**Lab Report**

**Lab 4 - User Generated Content**

**Craig Broskow**

**September 23, 2015**

Introduction:

User generated content plays an important role within the gaming industry. At its simplest, user generated content (UGC) allows gamers to create modifications of varying sophistication ("mods") to games which have already been released to the public. This capability has been available for a number of games for quite a few years.

However, the gaming industry has also seen increased competition recently in seeking to engage customers (i.e., game owners) to their games in more personalized ways. This increased emotional attachment not only gives game companies increased opportunities to sell peripheral products (such as DLC), but also improves customer satisfaction with the game and its potential franchise. UGC is an important avenue to allow gamers to customize their games to their own personalities and within their circle of gaming associates (friends, competitors, school or workmates, etc.).

Methods:

In the simple games in this lab, like "Wheel of Fortune" and "Trivial Quest", much of the code is involved with handling the user interface. However, those methods have been documented in previous labs and primarily involve displaying text and images, and responding to GUI events such as button clicks, etc.

The real meat of this lab involved the processing of text files, and the handling of "internal" or "embedded" text files that come with the game versus the handling of "external" or user-generated text files that are provided after the game has been built, but within the file structure that Unity has ready access to.

Embedded text files are accessed using the built-in Unity class "TextAsset" via the convenient "Resources.Load" method call. On the other hand, user generated content is handled slightly differently: the text file is stored in the folder path specified by the Unity property "Application.dataPath", and is accessed through the built-in "FileInfo" Unity class using the "OpenText" method call. In both cases, lines of text are read in using StringReader classes. For embedded text files, a stock StringReader class is used while for UGC text files the more specialized TextReader class is used.

Once the embedded or UGC text file is associated with a StringReader object, there is no difference in further processing. Both types of text files are read in line-by-line using the "ReadLine" method of the StringReader (or TextReader) object. The lines of text are read into string lists, and either processed directly using the string lists...or converted into equivalent string arrays and processed using arrays. String lists are used initially because they are more flexible than arrays when the total number of elements to be stored are not known at design/build time. However, once the string lists have been populated at run-time, the number of text lines is known and the string lists can be replaced by slightly more efficient string arrays.

At this point, all embedded or UGC text has been read, and the user interface can be customized for the actual playing of the game. The primary task of reading in text into string lists/arrays has been accomplished. In the case of "Wheel of Fortune", the GUI is populated with the letters that make up strings and they are uncovered as the game is played. For "Trivial Quest", the GUI displays the question as a text element, while the answers are displayed as text on buttons for the player to choose between. Score keeping and determining won/lost outcomes is straight-forward.

Conclusions:

The Unity C# code developed in this lab is primarily concerned with reading in text files for processing within Unity games. Embedded text files can provide "default" values or data sets, while external files can be read using very similar techniques...and allowing gamers to create their own user generated content to override the default values and data sets that came packaged with the game.

Although the techniques used in this lab are relatively simple, they provide substantial opportunity to expand the capability of games after they have been released...both to the benefit of gamers, and the companies that provide the extra level of customization that can increase the perceived value of their games.

Postlab Questions:

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

We don't know how many phrases and clues are contained in the sentences file before we start reading it. Lists are more flexible than arrays in that they do not need to be initialized to a certain element count. (Dynamic arrays are less efficient than lists because they need to be copied every time their size is changed.) Lists support the use of the "Add" method to efficiently add phrases and clues to the lists as they are read. Finally, lists can also be accessed by an index value (as is done in the StartGame method of the Engine class) so they can be used much like arrays.

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

The "text" property of a TextAsset object will load all the data from an external file, and it can then be parsed into the individual phrases and clues. Alternatively, the "ReadToEnd" method of a TextReader will do the same thing using standard C#. Although less efficient, TextReaders can read individual characters or blocks of characters using the "Read" or "ReadBlock" C# methods.

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

If the user wishes to customize the game with their own phrases and clues, they can include a "sentences" text file. However, not all users will do that, so the load script file will read the standard "embedded" text file whenever the user has not created or imported a custom text file.

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

As far as I can tell, it would be very unusual for FileInfo to be null, but possible. If for some reason that Unity data folder did not exist, or if the user did not have permission to read from that folder, I think FileInfo would be null. However, if the game installation completed successfully and the user was able to run the game, I don't think FileInfo would ever be null. Still, it does seem to be possible.

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

The easiest way (programming-wise) would be to provide a text box that the user could type in the path to their custom text file. However, that would be very error-prone and not user-friendly. A better way would be to use Microsoft's OpenFileDialog sub-class within the Windows.Form class to open a file-picker dialog box. I don't think Unity provides a file-picker dialog box, although there may be some available in the Asset Store.

Code:

// TriviaEngine.cs

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

using UnityEngine.EventSystems;

// Force unity to add the TriviaReferences script if it is not already on the object

[RequireComponent(typeof(TriviaReferences))]

public class TriviaEngine : MonoBehaviour {

const int NUM\_ANSWERS = 4; // number of answers per question

TriviaReferences refs;

private int questionCount = 0;

private int rightCount = 0;

private int wrongCount = 0;

private int currentQuestion = 0;

// private GameObject canvasObject;

private GameObject button1;

private GameObject button2;

private GameObject button3;

private GameObject button4;

private Text button1Text;

private Text button2Text;

private Text button3Text;

private Text button4Text;

private Text questionText;

private Text questionNumberText;

private string outputString;

private string[] questionArray;

private string[] correctAnswers;

private string[] wrongAnswers1;

private string[] wrongAnswers2;

private string[] wrongAnswers3;

private int correctButton = 0; // number of button containing the correct answer

private bool waitingForKey = false;

void Begin()

{

//Gets a link to the TriviaReferences script

refs = gameObject.GetComponent<TriviaReferences>();

// canvasObject = GameObject.Find("Canvas");

button1 = GameObject.Find("Answer1Button");

button2 = GameObject.Find("Answer2Button");

button3 = GameObject.Find("Answer3Button");

button4 = GameObject.Find("Answer4Button");

button1Text = button1.GetComponentInChildren<Text>();

button2Text = button2.GetComponentInChildren<Text>();

button3Text = button3.GetComponentInChildren<Text>();

button4Text = button4.GetComponentInChildren<Text>();

questionText = GameObject.Find("QuestionText").GetComponent<Text>();

questionNumberText = GameObject.Find("QuestionNumber").GetComponent<Text>();

questionArray = GetComponent<TriviaReferences>().questionArray;

correctAnswers = GetComponent<TriviaReferences>().correctAnswers;

wrongAnswers1 = GetComponent<TriviaReferences>().wrongAnswers1;

wrongAnswers2 = GetComponent<TriviaReferences>().wrongAnswers2;

wrongAnswers3 = GetComponent<TriviaReferences>().wrongAnswers3;

questionCount = questionArray.Length;

//Starts the game

StartGame();

}

public void PressButton1()

{

if (waitingForKey)

{

waitingForKey = false;

CheckButtonPress(1);

}

}

public void PressButton2()

{

if (waitingForKey)

{

waitingForKey = false;

CheckButtonPress(2);

}

}

public void PressButton3()

{

if (waitingForKey)

{

waitingForKey = false;

CheckButtonPress(3);

}

}

public void PressButton4()

{

if (waitingForKey)

{

waitingForKey = false;

CheckButtonPress(4);

}

}

public void CheckButtonPress(int buttonNumber)

{

if (correctButton == buttonNumber)

rightCount++;

else

wrongCount++;

currentQuestion++;

if (currentQuestion < questionCount)

{

correctButton = Random.Range(1, NUM\_ANSWERS + 1);

DisplayQuestionAndAnswers(correctButton, currentQuestion);

}

else

{

TriviaGameHelperScript.SetCorrectCount(rightCount);

TriviaGameHelperScript.SetWrongCount(wrongCount);

LoadGameOverScene();

}

} // end method CheckButtonPress

private void DisplayQuestionAndAnswers(int correctButton, int questionNumber)

{

questionText.text = questionArray[questionNumber];

switch (correctButton)

{

case 1:

button1Text.text = correctAnswers[questionNumber];

button2Text.text = wrongAnswers1[questionNumber];

button3Text.text = wrongAnswers2[questionNumber];

button4Text.text = wrongAnswers3[questionNumber];

break;

case 2:

button1Text.text = wrongAnswers1[questionNumber];

button2Text.text = correctAnswers[questionNumber];

button3Text.text = wrongAnswers2[questionNumber];

button4Text.text = wrongAnswers3[questionNumber];

break;

case 3:

button1Text.text = wrongAnswers1[questionNumber];

button2Text.text = wrongAnswers2[questionNumber];

button3Text.text = correctAnswers[questionNumber];

button4Text.text = wrongAnswers3[questionNumber];

break;

default:

button1Text.text = wrongAnswers1[questionNumber];

button2Text.text = wrongAnswers2[questionNumber];

button3Text.text = wrongAnswers3[questionNumber];

button4Text.text = correctAnswers[questionNumber];

break;

}

ShowQuestionNumber();

waitingForKey = true;

} // end method DisplayQuestionAndAnswers

public void StartGame()

{

rightCount = 0;

wrongCount = 0;

currentQuestion = 0;

correctButton = Random.Range(1, NUM\_ANSWERS + 1);

DisplayQuestionAndAnswers(correctButton, currentQuestion);

} // end method StartGame

private void ShowQuestionNumber()

{

outputString = "Question " + (currentQuestion + 1).ToString() + " / " + questionCount.ToString();

questionNumberText.text = outputString;

}

private void LoadGameOverScene ()

{

Application.LoadLevel("GameOverScene");

}

} // end class TriviaEngine

// TriviaLoadScript.cs

using UnityEngine;

using System.Collections;

using System.IO;

using System.Collections.Generic;

[RequireComponent(typeof(TriviaReferences))]

public class TriviaLoadScript : MonoBehaviour {

FileInfo originalFile;

TextAsset textfile;

TextReader reader;

public List<string> questions = new List<string>();

public List<string> answers = new List<string>();

// Use this for initialization

void Start () {

originalFile = new FileInfo(Application.dataPath + "/questions.txt");

if (originalFile != null && originalFile.Exists)

{

reader = originalFile.OpenText();

}

else

{

textfile = (TextAsset)Resources.Load("embedded2", typeof(TextAsset));

reader = new StringReader(textfile.text);

}

string lineOfText;

int lineNumber = 0;

while ((lineOfText = reader.ReadLine()) != null)

{

lineOfText = lineOfText.Replace("''", "'");

if (lineNumber%5 == 0)

questions.Add(lineOfText);

else

answers.Add(lineOfText);

lineNumber++;

}

SendMessage("Gather");

} // end method Start

} // end class TriviaLoadScript

// TriviaReferences.cs

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

using System.Collections.Generic;

public class TriviaReferences : MonoBehaviour {

//Variables to store all information, hidden from designer

[HideInInspector]

public List<string> questions = new List<string>();

[HideInInspector]

public List<string> answers = new List<string>();

[HideInInspector]

public string[] questionArray;

[HideInInspector]

public string[] correctAnswers;

[HideInInspector]

public string[] wrongAnswers1;

[HideInInspector]

public string[] wrongAnswers2;

[HideInInspector]

public string[] wrongAnswers3;

void Gather () {

int numQuestions = 0; // number of questions in input file

int answerCounter = 0; // counter for current element in answers list

questions = GetComponent<TriviaLoadScript>().questions;

answers = GetComponent<TriviaLoadScript>().answers;

numQuestions = questions.Count;

questionArray = new string[numQuestions];

correctAnswers = new string[numQuestions];

wrongAnswers1 = new string[numQuestions];

wrongAnswers2 = new string[numQuestions];

wrongAnswers3 = new string[numQuestions];

for (int i = 0; i < numQuestions; i++)

{

questionArray[i] = questions[i];

correctAnswers[i] = answers[answerCounter];

answerCounter++;

wrongAnswers1[i] = answers[answerCounter];

answerCounter++;

wrongAnswers2[i] = answers[answerCounter];

answerCounter++;

wrongAnswers3[i] = answers[answerCounter];

answerCounter++;

}

TriviaGameHelperScript.SetCorrectCount(0);

TriviaGameHelperScript.SetWrongCount(0);

//Inform the Engine to begin the game!

SendMessage("Begin");

} // end method Gather

} // end class TriviaReferences

// TriviaGameHelperScript.cs

using UnityEngine;

using System;

using System.Collections;

public static class TriviaGameHelperScript {

public static int correctCount {get; private set;}

public static int wrongCount {get; private set;}

public static void SetCorrectCount (int newCorrectCount)

{

correctCount = newCorrectCount;

}

public static void SetWrongCount (int newWrongCount)

{

wrongCount = newWrongCount;

}

} // end class TriviaGameHelperScript

// TriviaMainMenuScript.cs

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;

using System.Collections;

public class TriviaMainMenuScript : MonoBehaviour {

public void LoadGameScene ()

{

Application.LoadLevel("GameScene");

}

public void LoadInstructionsScene ()

{

Application.LoadLevel("InstructionsScene");

}

public void QuitGame ()

{

Application.Quit();

Debug.Log("Quit button clicked!");

}

} // end class TriviaMainMenuScript

// TriviaInstructionsScript.cs

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;

using System.Collections;

public class TriviaInstructionsScript : MonoBehaviour {

public void LoadGameScene ()

{

Application.LoadLevel("GameScene");

}

public void LoadMainMenuScene ()

{

Application.LoadLevel("MainMenuScene");

}

}

// TriviaGameOverScript.cs

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.EventSystems;

using System.Collections;

public class TriviaGameOverScript : MonoBehaviour {

// Use this for initialization

void Start () {

ShowCorrectCount();

ShowWrongCount();

}

public void ShowCorrectCount()

{

GameObject.Find ("CorrectCountText").GetComponent<Text>().text =

"Correct: " + TriviaGameHelperScript.correctCount.ToString();

}

public void ShowWrongCount()

{

GameObject.Find ("WrongCountText").GetComponent<Text>().text =

"Wrong: " + TriviaGameHelperScript.wrongCount.ToString();

}

public void QuitGame ()

{

Application.Quit();

Debug.Log("Quit button clicked!");

}

}