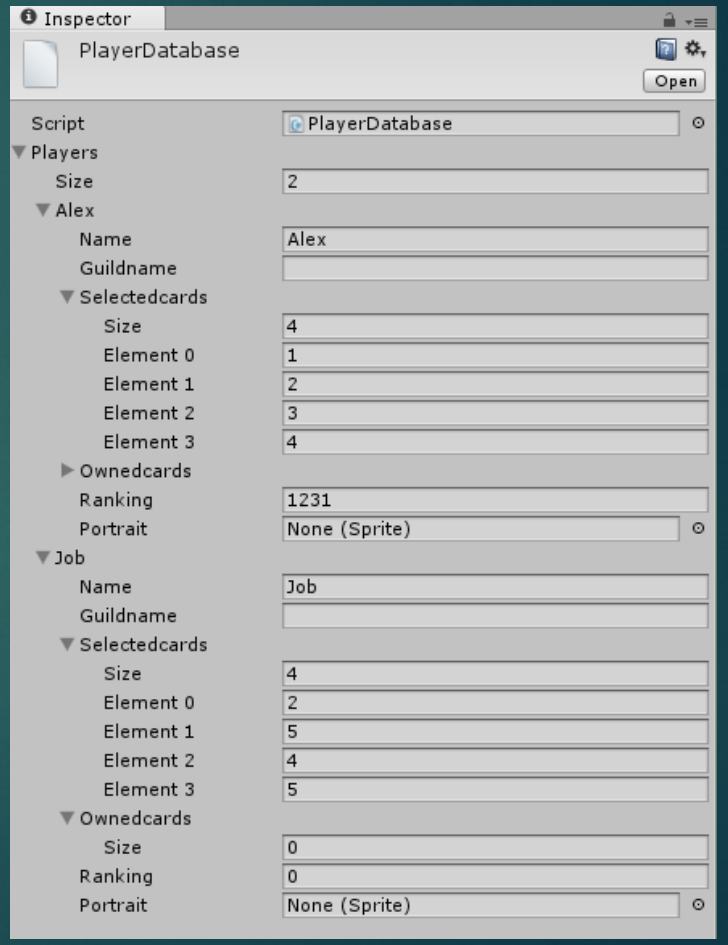
Scriptable Objects

SUSHI ROLL MOBILE GAME - ALEX MEESTERS (142783)

What is a scriptable object in Unity?

- It is an .asset file that stores information outside of mono behaviour. This means that the data will be kept even when exiting play mode.
- Scriptable objects allow for quicker prototyping, since no changes will ever get lost during the play/stop process of the game.
- Scriptable object data can be written to a .json or .xml
- Scriptable Objects are for Editor use only. They are not meant to be written during play-mode, only to be loaded. Unless you are using it to tweak settings.



Possible applications of scriptable objects

- **Having a big item / monster database**
  Which can be edited through an advanced editor window.
  Benefits of having this in engine, is that it can be tested quickly.
- **Preset player data**
  Quick testing of different player scenarios.
- **Game behavior settings**
  For tweaking of game behavior.
- **Neural network data**
  Saving neural data while you are in play mode,
  so that it does not get destroyed afterwards. (Editor Only)
- **Level data storage**
  When having a custom level editor within the Unity Engine editor.
- **Pluggable AI or Ability systems**
  It is possible to make an entity iterate over a list of scriptable objects
  that derive from an abstract class.

How are scriptable objects created?

Create a serializable class inherited from scriptable object

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

[System.Serializable]
public class CardDatabase : ScriptableObject {

    [SerializeField]
    public List<Card> Cards;
    public Sprite[] Frames;
    public Sprite[] Backgrounds;

    public void OnEnable ()
    {
        if (Cards == null) {Cards = new List<Card>();}
    }
    public void Add (Card card)
    {
        Cards.Add(card);
    }
    public void Remove (Card card)
    {
        Cards.Remove(card);
    }
    public void RemoveIndex (int index)
    {
        Cards.RemoveAt (index);
    }
    public int COUNT {
        get { return Cards.Count; }
    }

}
```

Example code from card database class.

Use the ScriptableObject.CreateInstance<Classname>() function and save it as an asset with AssetDatabase.CreateAsset() function.

```
using UnityEngine;
using System.Collections;
using UnityEditor;

public class MakeScriptableObjects {
    [MenuItem("Assets/Create/My Scriptable Object")]
    public static void CreateMyAsset()
    {
        CardDatabase asset = ScriptableObject.CreateInstance<CardDatabase>();

        AssetDatabase.CreateAsset(asset, "Assets/Resources/Datacontainers/CardDatabase.asset");
        AssetDatabase.SaveAssets();

        EditorUtility.FocusProjectWindow();

        Selection.activeObject = asset;
    }
}
```

Example code for creation of scriptable object. Editor Window Script.

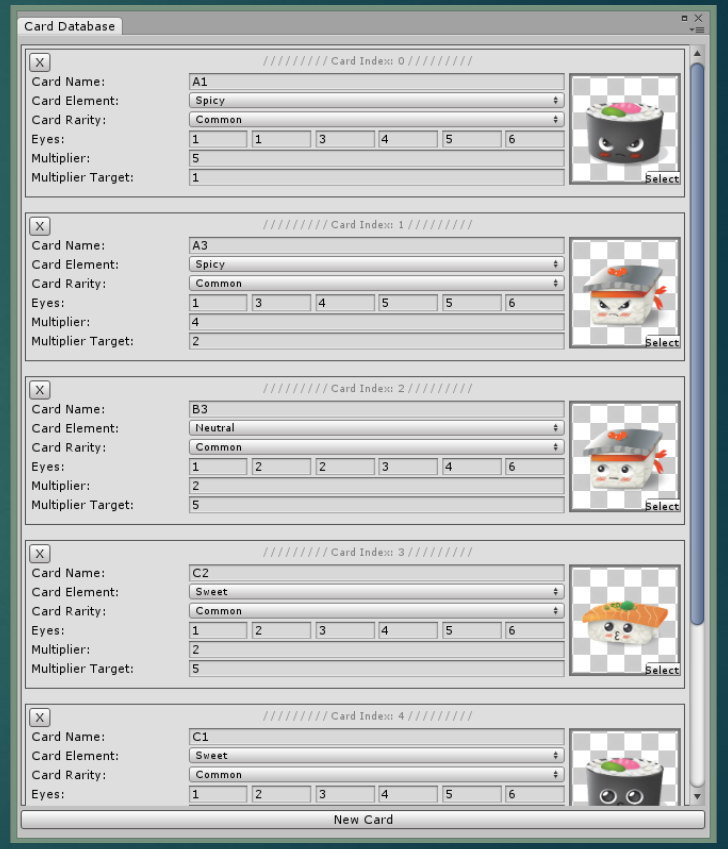
Why use a scriptable object database for a unity prototype?

Advantages

- It provides quick access to data.
- It keeps things organised.
- Access can be made designer friendly
- Easier to transfer data through source control
- All inserted data can be serialised (for example sprites)

Disadvantages

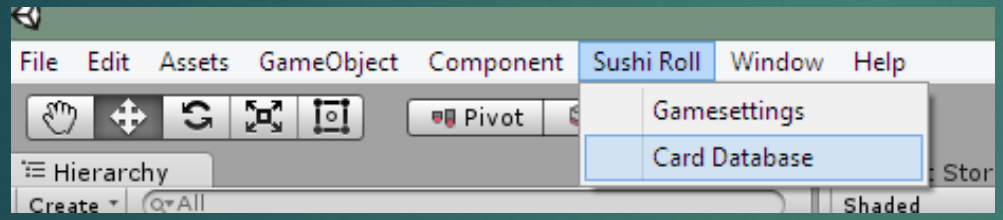
- Takes time to set up
- Takes time to reprogram if there are design changes



Card database editor I made for Sushi Roll.

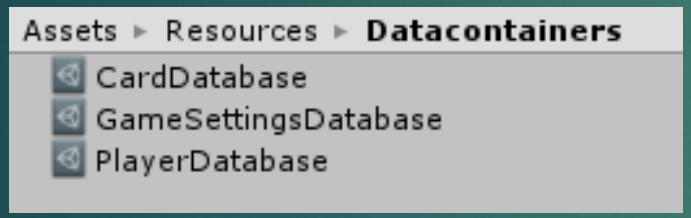
Quick access to data / Designer friendly access

- You can make editor scripts to get quick access to data just from one location. This is especially good for designers, the customizability of editor windows is good to ensure they are not making changes they should not make.



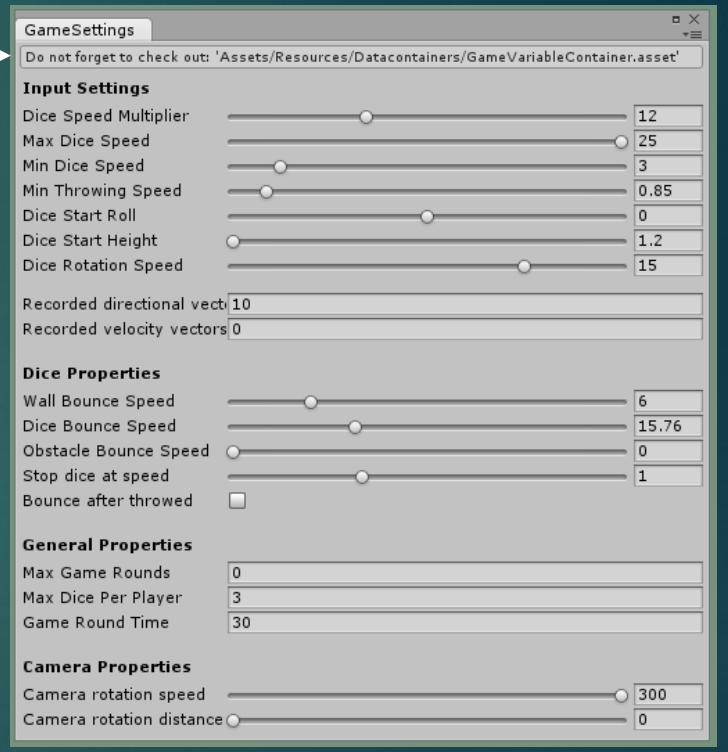
It keeps things organized

- All data files are at one place. No need to store the data in each scene. Meaning that all the data will be the same independently of what scene you are running.



Easier to transfer data over source control

- It is possible to create multiple instances of scriptable object, this can provide benefits for version control, example: say you have a item database with 200 items, where a designer wants to edit one item. Normally you would check out one big database file. In this case other designers can still edit the other files with ease.
- One of the designers of my team wanted to create and setup data for settings for the game. All he had to do is, change the information and commit the data file specified in "/resources/Datacontainers/PlayerDatabase" Ideally this would happen with the P4V Unity plugin, but I did not have the time to set this up.



All inserted data can be serialised

- One of the extra benefits I discovered was that all inserted data to an Scriptable Object is inserted into the build (Sprites for example), without the need to reference it to a object in the scene. It is important to still keep in mind that this can be a potential problem if you do not want all the data to be accessible. (To lower build size)

Image received from the card database
(Other data as well)

Takes time to set up

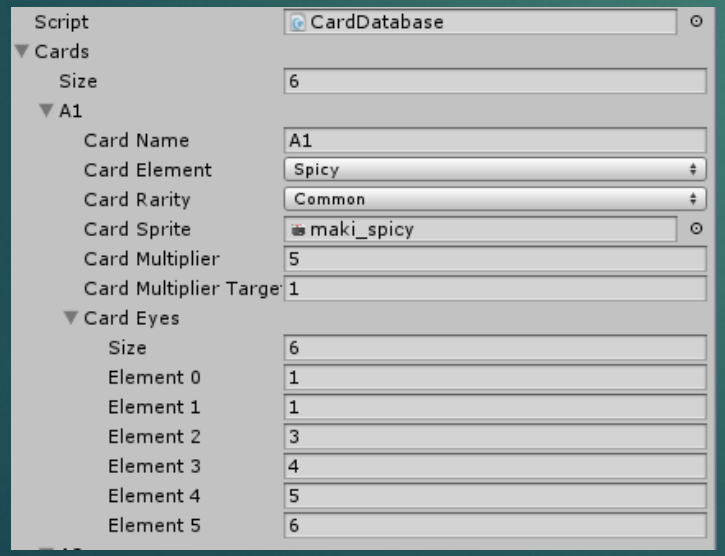
- You will have to create a system that prevents you from going mad. Loading data can be very hard to maintain, or at least it was for me. This is why I have created functions to make it easier to load data. But creating these take time.
- I am giving an example on the next pages of how I simplified loading data from scriptableobjects. The examples come from my my DatabaseController.cs

DatabaseController.cs

- Simply load in the scriptable objects with the Resources.Load() function.

```
public void Awake ()
{
    CardData = Resources.Load<CardDatabase> ("Datacontainers/CardDatabase");
    PlayerData = Resources.Load<PlayerDatabase> ("Datacontainers/PlayerDatabase");
    ReturnSpriteData = new Sprite[3];
}
```

Visualisation of card data in scriptableobject

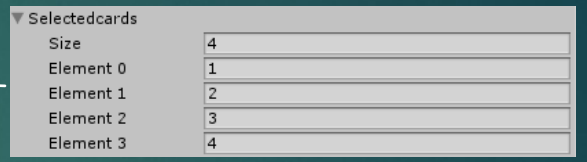


Visualisation of player data in scriptableobject

DatabaseController.cs

This is a function that returns me the current card that the player holds from the player database

```
private Card CardIndexOnPlayer (int cardindex,int player)
{
    return CardData.Cards[PlayerData.Players[player].Selectedcards[cardindex]];
}
```



These functions use the function above to know which card to load from the card database, and return that data.

```
public Sprite[] ReturnPlayerCardImageData (int cardindex,int player)
{
    // Get the border (element)
    ReturnSpriteData [0] = GetBorder (CardIndexOnPlayer(cardindex,player).CardRarity);
    // Get the background (element)
    ReturnSpriteData [1] = GetBackground (CardIndexOnPlayer(cardindex,player).CardElement );
    // Get the character
    ReturnSpriteData [2] = CardIndexOnPlayer(cardindex,player).CardSprite;

    return ReturnSpriteData;
}
```

Returns array of sprites to be used.
GetBorder() and Background() work like this:

```
private Sprite GetBackground (Card.Elements Element)
{
    int ElementIndex = -1;
    switch (Element) {
    case Card.Elements.Spicy: ElementIndex = 0; break;
    case Card.Elements.Neutral: ElementIndex = 1; break;
    case Card.Elements.Sweet: ElementIndex = 2; break;
    }
    return CardData.Backgrounds[ElementIndex];
}
```

```
public Card.Elements ReturnCardElementData (int cardindex,int player)
{
    // Find out which card the player currently has on that index
    return CardIndexOnPlayer(cardindex,player).CardElement;
}

public int[] ReturnCardEyeData (int cardindex,int player)
{
    return CardIndexOnPlayer(cardindex,player).CardEyes;
}

public float[] ReturnMultiplierData (int cardindex,int player)
{
    return new float[2] {
        CardIndexOnPlayer(cardindex,player).CardMultiplier,
        CardIndexOnPlayer(cardindex,player).CardMultiplierTargetNumber};
}
```

Simple functions that return data from the card class that is stored within the scriptable objects

AlexMeesters.nl

Example of a function that loads data using DatabaseController.cs

This sets the data of the four displayed cards. Using the the data from the scriptable objects, provided by the DatabaseController.cs

```
public void SetCardsToPlayer (int Player)
{
    for (int i = 0; i < 4; i++) {
        UICardScript [i].CardDisplayer.SetImageData( DataCtrlREF.ReturnPlayerCardImageData (i,Player) );
        UICardScript [i].SetCardData(DataCtrlREF.ReturnCardElementData(i,Player),
                                     DataCtrlREF.ReturnMultiplierData(i,Player),
                                     DataCtrlREF.ReturnCardEyeData(i,Player));
    }
}
```

Pass sprite array to card class.

Pass other data to card class.

Things to keep in mind when working with scriptable objects

- Remember to use EditorUtility.SetDirty(variable) when using a editor window script. Or else the changes will not be saved.
- Remember to mark a class that you want to be included within a scriptable object as [System.Serializable] and also add [SerializeField] to the variables within the class.
- Renaming scriptable object classes is not a smart idea, only do this if you want to reenter all the information. When you rename your scriptable object class, the scriptable object class will lose its reference towards the class. Switching the reference class resets all data contained within the scriptable object.

What I have learned

- How to apply scriptable objects to a Unity 3D project.
- How to access / write data to scriptable objects, while keeping it understandable
- How to create editor windows that are able to modify data from scriptable objects
- What the benefits are of using scriptable objects.

End

- Thank you for reading.