Unity Certified Associate 자격증 취득반 7회차

김 영 득

2023. 6. 15

혈흔 효과를 생성하는 로직

MonsterCtrl스크립트를 다음과 같이 수정한다

```
// 혈흔 효과 프리팹
private GameObject bloodEffect;
```
변수 추가

```
// BloodSprayEffect 프리팹 로드
bloodEffect = Resources.Load<GameObject>("BloodSprayEffect");
```
Start함수 안에 추가

```
// 총알의 충돌 지점
Vector3 pos = coll.GetContact(0).point;
// 총알의 충돌 지점의 법선 벡터
Quaternion rot = Quaternion.LookRotation(-coll.GetContact(0).normal);
// 혈흔 효과를 생성하는 함수 호출
ShowBloodEffect(pos, rot);
```
OnCollisionEnter함수 안에 추가

```
 void ShowBloodEffect(Vector3 pos, Quaternion rot)
{
    // 혈흔 효과 생성
    GameObject blood = Instantiate<GameObject>(bloodEffect, pos, rot, monsterTr);
    Destroy(blood, 1.0f);
}
```
ShowBloodEffect 함수 추가

OnTriggerEnter콜백 함수

씬 뷰에서 물리적인 충돌 여부를 확인했으면 PlayerCtrl스크립트에 충돌 콜백 함수를 추가한다
몬스터의 양손에 추가한 Sphere Collider에서 Is Trigger속성에 체크했기 때문에 OnCollision~계열의 함수 대신에 OnTrigger~ 계열의 콜백 함수가 호출된다

PlayerCtrl스크립트

변수 추가

```
// 초기 생명 값
private readonly float initHp = 100.0f;
// 현재 생명 값
public float currHp;
```

Start 함수에 추가

```
// HP 초기화
currHp = initHp;
```

OnTriggerEnter 함수 추가

```
// 충돌한 Collider의 IsTrigger 옵션이 체크됐을 때 발생
void OnTriggerEnter(Collider coll)
{
    // 충돌한 Collider가 몬스터의 PUNCH이면 Player의 HP 차감
    if (currHp >= 0.0f && coll.CompareTag("PUNCH"))
    {
        currHp -= 10.0f;
        Debug.Log($"Player hp = {currHp / initHp}");
        // Player의 생명이 0 이하이면 사망 처리
        if (currHp <= 0.0f)
        {
            PlayerDie();
        }
    }
}
```

PlayerDie 함수 추가

```
// Player의 사망 처리
void PlayerDie()
{
    Debug.Log("Player Die !");
}
```

몬스터 공격 중지

Player가 사망했을 때 호출되는 PlayerCtrl스크립트의 PlayerDie함수에서 모든 몬스터를 찾아 공격중지 함수를 호출하는 소스를 다음과 같이 추가한다

```
void PlayerDie()
    {
        Debug.Log("Player Die !");

        // // MONSTER 태그를 가진 모든 게임오브젝트를 찾아옴
        GameObject[] monsters = GameObject.FindGameObjectsWithTag("MONSTER");

        // // 모든 몬스터의 OnPlayerDie 함수를 순차적으로 호출
        foreach (GameObject monster in monsters)
        {
            monster.SendMessage("OnPlayerDie", SendMessageOptions.DontRequireReceiver);
        }
    }
```

PlayerDie함수에 추가한 내용은 씬 뷰에 있는 모든 몬스터를 태그로 검색한 후 배열에 저장한다
그다음 foreach구문을 사용해 배열의 처음부터 끝까지 순회하면서 OnPlayerDie함수를 호출한다
SendMessage함수는 첫 번째 인자로 전달한 함수명과 동일한 함수가 해당 게임오브젝트의 스크립트에 있다면 실행하라는 명령이다
여러 게임오브젝트의 함수를 호출하는 로직에 사용하면 효율적이다
두 번째 인자를 SendMessageOptions.DontRequireReceiver로 설정하면 호출한 함수가 없더라도 함수가 없다는 메시지를 반환받지 않겠다는 옵션이다
따라서 빠른 실행을 위해 반드시 이 옵션을 사용한다
위 로직에서 호출할 OnPlayerDie함수는 MonsterCtrl스크립트에 아직 정의하지 않았다
MonsterCtrl에 이 함수를 정의하고 몬스터가 공격 루틴이나 추적 루틴에서 벗어나도록 모든 코루틴 함수를 정지시키는 로직을 작성해보자

몬스터 공격 중지

MonsterCtrl스크립트의 선언부에 애니메이터 해시값을 저장할 변수를 지정하고 맨 마지막에 다음 함수를 추가한다

```
private readonly int hashPlayerDie = Animator.StringToHash("PlayerDie");
```

변수 추가

```
void OnPlayerDie()
{
    // 몬스터의 상태를 체크하는 코루틴 함수를 모두 정지시킴
    StopAllCoroutines();

    // 추적을 정지하고 애니메이션을 수행
    agent.isStopped = true;
    anim.SetTrigger(hashPlayerDie);
}
```

함수 추가

몬스터 공격 중지

MonsterCtrl스크립트의 선언부에 Speed파라미터에 대한 해시값을 추출하고 OnPlayerDie함수를 다음과 같이 수정한다

```
private readonly int hashSpeed = Animator.StringToHash("Speed");
```

변수 추가

```
void OnPlayerDie()
{
    // 몬스터의 상태를 체크하는 코루틴 함수를 모두 정지시킴
    StopAllCoroutines();

    // 추적을 정지하고 애니메이션을 수행
    agent.isStopped = true;
    anim.SetFloat(hashSpeed, Random.Range(0.8f, 1.2f)); // 스피드를 위해서 추가해야 하는 코드
    anim.SetTrigger(hashPlayerDie);
}
```

코드 추가

주인공의 사망 이벤트 처리

PlayerCtrl스크립트의 선언부에 델리게이트와 이벤트 함수를 선언하고 OnTriggerEnter 함수를 다음 스크립트와 같이 수정한다
PlayerDie 함수에서 현재 생성된 모든 몬스터를 태그로 찾아와 배열에 담고 그 배열을 순회하면서 SendMessage 함수로 메시지를 전달할 필요가 없다
간단히 이벤트만 호출하면 된다
PlayerDie 함수의 기존 코드는 모두 주석 처리한다

변수 추가

```
// 델리게이트 선언
public delegate void PlayerDieHandler();
// 이벤트 선언
public static event PlayerDieHandler OnPlayerDie;
```

함수 처리

```
// Player의 사망 처리
    void PlayerDie()
    {
        Debug.Log("Player Die !");

        // // MONSTER 태그를 가진 모든 게임오브젝트를 찾아옴
        // GameObject[] monsters = GameObject.FindGameObjectsWithTag("MONSTER");

        // // 모든 몬스터의 OnPlayerDie 함수를 순차적으로 호출
        // foreach (GameObject monster in monsters)
        // {
        //     monster.SendMessage("OnPlayerDie", SendMessageOptions.DontRequireReceiver);
        // }

        // 주인공 사망 이벤트 호출(발생)
        OnPlayerDie();
    }
```

이벤트를 정의하려면 델리게이트를 이용해 이벤트 함수의 원형을 선언해야 한다

이벤트는 언제 호출될지 모르기 때문에 정적 변수로 선언해야 한다
따라서 static 키워드를 사용하며 원형은 다음과 같다

Public static event 델리게이트명 이벤트명;

OnPlayerDie는 이벤트명이라고 하지만 변수의 일종이며 PlayerDieHandler는 OnPlayerDie변수의 타입일 뿐이다
즉, 다음과 같이 해석하면 이해가 빠를 것이다

Public static event 변수타입 변수명;

주인공의 사망 이벤트 처리

이벤트를 선언하고 발생시킬 로직을 완성했으므로 MonsterCtrl에서 발생하는 이벤트에 반응할 특정 함수를 연결한다
이벤트는 반드시 스크립트의 활성화 시점에 연결하고, 비활성화될 때 해제해야 한다
다음과 같이 MonsterCtrl스크립트의 Start 함수 앞에 OnEnable, OnDisable 함수를 추가한다

```
// 스크립트가 활성화될 때마다 호출되는 함수
void OnEnable()
{
    // 이벤트 발생 시 수행할 함수 연결
    PlayerCtrl.OnPlayerDie += this.OnPlayerDie;
}

 // 스크립트가 비활성화될 때마다 호출되는 함수
void OnDisable()
{
    // 기존에 연결된 함수 해제
    PlayerCtrl.OnPlayerDie -= this.OnPlayerDie;
}
```

OnEnable()과 OnDisable()은 스크립트가 활성화되거나 비활성화될 때 수행되는 함수로서 이벤트의 연결과 해지는 반드시 이 함수에서 처리해야 한다

이벤트 연결과 해지는 다음과 같은 문법을 사용한다

- **이벤트 연결(이벤트가 선언된 클래스명).(이벤트명)+=(이벤트 발생 시 호출할 함수)**
- **이벤트 연결(이벤트가 선언된 클래스명).(이벤트명)-=(이벤트 발생 시 호출할 함수)**

게임을 실행해보면 이전 방식과 차이점은 느낄 수 없지만, 이벤트 구동 방식으로 전환해도 같은 로직을 구현할 수 있다

몬스터의 사망 처리

몬스터가 총에 맞았을 때의 시각적인 효과는 앞서 구현했고 이제 몬스터가 죽는 애니메이션을 완성하자
주인공 캐릭터와 마찬가지로 몬스터에 생명 변수를 지정하고 OnCollisionEnter 함수에서 hp가 0 이하인지 판단해 사망했는지 판단한다
MonsterCtrl스크립트를 다음과 같이 수정한다

```
private readonly int hashDie = Animator.StringToHash("Die");

// 몬스터 생명 변수
private int hp = 100;
```

변수 추가

IEnumerator MonsterAction() **함수에서**

```
// 사망
case State.DIE:
    isDie = true;

    // 추적 정지
    agent.isStopped = true;

    // 사망 애니메이션 실행
    anim.SetTrigger(hashDie);

    // 몬스터의 Collider 컴포넌트 비활성화
    GetComponent<CapsuleCollider>().enabled = false;
break;
```

void OnCollisionEnter(Collision coll) **함수에서**

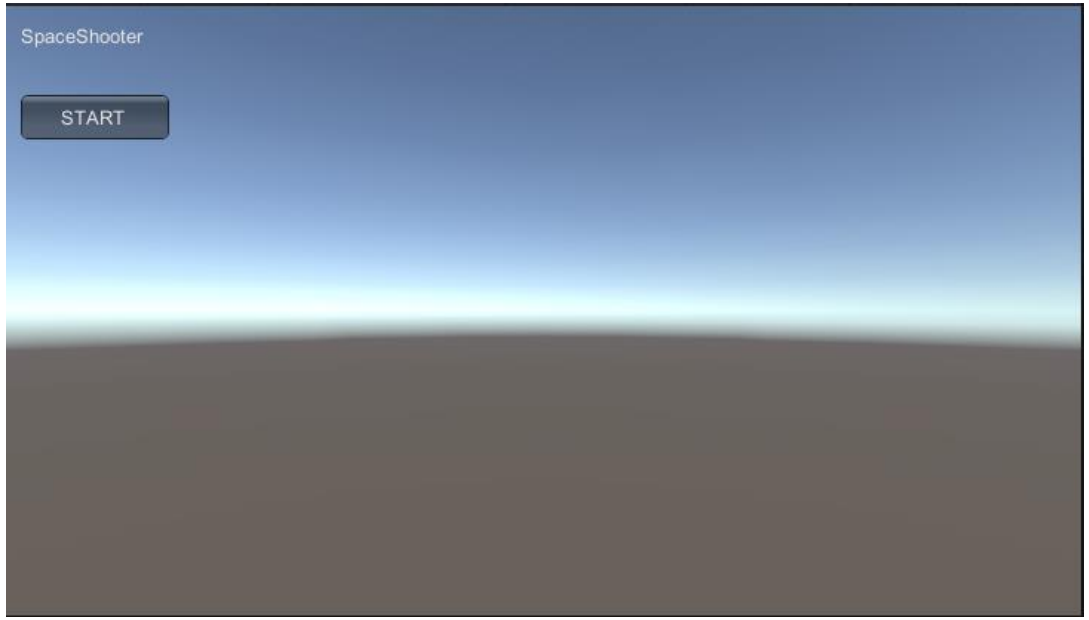
if (coll.collider.CompareTag("BULLET")) **안에다가**

```
// 몬스터의 hp 차감
hp -= 10;
if (hp <= 0)                    <- 추가
{
    state = State.DIE;
}
```

IMGUI

IMGUI(Immediate Made GUI)는 코드를 이용해 UI를 표시하는 방법으로 개발 과정에서 간단한 테스트용으로 사용한다
IMGUI는 OnGUI 함수에서 코드를 구현한다

```
using UnityEngine;

public class IMGUIDemo : MonoBehaviour
{
    private void OnGUI()
    {
        GUI.Label(new Rect(10, 10, 200, 50), "SpaceShooter");

        if(GUI.Button(new Rect(10,60,100,30),"START"))
        {
            Debug.Log("START Button clicked");
        }
    }
}
```



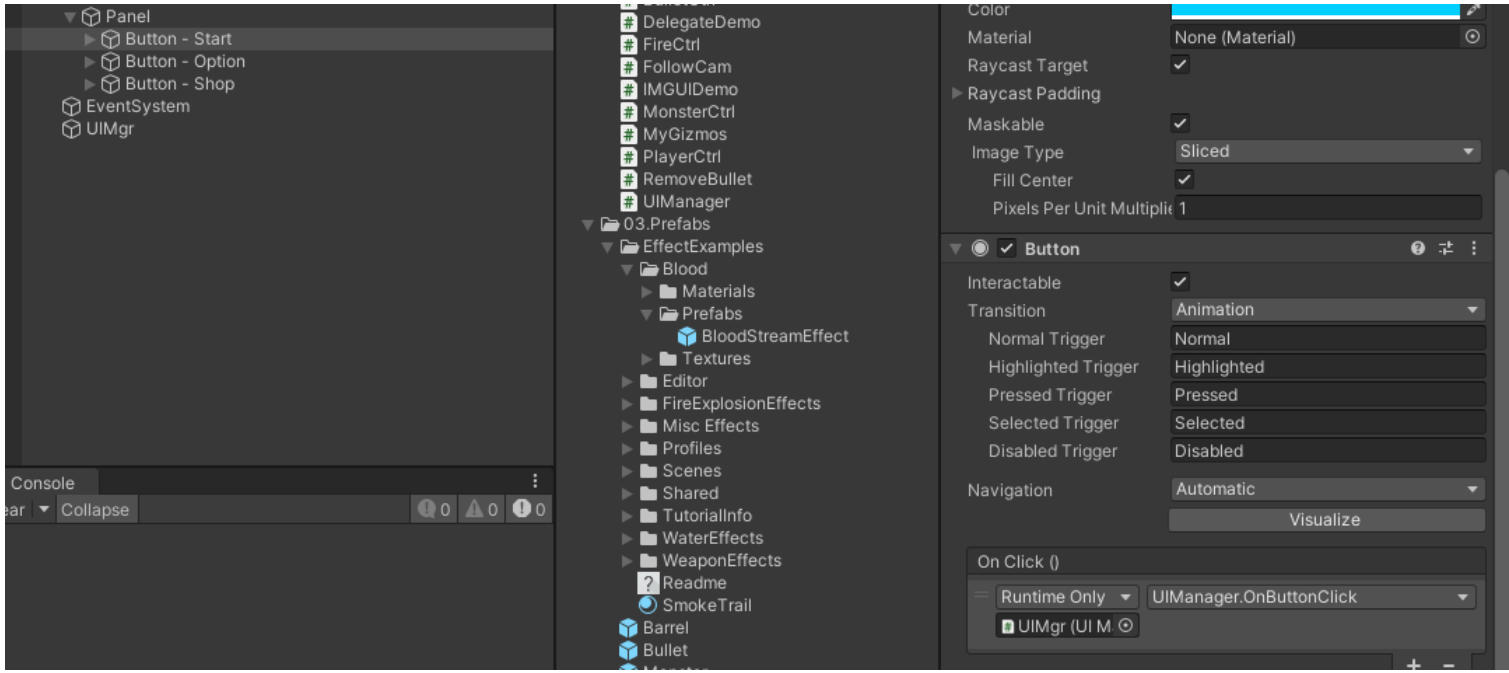
위 코드를 실행하면 오른쪽과 같이 게임뷰에 표시된다
버튼을 클릭하면 콘솔 뷰에 메시지가 출력되는 것을 볼 수 있다
OnGUI함수는 Update함수와 같이 매 프레임 발생하기 때문에 테스트가 끝난 후에는 반드시 제거해야 한다

IMGUI의 자세한 사용법은 다음 매뉴얼을 참조한다

- **https://docs.unity3d.com/kr/2021.1/Manual/gui-Basics.html**

Button Event

```
public class UIManager : MonoBehaviour
{
    public void OnButtonClick()
    {
        Debug.Log("Click Button");
    }
}
```



하이러키 뷰에 있는 UIMgr게임오브젝트를 Button-Start의 OnClick 이벤트의 Object항목에 연결하면 function목록에 GameObject, Transform, UIManager가 노출되며 연결할 수 있는 함수(메서드)목록이 펼쳐진다
그 중에서 UIManager -> OnButtonClick()을 선택하면 버튼 클릭 이벤트가 호출됐을 때 실행할 함수가 연결된 것이다
즉, Button-Start를 클릭하면 UIManager스크립트에 있는 OnButtonClick함수가 호출된다
게임을 실행해 StartButton을 클릭하면 OnButtonClick()함수가 호출되면서 콘솔 뷰에 다음과 같이 로그가 출력된다

스크립트에서 버튼 이벤트 처리하기

```
using UnityEngine;
using UnityEngine.UI;       // Unity-UI를 사용하기 위해 선언한 네임스페이스
using UnityEngine.Events;   // UnityEvent 관련 API를 사용하기 위해 선언한 네임스페이스

public class UIManager : MonoBehaviour
{
    // 버튼을 연결할 변수
    public Button startButton;
    public Button optionButton;
    public Button shopButton;

    private UnityAction action;

    void Start()
    {
        // UnityAction을 사용한 이벤트 연결 방식
        action = () => OnButtonClick(startButton.name);
        startButton.onClick.AddListener(action);

        // 무명 메서드를 활용한 이벤트 연결 방식
        optionButton.onClick.AddListener(delegate { OnButtonClick(optionButton.name); });

        // 람다식을 활용한 이벤트 연결 방식
        shopButton.onClick.AddListener(() => OnButtonClick(shopButton.name));
    }

    public void OnButtonClick(string msg)
    {
        Debug.Log($"Click Button : {msg}");
    }
}
```

먼저 2개의 네임스페이스를 추가한다
UnityEngine.UI는 스크립트에서 Unity UI 관련 컴포넌트에 접근하기 위해 명시하며
UnityEngine.Event는 UnityEvents에 관련된 API를 사용하기 위해 명시한다

특정 이벤트가 발생하면 호출할 함수를 연결하기 위해 AddListener(UnityAction call)함수를 사용한다
Button에는 OnClick이라는 버튼 이벤트 가 정의돼 있다
AddListener를 사용해 함수를 연결하는 방법은 UnityAction을 사용하거나 델리게이트를 사용하는 것이다
또한, 델리게이트를 람다식으로 표현할 수 있다

세 개의 버튼을 서로 다른 방식으로 구현했다

스크립트에서 버튼 이벤트 처리하기

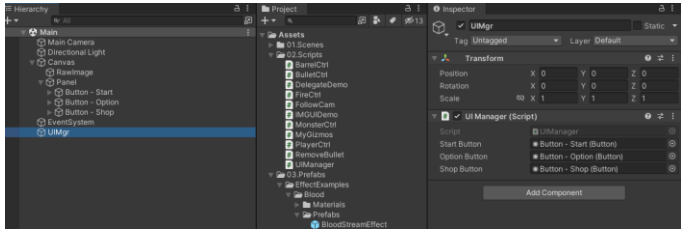
```
using UnityEngine;
using UnityEngine.UI;        // Unity-UI를 사용하기 위해 선언한 네임스페이스
using UnityEngine.Events;    // UnityEvent 관련 API를 사용하기 위해 선언한 네임스페이스

public class UIManager : MonoBehaviour
{
    // 버튼을 연결할 변수
    public Button startButton;
    public Button optionButton;
    public Button shopButton;

    private UnityAction action;

    void Start()
    {
        // UnityAction을 사용한 이벤트 연결 방식
        action = () => OnButtonClick(startButton.name);
        startButton.onClick.AddListener(action);

        // 무명 메서드를 활용한 이벤트 연결 방식
        optionButton.onClick.AddListener(delegate { OnButtonClick(optionButton.name); });

        // 람다식을 활용한 이벤트 연결 방식
        shopButton.onClick.AddListener(() => OnButtonClick(shopButton.name));
    }

    public void OnButtonClick(string msg)
    {
        Debug.Log($"Click Button : {msg}");
    }
}
```

AddListener인자로 사용할 action변수는 UnityAction타입으로 선언했다 UnityAction타입은 델리게이트로 정의돼 있으며 람다식 문법을 사용할 수 있다

두 번째 버튼은 무명 메서드 문법을 사용했다 무명 메서드는 델리게이트를 선언하고 델리게이트 타입으로 선언한 변수 없이 바로 사용할 수 있는 문법이다

세 번째 버튼은 무명 메서드 방식을 람다식 문법으로 사용한 경우로 무명 메서드 방식을 간략화한 방식이다

하이러키 뷰의 UIMgr을 선택하고 인스펙터 뷰에 노출된 3개의 버튼 속성에 Canvas -> Panel 하위에 있는 버튼을 각각 연결한다



생명 게이지 구현

생명 게이지에 대한 UI 작업이 완료됐다
이제 주인공의 생명 수치가 감소함에 따라 Fill Amount 속성을 수정해보자
PlayerCtrl스크립트를 다음과 같이 수정한다

```
// Hpbar 연결할 변수
private Image hpBar;
```

Start 함수 안에 추가

```
// Hpbar 연결
hpBar = GameObject.FindGameObjectWithTag("HP_BAR")?.GetComponent<Image>();
DisplayHealth();
```

OnTriggerEnter함수 안에 추가

```
DisplayHealth();
```

DisplayHealth 함수 추가

```
void DisplayHealth()
{
    hpBar.fillAmount = currHp / initHp;
}
```

전체 코드

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

public class PlayerCtrl : MonoBehaviour
{
    // 컴포넌트를 처리할 변수
    private Transform tr;

    // Animation 컴포넌트를 저장할 변수
    private Animation anim;

    // 이동 속력 변수 (public으로 선언되어 인스펙터 뷰에 노출됨)
    public float moveSpeed = 10.0f;

    // 회전 속도 변수
    public float turnSpeed = 80.0f;


    // 초기 생명 값
    private readonly float initHp = 100.0f;
    // 현재 생명 값
    public float currHp;

    // Hpbar 연결할 변수
    private Image hpBar;

    // 델리게이트 선언
    public delegate void PlayerDieHandler();
    // 이벤트 선언
    public static event PlayerDieHandler OnPlayerDie;

    // Start is called before the first frame update
    IEnumerator Start()
    {
        // Hpbar 연결
        hpBar = GameObject.FindGameObjectWithTag("HP_BAR")?.GetComponent<Image>();
        // HP 초기화
        currHp = initHp;
        DisplayHealth();

        // 컴포넌트를 추출해 변수에 대입
        tr = GetComponent<Transform>();
        anim = GetComponent<Animation>();

        // 애니메이션 실행
        anim.Play("Idle");

        turnSpeed = 0.0f;
        yield return new WaitForSeconds(0.3f);
        turnSpeed = 80.0f;
    }


    // Update is called once per frame
    void Update()
    {
        float h = Input.GetAxis("Horizontal");      // -1.0f ~ 0.0f ~ +1.0f
        float v = Input.GetAxis("Vertical");        // -1.0f ~ 0.0f ~ +1.0f
        float r = Input.GetAxis("Mouse X");

        // 전후좌우 이동 방향 벡터 계산
        Vector3 moveDir = (Vector3.forward * v) + (Vector3.right * h);

        // Translate(이동 방향 * 속력 * Time.deltaTime)
        tr.Translate(moveDir.normalized * Time.deltaTime * moveSpeed);

        // Vector3.up 축을 기준으로 turnSpeed만큼의 속도로 회전
        tr.Rotate(Vector3.up * turnSpeed * Time.deltaTime * r);

        // 주인공 캐릭터의 애니메이션 설정
        PlayerAnim(h, v);
    }


    void PlayerAnim(float h, float v)
    {
        // 키보드 입력값을 기준으로 동작할 애니메이션 수행
        if (v >= 0.1f)
        {
            anim.CrossFade("RunF", 0.25f);  // 전진 애니메이션 실행
        }
        else if (v <= -0.1f)
        {
            anim.CrossFade("RunB", 0.25f);  // 후진 애니메이션 실행
        }
        else if (h >= 0.1f)
        {
            anim.CrossFade("RunR", 0.25f);  // 오른쪽 이동 애니메이션 실행
        }
        else if (h <= -0.1f)
        {
            anim.CrossFade("RunL", 0.25f);  // 왼쪽 이동 애니메이션 실행
        }
        else
        {
            anim.CrossFade("Idle", 0.25f);   // 정지 시 Idle 애니메이션 실행
        }
    }

    // 충돌한 Collider의 IsTrigger 옵션이 체크됐을 때 발생
    void OnTriggerEnter(Collider coll)
    {
        // 충돌한 Collider가 몬스터의 PUNCH이면 Player의 HP 차감
        if (currHp >= 0.0f && coll.CompareTag("PUNCH"))
        {
            currHp -= 10.0f;

            DisplayHealth();

            Debug.Log($"Player hp = {currHp / initHp}");
            // Player의 생명이 0 이하이면 사망 처리
            if (currHp <= 0.0f)
            {
                PlayerDie();
            }
        }
    }

    // Player의 사망 처리
    void PlayerDie()
    {
        Debug.Log("Player Die !");
        // 주인공 사망 이벤트 호출(발생)
        OnPlayerDie();
    }

    void DisplayHealth()
    {
        hpBar.fillAmount = currHp / initHp;
    }
}
```

생명 게이지 구현

OnTriggerEnter 콜백 함수가 발생했을 때 currHp 값을 감산한 후 HpBar의 값을 변경시킨다

```
// 충돌한 Collider의 IsTrigger 옵션이 체크됐을 때 발생
void OnTriggerEnter(Collider coll)
{
    // 충돌한 Collider가 몬스터의 PUNCH이면 Player의 HP 차감
    if (currHp >= 0.0f && coll.CompareTag("PUNCH"))
    {
        currHp -= 10.0f;

        DisplayHealth();

        Debug.Log($"Player hp = {currHp / initHp}");

        // Player의 생명이 0 이하이면 사망 처리
        if (currHp <= 0.0f)
        {
            PlayerDie();
        }
    }
}
```

끝

다들 수고 했습니다