Laporan Progres Chararter Movement

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

public class CharacterMovement : MonoBehaviour
{
    public Rigidbody2D body;
    public float Speed;
    public Animator anim;
    private BoxCollider2D boxCollider;
    [SerializeField] private LayerMask LayerGround;
    [SerializeField] private LayerMask LayerWall;
    private float WallJumpCooldown;
    private float CharacterInput;

    public float coyoteTimejump = 0.2f;
    private float timeleft;
    private bool coyoteActive;

    public float power = 10f;
    private float powerbuffer = 0.1f;
    private float powerbeingpressed;

    public float fallingspeed = -10f;
    public float JumpModifier = 0.5f;

    public float maxJumpHoldTime = 0.2f;
    private float jumpHoldTimer;
    private bool isJumping;

    private bool canWallJump = true;

    private void Awake()
    {
        body = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        boxCollider = GetComponent<BoxCollider2D>();
    }

    private void Update()
    {
        CharacterInput = Input.GetAxis("Horizontal");
```

```
        body.velocity = new Vector2(CharacterInput * Speed, body.velocity.y);

        if (CharacterInput > 0.01f)
            transform.localScale = Vector3.one;
        else if (CharacterInput < -0.01f)
            transform.localScale = new Vector3(-1, 1, 1);

        anim.SetBool("run", CharacterInput != 0);

        if (isJumping && Input.GetKey(KeyCode.Space) && jumpHoldTimer <
maxJumpHoldTime)
        {
            jumpHoldTimer += Time.deltaTime;
        }

        if (WallJumpCooldown > 0.2f)
        {
            if (ifGeokionThewall() && ifGeokiGrounded())
            {
                body.gravityScale = 0;
                body.velocity = Vector2.zero;
            }
            else
                body.gravityScale = 1;

            if (Input.GetKeyDown(KeyCode.Space) && (ifGeokiGrounded() ||
ifGeokionThewall()))
            {
                powerbeingpressed = Time.time;
                jumpHoldTimer = 0f;
                Jump();
                isJumping = true;
            }
        }
        else WallJumpCooldown += Time.deltaTime;

        anim.SetBool("grounded", ifGeokiGrounded());

        if (ifGeokiGrounded())
        {
            timeleft = Time.time;
            coyoteActive = true;
        }
        else if (Time.time - timeleft > coyoteTimejump)
```

```
        {
            coyoteActive = false;
        }
    }

    private void FixedUpdate()
    {
        if (body.velocity.y < fallingspeed)
        {
            body.velocity = new Vector2(body.velocity.x, fallingspeed);
        }

        if (isJumping && body.velocity.y > 0 && (!Input.GetKey(KeyCode.Space) ||
jumpHoldTimer >= maxJumpHoldTime))
        {
            body.velocity = new Vector2(body.velocity.x, body.velocity.y * JumpModifier);
        }
    }

    private bool Jumpingwooosh()
    {
        bool bufferJump = Time.time - powerbeingpressed <= powerbuffer;
        bool Coyotetime = coyoteActive && !ifGeokiGrounded();

        return bufferJump || Coyotetime;
    }

    private void Jump()
    {
        if (ifGeokiGrounded() || Jumpingwooosh())
        {
            body.velocity = new Vector2(body.velocity.x, power);
            anim.SetTrigger("jump");
            isJumping = true;
            coyoteActive = false;
        }
        else if (ifGeokionThewall() && !ifGeokiGrounded())
        {
            if (canWallJump)
            {

                if (CharacterInput == 0)
                {
                    body.velocity = new Vector2(-Mathf.Sign(transform.localScale.x) * 10, 0);
```

```
                transform.localScale = new Vector3(-Mathf.Sign(transform.localScale.x),
transform.localScale.y, transform.localScale.z);
            }
            else
                body.velocity = new Vector2(-Mathf.Sign(transform.localScale.x) * 3, 6);

            WallJumpCooldown = 0;
            canWallJump = false;
        }
    }
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Ground"))
    {
        isJumping = false;
        anim.ResetTrigger("jump");
        canWallJump = true;
    }
}

private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Ground"))
    {
        isJumping = true;
    }
}

private bool ifGeokiGrounded()
{
    RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,
boxCollider.bounds.size, 0, Vector2.down, 0.1f, LayerGround);
    return raycastHit.collider != null;
}

private bool ifGeokionThewall()
{
    RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,
boxCollider.bounds.size, 0, new Vector2(transform.localScale.x, 0), 0.1f, LayerWall);
    return raycastHit.collider != null;
}
}
```

Dalam game platformer, tambahan mekanik ini meningkatkan responsivitas dan kontrol karakter dalam gameplay.
Berikut penjelasan rinci:

Clamp Fall: Mekanik ini memperlambat kecepatan jatuh karakter, membuat pergerakan vertikal terasa lebih halus dan terkendali. Ini bisa dicapai dengan menetapkan batas maksimum kecepatan jatuh atau menambah resistensi pada jatuh yang diatur oleh kode. Dengan begitu, karakter akan lebih lama melayang saat jatuh, terutama pada jarak pendek, sehingga pemain bisa lebih mudah mengontrol pendaratan.

Coyote Jump: Fitur ini memungkinkan pemain melakukan lompatan meskipun karakter sudah sedikit meninggalkan ujung platform. Biasanya diimplementasikan dengan memberikan "toleransi waktu" sesaat setelah karakter meninggalkan platform, yang memungkinkan input lompatan tetap terdeteksi. Ini memberi pemain sedikit ruang untuk kesalahan dan membuat game terasa lebih responsif, terutama untuk lompatlompat antar platform yang sempit.

Jump Buffer: Dengan jump buffer, saat pemain menekan atau menahan tombol lompat (misalnya spasi), game mencatat input ini selama beberapa milidetik, bahkan jika karakter masih berada di darat. Ini berarti bahwa jika pemain menekan tombol lompat sedikit lebih awal, karakter tetap akan melompat segera setelah menyentuh tanah. Mekanik ini juga memperhitungkan durasi penekanan tombol untuk mengatur tinggi lompatan—semakin lama tombol ditahan, semakin tinggi karakter melompat.

Wall Jump: Wall jump memungkinkan karakter untuk melompat dari dinding ketika berada dalam posisi menempel di dinding atau dalam kontak dengannya, meski tidak berada di tanah. Mekanik ini ideal untuk platforming vertikal atau saat pemain perlu mencapai area yang sulit dijangkau. Dalam skrip, wall jump diatur agar karakter bisa menendang dinding dan melompat ke arah berlawanan dengan daya dorong tertentu, tergantung arah input pemain. Jika tidak ada input, karakter akan melompat jauh dari dinding, memungkinkan pemain untuk mengatur ulang posisi. Ini meningkatkan fleksibilitas dan dinamika kontrol pemain dalam bergerak dan melompat, terutama di area platform yang kompleks. Wall jump juga memberikan variasi pada gameplay, memerlukan ketepatan timing dan respons cepat dari pemain untuk memanfaatkan mekanik ini secara efektif.