Exercise 5
Issac Newton presents
Crazy Tankxi 1700: *Momentum’s Revenge*

Overview
In this assignment, you will add a number of features to the tank game:

- Joystick control
- Actual physics
- Respawning of mines
- Powerups

Updating the level
Start by adding rigid body and colliders (remember to use the 2D versions) to the tank and projectile prefabs. We’ve listed our settings for sizes, mass, etc., in the appendix. You can deviate somewhat from these if you like, but you should generally stay close to these values.

Connecting your controller
Now connect your game controller to your computer (or two, if you have two). If you have an Xbox 360 controller or equivalent the game should be configured so that:

- Moving the left thumbstick up/down controls the “Vertical” axis, i.e. driving forward/backward
- Moving the right thumbstick left/right controls the “Horizontal” axis, i.e. turning left/right
- The left trigger button deploys your powerup, if you have one
- The right trigger button fires the tank’s gun

Configuring controller input
If you do not have an Xbox 360 controller, you may need to adjust the input mappings. You should also feel free to adjust them to your taste, whatever controller you might have. This is done in two separate areas of the unity editor.

The driving controls (the thumbsticks) are configured using the Edit>Project settings>Input menu. Under Axes, Horizontal1 is the axis for turning tank 1 (Cromartie), Horizontal2, is the axis for turning the other. Similarly, Vertical1 is the forward/backward drive control for tank 1, and Vertical2 for the other.

The firing buttons are configured directly in the Tank Control components of the individual tanks (not in the prefab, since the buttons need to be different for the two tanks). There are two fields here, Fire Button and Power Up Button, for firing and deploying the powerup, respectively. These will be set to “Joystick 1 Button X” for the tank 1 (Cromartie) or “Joystick 2 Button X” for the other tank. You need to try different button numbers until you get the button you want.

Moving the tank around

Now you're ready to start driving the tank around. Go to TankControl.cs and add an Update method.

Driving

Since we're using real physics, we need to control the forward motion of the tank using forces. One thing you could do would be to generate a force pointing forward, whose magnitude is proportional to the joystick setting. However, that means the tank can keep accelerating to an arbitrary speed. So instead, we'll implement a proportional controller or "P-controller" for regulating speed. This means we compute a desired forward velocity, and then apply a force proportional to the difference between our desired forward velocity and our actual forward velocity:

$$F_{\text{forward}} = k(s_{\text{desired}} - s_{\text{actual}})$$

Where k is a magic number that can be tuned to adjust the performance of the tank, and s_{desired} and s_{actual} are the speed we want the tank to be driving forward, and the speed the tank is actually driving forward, respectively. Use the following for these values

- k is the value in the `Acceleration` field
- s_{desired} is the reading on the `VerticalAxis` of the controller times the value in the `ForwardSpeed` field
- s_{actual} is computed from the rigid body's velocity (`tankRb.velocity`) and the forward direction of the tank (`transform.up`)

As we discussed in class, thumbsticks usually don't read as exactly zero even when they're in their neutral positions. So you should use the `DeadZone()` method included in the file to suppress small values from the thumb stick and treat them as zero values.

Steering

Note: when testing this out, you will notice that the tank slides all over the place; don't worry about that for the moment; we'll fix that in the next section.

Humans find physically realistic turning difficult to deal with. So we'll handle steering the way we did before: by directly setting the rotation of the tank. However, since the tank is under physics control, we won't set transform.rotation, we'll instead set the rigid body's rotation field (`tankRb.rotation`).

Write code to turn the tank by updating the rotation. The tank should turn at a speed of `RotationSpeed` times the setting on the `HorizontalAxis`. Once again, you should use `DeadZone()` to keep the tanks from turning when the tumbstick is in the neutral position.

Directional damping

You'll have noticed by now that the tank slides around the field when you turn. In real life, the treads of the tank produce forces that impede motion perpendicular to them. So let's add some directional damping forces. Write code to generate a force perpendicular to the tank (that is, in the direction of `transform.right`), that is opposite to the current velocity in that direction. So in other words:

$$F_{\text{right}} = -ks_{\text{right}}$$

Where s_{right} is the tank's current speed in the direction of `transform.right`. In my implementation, I just used $k = 1$, but you can try different values if you want to tune the performance of the tank (more/less skidding). That said, don't allow too much skidding.

Firing and recoil

Now add code to call FireProjectileIfPossible() when the FireButton key is down (holding down is okay). Note that we've modified the fire routine to apply a recoil force to the tank when it shoots. So the tank will be blown slightly backward by it.

Note: you will notice that projectiles now bounce off of walls and off of each other. This is deliberate; it's not a bug in your code.

Mines

Mines are largely the same as before, but with a couple of new twists.

Making an explosion

First, we'll make the mine generate a nice little explosion. Go to Mine.cs and add code to the OnTriggerEnter2D routine to instantiate the `ExplosionPrefab`. Set the explosion's position to the mine's position, and its rotation to zero (which is called `Quaternion.identity`, for reasons we'll discuss later).

Now go to the ExplosionWithForce prefab, and add a PointEffector2D with a large force (we used 1000), as well as a circle collider with a radius of 3. It already has an ExplosionController component and an animation. Add code to ExplosionController.cs so that it will destroy itself (i.e. the whole explosion game object) after `ExplosionDuration` seconds.

Respawning

Once the mine blows up, we want it to respawn elsewhere. So add another line to its trigger handler to reset the mine's position to `SpawnController.FindFreePosition(2)`. Now go to SpawnController.cs and write the code for FindFreePosition. It should return a random point on screen that is at least radius units away from any other objects. The dimensions of the screen can be found in the fields MinX, MaxX, MinY, and MaxY. You can use Random.Range() to get a random number, and you can use Physics2D.CircleCast to check a point to see if there are any objects nearby.

Writing a custom power-up

The gold star icon on the screen is the teleport power up. You can grab it by driving over it, then deploy it by pressing the power up button on your controller. It teleports you to a new location, then respawns the power up at a random location (using FindFreePosition).

We've written the power up code to be extensible so that power ups are an abstract class, of which teleport power ups are just a particular class. Read the code for the PowerUpController.cs and TeleportPowerup.cs to familiarize yourself with how they work. Now write a new power up class and make a new power up object to do something fun. Here are some ideas:

- A power up that makes you repel all other objects (including projectiles) for 10 seconds
- A power up that makes your opponent *attract* other objects for 5 seconds

- A power up that reverses the direction of all projectiles
- A power up that creates a wall in front of your tank
- A power up that freezes all projectiles in space so that they just sit there and effectively behave like mines (any new projectiles that were fired would behave normally)
- A power up that drops in your current location when you first press the button for deploying it, then when you press it again sucks all objects toward it for 10 seconds except for you (and the walls).

Have fun!

Turning it in

As before, copy your Assets and Project Settings directory into a new directory, and zip that directory. Then upload to Canvas.

Appendix: settings for components in our reference solution

- Prefabs
  - Tank ("Cromartie")
    - Rigid body
      - Mass 1
      - Angular drag 0.5
      - Linear Drag 0.75
    - Box collider
      - Size: X=0.8, Y=0.9
      - Material: TankMaterial
    - Tank Control
      - Forward speed = 20
      - Acceleration 2
      - Rotation speed 200
      - Fire cooldown 0.2
      - Recoil 10
  - ExplosionWithForce
    - Point effector
      - Force magnitude 1000
      - Force mode inverse linear
    - Circle collider
      - Radius 3
      - IsTrigger = UsedByEffector = true
  - Projectile
    - Rigid body
      - Mass 1,000,000 (it's more fun this way)
      - Linear drag 0 (angular drag doesn't matter)
    - Circle collider
      - Radius 0.15

            - Material: ProjectileMaterial

- GameObjects not in prefabs
    - Mine
        - Collider
            - Radius 0.75
            - IsTrigger
    - TeleportPowerup
        - Collider
            - Radius 1.1835