**在Unity3D实现的文件浏览器脚本**

Posted on 2013年01月18日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 287 次

TextFileFinder.cs

using UnityEngine;  
using System.Collections;  
public class TextFileFinder : MonoBehaviour {  
public static bool flag;  
public GUITexture text;  
protected string m\_textPath;  
public GUISkin skin;  
protected FileBrowser m\_fileBrowser;  
[SerializeField]  
protected Texture2D m\_directoryImage,  
m\_fileImage;  
protected void OnGUI () {  
GUI.skin=skin;  
if (m\_fileBrowser != null) {  
m\_fileBrowser.OnGUI();  
} else {  
OnGUIMain();  
}  
}  
void Update()  
{  
if(flag)  
{  
StartCoroutine(WaitLoad());  
flag=false;  
}  
}  
IEnumerator WaitLoad()  
{  
WWW aa=new WWW("file:///"+m\_textPath);  
print(aa.url);  
yield return aa;  
text.texture=aa.texture;  
}  
protected void OnGUIMain() {  
GUILayout.BeginHorizontal();  
GUILayout.Label("Text File", GUILayout.Width(100));  
GUILayout.FlexibleSpace();  
GUILayout.Label(m\_textPath ?? "none selected");  
if (GUILayout.Button("...", GUILayout.ExpandWidth(false))) {  
m\_fileBrowser = new FileBrowser(  
new Rect(10, 10, 400, 300),  
"Choose Text File",  
FileSelectedCallback  
);  
m\_fileBrowser.SelectionPattern = "\*.txt";  
m\_fileBrowser.DirectoryImage = m\_directoryImage;  
m\_fileBrowser.FileImage = m\_fileImage;  
}  
GUILayout.EndHorizontal();  
}  
protected void FileSelectedCallback(string path) {  
m\_fileBrowser = null;  
m\_textPath = path;  
}  
}

GUILayoutx.cs

using UnityEngine;  
public class GUILayoutx {  
public delegate void DoubleClickCallback(int index);  
public static int SelectionList(int selected, GUIContent[] list) {  
return SelectionList(selected, list, "List Item", null);  
}  
public static int SelectionList(int selected, GUIContent[] list, GUIStyle elementStyle) {  
return SelectionList(selected, list, elementStyle, null);  
}  
public static int SelectionList(int selected, GUIContent[] list, DoubleClickCallback callback) {  
return SelectionList(selected, list, "List Item", callback);  
}  
public static int SelectionList(int selected, GUIContent[] list, GUIStyle elementStyle, DoubleClickCallback callback) {  
for (int i = 0; i < list.Length; ++i) {  
Rect elementRect = GUILayoutUtility.GetRect(list[i], elementStyle);  
bool hover = elementRect.Contains(Event.current.mousePosition);  
if (hover && Event.current.type == EventType.MouseDown&& Event.current.clickCount == 1) {  
selected = i;  
Event.current.Use();  
} else if (hover && callback != null && Event.current.type == EventType.MouseDown && Event.current.clickCount == 2) {  
callback(i);  
Event.current.Use();  
} else if (Event.current.type == EventType.repaint) {  
elementStyle.Draw(elementRect, list[i], hover, false, i == selected, false);  
}  
}  
return selected;  
}  
public static int SelectionList(int selected, string[] list) {  
return SelectionList(selected, list, "List Item", null);  
}  
public static int SelectionList(int selected, string[] list, GUIStyle elementStyle) {  
return SelectionList(selected, list, elementStyle, null);  
}  
public static int SelectionList(int selected, string[] list, DoubleClickCallback callback) {  
return SelectionList(selected, list, "List Item", callback);  
}  
public static int SelectionList(int selected, string[] list, GUIStyle elementStyle, DoubleClickCallback callback) {  
for (int i = 0; i < list.Length; ++i) {  
Rect elementRect = GUILayoutUtility.GetRect(new GUIContent(list[i]), elementStyle);  
bool hover = elementRect.Contains(Event.current.mousePosition);  
if (hover && Event.current.type == EventType.MouseDown&& Event.current.clickCount == 1) {  
selected = i;  
Event.current.Use();  
} else if (hover && callback != null && Event.current.type == EventType.MouseDown && Event.current.clickCount == 2) {  
callback(i);  
Event.current.Use();  
} else if (Event.current.type == EventType.repaint) {  
elementStyle.Draw(elementRect, list[i], hover, false, i == selected, false);  
}  
}  
return selected;  
}  
}

FileBrowser.cs

using UnityEngine;  
using System;  
using System.IO;  
using System.Collections.Generic;  
public enum FileBrowserType {  
File,  
Directory  
}  
public class FileBrowser {  
// Called when the user clicks cancel or select  
public delegate void FinishedCallback(string path);  
// Defaults to working directory  
public string CurrentDirectory {  
get {  
return m\_currentDirectory;  
}  
set {  
SetNewDirectory(value);  
SwitchDirectoryNow();  
}  
}  
protected string m\_currentDirectory;  
// Optional pattern for filtering selectable files/folders. See:  
// http://msdn.microsoft.com/en-us/library/wz42302f(v=VS.90).aspx  
// and  
// http://msdn.microsoft.com/en-us/library/6ff71z1w(v=VS.90).aspx  
public string SelectionPattern {  
get {  
return m\_filePattern;  
}  
set {  
m\_filePattern = value;  
ReadDirectoryContents();  
}  
}  
protected string m\_filePattern;  
// Optional image for directories  
public Texture2D DirectoryImage {  
get {  
return m\_directoryImage;  
}  
set {  
m\_directoryImage = value;  
BuildContent();  
}  
}  
protected Texture2D m\_directoryImage;  
// Optional image for files  
public Texture2D FileImage {  
get {  
return m\_fileImage;  
}  
set {  
m\_fileImage = value;  
BuildContent();  
}  
}  
protected Texture2D m\_fileImage;  
// Browser type. Defaults to File, but can be set to Folder  
public FileBrowserType BrowserType {  
get {  
return m\_browserType;  
}  
set {  
m\_browserType = value;  
ReadDirectoryContents();  
}  
}  
protected FileBrowserType m\_browserType;  
protected string m\_newDirectory;  
protected string[] m\_currentDirectoryParts;  
protected string[] m\_files;  
protected GUIContent[] m\_filesWithImages;  
protected int m\_selectedFile;  
protected string[] m\_nonMatchingFiles;  
protected GUIContent[] m\_nonMatchingFilesWithImages;  
protected int m\_selectedNonMatchingDirectory;  
protected string[] m\_directories;  
protected GUIContent[] m\_directoriesWithImages;  
protected int m\_selectedDirectory;  
protected string[] m\_nonMatchingDirectories;  
protected GUIContent[] m\_nonMatchingDirectoriesWithImages;  
protected bool m\_currentDirectoryMatches;  
protected GUIStyle CentredText {  
get {  
if (m\_centredText == null) {  
m\_centredText = new GUIStyle(GUI.skin.label);  
m\_centredText.alignment = TextAnchor.MiddleLeft;  
m\_centredText.fixedHeight = GUI.skin.button.fixedHeight;  
}  
return m\_centredText;  
}  
}  
protected GUIStyle m\_centredText;  
protected string m\_name;  
protected Rect m\_screenRect;  
protected Vector2 m\_scrollPosition;  
protected FinishedCallback m\_callback;  
// Browsers need at least a rect, name and callback  
public FileBrowser(Rect screenRect, string name, FinishedCallback callback) {  
m\_name = name;  
m\_screenRect = screenRect;  
m\_browserType = FileBrowserType.File;  
m\_callback = callback;  
SetNewDirectory(Directory.GetCurrentDirectory());  
SwitchDirectoryNow();  
}  
protected void SetNewDirectory(string directory) {  
m\_newDirectory = directory;  
}  
protected void SwitchDirectoryNow() {  
if (m\_newDirectory == null || m\_currentDirectory == m\_newDirectory) {  
return;  
}  
m\_currentDirectory = m\_newDirectory;  
m\_scrollPosition = Vector2.zero;  
m\_selectedDirectory = m\_selectedNonMatchingDirectory = m\_selectedFile = -1;  
ReadDirectoryContents();  
}  
protected void ReadDirectoryContents() {  
if (m\_currentDirectory == "/") {  
m\_currentDirectoryParts = new string[] {""};  
m\_currentDirectoryMatches = false;  
} else {  
m\_currentDirectoryParts = m\_currentDirectory.Split(Path.DirectorySeparatorChar);  
if (SelectionPattern != null) {  
// string[] generation = Directory.GetDirectories(  
// Path.GetDirectoryName(m\_currentDirectory),  
// SelectionPattern  
// );  
// m\_currentDirectoryMatches = Array.IndexOf(generation, m\_currentDirectory) >= 0;  
string directoryName = Path.GetDirectoryName(m\_currentDirectory);  
string[] generation = new string[0];  
if(directoryName != null)  
{ //This is new: generation should be an empty array for the root directory.  
//directoryName will be null if it's a root directory  
generation = Directory.GetDirectories(  
directoryName,  
SelectionPattern );  
}  
m\_currentDirectoryMatches = Array.IndexOf(generation, m\_currentDirectory) >= 0;  
} else {  
m\_currentDirectoryMatches = false;  
}  
}  
if (BrowserType == FileBrowserType.File || SelectionPattern == null) {  
m\_directories = Directory.GetDirectories(m\_currentDirectory);  
m\_nonMatchingDirectories = new string[0];  
} else {  
m\_directories = Directory.GetDirectories(m\_currentDirectory, SelectionPattern);  
var nonMatchingDirectories = new List<string>();  
foreach (string directoryPath in Directory.GetDirectories(m\_currentDirectory)) {  
if (Array.IndexOf(m\_directories, directoryPath) < 0) {  
nonMatchingDirectories.Add(directoryPath);  
}  
}  
m\_nonMatchingDirectories = nonMatchingDirectories.ToArray();  
for (int i = 0; i < m\_nonMatchingDirectories.Length; ++i) {  
int lastSeparator = m\_nonMatchingDirectories[i].LastIndexOf(Path.DirectorySeparatorChar);  
m\_nonMatchingDirectories[i] = m\_nonMatchingDirectories[i].Substring(lastSeparator + 1);  
}  
Array.Sort(m\_nonMatchingDirectories);  
}  
for (int i = 0; i < m\_directories.Length; ++i) {  
m\_directories[i] = m\_directories[i].Substring(m\_directories[i].LastIndexOf(Path.DirectorySeparatorChar) + 1);  
}  
if (BrowserType == FileBrowserType.Directory || SelectionPattern == null) {  
m\_files = Directory.GetFiles(m\_currentDirectory);  
m\_nonMatchingFiles = new string[0];  
} else {  
m\_files = Directory.GetFiles(m\_currentDirectory, SelectionPattern);  
var nonMatchingFiles = new List<string>();  
foreach (string filePath in Directory.GetFiles(m\_currentDirectory)) {  
if (Array.IndexOf(m\_files, filePath) < 0) {  
nonMatchingFiles.Add(filePath);  
}  
}  
m\_nonMatchingFiles = nonMatchingFiles.ToArray();  
for (int i = 0; i < m\_nonMatchingFiles.Length; ++i) {  
m\_nonMatchingFiles[i] = Path.GetFileName(m\_nonMatchingFiles[i]);  
}  
Array.Sort(m\_nonMatchingFiles);  
}  
for (int i = 0; i < m\_files.Length; ++i) {  
m\_files[i] = Path.GetFileName(m\_files[i]);  
}  
Array.Sort(m\_files);  
BuildContent();  
m\_newDirectory = null;  
}  
protected void BuildContent() {  
m\_directoriesWithImages = new GUIContent[m\_directories.Length];  
for (int i = 0; i < m\_directoriesWithImages.Length; ++i) {  
m\_directoriesWithImages[i] = new GUIContent(m\_directories[i], DirectoryImage);  
}  
m\_nonMatchingDirectoriesWithImages = new GUIContent[m\_nonMatchingDirectories.Length];  
for (int i = 0; i < m\_nonMatchingDirectoriesWithImages.Length; ++i) {  
m\_nonMatchingDirectoriesWithImages[i] = new GUIContent(m\_nonMatchingDirectories[i], DirectoryImage);  
}  
m\_filesWithImages = new GUIContent[m\_files.Length];  
for (int i = 0; i < m\_filesWithImages.Length; ++i) {  
m\_filesWithImages[i] = new GUIContent(m\_files[i], FileImage);  
}  
m\_nonMatchingFilesWithImages = new GUIContent[m\_nonMatchingFiles.Length];  
for (int i = 0; i < m\_nonMatchingFilesWithImages.Length; ++i) {  
m\_nonMatchingFilesWithImages[i] = new GUIContent(m\_nonMatchingFiles[i], FileImage);  
}  
}  
public void OnGUI() {  
GUILayout.BeginArea(  
m\_screenRect,  
m\_name,  
GUI.skin.window  
);  
GUILayout.BeginHorizontal();  
for (int parentIndex = 0; parentIndex < m\_currentDirectoryParts.Length; ++parentIndex) {  
if (parentIndex == m\_currentDirectoryParts.Length - 1) {  
GUILayout.Label(m\_currentDirectoryParts[parentIndex], CentredText);  
} else if (GUILayout.Button(m\_currentDirectoryParts[parentIndex])) {  
string parentDirectoryName = m\_currentDirectory;  
for (int i = m\_currentDirectoryParts.Length - 1; i > parentIndex; --i) {  
parentDirectoryName = Path.GetDirectoryName(parentDirectoryName);  
}  
SetNewDirectory(parentDirectoryName);  
}  
}  
GUILayout.FlexibleSpace();  
GUILayout.EndHorizontal();  
m\_scrollPosition = GUILayout.BeginScrollView(  
m\_scrollPosition,  
false,  
true,  
GUI.skin.horizontalScrollbar,  
GUI.skin.verticalScrollbar,  
GUI.skin.box  
);  
m\_selectedDirectory = GUILayoutx.SelectionList(  
m\_selectedDirectory,  
m\_directoriesWithImages,  
DirectoryDoubleClickCallback  
);  
if (m\_selectedDirectory > -1) {  
m\_selectedFile = m\_selectedNonMatchingDirectory = -1;  
}  
m\_selectedNonMatchingDirectory = GUILayoutx.SelectionList(  
m\_selectedNonMatchingDirectory,  
m\_nonMatchingDirectoriesWithImages,  
NonMatchingDirectoryDoubleClickCallback  
);  
if (m\_selectedNonMatchingDirectory > -1) {  
m\_selectedDirectory = m\_selectedFile = -1;  
}  
GUI.enabled = BrowserType == FileBrowserType.File;  
m\_selectedFile = GUILayoutx.SelectionList(  
m\_selectedFile,  
m\_filesWithImages,  
FileDoubleClickCallback  
);  
GUI.enabled = true;  
if (m\_selectedFile > -1) {  
m\_selectedDirectory = m\_selectedNonMatchingDirectory = -1;  
}  
GUI.enabled = false;  
GUILayoutx.SelectionList(  
-1,  
m\_nonMatchingFilesWithImages  
);  
GUI.enabled = true;  
GUILayout.EndScrollView();  
GUILayout.BeginHorizontal();  
GUILayout.FlexibleSpace();  
if (GUILayout.Button("Cancel", GUILayout.Width(50))) {  
m\_callback(null);  
}  
if (BrowserType == FileBrowserType.File) {  
GUI.enabled = m\_selectedFile > -1;  
} else {  
if (SelectionPattern == null) {  
GUI.enabled = m\_selectedDirectory > -1;  
} else {  
GUI.enabled = m\_selectedDirectory > -1 ||  
(  
m\_currentDirectoryMatches &&  
m\_selectedNonMatchingDirectory == -1 &&  
m\_selectedFile == -1  
);  
}  
}  
if (GUILayout.Button("Select", GUILayout.Width(50))) {  
TextFileFinder.flag=true;  
if (BrowserType == FileBrowserType.File) {  
m\_callback(Path.Combine(m\_currentDirectory, m\_files[m\_selectedFile]));  
} else {  
if (m\_selectedDirectory > -1) {  
m\_callback(Path.Combine(m\_currentDirectory, m\_directories[m\_selectedDirectory]));  
} else {  
m\_callback(m\_currentDirectory);  
}  
}  
}  
GUI.enabled = true;  
GUILayout.EndHorizontal();  
GUILayout.EndArea();  
if (Event.current.type == EventType.Repaint) {  
SwitchDirectoryNow();  
}  
}  
protected void FileDoubleClickCallback(int i) {  
if (BrowserType == FileBrowserType.File) {  
m\_callback(Path.Combine(m\_currentDirectory, m\_files[i]));  
}  
}  
protected void DirectoryDoubleClickCallback(int i) {  
SetNewDirectory(Path.Combine(m\_currentDirectory, m\_directories[i]));  
}  
protected void NonMatchingDirectoryDoubleClickCallback(int i) {  
SetNewDirectory(Path.Combine(m\_currentDirectory, m\_nonMatchingDirectories[i]));  
}  
}