Flexible save system

Introduction

The flexible save system is an easy-to-use implementation to store runtime data for different GameObjects. You can save and load at runtime, deleted objects will get back spawned if they were saved and exist as prefab.

How does it work?

Each GameObject which shuld save data must inherit from an interface: ISaveable and have to implement some functions. The SaveLoadSystem object will collect all object data and serialized to a binary file.

To respawn objects, a prefab for this object will is necessary. I created a scriptable object which holds all prefabs needed to respawn your saved objects. It should exist only one instance of this scriptable object in the project. Each prefab will get a prefab ID.

Each saveable object will also get an ID. This ID must be unique.

Usage

Let's say you want to implement the save system for you Player:

In the Player script, inherit from ISaveable.

```
4   using SaveLoadSystem;
5
6   [RequireComponent(typeof(SaveableEntity))]
    Unity-Skript (1 Objektverweis) | 0 Verweise
7   public class Player : MonoBehaviour, ISaveable
8   {
9       [SerializeField] int age;
10      [SerializeField] string language;
```

Import the namespace "SaveLoadSystem".
I recommend to add the "RequireComponent" part from line 6, because you need a SaveableEntity as component for each object which inherits from this interface.

Create a struct which holds all saveable data

```
18      [System.Serializable]
        5 Verweise
19      struct PlayerData
20      {
21          public int age;
22          public string language;
23      }
```

The struct must be declared serializable.
You can only use serializable types in it, Vector3 and Quaternion for example will not work.
For Vector2, Vector3, Vector4 and Quaternion, use these structs:

```
SaveableEntity.Vector2Data vec2;
SaveableEntity.Vector3Data vec3;
SaveableEntity.Vector4Data vec4;
```

Implement the methos from the interface.

```
2 Verweise
public object SaveState()
{
    // Instantiate the struct which contains the data we want to save and return it as object
    return new PlayerData()
    {
        age = age,
        language = language
    };
}
```

This function will return to data which will be stored in the binary file.
This function will be called when the object gets saved.

```
public void LoadState(object state)
{
    PlayerData data = (PlayerData)state; // Receive a object which we need to
                                         // cast to extract our loaded data
    this.age = data.age;
    this.language = data.language;
}
```

The reverse function will load in the casted struct.
This function will be called when the savefile gets loaded.

```
public bool NeedsToBeSaved()
{
    return true;
}
```

In this function, you can decide, if this object should be stored when the store event gets called.

```
public bool NeedsReinstantiation()
{
    return true;
}
```

If the object you are loading needs to be reinstantiated, you have to return true. If it is enough if the saved data is loaded back to the maybe existing object than you can return false.

This option will be set dominant if you have multiple ISavable Interfaces on an object with a SaveableEntity component.

This option will only work if a prefab is defined, otherwise a reinstantiation is not possible.

```
53      public void GotAddedAsChild(GameObject obj, GameObject hisParent)
54      {
55
56      }
```

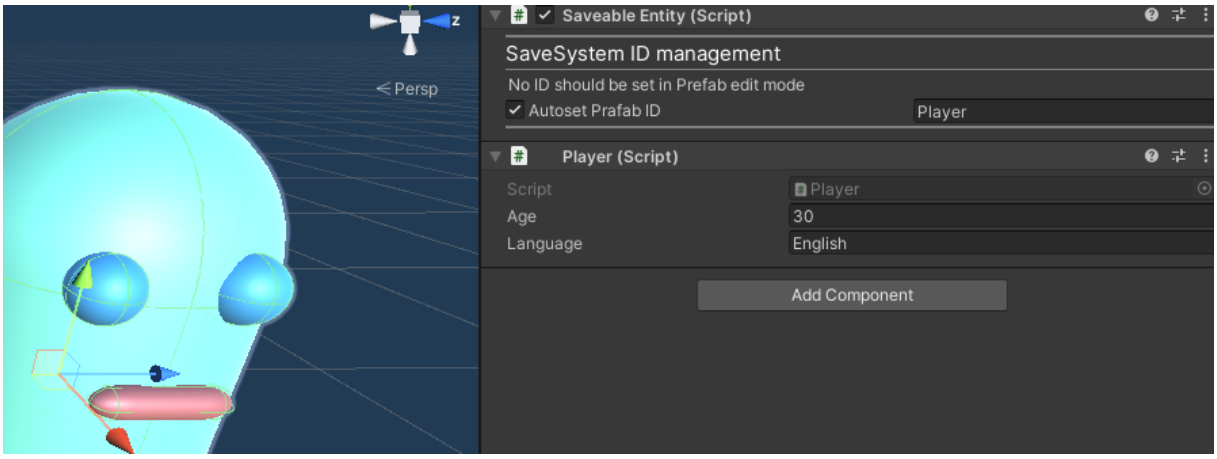
When a save gets loaded, objects will get Instantiated back to rebuild the structure of the original object. This function will get called, when during the load process, any object got added as child in any child of this object. In between this object and the <obj> is no other ISaveable. This event will only be triggered on the next higher ISaverable relative to the <obj>.

```
60      public void PostInstantiation(object state)
61      {
62          PlayerData data = (PlayerData)state;
63
64      }
```

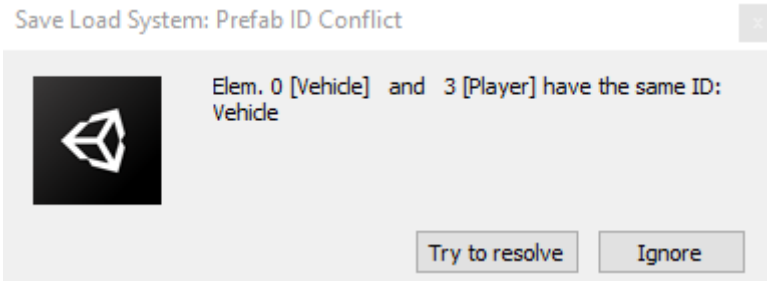
This function will receive the same data as LoadState().
The difference here is, that you may want to connect things with other saved objects. In this case we can't do that in the LoadState() function because the desired target object, may not exist yet.

So in this function you can search your targets. Use SaveableEntity.FindID(string id) to search for an id. You need to save the id of the target when saving the object to be able to find the correct object again.

Attach the script to your prefab/object

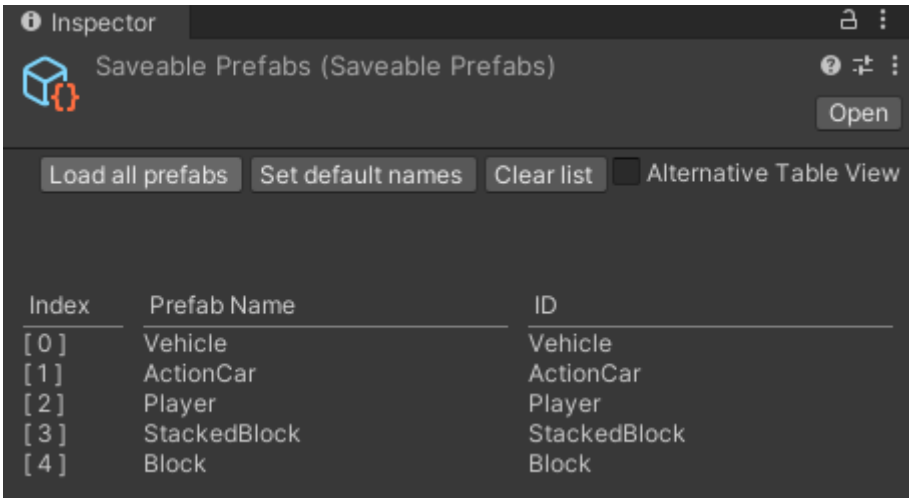


In case you have a prefab, you can set the Prefab ID here. This will also be done automatically.
If you have multiple Prefabs with the same name, change the prefab ID here manually.



If you forget to set a different prefab ID, than you will get a warning when you start playing the game. You can then click on “Try to resolve” to automatically change the ID’s so that each ID exists only once.

Create the Saveable prefab list

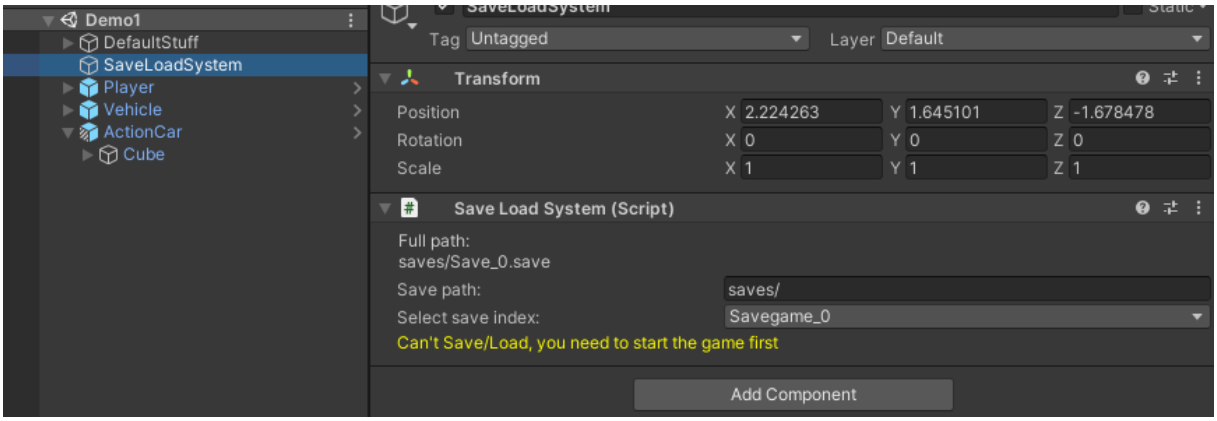


In case no such instance already exists (check the folder “SaveLoadSystem/Resources” first) you must create a prefab instance of type SaveablePrefabs in a Resources folder. Make sure that only one instance exists in the project.

Your prefabs which have the “SaveableEntity” component attached to it will automatically added to this list. The “SaveableEntity” component must be attached to the root GameObject of the prefab and also the script which holds the ISaveable interface.

If you can’t see the Table shown in the picture, when you click on the SaveablePrefabs scriptable object, then you can click on the toggle button “Alternative Table View”.
I had this issue but don’t know why, it must be some bug in Unity. I had that problem on Unity 2020.3 and tested it on 2021.3 and there it works fine.

Add the SaveLoadSystem to the scene



Insert the prefab: “SaveLoadSystem/SaveLoadSystem.prefab” in your scene. This will manage the loading and saving. you can then change the path in which the saves will be stored. They will be stored relative to Application.persistentDataPath. If you want any other path than that, you have to use a custom save script which will set the desired paths.

I created some slots you can chose to save and load when using the editor buttons but when you call the save/load events via script, you can specify the exact path of each savegame you want to save/load.

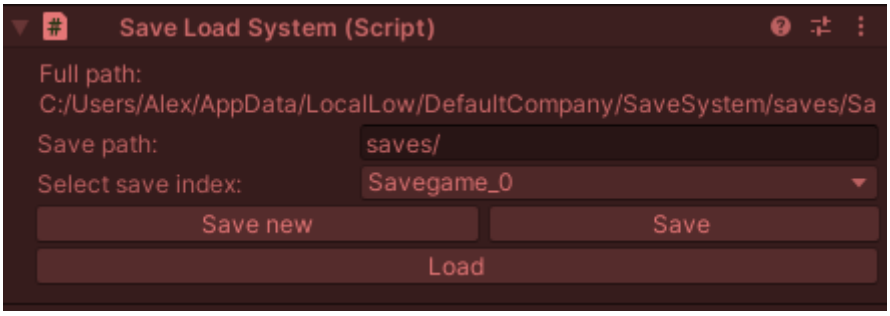
Just call:

```
SaveLoadSystem.basePath = "...";
SaveLoadSystem.savePath = "...";
SaveLoadSystem.saveName = "...";
```

The full path of the save will be assembled like this;

Path = basePath + savePath + saveName

So you have to add the “/” for directories and the “.xy” file ending.



When you now start the game, you can use the editor interface from the SaveLoadSystem script to save and load your scenes to test stuff, later you will implement the save load mechanic via a script.

Save new, will override any existing save.

Save will not override, it will add to an existing savegame.

Contact

alexkrieg@gmx.ch