Shooting Script

```
//Gun stats
public int damage;
public float timeBetweenShooting, spread, range, reloadTime, timeBetweenShots;
public int magazineSize, bulletsPerTap;
public bool allowButtonHold;
int bulletsLeft, bulletsShot;

//bools
bool shooting, readyToShoot, reloading;

//Reference
public Camera fpsCam;
public Transform attackPoint;
public RaycastHit rayHit;
public LayerMask whatIsEnemy;
public AudioSource shootsound;
public Animation ReloadAnimation;

//Graphics
public GameObject muzzleFlash;
// public bulletHoleGraphic;
public TextMeshProUGUI text;

private void Awake()
{
    bulletsLeft = magazineSize;
    readyToShoot = true;
}

private void Update()
{
    MyInput();

    //SetText
    text.SetText(bulletsLeft + " / " + magazineSize);

    if (bulletsLeft > 0)
    {
        //Anim
        if (Input.GetButtonDown("Fire1"))
        {
            GetComponent<Animator>().SetTrigger("Shooting");
        }
        //Sound
        if (Input.GetButtonDown("Fire1"))
        {
            shootsound.Play();
        }
    }
```

```
        //RayCast
        if (Physics.Raycast(fpsCam.transform.position, direction, out rayHit, range, whatIsEnemy))
        {
            //Debug.Log(rayHit.collider.name);

            if (rayHit.collider.CompareTag("Enemy"))
                rayHit.collider.GetComponent<ShootingAi>().TakeDamage(damage);
            else if (rayHit.collider.CompareTag("ExplosiveBarrel"))
            {
                rayHit.collider.GetComponent<ExplosiveBarrelScript>().explode = true;
            }
        }

        muzzleFlash.GetComponent<ParticleSystem>().Play();
        bulletsLeft--;
        bulletsShot--;

        Invoke("ResetShot", timeBetweenShooting);

        if(bulletsShot > 0 && bulletsLeft > 0)
        Invoke("Shoot", timeBetweenShots);
    }
    private void ResetShot()
    {
        readyToShoot = true;
    }
    private void Reload()
    {
        GetComponent<Animator>().SetBool("Reloading", true);
        reloading = true;
        Invoke("ReloadFinished", reloadTime);
    }
    private void ReloadFinished()
    {
        GetComponent<Animator>().SetBool("Reloading", false); ;
        bulletsLeft = magazineSize;
        reloading = false;
```

Player Tracker script

```
public class PlayerTracker : MonoBehaviour
{
    public Transform trackedObject;
    public float maxDistance = 10;
    public float moveSpeed = 5;
    public float updateSpeed = 10;
    [Range(0, 10)]
    public float currentDistance = 5;
    private string moveAxis = "Mouse ScrollWheel";
    private GameObject ahead;
    private MeshRenderer _renderer;
    public float hideDistance = 1.5f;

    void Start()
    {
        ahead = new GameObject("ahead");
        _renderer = trackedObject.gameObject.GetComponent<MeshRenderer>();
    }

    // Update is called once per frame
    void LateUpdate()
    {
        ahead.transform.position = trackedObject.position + trackedObject.forward * (maxDistance * 0.25f);
        currentDistance += Input.GetAxisRaw(moveAxis) * moveSpeed * Time.deltaTime;
        currentDistance = Mathf.Clamp(currentDistance, 0, maxDistance);
        transform.position = Vector3.MoveTowards(transform.position, trackedObject.position + Vector3.up * currentDistance - trackedObject.forward * (currentDistance + maxDistance *
        transform.LookAt(ahead.transform);
        _renderer.enabled = (currentDistance > hideDistance);
```

Player Movement Script

```
public float speed = 12f;
public float gravity = -9.81f;
public float jumpHeight = 3f;

public Transform groundCheck;
public float groundDistance = 0.4f;
public LayerMask groundMask;

Vector3 velocity;
bool isGrounded;

void Start()
{

}

// Update is called once per frame
void Update()
{

    isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

    if(isGrounded && velocity.y < 0)
    {
        velocity.y = -2f;
    }

    float x = Input.GetAxis("Horizontal");
    float z = Input.GetAxis("Vertical");

    Vector3 move = transform.right * x + transform.forward * z;

    controller.Move(move * speed * Time.deltaTime);

    if(Input.GetButtonDown("Jump") && isGrounded)
    {
        velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
    }

    velocity.y += gravity * Time.deltaTime;

    controller.Move(velocity * Time.deltaTime);
}
```

```
public float mouseSensitivity = 100f;
public Transform playerBody;
float xRotation = 0f;

void Start()
{
    Cursor.lockState = CursorLockMode.Locked;
}

// Update is called once per frame
void Update()
{
    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

    xRotation -= mouseY;
    xRotation = Mathf.Clamp(xRotation, -90f, 90f);

    transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
    playerBody.Rotate(Vector3.up * mouseX);

}
```

Pause Menu

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

public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void BackToMenu()
    {
        Debug.Log("Menu");
        SceneManager.LoadScene(0);
    }

    public void QuitGame()
    {
        Debug.Log("QUIT");
        Application.Quit();
    }
}
```

Menu Script


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


public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void QuitGame()
    {
        //Debug.Log("QUIT");
        Application.Quit();
    }
}
```

Some of ingame screens:

Other movement script

```
public class MovementScript : MonoBehaviour
{

    //Assingables
    public Transform playerCam;
    public Transform orientation;

    //Other
    private Rigidbody rb;

    //Rotation and look
    private float xRotation;
    private float sensitivity = 50f;
    private float sensMultiplier = 1f;

    //Movement
    public float moveSpeed = 4500;
    public float maxSpeed = 20;
    public bool grounded;
    public LayerMask whatIsGround;

    public float counterMovement = 0.175f;
    private float threshold = 0.01f;
    public float maxSlopeAngle = 35f;

    //Crouch & Slide
    private Vector3 crouchScale = new Vector3(1, 0.5f, 1);
    private Vector3 playerScale;
    public float slideForce = 400;
    public float slideCounterMovement = 0.2f;

    //Jumping
    private bool readyToJump = true;
    private float jumpCooldown = 0.25f;
    public float jumpForce = 550f;

    //Input
    float x, y;
    bool jumping, sprinting, crouching;

    //Sliding
    private Vector3 normalVector = Vector3.up;
    private Vector3 wallNormalVector;

    void Awake()
    {
        rb = GetComponent<Rigidbody>();
    }

    void Start()
    {
        playerScale = transform.localScale;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }


    private void FixedUpdate()
    {
        Movement();
    }

    private void Update()
    {
        MyInput();
        Look();
    }
```

```
private void MyInput()
{
    x = Input.GetAxisRaw("Horizontal");
    y = Input.GetAxisRaw("Vertical");
    jumping = Input.GetButton("Jump");
    crouching = Input.GetKey(KeyCode.LeftControl);

    //Crouching
    if (Input.GetKeyDown(KeyCode.LeftControl))
        StartCrouch();
    if (Input.GetKeyUp(KeyCode.LeftControl))
        StopCrouch();
}

private void StartCrouch()
{
    transform.localScale = crouchScale;
    transform.position = new Vector3(transform.position.x, transform.position.y - 0.5f, transform.position.z);
    if (rb.velocity.magnitude > 0.5f)
    {
        if (grounded)
        {
            rb.AddForce(orientation.transform.forward * slideForce);
        }
    }
}

private void StopCrouch()
{
    transform.localScale = playerScale;
    transform.position = new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z);
}

private void Movement()
{
    //Extra gravity
    rb.AddForce(Vector3.down * Time.deltaTime * 10);

    //Find actual velocity relative to where player is looking
    Vector2 mag = FindVelRelativeToLook();
    float xMag = mag.x, yMag = mag.y;

    //Counteract sliding and sloppy movement
    CounterMovement(x, y, mag);

    //If holding jump && ready to jump, then jump
    if (readyToJump && jumping) Jump();

    //Set max speed
    float maxSpeed = this.maxSpeed;

    //If sliding down a ramp, add force down so player stays grounded and also builds speed
    if (crouching && grounded && readyToJump)
    {
        rb.AddForce(Vector3.down * Time.deltaTime * 3000);
        return;
    }

    //If speed is larger than maxspeed, cancel out the input so you don't go over max speed
    if (x > 0 && xMag > maxSpeed) x = 0;
    if (x < 0 && xMag < -maxSpeed) x = 0;
    if (y > 0 && yMag > maxSpeed) y = 0;
    if (y < 0 && yMag < -maxSpeed) y = 0;

    //Some multipliers
    float multiplier = 1f, multiplierV = 1f;

    // Movement in air
    if (!grounded)
    {
        multiplier = 0.5f;
        multiplierV = 0.5f;
    }
```

AI walking script

```
public class WalkToPoint : MonoBehaviour
{
    [SerializeField] private NavMeshAgent _agent;
    [SerializeField] private List<GameObject> _destination;
    [SerializeField] private List<float> _wait;
    //[SerializeField] private Animator _animator;

    int i = 0;

    private bool checkLocation(Vector3 objLocation, Vector3 pointLocation)
    {
        if (objLocation.x == pointLocation.x && objLocation.z == pointLocation.z)
        {
            return true;
        }
        else { return false; }
    }

    private IEnumerator mustWait()
    {
        _agent.isStopped = true;
        //_animator.SetBool("isWalking", false);
        yield return new WaitForSeconds(_wait[i]);
        _agent.isStopped = false;
        //_animator.SetBool("isWalking", true);
    }

    void Start()
    {
        _agent = GetComponent<NavMeshAgent>();
        //_animator = GetComponent<Animator>();
    }

    void Update()
    {
        _agent.SetDestination(_destination[i].transform.position);

        if (checkLocation(_agent.transform.position, _destination[i].transform.position))
        {
            if (_wait[i] > 0) StartCoroutine(mustWait());

            i++;

            if (i == _destination.Count)
            {
                i = 0;
                _agent.transform.position = _destination[i].transform.position;
            }
        }
    }
```