using System;

using System.Collections.Generic;

using System.IO;

using System.Xml;

using System.Xml.Serialization;

using UnityEngine;

using UnityEngine.SceneManagement;

/// <summary>

/// Class for handling persistent game data

/// </summary>

public class GameControll : MonoBehaviour

{

#region Public Fields

public static GameControll Instance;

public GameData gameData = new GameData();

public Scene openScene;

private string savePath;

public string SavePath

{

get

{

if (savePath != null)

return savePath;

else

{

savePath = Application.persistentDataPath + "/SavedGames/";

return savePath;

}

}

}

#endregion Public Fields

#region Private Fields

private const string FILE\_EXTENSION = ".xml";

// Save Load Data

private string saveFile;

#endregion Private Fields

#region Public Methods

/// <summary>

/// Deletes the save file if it exists and errors out otherwise.

/// </summary>

/// <param name="saveFile"></param>

public void DeleteSaveFile(string saveFile)

{

if (File.Exists(SavePath + saveFile + FILE\_EXTENSION))

{

File.Delete(SavePath + saveFile + FILE\_EXTENSION);

}

else

{

Debug.LogError("Failed to delete non existant file " + SavePath + saveFile + FILE\_EXTENSION);

}

}

/// <summary>

/// Checks if the save file exists in the file system

/// </summary>

/// <param name="testFileName"></param>

/// <returns>True if it exists and false otherwise</returns>

public bool DoesFileExist(string testFileName)

{

foreach (GameData data in GetAllSaveFiles())

{

if (data.lastSaveFile == testFileName)

{

return true;

}

}

return false;

}

/// <summary>

/// Create a new file name, check for existing files of same player name

/// </summary>

/// <returns></returns>

public string GenerateNewSaveName()

{

int attempt = 0;

string newSaveName = "";

while (newSaveName == "")

{

// Save Name is Player Name

string checkString = gameData.playerName;

// Add a number if original already taken

if (attempt != 0) checkString += attempt;

if (!File.Exists(SavePath + checkString))

{

// Make the check string the new file name

newSaveName = checkString;

}

attempt++;

}

return newSaveName;

}

/// <summary>

/// Gets a list of all save files in the save directory.

/// </summary>

/// <returns></returns>

public List<GameData> GetAllSaveFiles()

{

List<GameData> allSaves = new List<GameData>();

// Check Save Path

foreach (string fileName in Directory.GetFiles(SavePath))

{

// Get Player Data for Each File

allSaves.Add(GetSaveFile(fileName));

}

return allSaves;

}

/// <summary>

/// Finds the value associated with the flag

/// </summary>

/// <param name="flagName"></param>

/// <returns></returns>

public int GetFlag(string flagName)

{

GameFlag flag = gameData.gameFlags.Find(x => x.flag == flagName);

// Create Non-existant flags but default to 0

if (flag == null)

{

SetFlag(flagName, 0);

return 0;

}

return flag.value;

}

/// <summary>

/// Checks if a particular level has been cleared yet or not

/// </summary>

/// <param name="level">Level to check</param>

/// <returns>True if cleared and false otherwise</returns>

public bool GetLevelCleared(int level)

{

return GetFlag("level" + level + "cleared") == 1 ? true : false;

}

/// <summary>

/// Load game data from file for active use

/// </summary>

/// <param name="gameName"></param>

/// <returns></returns>

public void LoadGame(string gameName)

{

CheckDirectory();

// Assemble path to file to load game from

String fullFilePath = SavePath + gameName + FILE\_EXTENSION;

if (File.Exists(fullFilePath))

{

// Put it into a file

Debug.Log("Deserializing " + fullFilePath);

FileStream fs = File.Open(fullFilePath, FileMode.Open);

// Deserialize the XML Save File (Using XmlSerializer instead of BinarySerializer)

XmlSerializer xmlSerializer = new XmlSerializer(typeof(GameData));

XmlReader reader = XmlReader.Create(fs);

gameData = xmlSerializer.Deserialize(reader) as GameData;

fs.Close();

// Loads the scene from which the game was saved

SceneManager.LoadSceneAsync(gameData.savedScene, LoadSceneMode.Single);

}

else

{

Debug.Log("Failed to save to file " + fullFilePath);

}

}

/// <summary>

/// Save all game data to file

/// </summary>

/// <param name="saveFile"></param>

public void SaveGame(string saveFile)

{

CheckDirectory();

// Update saveFile name

if (saveFile == null)

{

saveFile = GenerateNewSaveName();

}

this.saveFile = saveFile;

// FileStream fs = File.Create(GameDic.Instance.SavePath + saveFile);

UpdateSaveData(saveFile);

string fullSavePath = SavePath + saveFile + FILE\_EXTENSION;

FileStream fs;

// Create a file or open an old one up for writing to

if (!File.Exists(fullSavePath))

{

fs = File.Create(fullSavePath);

}

else

{

fs = File.OpenWrite(fullSavePath);

}

XmlSerializer serializer = new XmlSerializer(typeof(GameData));

TextWriter textWriter = new StreamWriter(fs);

serializer.Serialize(textWriter, gameData);

fs.Close();

Debug.Log("Game Saved to " + fullSavePath);

}

/// <summary>

/// Set Current Save Related Information on gameData

/// </summary>

/// <param name="saveFile"></param>

private void UpdateSaveData(string saveFile)

{

gameData.lastSaveFile = saveFile;

gameData.lastSaveTime = DateTime.Now.ToBinary();

gameData.savedScene = SceneManager.GetActiveScene().name;

}

// For flag storing and getting

public void SetFlag(string flagName, int value)

{

// Overwrite Old Key/Values

GameFlag oldFlag = gameData.gameFlags.Find(x => x.flag == flagName);

// Either update the value or add a new one if it does not exist

if (oldFlag != null)

{

oldFlag.value = value;

}

else

{

// Does not exist in list

gameData.gameFlags.Add(new GameFlag(flagName, value));

}

}

#endregion Public Methods

#region Private Methods

/// <summary>

/// Checks if the file has not yet been created

/// </summary>

/// <param name="saveFile"></param>

/// <returns></returns>

private bool IsNewFile(string saveFile)

{

return !File.Exists(SavePath + saveFile + FILE\_EXTENSION);

}

/// <summary>

/// Initialization

/// </summary>

private void Awake()

{

//Check if instance already exists

if (Instance == null)

{

//if not, set instance to this

Instance = this;

// Find objects on level - necessary to call directly for first load

SceneManager.sceneLoaded += OnSceneLoaded;

openScene = SceneManager.GetActiveScene();

}

//If instance already exists and it's not this:

else if (Instance != this)

//Then destroy this. This enforces our singleton pattern, meaning there can only ever be one instance of a GameManager.

Destroy(gameObject);

// Sets this to not be destroyed when reloading scene

DontDestroyOnLoad(gameObject);

}

/// <summary>

/// Checks to see if the SavePath directory exists and creates a new one of it does not.

/// </summary>

private void CheckDirectory()

{

// Check if directory exists, if not create it

if (!Directory.Exists(SavePath))

{

Directory.CreateDirectory(SavePath);

}

}

/// <summary>

/// Retrieves the data stored inside of a save file

/// </summary>

/// <param name="fullFilePath"></param>

/// <returns></returns>

private GameData GetSaveFile(string fullFilePath)

{

if (File.Exists(fullFilePath))

{

// Old Binary Formmater Method BinaryFormatter bf = new BinaryFormatter(); FileStream

// fs = File.Open(fullFilePath, FileMode.Open);

// Put it into a file PlayerData data = (PlayerData)bf.Deserialize(fs);

// fs.Close();

// XML SERIALIZER TEST INSTEAD OF BINARYFORMATTER

FileStream fs = File.Open(fullFilePath, FileMode.Open);

XmlSerializer xmlSerializer = new XmlSerializer(typeof(GameData));

XmlReader reader = XmlReader.Create(fs);

GameData data = xmlSerializer.Deserialize(reader) as GameData;

fs.Close();

return data;

}

else

{

Debug.LogError("Failed to save to file " + fullFilePath);

return null;

}

}

/// <summary>

/// Make sure that the save / load directory exists.

/// </summary>

/// <param name="scene"></param>

/// <param name="mode"></param>

private void OnSceneLoaded(Scene scene, LoadSceneMode mode)

{

CheckDirectory();

}

#endregion Private Methods

}

[Serializable]

public class GameFlag

{

#region Public Fields

public string flag;

public int value;

#endregion Public Fields

#region Public Constructors

public GameFlag()

{

}

public GameFlag(string flag, int value)

{

this.flag = flag;

this.value = value;

}

#endregion Public Constructors

}

[Serializable]

public class GameData

{

#region Public Fields

public int currentChapter;

public List<GameFlag> gameFlags;

public float health;

public string playerName;

// Needs properties to access

[NonSerialized]

public Vector3 playerPosition;

public string lastSaveFile;

public long lastSaveTime;

public string savedScene;

public int upgradePoints = 0;

public int upgradePointsSpent = 0;

#endregion Public Fields

#region Public Constructors

/// <summary>

/// Default Constructor for New Game - Contains Starting Stats

/// </summary>

public GameData()

{

playerPosition = Vector3.zero;

health = 100;

playerName = "Jill";

upgradePoints = 0;

upgradePointsSpent = 0;

currentChapter = 1;

savedScene = "";

gameFlags = new List<GameFlag>();

}

#endregion Public Constructors

#region Public Properties

// Can't serialize a vector so needs to be broken down into 3 properties

public float PlayerPositionX

{

get

{

return playerPosition.x;

}

set

{

playerPosition.x = value;

}

}

public float PlayerPositionY

{

get

{

return playerPosition.y;

}

set

{

playerPosition.y = value;

}

}

public float PlayerPositionZ

{

get

{

return playerPosition.z;

}

set

{

playerPosition.z = value;

}

}

#endregion Public Properties

}