|  |  |
| --- | --- |
| **项目编号** |  |
| **文档编号** |  |
| **密 级** |  |

**代码清单**

**V1.0**

**广西民族大学**

评审日期： 2021年6月24日

编写人：李荣靖

历史修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 修改编号 | 修改日期 | 修改后版本 | 修改位置 | 修改内容概述 | 审核人 |
| 001 | 2021.6.24 | 1.0 | 全部 | 初始发布版本 | 苏杰平 |
| 002 |  |  |  |  |  |

目录

[1开发平台与工具 4](#_Toc75419235)

[1.1Unity 2018.1.7 4](#_Toc75419236)

[2程序设计 4](#_Toc75419237)

[2.1主要代码 4](#_Toc75419238)

# 1开发平台与工具

## 1.1Unity 2018.1.7

Unity是实时3D互动内容创作和运营平台。包括游戏开发、美术、建筑、汽车设计、影视在内的所有创作者，借助Unity将创意变成现实。Unity平台提供一整套完善的软件解决方案，可用于创作、运营和变现任何实时互动的2D和3D内容，支持平台包括手机、平板电脑、PC、游戏主机、增强现实和虚拟现实设备。

# 2程序设计

## 2.1主要代码

HandgunScriptLPFP.cs

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

// ----- Low Poly FPS Pack Free Version -----

public class HandgunScriptLPFP : MonoBehaviour {

//Animator component attached to weapon

Animator anim;

[Header("Gun Camera")]

//Main gun camera

public Camera gunCamera;

[Header("Gun Camera Options")]

//How fast the camera field of view changes when aiming

[Tooltip("How fast the camera field of view changes when aiming.")]

public float fovSpeed = 15.0f;

//Default camera field of view

[Tooltip("Default value for camera field of view (40 is recommended).")]

public float defaultFov = 40.0f;

public float aimFov = 15.0f;

[Header("UI Weapon Name")]

[Tooltip("Name of the current weapon, shown in the game UI.")]

public string weaponName;

private string storedWeaponName;

[Header("Weapon Sway")]

//Enables weapon sway

[Tooltip("Toggle weapon sway.")]

public bool weaponSway;

public float swayAmount = 0.02f;

public float maxSwayAmount = 0.06f;

public float swaySmoothValue = 4.0f;

private Vector3 initialSwayPosition;

[Header("Weapon Settings")]

public float sliderBackTimer = 1.58f;

private bool hasStartedSliderBack;

//Eanbles auto reloading when out of ammo

[Tooltip("Enables auto reloading when out of ammo.")]

public bool autoReload;

//Delay between shooting last bullet and reloading

public float autoReloadDelay;

//Check if reloading

private bool isReloading;

//Holstering weapon

private bool hasBeenHolstered = false;

//If weapon is holstered

private bool holstered;

//Check if running

private bool isRunning;

//Check if aiming

private bool isAiming;

//Check if walking

private bool isWalking;

//Check if inspecting weapon

private bool isInspecting;

//How much ammo is currently left

private int currentAmmo;

//Totalt amount of ammo

[Tooltip("How much ammo the weapon should have.")]

public int ammo;

//Check if out of ammo

private bool outOfAmmo;

[Header("Bullet Settings")]

//Bullet

[Tooltip("How much force is applied to the bullet when shooting.")]

public float bulletForce = 400;

[Tooltip("How long after reloading that the bullet model becomes visible " +

"again, only used for out of ammo reload aniamtions.")]

public float showBulletInMagDelay = 0.6f;

[Tooltip("The bullet model inside the mag, not used for all weapons.")]

public SkinnedMeshRenderer bulletInMagRenderer;

[Header("Grenade Settings")]

public float grenadeSpawnDelay = 0.35f;

[Header("Muzzleflash Settings")]

public bool randomMuzzleflash = false;

//min should always bee 1

private int minRandomValue = 1;

[Range(2, 25)]

public int maxRandomValue = 5;

private int randomMuzzleflashValue;

public bool enableMuzzleflash = true;

public ParticleSystem muzzleParticles;

public bool enableSparks = true;

public ParticleSystem sparkParticles;

public int minSparkEmission = 1;

public int maxSparkEmission = 7;

[Header("Muzzleflash Light Settings")]

public Light muzzleflashLight;

public float lightDuration = 0.02f;

[Header("Audio Source")]

//Main audio source

public AudioSource mainAudioSource;

//Audio source used for shoot sound

public AudioSource shootAudioSource;

[Header("UI Components")]

public Text timescaleText;

public Text currentWeaponText;

public Text currentAmmoText;

public Text totalAmmoText;

[System.Serializable]

public class prefabs

{

[Header("Prefabs")]

public Transform bulletPrefab;

public Transform casingPrefab;

public Transform grenadePrefab;

}

public prefabs Prefabs;

[System.Serializable]

public class spawnpoints

{

[Header("Spawnpoints")]

//Array holding casing spawn points

//Casing spawn point array

public Transform casingSpawnPoint;

//Bullet prefab spawn from this point

public Transform bulletSpawnPoint;

//Grenade prefab spawn from this point

public Transform grenadeSpawnPoint;

}

public spawnpoints Spawnpoints;

[System.Serializable]

public class soundClips

{

public AudioClip shootSound;

public AudioClip takeOutSound;

public AudioClip holsterSound;

public AudioClip reloadSoundOutOfAmmo;

public AudioClip reloadSoundAmmoLeft;

public AudioClip aimSound;

}

public soundClips SoundClips;

private bool soundHasPlayed = false;

private void Awake ()

{

//Set the animator component

anim = GetComponent<Animator>();

//Set current ammo to total ammo value

currentAmmo = ammo;

muzzleflashLight.enabled = false;

}

private void Start () {

//Save the weapon name

storedWeaponName = weaponName;

//Get weapon name from string to text

currentWeaponText.text = weaponName;

//Set total ammo text from total ammo int

totalAmmoText.text = ammo.ToString();

//Weapon sway

initialSwayPosition = transform.localPosition;

//Set the shoot sound to audio source

shootAudioSource.clip = SoundClips.shootSound;

}

private void LateUpdate () {

//Weapon sway

if (weaponSway == true) {

float movementX = -Input.GetAxis ("Mouse X") \* swayAmount;

float movementY = -Input.GetAxis ("Mouse Y") \* swayAmount;

//Clamp movement to min and max values

movementX = Mathf.Clamp

(movementX, -maxSwayAmount, maxSwayAmount);

movementY = Mathf.Clamp

(movementY, -maxSwayAmount, maxSwayAmount);

//Lerp local pos

Vector3 finalSwayPosition = new Vector3

(movementX, movementY, 0);

transform.localPosition = Vector3.Lerp

(transform.localPosition, finalSwayPosition +

initialSwayPosition, Time.deltaTime \* swaySmoothValue);

}

}

private void Update () {

//Aiming

//Toggle camera FOV when right click is held down

if(Input.GetButton("Fire2") && !isReloading && !isRunning && !isInspecting)

{

gunCamera.fieldOfView = Mathf.Lerp (gunCamera.fieldOfView,

aimFov, fovSpeed \* Time.deltaTime);

isAiming = true;

anim.SetBool ("Aim", true);

if (!soundHasPlayed)

{

mainAudioSource.clip = SoundClips.aimSound;

mainAudioSource.Play ();

soundHasPlayed = true;

}

}

else

{

//When right click is released

gunCamera.fieldOfView = Mathf.Lerp(gunCamera.fieldOfView,

defaultFov,fovSpeed \* Time.deltaTime);

isAiming = false;

anim.SetBool ("Aim", false);

}

//Aiming end

//If randomize muzzleflash is true, genereate random int values

if (randomMuzzleflash == true) {

randomMuzzleflashValue = Random.Range (minRandomValue, maxRandomValue);

}

//Timescale settings

//Change timescale to normal when 1 key is pressed

if (Input.GetKeyDown (KeyCode.Alpha1))

{

Time.timeScale = 1.0f;

timescaleText.text = "1.0";

}

//Change timescale to 50% when 2 key is pressed

if (Input.GetKeyDown (KeyCode.Alpha2))

{

Time.timeScale = 0.5f;

timescaleText.text = "0.5";

}

//Change timescale to 25% when 3 key is pressed

if (Input.GetKeyDown (KeyCode.Alpha3))

{

Time.timeScale = 0.25f;

timescaleText.text = "0.25";

}

//Change timescale to 10% when 4 key is pressed

if (Input.GetKeyDown (KeyCode.Alpha4))

{

Time.timeScale = 0.1f;

timescaleText.text = "0.1";

}

//Pause game when 5 key is pressed

if (Input.GetKeyDown (KeyCode.Alpha5))

{

Time.timeScale = 0.0f;

timescaleText.text = "0.0";

}

//Set current ammo text from ammo int

currentAmmoText.text = currentAmmo.ToString ();

//Continosuly check which animation

//is currently playing

AnimationCheck ();

//Play knife attack 1 animation when Q key is pressed

if (Input.GetKeyDown (KeyCode.Q) && !isInspecting)

{

anim.Play ("Knife Attack 1", 0, 0f);

}

//Play knife attack 2 animation when F key is pressed

if (Input.GetKeyDown (KeyCode.F) && !isInspecting)

{

anim.Play ("Knife Attack 2", 0, 0f);

}

//Throw grenade when pressing G key

if (Input.GetKeyDown (KeyCode.G) && !isInspecting)

{

StartCoroutine (GrenadeSpawnDelay ());

//Play grenade throw animation

anim.Play("GrenadeThrow", 0, 0.0f);

}

//If out of ammo

if (currentAmmo == 0)

{

//Show out of ammo text

currentWeaponText.text = "OUT OF AMMO";

//Toggle bool

outOfAmmo = true;

//Auto reload if true

if (autoReload == true && !isReloading)

{

StartCoroutine (AutoReload ());

}

//Set slider back

anim.SetBool ("Out Of Ammo Slider", true);

//Increase layer weight for blending to slider back pose

anim.SetLayerWeight (1, 1.0f);

}

else

{

//When ammo is full, show weapon name again

currentWeaponText.text = storedWeaponName.ToString ();

//Toggle bool

outOfAmmo = false;

//anim.SetBool ("Out Of Ammo", false);

anim.SetLayerWeight (1, 0.0f);

}

//Shooting

if (Input.GetMouseButtonDown (0) && !outOfAmmo && !isReloading && !isInspecting && !isRunning)

{

anim.Play ("Fire", 0, 0f);

muzzleParticles.Emit (1);

//Remove 1 bullet from ammo

currentAmmo -= 1;

shootAudioSource.clip = SoundClips.shootSound;

shootAudioSource.Play ();

//Light flash start

StartCoroutine(MuzzleFlashLight());

if (!isAiming) //if not aiming

{

anim.Play ("Fire", 0, 0f);

muzzleParticles.Emit (1);

if (enableSparks == true)

{

//Emit random amount of spark particles

sparkParticles.Emit (Random.Range (1, 6));

}

}

else //if aiming

{

anim.Play ("Aim Fire", 0, 0f);

//If random muzzle is false

if (!randomMuzzleflash) {

muzzleParticles.Emit (1);

//If random muzzle is true

}

else if (randomMuzzleflash == true)

{

//Only emit if random value is 1

if (randomMuzzleflashValue == 1)

{

if (enableSparks == true)

{

//Emit random amount of spark particles

sparkParticles.Emit (Random.Range (1, 6));

}

if (enableMuzzleflash == true)

{

muzzleParticles.Emit (1);

//Light flash start

StartCoroutine (MuzzleFlashLight ());

}

}

}

}

//Spawn bullet at bullet spawnpoint

var bullet = (Transform)Instantiate (

Prefabs.bulletPrefab,

Spawnpoints.bulletSpawnPoint.transform.position,

Spawnpoints.bulletSpawnPoint.transform.rotation);

//Add velocity to the bullet

bullet.GetComponent<Rigidbody>().velocity =

bullet.transform.forward \* bulletForce;

//Spawn casing prefab at spawnpoint

Instantiate (Prefabs.casingPrefab,

Spawnpoints.casingSpawnPoint.transform.position,

Spawnpoints.casingSpawnPoint.transform.rotation);

}

//Inspect weapon when pressing T key

if (Input.GetKeyDown (KeyCode.T))

{

anim.SetTrigger ("Inspect");

}

//Toggle weapon holster when pressing E key

if (Input.GetKeyDown (KeyCode.E) && !hasBeenHolstered)

{

holstered = true;

mainAudioSource.clip = SoundClips.holsterSound;

mainAudioSource.Play();

hasBeenHolstered = true;

}

else if (Input.GetKeyDown (KeyCode.E) && hasBeenHolstered)

{

holstered = false;

mainAudioSource.clip = SoundClips.takeOutSound;

mainAudioSource.Play ();

hasBeenHolstered = false;

}

//Holster anim toggle

if (holstered == true)

{

anim.SetBool ("Holster", true);

}

else

{

anim.SetBool ("Holster", false);

}

//Reload

if (Input.GetKeyDown (KeyCode.R) && !isReloading && !isInspecting)

{

//Reload

Reload ();

if (!hasStartedSliderBack)

{

hasStartedSliderBack = true;

StartCoroutine (HandgunSliderBackDelay());

}

}

//Walking when pressing down WASD keys

if (Input.GetKey (KeyCode.W) && !isRunning ||

Input.GetKey (KeyCode.A) && !isRunning ||

Input.GetKey (KeyCode.S) && !isRunning ||

Input.GetKey (KeyCode.D) && !isRunning)

{

anim.SetBool ("Walk", true);

} else {

anim.SetBool ("Walk", false);

}

//Running when pressing down W and Left Shift key

if ((Input.GetKey (KeyCode.W) && Input.GetKey (KeyCode.LeftShift)))

{

isRunning = true;

} else {

isRunning = false;

}

//Run anim toggle

if (isRunning == true) {

anim.SetBool ("Run", true);

} else {

anim.SetBool ("Run", false);

}

}

private IEnumerator HandgunSliderBackDelay () {

//Wait set amount of time

yield return new WaitForSeconds (sliderBackTimer);

//Set slider back

anim.SetBool ("Out Of Ammo Slider", false);

//Increase layer weight for blending to slider back pose

anim.SetLayerWeight (1, 0.0f);

hasStartedSliderBack = false;

}

private IEnumerator GrenadeSpawnDelay () {

//Wait for set amount of time before spawning grenade

yield return new WaitForSeconds (grenadeSpawnDelay);

//Spawn grenade prefab at spawnpoint

Instantiate(Prefabs.grenadePrefab,

Spawnpoints.grenadeSpawnPoint.transform.position,

Spawnpoints.grenadeSpawnPoint.transform.rotation);

}

private IEnumerator AutoReload () {

if (!hasStartedSliderBack)

{

hasStartedSliderBack = true;

StartCoroutine (HandgunSliderBackDelay());

}

//Wait for set amount of time

yield return new WaitForSeconds (autoReloadDelay);

if (outOfAmmo == true) {

//Play diff anim if out of ammo

anim.Play ("Reload Out Of Ammo", 0, 0f);

mainAudioSource.clip = SoundClips.reloadSoundOutOfAmmo;

mainAudioSource.Play ();

//If out of ammo, hide the bullet renderer in the mag

//Do not show if bullet renderer is not assigned in inspector

if (bulletInMagRenderer != null)

{

bulletInMagRenderer.GetComponent

<SkinnedMeshRenderer> ().enabled = false;

//Start show bullet delay

StartCoroutine (ShowBulletInMag ());

}

}

//Restore ammo when reloading

currentAmmo = ammo;

outOfAmmo = false;

}

//Reload

private void Reload () {

if (outOfAmmo == true)

{

//Play diff anim if out of ammo

anim.Play ("Reload Out Of Ammo", 0, 0f);

mainAudioSource.clip = SoundClips.reloadSoundOutOfAmmo;

mainAudioSource.Play ();

//If out of ammo, hide the bullet renderer in the mag

//Do not show if bullet renderer is not assigned in inspector

if (bulletInMagRenderer != null)

{

bulletInMagRenderer.GetComponent

<SkinnedMeshRenderer> ().enabled = false;

//Start show bullet delay

StartCoroutine (ShowBulletInMag ());

}

}

else

{

//Play diff anim if ammo left

anim.Play ("Reload Ammo Left", 0, 0f);

mainAudioSource.clip = SoundClips.reloadSoundAmmoLeft;

mainAudioSource.Play ();

//If reloading when ammo left, show bullet in mag

//Do not show if bullet renderer is not assigned in inspector

if (bulletInMagRenderer != null)

{

bulletInMagRenderer.GetComponent

<SkinnedMeshRenderer> ().enabled = true;

}

}

//Restore ammo when reloading

currentAmmo = ammo;

outOfAmmo = false;

}

//Enable bullet in mag renderer after set amount of time

private IEnumerator ShowBulletInMag () {

//Wait set amount of time before showing bullet in mag

yield return new WaitForSeconds (showBulletInMagDelay);

bulletInMagRenderer.GetComponent<SkinnedMeshRenderer> ().enabled = true;

}

//Show light when shooting, then disable after set amount of time

private IEnumerator MuzzleFlashLight ()

{

muzzleflashLight.enabled = true;

yield return new WaitForSeconds (lightDuration);

muzzleflashLight.enabled = false;

}

//Check current animation playing

private void AnimationCheck ()

{

//Check if reloading

//Check both animations

if (anim.GetCurrentAnimatorStateInfo (0).IsName ("Reload Out Of Ammo") ||

anim.GetCurrentAnimatorStateInfo (0).IsName ("Reload Ammo Left"))

{

isReloading = true;

}

else

{

isReloading = false;

}

//Check if inspecting weapon

if (anim.GetCurrentAnimatorStateInfo (0).IsName ("Inspect"))

{

isInspecting = true;

}

else

{

isInspecting = false;

}

}

}

// ----- Low Poly FPS Pack Free Version -----

Welcome.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Welcome : MonoBehaviour {

public void LoadGame()

{

Application.LoadLevel("Main");

}

// Use this for initialization

//void Start () {

//}

// Update is called once per frame

//void Update () {

//}

}

HideMouse.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class HideMouse : MonoBehaviour {

// Use this for initialization

void Start () {

Cursor.lockState = CursorLockMode.Locked;//隐藏鼠标指针

Cursor.visible = false;

}

// Update is called once per frame

void Update () {

}

}