Exercise 3: Using CSV Files

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.

**Getting Started**

Download the zip file below and unzip the file somewhere on your computer.

**1 3 Exercise 3 Materials**

ZIP File

Open the project in Unity. You can do this by navigating to the Exercise3\Assets\scenes folder and double clicking scene0.

Run the game to see how it works.

Work your way through the Unity editor and the included scripts to make sure you understand how they work. The ConfigurationData class isn’t actually used at this point, but you’ll change that soon! Notice that the Driver Update method accesses the ConfigurationUtils TeddyBearMoveUnitsPerSecond property and the Shooter Start method accesses the ConfigurationUtils CooldownSeconds property.

**Problem 1 – Start using the ConfigurationData class**

Add a static field to the ConfigurationUtils class to hold a ConfigurationData object.

Add code to the ConfigurationUtils Initialize method to call the ConfigurationData constructor to populate your new field. Change all the ConfigurationUtils properties to return the appropriate properties from your new field instead of the hard-coded values they currently return.

Run your game. It should work just like it did before you made these changes.

**Problem 2 – Create the configuration data file**

Create a new StreamingAssets folder (capitalize it exactly as shown) in your Project window; this is where you’ll save your csv file. Putting the file there makes it automatically get included in our build and also makes it so we can use Application.streamingAssetsPath to get the file location from our script, which will work both in the editor and if you distribute your game.

Create a csv (comma-separated value) file containing the configuration data; you should call it ConfigurationData.csv (or you can change the file name in the ConfigurationData class). The first line in the file should be a comma-separated list of the value names and the second line in the file should be a comma-separated list of the values. This is the format you’d expect to see exported from a spreadsheet that game designers were using to tune the game. The ConfigurationData class only has two fields for configuration data, so you only need to include names and values for those two fields in the file.

Save the csv file in your StreamingAssets folder.

**Problem 3 – Use configuration data from file**

Add code to the ConfigurationData constructor to open the configuration data file using Application.streamingAssetsPath and your file name as the full file name. Be sure to include this code in an exception handler. You should have a catch block that catches all exceptions (but doesn’t do anything in the catch block) and you should have a finally block that closes the file if it’s not null. You’ll need to add a System.IO using directive to get access to the StreamReader class and a System using directive to get access to the Exception class.

Add code to extract the values from the second line in the file and populate the fields with them. You should do this in the SetConfigurationDataFields method I provided in the ConfigurationData class.

When you run your, game it should work just like it did before this step.

Note: You should make sure this is working by changing values in your csv file and making sure those changes have the expected effect in the game (be sure to close the csv file before running your game). This should work fine in the editor and for standalone Windows and Mac builds, but WebGL builds don’t seem to read from the csv file even though it’s included in the build. Numerous people have posted this problem on the web, but from what I can tell no one has ever answered those posts. That’s why we’ve made sure the ConfigurationData class works with default values if the file read fails.