Victor Haskins

Game Programming 2

Lab 4 Report and Post-Lab Questions

Introduction

This lab focuses on the class modifying an existing program, a version of Wheel of Fortune, to work read in user generated content from a text document with a specified format. It focused not only on how to properly implement this, but to also add some error checking and an embedded file that will work as a fallback if the UGC document isn't recognized or is misfiled.

After that, we create a simple trivia game using the same techniques to read in the content and display it as necessary, also showing the total number of questions, and how many questions have been answered correctly.

Methods

After running the game a few times, we are guided through the process of reading in an embedded file. We first need to add the libraries System.IO, for file reading, and System.Collections.Generic, for using lists. We then create a TextAsset variable for use in reading our text file. After it is initialized to our embedded file, it can be called with a textReader variable that is set as a StringReader for the TextAsset. We then create string lists to house the information and set a while loop that checks if there are still lines of text to read in and stores odd lines in the "sentences" list while the even numbers have clues. After each line is read into their respective lists, a counter is incremented to keep track of readings.

Next is to have another file provided, the Resources file, call its Gather() function and use the text read in from the embedded file on the LoadScript to create the game level. We start by calling the SendMessage("Gather"); function but that alone will not work. It will only send messages to scripts and objects on the same object. To make sure that the LoadScript will have a target for the message, we include the line [RequireComponent(typeof(References))] directly above the class's start. As it is now, the embedded file will be read and the new data is incorporated into the Wheel of Fortune game.

The next big step is to incorporate UGC files. It is important to have a file that will explain the UGC format to the user so it can be read into the game properly. We then add code around the initialization of our TextAsset by first checking for a file called "sentences.txt" to be stored in the application datapath. This is done by creating a new global variable of the type "FileInfo" and initializing it to the datapath with the file "sentences.txt" If the file is found and not equal to null, we set the TextAsset to that file instead of the embedded file.

With a new file created, we test if the code works by building the project, finding where the data and application are stored, opening the "buildData" folder, placing the sentences text file there, and then run the application. If the game was set up corectly, it should read from your new UGC input.

Lastly, we create our own little trivia game by reading information just like the LoadScript file, display it to the screen along with supplementary material such as the total number of questions and the number of which were correctly guessed. We should have an embedded file as a fallback.

Conclusion

It is important to be able to create tools that can quickly and reliably read information from the user to create new content. The lab was a good launching point for the class.

Post-Lab

*1. Why are lists used instead of arrays in the LoadScript?*

List elements are easier to add or remove and the lengths of lists can be dynamically altered with these additions/subtractions.

*2. What other method(s) could you use to load the data from an external file?*

StreamReader

*3. What is the purpose of having two loading systems in your load script file?*

Redundancy. You want to have a fallback if the user document is incompatible, mislabeled, or misfiled.

*4. Will FileInfo ever be null? Why/Why not?*

If the file exists, but there is nothing in it; a blank file.

*5. How could you allow the player to tell the engine where their file is that contains their custom sentences/phrases?*

Offer an option at the beginning of the game for the user to either enter the game's location or to browse for it.