2022 NPIXEL 하계 인턴 프로젝트

신건호

Project_PW

Project PW

2022 NPIXEL 하계 인턴 프로젝트

개발 기간 : 2022.07.04 ~ 2022.09.08
역할 : 프로젝트 매니저, 네트워크 개발
개발 환경 : Unity Engine 2022.2.0a18, PUN2, C#, GitLab
개발 인원 : 4 명
개요 : 웜즈, 포트리스의 턴제 포격 시스템을 기반으로 한 멀티 게임 제작

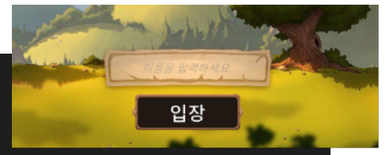
네트워크

PUN2를 사용한 네트워크

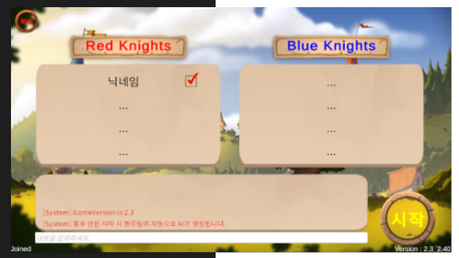
유니티에서 멀티 플레이 환경을 만들기 위해 PUN2를 사용하였습니다. 로비에서는 서버 연결 로직과 유저의 상태 관리를 분리하기 위해 NetworkManager는 Photon에 접속하고 Room을 만드는 네트워크 기능에 집중 시키고, LobbyManager가 플레이어의 입력에 따른 상태를 관리하도록 하였습니다. 이후 게임 진행 중 각 클라이언트에서 입력된 플레이어의 행동은 RPC함수 호출을 통해 연결하였습니다.

```
// [NetworkManager] : 서버 연결
Unity 메시지 | 참조 0개
private void Start()
{
    PhotonNetwork.ConnectUsingSettings();          //Photon Cloud에 연결되는 시작 지점
    PhotonNetwork.GameVersion = Application.version;
}
```

```
//[LobbyManager] : 서버에 연결된 이후 접속 활성화
참조 9개
public override void OnConnectedToMaster()
{
    _enterButton.interactable = true;      //서버에 연결되면 입장 버튼 활성화
}
```



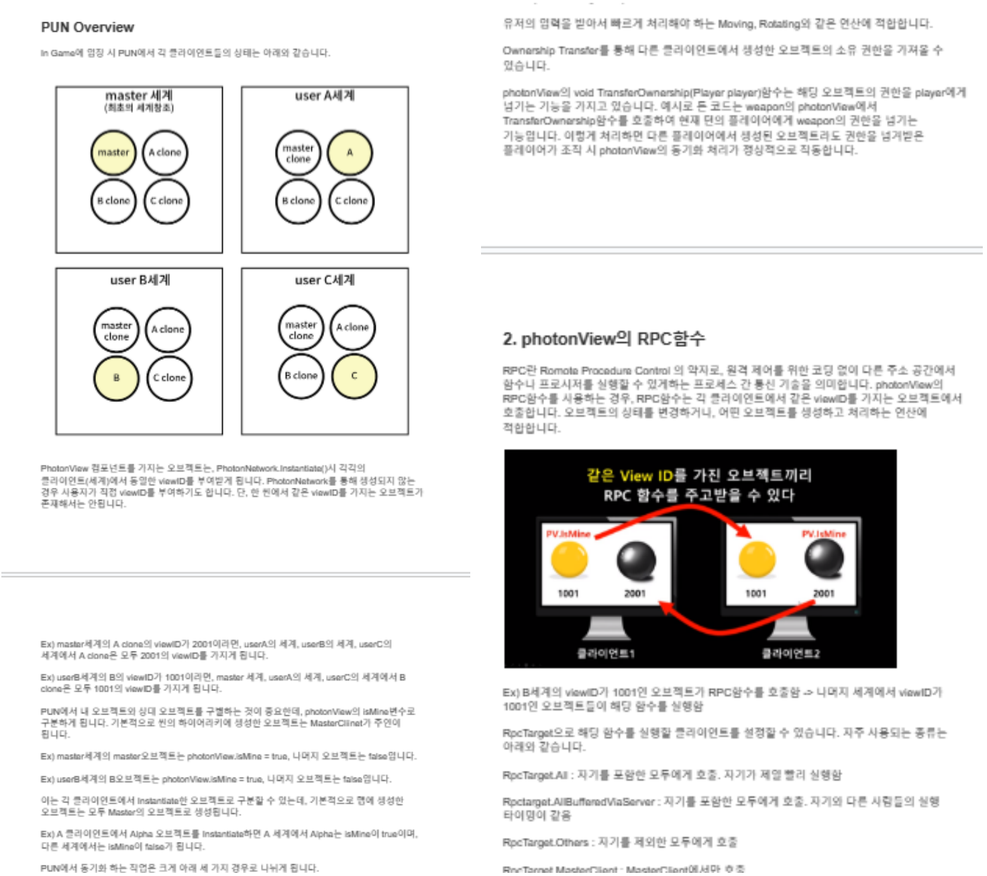

```
//[LobbyManager] : 플레이어 행동에 따른 Lobby 업데이트
참조 0개
public void OnUpdatePlayerNameInput(TMP_InputField nameInput)
{
    PhotonNetwork.NickName = nameInput.text;          //플레이어 닉네임 설정
}
참조 0개
public void OnEnterButton()
{
    if (PhotonNetwork.LocalPlayer.NickName == "")
    {
        Debug.Log("게임에 사용될 닉네임을 입력해주세요.");
        return;
    }

    NetworkManager.s_instance.CreateOrJoinRoom();
}
참조 0개
public void OnEndEditEvent()
{
    if (PhotonNetwork.LocalPlayer.NickName == "")
    {
        Debug.Log("게임에 사용될 닉네임을 입력해주세요.");
        return;
    }

    if (Input.GetKeyDown(KeyCode.Return))
    {
        NetworkManager.s_instance.CreateOrJoinRoom();
    }
}
```



네트워크

PUN2를 사용한 네트워크

게임의 네트워크 구성이 안정화 된 이후로
PUN2를 잘 모르는 팀원들을 위해 PUN을 통해 사용하는
네트워크의 구조와 프로젝트에 사용되고 있는 PUN2의
기능들을 안내하는 Wiki를 작성하였습니다. 이를 통해
팀원 전체의 네트워크 구조에 대한 이해를 높임과 동시에
불필요한 검색 시간과 질문을 최소화하여 업무 효율을
높일 수 있었습니다.

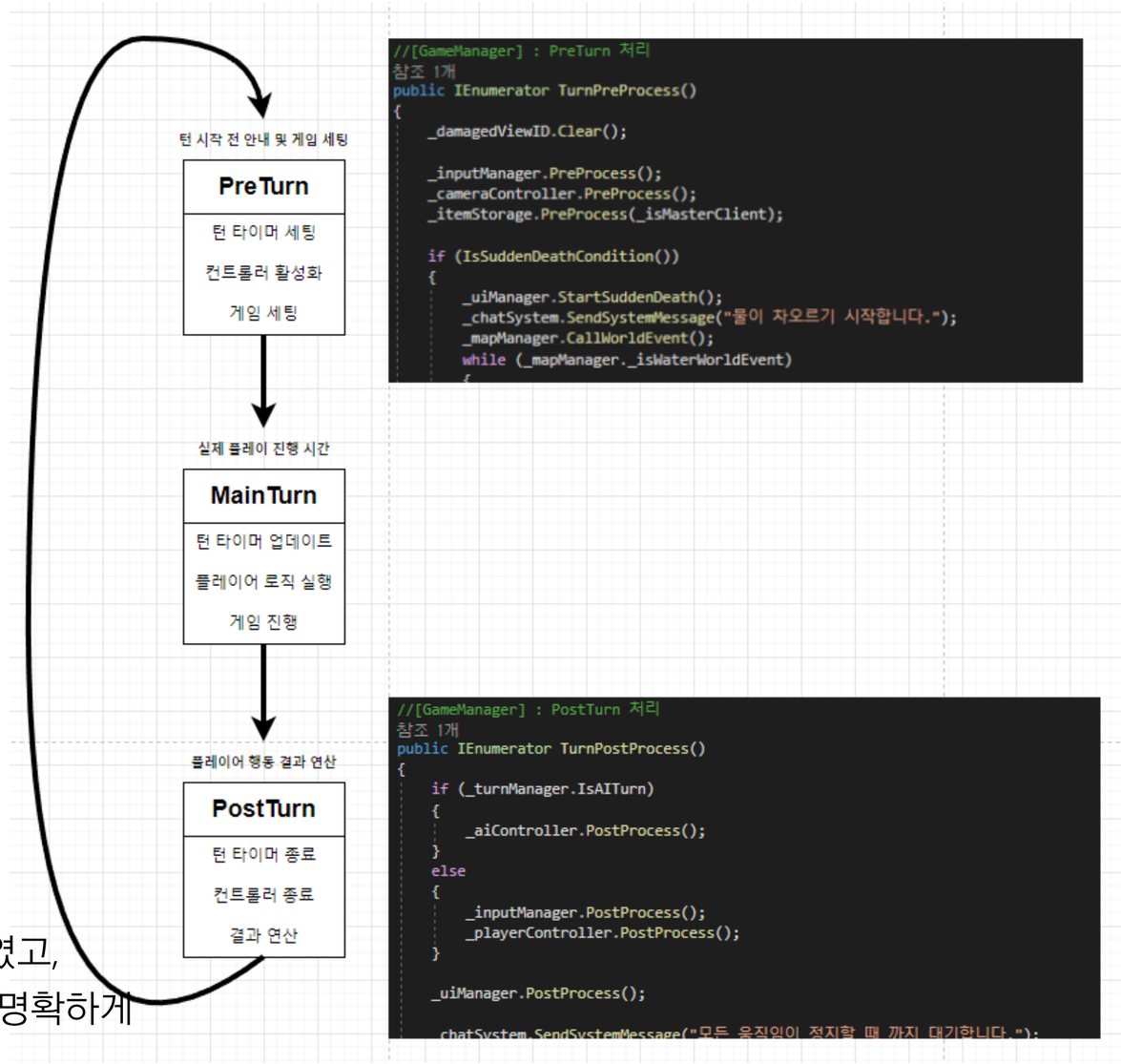


https://docs.google.com/document/d/1y2xm8Rc2oFW9mGBspb2lpEaVDyYbidV5bHzhKFKoJlA/edit

로직 설계

게임 로직과 턴 시스템 구성

네트워크 구성 업무를 하며 시스템의 로직이 어떻게 흘러갈지 고민을 많이 하게 되었고, 이는 자연스럽게 게임 전체 로직 설계에 영향을 주게 되었습니다.
가장 먼저 턴 시스템 구조 설계 작업을 진행하였으며, 웜즈와 포트리스의 턴 시스템을 연구하여 한 턴이 크게 세 파트로 나뉨을 알 수 있었습니다.
따라서 PreTurn-MainTurn-PostTurn으로 각 파트를 구분하여 각 단계의 역할을 정리 후 팀원들에게 공유하였고, 이에 따라 GameManager에서 함수 호출 구조를 보다 명확하게 정리하였습니다.



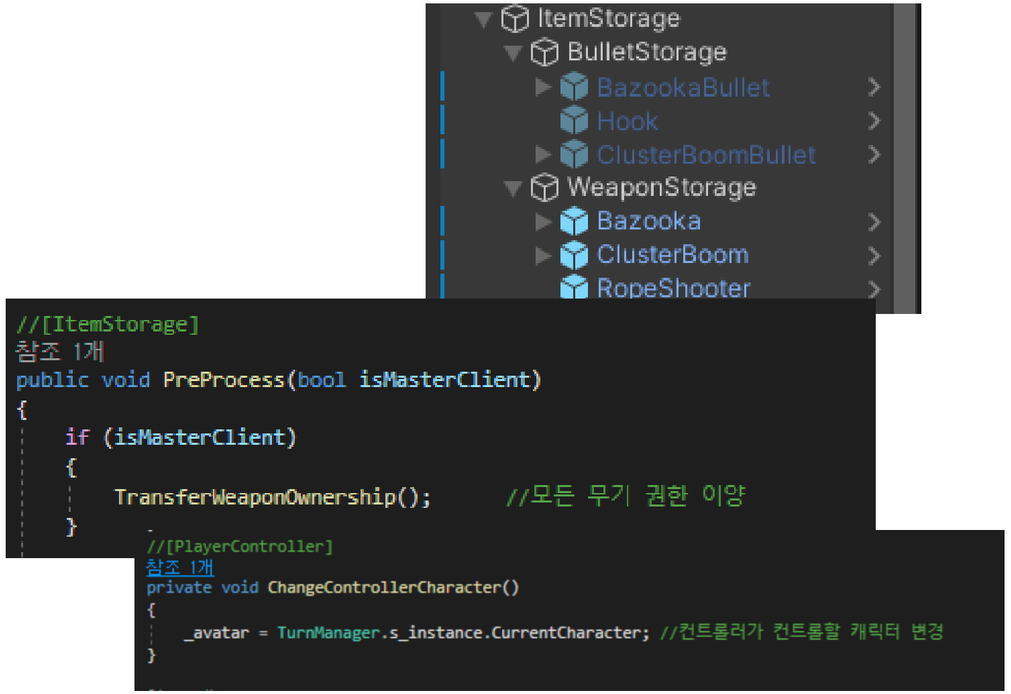

```
//[GameManager] : PreTurn 처리
참조 1개
public IEnumerator TurnPreProcess()
{
    _damagedViewID.Clear();

    _inputManager.PreProcess();
    _cameraController.PreProcess();
    _itemStorage.PreProcess(_isMasterClient);

    if (IsSuddenDeathCondition())
    {
        _uiManager.StartSuddenDeath();
        _chatSystem.SendSystemMessage("물이 차오르기 시작합니다.");
        _mapManager.CallWorldEvent();
        while (_mapManager._isWaterWorldEvent)
        {
```

```
//[GameManager] : PostTurn 처리
참조 1개
public IEnumerator TurnPostProcess()
{
    if (_turnManager.IsAITurn)
    {
        _aiController.PostProcess();
    }
    else
    {
        _inputManager.PostProcess();
        _playerController.PostProcess();
    }

    _uiManager.PostProcess();

    _chatSystem.SendSystemMessage("모든 움직임이 정지할 때 까지 대기합니다.");
```

로직 설계

게임 로직과 턴 시스템 구성

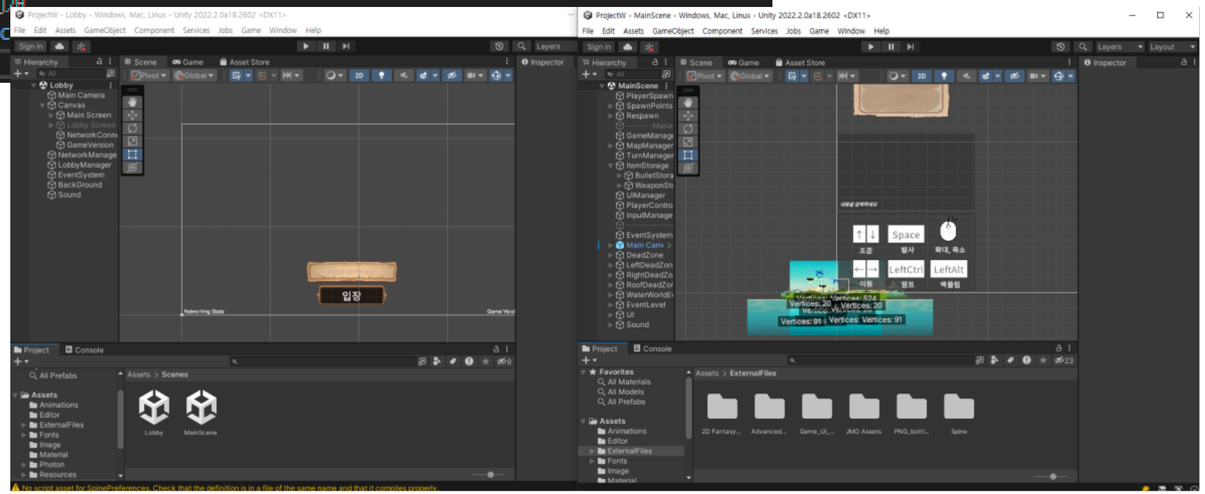
시스템을 명확히 설계한 다음, 게임 로직에서 최적화 할 수 있는 부분을 검토하였습니다. 이에 기존에 각각의 플레이어가 독립적으로 컨트롤러와 무기를 보유하고 있던 부분을, 한 턴에 오직 한 명의 플레이어만 플레이가 가능하다는 것에 기반하여 한 명 분의 컨트롤러와 무기만 관리하여 PreTurn 단계에 모든 권한을 이양하는 방식으로 변경하여 Scene에 불필요한 리소스가 남는 것을 방지하였습니다.



환경 개선

효율적인 작업을 위한 환경 개선

```
[Header("NextTurn Button")]
public Button _nextTurnButton = null;

참조 0개
public void OnNextTurnButton()
{
    photonView.RPC("MoveNextTurn", RpcTarget.All);
}

[PunRPC]
참조 0개
public
{
```



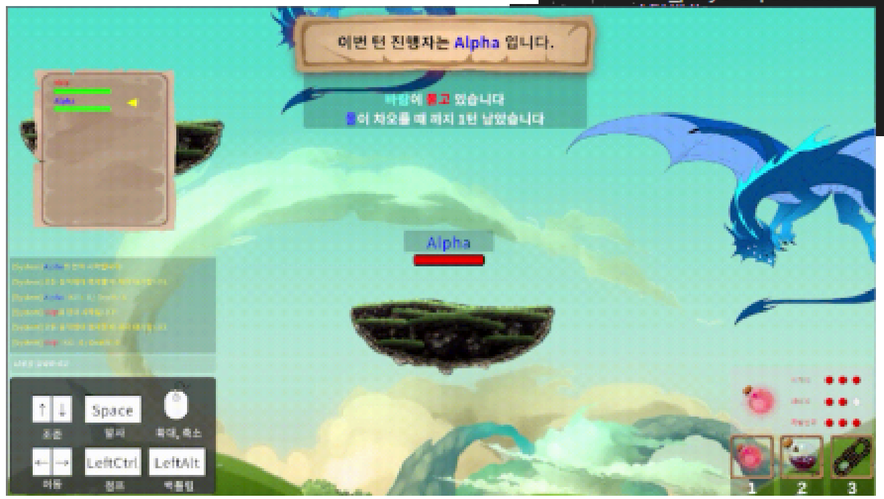
멀티 플레이가 기본인 게임을 개발하는 만큼,
멀티 플레이 테스트 시간을 줄이도록 하였습니다.
가장 먼저 Symbolic link를 사용하여 여러개의 유니티
클라이언트를 실행할 수 있도록 하여 팀원 개개인이
멀티 환경을 테스트할 수 있게 하였으며,
각 플레이어의 상태 확인을 위해 게임이 진행되며
기다리는 시간을 없애기 위해 턴 타이머 시간 설정,
턴 스킵 등의 기능을 개발자 기능으로 추가하였습니다.

https://support.unity.com/hc/en-us/articles/115003118426-Running-multiple-instances-of-Unity-referencing-the-same-project

환경 개선

효율적인 작업을 위한 환경 개선

```
//[ChatSystem]
참조 0개
public void OnEndEditEvent()
{
    if (Input.GetKeyDown(KeyCode.Return))
    {
        //TODO : 빌드 전 삭제 필요
        if (_playerInput.text.Contains("/over"))
        {
            TurnManager.s_instance.DecreaseRemainTime();
            _playerInput.text = "";
```

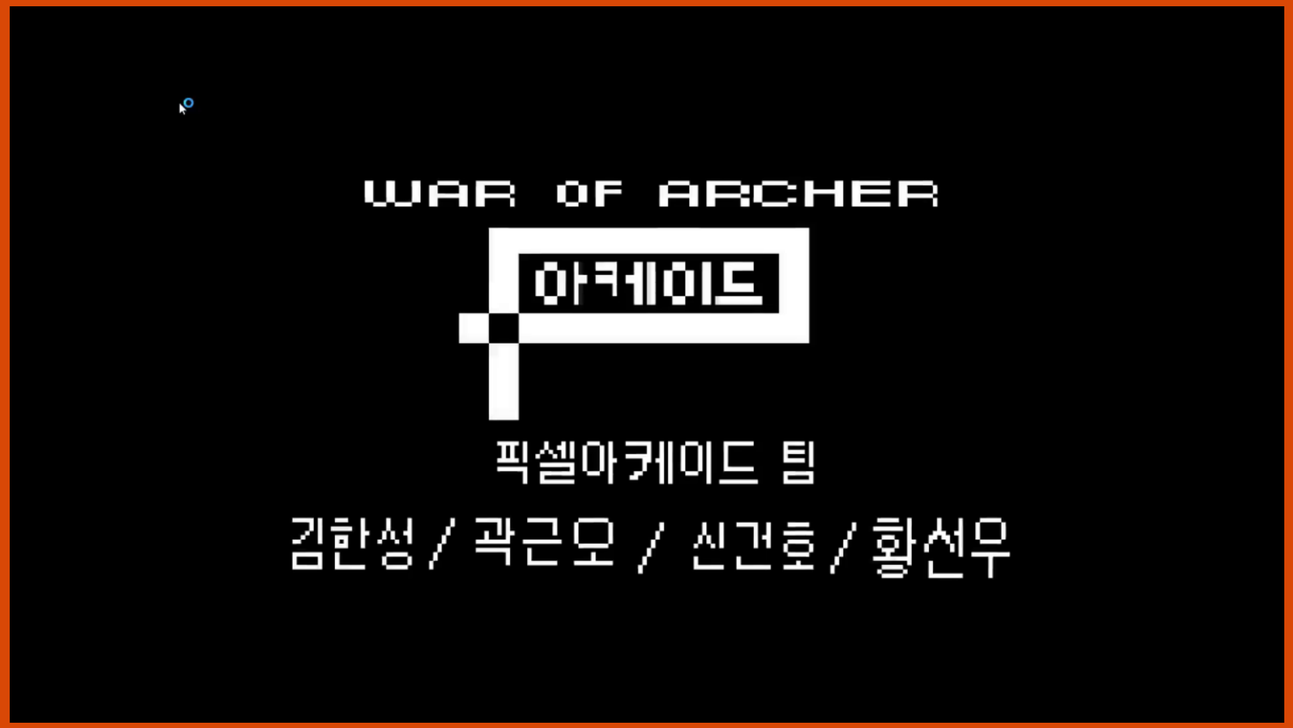


테스트를 진행하며 상황에 따른 로그를 자세히 확인하기 위해 채팅 시스템을 변형하여 시스템 메시지를 채팅창에서 확인할 수 있도록 하였습니다. SendSystemMessage함수에 메시지만 적으면 내부적으로 처리하여 채팅창에 로그가 나오도록 캡슐화 하였습니다. 또한 기존에 Button UI로 처리하던 개발자 기능을 /function으로 채팅창에서도 작동되게 하여 팀원들이 보다 편하게 개발자 기능을 사용할 수 있도록 만들었습니다.

```
//[ChatSystem] : 시스템 메시지 함수만 공개하여 로그 메시지만 적으면 채팅창에 나오도록 구성
참조 6개
public void SendSystemMessage(string msg)
{
    if (PhotonNetwork.IsMasterClient)
    {
        string message = "<color=yellow>" + "[System] " + msg + "</color>";
        photonView.RPC("UpdateChatText", RpcTarget.All, message);
    }
}
//[ChatSystem] : 내부 처리는 private으로 비공개
[PunRPC]
참조 0개
private void UpdateChatText(string msg)
{
    for (int i = _chatTextList.Count - 1; i > 0; i--)
    {
        _chatTextList[i].text = _chatTextList[i - 1].text;
    }
    _chatTextList[0].text = msg;
}
```

wjrchorh135@gmail.com

https://github.com/Icefin



ProjectPW Portfolio