` parameter to access the required information). If the game object that collided with the asteroid is a bullet, destroy both the bullet and the asteroid.
3. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should see that the asteroids are destroyed when they collide with a bullet.

Step 6: Make bullets move

For this step, you're making it so the bullets actually move. The tricky part is making it so bullets move in the direction the ship is pointing.

1. Add a Rigidbody 2D component to the Bullet prefab.
2. Add a `Bullet` script to the scripts folder in the Project window and open the script with Visual Studio.
3. Add a documentation comment above the class.
4. Declare a new public `ApplyForce` method with a return type of `void` and a `Vector2` parameter for the direction of the force to apply.
5. Add a documentation comment above the method
6. In the body of the method, declare a constant for the magnitude of the force to apply to the bullet. You'll probably need to experiment with this to get a reasonable number.
7. In the body of the method apply an impulse force of the force magnitude * the force direction (the parameter) to the `Rigibody2D` component.
8. Go to the code in the `Ship` script where you actually fire a bullet. After instantiating the bullet, add code to get the `Bullet` component attached to the bullet game object and call the `ApplyForce` method for that `Bullet` object. You should be happy to know that the

`thrustDirection` field in the `Ship` script is exactly what we need to pass in as the argument to this method.

9. Attach the `Bullet` script as a component of the Bullet prefab.
10. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to shoot moving bullets that go in the direction the ship is facing.

Step 7: Destroy bullets after a certain period of time

For this step, you're making it so each bullet is removed from the game when it's been "alive" for a certain period of time. If we left bullets alive forever, the player could just fill the screen with bullets and just dodge the asteroids until they're all destroyed (especially after we implement screen wrapping for the bullets).

1. Copy the Timer.cs file from the PI4 folder in your repo into the Scripts folder for your project.
2. Open the `Bullet` script with Visual Studio.
3. Add a constant field to hold how long the bullet will live (2 seconds) and a `Timer` field that will hold the "death timer" for the bullet.
4. Add code to the `Start` method to add a Timer component, set the timer duration, and run the timer.
5. Add code to the `Update` method to kill the bullet when the death timer is finished.
6. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to see bullets disappear after about 2 seconds. You can always temporarily change the constant in the `Bullet` script to shorten their lifespan to make this easier to see.

Step 8: Make bullets screen wrap

For this step, you're making it so the bullets screen wrap. This is really easy with our `ScreenWrapper` script!

1. Drag the `ScreenWrapper` script onto the Bullet prefab.
2. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to see bullets screen wrap.

Step 9: Set bullets to not collide with each other

For this step, you're making it so the bullets don't collide with each other; it would look weird for bullets to bounce off each other in the game.

1. 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 Bullet row and

Bullet column (it will show Bullet/Bullet if you hold the mouse over the box). This disables collisions between the bullets.

2. Use GitHub Desktop to commit your changes to your local and remote repos.

When you run your game, you should be able to see that bullets don't collide with each other. You'll probably have to do some driving and shooting to actually see this.

That's the end of this assignment. Just one more increment to go!

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 PI4 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.