Tools for unity

Gaëtan Blaise-Cazalet - Head of programming section

Scriptable objects

Why using scriptable objects?

Imagine you want to finetune your game. Best way would be to change game values at runtime, then save those values.

Prefab system allow you to change values, but with two restrictions :

- If you stop the game the variable takes its previous value. You can go around by copying component state, but it is not handy, and won't work if you need to finetune values in different prefabs.
- You need to propagate the changes to other prefabs, which won't work if prefabs are already instanciated.

What are scriptable objects?

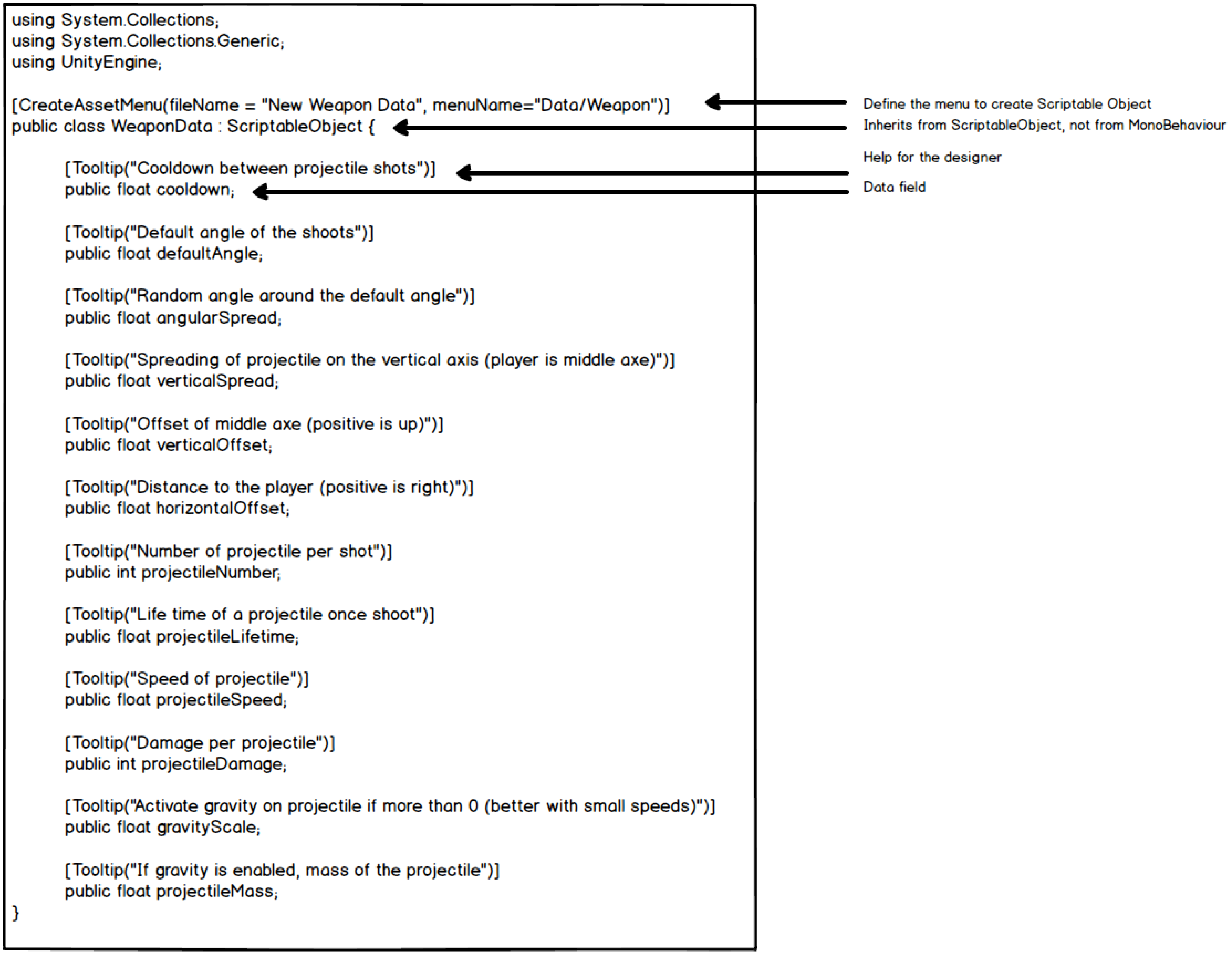
Scriptable objects are serializable (= convertible to files) data sets, that value can be changed anytime, whether the game is running or not. When a scriptable object value changes, the change is propagated into game objects that use the scriptable object.

Scriptable objects must be created inside unity.

Example

Platform shooting game.

Defining a Scriptable object



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

[CreateAssetMenu(fileName = "New Weapon Data", menuName="Data/Weapon")]
public class WeaponData : ScriptableObject {

        [Tooltip("Cooldown between projectile shots")]
        public float cooldown;

        [Tooltip("Default angle of the shoots")]
        public float defaultAngle;

        [Tooltip("Random angle around the default angle")]
        public float angularSpread;

        [Tooltip("Spreading of projectile on the vertical axis (player is middle axe)")]
        public float verticalSpread;

        [Tooltip("Offset of middle axe (positive is up)")]
        public float verticalOffset;

        [Tooltip("Distance to the player (positive is right)")]
        public float horizontalOffset;

        [Tooltip("Number of projectile per shot")]
        public int projectileNumber;

        [Tooltip("Life time of a projectile once shoot")]
        public float projectileLifetime;

        [Tooltip("Speed of projectile")]
        public float projectileSpeed;

        [Tooltip("Damage per projectile")]
        public int projectileDamage;

        [Tooltip("Activate gravity on projectile if more than 0 (better with small speeds)")]
        public float gravityScale;

        [Tooltip("If gravity is enabled, mass of the projectile")]
        public float projectileMass;
}
```

Define the menu to create Scriptable Object
Inherits from ScriptableObject, not from MonoBehaviour
Help for the designer
Data field

Using a scriptable object

Using a scriptable object in a prefab/game object code

Reference to scriptable object

Usage

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

public enum Shooter { Player, Enemy };

public class WeaponShoot : MonoBehaviour {

	[SerializeField] Projectile projectile;
	[SerializeField] WeaponData weaponData;
	[SerializeField] Shooter shooter;

	float cooldownCounter;

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {
		cooldownCounter += Time.deltaTime;
		if(Input.GetButton("Fire1") && cooldownCounter > weaponData.cooldown) {
			Vector3 projectilePosition = transform.position;
			projectilePosition.x += weaponData.horizontalOffset;
			projectilePosition.y += weaponData.verticalOffset;
			for(int i = 0; i < weaponData.projectileNumber; i++) {
				projectilePosition.y += Random.Range(-weaponData.verticalSpread, weaponData.verticalSpread);
				Projectile p = Instantiate(projectile, projectilePosition, Quaternion.identity);
				float angle = weaponData.defaultAngle + Random.Range(-weaponData.angularSpread, weaponData.angularSpread);
				p.Setup(angle, weaponData.projectileSpeed, weaponData.projectileLifetime,
					weaponData.projectileDamage, weaponData.gravityScale, weaponData.projectileMass, shooter);
			}
			cooldownCounter = 0;
		}
	}
}
```

Creating scriptable object

Use the menu Assets/Create/Data/... to create a scriptable object. It will appear in your assets.

Edit the values, then drag the scriptable object to your prefab.

Finetuning your game

Convert all your game design values to scriptable objects.

The dependancy injection problem
some words about it

What is the problem?

When you want to create your game with prefabs, you need to set all values and link all prefabs (dependencies) by hand. You have to drag and drop each prefab in each other prefab or game object. If you choose the wrong prefab or forget a value (or unset it accidently), your game won't work and it will be very intricate to find the error.

Separation of Concerns

Separation of Concerns is a Object oriented principle that states that a object shall only deals with its own problems. Other problems mean other classes.

In our case, our game object is dealing with two problems:

- Its internal logic
- The gathering of all dependencies before it starts

We want the second problem to be dealt with differently.

Inversion of Control

The idea of inversion of control is : all dependencies of an object should be injected from outside the object. So that the object has not to suppose anything about how the dependencies are injected.

Usually, dependencies are injected at game / scene startup.

Dependency injection with scriptable objects

We will use dependency injection to set all game values before the game starts.

We will create a unique global object (also called a Singleton) that will hold dependencies into a Scriptable objects.

Create a singleton to hold data

Singleton ?

A class that can be accessed globally and have a unique instance.

Our singleton will hold initialization data.

Code

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

public class DataAccess {

	public EngineMoveData EngineMoveData { get { return engineMoveData; } }
	EngineMoveData engineMoveData;

	public void Load() {
		engineMoveData = AssetDatabase.LoadAssetAtPath(
			"Assets/Data/Move/EngineMoveData.asset",
			typeof(EngineMoveData)
		) as EngineMoveData;
	}

	// Singleton
	private DataAccess() {
		Load();
	}

	private static DataAccess instance;
	public static DataAccess Instance {
		get {
			if (instance == null) instance = new DataAccess();
			return instance;
		}
	}

}
```

Classic class, not MonoBehaviour

Property to access data

At first call, load data from Scriptable Object

Private constructor, so the Singleton cannot be constructed from outside

Load the data when instance is constructed

Private instance and public accessor, so that the first access construct the instance.

Static means it can be accessed from the class, and not from a constructed object.

Dependency Injection Yourself

Reorganize your game data and code to use Dependency injection at startup, and get rid of complex prefab dependencies.

Custom editor 1/2

A custom scriptable object

Create a Editor folder anywhere in the Asset folder. Usually in the script folder.

For instance, i want a new Scriptable object that hold tiles for a tileset.

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

[CreateAssetMenu(fileName = "New Tile Data", menuName="Data/Tiles")]
public class TileData : ScriptableObject {

  public string mapName;
  public Sprite[,] mapSprites;
}
```

Edit the DataAccess class so it can hold a TileData scriptable object.

The problem is there is no default sprite field for the TileData Scriptable Object. We will create one. Create a TileEditor in the Editor folder.

Custom editor 2/2

Custom sprite fields for the Scriptable object

We have to code to edit the scriptable object's list of sprites :

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

[CustomEditor(typeof(TileData))]
public class TileEditor : Editor
{

  TileData comp;
  static bool showTileEditor = false;

  public void OnEnable()
  {
    comp = (TileData)target;
    if (comp.mapSprites == null)
    {
      comp.mapSprites = new Sprite[1, 1];
    }
  }

  public override void OnInspectorGUI()
  {
    comp.mapName = EditorGUILayout.TextField("Name", comp.mapName);

    int width = EditorGUILayout.IntField("Map Sprite Width", comp.mapSprites.GetLength(0));
    int height = EditorGUILayout.IntField("Map Sprite Height", comp.mapSprites.GetLength(1));
    if (width != comp.mapSprites.GetLength(0) || height != comp.mapSprites.GetLength(1))
    {
      comp.mapSprites = new Sprite[width, height];
    }

    showTileEditor = EditorGUILayout.Foldout(showTileEditor, "Tile Editor");

    if (showTileEditor)
    {
      for (int h = 0; h < height; h++)
      {
        EditorGUILayout.BeginHorizontal();
        for (int w = 0; w < width; w++)
        {
          comp.mapSprites[w, h] = (Sprite)EditorGUILayout.ObjectField(
                    comp.mapSprites[w, h], typeof(Sprite), false, GUILayout.Width(65f), GUILayout.Height(65f)
          );
        }
        EditorGUILayout.EndHorizontal();
      }
    }
  }

}
```

Going further

Editor customization API is quite big. You can start here:
https://www.raywenderlich.com/7751-unity-custom-inspectors-tutorial-getting-started