GUN SYSTEEM

Jordy Teunis

Contents

Introduction

The goal of the gun system is that it is easy to add a new gun to the game with the required criteria and that it is easy to understand and adjust. The gun system will use a scriptable object to store all the gun's information, and a script containing the gun functions so you can shoot in the project. To properly master this system, I researched the best way to set this up and practiced with different guns so that it becomes a quick and convenient system to use.

Methods

1. Analysing: Researching what a good gun system is
    - Type: Literature and audiovisual research
    - Description: Researching what the best and most adaptable gun system is to integrate with the project.
    - Sources: Unity Documentation and YouTube
2. Designing: Preparing the gun system
    - Type: The requirements for the gun system
    - Description: Mapping out the requirements for the gun system to have an overview.
3. Realizing: Gun system in the project
    - Type: Integrating the gun system in the project
    - Description: Making the gun system usable in the project.

Execution

1. Investigate what a good gun system is

From the various sources, it turns out that there are several approaches to creating a gun system. In general, you need a gun script that handles all the physics of the gun, and a script where you store all the variables for a specific gun so that you can easily add different weapons.

The gun object is always placed in the camera/camera holder so that the gun is clearly visible in first person and moves automatically with the camera through the integrated Unity system. This way, you can immediately place the gun in the desired location without much effort.

2. Preparing the gun system

Several objects are needed to complete the gun system. A gun system that is clear, easily adjustable, and simple includes creating a camera holder and gun holder in the player object and adding the camera and gun to them. As a child of the gun object, the gun model is added, making it clear which object the script uses and which is the model.

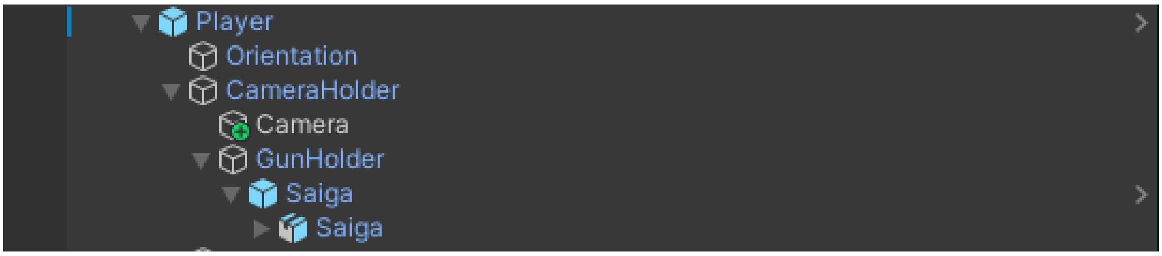


Figure 1: Hierarchy of the gun system.

The best way to store gun information is by using scriptable objects. In them, you define which variables the gun needs, such as reload speed. Because it's a scriptable object, it's easy to quickly add a new weapon since you only need to use one script for different guns.

A scriptable object is quick and easy to adjust and uses little to no performance to use, making it the best solution to store information about the gun.

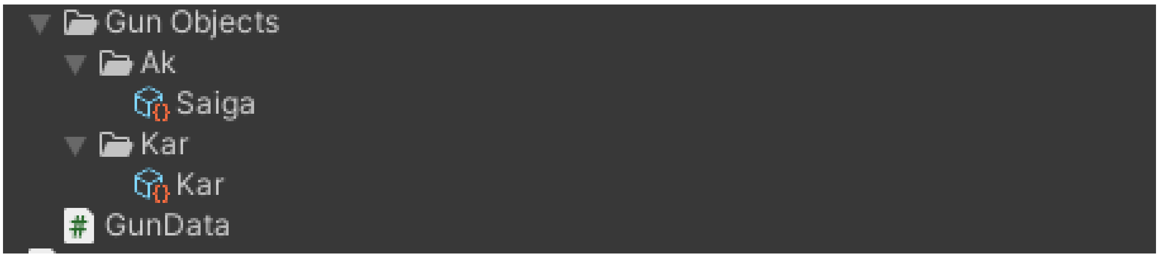


Figure 2: GunData

3. Gun system inside the project

Step 1: Setup of the Gun in the Hierarchy

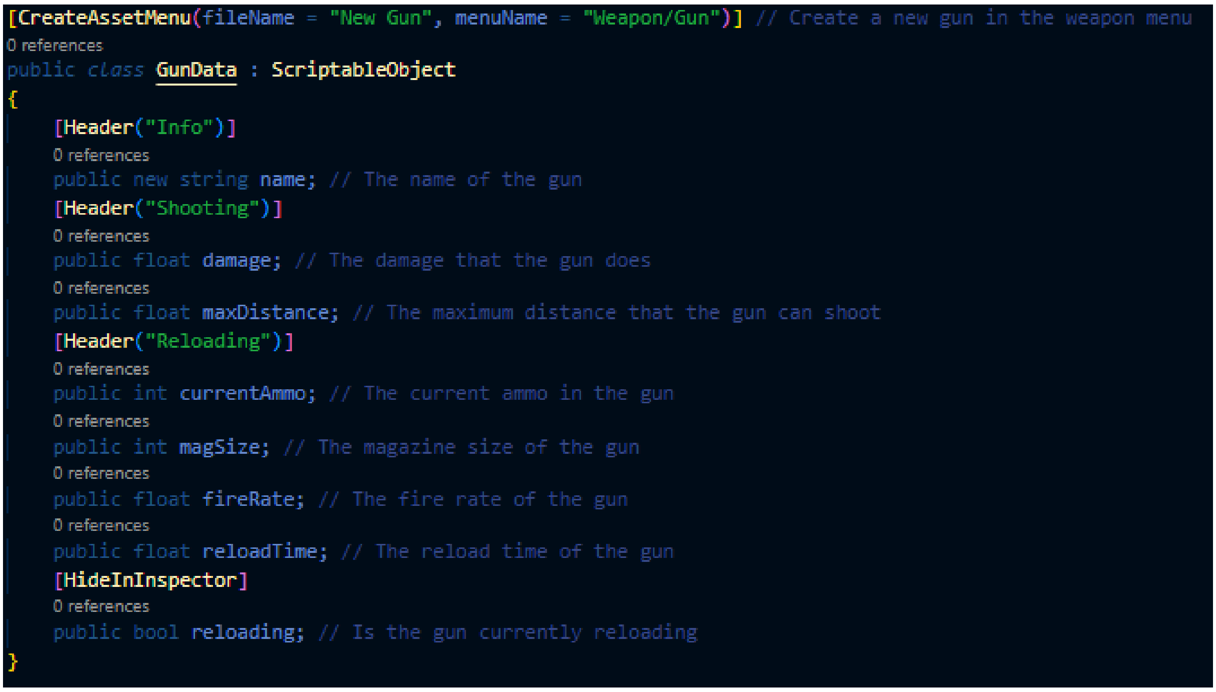
1. Add a CameraHolder to the Player:
    - Create a new GameObject as a child of the Player and name it "CameraHolder".
    - Add the Camera to the CameraHolder as a child.
2. Create a new GameObject for the Gun:
    - Create a new GameObject as a child of the CameraHolder and name it "GunHolder".
    - Add a new GameObject as a child of GunHolder and name it "Gun".
    - Add a script component to the Gun object named Gun.

(See hierarchy in 2. Prepare gun system)

Stap 2: GunData Scriptable Object

Create a GunData script in your project. In this script, we store all the information about a gun. Create a CreateAssetMenu in this script so you can easily add a gun with GunData through the create menu. Add the following variables:

- name
- damage
- maxDistance
- currentAmmo
- magSize
- fireRate
- reloadTime
- reloading



```
[CreateAssetMenu(fileName = "New Gun", menuName = "Weapon/Gun")] // Create a new gun in the weapon menu
0 references
public class GunData : ScriptableObject
{
    [Header("Info")]
    0 references
    public new string name; // The name of the gun
    [Header("Shooting")]
    0 references
    public float damage; // The damage that the gun does
    0 references
    public float maxDistance; // The maximum distance that the gun can shoot
    [Header("Reloading")]
    0 references
    public int currentAmmo; // The current ammo in the gun
    0 references
    public int magSize; // The magazine size of the gun
    0 references
    public float fireRate; // The fire rate of the gun
    0 references
    public float reloadTime; // The reload time of the gun
    [HideInInspector]
    0 references
    public bool reloading; // Is the gun currently reloading
}
```

Figure 3: GunData script

Once this script is saved, you can now create a gun in the project folder via the create menu and give it a name of your choice. Then fill in the values that best suit the gun you are creating.

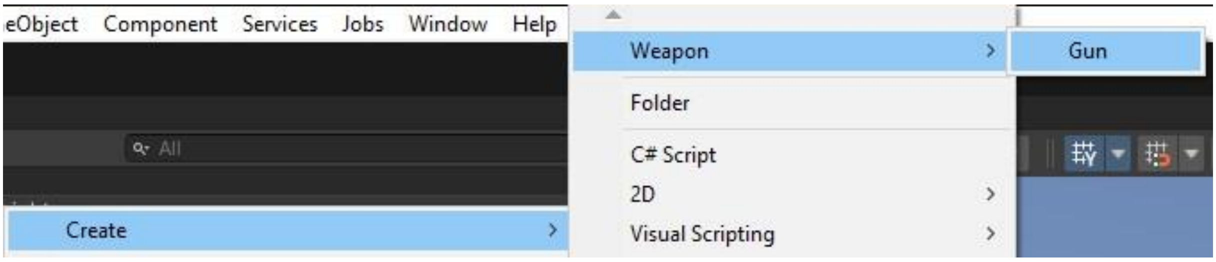
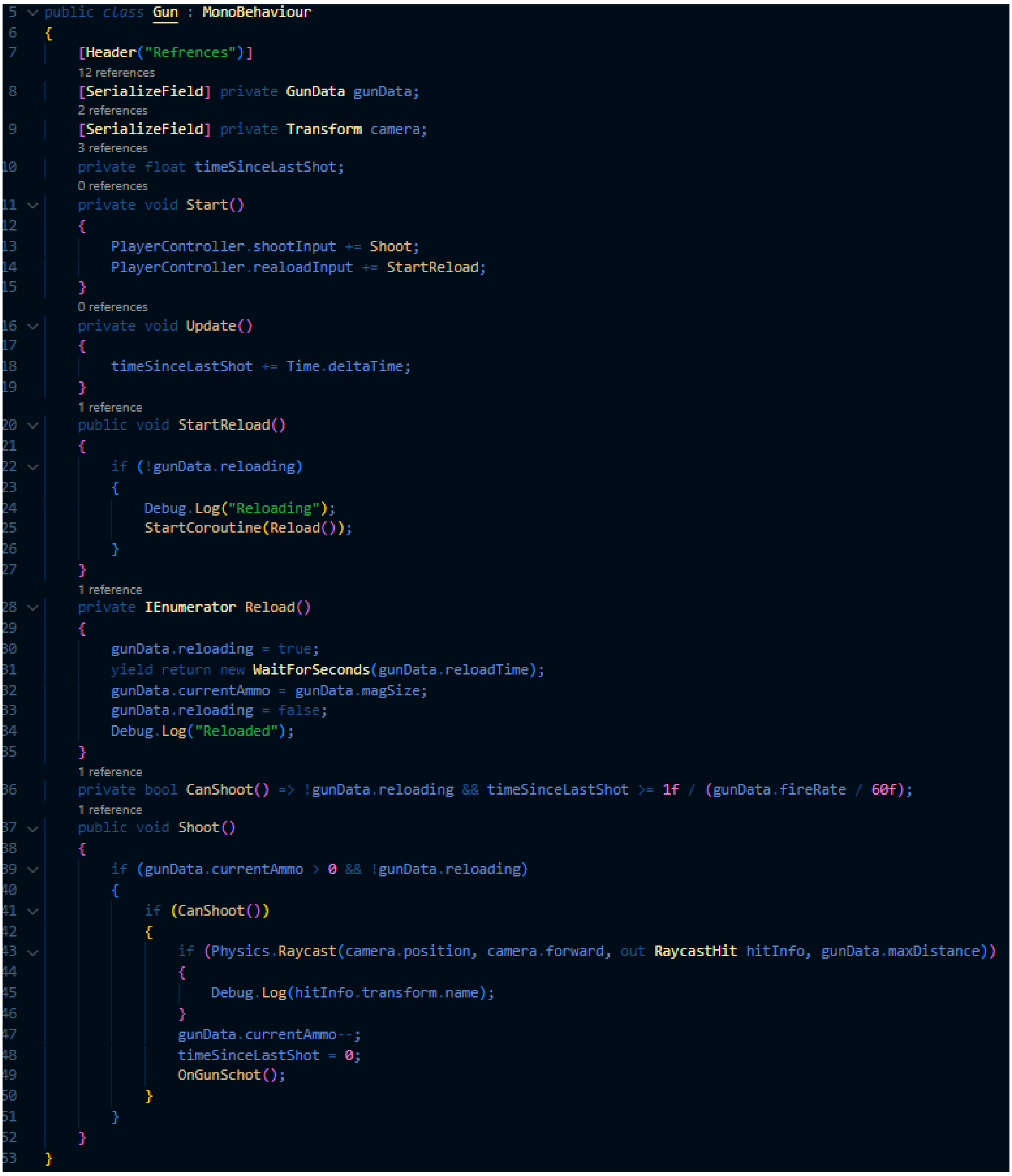


Figure 4: Create Gun via GunData

Step 3: Gun script

Adjust the newly created Gun script to reference GunData so it can be used by the Gun script. Then make a simple shooting system using the variables created in GunData so that the Gun script can be used for each gun.



```
public class Gun : MonoBehaviour
{
    [Header("Refrences")]
    12 references
    [SerializeField] private GunData gunData;
    2 references
    [SerializeField] private Transform camera;
    3 references
    private float timeSinceLastShot;
    0 references
    private void Start()
    {
        PlayerController.shootInput += Shoot;
        PlayerController.realoadInput += StartReload;
    }
    0 references
    private void Update()
    {
        timeSinceLastShot += Time.deltaTime;
    }
    1 reference
    public void StartReload()
    {
        if (!gunData.reloading)
        {
            Debug.Log("Reloading");
            StartCoroutine(Reload());
        }
    }
    1 reference
    private IEnumerator Reload()
    {
        gunData.reloading = true;
        yield return new WaitForSeconds(gunData.reloadTime);
        gunData.currentAmmo = gunData.magSize;
        gunData.reloading = false;
        Debug.Log("Reloaded");
    }
    1 reference
    private bool CanShoot() => !gunData.reloading && timeSinceLastShot >= 1f / (gunData.fireRate / 60f);
    1 reference
    public void Shoot()
    {
        if (gunData.currentAmmo > 0 && !gunData.reloading)
        {
            if (CanShoot())
            {
                if (Physics.Raycast(camera.position, camera.forward, out RaycastHit hitInfo, gunData.maxDistance))
                {
                    Debug.Log(hitInfo.transform.name);
                }
                gunData.currentAmmo--;
                timeSinceLastShot = 0;
                OnGunSchot();
            }
        }
    }
}
```

Figure 5: simple shooting script

When GunData is used as shown in the example above, every different kind of gun you create can be used with this script. You only need to replace the GunData when you want to use a different gun.

Validation

I tested the system with various weapons. Thanks to the different GunData configurations, it's easy to differentiate between weapons during gameplay.

Reflection

I learned how to set up and integrate a good gun system into a project. The setup I chose is easy to understand and reusable. By learning to use scriptable objects, I now know a quick and efficient way to store and apply variables.

In the future, I will use clear names when creating scriptable objects to avoid renaming them later. I don't plan to change the approach, as I first conducted research on what's needed for a gun system and then applied it to the project.

Sources

Literature

- SamMurphy. (n.d.). GitHub - SamMurphy/Unity-Gun-System: Basic FPS mechanics in Unity 5. GitHub
- Tutorial - Make a ScriptableObject-Based Gun System from Scratch. (2022, October 11). Unity Discussions. Link
- Unity Technologies. (n.d.). Unity Manual: ScriptableObject. Unity Docs

Audiovisual

- Rytech. (2022, August 10). **MODULAR WEAPON SYSTEM in Unity in under 4 minutes** YouTube
- Plai. (2022, January 23). **Unity Basic Weapon System Tutorial** YouTube