

# Exercise 4

For this assignment, you will play with a simple 2D game based on [Boom Blox](#) (predecessor to *Angry Birds*). In the game, you use the mouse to pull back a ball in an invisible slingshot on the lefthand side of the screen. When you let go of it, it flies toward the blocks on the right. You score points for the blocks you knock to the ground. If you want to play a completed version of the game, download one of the solution versions from Canvas. They're just binary versions, not source, so you can't copy code from them. But they'll let you see what the game is supposed to be like.

## ProjectileThrower.cs

Begin by reading the code and familiarizing yourself with its operation

- Implement the procedure `WaitingForPhysicsToSettle()` so that it returns true if there is any `RigidBody2D` that is on screen and awake (in the physics sense).
- Modify `Update()` so that it reloads the level if the projectile has been fired (`firingState` is at `Firing` or beyond) and we aren't waiting for physics to settle.
- Modify `Update()` so it will also reload the level if the player presses the escape key.

## TargetBox.cs

Again, begin by familiarizing yourself the with the code.

- Write an `OnCollisionEnter2D` method to call the `Scored()` procedure when the box hits the ground (the ground object is tagged with the tag "Ground").
- Now write the `Scored()` procedure. It should turn the target box green (set the color field of the game object's sprite renderer to green, and call `ScoreKeeper.AddToScore(float)`, passing it the mass of this targetbox as an argument (so heavier things are worth more).

**Important:** `Scored()` should only call `AddToScore()` once, when that particular `TargetBox` first hits the ground. If it bounces and hits the ground repeatedly, the player should still only get points once, when that box first hits the ground.

## Making an exploding box

Now make a box that explodes if the projectile hits it. There is a second prefab, in addition to `TargetBox`, called `ExplosiveBox`. It contains:

- A `BoxCollider` and `SpriteRenderer` as usual
- A `PointEffector2D` to generate a repulsive force when the box detonates
- A circular trigger collider to work with the `PointEffector2D`. When the point effector activates, all objects in the trigger area are delivered a large repulsive force.
- A `Bomb` component that you need to fill in.

Go to the `Bomb.cs` file. Add:

- A method, let's call it `Destruct()` that destroys the box.
- A method, let's call it `Boom()`, that makes the box blow up by:
  - Turning on the `PointEffector2D` (set its `enabled` field to `true`)
  - Turning off the box's `SpriteRenderer` (so you can't see the box part anymore)
  - Instantiating the explosion prefab by saying:
 

```
Instantiate(ExplosionPrefab, transform.position, Quaternion.identity, transform.parent);
```

 this will create a little explosion animation where the box used to be, but will make it be a child of the box's parent, so that when the box goes away, it doesn't take the explosion with it.
  - Schedules a call to `Destruct()` to occur in 0.1 seconds. You can do that using [Invoke](#).
  - The net effect of this will be to run the point effector for 0.1 seconds, while replacing the box with an explosion animation.
- An `OnCollisionEnter2D()` method that calls `Boom()` when an object hits the box with a velocity of at least `ThresholdForce`. You can get the velocity of the collision from the argument to `OnCollisionEnter2D`. The idea is that you only want the box to explode if it's hit hard enough. We're using the velocity as a proxy for how hard the object is hit.

## Making a new level

Now make at least one new level. Experiment with different designs, and feel free to change parameters such as the mass of the projectile, or the strength of gravity. You do not need to make more than one level, but at least one level should include one of your exploding boxes.

Try to make it engaging; remember that your classmates will be playing it!

## Making sure your code doesn't have issues

Now you want to make sure your code doesn't have any errors in it. First, let's make sure it compiles without any warnings. In Visual Studio, choose `Build>Rebuild Solution` from the menu and make sure the error list at the bottom of the window doesn't have any errors or compiler warnings.

Now go to the Unity window and find the Unity "Console". You'll find it in the Console tab in the bottom pane of the window. This is where exceptions get printed if your code throws an exception. You can also display messages here manually using Unity's `Debug.Log()` method. However, the final code you turn in for your project should not call `Debug.Log()` or otherwise print any messages in the console window.

Run your project. Let it run for a minute or so, pressing buttons and moving the joysticks around, just to make sure no errors happen and you don't have any `Debug.Log()` calls left.

## IMPORTANT: Turning it in

For this assignment, you'll have to turn in your project, not just the CS files (because you're making a new level). To turn it in:

- Make a new directory

- Copy just your Assets, Packages, and Project Settings, and User settings directories into the new directory (copy, don't move)
- Make a zip file of the new directory
- Upload it to canvas

DO NOT INCLUDE THE LIBRARY, TEMP, or LOGS FOLDERS!

## Credits

The [toon explosion animation](#) is thanks to Red Shark Game Studio.