Lab Assignment #1: Extended Roll-A-Ball Game

In this first lab assignment we will take two weeks. This will allow us to make sure that you are set up and you can work through any technical problems. And, for you to get used to following the written instructions *carefully*.

The lab assignment will consist of two parts. In part 1, to be completed by **January 17** you will complete an introductory tutorial. In part 2, you will extend the work you have done and improve your game. You will submit both part 1 and Part 2 by **January 24** on GitHub.

Unity uses C# not Java

Java and C# are so close we will not cover the differences in class formally, you are expected to pick up on this on your own. Students have had no problem making this transition in past years. However, here is a great reference on C# for Java Developers that covers the main differences, it is worth skimming this before you go any further.

Part 1 - to be completed January 17

The goal of first lab is to get every one acquainted with Unity and the lab environment. There will be nothing to submit for Part 1.

First, you will need to initiate the project by following the GitHub invitation link, cloning it to your local computer and opening it in Unity. See Instructions for Working with GitHub.

Here's the invitation link for Unity Version 2021.3.4f1 (on the lab machines): Invite

Once you have opened up your project in Unity, you can start the Roll-A-Ball Tutorial.

Note: Watch but do not complete the steps in the very first video ("Setting up the Game"), since you do this by cloning the repo from GitHub and opening it in Unity. Also, you do not need to complete the steps in the final section ("Building the Game"). Instead you will submit your project on GitHub.

Part 2 - to be completed January 24th

In Part 2, you will Extend your Roll-A-Ball game so that it is more like a game. At the end of this assignment your game will be able to start and restart, and you will be able to "win" or "lose" the game!

1. Add a Count Down Timer

You will add a timer to the game, so that if the timer counts down to zero you lose the game.

Requirements

- The current time remaining will be displayed in the top right hand corner
- The time should be initialized to 30 seconds when the game starts (this should make your game really easy to win)
- The timer will count down to zero
- If zero is reached before all pickups are collected then the message "Time's Up! You Lose!" is displayed.

- The timer should stop if the game is lost... or won.

Steps/Hints

- Similar to adding the score text, add a new UI TextMeshPro element for the time, that is displayed in the top right corner of the screen.
- Add in variables for tracking the time to your player controller class.
- Add an Update method to your player controller script (similar to your FixedUpdate method). With this method you will be able to update UI and game state information, it is run with each frame of the game.
- You can get time for the game by using - You can get the current time for the game by using a property of the Time class.

```
(int) Time.timeSinceLevelLoad;
```

- You may want to create a function to update the time text, similar to what you did with the score.
- Your timer should stop counting down at zero or when the game is won.
- The losing message (above) could be displayed using the winning text UI element that has already been created.

2. Make Your Game More Game-Like

Requirements

- Your game should not start immediately, instead add a welcome message that says "Welcome to -Your Name-'s Roll-A-Ball. Push 'S' to Start."
    - Use your actual name in the message above
- When 'S' is pushed your game will start: the timer will start counting down, and the welcome message will disappear.
- Before 'S' is pushed you should not be able to move the ball, but the timer and score text should be displayed properly.
- Before starting and at the completion of the game (win or lose), the ball should not be able to be controlled any more... i.e., the controls won't work.
- Add an additional message to the end game message (i.e., the win or lose message) that says "Push R to Restart."
- When you push 'R' your game will restart
- The code snippet below provides important methods about meeting the requirements above.
- If you find your text is too long to be displayed then have a look at the properties of the Win Text in the Inspector.
- Consider adding two bool variables to your class to indicate the current game state (e.g., gameStarted and gameOver), and provide appropriate if statements to enable the correct game behaviour.

```
//the method call below returns a bool indicating if a key has
//been pushed since the last frame
Keyboard.current.sKey.wasReleasedThisFrame
Keyboard.current.rKey.wasReleasedThisFrame
```

```
//reload your game scene using the following line
//see the hint above for dealing with a darker scene after reload
SceneManager.LoadScene( SceneManager.GetActiveScene().name );

//to use the line above you will need to first import the package
//containing the SceneManager
using UnityEngine.SceneManagement;
```

Test and Submit

- Playtesting is an essential part of game development. Play your game... a lot. Does it work all the time? If not, see if you can fix any problems. Give your self enough time to ask questions on Teams, if you are stuck.
- Once you are happy with your work. Save it in Unity and then Commit and Push it to GitHub by following the instructions on the course website.