Exercise 2: Moving Tank!s around

Overview

In this assignment, you'll reimplement a simple version of the Tank arcade game. This is a two-player game that you'll initially control using the keyboard, with one player using the arrow keys and the other using the WASD keys. In later assignments, we'll add better physics, joystick support, network support, and perhaps an AI opponent.

You should begin by unzipping the assignment file and deleting the zip file if you haven't already. Now open up the game in Unity (i.e. go into Assets and open the scene file). The game includes code for four new Component types:

- TankControl
  Implements control of the tanks themselves.
- ScoreManager
  Tracks the scores of the players and displays them.
- Projectile
  Implements the shells fired by the tanks
- WallOfDeath
  Implements a collider that destroys whatever hits it (projectiles or tanks).

We're providing you with placeholder files for these. You should update them to implement the following functionality.

The TankControl component

This is the most complicated component, since you have to implement several bits of functionality.

Driving the tank

The component has fields for what keys specify going forward, turning, and firing. Start by implementing an Update() routine that checks the forward and turning keys and updates the position and rotation of the tank accordingly.

To update the position, you'll want to use the transform component, which Unity conveniently has already placed in a field called transform. You can update the position of the tank just by setting **transform.position**. Transform.position is of course a vector. Unity stores a unit vector pointing in the tank's forward direction in **transform.up** (bad name, but it's because Unity started out as 3D only). We've provided a **ForwardSpeed** field for controlling the speed of the tank. Add the appropriate multiple of transform.up to transform.position so as to ensure that the tank goes at the right speed.

Updating the rotation is more of a pain. Unity doesn't have a 2D interface to rotation; it wants you to specify a quaternion (like a complex number, only far weirder), which we haven't talked about yet. So we've built a property into the code called **Rotation** that lets you directly change the rotation of the tank in degrees. We've also included a field RotationSpeed, which specifies the number of degrees per

second to turn. Write code to check the keyboard turning keys and increase or decrease Rotation at the appropriate rate when one of them is pressed.

Shooting

Finally, you'll want to check the firing key (check the fields of the object) and instantiate a Projectile when it's pressed. There are a couple of subtleties to deal with here.

The first is that when you create the projectile, you have to specify a starting position and firing direction for it. You also want to mark who fired it so the game knows who to give credit to when it hits someone. We've provided a method called Init in the Projectile component that takes

- The GameObject that is creating the projectile (not the TankControl but the GameObject containing it)
- The initial position for the projectile, and
- The direction for it to move in

So Instantiate the Projectile prefab, then get the Projective component from inside it, and call its Init method. You should set the initial position to be 1.5 units in front of the tank (remember you can get the tank's position and a unit vector for the forward direction using the information in the previous section).

Important: you haven't yet written an Update() routine for the projectile, so the Projectile won't move yet. If this is annoying you, feel free to skip ahead to implementing the Projectile (below) and then come back.

The other subtlety is that you don't want to allow the player to shoot infinitely quickly. So you want to implement a **cooldown timer** that prevents the player from shooting more often than every **FireCooldown** seconds (FireCooldown is another field in the component). To do that, you want to make a field to hold the next time at which firing will be enabled. Then only fire when the time is past that point in time. When you do fire, also update the field holding the next allowable firing time.

Handling collisions

Finally, write a OnTriggerEnter2D method to detect collisions with the tank. When an object hits the tank, check its **tag**. It's provided in the object's **tag field**, so you can just switch on it:

- If the object is tagged "Projectile", you've been shot! Call ScoreManager.IncreaseScore to increase the score of the person who shot you by 10. You can find out who shot you by looking at the Creator field inside the Projectile component of the gameobject that hit you.
- If it's tagged "Mine", you drove into a mine. Reduce your own score by 20 points.
- In either case, destroy the object you hit.

The Projectile component

Now write an **Update()** routine for the Projectile component to update the position of the projectile so that it moves at the rate specified in the velocity field. Also modify it so that if the projectile moves too far, it destroys itself. In particular, modify it so that if the absolute value of X or Y coordinate of the projectile is more than **Border** (a field we put in to represent how big the world is), then the projectile should destroy itself.

The WallOfDeath

Now modify this component to detect collisions with itself and destroy whatever hits it. If the thing that hit it is a tank, decrease its score by 50.

The ScoreManager component

This has two instance variables for you to deal with:

- Players
  An array of the GameObjects of the different tanks. There are only two in the level we distributed, but we've written the code so that more can be supported.
- Scores
  An array of the different players' scores. It's a parallel array to Players, i.e. the score of the player in position n of Players is stored in position n of Scores.
- theScoreManager
  ScoreManager is an example of the singleton pattern: there's only ever supposed to be one instance of it. Under normal circumstances, we'd just make it a static class. But the Unity editor won't let you edit the fields of a static class. So in order to allow you to specify the players through the Unity editor, we're making it be a singleton component.

All you have to do for the ScoreManager is to fill in the IncreaseScore method. It takes the player and the change in score. Look the player up in the Players array to find its position in the array. That player's score is at the same position in the Scores array. Update it appropriately. Two things to remember:

- IncreaseScore is a static method, meaning it's not run on an instance of an object. However, the Players and Scores are stored in a specific instance. So how do you get a hold of that Players and Scores?
- Once you have the Players array, you can search it using Array.IndexOf(). You don't have to write your own loop to search it (although you can if you like).

Turning it in

When you're finished, make a zip file containing just your TankControl.cs, Projectile.cs, WallOfDeath.cs, and ScoreManager.cs files, and upload it to Canvas.