

포트폴리오

권혁진

보유기술

- C#/C++/유니티/몽고디비/aws람다/EFcore/mysql

개발경험

- DP 프로젝트 - 주사위 기반 PvP 게임 (21.09 ~ 22.12, 미출시)
- LE 프로젝트 - 수집형 RPG 게임 (22.12 ~ 24.03, 미출시)

DP 프로젝트

-신규 프로젝트로 처음부터 개발에 참여했습니다

-인게임 코어 플레이 개발을 담당하였습니다

1. 인게임 UI개발

-주사위 배팅 시스템

-상단 상황판

-전투 결과 및 보상창

2. 코어 플레이 개발

-전투 라운드 캐릭 움직임

-캐릭터 동작 및 스킬 시스템

-라운드 누적에 따른 캐릭 보유 및 진화상태

-포톤을 이용한 멀티플레이

3. 업무 지원 툴 개발

-캐릭터 밸런스 조절을 위한 테스트 씬

-유닛 이펙트 설정 툴

4. 최적화

-리소스 관리를 통한 메모리 최적화

-병목 지점 분석 및 최적화

-로딩 및 스파이크 지점 감소를 통한 유저 경험 개선

LE 프로젝트

- 신규 프로젝트로 처음부터 개발에 참여했습니다
- 서버와 클라이언트 개발을 담당하였습니다

1. 서버 개발

- MongoDB와 AWS를 활용한 서버리스 환경 구축
- MongoDB App Service를 활용해 초기 서버리스 환경을 구축하고 이후 AWS로 마이그레이션 진행
- JWT를 사용한 로그인 및 검증 시스템 개발로 보안성 강화
- Node.js를 활용한 다양한 서버 기능 개발
- 기획 지원 및 데이터 관리 웹페이지 개발
- React와 MongoDB App Service를 연동하여 관리자와 기획자가 데이터베이스를 갱신할 수 있는 웹페이지 구현
- 웹이브 형식의 스테이지나 데이터 테이블을 직관적으로 수정 및 추가 가능한 UI 도구 제공

2. 클라이언트 개발

- 실시간 데이터 갱신 및 최적화
- Addressable Asset, Scriptable Object, S3 Bucket을 활용하여 실시간 상품 갱신 및 데이터 동기화 기능 구현
- 중복 에셋 제거 및 클라이언트 최적화로 메모리 사용량 감소 및 성능 개선
- 기타 주요 작업
- Firebase를 활용한 푸시 메시지 기능 구현
- AdMob과 애드X를 통해 광고 시스템 구축
- Google Console과 Unity를 활용한 결제 시스템 개발
- Google Play SDK와 MongoDB App Service를 연동한 로그인 기능 구현

3d게임

- 유튜브 링크 : <https://youtu.be/QYXGzUekVSs>

캐릭터 조작

- 뉴 플레이어 인풋 시스템을 이용하여 구현
- 이동 : wasd
- 공격 : 좌클릭
- 아이템 사용 1,2,3
- 무기 변경 : 4,5,6
- 락온 : tab
- 인벤토리 : I
- 캐릭터 : c

Fsm구현 (CreatureStateMachine)

- 제네릭으로 구현
- 몬스터 및 플레이어 공용
사용 가능

```
private State<T> _currentState; // 현재 상태
private StateCache<T> _stateCache = new StateCache<T>(); // 상태 캐싱
private Dictionary<Type, Type>, Func<bool>> _transitions = new Dictionary<Type, Type>, Func<bool>>

test <test>
public State<T> CurrentState => _currentState;

//상태 변경
Frequently called 3 usages test <test>
public void ChangeState(Type stateType){...}

//상태 추가
9 usages test <test>
public void AddState(State<T> state){...}

//상태 전이 조건
10 usages test <test>
public void AddTransition<U, V>(Func<bool> condition) where U : State<T> where V : State<T>{...}

//글로벌 상태 전이 조건
2 usages test <test>
public void AddGlobalTransition<V>(Func<bool> condition) where V : State<T>{...}

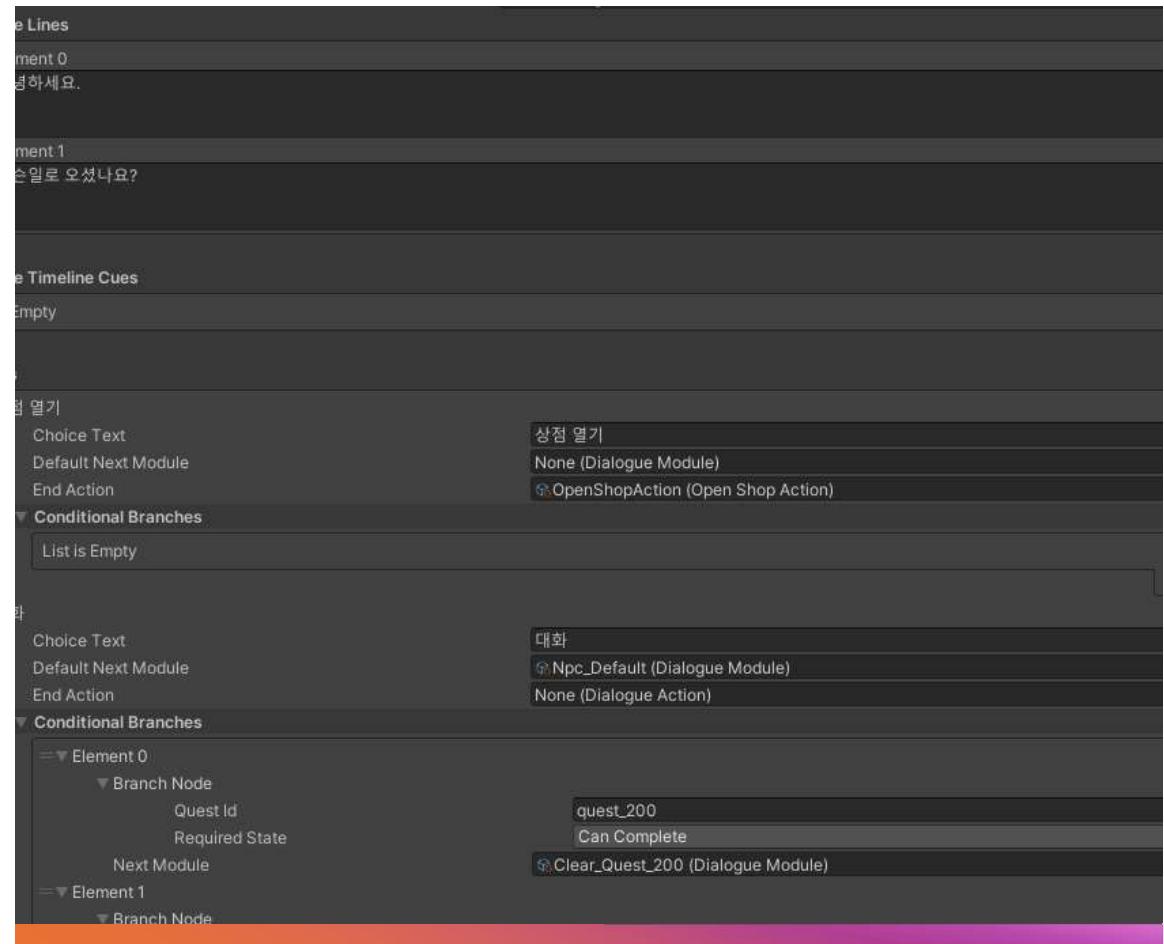
//상태에 따른 업데이트 실행
Frequently called 2 usages test <test>
public void Update(){...}
```


플레이어 이동

- 캐릭터 컨트롤러를 이용한 기본 이동
- 자동이동시 네브매쉬 에이전트로 변경 후 자동이동 실행
- 사다리는 오프 매쉬 링크를 이용하여 이동

Npc구현(NpcDialogue)

- 조건에 따른 npc 대화 내용 구현
- dialogueLines를 통한 대화 내용 구현
- dialogueTimelineCues를 이용한 대화 도중 이벤트 실행
- Choices를 통한 선택지 구현
- conditionalBranches를 이용한 대화 분기
- DialogueAction을 이용한 대화 전후 이벤트 실행



Npc구현(NpcDialogue)

- 퀘스트 추가, 클리어, 상점
오픈, 이벤트 실행을
DialogueAction을 할당하
여 구현

```
public class AddQuestAction : DialogueAction
{
    [SerializeField] private string questId = "quest_";

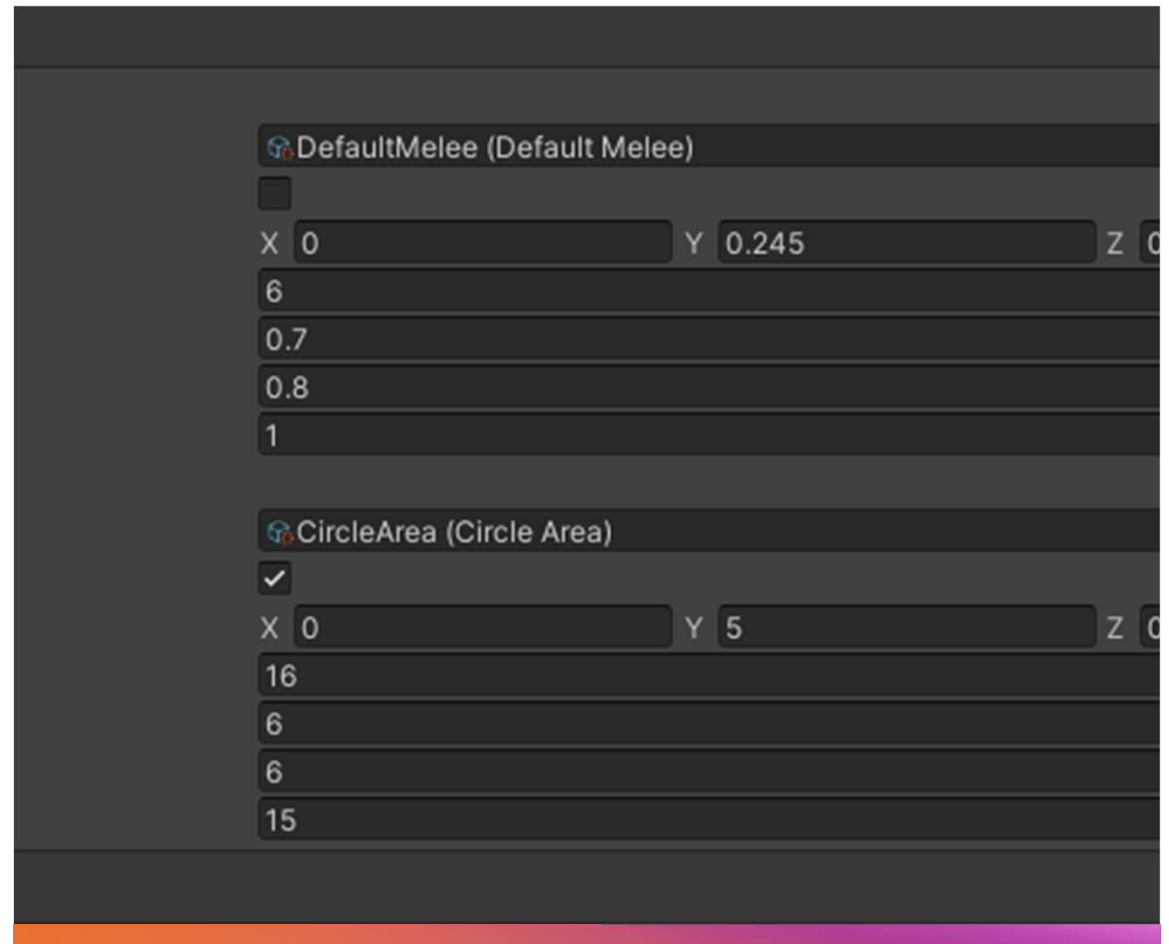
    public override void Execute()
    {
        Managers.QuestManager.AddQuest(questId);
    }
}
```

```
public class OpenShopAction : DialogueAction
{
    [SerializeField] private string shopId = "shop_";

    public override void Execute()
    {
        PopupUIManager.Instance.OpenPopupById(PopUpName.ShopUI, data: shopId);
    }
}
```

몬스터 구현

- 스크립터블 오브젝트를 이용한 몬스터 구현
- SkillAction을 상속받은 스크립터블 오브젝트를 할당하여 스킬 액션 구현
- 미리 지정해둔 스탯 정보를 SkillAction에 매개변수로 보내서 실행



퀘스트 구현

- 퀘스트가 추가 될시 이벤트 해당하는 조건의 트리거에 퀘스트를 할당하여 진행도 갱신

```
test <test>
public static event Action<string, int> OnMonsterKilled;

test <test>
public static event Action<string, int> OnItemCollected;

test <test>
public static event Action<Vector3> OnPlayerMoved;

Frequently called 1 usage test <test>
public static void TriggerMonsterKilled(string monsterId, int count)
{
    OnMonsterKilled?.Invoke(monsterId, count);
}

Frequently called 2 usages test <test>
public static void TriggerItemCollected(string itemId, int count)
{
    OnItemCollected?.Invoke(itemId, count);
}

Frequently called 2 usages test <test>
public static void TriggerPlayerMoved(Vector3 position)
{
    OnPlayerMoved?.Invoke(position);
}
```

퀘스트

- Quest 데이터를 할당받아
퀘스트 정보를 갱신

```
Frequently called 9 usages
public QuestData Data { get; private set; }

test <test>
public event Action OnUpdateProgress;

Frequently called 3 usages test <test>
protected Quest(QuestData data)
{
    Data = data;
}

Frequently called 1 usage 3 overrides test <test>
public abstract void Subscribe();

Frequently called 1 usage 3 overrides test <test>
public abstract void Unsubscribe();

Frequently called 2 usages 3 overrides test <test>
public abstract bool CanComplete();

Frequently called 2 usages 3 overrides test <test>
public abstract string GetProgress();

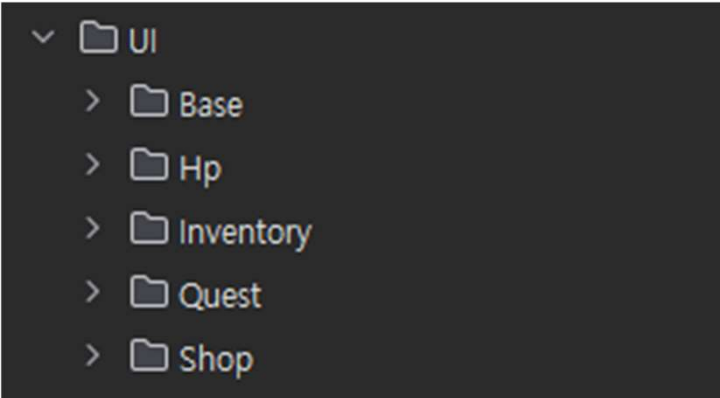
Frequently called 3 usages test <test>
protected virtual void InvokeOnUpdateProgress()
{
    OnUpdateProgress?.Invoke();
}
```

이벤트 실행

- 타임 라인 디렉터, 타임라인 에셋, 시그널 리시버를 이용하여 구현

매니저 구현 방식

- 데이터를 다루는 부분이랑 UI를 수정하는 부분을 나눠서 구현



```
▼ UI
  > Base
  > Hp
  > Inventory
  > Quest
  > Shop
```

```
private ObjectManager _obj;
private DropManager _drop;
private PoolManager _pool;
private InventoryManager _inventory;
private DataManager _data;
private ShopManager _shop;
private GameStateManager _gameState;
private QuestManager _quest;
private SoundManager _sound;
```


2d 멀티 게임

- 유튜브 링크 : <https://youtu.be/TM2gDkTecgg>
- C#/Unity/Efcore를 이용하여 개발