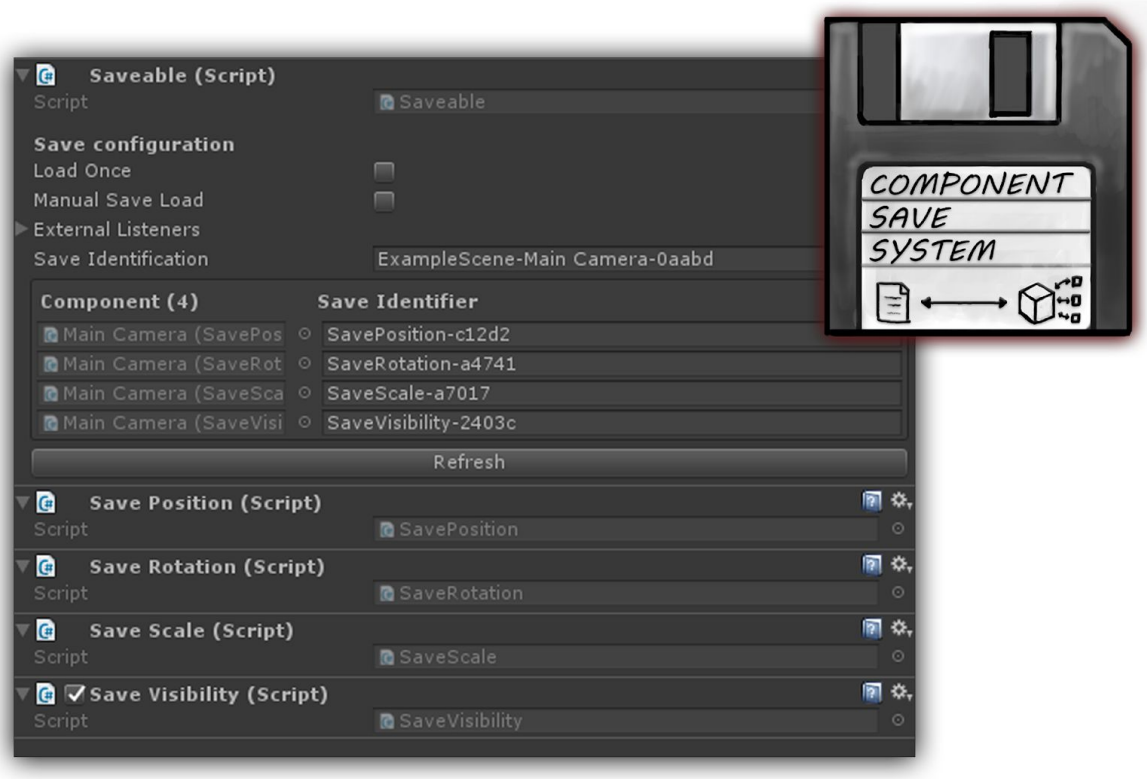


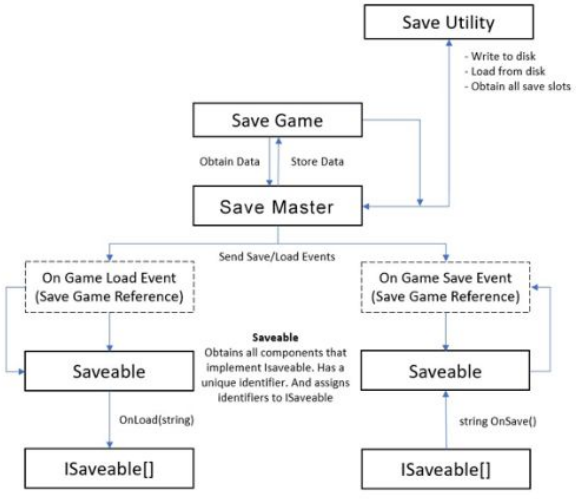
INTRODUCTION

"Component Save System" is a free save system for Unity, created by Alex Meesters from Lowscope. The plugin was originally made for the RPG Farming Kit, but has been expanded upon to work in multiple projects. The decision to make this plugin free was because there are already a lot of paid alternatives to saving for Unity, such as Easy Save.

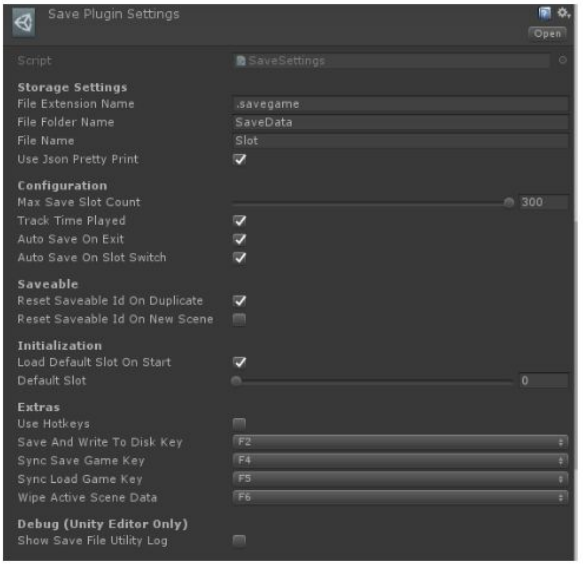
I felt that an inclusion of a system would be good for the ecosystem of Unity projects. Making it possible to include a free save tool into sample games or kits. I also felt like Unity lacked an easy way of saving components, akin to the current modularity the engine has.

HOW DOES IT WORK?

1. Once the plugin has been added to your project, a global instance called Save Master will be instantiated before any scene is loaded. This makes it as easy as possible to get started with the plugin, making it plug and play out of the box.
2. On startup it tries to load and create a save by default, based on a save slot. This can be turned off in a settings window.
3. In order to save and load save games, it uses the Save Utility to get/create the appropriate files.
4. Once the Savegame class has been retrieved or created, the save master will inform all currently subscribed Saveable of this reference, as well as all newly spawned objects that contain Saveable components.
5. Once a Saveable component is informed (On Load), it will attempt to apply all data from the save game to each component that implements ISaveable. The identification is based on GUIDS. <SceneName><GameObjectName + GUID><ComponentName + GUID>.
6. Once a Saveable component gets destroyed it will sent the current state of all the ISaveable components back to the save game.
7. Once the Savemaster gets destroyed or the game gets paused (out of focus in Android) the current savegame will get written to the harddisk. You can also call SaveMaster.WriteActiveSaveToDisk() to force this, and you can also disable the auto write of a save in the configuration settings in case you want to have specific save spots in the game.



Preview of how the architecture works



Preview of the possible configuration options

WHAT DOES THIS SYSTEM ADD IN COMPARISON TO OTHER SAVE SYSTEMS?

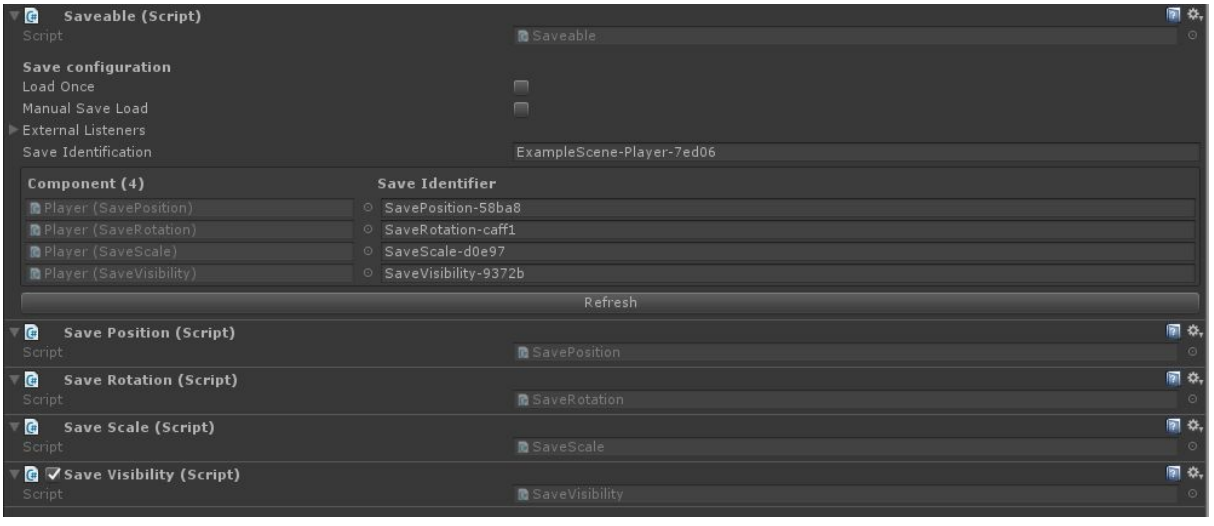
Most save systems do not handle the work of giving each object a unique identification. For instance, I want two enemies in my scene with a uniquely saved position, visibility and such. You could store a string like this with a popular saving system:

```
ASaveSystem.Set<string>("my saved content", "<scenename><objectname><scriptname>")
```

However, once you rename your object or scene name you already run into issues. Also, you would have to rewrite the same implementation in each script, each time to create a proper identification. It could easily turn into a mess, adding more work to turn it into a extendable system. In this case, most of that legwork has already been done for you. Only requiring you to write implementations for your components.

ADD A "SAVEABLE" COMPONENT TO AN OBJECT AND LET THE SAVE SYSTEM HANDLE THE REST FOR YOU.

Once you add a Saveable component to the root of a game object, it will fetch all components that implement ISaveable, and serve as a messener and identifier for the object and the components that implement the interface. The saveable is responsible for sending messages to all components that implement ISaveable, for instance once a save or load request has been made. Once a Saveable gets destroyed, the data gets automatically sent to the save game file, meaning that when you switch scenes or exit to the main menu everything is kept saved.



Example of the "Saveable" component, which is added to the root of a object that you want to save components on

HOW TO CREATE YOUR OWN SAVEABLE COMPONENTS

Creating a saveable component is quite straightforward, atleast if you use a IDE like Microsoft Visual Studio or Rider. Which make it very easy to implement interfaces into existing classes by pressing ALT+Enter.
For your component you have to add this using statement:

```
using Lowscope.Saving;
```

Afterwards you can add the ISaveable interface to your component. Which adds three methods to your class.

```
public void OnLoad(string data)
{
    // Data has been given to this component
    // I can now do something with it!
}

public string OnSave()
{
    return "The data that I want to save";
}

public bool OnSaveCondition()
{
    // Should I save, return true or false to potentially improve performance.
    return true;
}
```

You may be wondering, how can I use a string to send and retrieve data and make it usable for my object? The easiest answer is to use JSON. A text based representation of data. Unity has a built-in tool to turn objects into a JSON string and back into an object. This is called the JSONUtility. Below is an example that uses this utility. This same example is also shown on the github repository.

```
using Lowscope.Saving;
using UnityEngine;

public class TestScript : MonoBehaviour, ISaveable
{
    [System.Serializable]
    public class Stats
    {
        public string Name = "Test Name";
        public int Experience = 100;
        public int Health = 50;
    }

    [SerializeField]
    private Stats stats;

    // Gets synced from the SaveMaster
    public void OnLoad(string data)
    {
        stats = JsonUtility.FromJson<Stats>(data);
    }

    // Send data to the Saveable component, then into the SaveGame (On request of the save master)
    // On autosave or when SaveMaster.WriteActiveSaveToDisk() is called
    public string OnSave()
    {
        return JsonUtility.ToJson(stats);
    }

    // In case we don't want to do the save process.
    // We can decide within the script if it is dirty or not, for performance.
    public bool OnSaveCondition()
    {
        return true;
    }
}
```

The representation of this data will look like this once saved:

```
ô{
    "metaData": {
        "gameVersion": 0,
        "creationDate": "11/26/2019 2:47:31 PM",
        "timePlayed": "00:00:20"
    },
    "saveData": [
        {
            "guid": "TestScene-TestScriptGameobject-d4dbf-TestScript-ac11c",
            "data": "{\"Name\":\"Test Name\",\"Experience\":100,\"Health\":50}"
        }
    ]
}
```

STORING SPAWNED OBJECTS. FOR INSTANCE PICKUPS THAT YOU DROP/PICKUP

Within the SaveMaster class there is a method called
SpawnSavedPrefab(InstanceSource source, string filePath)
Currently there is only support of the Resources folder as a source. How it works is, each scene has a Instance Manager, which keeps track of what has been spawned, and the save identification it contained. Using the given path it will spawn it again using the tracked data.

CREATOR & CONTACT INFO

This asset has been developed by Alex Meesters from Lowscope.
For any questions, you can ask them through info@low-scope.com