Exercise 14: Jump That Game Object

Although this exercise isn't worth any points, it gives you valuable programming experience. You're almost definitely going to have to complete the exercises to succeed in the course.

**Problem 1 - Add a game object to the scene**

Create a new 2D Unity project named Exercise14. Add a new scenes folder in the Project window and save the current scene as scene0. Add a new sprites folder in the Project window and use your OS to copy a sprite of your choosing into that folder. Drag the sprite into the Hierarchy window to create a game object in the scene. Run the game and watch nothing happen.

**Problem 2 - Write a script**

Select the game object in the Hierarchy window and change the Transform X value to a value that moves the game object near the right edge of the Game view. Remember that number. Do the same thing to the Transform Y value to move the game object near the top edge of the Game view and remember that number as well.

Side note: We can actually get the values of the edges of the screen using the Screen class within our scripts, but let's focus on selection in this exercise.

Create a new scripts folder in the Project window and create a new C# script in that folder called Jumper. Open the new script in your IDE and add a documentation comment for the class.

Add a //  jump  location  support comment and declare float minX, maxX, minY, and maxY constants just above the Start method in the script. We'll use these constants to make sure our game object stays in the Game view when we jump it to a new location, so give those constants appropriate values based on your changes to Transform X and Y at the beginning of this problem.

Delete the Start method and make sure your code compiles without errors.

Add a //  timer  support comment and declare a float TotalJumpDelaySeconds constant and set it to 1 just below your other constants (remember to leave a blank line above your comment). Declare a float elapsedJumpDelaySeconds variable and set it to 0 just below your new constant.

Add a //  update  timer  and  check  if  it′s  done comment at the start of the Update method and add code below the comment to update the elapsedJumpDelaySeconds based on the amount of time the previous frame in the game took to execute. Luckily, we can get that precise information using Time.deltaTime. The elapsed time goes up as the game runs, so adding this value to the current value of elapsedJumpDelaySeconds is the correct approach.

Add an if statement just below the line of code above that checks to see if the timer is done. The Boolean expression for the if statement should check to see if elapsedJumpDelaySeconds is greater than or equal to TotalJumpDelaySeconds. Make sure you understand why this Boolean expression will be true when the timer is done.

In the if body of the code you wrote in the previous step, add code to reset the timer (setting elapsedJumpDelaySeconds to 0 resets the timer) and change the location of the game object the script is attached to a new random X and Y location (this is where you use the minX, maxX, minY, and maxY constants). The easiest way to do this is to set transform.position to a new Vector3 object with a random X value, a random Y value, and 0 for Z.

Run the game and swear because nothing happens.

**Problem 3 - Finish the solution and play around**

Attach the Jumper script to the game object in the Hierarchy window. Run the game and watch your game object jump to a new location every second.

Change the TotalJumpDelaySeconds constant to something very small to see some freaky, epilepsy-inducing behavior!