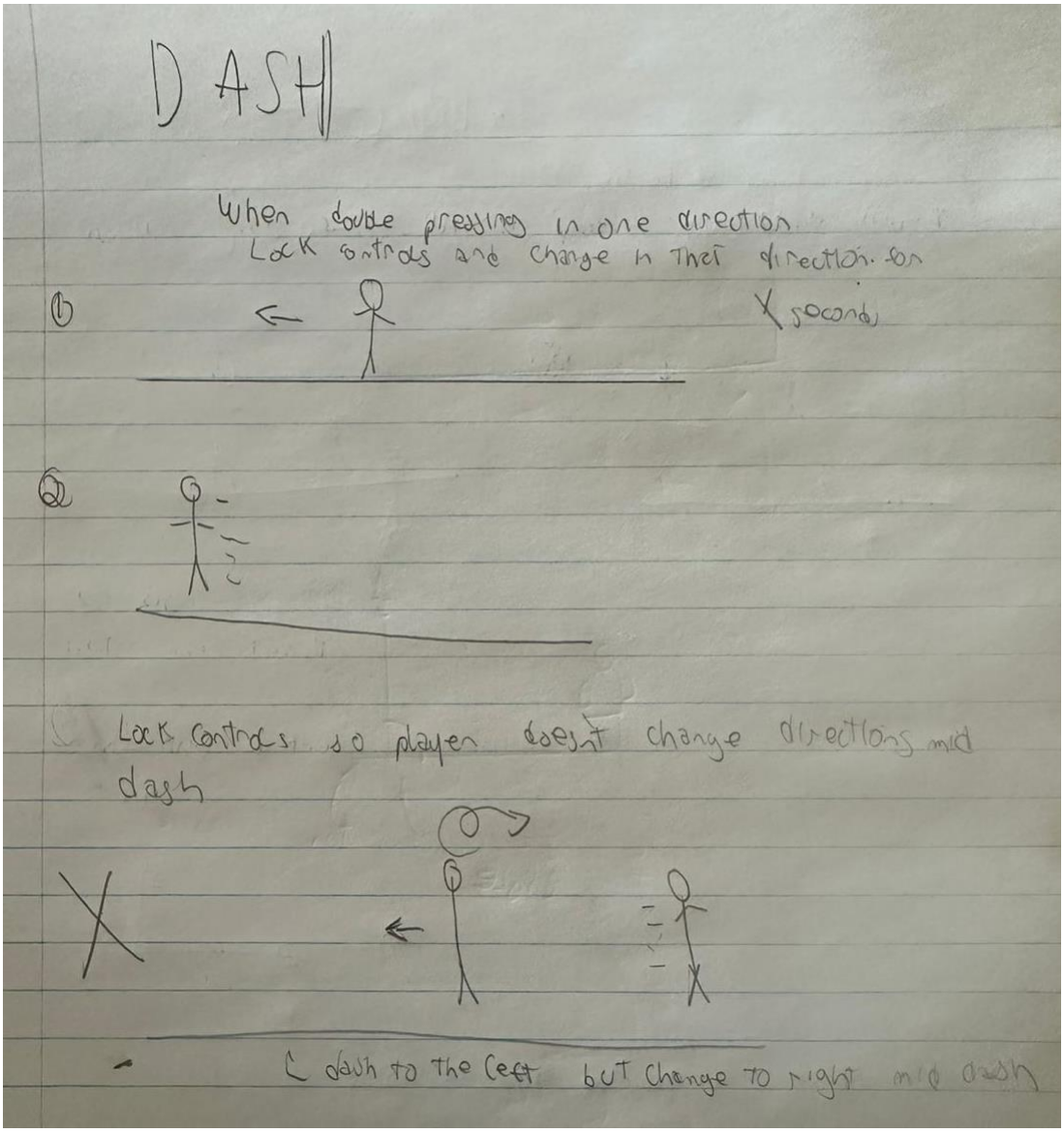
Assignment 2 journal

Task 1 dash (1 hour)

Planning

For the dash the player will dash in the direction they are walking towards for an detemined amount of time by a variable. When the player taps the walk button twice in a short time frame they will dash. When dashing the player's controler is locked and they cannot move, so that they wont be able to change directions mid dash. When the dash is over it will start a cooldown in which the player wont be able to dash until over. When the dash is over the player should be able to dash again



Batch 1(30 min)

I first stared setting up the new state for the player to apply the animation when the player dashes

New state for dashing :

```
private readonly int WalkingHash = Animator.StringToHash("Walking");
private readonly int IdleHash = Animator.StringToHash("Idle");
private readonly int DeadHash = Animator.StringToHash("Dead");
private readonly int jumpingHash = Animator.StringToHash("Jumping");
private readonly int dashHash = Animator.StringToHash("Dashing");
```

```
1 reference
private void UpdateVisuals()
{
    if (playerController.previousState != playerController.currentState)
    {
        switch (playerController.currentState)
        {
            case PlayerController.PlayerState.idle:
                animator.CrossFade(IdleHash,0);
                break;
            case PlayerController.PlayerState.walking:
                animator.CrossFade(WalkingHash, 0);
                break;
            case PlayerController.PlayerState.jumping:
                animator.CrossFade(jumpingHash, 0);
                break;
            case PlayerController.PlayerState.dead:
                animator.CrossFade(DeadHash, 0);
                break;
            case PlayerController.PlayerState.dahsing:
                animator.CrossFade(dashHash, 0);
                break;
            default:
                break;
        }

    }
}
}
```

After it I started setting up the variables

```
[Header("Dash")]
[SerializeField] private bool IsDashing;
[SerializeField] private float dashMultiplier;
[SerializeField] private float DashTimer;
[SerializeField] private float DashMaxTimer;
[SerializeField] private float dashDuration;
```

A Boolean for dashing to check if the player is dashing to disable their input so the player can dash without modifying movement midway trough, a timer to check if the player pressed the button twice in the time window,

So I stated by attempting by creating a code that when the player pressed the movement button twice in a short time window it would disable the player input and dash in the direction the player is facing

```
//Dash
1 reference
private void GetDashInput()
{
    //get input
    if (Input.GetButtonDown("Horizontal"))
    {
        Debug.Log("got into dash timer");
        //start timer
        DashTimer += Time.deltaTime;
        if (DashTimer >= DashMaxTimer) // if during window
        {
            //check if pressed the button again
            if (Input.GetButtonDown("Horixaontal"))
            {
                IsDashing = true;

            }

        }
        else // set it back to 0
        {
            DashTimer = 0;
        }
    }
```

The first function gets the player input

```
1 reference
private void applyDash()
{
    if(IsDashing)//check if dashing
    {
        dashDuration += Time.deltaTime; // increase timer
        if (currentDirection == FacingDirection.right)
        {
            if (dashDuration < 1) // check if timer is done
            {
                velocity.x += accelerationRate * dashMultiplier * Time.deltaTime;
                Debug.Log("right");
            }
            else // if timer is done set dashing off
            {
                dashDuration = 0;
                IsDashing = false;
            }
        }
         else if (currentDirection == FacingDirection.left)
        {
            if (dashDuration < 1)// check if timer is done
            {
                velocity.x += accelerationRate * (-1 * dashMultiplier) * Time.deltaTime;
                Debug.Log("left");
            }
            else // if timer is done set dashing off
            {
                dashDuration = 0;
                IsDashing = false;
            }
        }
    }
}


}
```

If is dashing, check the facing direction and apply a timer and apply the dash during the timer, so I the dash would last a second

After attempting it I realized I would not be able to make it work by pressing it twice so I had the idea to swap it to when the player pressed shift, they would dash

Batch 2 (30 minutes)

I deleted both variables to check for timer between presses, I do know they might come in handy later when adding a cooldown for the dash I will first focus on making the dash functional

```
73
74      [Header("Dash")]
75      [SerializeField] private bool IsDashing;
76      [SerializeField] private float dashMultiplier;
77      [SerializeField] private float dashDuration;
78
```

The new code now uses shift to dash

```
//Dash
1 reference
private void GetDashInput()
{
    //get input
    if (Input.GetKeyDown(KeyCode.LeftShift))
    {
        Debug.Log("got into dash timer");
        IsDashing = true;


    }
}
```

When the player dashes the controls are locked and the player dashes for a determined amount of seconds, when the time ends the player is able to move the player in the direction they want again

```
157
158         if(!IsDashing)
159         {
160             MovementUpdate(playerInput);
161             jumpUpdate();
162         }
```

Now I went to work on the cooldown

The plan is after the player dashes they enter a cooldown for a determined amount of seconds before being the dash again

For the new code I added this timer for the cooldown

```
2 references
private void DashCooldown()
{
    Debug.Log("fell here");
    dashTime += Time.deltaTime;
    if (dashTime >= dashTimeMax && canDash == false)
    {
        canDash = true;
        dashTime = 0;
    }
    else
    {
        canDash = false;
    }
}

}
```

Now when the player dashes they call the dashcooldown function

```
        else if (currentDirection == FacingDirection.left)
        {
            if (dashTime < dashTimeMax)// check if timer is done
            {
                velocity.x += accelerationRate * (-1 * dashMultiplier) * Time.deltaTime;
                Debug.Log("left");
                DashCooldown();
            }
            else // if timer is done set dashing off
            {
                dashTime = 0;
                IsDashing = false;
            }
        }
    }
```

The cooldown was not working correctly.

I realized I was suing dashTime instead of dash cooldown, the new updated code below

```
    2 references
    private void DashCooldown()
    {
        Debug.Log("fell here");
        cooldown += Time.deltaTime;
        if (cooldown >= dashTimeMax && canDash == false)
        {
            canDash  Tab to go to next
            cooldown = 0;
        }
        else
        {
            canDash = false;
        }
    }

}
```

The code works better now but still presents flaws.

I realized that by calling the variable only in the run function it stops the variable to finish its cooldown timer.

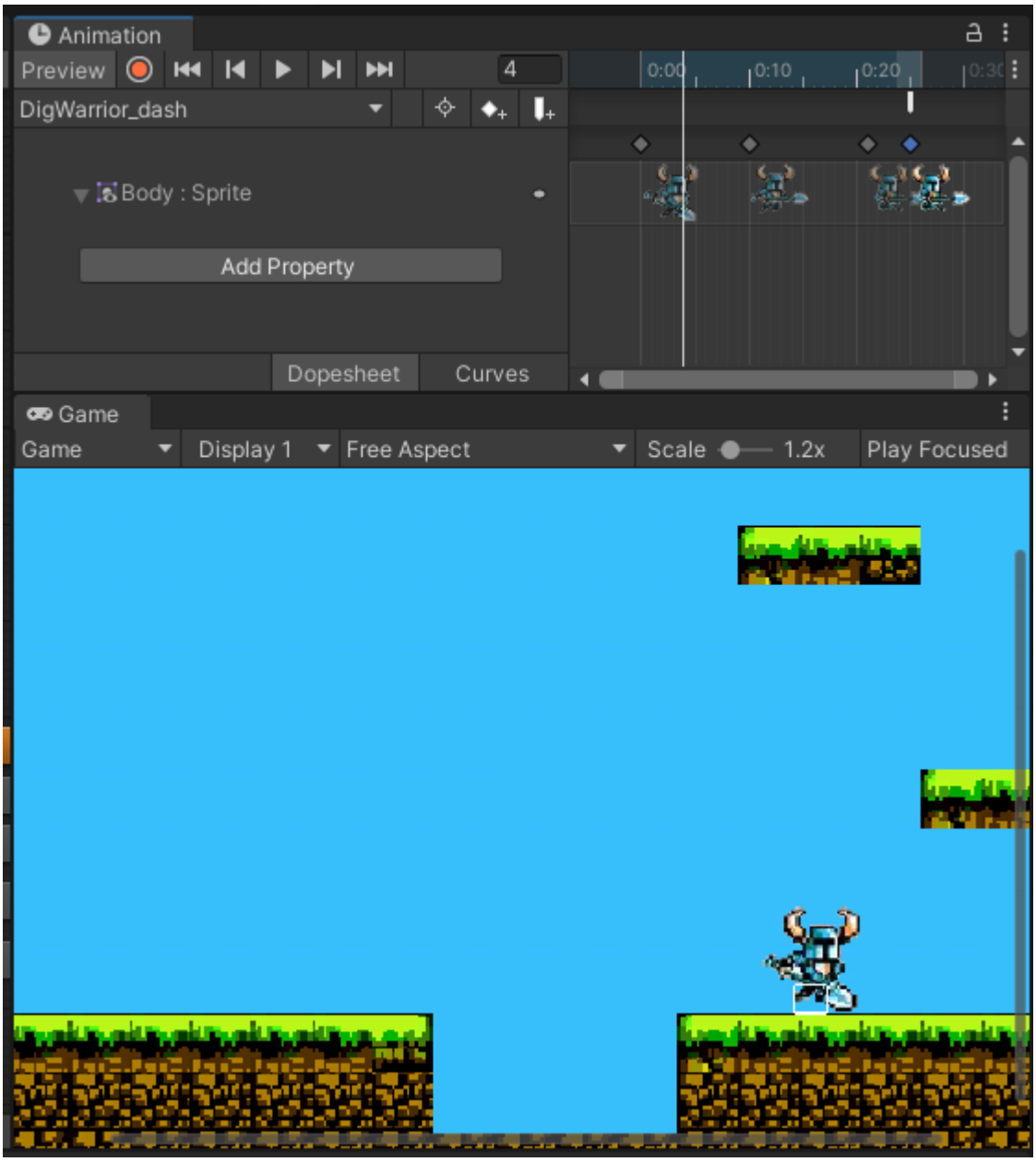
I moved the code to updated and set the can dash after dashing false

```
        else if (currentDirection == FacingDirection.left)
        {
            if (dashTime < dashTimeMax)// check if timer is done
            {
                velocity.x += accelerationRate * (-1 * dashMultiplier) * Time.deltaTime;
                Debug.Log("left");
                canDash = false;
```

```
    previousState = currentState;
    if (!canDash)
    {
        DashCooldown();
    }
    if (canDash)
    {
        GetDashInput();
    }
    if(IsDashing)
    {
        applyDash();
    }
```

Dash works perfectly with cooldown

The animations weren't working as intended but after changing the property of the animation it started to work



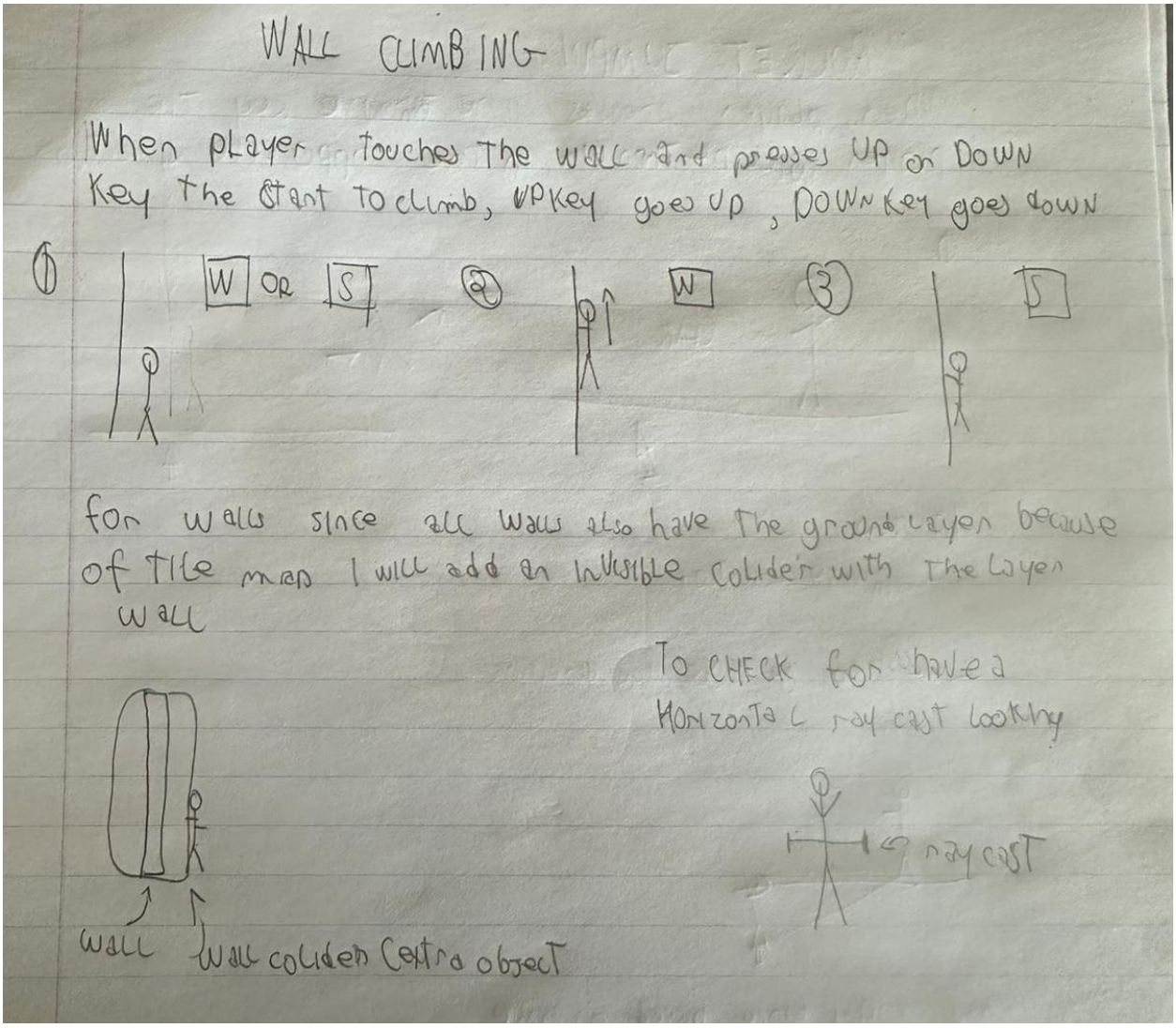
Task 1 complete (1 hour)

Task 2 wall climbing (53 minutes)

Planning

The player when touching a wall they will be able to press the up or down key to initiate wall climbing. When wall climbing the player can press the UP key to go up and the DOWN key to go down, while climbing the gravity is disabled. The tile map includes the walls so that means that the

walls are already part of the ground layer, to work around this I will add an extra invisible object with a collider that I will put in the wall layer so that the player will be able to walk up walls. The player will check for walls using a horizontal raycast that will look for the wall layer



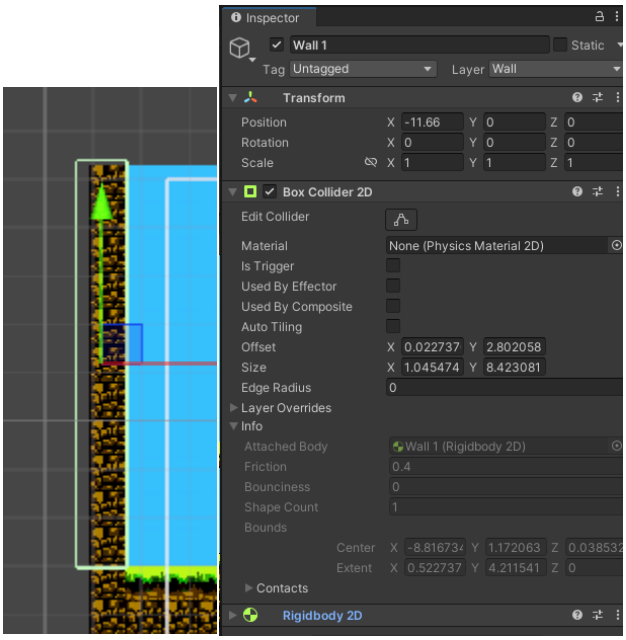
Batch 3 (30 minutes)

The first step to making this mechanic will be creating horizontal raycast or boxcast so that it checks when the player is touching the wall.

I chose to use a overlap box for the code, the player will only start climbing the wall when they press the key to start going up or down, otherwise they would instantly stick to walls mid fall

```
1 reference
private void DetectWall()
{
    if(Physics2D.OverlapBox(transform.position, boxSize,0,wallCheckerLayerMask))
    {
        if(Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.S))
        {
            isClimbing = true;

        }
    }
    else
    {
        isClimbing = false;
    }
}
```

To detect the wall the player will need to be against the wall and the wall have a wall layer attached to it, since the game uses a tile system, and all of the tiles are in the layer ground I created invisible objects with colliders on the walls to layer them as walls



For the code to climb the wall itself I check if the player is climbing the wall and if the player is disabling gravity and change their Y speed based on input

```
1 reference
private void climbWall(Vector2 playerInput)
{
    if(isClimbing)
    {
        velocity.y += climbSpeed * playerInput.y * Time.deltaTime;


    }

}

}
```

Disabling gravity

```
body.velocity = velocity;
if (!isClimbing)
{
    if (!isGrounded)
    {
        velocity.y += gravity * Time.deltaTime;
    }
}
else...

}
```

Getting input

```
Vector2 playerInput = new Vector2();
Vector2 playerInputY = new Vector2();
playerInputY.y = Input.GetAxisRaw("Vertical");
playerInput.x = Input.GetAxisRaw("Horizontal");
DetectWall();
climbWall(playerInputY);
```

Even with the player being able to detect walls they were not yet able to climb them

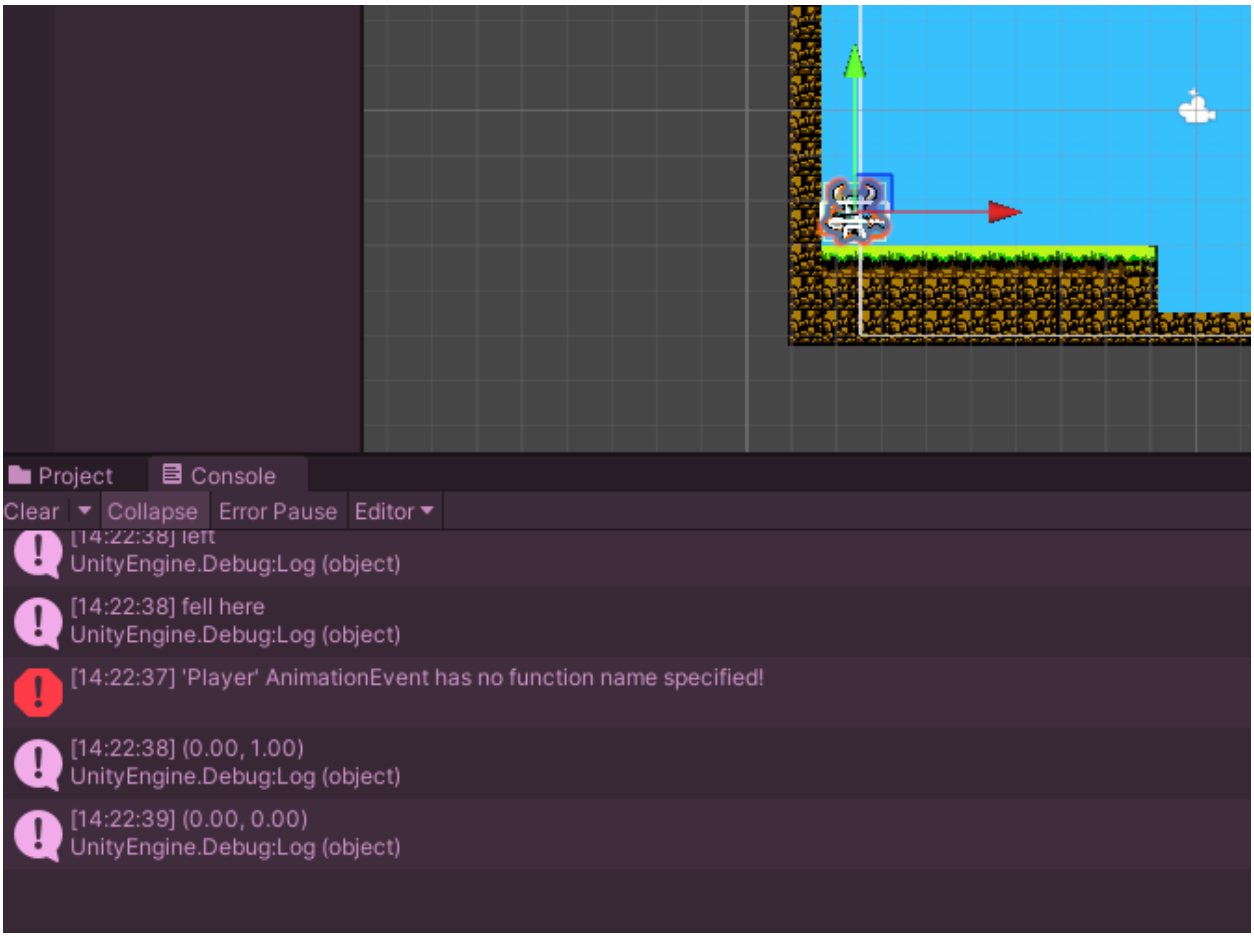
I encountered a bug that when the player would jump, they would not be able to walk anymore, I realized the bug was located in the area in which I disable gravity by checking if the player is climbing. Since when I'm checking if the player is not climbing and then if they grounded this would enable gravity and never disable it again, setting the player velocity to always be 0 on X

Batch 4 (23 minutes)

To fix this I set that check when the player is climbing instead of when they're not climbing

```
body.velocity = velocity;

if(isClimbing)
{
    velocity.y = 0;
}

else if (!isGrounded)// in air
{
   velocity.y += gravity * Time.deltaTime
}

else
{
    velocity.y = 0;
}
```

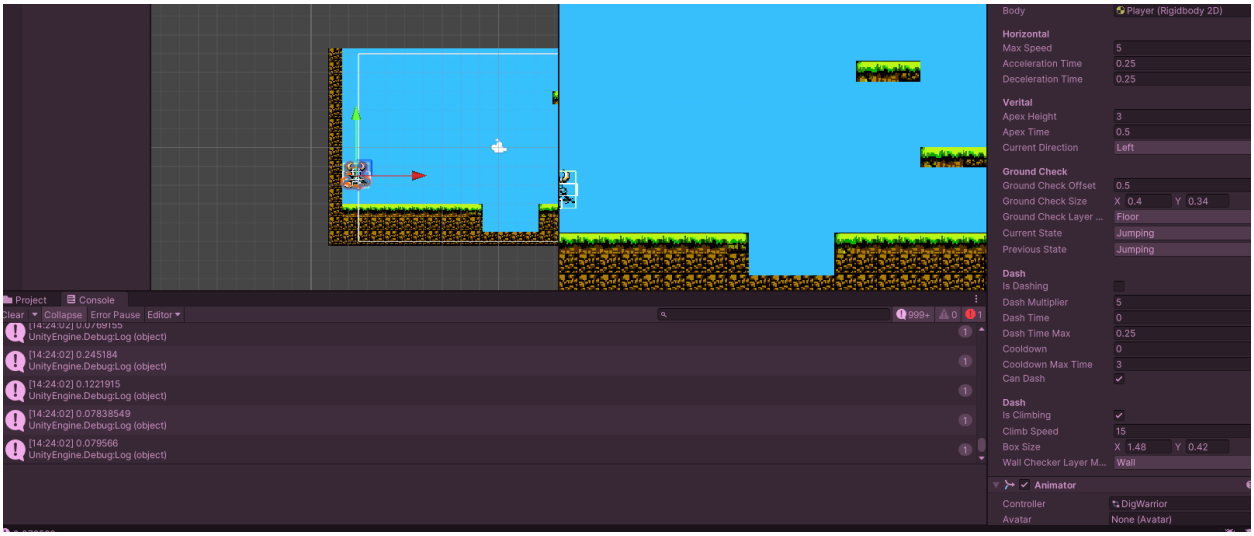
This fixed the bug but the player would still not walk up the walls



The player was getting Y input but was not climbing

After checking the velocity, I realized the climb speed was 0 so that why it was not climbing

Player is not correctly walking up walls



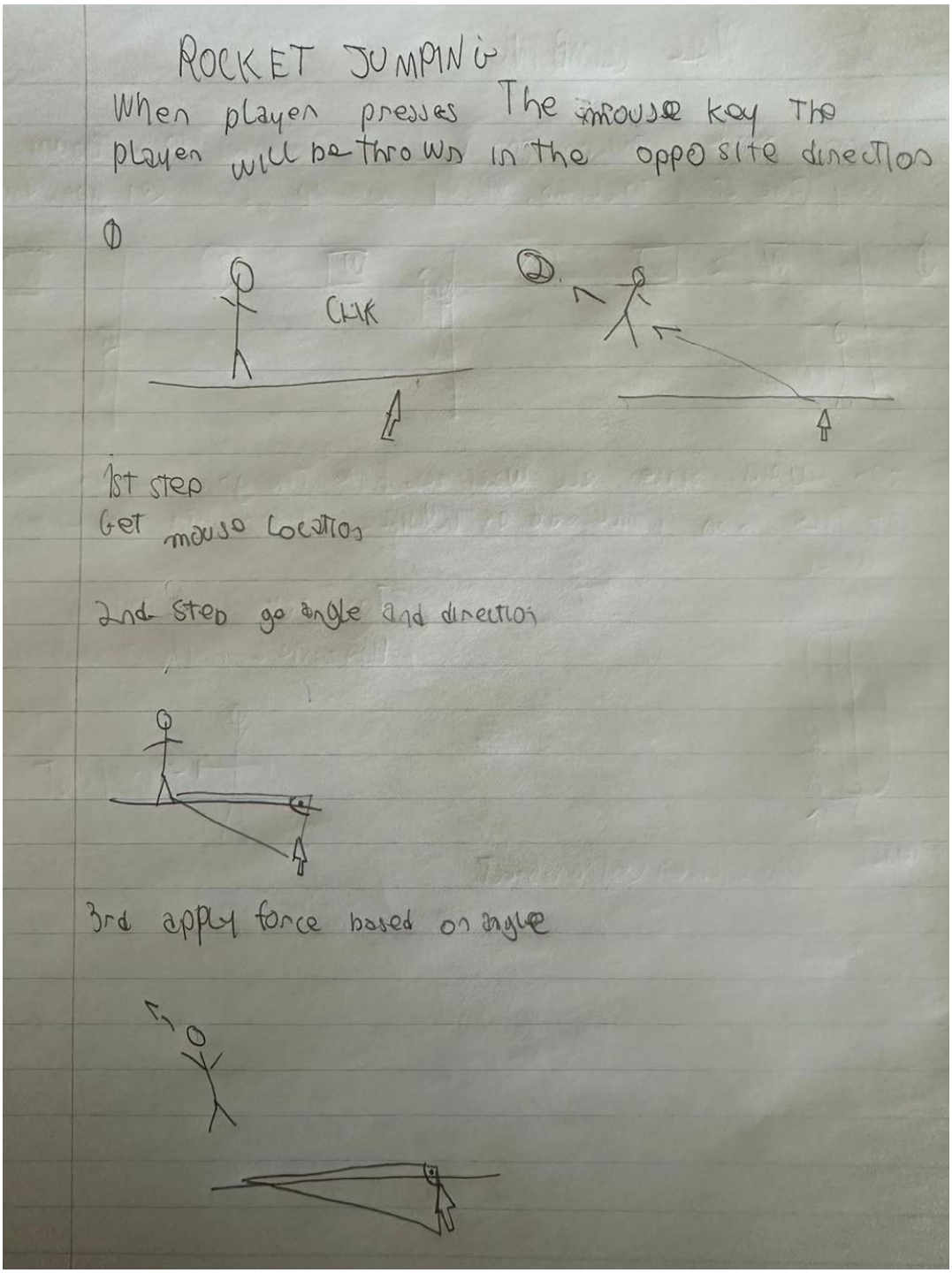
Player successfully climbing up wall

Task 2 complete (53 minutes)

Task 3 Rocket Jumping

Planning

The player when pressing the right mouse key, they should be flung in the opposite direction of the mouse. To get this done I will need to first get the mouse position, then use it to get the direction. With the direction I should be able to get the angle and then apply the force on the player game object based on the angle which will be the tangent (using atan2) to shoot the player in the opposite direction



Batch 5 (30 minutes)

For this task since its more confusing to complete I had to first break it down into smaller steps

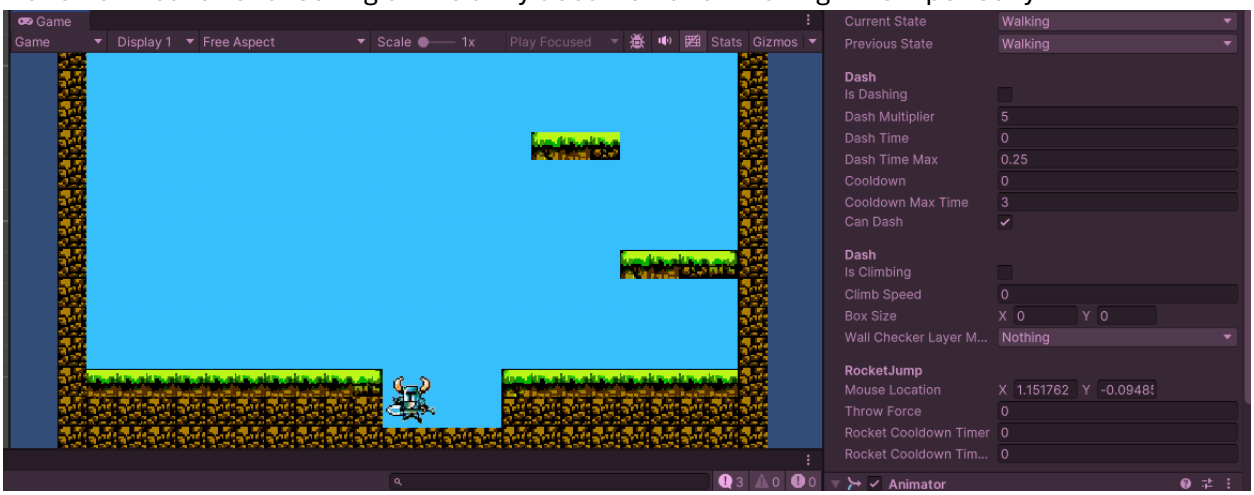
1. Get mouse location
2. Transform mouse location to angle
3. Get angle relative to player

4. Check if player clicks
5. Apply force with the angle
6. Create cooldown

So for first step I got the location of the mouse and locked it to the camera to the world

```
401     // RocketJump
        1 reference
402     private void getMouseLocation()
403     {
404         mouseLocation = Input.mousePosition;
405         mouseLocation = Camera.main.ScreenToWorldPoint(mouseLocation);
406
407
408     }
409
410 }
```

When first starting the 1st step I was a bit confused since I didn't remember how to use Screen to world Point but after checking on the unity documentation making it work perfectly



Now I needed to get the angle between the player and the mouse position

I started trying to get the angle I had a bit of difficulty; I reviewed the past slides to check for the angle and I saw I needed to get tangent of the location for the angle

I came up with this code

```
1 reference
private void getMouseLocation()
{
    mouseLocation = Input.mousePosition;
    mouseLocation = Camera.main.ScreenToWorldPoint(mouseLocation);
    Debug.DrawLine(transform.position, mouseLocation);

    //subtract to get direction and normalizing to set it to a lenght of 1
    Vector2 direction = (mouseLocation - transform.position).normalized;


    //get the angle using atan2 to get tangent
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;

}
```

This would get the direction by subtracting to get direction and then used the tangent to get the angle

Batch 6 (30 minutes)

After getting the force I noticed that I had a float, and I needed have a vector 2 to use applyForce

```
// RocketJump
1 reference
private void getMouseLocation()
{
    mouseLocation = Input.mousePosition;
    mouseLocation = Camera.main.ScreenToWorldPoint(mouseLocation);
    Debug.DrawLine(transform.position, mouseLocation);

    //subtract to get direction and normalizing to set it to a lenght of 1
    Vector2 direction = (mouseLocation - transform.position).normalized;


    //get the angle using atan2 to get tangent
    // OLD CODE angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    angle =  new Vector2(mouseLocation.x - transform.position.x, mouseLocation.y - transform.position.y);



}
1 reference
private void RocketJump()
{
    if (Input.GetMouseButton(0))
    {
        body.AddForce(throwForce * angle, ForceMode2D.Impulse);
    }
}
}
```

The new code I just get the direction and put it into a vector2, then I apply it to the rigid body when the player clicks

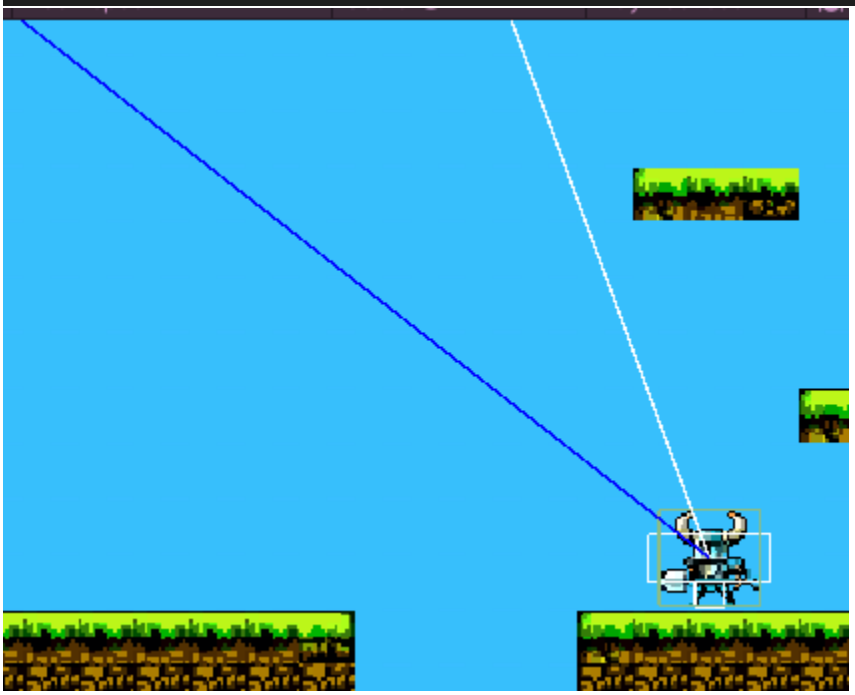
This is the new code

```
// RocketJump
1 reference
private void getMouseLocation()
{
    mouseLocation = Input.mousePosition;
    mouseLocation = Camera.main.ScreenToWorldPoint(mouseLocation);
    Debug.DrawLine(transform.position, mouseLocation);

    angle =  new Vector2(mouseLocation.x - transform.position.x, mouseLocation.y - transform.position.y);
}
1 reference
private void RocketJump()
{
    if (Input.GetMouseButton(0))
    {
        Debug.Log("boom Im flying");
        body.AddForce(throwForce * angle, ForceMode2D.Impulse);
    }
}
}
```

The code wasn't working correctly

I added this new line to get the direction to make sure the direction is correct

```
Debug.DrawRay(transform.position, angle * throwForce, Color.blue, 1f);
```



The blue line shows correctly where the player should've been throwing at, the white line should've been in the opposite direction but its pointing at the direction it is because I rushed the mouse to pause so I could take the screenshot.

I have the idea that the because the y velocity is being set as 0 the player can't be sent flying

After commenting the parts of the code the velocity.y is set to 0 it still didn't work

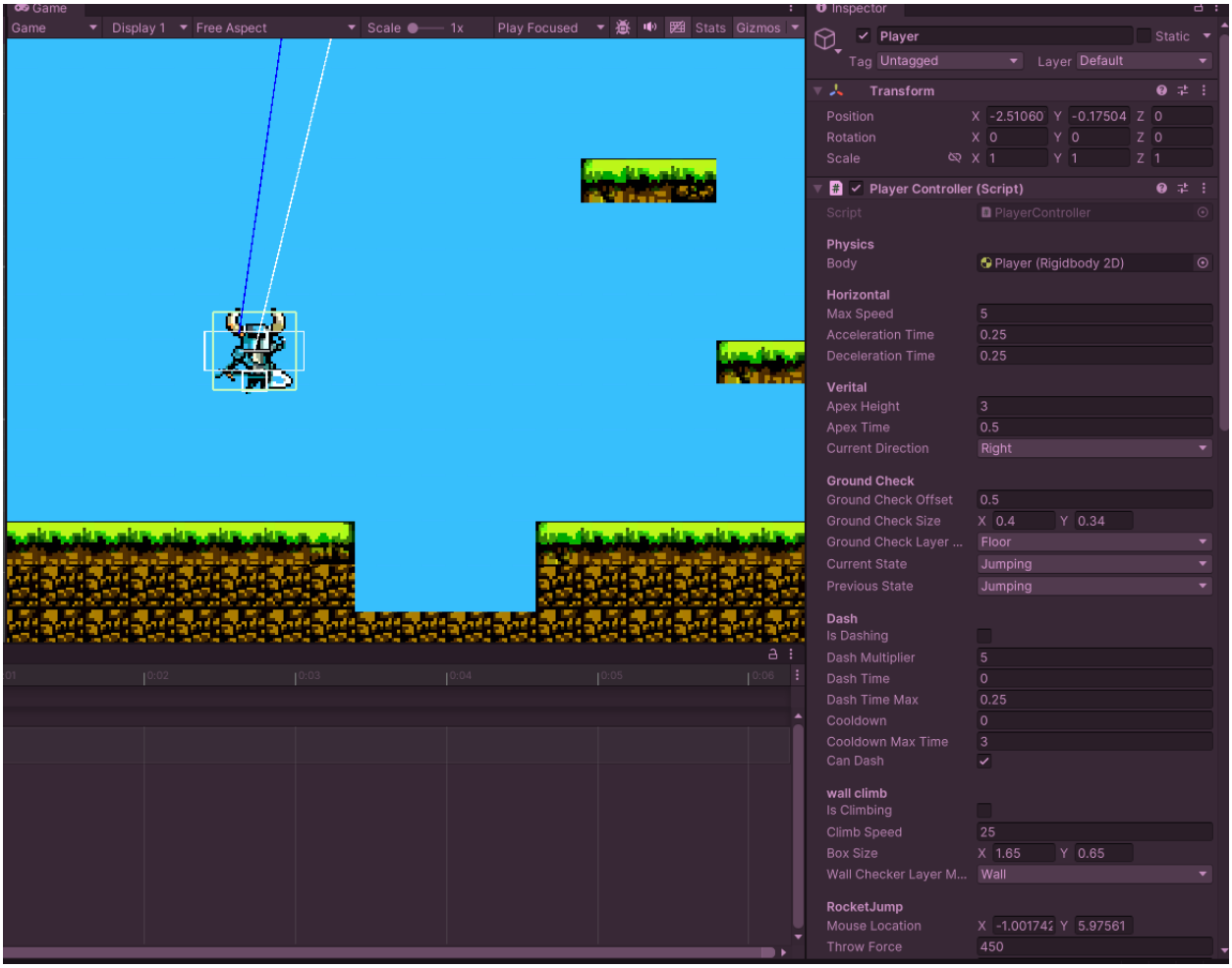
Batch 7 (1 hour and 15 minutes)

After reading more carefully the code I realized I used Impulse instead of Force, making the force very low

This is the code for the new rocket jump

```
// RocketJump
1 reference
private void getMouseLocation()
{
    mouseLocation = Input.mousePosition;
    mouseLocation = Camera.main.ScreenToWorldPoint(mouseLocation);
    Debug.DrawLine(transform.position, mouseLocation);

    Vector2 directionMouse =  new Vector2(mouseLocation.x - transform.position.x, mouseLocation.y - transform.position.y);

    angle = - directionMouse.normalized;
}
1 reference
private void RocketJump()
{
    if (Input.GetMouseButtonDown(0))
    {
        Debug.DrawRay(transform.position, angle * throwForce, Color.blue, 1f);

        Debug.Log("boom Im flying");
        body.AddForce((10*throwForce) * angle, ForceMode2D.Force);

    }
}
```

The throw force is multiplied by 10 because of how small it was.



Player flying towards blue line

Task 3 complete