Exercise 1: 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.

Get Started – Clone the Repository

1. Click on the appropriate link then accept the assignment to create your repository for submitting your work:
    a. **Gallant AM**: https://classroom.github.com/a/BsQTiJDf
    b. **Gallant PM**: https://classroom.github.com/a/rI-gLIxC
    c. **Nunn AM**: https://classroom.github.com/a/T5VpcU2B
    d. **Nunn PM**: https://classroom.github.com/a/FFwZKfVv
    e. **Wijaya AM**: https://classroom.github.com/a/XD5AQbkY
    f. **Wijaya PM**: https://classroom.github.com/a/Irtl6IFy
2. In GitHub Desktop, clone the repository you just created to your desktop.

Create your Unity project and prepare for GitHub tracking

3. Use Unity Hub to create a new 2D Unity project named Exercise1. Save the project in your new repository folder.
4. Once the project opens in Unity, go to File Explorer and move the **_UnityProjectRoot.gitignore** file into the Unity project folder and rename it to **.gitignore**
5. Go to GitHub desktop and commit your changes with the message: "Create initial Unity project".
    a. Make sure there are only about 30 files being committed.
    b. If you have thousands of changed files, return to step 2 to make sure you've named the gitignore file properly and that it is placed at the root of the *Unity* project not in its original location.
    c. **Ask for help if you are unsure**.
6. Push your changes to the remote.

At this point you are ready to proceed with this assignment. We encourage you to make interim commits as you go. Use your commit message to indicate which step (e.g.: "Completed through step 5").

Problem 1 - Add a game object to the scene

7. Rename the SampleScene as Scene0.
8. Add a new Sprites folder in the Project window and use your OS to copy a sprite of your choosing into that folder.
9. Drag the sprite into the Hierarchy window to create a game object in the scene.

10. Run the game and watch nothing happen.
11. In GitHub Desktop, commit your work with message: “Completed problem 1”.

Problem 2 - Write a script

12. 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.
    a. 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.
13. 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 Visual Studio and add a documentation comment for the class.
14. 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.
15. Delete the `Start` method and make sure your code compiles without errors.
16. 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.
17. 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.
18. 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.
19. 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 best way to do this is to set a local variable (I called mine `position`) to `transform.position`, change the `x` and `y` properties of `position` to random X and Y values, then set `tranform.position` to `position`.
20. Run the game and swear because nothing happens.
21. In GitHub Desktop, commit your changes with the commit message: “Completed problem 2”.

Problem 3 - Finish the solution and play around

22. In Unity Editor, 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.
23. Change the `TotalJumpDelaySeconds` constant to something very small to see some freaky, epilepsy-inducing behavior!
24. Make sure you save your project.

Submit your work

25. In GitHub Desktop, commit your changes with the commit message: “Completed problem 3”.
26. Push your changes to the remote.
27. Return to CodeHS and respond to the prompt, then submit and continue to the next lesson.