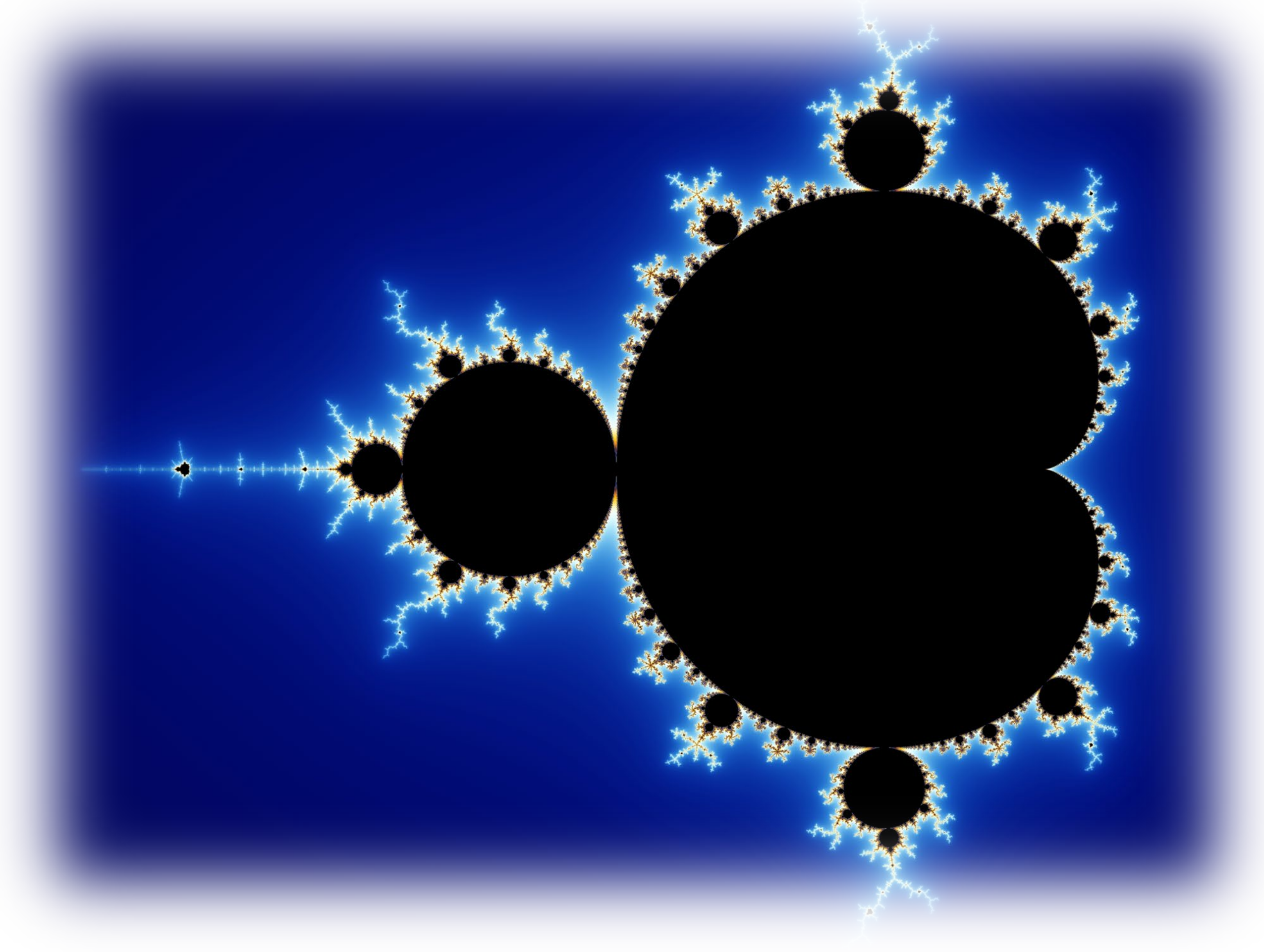


SAVE & LOAD

Ci410

TYPES OF SAVES

Between levels

- Use GameManager (singleton), will stores in RAM

Between sessions

- Use disk, convert RAM data into something which can be stored, and more importantly loaded again from disk (or Web)

DESIGNING FOR SERIALISATION

Serialisation = Saving

De-serialisation = Loading

Purpose
Record game state at a moment in time
- Before difficult mission
- Before Machine is turned off
- Autosave, in case of crash/death

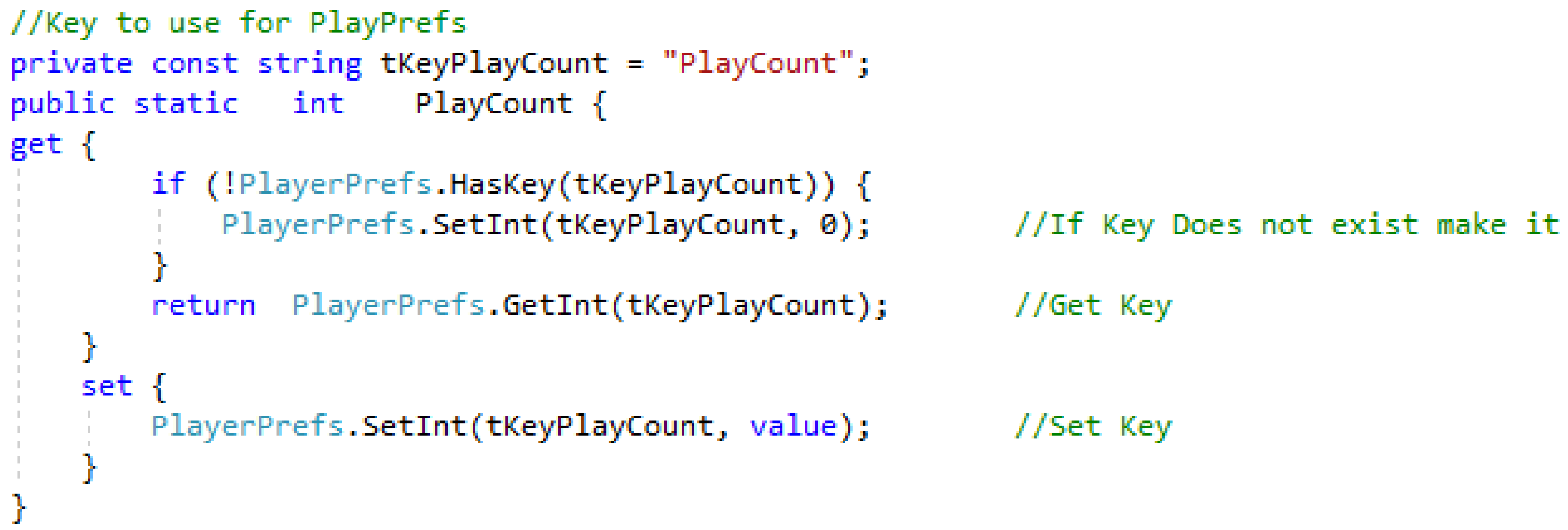
Also called “Persisting”

SIMPLE SAVE METHODS

PlayerPrefs, very easy to use

Stores simple types: int, float, string

Uses Key:Value (dictionary like) system

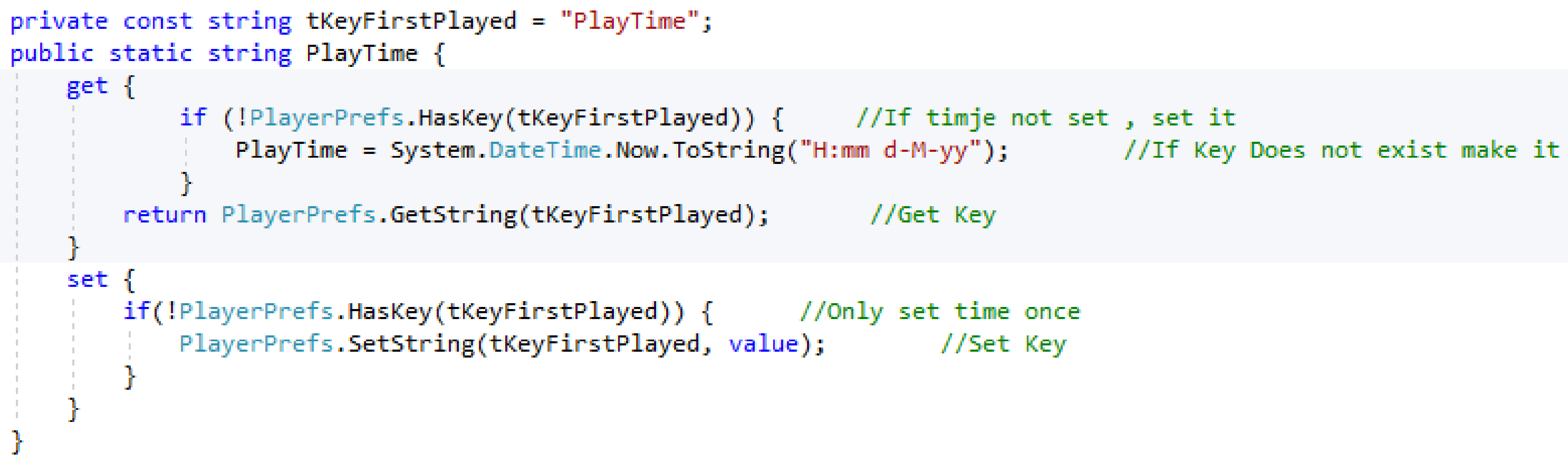


```
//Key to use for PlayPrefs
private const string tKeyPlayCount = "PlayCount";
public static   int    PlayCount {
get {
        if (!PlayerPrefs.HasKey(tKeyPlayCount)) {
            PlayerPrefs.SetInt(tKeyPlayCount, 0);        //If Key Does not exist make it
        }
        return  PlayerPrefs.GetInt(tKeyPlayCount);       //Get Key
    }
    set {
        PlayerPrefs.SetInt(tKeyPlayCount, value);        //Set Key
    }
}
```

FIRST PLAY TIME

Example2

- Storing & retrieving the first time the player played the game

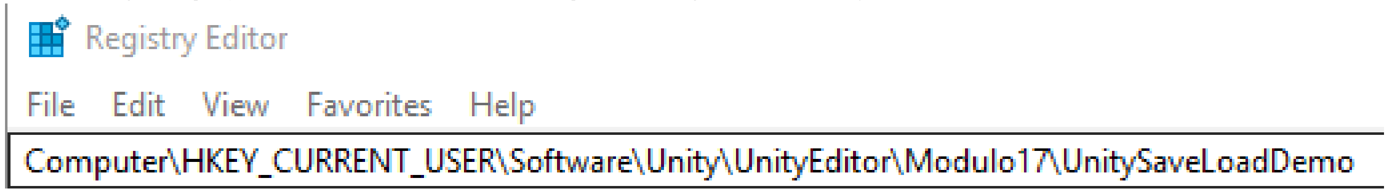


```
private const string tKeyFirstPlayed = "PlayTime";
public static string PlayTime {
    get {
            if (!PlayerPrefs.HasKey(tKeyFirstPlayed)) {     //If timje not set , set it
                PlayTime = System.DateTime.Now.ToString("H:mm d-M-yy");        //If Key Does not exist make it
            }
        return PlayerPrefs.GetString(tKeyFirstPlayed);       //Get Key
    }
    set {
        if(!PlayerPrefs.HasKey(tKeyFirstPlayed)) {      //Only set time once
            PlayerPrefs.SetString(tKeyFirstPlayed, value);        //Set Key
        }
    }
}
```

WHERE DOES IT STORE?

PlayerPrefs

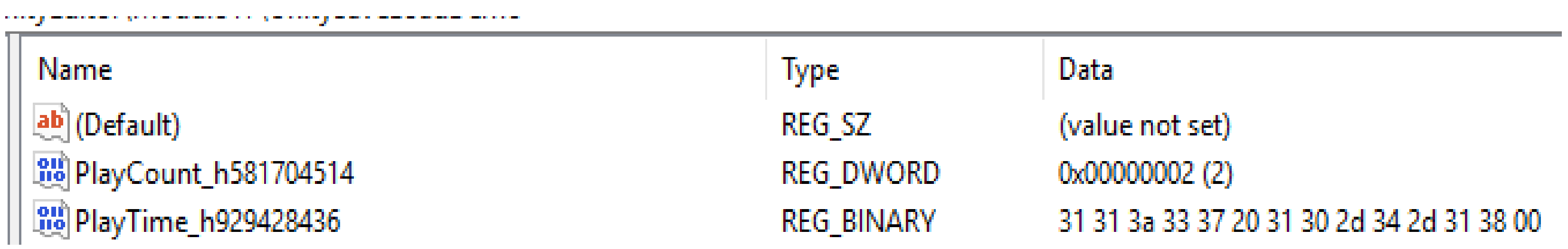
- Stored in various places depending on build
- PC Registry (various locations depending on build)



Computer\HKEY_CURRENT_USER\Software\Unity\UnityEditor\Modulo17\UnitySaveLoadDemo

- Mac Text file

All user editable & not safe



Name	Type	Data
(Default)	REG_SZ	(value not set)
PlayCount_h581704514	REG_DWORD	0x00000002 (2)
PlayTime_h929428436	REG_BINARY	31 31 3a 33 37 20 31 30 2d 34 2d 31 38 00

PLAYERPREFS AMATEUR CHOICE

Player could edit any data and cheat

Does not deal with versions being updated

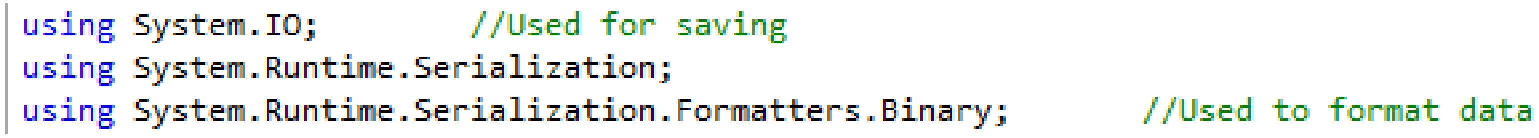
Generally Slow

Easley becomes bloated

However: If you need a quick fix, it’s a good starting point & it beats not saving

A PROFESSIONAL APPROACH

Using C# System calls (not Unity)



```
using System.IO;         //Used for saving
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;       //Used to format data
```

Stores binary formatted data to file system

Can only serialise simple data: int, float, double, string

Needs an open FileStream to work

- May fail and cause exception (crash)

SAVE CODE

Save code without error checking

Get SavePath+Filename (Unity knows where its safe to store stuff)

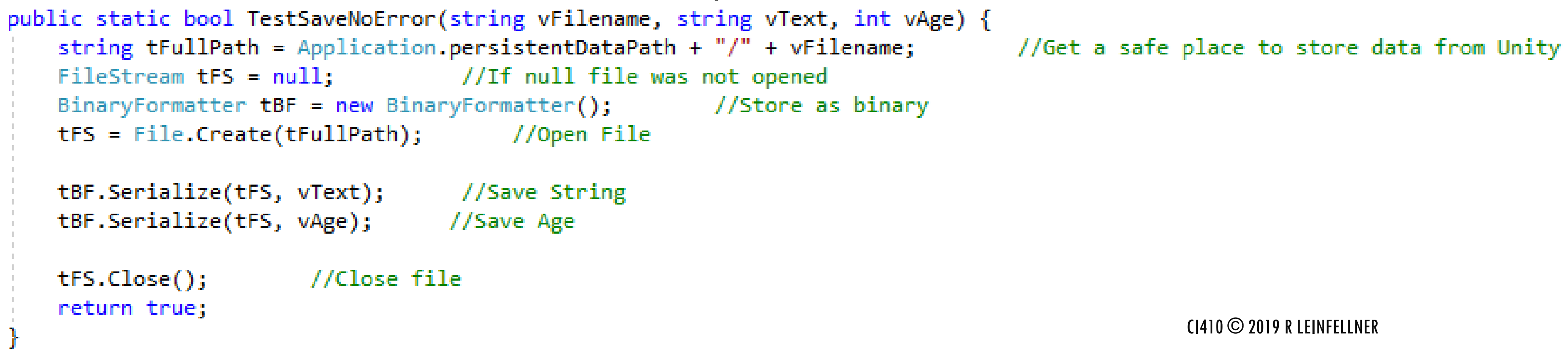
Get BinaryFormatter (this knows how to write basic types as binary)

Open File for writing (will overwrite old file of same name)

Save Name (string)

Save Age (int)

Close file
NB: File writes are cached and data may not be written till closed



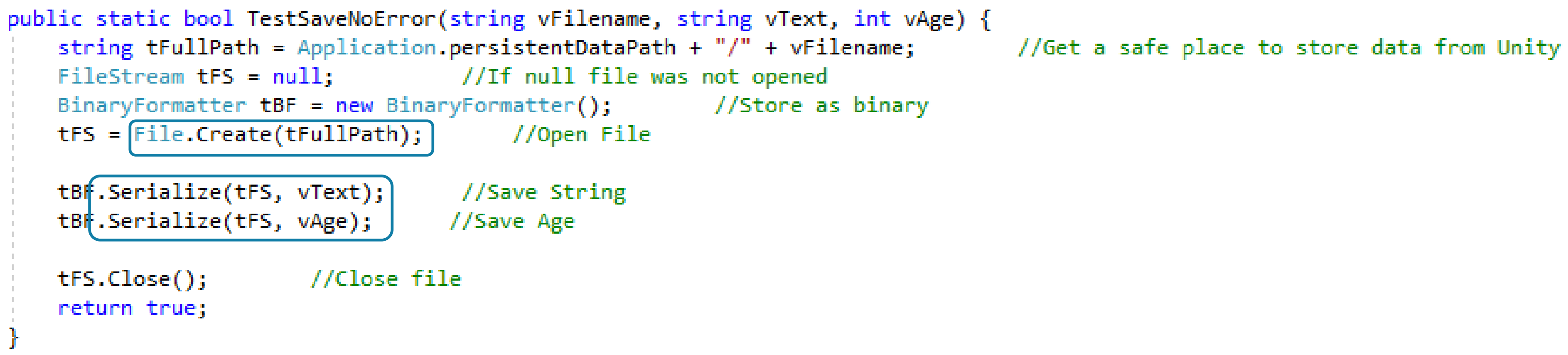
```
public static bool TestSaveNoError(string vFilename, string vText, int vAge) {
    string tFullPath = Application.persistentDataPath + "/" + vFilename;        //Get a safe place to store data from Unity
    FileStream tFS = null;           //If null file was not opened
    BinaryFormatter tBF = new BinaryFormatter();        //Store as binary
    tFS = File.Create(tFullPath);        //Open File

    tBF.Serialize(tFS, vText);       //Save String
    tBF.Serialize(tFS, vAge);       //Save Age

    tFS.Close();        //Close file
    return true;
}
```

DEALING WITH ERRORS

File.Create() may fail: Full disk, Write Error

Serialize may fail: Unknown data type

Uncaught exception == “Game Over Man!”
These are common recoverable errors, however the exception would terminate the code in a stand alone build



```
public static bool TestSaveNoError(string vFilename, string vText, int vAge) {
    string tFullPath = Application.persistentDataPath + "/" + vFilename;        //Get a safe place to store data from Unity
    FileStream tFS = null;          //If null file was not opened
    BinaryFormatter tBF = new BinaryFormatter();        //Store as binary
    tFS = File.Create(tFullPath);        //Open File

    tBF.Serialize(tFS, vText);      //Save String
    tBF.Serialize(tFS, vAge);      //Save Age

    tFS.Close();        //Close file
    return true;
}
```

TRY/CATCH/FINALLY

```
try {

//this code which may crash

} catch (Exception) {

//Get what went wrong BUT don’t crash

} finally {

//Always do this, clear up resources
}
```

“SAFE” SAVE CODE

Catching exceptions with try/catch/finally

An exception is thrown when an error occurs

```
public static  bool    TestSave(string vFilename, string vText, int vAge) {
    bool tSuccess = false;
    string tFullPath = Application.persistentDataPath + "/" + vFilename;        //Get a safe place to store data from Unity
    FileStream tFS = null;            //If null file was not opened
    try {
        BinaryFormatter tBF = new BinaryFormatter();             //Store as binary
        tFS = File.Create(tFullPath);        //Open File
        tBF.Serialize(tFS, vText);      //Save String
        tBF.Serialize(tFS, vAge);       //Save Age
        tSuccess = true;              //Only if we get here have we been successful
    } catch (Exception tE) {          //Deal with error
        Debug.LogErrorFormat("Save Error:", tE.Message);
    } finally {     //Make sure file is closed, if it was open
        if (tFS != null) {
            tFS.Close();        //Close file
        }
    }
    return tSuccess;
}
```

All exceptions in here are caught

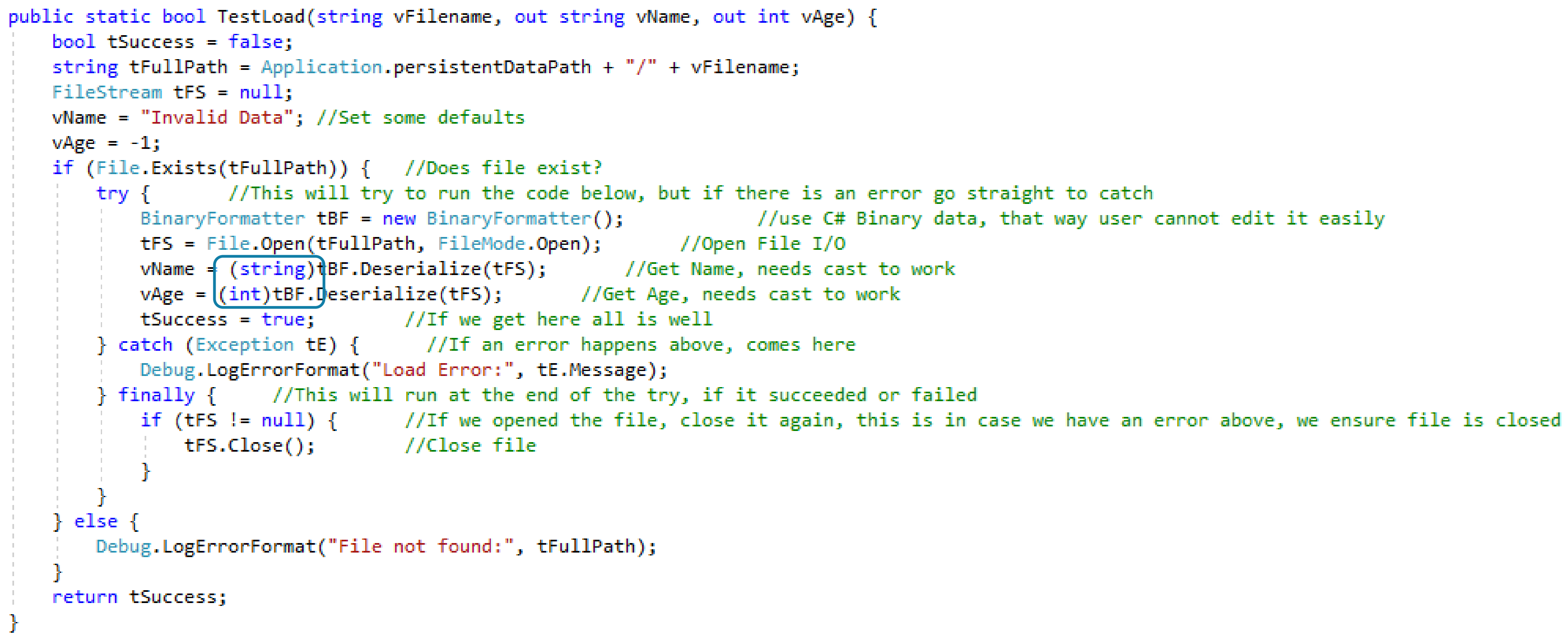
WHY EXCEPTIONS?

```
if() {
   if() {
     if() {
        if() {
          if() {
          } else {Error}
        } else {Error}
     } else {Error}
   } else {Error}
} else {Error}
```

LOADING DATA BACK IN

Roughly the same as Saving

NB: (string) & (int) casts on Deserialize()



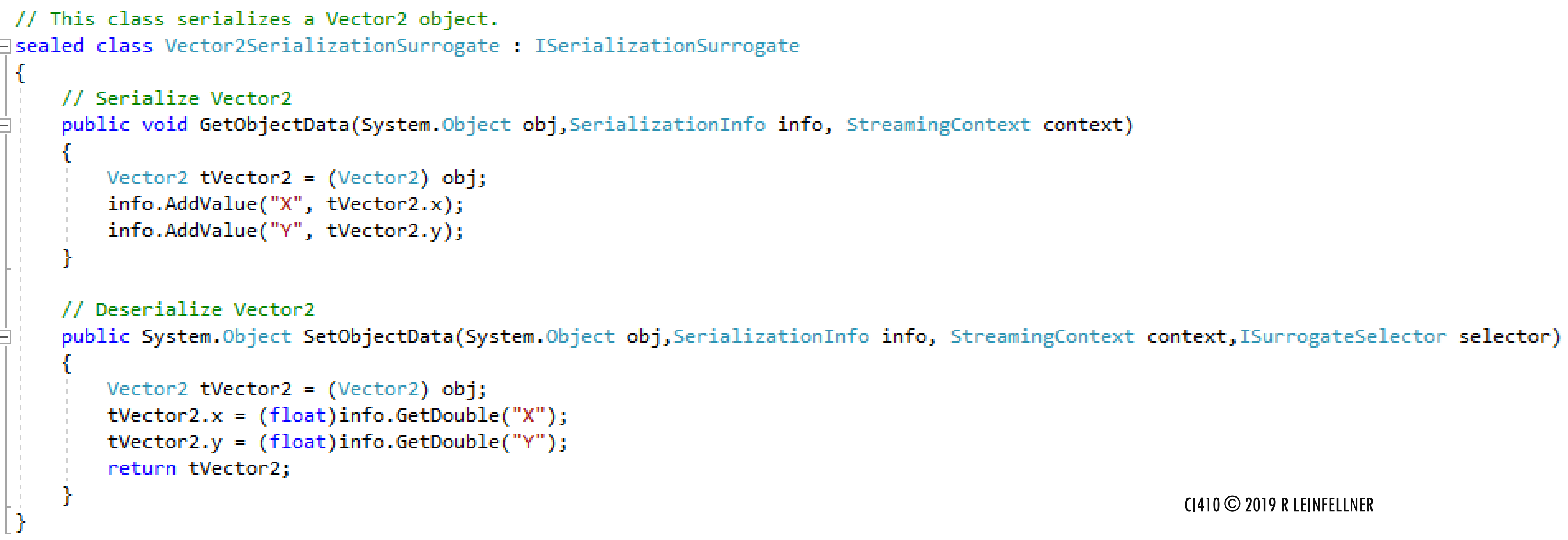
```csharp
public static bool TestLoad(string vFilename, out string vName, out int vAge) {
    bool tSuccess = false;
    string tFullPath = Application.persistentDataPath + "/" + vFilename;
    FileStream tFS = null;
    vName = "Invalid Data"; //Set some defaults
    vAge = -1;
    if (File.Exists(tFullPath)) {   //Does file exist?
        try {       //This will try to run the code below, but if there is an error go straight to catch
            BinaryFormatter tBF = new BinaryFormatter();            //use C# Binary data, that way user cannot edit it easily
            tFS = File.Open(tFullPath, FileMode.Open);       //Open File I/O
            vName = (string)tBF.Deserialize(tFS);       //Get Name, needs cast to work
            vAge = (int)tBF.Deserialize(tFS);       //Get Age, needs cast to work
            tSuccess = true;        //If we get here all is well
        } catch (Exception tE) {      //If an error happens above, comes here
            Debug.LogErrorFormat("Load Error:", tE.Message);
        } finally {     //This will run at the end of the try, if it succeeded or failed
            if (tFS != null) {      //If we opened the file, close it again, this is in case we have an error above, we ensure file is closed
                tFS.Close();        //Close file
            }
        }
    } else {
        Debug.LogErrorFormat("File not found:", tFullPath);
    }
    return tSuccess;
}
```

COMPLEX DATA

Vectors, Color, Quaternion etc.

Need to write your own formatter based on ISerializationSurrogate

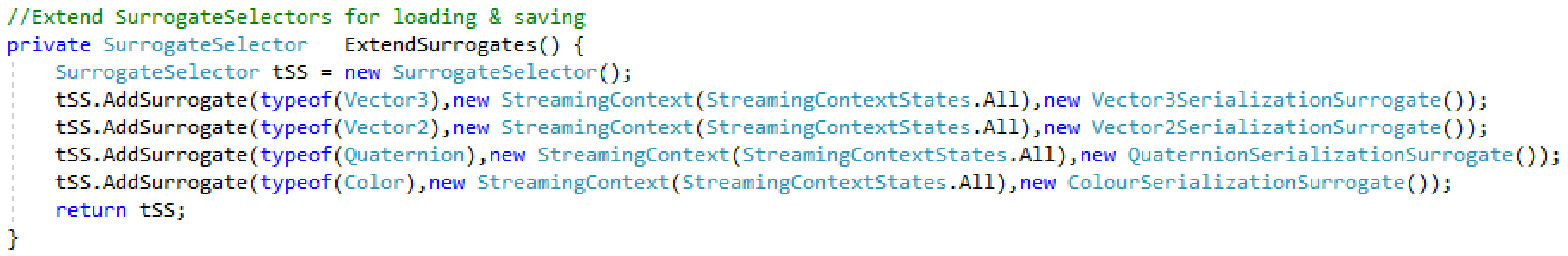
Formatter will use yours when it can't find its own



```
// This class serializes a Vector2 object.
sealed class Vector2SerializationSurrogate : ISerializationSurrogate
{
    // Serialize Vector2
    public void GetObjectData(System.Object obj,SerializationInfo info, StreamingContext context)
    {
        Vector2 tVector2 = (Vector2) obj;
        info.AddValue("X", tVector2.x);
        info.AddValue("Y", tVector2.y);
    }

    // Deserialize Vector2
    public System.Object SetObjectData(System.Object obj,SerializationInfo info, StreamingContext context,ISurrogateSelector selector)
    {
        Vector2 tVector2 = (Vector2) obj;
        tVector2.x = (float)info.GetDouble("X");
        tVector2.y = (float)info.GetDouble("Y");
        return tVector2;
    }
}
```

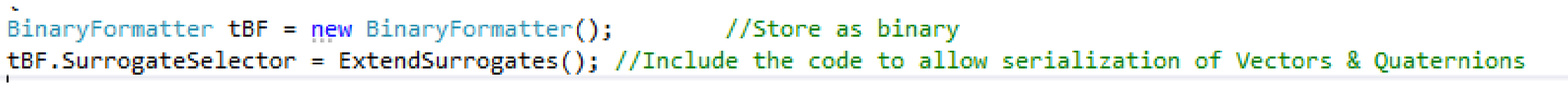
TELLING THE SERIALIZER

Make up a new Selector



```
//Extend SurrogateSelectors for loading & saving
private SurrogateSelector   ExtendSurrogates() {
    SurrogateSelector tSS = new SurrogateSelector();
    tSS.AddSurrogate(typeof(Vector3),new StreamingContext(StreamingContextStates.All),new Vector3SerializationSurrogate());
    tSS.AddSurrogate(typeof(Vector2),new StreamingContext(StreamingContextStates.All),new Vector2SerializationSurrogate());
    tSS.AddSurrogate(typeof(Quaternion),new StreamingContext(StreamingContextStates.All),new QuaternionSerializationSurrogate());
    tSS.AddSurrogate(typeof(Color),new StreamingContext(StreamingContextStates.All),new ColourSerializationSurrogate());
    return tSS;
}
```

Tell BinaryFormatter about it, it will take the data and convert it to something which can be stored



```
BinaryFormatter tBF = new BinaryFormatter();         //Store as binary
tBF.SurrogateSelector = ExtendSurrogates(); //Include the code to allow serialization of Vectors & Quaternions
```

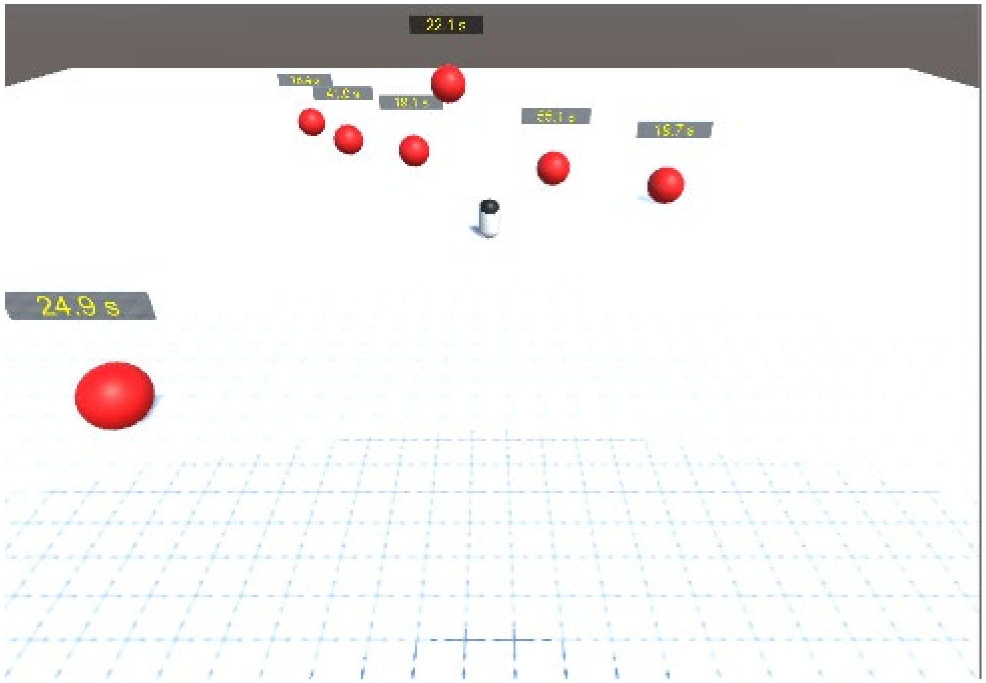
PLANNING A GAME SAVE

Saving is very similar to Networking

Save only what you need to recreate the game state

EG.

A number of GameObjects



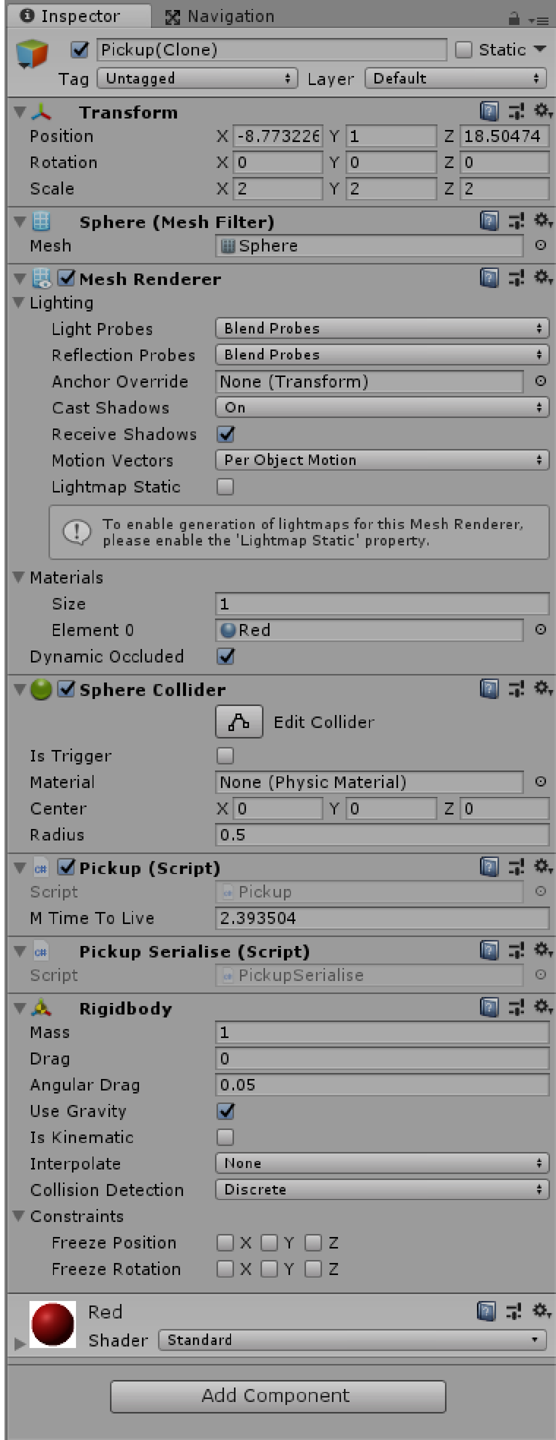
LOTS OF DATA

Positions, Rotations, Scale

Velocity, Mesh, Collider

Etc….

However you only save what you need to recreate the state


Inspector | Navigation

Pickup(Clone) | Static

Tag: Untagged | Layer: Default

Transform

	X	Y	Z
Position	-8.773226	1	18.50474
Rotation	0	0	0
Scale	2	2	2

Sphere (Mesh Filter)

Mesh: Sphere

Mesh Renderer

Lighting
- Light Probes: Blend Probes
- Reflection Probes: Blend Probes
- Anchor Override: None (Transform)
- Cast Shadows: On
- Receive Shadows: ✓
- Motion Vectors: Per Object Motion
- Lightmap Static: ☐

To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.

Materials
- Size: 1
- Element 0: Red

Dynamic Occluded: ✓

Sphere Collider

Edit Collider

- Is Trigger: ☐
- Material: None (Physic Material)
- Center: X 0 Y 0 Z 0
- Radius: 0.5

Pickup (Script)

- Script: Pickup
- M Time To Live: 2.393504

Pickup Serialise (Script)

- Script: PickupSerialise

Rigidbody

- Mass: 1
- Drag: 0
- Angular Drag: 0.05
- Use Gravity: ✓
- Is Kinematic: ☐
- Interpolate: None
- Collision Detection: Discrete

Constraints
- Freeze Position: ☐ X ☐ Y ☐ Z
- Freeze Rotation: ☐ X ☐ Y ☐ Z

Red
Shader: Standard

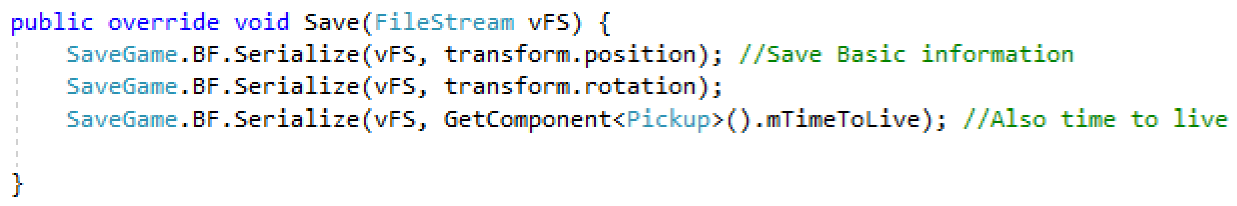
Add Component

USING PREFABS

How does the in-game object differ from its default in the prefab?

Its now at a new position, moving with some velocity, and at a unique angle.

If we spawn the prefab and overwrite just those variables it will be the same as it was in game!



```
public override void Save(FileStream vFS) {
    SaveGame.BF.Serialize(vFS, transform.position); //Save Basic information
    SaveGame.BF.Serialize(vFS, transform.rotation);
    SaveGame.BF.Serialize(vFS, GetComponent<Pickup>().mTimeToLive); //Also time to live

}
```

GAME SAVE DESIGN

We save an object count

Follow it with any number of DataObjects which require saving, each object is preceded by the name of the prefab it will be recreated with

Loading works in reverse

We get the object count so we know how many to load

Each object knows what it will be from the prefab name

ObjectCount (int)

Object 1

- PrefabName (string)
- SaveData

Object 2

- PrefabName (string)
- SaveData

Object N

- PrefabName (string)
- SaveData

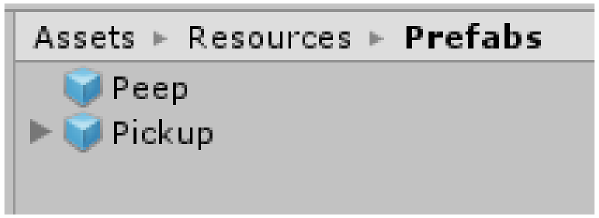
CHICKEN & EGG (SELF CREATION)

Objects know how to create themselves from their prefab on disk rather than linked in the IDE

```
string tPrefabName = (string)BF.Deserialize(tFS); //Which Prefab should we load
var tPrefab = Resources.Load<GameObject>(tPrefabName); //Load prefab
if (tPrefab == null) throw new Exception("No Prefab found");  //Throw error
var tLoadObject = Instantiate(tPrefab).GetComponent<Serialise>(); //Make new prefab and get its Serialise Component
tLoadObject.Load(tFS); //Set positions
```

Will load from Resource Folder, anything loaded MUST be in there

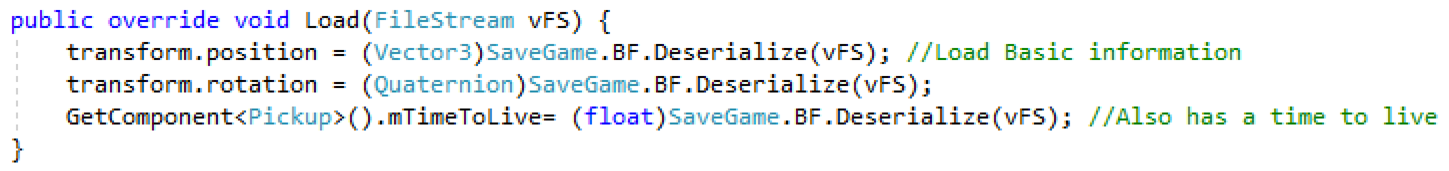
Only this folder is copied to build
Put run time loaded resources in here only



OBJECTS SELF LOAD/SAFE

Only saves/loads what it needs

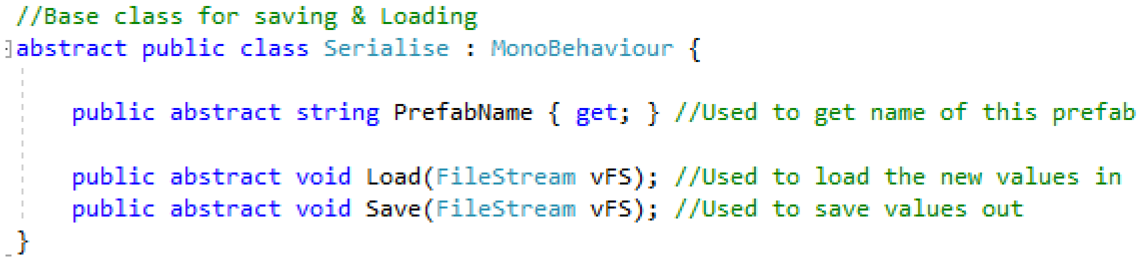
Creates itself from prefab
Then loads in the values to reflect the state of the game when it was saved



```
public override void Load(FileStream vFS) {
    transform.position = (Vector3)SaveGame.BF.Deserialize(vFS); //Load Basic information
    transform.rotation = (Quaternion)SaveGame.BF.Deserialize(vFS);
    GetComponent<Pickup>().mTimeToLive= (float)SaveGame.BF.Deserialize(vFS); //Also has a time to live
}
```

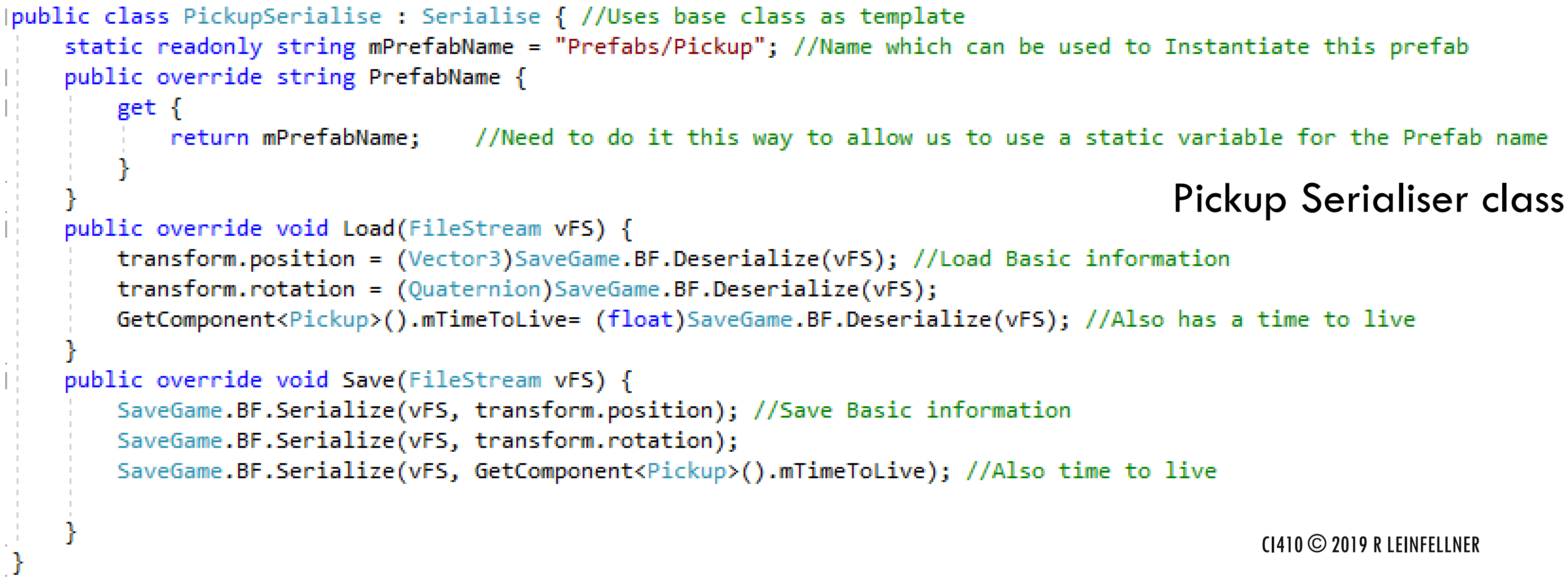
EACH OBJECT KNOWS WHAT TO LOAD

Each object for saving has a Component inherited from a abstract base class called Serialise



```
//Base class for saving & Loading
abstract public class Serialise : MonoBehaviour {

    public abstract string PrefabName { get; } //Used to get name of this prefab

    public abstract void Load(FileStream vFS); //Used to load the new values in
    public abstract void Save(FileStream vFS); //Used to save values out
}
```

Base class

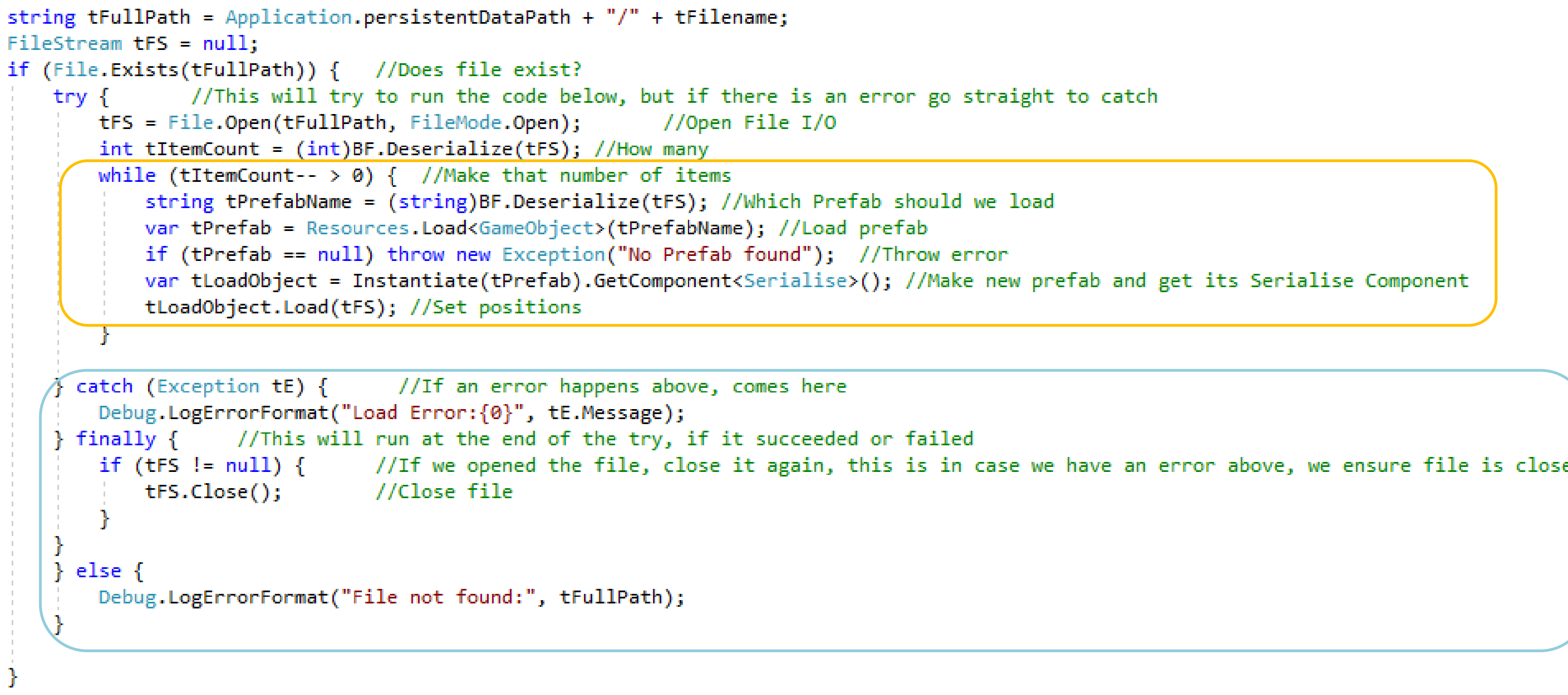


```
public class PickupSerialise : Serialise { //Uses base class as template
    static readonly string mPrefabName = "Prefabs/Pickup"; //Name which can be used to Instantiate this prefab
    public override string PrefabName {
        get {
            return mPrefabName;    //Need to do it this way to allow us to use a static variable for the Prefab name
        }
    }
    public override void Load(FileStream vFS) {
        transform.position = (Vector3)SaveGame.BF.Deserialize(vFS); //Load Basic information
        transform.rotation = (Quaternion)SaveGame.BF.Deserialize(vFS);
        GetComponent<Pickup>().mTimeToLive= (float)SaveGame.BF.Deserialize(vFS); //Also has a time to live
    }
    public override void Save(FileStream vFS) {
        SaveGame.BF.Serialize(vFS, transform.position); //Save Basic information
        SaveGame.BF.Serialize(vFS, transform.rotation);
        SaveGame.BF.Serialize(vFS, GetComponent<Pickup>().mTimeToLive); //Also time to live

    }
}
```

Pickup Serialiser class

CREATE OBJECT ON LOAD

Uses Get the name of the prefab, instantiates it, then asks it to load the data it needs



```
string tFullPath = Application.persistentDataPath + "/" + tFilename;
FileStream tFS = null;
if (File.Exists(tFullPath)) {   //Does file exist?
    try {       //This will try to run the code below, but if there is an error go straight to catch
        tFS = File.Open(tFullPath, FileMode.Open);       //Open File I/O
        int tItemCount = (int)BF.Deserialize(tFS); //How many
        while (tItemCount-- > 0) {  //Make that number of items
            string tPrefabName = (string)BF.Deserialize(tFS); //Which Prefab should we load
            var tPrefab = Resources.Load<GameObject>(tPrefabName); //Load prefab
            if (tPrefab == null) throw new Exception("No Prefab found");  //Throw error
            var tLoadObject = Instantiate(tPrefab).GetComponent<Serialise>(); //Make new prefab and get its Serialise Component
            tLoadObject.Load(tFS); //Set positions
        }

    } catch (Exception tE) {      //If an error happens above, comes here
        Debug.LogErrorFormat("Load Error:{0}", tE.Message);
    } finally {     //This will run at the end of the try, if it succeeded or failed
        if (tFS != null) {      //If we opened the file, close it again, this is in case we have an error above, we ensure file is close
            tFS.Close();        //Close file
        }
    }
    } else {
        Debug.LogErrorFormat("File not found:", tFullPath);
    }
}
```

WORKSHOP

Download example code
https://github.com/RLTeachGit/UnitySaveLoad2.git

Test Load & Save

Work Along Gist (link in Student Central)

We will add code to make pickups different colours

Self directed work

Add code to Spawner to add different scores to each Pickup

Add Score UI and have player score per pickup touched

Ensure the scores are stored and loaded

PopQuiz: What's the problem with Pickup code NB: Pickup destroyed after a period of time after they touch it, how could this break the save/load?