Fierce Combat

- TEAM 3 IDIOTS –

20191332 김호현
20191270 최민규
20191279 한세욱

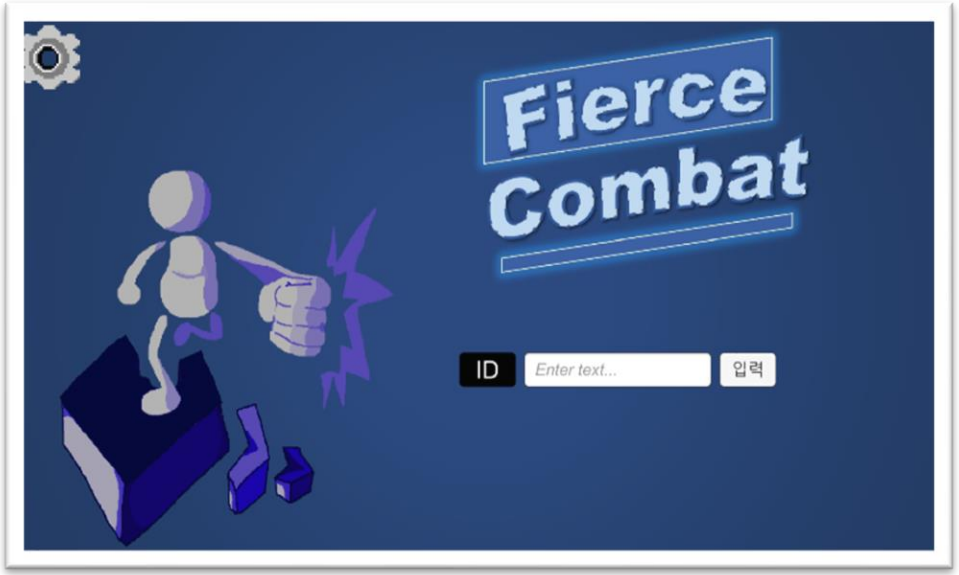
1. 프로젝트 요약

[프로젝트 명칭]

Fierce Combat (맹렬한 전투)

[게임 요약]

1:1 대전 위주의 온라인 네트워크 게임으로 다양한 기술과 스킬, 필살기를 적절히 조합하여 전투하는 게임입니다.



2. 담당 부분

기획

- 김호현

게임 규칙 및 게임 시스템 설계
전투 방식 기획
각 기능 별 기획 문서 작성
스테이지 설계

<기획 보조>
최민규, 한세욱

프로그래밍

- 최민규

사운드 설정
채팅창 구현
방 리스트 관리
로비, 대기 룸 생성 준비
게임 결과 구성

- 한세욱

HUD 작업
몬스터 AI 구현 및 동기화
스테이지 내 오브젝트 구현
게임 진행 관리 및 스테이지 관리자

- 김호현

플레이어 캐릭터 구현
전투 시스템 구현 및 동기화
스킬 아이템 구현
HUD 네트워크 동기화
카메라 연출 구현

그래픽

- 김호현

캐릭터 및 일부 배경 그래픽 작업
전투 이펙트 작업
이모티콘 그래픽 작업
2D, 3D 애니메이션 작업
타이틀, 로비 작업

3. 작업 일정

10/23~10/31	11/1~12/10	12/11~12/12	12 / 13
게임 기능 구현	게임 기능 구현	게임 테스트	최종적으로 게임 검토 및 수정
게임 리소스 제작	외부 Asset 적용 (캡스톤디자인 지원)	게임 내 버그, 문제점 수정	발표 자료 준비
회의 및 역할 분담	회의 및 역할 분담	개발 관련 회의	제출용 프로젝트 영상 준비

4. 구현 내용

<타이틀 화면>

설정

- 기본적인 구조는 기존 네트워크 게임의 구조를 가짐

  (서버 접속 -> Id 입력 -> 확인(Enter or Click) -> 로비 입장)

- 타이틀 화면에서 전체 사운드 크기를 조절할 수 있도록 구성

4. 구현 내용

<사운드 설정 원리>

```
public static float s_volume = 1;

// 현재 맵에 있는 데이터를 저장하는 용도 (맵에 배치된 오디오 리스트만 가짐 -> 초기값)
private AudioSource[] audio_Array;

// 추가되는 요소를 저장하는 요소 (전체 오디오 리스트를 가짐)
private List<AudioSource> audio_List = new List<AudioSource>();
```

```
void Start()
{
    // 맵에 배치된 오디오를 모두 배열로 데이터를 받는다.
    audio_Array = FindObjectsOfType<AudioSource>();

    // 배열로는 데이터를 추가하기 어렵기 때문에, 배열 -> List로 저장하고, 추가 오디오를 추가한다.
    foreach (AudioSource audio in audio_Array)
    {
        audio.GetComponent<AudioSource>().volume = Mingyu_SoundManager.s_volume;
        audio_List.Add(audio);
    }
```

```
// 사운드를 추가할 때, 사용 (사운드 생성 시, 사용 -> 호현이가 호출)
참조 0개
public void Add_AudioSouce(AudioSource spawnAudio)
{
    spawnAudio.GetComponent<AudioSource>().volume = Mingyu_SoundManager.s_volume;
    audio_List.Add(spawnAudio);
}
```

- 사운드 크기 변수를 static으로 선언
  -> 씬이 바뀌더라도, 값이 유지
- 각 맵에 배치하여, 시작할 때 Start 함수에서 맵에 있는 AudioSource를 배열로 받음
  -> Find...함수의 반환형이 배열
- 추가적으로 생성되는 사운드는 배열의 특성상 추가하기 어려워 List를 사용하여 추가함

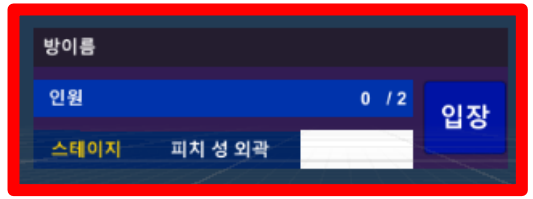
4. 구현 내용

<로비>

채팅창
로비에서 채팅을 구현하기 위해, Chat Service 서버를 동시에 구동

-> Discord 수업 때, voice와 채팅을 동시에 사용하도록 서버를 여러 개 쓰는 것에서 힌트를 얻음.

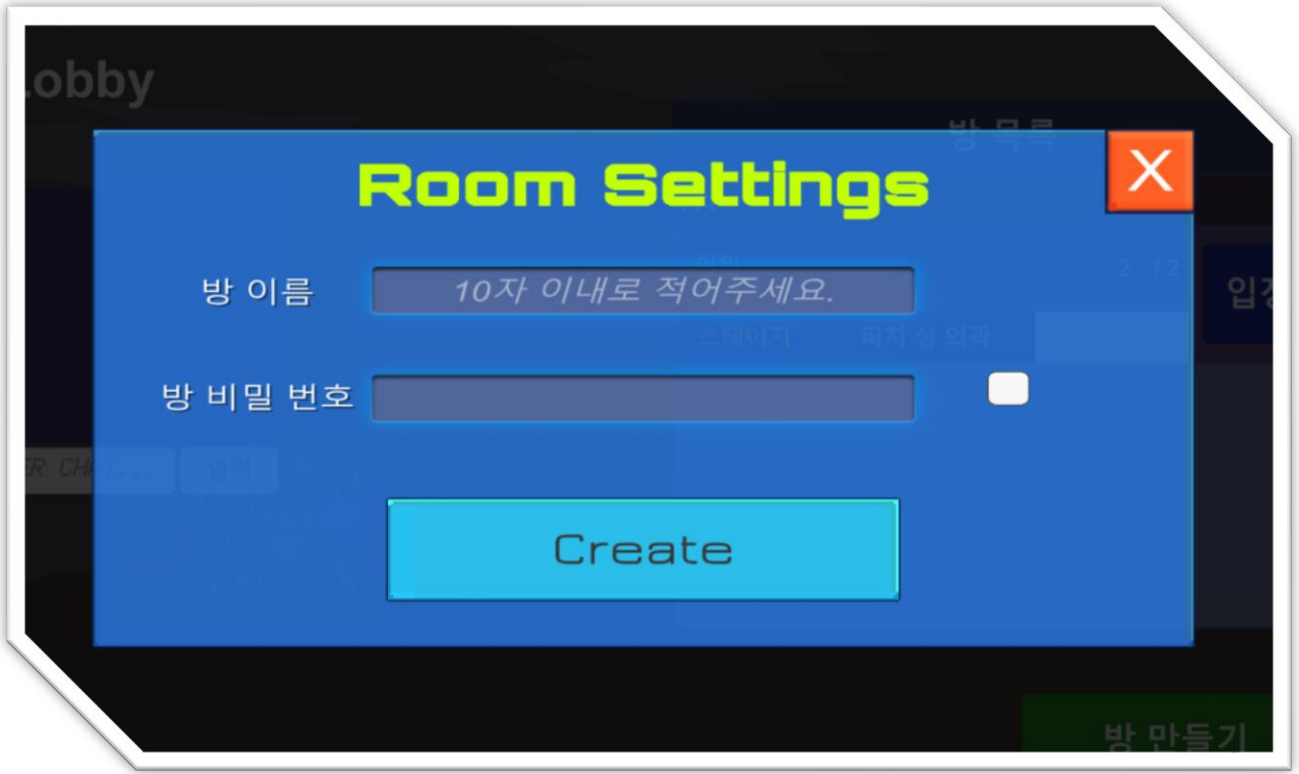
방 생성을 진행 시,
OnRoomListUpdate 함수에서 아래 프리팹을 인스턴스화 진행.

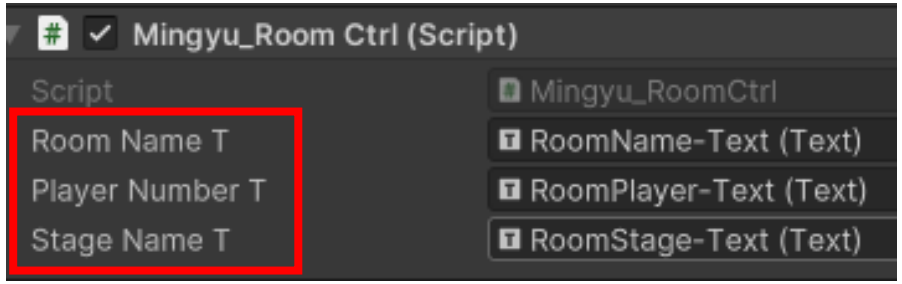


상대방이 해당 방에 들어올 때,
프리팹의 스크립트에서 Room Index값을 가져와 방에 접근할 수 있도록 구성

4. 구현 내용

<로비>

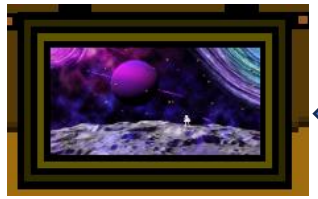




방 생성시, Mingyu_Room_Ctrl의 **변수** 값을 설정해 방 이름, 인원, 스테이지 이름을 변경할 수 있도록 구성

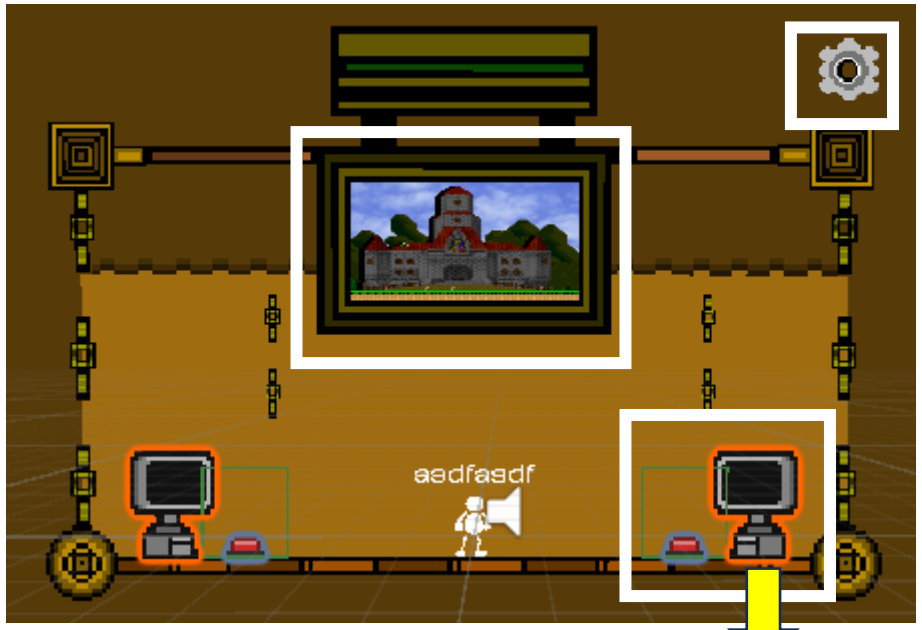
4. 구현 내용

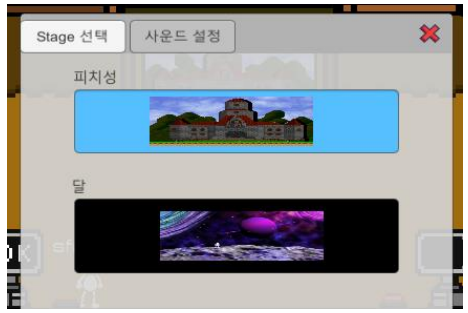
<대기 룸 생성 및 준비>



TV를 통해, 현재 선택한 맵에 정보를 알 수 있음

이는 MasterClient가 설정을 통해 변경이 가능함.





If) MasterClient가 누를 경우

else

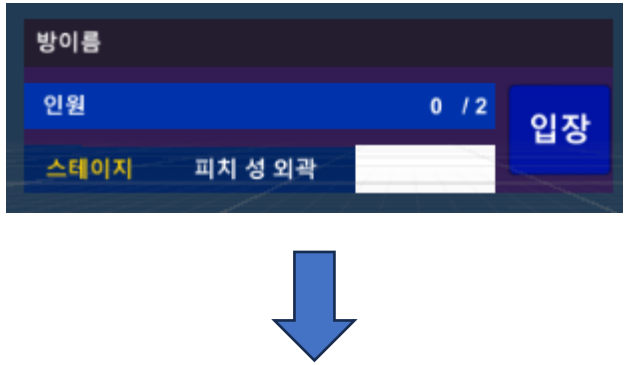
버튼의 Trigger Collider에 플레이어 태그가 닿으면, OK 스크린으로 변함

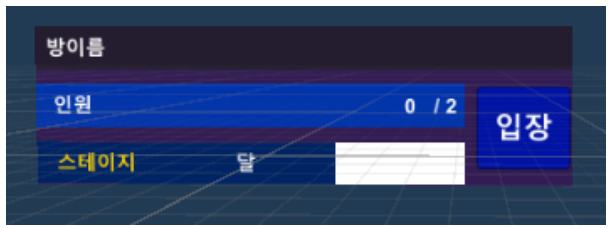
➔ 해당 두 모니터가 OK 스크린을 띄우면 게임이 실행

4. 구현 내용

<대기 룸 생성 및 준비>

```
public void BtnClick_StageSelect()
{
    if (PhotonNetwork.CurrentRoom.CustomProperties["stageName"] != null)
    {
        ExitGames.Client.Photon.Hashtable customRoomProperties = new ExitGames.Client.Photon.Hashtable();
        customRoomProperties.Add("stageName", select_StageName);
        PhotonNetwork.CurrentRoom.SetCustomProperties(customRoomProperties);
    }
}
```



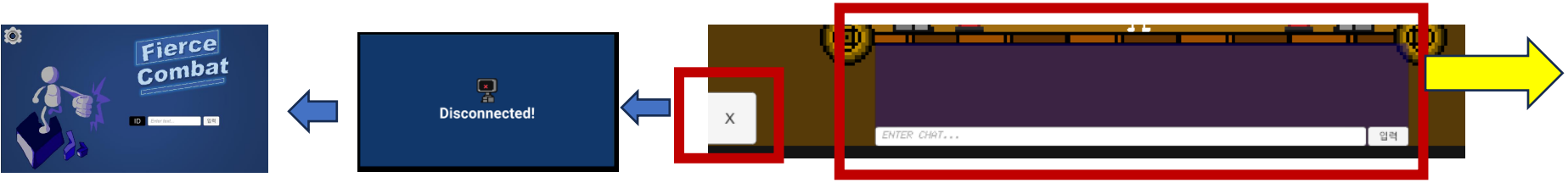


미리 Hashtable의 변수에 stageName을 추가.
해당 룸의 stage변수가 존재한다면, 값을 추가하고 CustomProperty값을 바꿔준다.

이를 통해, 해당 OnRoomListUpdate 함수를 호출하고 스테이지 이름을 바꿔준다.

4. 구현 내용

<대기 룸 생성 및 준비>



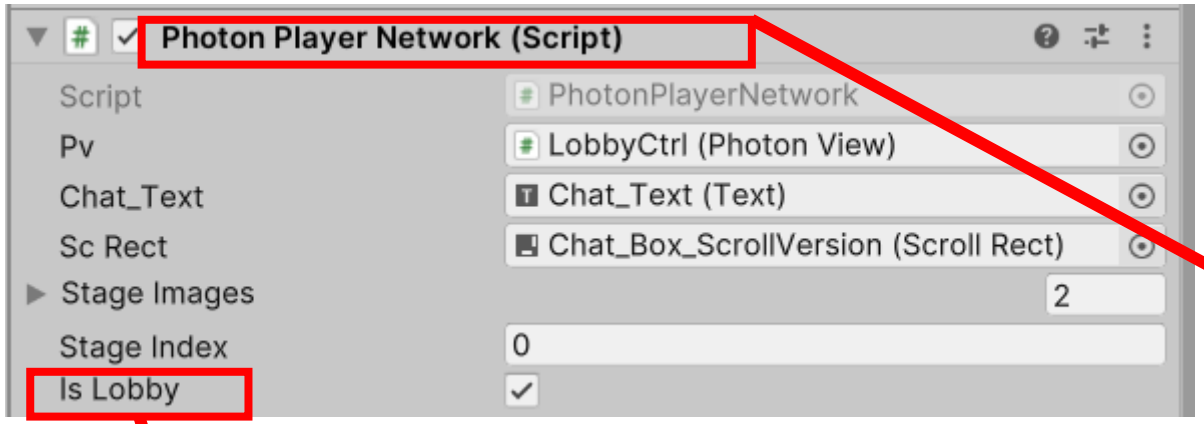
대기방에서의 채팅 구현
해당 부분은 RPC를 통해 채팅을 구현함

해당 버튼을 누르면, 연결이 끊어짐 -> 타이틀 UI로 넘어감

4. 구현 내용

<대기 룸 생성 및 준비>

- 대기룸 내 캐릭터 생성 -

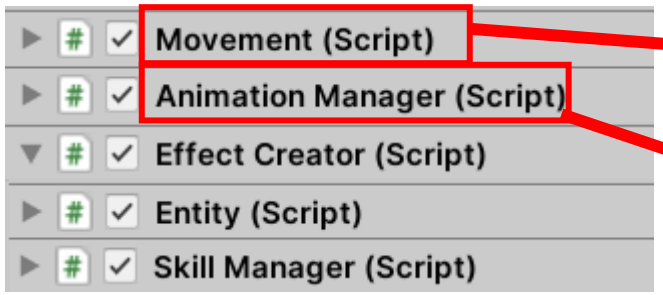


플레이어 생성 및 네트워크 룸 관리하는 클래스 사용

Is Lobby는 대기 룸에서는 true이다. 해당 변수가 대기 룸에서 별도로 대기 룸 캐릭터를 생성 가능하게 함. 대기 룸에서는 캐릭터가 공격이 불가능하고 이모티콘 사용과 간단한 이동만 가능.

4. 구현 내용

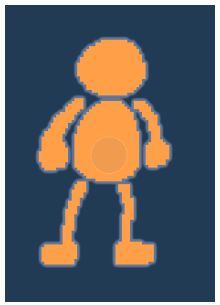
<플레이어 캐릭터>

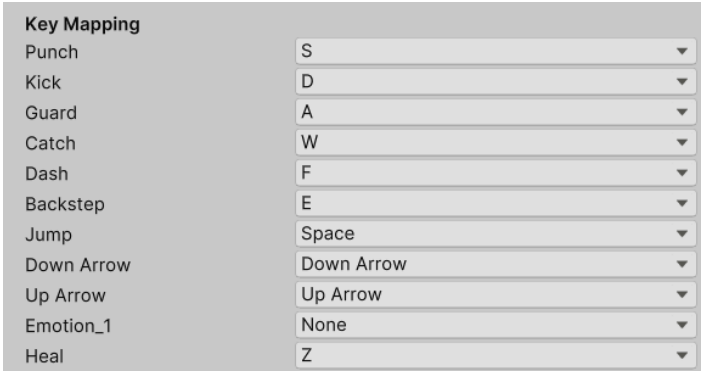


이동 제어 및 각종 힘 작용 기능을 함수로 가짐.
- 캐릭터가 밀려나거나 날라가게 하는 함수 존재

애니메이션을 제어하며, 이동을 제외한 다른 동작을 특정 입력 키에 따라 실행되도록 한다.

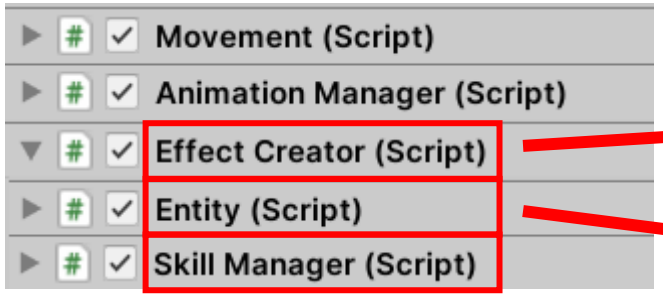
여기서 입력키는 직접 변수처럼 선택 가능해
대기 룸용 캐릭터의 경우 해당 부분의 키를 할당하지 않아 움직이게만 가능하도록 구성함.





4. 구현 내용

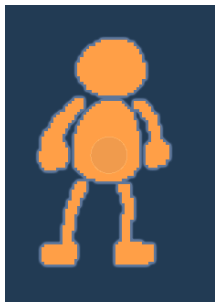
<플레이어 캐릭터>

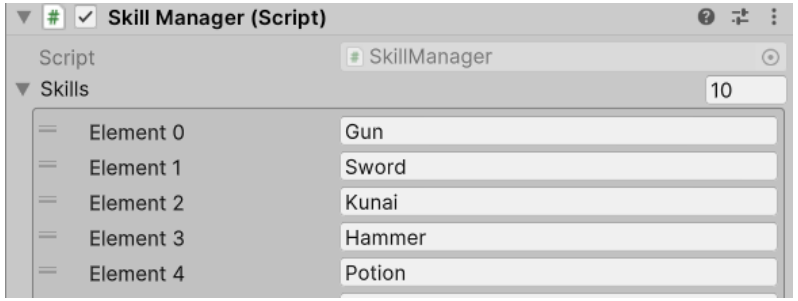


캐릭터의 상태나 동작에 따라 Effect를 생성해주는 클래스이다.

체력, MP, 데미지, 공격 처리를 담당하는 클래스

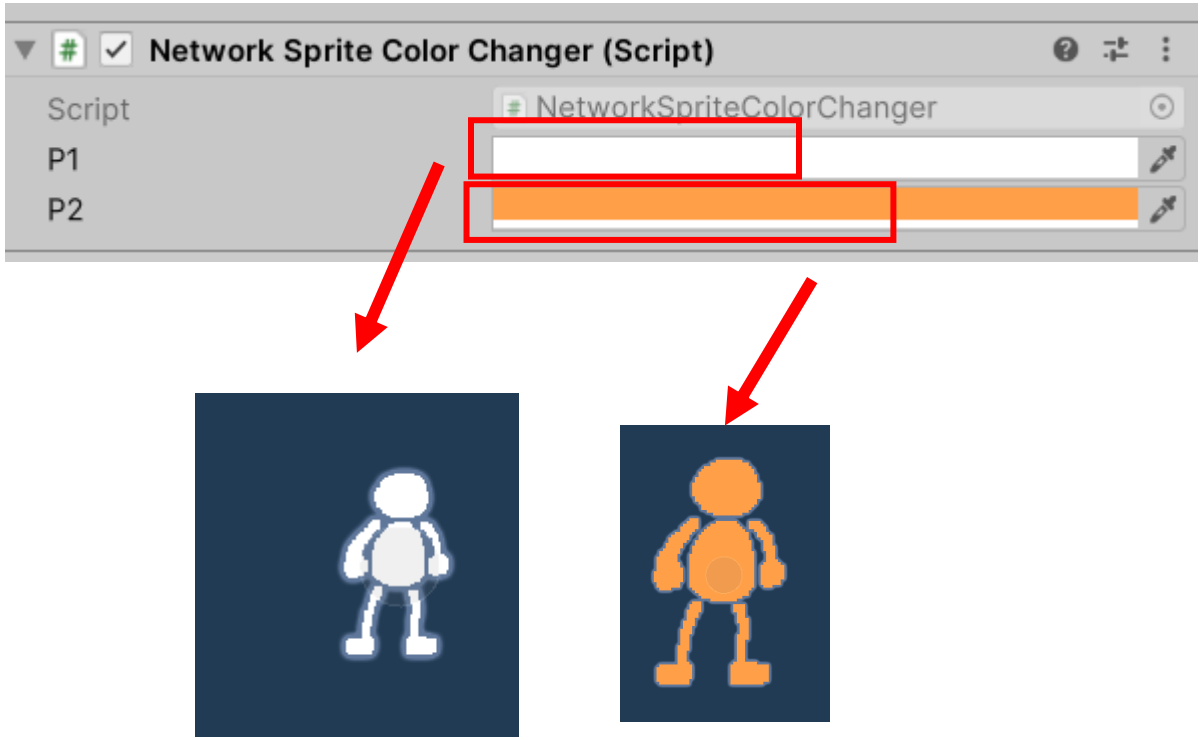
스킬 추가 삭제 기능 존재.
현재 스킬이 무엇이 있는지 저장. 필살기도 여기서 관리





4. 구현 내용

<플레이어 캐릭터>



Network Sprite Color Changer

- 플레이어가 MasterClient이면 하얀색,
- 반대의 경우 주황색으로 변경시킨다.

4. 구현 내용

<플레이어 네트워크 동기화>

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        stream.SendNext(posX);
        stream.SendNext(posY);
        stream.SendNext(rot);
        stream.SendNext(combo);
    }
    else
    {
        posX = (float)stream.ReceiveNext();
        posY = (float)stream.ReceiveNext();
        rot = (bool)stream.ReceiveNext();
        combo = (int)stream.ReceiveNext();
    }
}
```

Photon Player 클래스

플레이어를 네트워크 상에 동기화 시켜 줌.

<동기화 대상>

애니메이션 : RPC 동기화

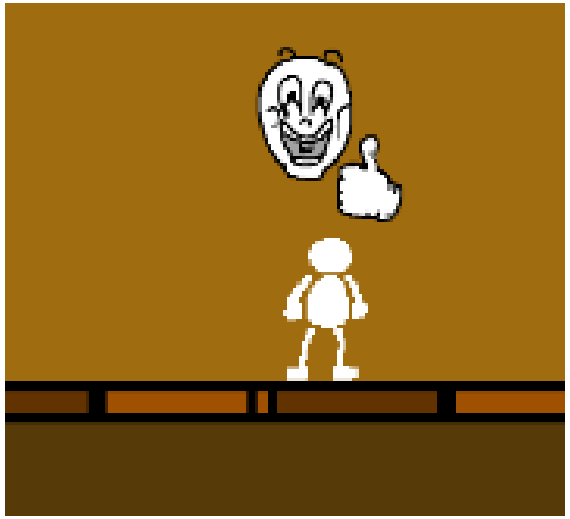
위치 값 : x,y 동기화

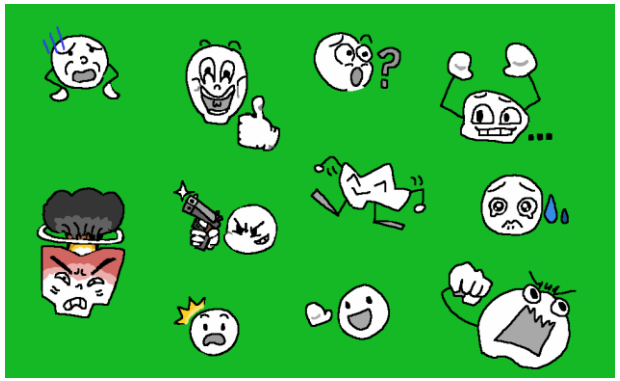
회전 값 : 좌우만 필요하므로 Bool 값으로 전달.

콤보 횟수 : 화면에 보이는 콤보 횟수 전달

4. 구현 내용

<캐릭터 이모티콘 기능>





F1 ~ F11 까지 버튼으로 각종 감정 표현이 가능하다.
해당 이모티콘은 RPC를 통해 **이모티콘 오브젝트**가 애니메이션을 실행하도록 하였다.

캐릭터의 자식 오브젝트임.

```
foreach (KeyCode key in keyValues)
{
    if (Input.GetKeyDown(key))
    {
        if ((int)key >= 282 && (int)key <= 292)
        {
            int keyValue = (int)key - 281;
            emoticon.SetValue(keyValue);
            if(network)
                network.pv.RPC("ShowEmoticon", RpcTarget.Others, keyValue);
            break;
        }
    }
}
```

F1~F11까지 버튼을 일일이 if문 처리하지 않고,
반복문으로 F1~F11까지의 키 값이 눌릴 때만
감지해 해당 키 값에 따라 처리함.

4. 구현 내용

<전투 시스템>

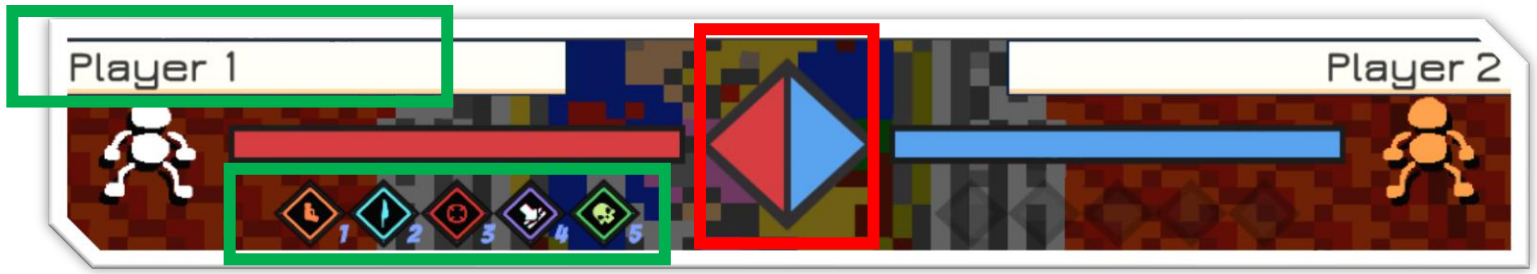
- 스테이지 관리자가 전투 상황 관리하며 체력 0인 플레이어가 패배하도록 함.
  (동점은 없음. 반드시 패자 존재)

- 각종 공격 기술로 결투가 가능. 스킬의 경우 스테이지에서 획득해야 사용 가능.
  - **스킬은 체력 바 하단에 각 스킬 사용 키(1~5)와 함께 표시됨.**

- 화면 중앙 십자가의 MP가 전부 차면 이를 전부 소모해 필살기**(스킬로 분류)**를 쓸 수 있음.

- **전투 중 체력, MP, 보유 스킬, 플레이어 닉네임에 대한 값은** NetworkManagerAll 오브젝트를 통해 **네트워크 상에서 실시간 공유됨.**

닉네임 표시



스킬 표시

MP 표시

4. 구현 내용

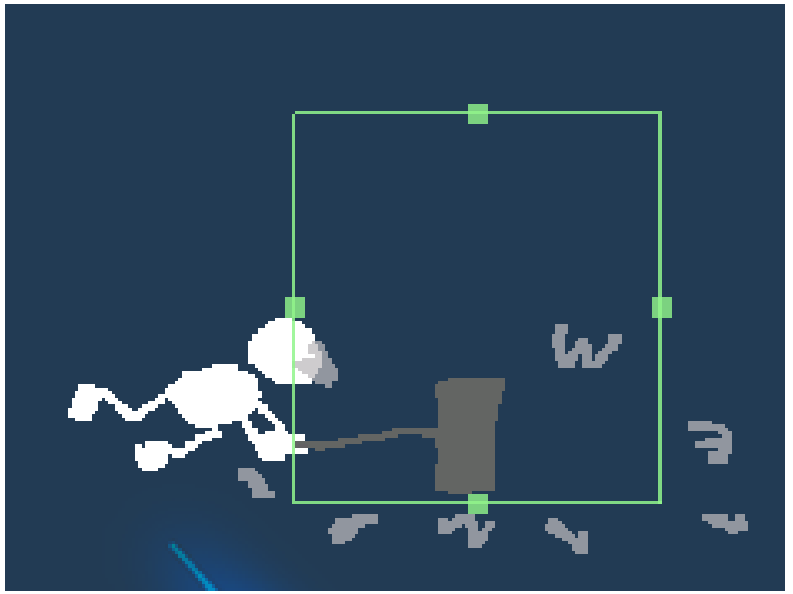
<전투 시스템>

- 전투에서의 히트 판정

```
public void Attack()
{
    Vector2 start = attackPos.transform.position;
    RaycastHit2D[] hit;
    hit = Physics2D.RaycastAll(start, transform.right, attackLength, target);

    foreach (RaycastHit2D hitTarget in hit)
    {
        if (hitTarget.collider.gameObject != gameObject)
        {
            Entity enemy = hitTarget.collider.gameObject.GetComponent<Entity>();
            if (enemy)
            {
                float temp = enemy.GetHp();
                enemy.flyingDamagedPower = flyingAttackForce;
                if (transform.localEulerAngles.y == 180)
                    enemy.Damaged(attackForce, -thrustpower);
                else
                    enemy.Damaged(attackForce, thrustpower);
                if (temp > enemy.GetHp())
                {
```

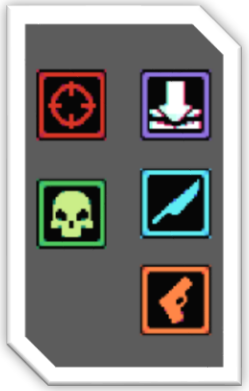
1. Raycast를 활용해 낭비되는 충돌 영역 최소화



2. 단, 일부 기술의 경우 높은 히트 판정을 위해 박스 콜라이더를 사용함.

4. 구현 내용

<스킬 시스템>



- 스킬 아이템은 5가지로 구성

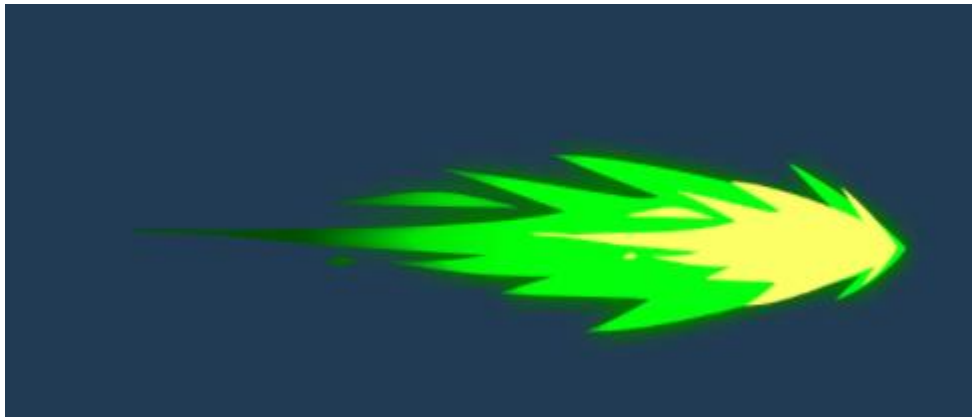
(총, 검, 망치, 수리검, 변신 물약)

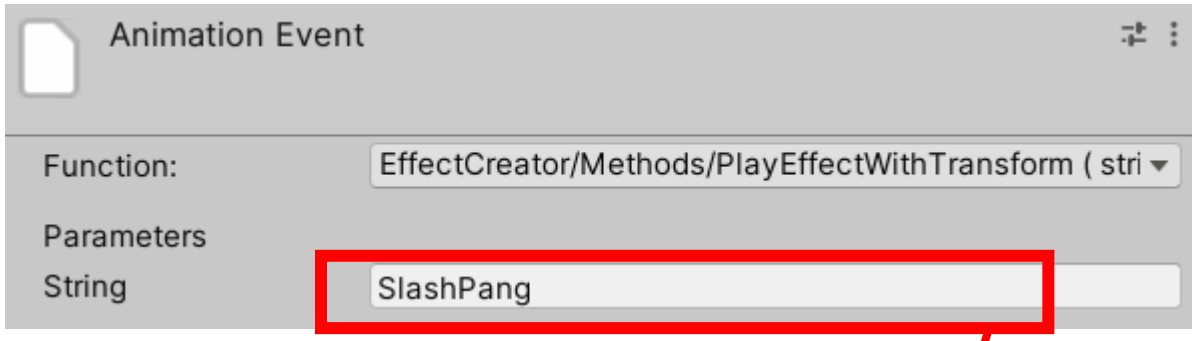
- 스킬은 맵에 배치된 스킬 생성자가 랜덤으로 PhotonNetwork.Instantiate를 통해 생성한다.

```
참조 1개
IEnumerator SpawnItem()
{
    int value = 0;
    while (true)
    {
        yield return new WaitForSeconds(5);
        if (target == null)
        {
            yield return new WaitForSeconds(3);
            value = Random.Range(0, 51);
            if(value < 11)
            {
                value = Random.Range(3, 5);
            }
            else
            {
                value = Random.Range(0, 3);
            }
            switch (value)
            {
                case 0:
                    itemName = "SKILL-ITEM (Gun)";
                        break;
                case 1:
                    itemName = "SKILL-ITEM (Sword)";
                    break;
                case 2:
                    itemName = "SKILL-ITEM (Kunai)";
                    break;
                case 3:
                    itemName = "SKILL-ITEM (Hammer)";
                    break;
                case 4:
                    itemName = "SKILL-ITEM (Potion)";
                    break;
            }
            target = PhotonNetwork.Instantiate("Item/"+ itemName, transform.position, Quaternion.identity);
        }
    }
}
```

4. 구현 내용

<스킬 시스템>





- 스킬은 대부분 기본 공격과 비슷하게 구현했으며, RPC를 통해 네트워크로 동기화하였다.
- **생성 객체의 경우 애니메이션이 재생될 때 애니메이션 이벤트를 추가해 네트워크에서 동기화되도록 함.**

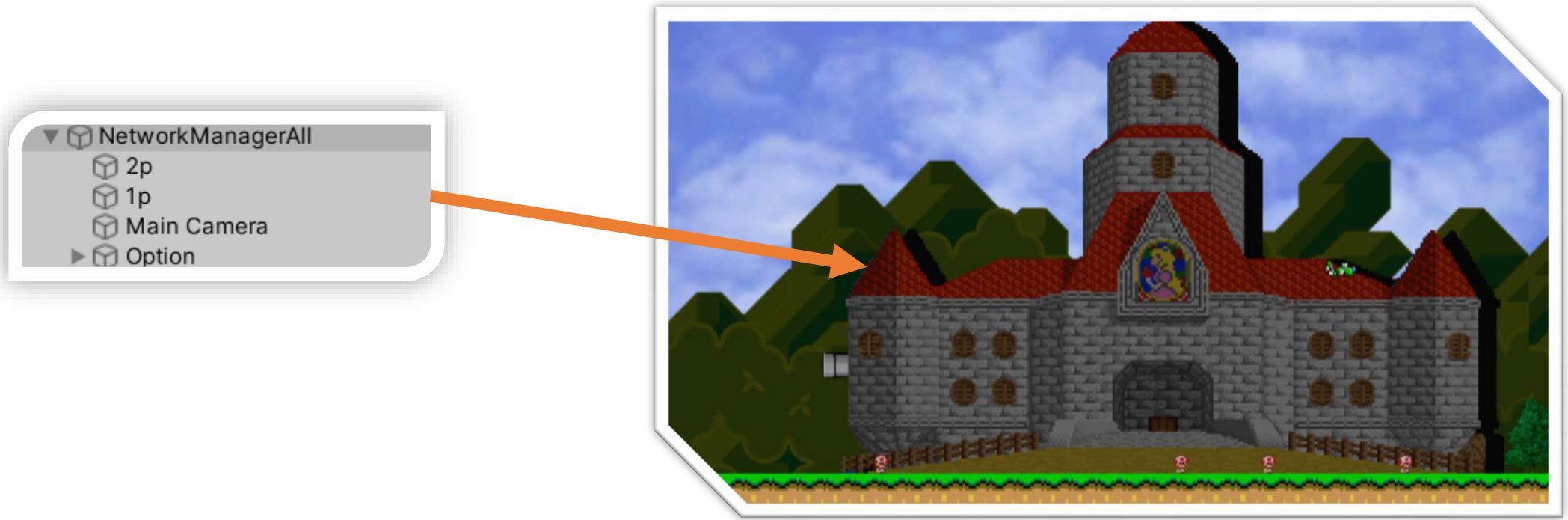
생성할 객체 프리펩 이름

4. 구현 내용

<스테이지 관리 시스템>

스테이지 네트워크 동기화용 통합형 세팅 오브젝트를 이용한 전투 스테이지 관리

- 네트워크 동기화 및 스테이지 관리 시스템을 하나의 오브젝트**(NetworkManagerAll)**에 자식으로 두어 프리펩화 가능하게 구성함.
- 해당 오브젝트 **(NetworkManagerAll)**를 다른 씬에 배치하고, 약간의 설정을 해주면 해당 씬에서도 게임 플레이 가능.

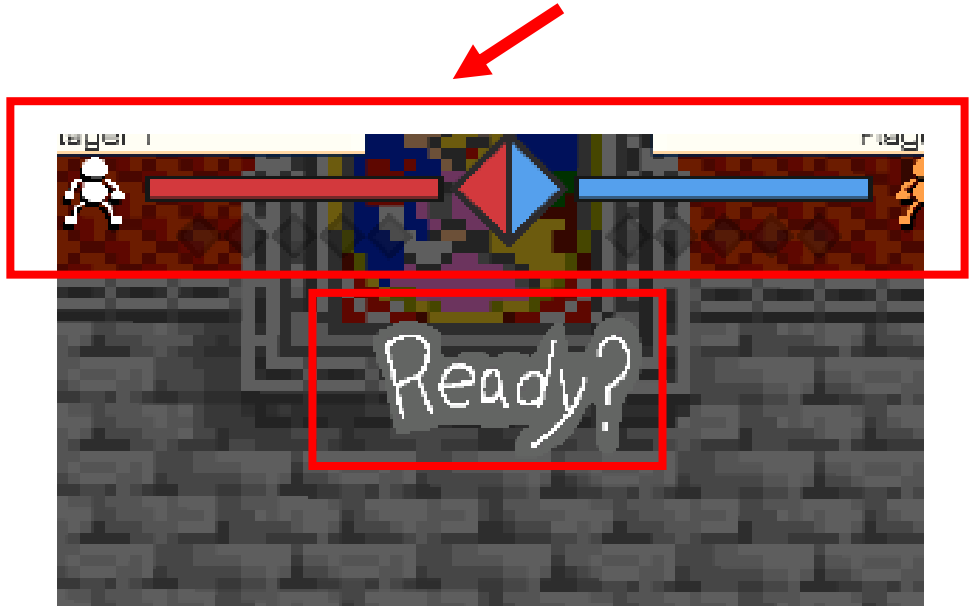


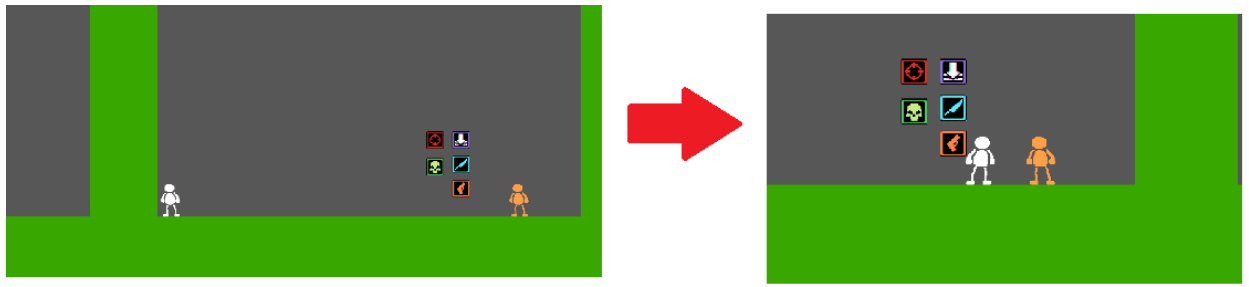
4. 구현 내용

<스테이지 관리 시스템>

스테이지 네트워크 동기화용 통합형 세팅 오브젝트를 이용한 전투 스테이지 관리

- NetworkManagerAll에 존재하는 오브젝트 및 클래스 객체
  - 카메라 2개의 타겟을 초점으로 삼고 움직이는 카메라
  - 스테이지 내 오브젝트를 관리하는 **스테이지 매니저** 객체
    => 현재 플레이어 감지
    => HUD 및 UI 객체와 이를 네트워크 동기화 시키도록 하는 클래스 객체

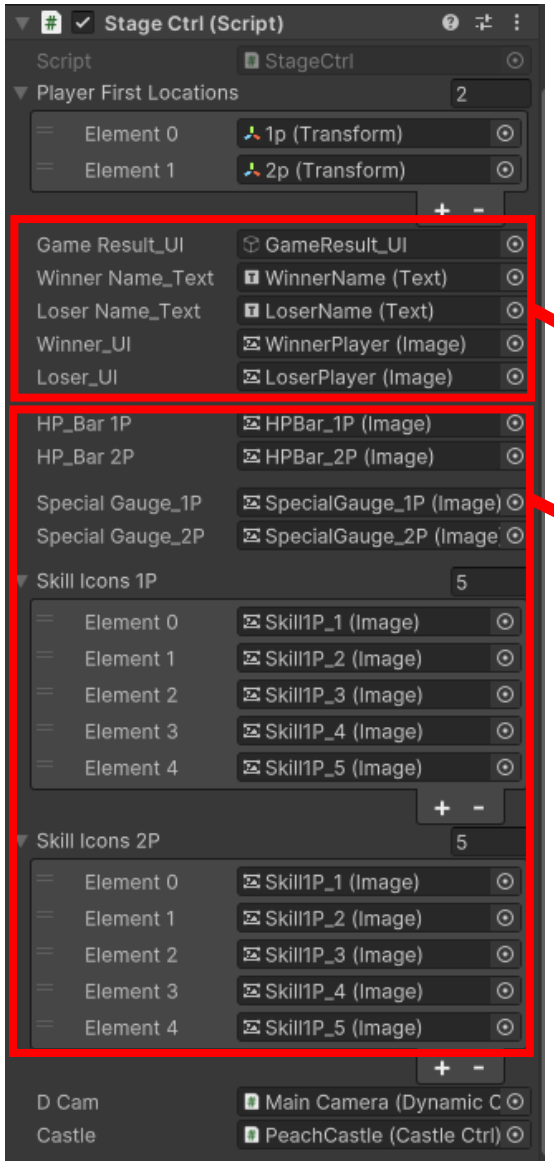




카메라
플레이어의 거리에 따라 확대 및 축소 자동으로 이뤄짐.
또한, 데미지를 입거나 스킬에 따라 화면이 흔들리거나 움직이는 효과 적용함.

4. 구현 내용

<스테이지 매니저>

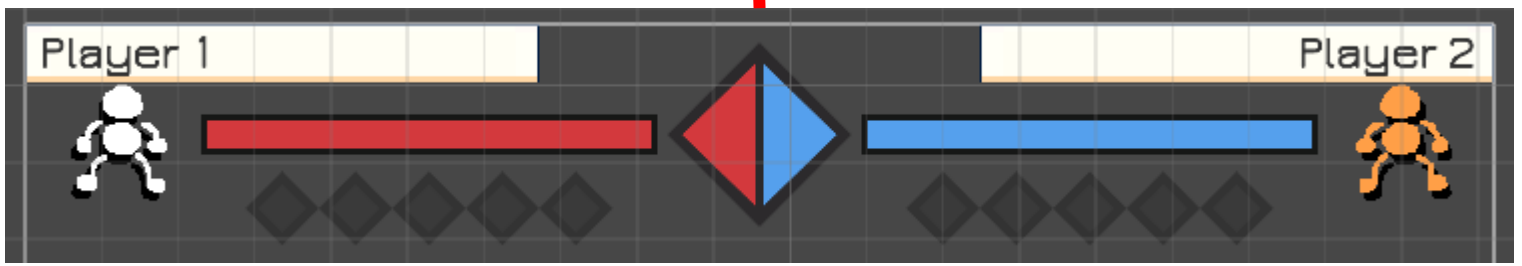


스테이지 관리자 스크립트를 스테이지 별로 배치하여 효율적으로 관리 할 수 있다.

● 스테이지 매니저의 역할

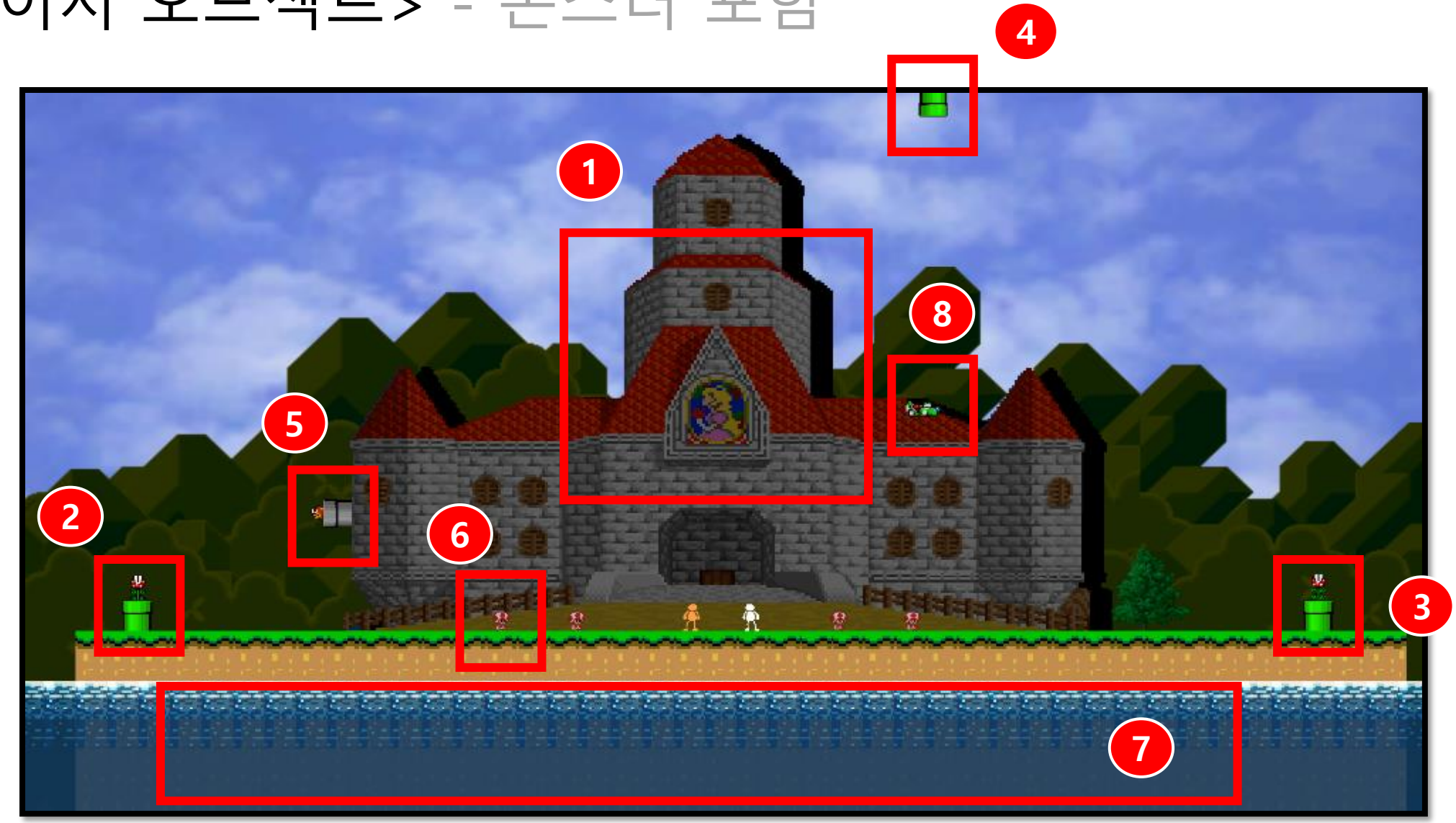
-플레이어의 배열 생성 및 관리
-플레이어의 체력 체크 및 게임의 승패 여부 확인
-게임의 승패 결과에 따라 UI 출력

-UI 관리 오브젝트 생성 및 관리
(체력, 스킬, 필살기 게이지 등)



4. 구현 내용

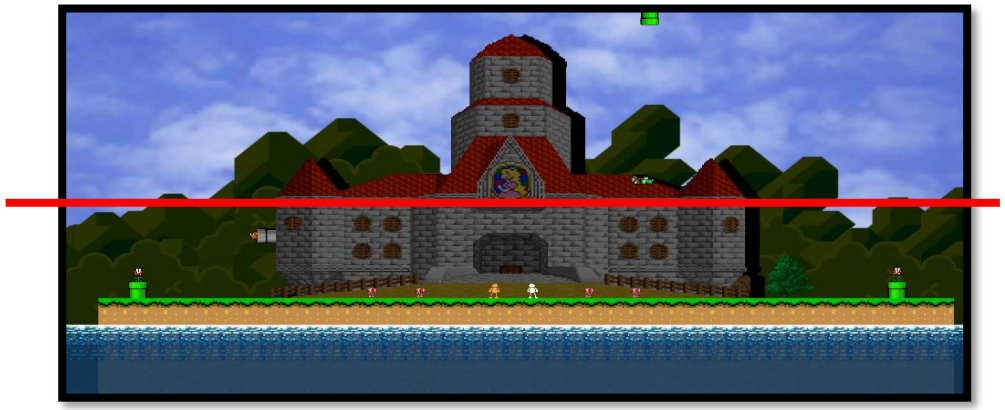
<스테이지 오브젝트> - 몬스터 포함

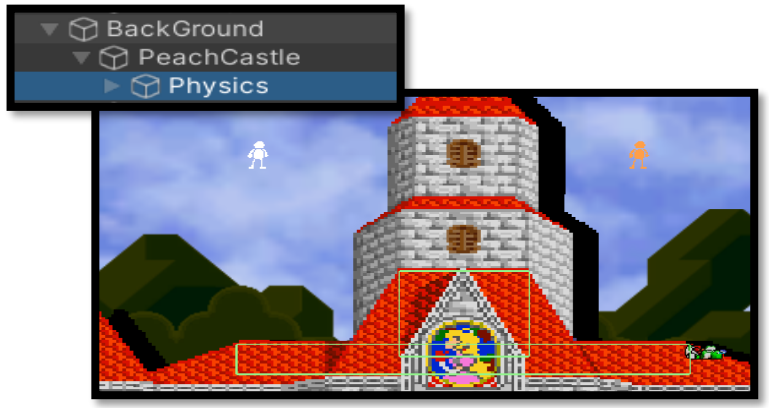


4. 구현 내용

<스테이지 오브젝트> - 몬스터 포함

1 피치성 표시 변화 구현





- 빨간 선 위로 캐릭터가 올라가면 피치 성 색이 밝게 바뀌고
콜라이더가 활성화된다.

- 반대로 빨간 선 아래로 내려가면 처음 상태로 되돌아간다.

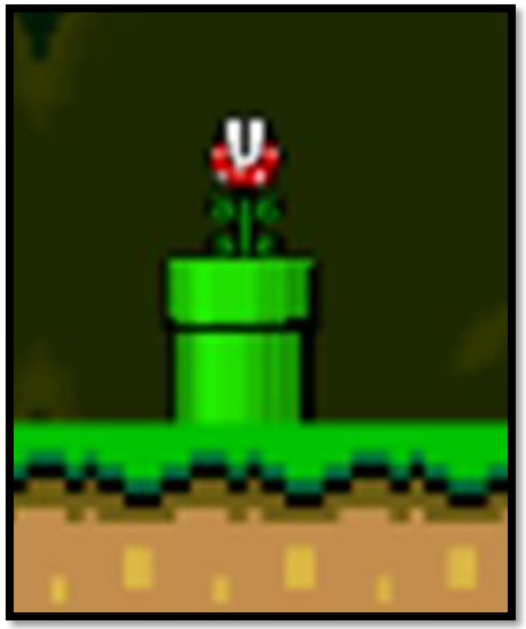
⇒ Physics라는 이름의 오브젝트를 이용했다.

4. 구현 내용

<스테이지 오브젝트> - 몬스터 포함

"배관과 토관 " + 꽃은 네트워크 동기화

2 3 입구 → 4 출구



마리오 게임과 동일하게 토관에서 꽃이 일정한 주기로 나온다.
꽃은 토관 위에 사람 없을 때만 나오며
부딪히면 데미지를 입으며 밀려 난다.

- 이 토관에 들어가면 4번 토관으로 워프 가능하다.

꽃은 PhotonView를 가지고 위치값이 동기화 된다.

4. 구현 내용

<스테이지 오브젝트> - 몬스터 포함

5 성에서 나오는 굼바 + 네트워크 동기화

토관에서 굼바가 일정하게 생성된다. **(최대 3마리)**

굼바는 앞으로 가다 벽을 만나면 다른 방향으로 움직이며
플레이어가 닿으면 데미지를 입고 굼바가 Destroy된다.

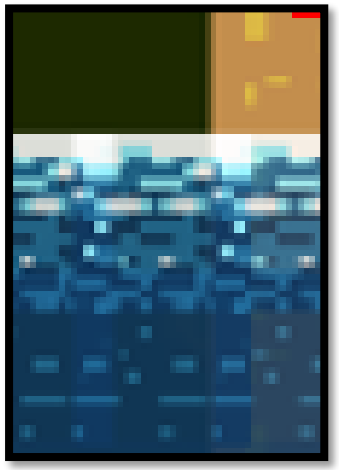
•굼바 밟으면 데미지 없이 굼바가 Destroy 되며,
플레이어가 일정 높이 점프한다.

굼바는 PhotonView를 가지고, PhotonNetwork.Instantiate로
생성되어 1P, 2P 모두에게 같이 보이고 동기화 됨

4. 구현 내용

<스테이지 오브젝트> - 몬스터 포함

7 호수, 물고기 + 물고기는 네트워크 동기화

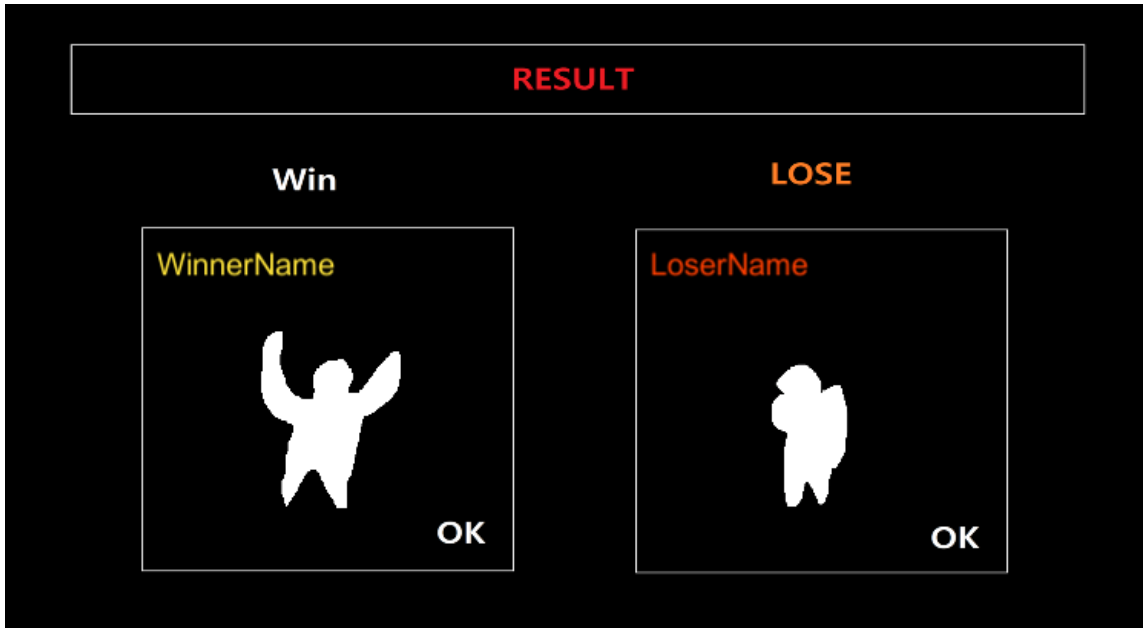


호수에선 물고기가 일정한 주기로
드물게 점프하여 데미지를 준다.

물고기 역시 PhotonView를 가지고
위치값과 회전값이 동기화 된다.

4. 구현 내용

< 게임 결과 구성 >



MasterClient Winner

User Winner

- 마스터 클라이언트는 흰색이고, 다른 유저는 주황색을 띈다.
어떤 플레이어가 이겼는지 코루틴을 통해 확인하고, 색깔과 이름을 변경해준다.

4. 구현 내용

< 게임 결과 구성 >

```
//승리 조건
if (superPlayer.isDie == true)
{
    result = GameResult.Draw;
    Debug.Log("Draw");
}
else if (PlayerList[1].GetHp() <= 0)
{
    result = GameResult.Win;
    Debug.Log(PlayerList[0].name + " Win");

    StartCoroutine(View_GameResult(PlayerList[0]));
}
else if (PlayerList[0].GetHp() <= 0)
{
    result = GameResult.Lose;
    Debug.Log(PlayerList[1].name + " Win");

    StartCoroutine(View_GameResult(PlayerList[1]));
}
```

```
private void Set_ResultData(string winnerName)
{
    if (PhotonNetwork.MasterClient.NickName == winnerName)
    {
        WinnerName_Text.text = PhotonNetwork.MasterClient.NickName;
        LoserName_Text.text = PhotonNetwork.PlayerList[1].NickName;

        Loser_UI.color = new Color(1.0f, 127 / 255.0f, 39 / 255.0f);
    }
    else
    {
        LoserName_Text.text = PhotonNetwork.MasterClient.NickName;
        WinnerName_Text.text = PhotonNetwork.PlayerList[1].NickName;

        Winner_UI.color = new Color(1.0f, 127 / 255.0f, 39 / 255.0f);
    }
}
```

코루틴을 통해, 플레이어 리스트의 체력을 검사하고, 승리한 플레이어를 넘긴다.
해당 플레이어의 PhotonView를 가져와, Owner변수값을 통해 현재 플레이어의 NickName을 전달한다.

만약, Master Client가 이겼다면, Loser의 색을 바꿔준다.
반대도 비슷한 방식으로 구성해준다.

4. 구현 내용

<기타>

미니맵 표시
- 게임 내 상황을 전체적으로 살펴 볼 수 있는 기능.

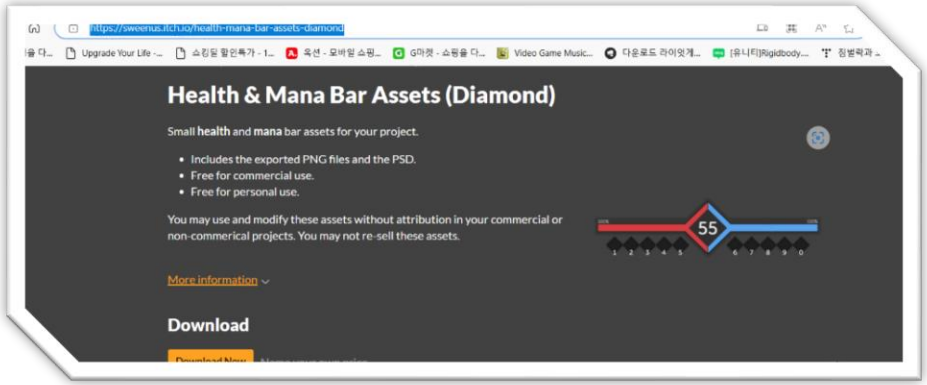
플레이어 감지 시스템
- 룸에서 나갈 때나 게임 중 상대 플레이어가 갑자기 나갈 시 이를 감지해 적절한 설정을 해준 뒤 씬을 이동시킨다.

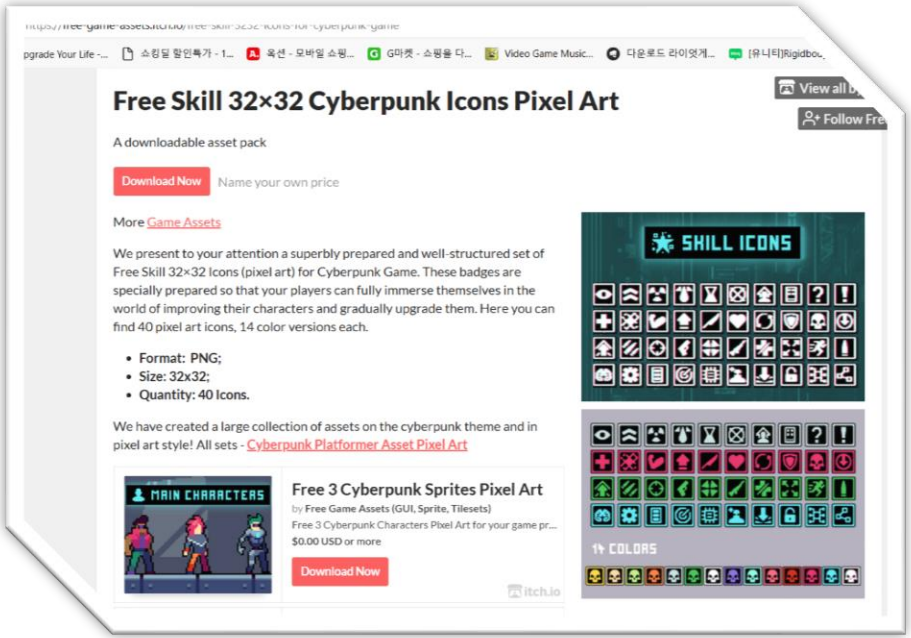
- **게임 도중 플레이어가 나가면 게임을 중단 시킴.**

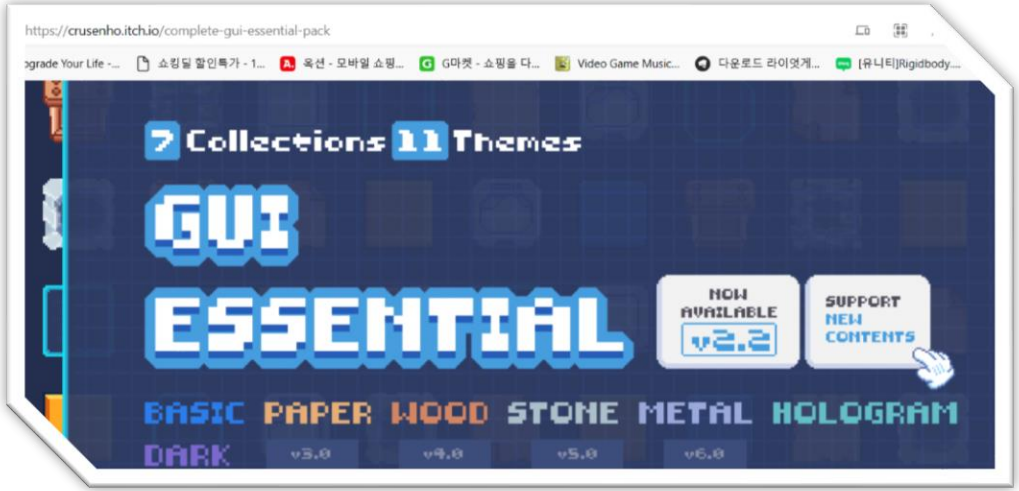
=> 중간에 다른 플레이어 난입 불가

5. 사용 Asset 및 작업 툴

Itch.io
- 인디 게임 커뮤니티에서 게임 관련 무료 에셋을 다운받아 사용함.

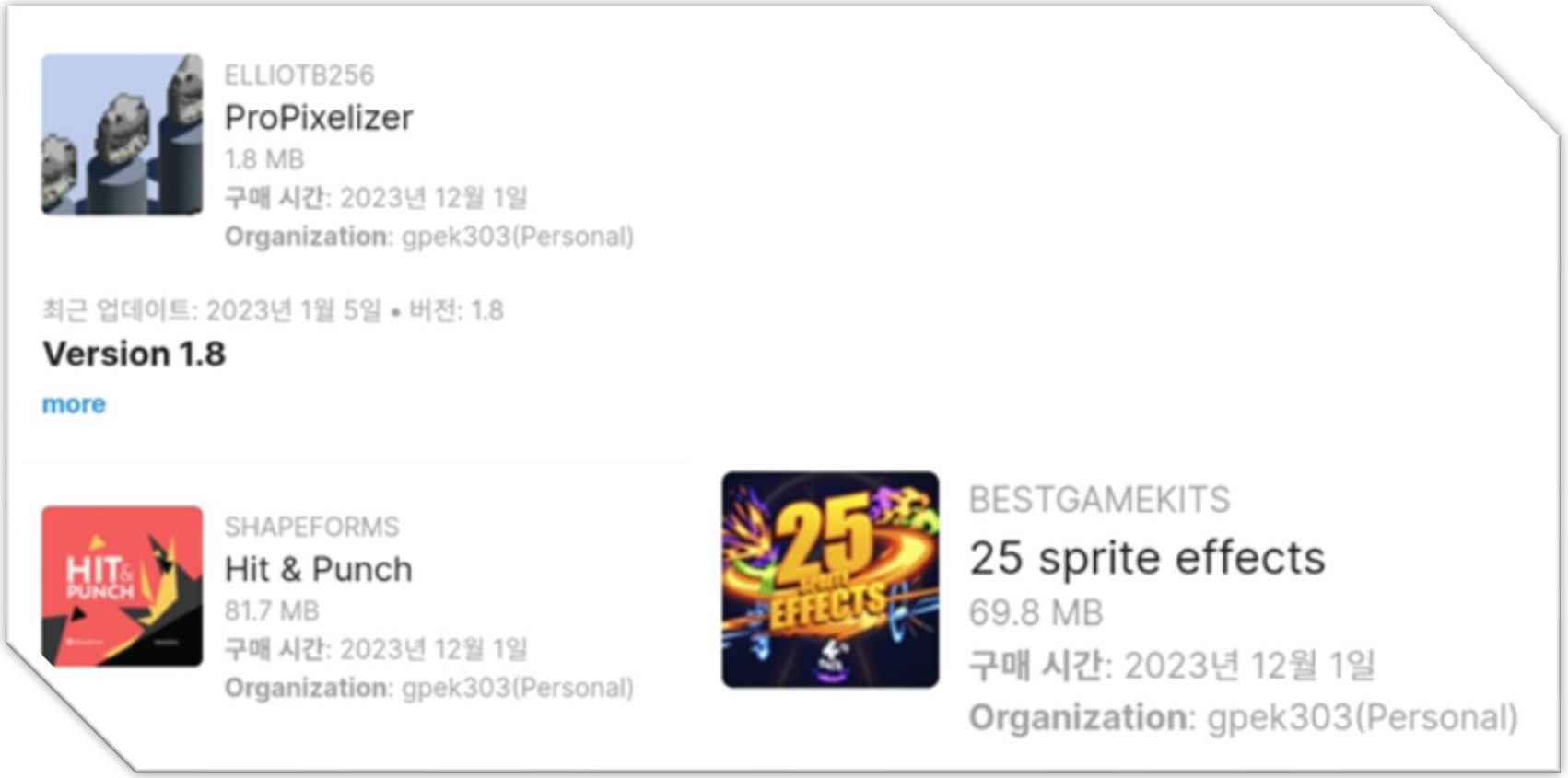






5. 사용 Asset 및 작업 툴

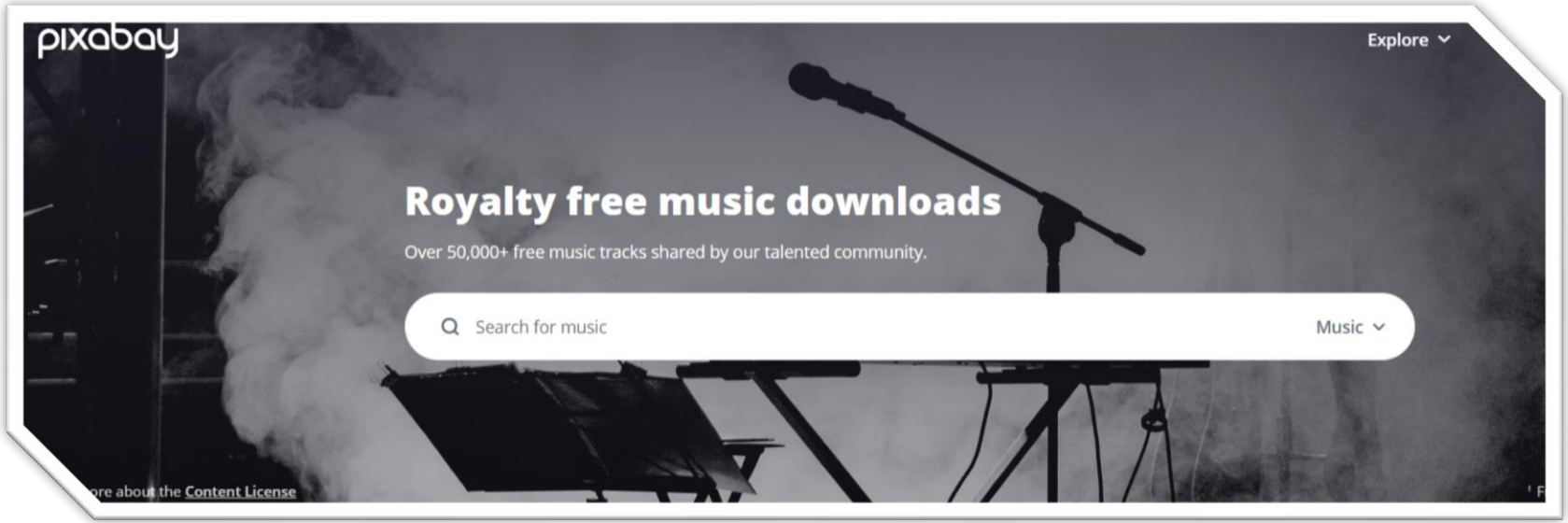
Unity Asset Store
- CH+ 캡스톤 디자인에 참여해 구매한 에셋을 이용하여 특정 이펙트나 사운드 부분에 활용.



5. 사용 Asset 및 작업 툴

Pixabay

- 무료 음악, 사운드 이펙트, 이미지 등을 공유하는 사이트로 해당 사이트에서 여러 리소스를 구해 사용하였다.

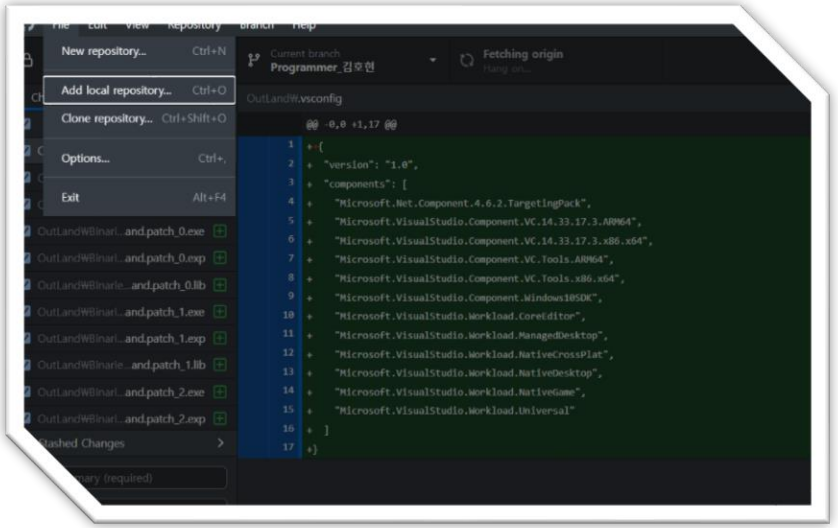


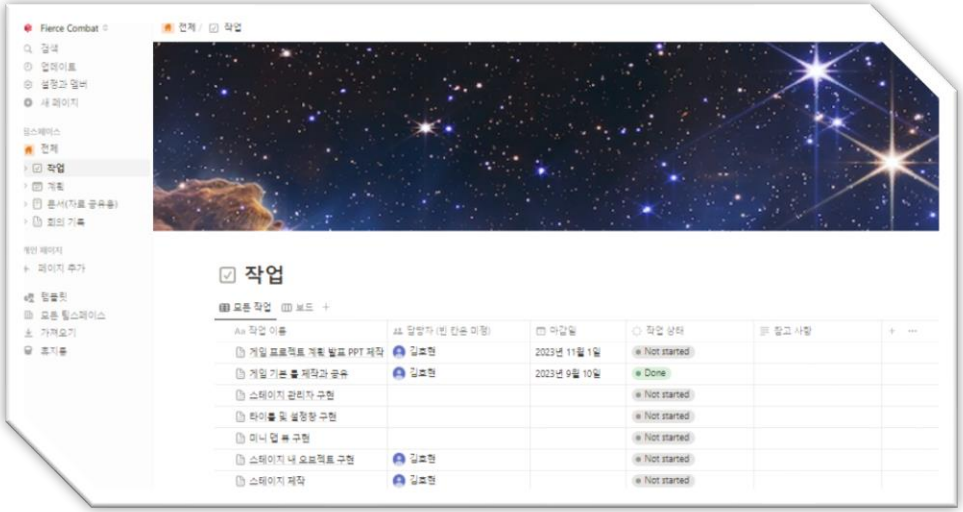
- 앞서 설명한 사이트 외 Asset의 경우 특정 게임에서 추출된 리소스를 활용하거나 자체 제작.

5. 사용 Asset 및 작업 툴

- 작업 툴
  - Unity, Photon : 게임 제작
  - 3ds max, Blender, Pixel Studio, GIMP
    : 리소스 제작

  - GitHub : 프로젝트 관리
  - Notion : 회의 내용 보관 및 개발 진행 보고
  - Discord : 회의 진행 및 대화 창구





6. 후기

<김호현>

네트워크 동기화 과정에서 생각보다 많은 버그를 만나야 했지만

결국 프로젝트 완성에 성공했고, 무엇보다 이러한 과정에서 여러 네트워크 게임 제작에 필요한 기술을 획득해 좋았던 것 같다.

<최민규>

네트워크 게임을 먼저 만들어본 경험이 있는 호현이에게 많은 것을 배웠다.

해당 게임을 제작하면서 네트워크 게임의 로직을 읽고 수정하는 것이 생각보다 쉽지 않다는 것을 느꼈다.

혼자만의 싱글 게임을 만들다가 같이 할 수 있는 네트워크 게임을 할 수 있다는 것이 굉장히 즐겁다는 것을 배웠다.

<한세욱>

Photon을 사용해서 네트워크 게임을 처음 만들어 보았는데, 사용이 낯설었지만 간단하게 멀티플레이 게임이 만들어져서 신기했다. 또한 여러가지 오브젝트들을 동기화 하면서 Photon 사용에 익숙해 진 것 같다.

네트워크 게임을 만들면서 둘 이상 같이 테스트를 해보면서 동기가 부여되는 느낌 또한 받았다.

7. 영상

기말 과제.MP4 참조