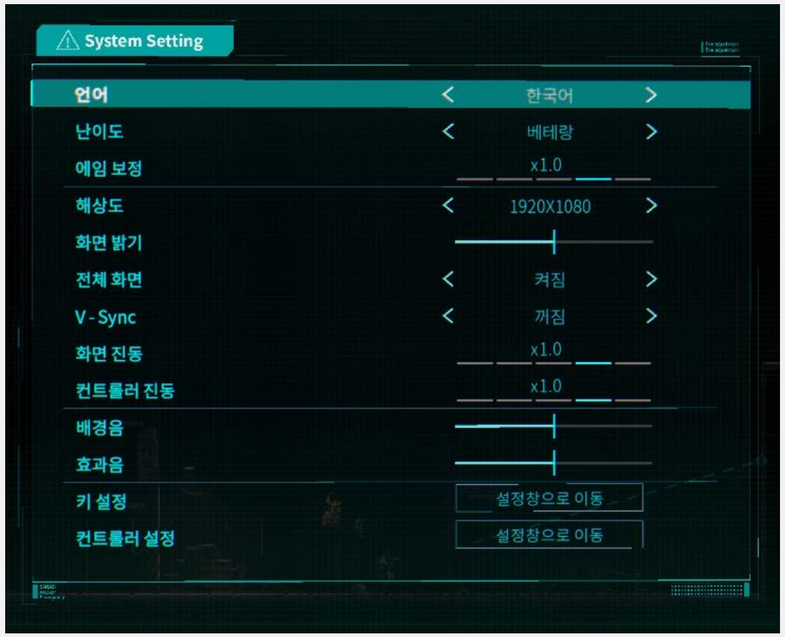
SANABI Copy Practice
PORTFOLIO 2023 by Unity

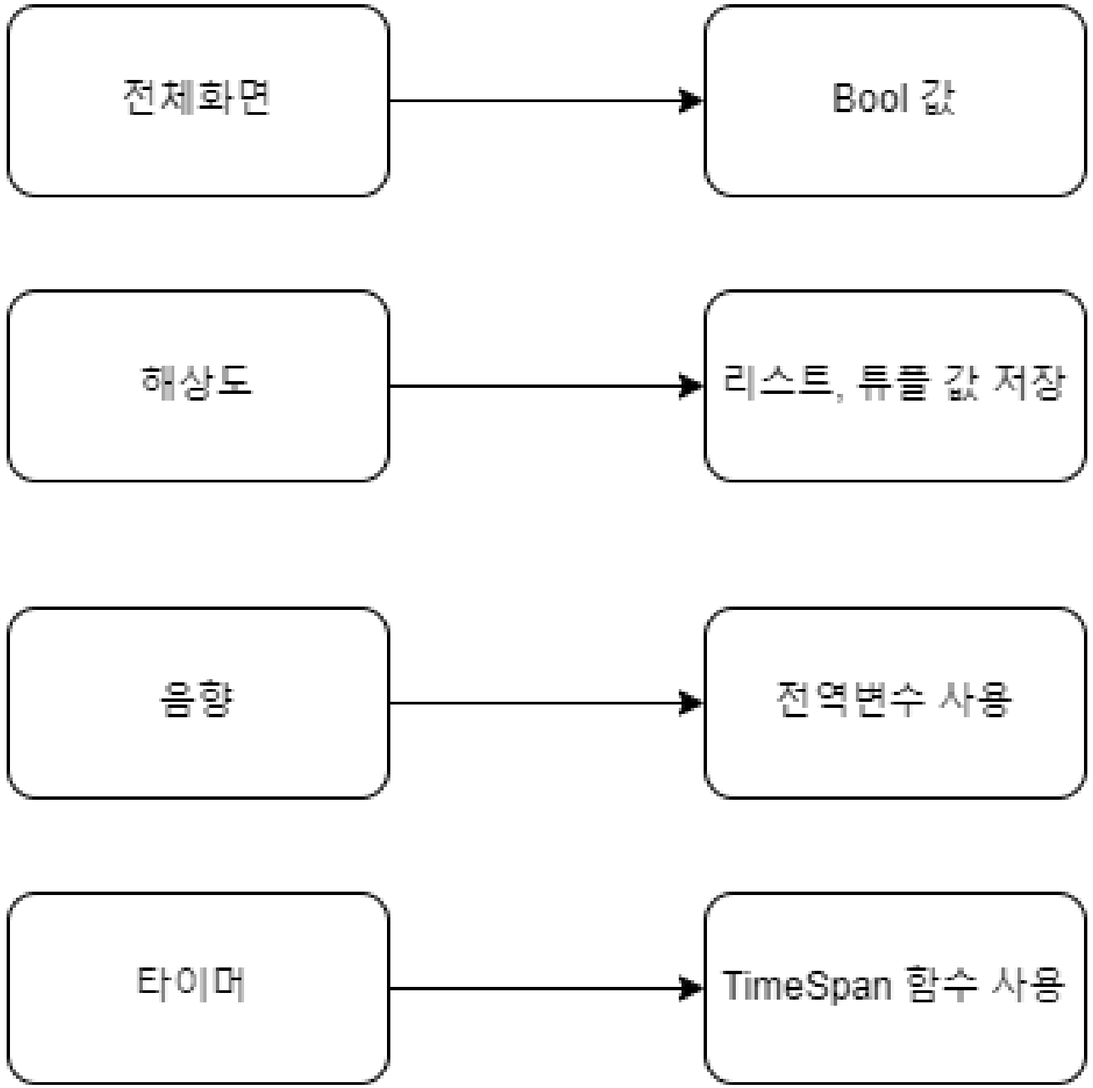
목차

본 포트폴리오는 산나비(SANABI) 모작이며
다른 용도로는 절대 사용되지 않습니다.

1. 버튼과 기본 시스템

순서도





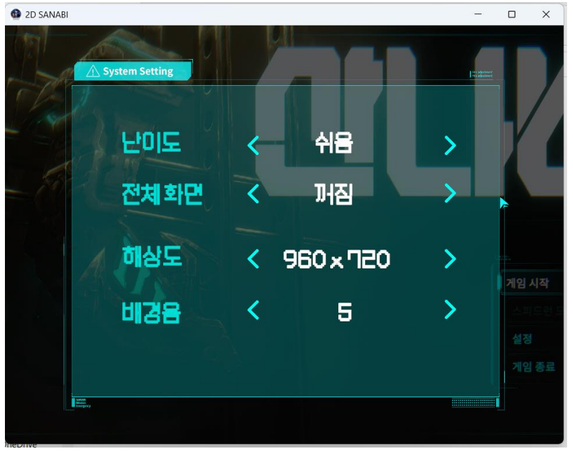
각 기능에 맞게 함수와 변수를 적절히 사용하였습니다.

해상도 설정


```
void Start()
{
    StateText();
    StartCoroutine(UpdateCoroutine());


    Screen.SetResolution(1920, 1080, true);

    if (PlayerPrefs.HasKey("ChangeWidth"))
    {
        fixedWidth = PlayerPrefs.GetInt("ChangeWidth");
        fixedHeight = PlayerPrefs.GetInt("ChangeHeight");
    }
    else
    {
        fixedWidth = Screen.width;
        fixedHeight = Screen.height;
        PlayerPrefs.SetInt("ChangeWidth", (int)fixedWidth);
        PlayerPrefs.SetInt("ChangeHeight", (int)fixedHeight);
        PlayerPrefs.Save();
    }
}
```

```
IEnumerator UpdateCoroutine()
{
    while (true)
    {
        screenSizeSettingText.text = screenSizeTextList[ScreensizeCount];
        resolutionSettingText.text = resolutionTextList[resolutionCount];
        backSoundSettingText.text = $"{soundCount}";

        yield return null;
    }
}
```

화면의 해상도를 설정하고,
HasKey로 변수 키가 저장되어 있는지 확인하여
초기화하거나 **PlayerPrefs.Save()**하여 변경된 값들을 저장합니다.



해상도 조절과 풀스크린, 사운드매니저 사용

튜플로 **string**형 리스트 변수 2개를 선언하여 해상도의 텍스트를 바꾸는데 사용하였습니다.

타이머 계산


```
[SerializeField]
string m_Timer = @"00:00:00.00";
bool m_IsPlaying;
[SerializeField]
float m_TotalSeconds; // 카운트 다운 전체 초(5분 X 60초)
public Text m_Text;

void Start()
{
    m_Timer = CountdownTimer(false); // Text에 초기값을 넣어 주기 위해
    m_IsPlaying = true;
}
```

```
string CountdownTimer(bool IsUpdate = true)
{
    if (IsUpdate)
        m_TotalSeconds += Time.deltaTime;

    TimeSpan timespan = TimeSpan.FromSeconds(m_TotalSeconds);
    string timer = string.Format("{0:00}:{1:00}:{2:00}.{3:00}",
        timespan.Hours, timespan.Minutes, timespan.Seconds, timespan.Milliseconds / 10);

    return timer;
}

private void SetZero()
{
    m_Timer = @"00:00:00.00";
    m_TotalSeconds = 0;
    m_IsPlaying = false;
}
```

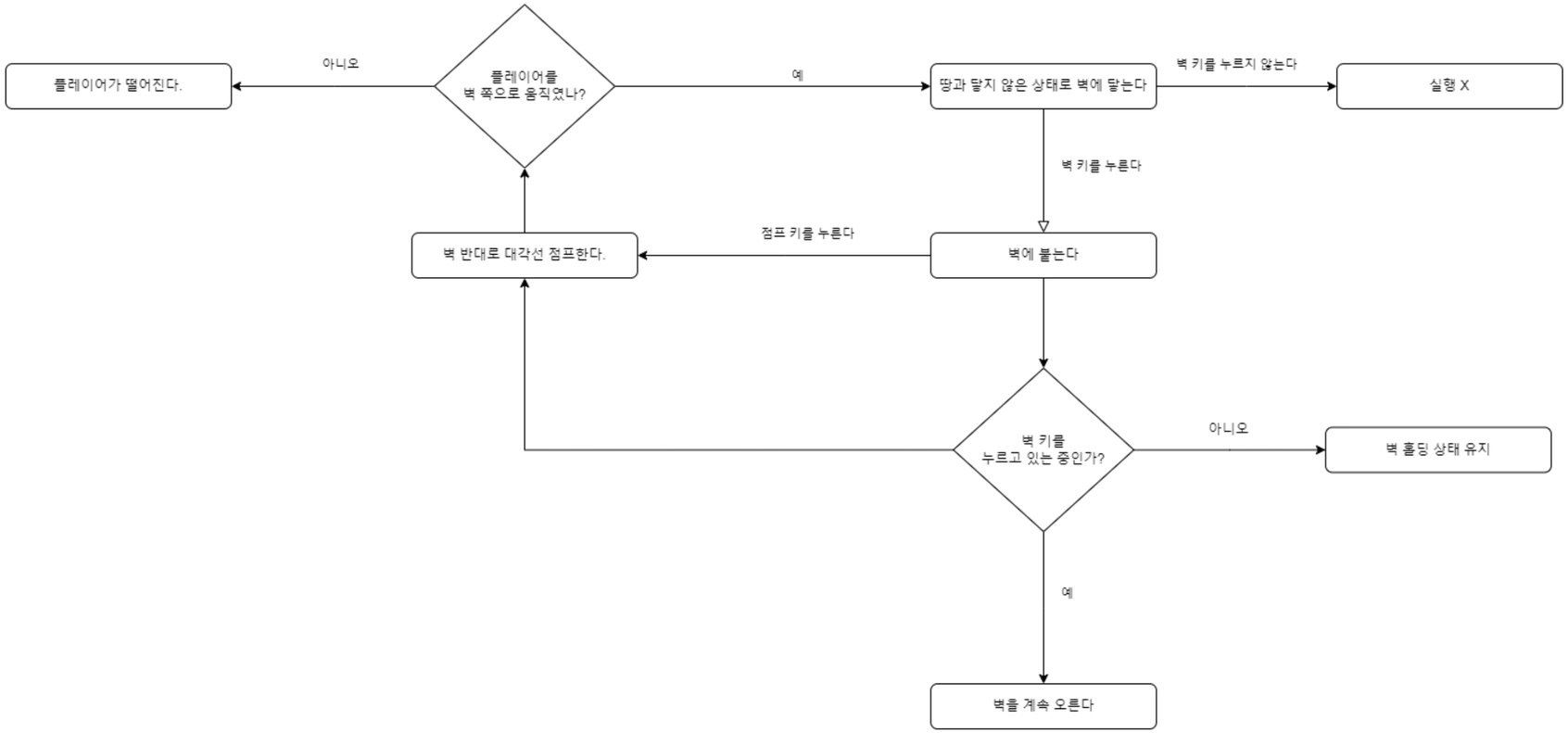
```
void Update()
{

    if (m_IsPlaying)
    {
        m_Timer = CountdownTimer();
    }

    // 사망했을 때를 기점으로
    if (m_TotalSeconds <= 0)
    {
        SetZero();
    }

    if (m_Text)
        m_Text.text = m_Timer;
}
```

TimeSpan 함수로 시간 연산을 하여
String.Format으로 문자열로 나타내었습니다

플레이어 사망 시 초기화 로직입니다

2. 플레이어 움직임 및 충돌

순서도



플레이어가 벽에서 할 수 있는 상황들을 모두 고려하였습니다.

플레이어 움직임 호출

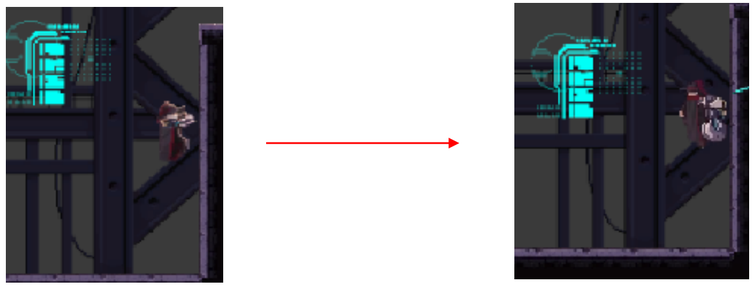

```
void Update()
{
    float x = Input.GetAxis("Horizontal");
    float y = Input.GetAxis("Vertical");
    Vector2 direction = new Vector2(x, y);

    Run(direction);

    WallGrab();
    WallClimb();

    if (isWallGrab)
    {
        rigid.gravityScale = 0f;
        rigid.velocity = new Vector2(rigid.velocity.x, 0);
        float speedModifier = y > 0 ? 0.7f : 1.5f;
        rigid.velocity = new Vector2(rigid.velocity.x, y * (runSpeed * speedModifier));

        if (collision.IsOnLeftWall)
        {
            spriteRenderer.flipX = true;
        }
        else if (collision.IsOnRightWall)
        {
            spriteRenderer.flipX = false;
```

수평키를 받으면 좌우로 힘이 가해지고
수직키를 받으면 중력이 0일 때 입력값으로 힘이 가해지는 효과를 얻을 수 있습니다.

```
void Run(Vector2 dir)
{
    if (isWallGrab)
    {
        return;
    }

    if (collision.IsOnGround && dir.x != 0)
    {
        animationState.ChangeAnimationState(animationState.player_Run);
    }
    else if (collision.IsOnGround && dir.x == 0)
    {
        animationState.ChangeAnimationState(animationState.player_Idle);
    }

    rigid.velocity = new Vector2(dir.x * runSpeed, rigid.velocity.y);
}
```

플레이어가 벽타기를 하지 않을 때
플레이어의 힘을 비교해서
복잡한 애니메이션 선 연결을 하지않고
애니메이션 클립 이름에 따라
매개변수로 만들어서 함수가 호출되면
플레이어의 애니메이션 상태가 바뀝니다.

```
static string currentState;
static Animator animator;

void Start()
{
    animator = GetComponent<Animator>();
    currentState = "player_Idle";
}

public static void ChangeAnimationState(string newState)
{
    // Stop the same animation from interrupting itself
    if (currentState == newState)
    {
        //Debug.Log("currentState == newState");
        return;
    }

    // Play the animation
    animator.Play(newState);

    // Reassign the current state
    currentState = newState;
}
```



플레이어 벽점프


```
public void WallJump()
{
    isWallJumping = true;
    wallJumpingTime = 0f;

    if (collision.IsOnLeftWall)
    {
        spriteRenderer.flipX = false;
    }
    else if (collision.IsOnRightWall)
    {
        spriteRenderer.flipX = true;
    }

}
```

```
if (wallJumpingTime < wallJumpingTimer)
{
    wallJumpingTime += Time.deltaTime;

    if (collision.IsOnLeftWall)
    {
        rigid.velocity = new Vector2(wallJumpingPower.x, wallJumpingPower.y);
    }
    else if (collision.IsOnRightWall)
    {
        rigid.velocity = new Vector2(-1 * wallJumpingPower.x, wallJumpingPower.y);
    }
}

if (collision.IsOnGround || collision.IsOnWall)
{
    isWallJumping = false;
}

if (isWallJumping)
{
    rigid.gravityScale = playerMovement.playerRigidValue;
}
```

플레이어가 벽에 붙어있을 때 점프하면
스프라이트 이미지가 반전됩니다.

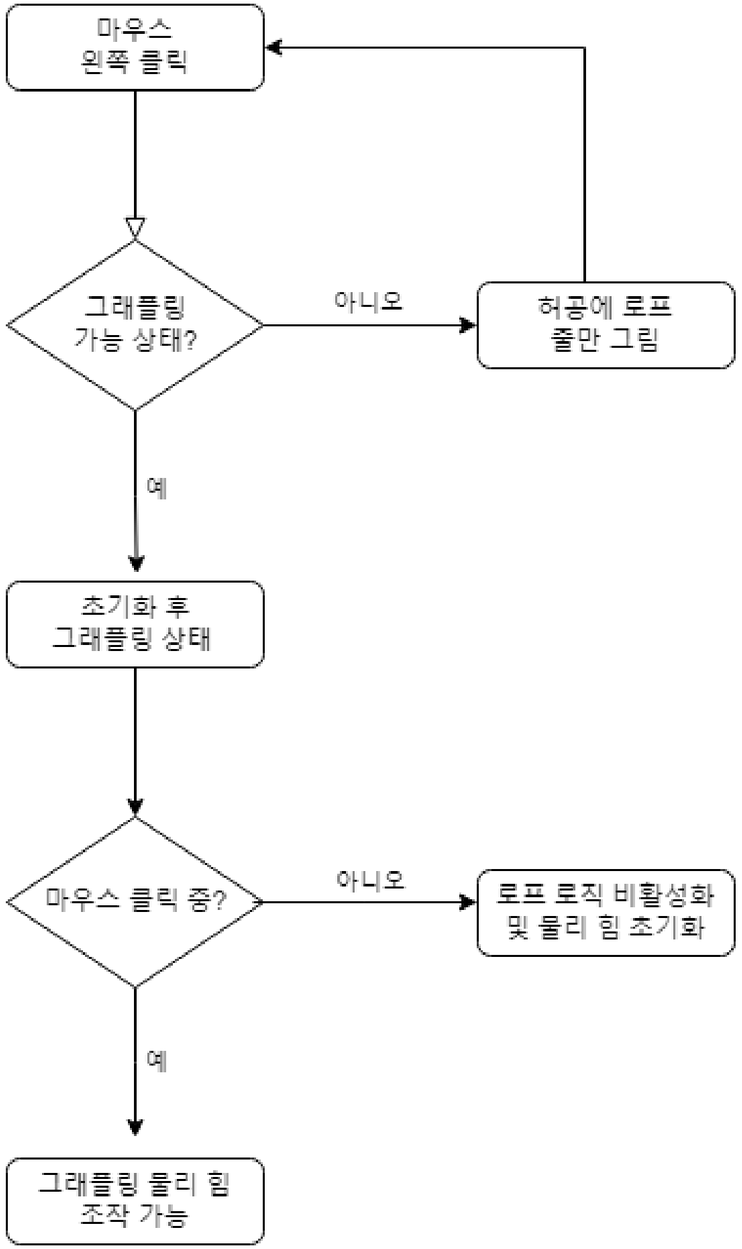
Bool값으로 벽점프를 체크하고
적절한 값을 사용해 일반적인 플랫포머의
포물선 점프와 비슷하게 구현하였습니다.

게임 Celeste 참고

https://store.steampowered.com/app/504230?snr=5000_5100___primarylinks

3. 그래플링 건 로직

순서도



플레이어가 마우스를 누름에 따라 실행되는 로직입니다.

그래플링 속도


```
//마우스를 누르면 레이캐스트에 부딪힌 오브젝트를 _hit.point로 지정
//부딪힌 물체가 없으면 선만 그리기
void SetGrapplePoint()
{
    Vector2 distanceVector = m_camera.ScreenToWorldPoint(Input.mousePosition) - gunPivot.position;
    RaycastHit2D hit = Physics2D.Raycast(rayPoint.position, distanceVector.normalized);

    if (hit.transform != null && IsGrappableLayer(hit.transform.gameObject.layer)
        && Vector2.Distance(rayPoint.position, hit.point) <= maxDistnace)
    {
```

```
//마우스를 누르는 중이면 Flip과 줄 길이 조정, launchToType 기능
void Grappling()
{
    animationState.ChangeAnimationState(animationState.player_Grappling);



    float VelocityX = Mathf.Abs(rigid.velocity.x);
    float VelocityY = Mathf.Abs(rigid.velocity.y);

    if (VelocityX > maxGrapplingHorizontalSpeed)
    {
        rigid.velocity = new Vector2(Mathf.Sign(rigid.velocity.x) * maxGrapplingHorizontalSpeed, rigid.velocity.y);
    }
    if (VelocityY > maxGrapplingVerticalSpeed)
    {
        rigid.velocity = new Vector2(rigid.velocity.x, Mathf.Sign(rigid.velocity.y) * maxGrapplingVerticalSpeed);
    }
```

```
float grappleYPosition = grapplePoint.y;
bool isHigher = grappleYPosition > transform.position.y;

if (Input.GetKey(KeyCode.A))
{
    leftGrappleVelocityX = (isHigher ? -1f : 1f) * GrapplingSpeed;
    rigid.AddForce(leftGrappleVelocityX * rightDirection);
}
else if (Input.GetKey(KeyCode.D))
{
    rightGrappleVelocityX = (isHigher ? 1f : -1f) * GrapplingSpeed;
    rigid.AddForce(rightGrappleVelocityX * rightDirection);
}
```

Math.Abs로 힘의 절댓값을 구하여
해당 축의 부호를 힘에 따라
Math.Sign으로 반환하였습니다.

이는 속도 값 제한 로직입니다.

거꾸로 그래플링 시
힘 방향을 거꾸로 하여
자연스럽게 합니다.

이곳 Y를 기준으로 합니다.

그래플링 한 지점을 **grappleYPosition**으로 하고
bool값으로 y 위치를 **삼항연산자**로 비교하여
힘의 부호값을 정합니다.

총을 회전하는 최적의 각도 계산


```
//mousePos에 따라서 총의 방향과 회전
void RotateGun(Vector3 lookPoint, bool allowRotationOverTime)
{
    Vector3 distanceVector = lookPoint - gunPivot.position;

    float angle = Mathf.Atan2(distanceVector.y, distanceVector.x) * Mathf.Rad2Deg;
    if (rotateOverTime && allowRotationOverTime)
    {
        gunPivot.rotation = Quaternion.Lerp(gunPivot.rotation, Quaternion.AngleAxis(angle, Vector3.forward), Time.deltaTime * rotationSpeed);
    }
    else
    {
        gunPivot.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
    }
}
```

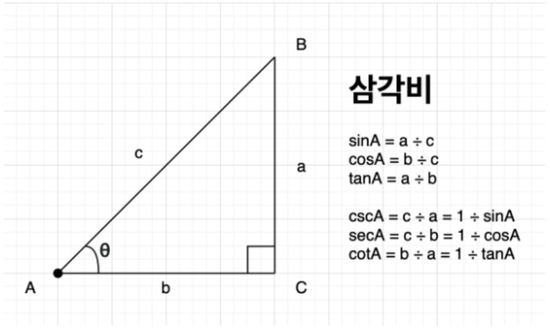
Atan2를 사용하여 삼각함수의 두 점 간의 각도를 라디안으로 반환합니다.

Rad2Deg는 **라디안**을 **각도**로 변환하는 **변환상수**입니다.
이것으로 z값을 정확히 구했습니다.

구한 z값을 이용하여
Quaternion.AngleAxis , Quaternion.Lerp를 사용하여
부드러운 그래플링 건 방향 회전을 구현하였습니다.

Atan2에 대하여

아크탄젠트란?



삼각비

아크탄젠트(arctangent)는 역탄젠트라고도 하며 **탄젠트의 역함수**이다. 아크탄젠트를 이용하면 위 사진에서의 θ의 각도를 구할 수 있다.

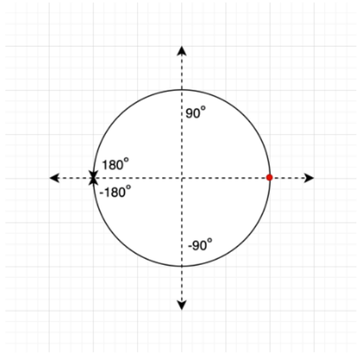
모든 프로그래밍 언어에는 아크탄젠트를 계산할 수 있도록 Math 모듈에 atan(y / x)과 atan2(y, x) 함수를 지원한다.

삼각함수의 나머지 두 변의 값만 알고 있다면
각(세타)를 tan의 역함수인 아크 탄젠트(Atan)를
구할 수 있습니다.

atan과 atan2의 차이점

atan과 atan2은 두 점 사이의 θ의 절대각을 구하는 함수인데 왜 두가지로 나뉘었을까?

atan은 두 점 사이의 **탄젠트값을 받아** 절대각을 **-π/2 ~ π/2**의 라디안 값으로 반환한다. (-90 ~ 90도)



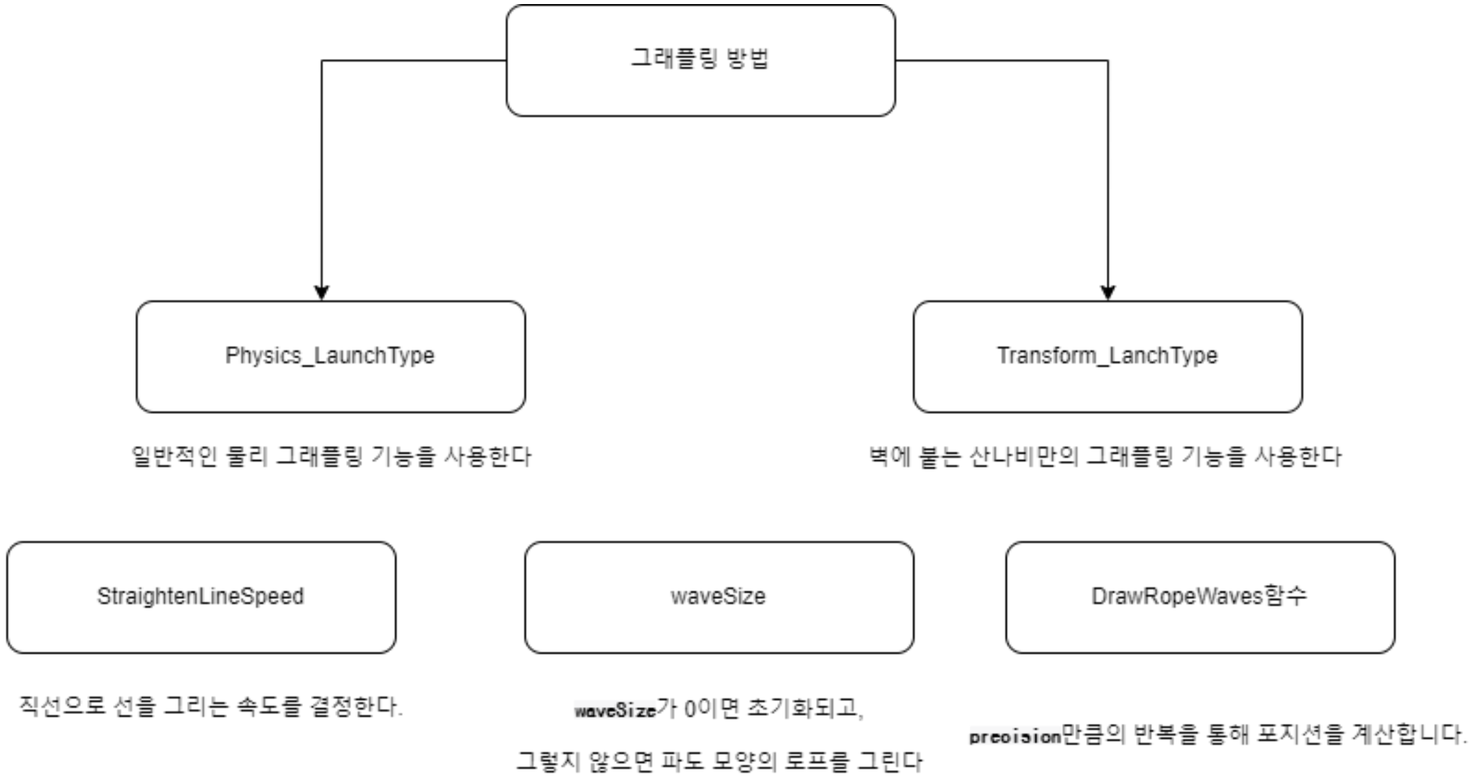
atan2의 리턴값 범위

atan2는 두 점 사이의 **상대좌표(x, y)를 받아** 절대각을 **-π ~ π**의 라디안 값으로 반환한다. (-180 ~ 180도)

유니티에서 Atan을 거의 쓰지 않는 이유는
-90 ~ 90 이면 180도까지밖에 범위가 안되고
X / Y 계산식에서 Y가 0이 나오면
X 상관없이 값이 0이 되버리기 때문입니다.

4. 그래플링 로프 메소드

순서도



플레이어의 그래플링 줄의 기능을 정렬하였습니다.

그래플링 상태 호출

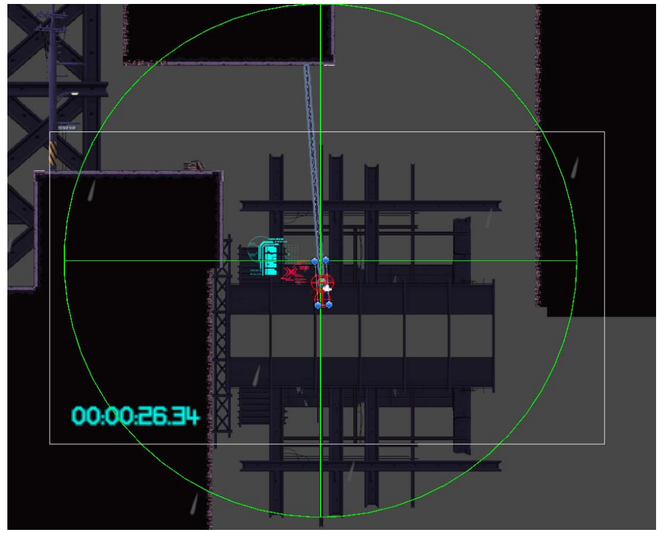

```
//그래플링 방법
public void Grapple()
{
    distanceJoint2D.autoConfigureDistance = false;
    if (!launchToPoint)
    {
        if (autoConfigureDistance)
        {
            distanceJoint2D.autoConfigureDistance = true;
        }

        distanceJoint2D.connectedAnchor = grapplePoint;
        distanceJoint2D.enabled = true;
    }
```

```
else
{
    switch (launchType)
    {
        case LaunchType.Physics_Launch:
            distanceJoint2D.connectedAnchor = grapplePoint;

            Vector2 distanceVector = rayPoint.position - playerTransform.position;

            distanceJoint2D.distance = distanceVector.magnitude;
            distanceJoint2D.enabled = true;

            break;
        case LaunchType.Transform_Launch:
            rigid.gravityScale = 0;
            playerMovement.MakeVelocityZero();

            break;
    }
}
```

그래플링 로프에서 호출하는 상태를 변신하는 로직입니다.

autoConfigureDistance 항목은 **distance**거리 제한을 없앱니다.

Physics_LaunchType에서**distance**에 **magnitude**로 벡터를 반환하여 그래플링 지점까지 길이를 설정하고 활성화 해줍니다.



Distance거리 표시

그래플링 상태 재정비


```
void OnEnable()
{
    for (int i = 0; i < precision; i++)
    {
        rope_lineRenderer.SetPosition(i, firePoint.position);
    }
    moveTime = 0;
    rope_lineRenderer.positionCount = precision;
    waveSize = StartWaveSize;
    isStraightLine = false;
    rope_lineRenderer.enabled = true;
}

void OnDisable()
{
    rope_lineRenderer.enabled = false;
    isGrappling = false;
}
```

비활성화 시 호출되는**OnEnable**함수입니다.
실행 시 초기화 로직이 실행됩니다

```
void DrawRope()
{
    if (!isStraightLine)
    {
        if (Mathf.Approximately(rope_lineRenderer.GetPosition(precision - 1).x, grapplingGun.GetgrapplePoint.x))
        {
            isStraightLine = true;
        }
        else
        {
            DrawRopeWaves();
        }
    }
```

두 발사지점x를 부동소수점까지 비교해서 같거나
오차가 크지 않다면 직선으로 선을 그립니다.

로프 속도


```
else
{
    if (!isGrappling)
    {
        grapplingGun.Grapple();
        isGrappling = true;
    }

    if (waveSize > 0)
    {
        waveSize -= Time.deltaTime * straightenLineSpeed;
        DrawRopeWaves();
    }
    else
    {
        waveSize = 0;

        if (rope_lineRenderer.positionCount != 2) { rope_lineRenderer.positionCount = 2; }

        DrawRopeNoWaves();
    }
}
```

StraightenLineSpeed는 직선 줄의 그리는 속도를 결정합니다.

waveSize는 웨이브의 크기를 나타냅니다.

waveSize == 0이면 초기화합니다.

```
//로프를 그리는 코드들
void DrawRopeWaves()
{
    rope_lineRenderer.positionCount = 0;

    for (int i = 0; i < precision; i++)
    {
        float delta = (float)i / ((float)precision - 1f);
        Vector2 targetPosition = Vector2.Lerp(firePoint.position, grapplingGun.GetgrapplePoint, delta);// + offset;
        Vector2 currentPosition = Vector2.Lerp(firePoint.position, targetPosition, ropeProgressionCurve.Evaluate(moveTime) * ropeProgressionSpeed);

        rope_lineRenderer.positionCount++;
        rope_lineRenderer.SetPosition(i, currentPosition);
    }
}
```

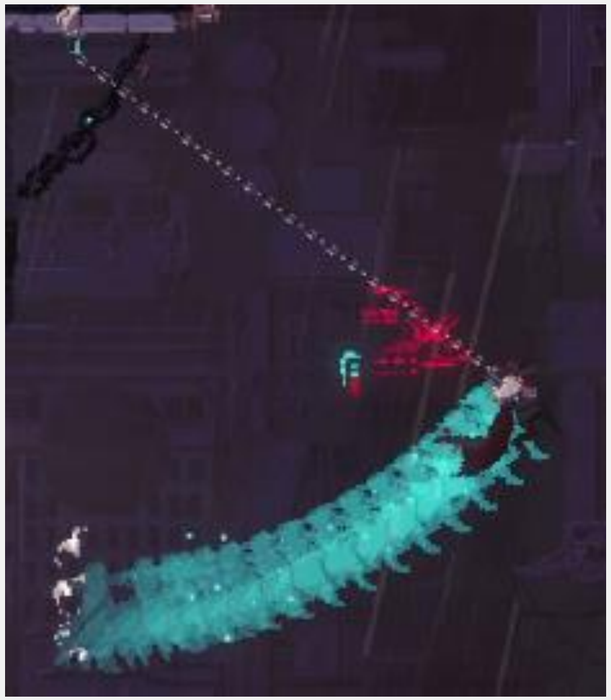
로프를 파도 모양으로 그립니다.
반복문으로 **precision**만큼
포지션 계산합니다

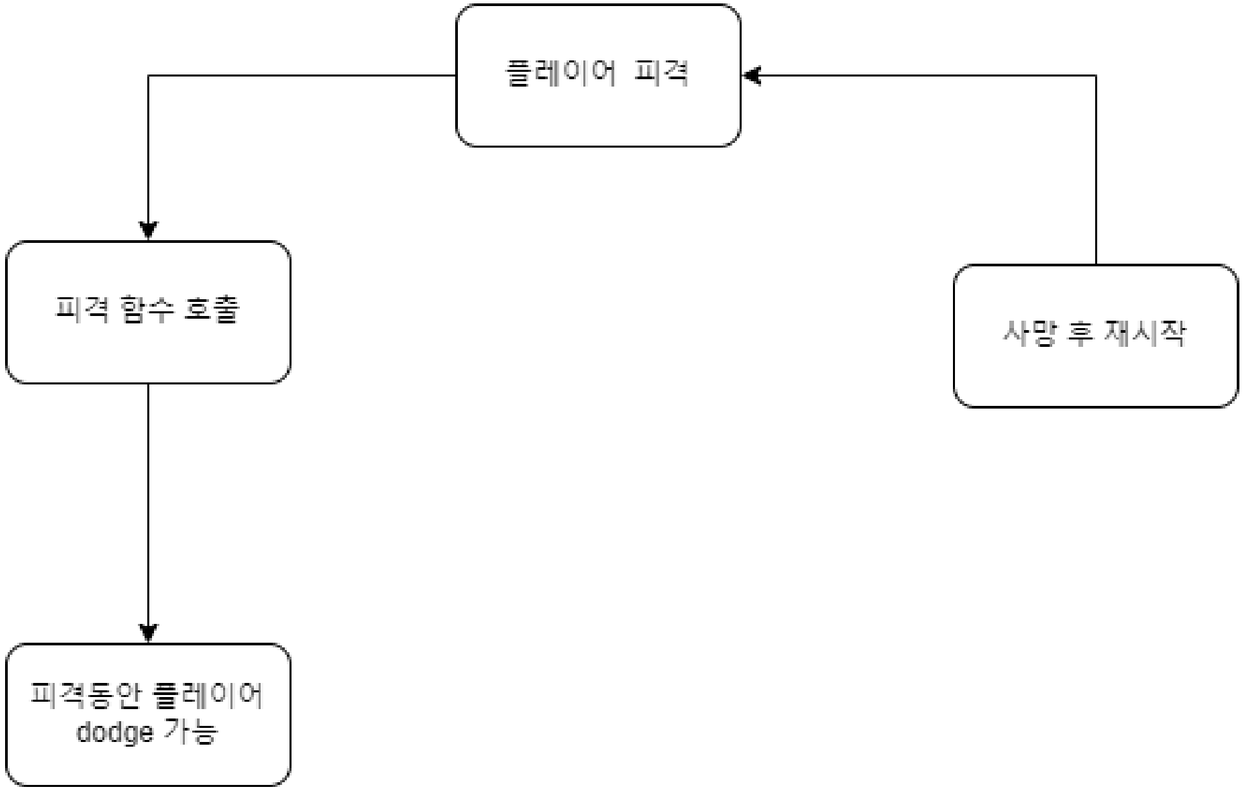
```
void DrawRopeNoWaves()
{
    rope_lineRenderer.SetPosition(0, firePoint.position);
    rope_lineRenderer.SetPosition(1, grapplingGun.GetgrapplePoint);
}
```

파도 모양 없이 직선 로프를 그립니다.

5. 추가 기능

순서도





플레이어의 피격 함수 호출입니다.

체력바와 피격 대쉬


```
IEnumerator UpdateCoroutine()
{
    while (true)
    {
        Vector3 targetPosition = new Vector3(playerPos.position.x - 5f, playerPos.position.y + 2.5f, transform.position.z);
        transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref vel, 0.2f);

        yield return null;
    }
}

public void HealthBarDecrease()
{

    if (healthCount > 0)
    {
        healthList[healthCount].SetActive(false);
        healthList[healthCount - 1].SetActive(true);

        healthCount--;
    }
    else
    {
        playerHealth.Respawn();
    }
}
```

플레이어 위치 주변으로 이동할 목표 위치를 계산합니다.

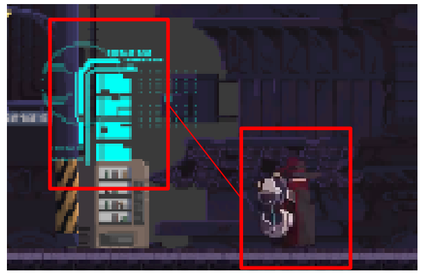
SmoothDamp는 현재 위치에서 목표 위치로 부드럽게 이동합니다.

ref vel을 사용하여 부드러운 이동에 필요한 속도 값을 관리합니다.
이전 속도를 참조하여 새로운 속도를 계산합니다.

0.2f는 이동 속도를 제어하는 매개변수입니다.
값이 작을수록 더 빠르게 이동합니다.

```
//0.5초안에 누르면 회피가능
if (canDodgeTimer <= canDodgeTime)
{
    canDodgeTimer += Time.deltaTime;


    if (Input.GetKeyDown(KeyCode.Space) && isCanInputDodge)
    {
        isCanInputDodge = false;
        canDodgeTimer = 0f;

        //회피 대쉬 조작
        float power = 1000f;
        if (Input.GetKey(KeyCode.W))
        {
            rigid.AddForce(Vector3.up * power);
        }
        else if (Input.GetKey(KeyCode.S))
        {
            rigid.AddForce(Vector3.down * power);
        }
        else if ( Input.GetKey(KeyCode.D))
        {
            rigid.AddForce(Vector3.right * power);
        }
```

피격 시 스페이스 바를 눌렀을 때 방향키에 따라
자연스럽게 대쉬할 수 있습니다.



플레이어에 따라
위치 천천히 따라가기

플레이어 피격

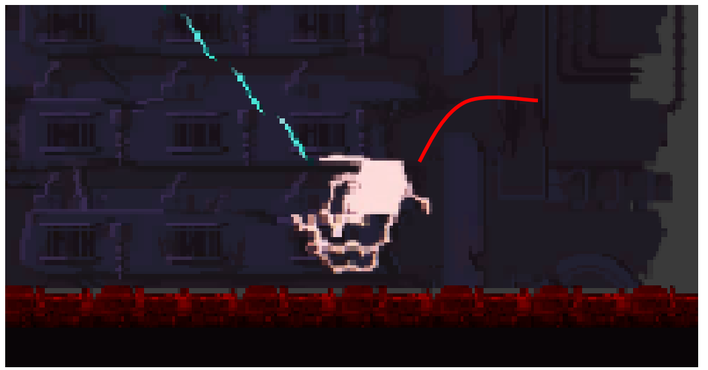

```
void OnDamaged(Vector2 targetPos)
{
    canDodgeTimer = 0f;
    isCanInputDodge = true;
    isOkJumpingAnim = false;
    //충돌시 플레이어는 무적 레이어
    if (!collision.isUnstoppable)
    {
        collision.isUnstoppable = true;
    }
    AnimationState.ChangeAnimationState("player_Damaged");


    //한대 맞으면 튕겨나가게
    int dir = transform.position.x - targetPos.x > 0 ? 1 : -1;
    int Ydir = 1;
    int rigidSize = 10;

    playerMovement.MakeVelocityZero();

    //오른쪽으로 맞으면 오른쪽으로 튕겨나가고 왼쪽으로 맞으면 왼쪽으로 튕겨나가고
    rigid.AddRelativeForce(new Vector2(dir, Ydir) * rigidSize, ForceMode2D.Impulse);

    Invoke("OffDamaged", 3f);
}
```

피격 위치에 따라 즉각적으로
플레이어가 튕겨나가는 방향이 정해집니다.(**dir**)

```
public void Respawn()
{
    Scene currentScene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(currentScene.buildIndex);

    Time.timeScale = 1f;
}

void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Bullet"))
    {
        TakeDamageSystem();
        OnDamaged(collision.transform.position);
        healthBar.HealthBarDecrease();
    }

    if (collision.CompareTag("DieZone"))
    {
        Respawn();
    }

}
```

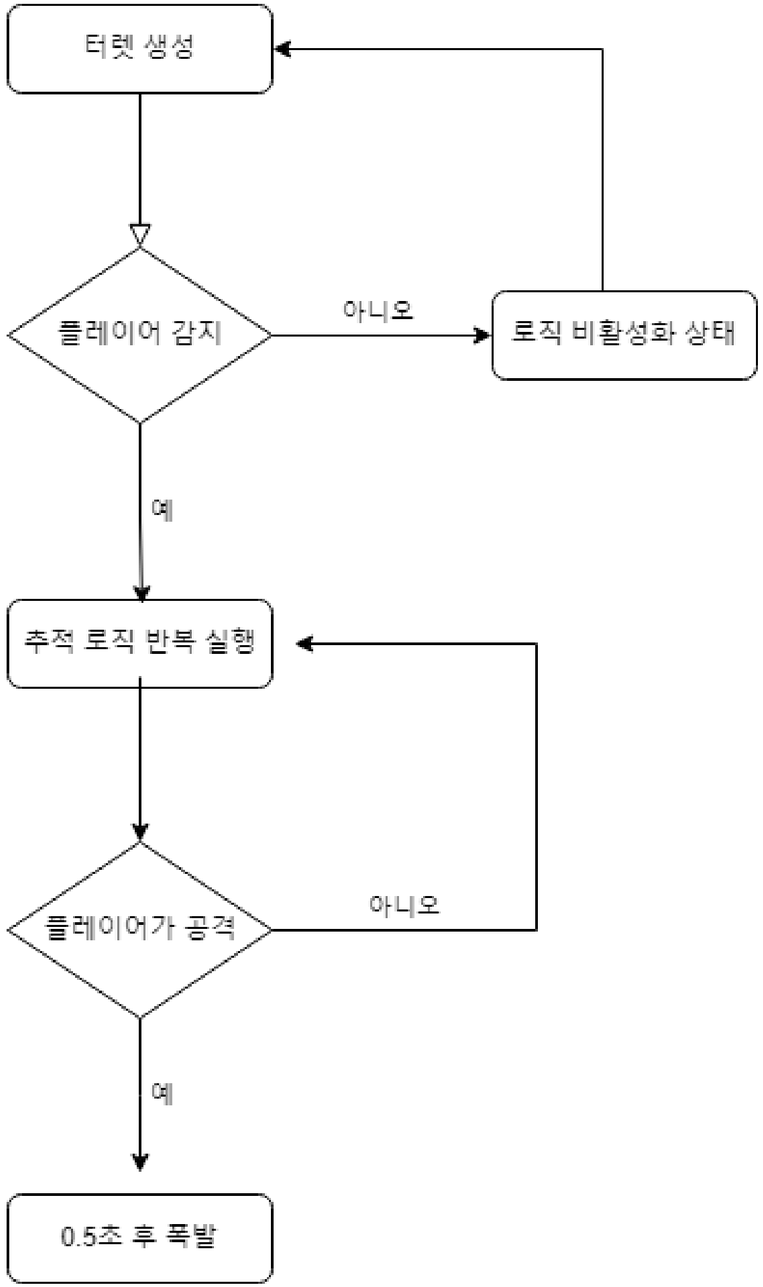
GetActiveScene으로 이번 씬을 **currentScene**에 넣고
사망하였다면 호출합니다.

만약 체력이 충분하다면 피격 함수를 호출합니다.



6. 터렛

순서도



터렛건의 플레이어 추적 로직입니다.

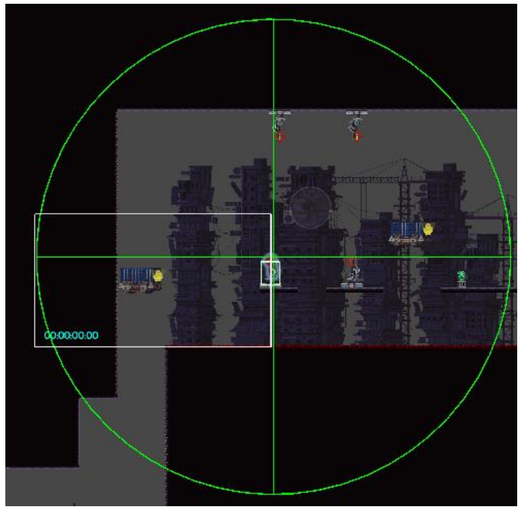
터렛 플레이어 추적


```
void Update()
{
    Vector2 direction = playerPos.position - firePoint.position;

    RaycastHit2D[] hits = Physics2D.RaycastAll(firePoint.position, direction, radius, targetLayer);

    bool hitEnemy = false;

    foreach (RaycastHit2D hit in hits)
    {
        if (hit.collider.gameObject.layer == 3 || hit.collider.gameObject.layer == 6)
        {
            Debug.Log(hit.collider.gameObject.layer);
            hitEnemy = true;
            break;
        }
    }
```

RaycastHit2D의 All을 받아서 만약 해당되는 레이어의 게임오브젝트가 **radius**값 안에 체크되었다면 매 프레임 공격을 실행합니다.

```
if (hitEnemy) //플레이어를 hit으로 바라볼 때
{
    if (startTimer < startTime)
    {
        startTimer += Time.deltaTime;
        Rope();
        RotateGun();
    }
    else
    {
        if (curAmmo > 0)
        {
            //StartShot
            if (shootTime >= shootTimer)
            {
                StartCoroutine(Shot());
                curAmmo--;
                shootTime = 0f;
                canShoot_lineRenderer.enabled = false;
            }
            else
            {
                shootTime += Time.deltaTime;
            }
        }
```

매 프레임 탄약 수를 체크합니다.



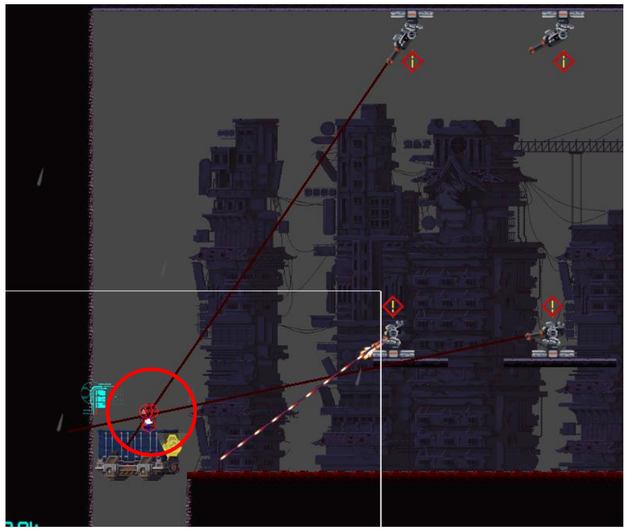
터렛 건바디 회전


```
void Rope()
{
    canShoot_lineRenderer.SetPosition(0, firePoint.position);
    canShoot_lineRenderer.SetPosition(1, firePoint.position + firePoint.up * radius);
    canShoot_lineRenderer.enabled = true;
}

void RotateGun()
{
    float angle = Mathf.Atan2(gunBody.position.y - playerPos.position.y, gunBody.position.x - playerPos.position.x) * Mathf.Rad2Deg + 90f;
    gunBody.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
}
IEnumerator Shot()
{
    GameObject bullet = ObjectPoolManager.Instance.BulletGetQueue();//옵젝풀매니저의 총알 객체 생성
    bullet.transform.SetPositionAndRotation(firePoint.position, firePoint.rotation);

    anim.SetTrigger("Turret_Shoot");
    yield return null;
}

void OnDrawGizmosSelected()
{
    Gizmos.color = Color.green;
    Gizmos.DrawWireSphere(firePoint.position, radius);
}
```

이것 또한 플레이어 건 회전처럼 **Atan2**를 이용하여 라디안을 변환하여 터렛 건바디의 회전을 구현하였습니다.

```
    else if (curAmmo == 0)
    {
        //Reload
        if (reloadTime >= reloadTimer)
        {
            curAmmo = chargeAmmoCount;
            reloadTime = 0;
        }
        if (reloadTime < reloadTimer)
        {
            reloadTime += Time.deltaTime;
            Rope();
            RotateGun();
        }
    }
}
```

탄약 수 0에 따라 재장전 시간이 있습니다.

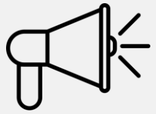


분석 및 결론

처음에는 **distanceJoint2D**말고 많이 쓰이는 **springJoint2D**를 위주로 그래플링 훅을 만들면서 산나비보다 스파이더맨 느낌이 나서 많이 당황스러웠습니다.

그래플링 훅 지점에서 **distanceJoint2D**의 기능이 물리 연산과 겹치면서 많이 애를 먹었습니다.

이로 인해 여러가지 시도하면서 로프액션의 물리를 조절하는 함수와 로직, 라인 그리는 방법을 많이 경험하게 되었습니다.



물리 연산과 회전에 쓰이는 삼각함수 등에 대해
꽤나 C#과 수학의 연관성에 재미를 느낄 수 있었습니다.

출처 표기

#Atan2 블로그 이미지 출처

https://spiralmoon.tistory.com/entry/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9D%B4%EB%A1%A0-%EB%91%90-%EC%A0%90-%EC%82%AC%EC%9D%B4%EC%9D%98-%EC%A0%88%EB%8C%80%EA%B0%81%EB%8F%84%EB%A5%BC-%EC%9E%AC%EB%8A%94-atan2

#템플릿 출처

흰색 주황색 회색 심플하고 미니멀한 컨퍼런스 연구 교육 프레젠테이션

https://www.canva.com/ko_kr/templates/EAFlIiinzHk/

#이미지 출처

1.산나비 인게임 내 이미지 사용

https://store.steampowered.com/app/1562700/SANABI/

https://twitter.com/WonderPotion

#PPT Font 출처

소제목**(Pretendard ExtraBold, 16pt)**

본문 (Pretendard, 12pt)

(Nanum Gothic Bold, 44pt)

DOWNLOAD : Pretendard

감사합니다!

질문이 있으신가요?