Saving and loading in Binary Data in Unity

1. Inside your player script create a new C# script called “Stats”
2. Remove all the scripts so you are left with public class Stats, remove MonoBehavior as well.
3. Attach this script

* Public class Stats

{

Public int Life;

Public int Attack;

Public int Vitality;

Public int Defense;

}

1. Hop back into your player class with MonoBehaviour and create a variable that links the stats. Do this inside the class.

* Ex:

Public class Player : MonoBehaviour

{

Public Stats myStats;

Private void Update()

{ ….

}

}

1. Now the Inspector will see the class, but it cannot read the data, to do this you must add this line of code below to read the data in your “blueprint” class (Stats class).

* Ex:

using UnityEngine;

[System.Serializable]

1. Now hop back to the player class, you will now be able to see the individual int variables that link to life, attack, vitality, and defense.
2. The next part is to map the stats to the UI, to do this create the UI class which defines methods that are used for the game, here is how we map the stats.

public class UIManager : MonoBehaviour

{

Singleton

public TextMeshProUGUI LifeStat;

public TextMeshProUGUI VitalityStat;

public TextMeshProUGUI DefenseStat;

public TextMeshProUGUI AttackStat;

public void ReadStats()

{

…..

MapValuesToPlayer();

}

void MapValuesToPlayer()

{

Player player = GameObject.FindObjectofType<Player>();

player.myStats.Life = int.Parse(LifeStat.text);

player.myStats.Vitality = int.Parse(VitalityStat.text);

player.myStats.Attack = int.Parse(AttackStat.text);

player.myStats.Defense = int.Parse(DefenseStat.text);

}

}

….

1. Now that we have the values mapped to the UI, we can take the stats and save them, to do this we will require a filestream, a binary formatter for data, and a serialization method to write to the file. You will also require to write the code that includes these tools at the top of the C# script. Also, you must create a link to the player for the script.

* Ex:

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

public class SaveManager : MonoBehaviour

{

private player \_player;

private void Awake()

{

\_player = GameObject.FindObjectOfType<Player>();

}

public void Save()

{

//Create a file or open a file to save to

FileStream file = new FileStream(Application.persistantDataPath + “/Player.dat”, FileMode.OpenorCreate);

try{

// Binary Formatter – allows us to write data to file

BinaryFormatter formatter = new BinaryFormatter();

// serialization method to WRITE data to file

Formatter.Serialize(file, \_player.myStats);

}

Catch(SerializationException e)

{

Debug.LogError(“There was an issue serializing this data + e.Message);

}

Finally {

file.Close();

}

1. Now we want to save the position in where the player is at, to do this we will require a struct that updates the values, instead of arrays, this can save colours, position, etc. go back to stats and write this above the method.

* Ex:

public struct SerializableVector3

{

public float x;

public float y;

public Vector3 GetPos()

{

return new Vector3(x,y);

}

}

Now return to The Player Script, and in the update method, write this code for position

transform.position = myStats.myPos.GetPos;

don’t forget to add this code in your Stats Class

public SerializableVector3 myPos;

1. Now go to the Hierarchy (the tab above the scripts), and create a empty, and then name it Save Manager, and attach the Save Manager Script (click and drag). Now, click on your Canvas UI (display for menu screen), and click on the button you created that has in the Text field “Save”. Drag the “Save Manager” hierarchy option to the “OnClick” method to the right. Hop into the Save Manager and call the method “Save()”.
2. Now, to detect the MyPos coordinates, we must add

* [System.Serializable]

Above the struct. Now we can detect it as seriablizable data.

1. Now, we do the same thing for loading, in the savemanager script write this code.

* Ex:

public Void Load()

{

FileStream file = new FileStream(Application.persistantDataPath + “/Player.dat , FileMode.Open);

try

{

BinaryFormatter formatter = new BinaryFormatter();

\_player.myStats = (Stats)formatter.Deserialize(file);

}

catch(SerializationException e)

{

Debug.LogError(“Error Deserializing Data” + e.Message);

}

finally

{

file.Close();

}

You can now choose to add multiple button screens that have the saved data so you can load it later. This method is great for cloaking data.