2024년 PBL II 최종보고서

조합 메카닉 멀티엔딩 2D 플랫포머 게임

새바람 **(SaeBaram)**

경상국립대학교 컴퓨터공학과

팀명	새바람
소속	컴퓨터공학과
팀원	김현찬, 고희수, 배정훈, 이경현

1. 요약문

‘새바람’은 포스트 아포칼립스를 배경으로 하여 부모님을 찾기 위해 황폐한 세계를 탐험하고 이 과정에서 만난 사람들과의 이야기를 통해 주인공이 성장하는 조합 메카닉 멀티엔딩 2D 플랫포머 게임을 만드는 프로젝트이다.

2. 서론

2.1. 추진배경 및 목적

목적

조합 메카닉 멀티엔딩 플랫포머 2D 게임 데모 개발

게임

- 지금까지 학과에서 배운 전공지식을 활용하기에 적합하며,
- 팀 구성원의 희망 진로와 맞닿아 있기 때문에 선택했다.

조합

- 플레이어가 게임 속 세상의 탐색을 만끽하도록 한다.

그림 01. 인벤토리 팝업 UI

그림 02. Chapter. 0 지형에 사용되는 배경

- 다양한 공간에서 얻은 재료는 플레이어의 상상력을 자극할 수 있고, 또 다른 곳으로 발걸음을 옮기게 되는 원동력이 되어준다.
- 조합으로 문제상황의 해결책이 되어주는 메카닉을 만들 수 있으며, 조합은 플레이어가 게임 세상의 다양한 부분에 집중하도록 돕는다.

그림 03. 아이템 조합 시스템 UI

- 조합은 플레이어가 세상에서 획득한 재료를 활용하는 방법이 되어주며, 이렇게 만들어낸 메카닉은 그 자체로 플레이어에게 성취감이 되어줄 수 있으며, 이를 통해 문제상황을 해결해 낼 때도 성취감을 제공해 줄 수 있다.

메카닉

- 프로젝트 “새바람”에서 메카닉은 전반적인 게임 플레이를 이끌어나가는 중요한 키워드가 되어준다.

- 메카닉은 문제상황의 해결책이 되며, 세계관에서 사람과 사람 간의 연결고리를 형성하는 매개체가 된다.

그림 04. 탐험 중 장애물을 만난 주인공

그림 05. 장애물 극복을 위해 메카닉을 획득

그림 06. 장애물을 극복한 주인공

- 메카닉이 게임에서 궁극적으로 전달하고자 하는 메시지인 가족애와 인류애를 이끌어낼 수 있는 소재로 동작하기를 기대한다.
    - 새바람은 조손 가정의 주인공 ‘가온’의 이야기를 담았다.
    - 전쟁에서 사용된 메카닉 장비로 황폐화된 세상이지만, ‘가온’은 메카닉 장비를 통해 자신의 세상을 더 넓혀간다.

그림 07. 주인공 ‘가온’과 할아버지

멀티엔딩

- 플레이어의 선택 중요성을 더하기 위한 멀티엔딩은 플레이어가 적극적으로 게임의 흐름과 결말에 영향을 미칠 수 있음을 플레이어에게 알린다.

그림 08. NPC 상호작용 이해를 돕기 위한 이미지

- 이러한 구조는 플레이어에게서 게임과 게임 내 요소들과의 적극적인 상호작용을 기대할 수 있고, 게임에 더욱 몰입할 수 있도록 한다.
- 멀티엔딩은 게임의 재 플레이 가치를 극대화할 수 있다. 플레이어의 선택에 따라 다양한 흐름으로 분기할 수 있는 구조는 플레이어에게 자신이 경험해 보지 못한 흐름에 대한 궁금증을 유발한다.
- 이는 다 회차 플레이로 이어지고, 궁극적으로 플레이어에게 게임 세계관과 자신의 플레이 경험에 대한 더욱 견고한 애착을 형성하는 계기가 된다.

플랫포머 **2D**

- 개발 환경과 현재의 기술적, 시간적 자원을 고려하여 2D 도트 그래픽 플랫포머 장르를 선택했다. 또한 다양한 플레이어에게 친숙한 장르이면서 다른 소재,
- 타 장르와 융합하기 유연한 장르라고 판단하여, 기본 장르로 채택했다.

3. 프로젝트 목표 및 주요내용

3.1. 프로젝트 목표

목표

포스트 아포칼립스 시대의 사람들의 이야기를 바탕으로한 메카닉 멀티엔딩 2D 플랫포머 게임 데모 개발

3.2. 프로젝트 개발 내용

프로젝트 개발 내용	
주요기능	기능 대분류
시스템	인벤토리 시스템
	다이얼로그 시스템
	캐릭터 컨트롤 시스템
	컷신 시스템
	전투 시스템
	보스 전투 시스템
	상점 시스템
	플랫폼 시스템
	사운드 시스템
레벨 디자인	Chapter 0

4. 설계 및 구현

4.1. 요구사항 분석

기능 요구사항 명세서

주요기능	기능 대분류	기능 소분류	세부항목	코멘트
시스템	인벤토리 시스템	아이템 생성 (불러오기)	아이템 이름	
			아이템 id	
			아이템 품질	
			아이템 스택 여부	
			아이템 이미지 바인드	
			아이템 갯수	
		아이템 이동	슬롯 간 아이템 이동	
			동일 아이템 스택	

		아이템 저장	아이템 id	
			아이템 갯수	
			아이템 위치	
			아이템 품질	
			아이템 로그	
			아이템 강화도	
		아이템 로그	아이템 로그	
		팝업 UI 시스템	제작	
			강화	
			퀘스트	
			다이어리	
			지도	
			마지막 페이지	
		아이템 조합	조합 슬롯 간 아이템 이동	
			조합 가능 유무 파악	
			조합	
		아이템 강화	강화 슬롯 간 아이템 이동	
			강화 가능 유무 파악	
			강화	
	다이얼로그 시스템	대화	캐릭터 스크립트 출력	상호작용 키 입력 시
			선택지 출력	선택에 따라 다른 스크립트 출력
	캐릭터 시스템	캐릭터 이동	캐릭터 이동	
			이동 애니메이션	
		상호작용	말걸기	
			분해	
			채집	
			공격	
			아이템 습득	상시 적용
		평판	평판 상승	

			평판 하락	
		능력치	용기 상승	
			지혜 상승	
			손재주 상승	
	컷신 시스템	컷신 시스템	카메라 효과 - 레터박스	
			카메라 효과 - 흔들기	
			카메라 효과 - 줌 인	
			카메라 효과 - 줌 아웃	
			애니메이션	
			스크립트 출력	
			일러스트 출력	
	전투 시스템	일반 몬스터	순찰	
			감지	
			추격	
			공격	
	보스 전투 시스템	보스 룸	룸 진입	체력바 시각화 보스 활성화
		보스 몬스터	근접 공격 패턴	
			돌진 패턴	플랫폼 생성
			포효 패턴	장애물 추락
	상점 시스템	구매	플레이어 돈 차감	
			플레이어 인벤토리에 아이템 추가	
			상인 인벤토리의 아이템 차감	
		판매	플레이어 돈 추가	
			플레이어 인벤토리의 아이템 차감	
			상인 인벤토리의 아이템 추가	
		상점 활성화/ 비활성화 단풍 상점 활성화 / 비활성화		플레이어 평판에 따라 판정
	플랫폼 시스템	일반 플랫폼	상시 충돌	

		특수 플랫폼	조건부 통과	
	사운드 시스템	사운드 설정	전체소리 트랙바 변경	
			전체소리 음소거	
			배경음악 트랙바 변경	
			배경음악 음소거	
			효과음 트랙바 변경	
			효과음 음소거	
레벨 디자인	Chapter 0	튜토리얼	몬스터 별 능력치 및 npc 배치	

표 1. 기능 요구사항 명세서

비기능 요구사항 명세서

성능 요구사항

주요 기능	대분류	소분류	요구사항
시스템	전체	평상시 프레임 속도	평균적으로 75 FPS를 유지해야 한다.
		최악 상황의 프레임 속도	60 FPS 이상을 유지해야 한다.
		맵 로딩	50ms안에 이루어져야한다.
	인벤토리 시스템	아이템 생성	50ms 안에 이루어져야한다.
		아이템 이동	50ms안에 이루어져야한다.
		아이템 저장	50ms안에 이루어져야한다.
		아이템 로그	50ms안에 이루어져야한다.
		팝업 UI 시스템	50ms안에 이루어져야한다.
		아이템 조합	50ms안에 이루어져야한다.
		아이템 강화	50ms안에 이루어져야한다.
	다이얼로그 시스템	대화	사용자가 상호작용키를 누르고50ms안에 대화창이 나와야 한다.
	캐릭터 시스템	캐릭터 이동	50ms안에 이루어져야한다.
		대상 상호작용	50ms안에 이루어져야한다.
		평판	50ms안에 이루어져야한다.
	컷신 시스템	컷신 시스템	50ms안에 이루어져야한다.
	전투 시스템	일반 몬스터	50ms안에 이루어져야한다.

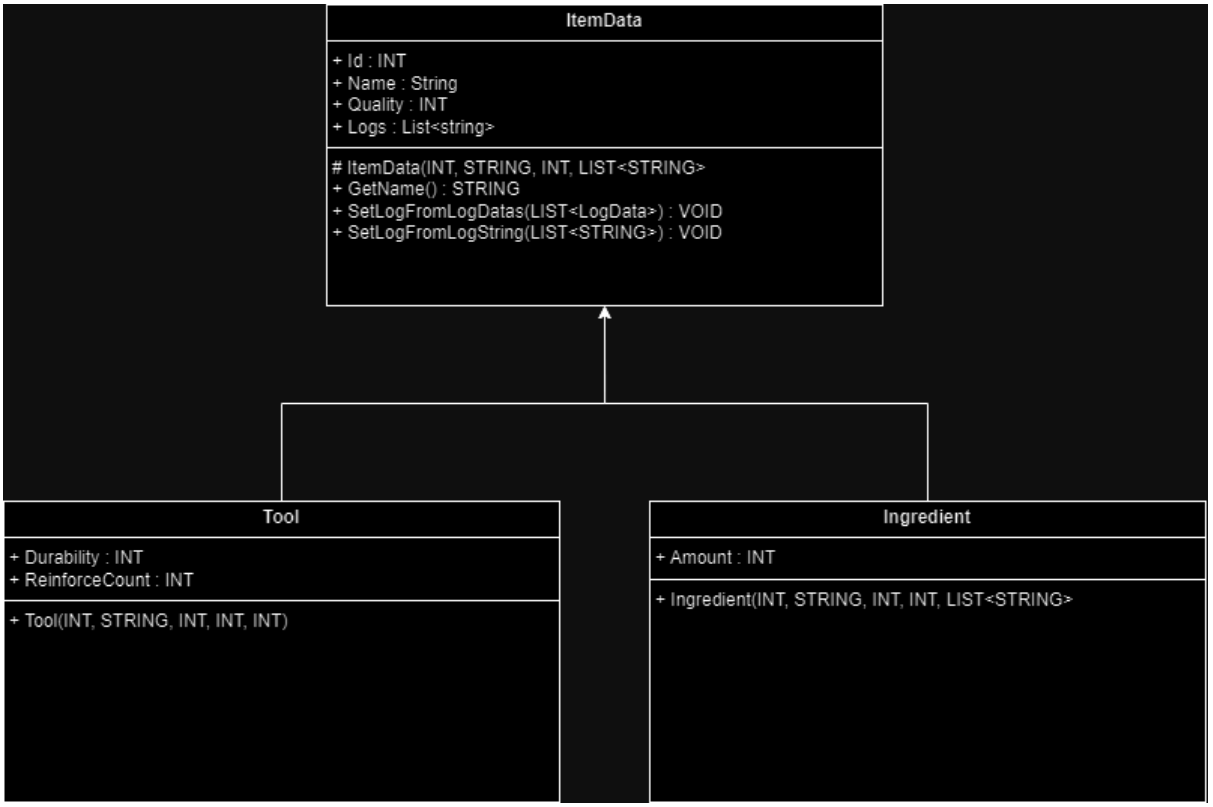
	보스 전투 시스템	보스 룸	50ms안에 이루어져야한다.
		보스몬스터	50ms안에 이루어져야한다.
	조합 시스템	조합	50ms안에 이루어져야한다.
	상점 시스템	구매	50ms안에 이루어져야한다.
		판매	50ms안에 이루어져야한다.
	사운드 시스템	사운드 재생	50ms안에 이루어져야한다.
		사운드 정지	50ms안에 이루어져야한다.
	플랫폼 시스템	일반 플랫폼	50ms안에 이루어져야한다.
		특수 플랫폼	50ms안에 이루어져야한다.

표 2. 비기능 요구사항 명세

4.2. 시스템 설계

[아이템 데이터 저장 방식**]**

관련 데이터 저장 경로 : C:\Users\USER\AppData\LocalLow\Saebaram\Project_SaeBaram



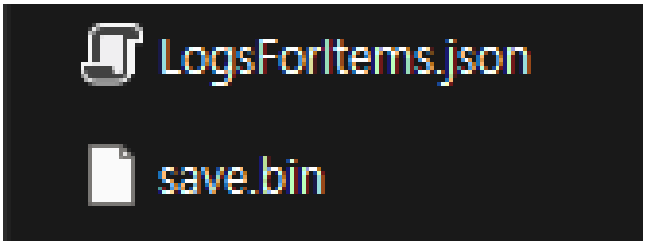


그림 09. 데이터 저장 파일과 아이템 인스턴스 구현방법

그림 9는 도구 아이템인 Tool 클래스와 재료 아이템인 Ingredient 클래스가 ItemData 클래스를 상속받는 것을 보여준다. ItemData 클래스는 Tool, Ingredient 클래스가 공통으로 요구하는 필드와 메소드를 정의한다. Tool 내구도, 강화도 필드가 존재하지만, Ingredient 클래스는 Amount 필드가 필요하다.

```
procedure ItemInfos()
   for i < NumberOfInventorySlots do
      quality <- (InventoryTable[i] >> 14)
      id <- (InventoryTable >> 6) % 255
      durability <- (InventoryTable[i] & 63) >> 2

      if name == NONE
         then itemData = DummyItem()
      else if id >= BOUNDARY:
         then itemData = Tool()
      else
         then itemData = Ingredient()
end procedure
```

표 3. 게임 데이터 로딩 알고리즘

아이템에 대한 정보는 다음과 같은 2바이트로 표시된다. 인게임에 표시된 아이템은 ushort 형 배열로 관리된다. 인벤토리 UI 팝업을 닫거나 게임을 종료하면 배열의 데이터가 그림 11과 같이 저장된다.
게임 시작 시 그림 11과 같이 구성된 save.bin 파일을 읽고 아이템 종류 비트에 따라 도구, 재료 아이템인지 구분한다. 표 3은 게임 데이터 로딩 알고리즘이다.

그림 10. 인게임 아이템 표시

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	3F	E1	60	21	7B	40	BF	40	FC	40	00	00	00	00
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00				

그림 11. 데이터 상 저장된 아이템 정보

퀄리티 (2비트)	아이템 종류 (8비트)	아이템 내구도 (4비트)	강화도 (2비트)

표4. 도구 데이터 프레임

퀄리티 (2비트)	아이템 종류 (8비트)	아이템 갯수 (6비트)

표5. 재료 데이터 프레임

57663(10)				1110 0001 0011 1111(2)			
11 (2)	퀄리티 **3** **(10)**	10000100 (2)	**ID** **132** **(10)**	1111 (2)	내구도 **15** **(10)**	11 (2)	강화도 **3** **(10)**

표 6. 도구(우주 최강 돌 단검) 데이터

16575(10)				0100 0000 1011 1111(2)	
01 (2)	퀄리티 **1** **(10)**	00000010 (2)	**ID** **2** **(10)**	111111 (2)	갯수 **63** **(10)**

표 7. 재료(돌) 데이터

표4, 표5는 도구 데이터와 재료 데이터의 프레임을 나타낸다. 도구 아이템은 중첩이 불가능하지만 아이템 내구도, 강화도가 존재한다. 재료 데이터 프레임은 아이템 개수가 필요하다.

[조합식 관리**]**

아이템 조합 시스템은 게임 플레이의 핵심 요소로서, 효율적이고 신속한 검색, 수정, 삽입 작업이 중요하다. 이를 위해, 다양한 자료구조를 검토한 결과, 해시 테이블이 가장 적절하다고 판단했다.

해시 테이블은 평균적으로 O(1)의 시간 복잡도를 가지므로, 데이터의 검색, 수정, 삽입이 빈번하게 발생하는 상황에서 매우 안정적인 성능을 제공한다. 이는 특히 아이템 조합과 같은 높은 빈도의 데이터 접근이 요구되는 시스템에서 중요한 장점이다.

아이템 조합 시스템의 구현에 있어, 각 조합 레시피를 키-값 쌍으로 저장하고, 키는 입력 아이템들의 고유 식별자 배열을 해싱하여 생성한다. 이를 통해 특정 조합 레시피를 찾거나 수정, 추가할 때마다 빠르고 일관된 성능을 유지할 수 있다.

이 시스템은 Scriptable Object를 사용하여 유니티 에디터 내에서 직관적인 데이터 관리를 가능하게 했으며, 런타임 중에도 조합 레시피를 동적으로 추가하고 수정할 수 있도록 설계했다.

#(슬롯1의 아이템 id, 슬롯2의 아이템 id, 슬롯3의 아이템 id)

그림 12.조합식을 검색하기 위한 키의 구성요소

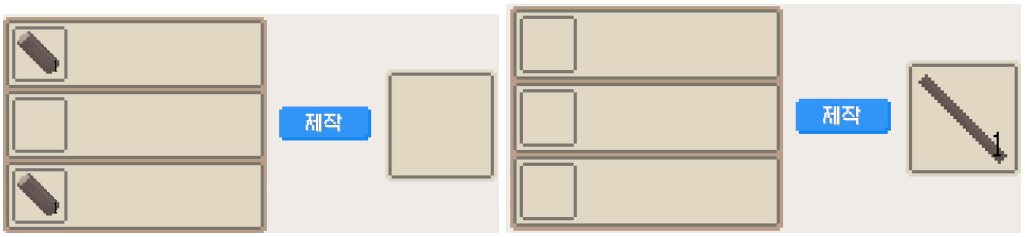


그림 13. 조합 인터페이스를 활용하는 모습. 제작 버튼을 누르기 전 (좌) / 제작 버튼을 누른 후 (우)

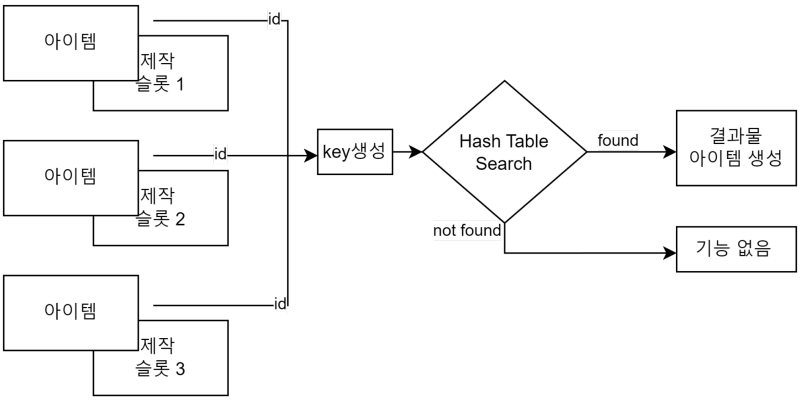


그림 14. 제작이 이루어지는 과정을 표현한 도식

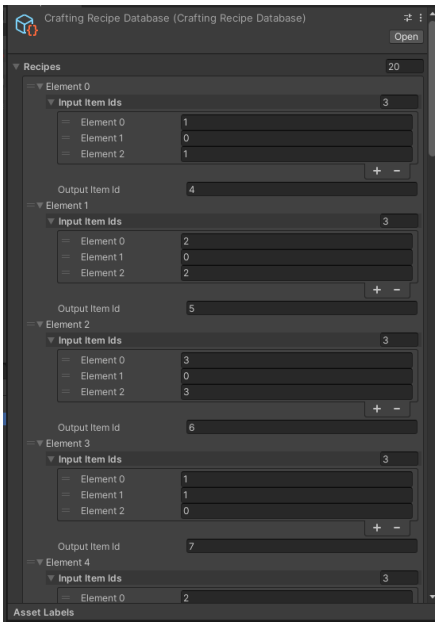


그림 15. Unity Editor에서 직관적인 조합식 관리를 돕는 Scriptable Object의 인터페이스

[State Pattern]

몬스터를 포함한 다양한 객체의 상태 전환을 효율적으로 관리하기 위해 상태 패턴(State Pattern)을 사용한다.

상태 패턴은 객체의 상태를 별도의 클래스로 정의하여 상태 전환을 간단하고 명확하게 관리할 수 있게 한다. 이 패턴을 사용하면 객체가 특정 상태에 따라 다른 동작을 수행할 수 있으며, 상태 전환 로직을 각 상태 클래스에 분리하여 캡슐화할 수 있다. 이는 코드의 유지보수성을 높이고, 확장 가능성을 제공하는 데 큰 도움이 된다.

몬스터 관리 시스템에서 상태 패턴을 적용하여 몬스터가 순찰 상태, 추적 상태, 공격 상태 등 여러 상태를 명확하게 관리할 수 있다. 각 상태는 기본 interface를 상속받는 기반 상태 클래스를 다시 상속받는 개별 클래스로 구현되었으며, 상태 간 전환은 간단한 메서드 호출을 통해 이루어진다. 이를 통해 복잡한 상태 전환 로직을 단순화하고, 코드의 가독성과 확장성을 높일 수 있다.

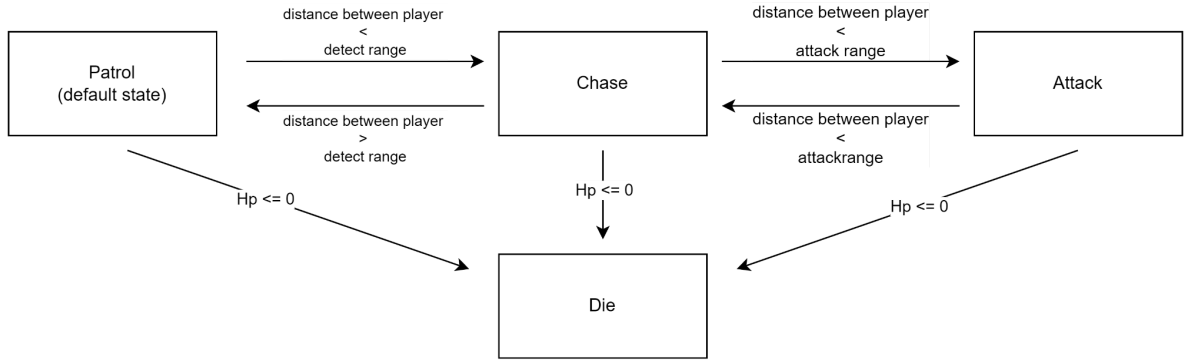


그림 16. 기본적인 몬스터가 따르는 State의 구조

```
public abstract class MobController {
    protected IMobState currentState;

    public IMobState GetCurrentState { return currentState; }

    protected abstract void ChangeState(IMobState newState);

    …
}
```

표 7. 기본적인 몬스터를 관리하는 클래스

```
public interface IMobState
{
   void Enter(MobController mob);
   void Execute();
   void Exit();
   …
}

public class MobStateBase : IMobState
{
   …
}
```

표 8. 몬스터의 모든 State 클래스가 상속받아야 하는 interface와 기반 클래스

```
public class PatrolState : MobStateBase
{
   private MobController mob;

   public PatrolState() { … }

   public void Enter(MobController mob)  { … }

   public void Execute()
   {
      if (mob.IsPlayerDetected())
```

```
    {
        mob.ChangeState(new ChaseState());
        return;
    }

    …
  }

  public void Exit() { … }
}

public class ChaseState : MobStateBase
{
  private MobController mob;
  private Transform playerTransform;

  public void Enter(MobController mob)  { … }

  public void Execute()
  {
    if (distanceToPlayer <= mob.GetAttackRange())
        mob.ChangeState(new AttackState());
    else if (distanceToPlayer > mob.GetChasableRange())
        mob.ChangeState(new PatrolState());
    else  { … }

  }

  public void Exit()  { … }
}

public class AttackState : MobStateBase
{
  private MobController mob;

  public void Enter(MobController mob)  { … }

  public void Execute()
  {
    if (distanceToPlayer <= mob.GetAttackRange())
        if (Time.time - lastAttackTime > attackCooldown)
            Attack();
    else
        mob.ChangeState(new ChaseState());
  }

  public void Exit()  { … }

  private void Attack()  { … }
}
```

표 9. 몬스터의 기본 Patrol, Chase, Attack State

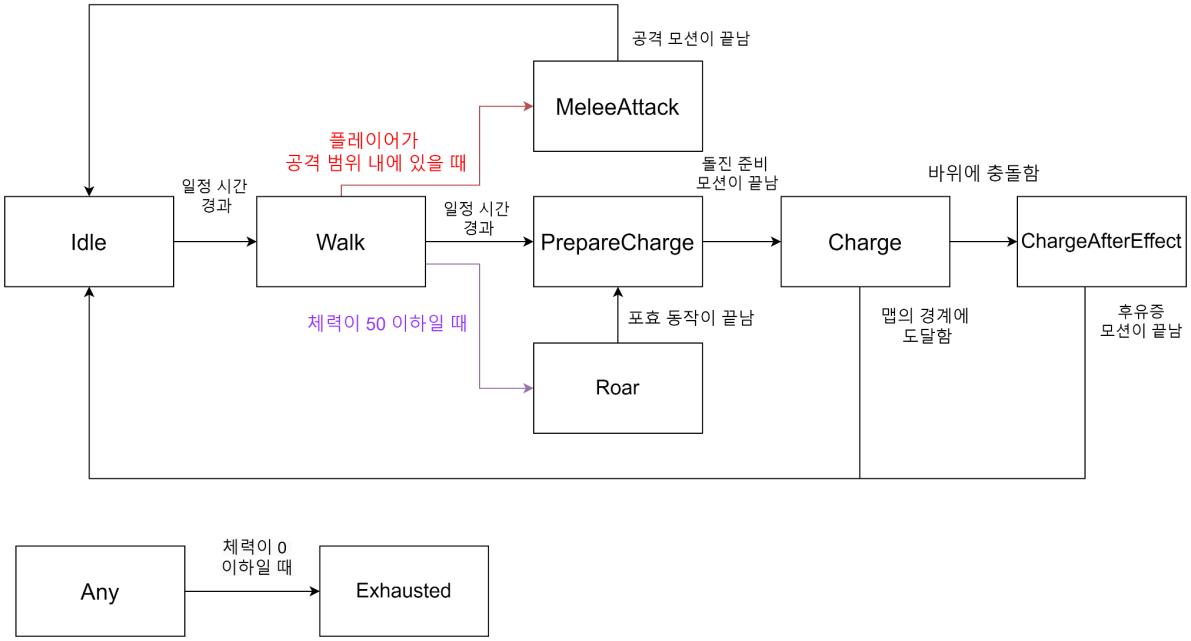


그림 17. 첫번째 보스의 행동 패턴을 나타내는 State Diagram

그림 17은 첫번째 챕터 보스의 공격 패턴을 위해 구현한 State 들의 흐름을 나타낸다. 플레이어가 보스의 공간으로 최초 진입할 때 Boss는 활성화되며, 이 때 IdleState로 초기화된다. 이후에 플레이어의 조작에 따라 다양하게 분기되는 이 구조는 다양한 공격 패턴의 형태로 플레이어에게 난이도를 제공한다.

[싱글톤을 활용한 시스템 매니저 계층구조]

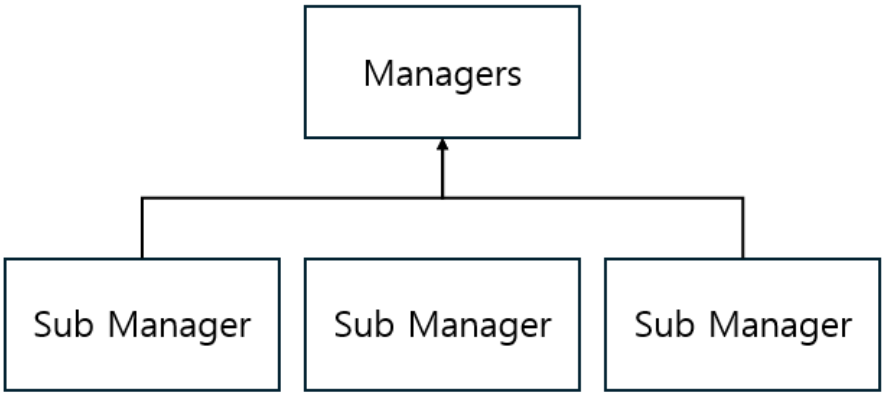


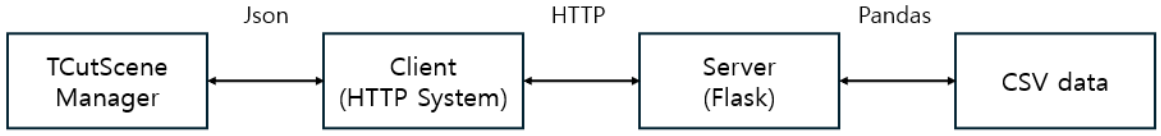
그림 18. 시스템 매니저 계층 구조 다이어그램

그림 18은 시스템 매니저 계층 구조를 나타낸 다이어그램이다. Managers 객체는 시스템 파라미터들을 관리하는 서브 매니저 객체들의 집합이다. 따라서 접근 용이성을 위해 싱글톤 패턴을 적용하였다.

Managers	DataManager	아이템 데이터, 인벤토리, 퀵슬롯, 로그 등을 관리
	PoolManager	게임 오브젝트의 재사용을 위해 풀을 생성하고, 사용된 오브젝트를 풀에 반환하는 기능을 제공
	ResourceManager	리소스 로딩, 인스턴스화, 파괴를 관리

	SceneManagerEx	Scene을 로드하고, 전환하며, 현재 Scene의 상태를 관리
	SoundManager	배경음악(BGM)과 효과음(Effet)을 재생하는 AudioSource를 관리
	CraftingManager	조합 레시피 데이터베이스를 로드하고, 조합 슬롯 및 결과물 슬롯을 관리
	UIManager	팝업 UI와 씬 UI를 관리하며, UI 요소의 생성, 시각화, 삭제를 담당
	ReputeManager	플레이어의 평판을 실시간으로 관리한다.

[클라이언트 서버 간 통신]



게임 스크립트를 로컬에 저장하는 방식을 사용하게 되면 스크립트 내용에 변경 사항이 있을 때마다 업데이트를 진행해야 한다는 단점이 있다. 따라서 스크립트 파일을 서버에 저장하고 클라이언트가 스크립트 파일을 서버에서 로드한다.

4.3. 테스트케이스 설계

주요기능	기능 대분류	기능 소분류	기대하는 결과	수행결과
시스템	인벤토리 시스템	아이템 생성 (불러오기)	저장된 인벤토리 아이템 로드한다.	통과
		아이템 이동	인벤토리 팝업에서 아이템 위치를 옮길 수 있다.	통과
		아이템 저장	현재 인벤토리 아이템을 저장한다.	통과
		아이템 로그	이벤트 발생 시 아이템 로그를 추가한다.	통과
		팝업 UI 시스템	이벤트 발생 시 팝업 UI 시스템이 동작한다.	통과
		아이템 조합	아이템 조합 시 새로운 조합식에 따라 새로운 아이템이 등장한다.	통과
		아이템 강화	아이템 강화 시 아이템 능력치가 조정된다.	미실시
	다이얼로그 시스템	대화	Ink 스크립트가 표시된다.	통과
	캐릭터 시스템	캐릭터 이동	WASD, Shift, Space 를 통해 캐릭터를 이동한다.	통과

		상호작용	상호작용 버튼을 통해 다른 객체와 상호작용한다.	통과
		평판	이벤트 발생에 따라 평판이 변화한다.	통과
		능력치	캐릭터 능력치가 반영된다.	미실시
	컷신 시스템	컷신 시스템	컷신화면이 동작된다.	통과
	전투 시스템	일반 몬스터	일반 몬스터와 전투하여 몬스터의 HP, 상태 등을 변화시킨다.	통과
	보스 전투 시스템	보스 룸	사용자가 특정 위치에 도달하면 보스 룸 시스템이 전투 시작을 트리거한다.	통과
		보스 몬스터	사용자가 보스 몬스터와 전투를 하여 캐릭터 혹은 몬스터의 상태를 변경시킨다.	통과
	상점 시스템	상점 활성화/ 비활성화 단풍 상점 활성화 / 비활성화	상점 NPC와 상호작용을 할 수 있다. 또한 평판에 따라 NPC의 상태가 변경된다.	통과
	플랫폼 시스템	일반 플랫폼		
		특수 플랫폼		
	사운드 시스템	사운드 설정	캐릭터의 주변환경, 상호작용에 따라 적절한 사운드가 출력된다.	통과
레벨 디자인	Chapter 0	튜토리얼		

표 12. 테스트 케이스 명세서

[구체적인 시나리오**]**

- 게임 스토리에 관한 요약
    - 텍스트 위주의 컷씬

1. 초기 단풍 상점이 나타난 것을 보여준다
    - NPC 기능
    - 평판 기능
    - 단풍 삼정 그래픽 리소스
2. 나무**,** 돌 광맥 곡괭이로 채집한다
    - 엔티티 필요
        - 엔티티 상호작용 기능
        - 아이템 드롭 기능
    - 아이템 획득 기능
    - 도구 그래픽 리소스
3. 채집 후 아이템 조합 가능함을 보여준다**.** 돌 곡괭이를 제작한다
    - 조합 기능
    - 아이템 손에 들기
    - 도구 그래픽 리소스

4. 근처에 있는 **NPC**로부터 장벽을 깨달라는 요청을 받는다
    - NPC 대화
    - 퀘스트 시스템
    - 장벽 그래픽 리소스
    - NPC 그래픽 리소스
5. 하지만 돌 곡괭이로 격파할 수 없음을 보여준다.
    - 엔티티 상호작용 기능
        - 불가능한 버전
    - 장애물 엔티티와 상호작용
        - 불가능한 버전
6. 드릴을 제작할 수 있도록 한다.
    - 조합 기능
    - 드릴 그래픽 리소스
7. 하지만 드릴로도 잘 격파가 되지 않는다.
    - 엔티티 상호작용 기능
        - 불가능한 버전
8. 더 좋은 품질의 드릴을 획득할 수 있도록 한다.
    - 조합 기능
9. 장벽이 격파된다.
    - 장애물 엔티티와 상호작용
        - 성공한 버전
10. 평판이 상승한 모습을 보여준다.
11. 우호 상점 등장
    - NPC 기능
    - 평판 기능
    - 우호상점 그래픽 리소스
12. 보스 방에 진입하며 컷씬이 등장한다.
    - 보스 방 기능
    - 컷씬 기능
    - 보스 그래픽 리소스
13. 보스 전투
    - 보스 기능

4.3. 개발환경

Unity Editor 2022.3.16f1

IDE : JetBrains Rider, Visual Studio, Visual Studio Code

협업 : Git, Notion, Google Spreadsheet

그래픽 : Clipstudio, Photoshop

4.4. 구성요소별 기능 구현

[DialogueManager]

전체적인 대화를 관리하는 매니저로 npc와 player의 행동에 따라 npc에게 저장된 대화를 출력하고 선택을 가능하게 하며 대화를 끝내주는 대화 전체의 흐름을 조절해 준다.

```
public class DialogueManager : MonoBehaviour
{
    private static DialogueManager instance;  // 싱글턴 인스턴스
    private bool dialogueIsPlaying;  // 대화 진행 상태
    private Story currentStory;  // 현재 진행 중인 스토리
    private NpcData npcdata;  // 현재 대화 중인 NPC 데이터
    private UI_DialoguePopup popup;  // 대화 UI 팝업

    private void Awake()
    {
        // 인스턴스 초기화 및 데이터 로드 준비
    }

    public static DialogueManager GetInstance()
    {
        // 싱글턴 인스턴스 반환
    }

    public int GetQuestIndex(int id)
    {
        // NPC의 퀘스트 인덱스 반환
    }

    public void SetQuestIndex(int id, int idx)
    {
        // NPC의 퀘스트 인덱스 설정
    }

    private void Update()
    {
        // 대화 중 플레이어 상호작용 감지 및 대화 진행
    }

    public void GetTalk2(TextAsset dialogue, NpcData npc)
    {
        // 대화를 시작하고 UI 초기화
    }

    private void ExitDialogueMode()
    {
        // 대화 모드를 종료하고 플레이어 상호작용 재활성화
    }

    private void ContinueStory()
    {
```

```
        // 대화를 이어가며, 태그와 선택지를 처리
    }

    private void HandleTags(List<string> currentTags)
    {
        // 대화 중 태그를 처리하여 UI 요소 업데이트
    }

    private void DisplayChoices()
    {
        // 현재 대화 선택지를 UI에 표시
    }

    private IEnumerator SelectFirstChoice()
    {
        // 첫 번째 선택지를 자동 선택
    }

    public void MakeChoice(int choice)
    {
        // 플레이어의 선택을 처리하고 퀘스트 및 상점 여부 확인
    }

    public void SetVisibleNpc()
    {
        // NPC의 가시성 설정
    }
}
```

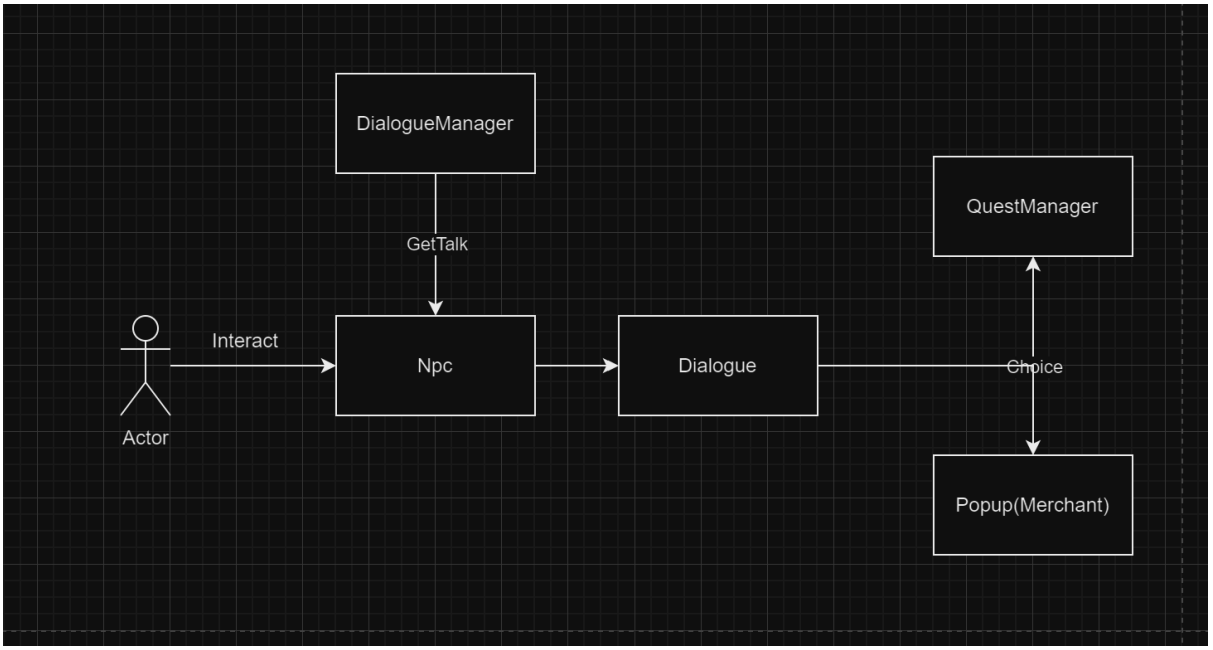


그림19.대화 관리 매니저 및 실제 대화

[Quest]

각각의 퀘스트는 Abstract Class인 QuestData를 통해 정의되며 모든 Qeust들은 QuestManager을 통해 관리된다. 그리고 각각의 Quest의 Quest State는 Player 나 Entity의 상태에 따라 Unity event를 Invoke시키는 것으로 업데이트 된다.

```
public class DestroyBlock : QuestData
{
    public UnityEvent QuestCompleted;
    private static DestroyBlock instance;


    public override string getQuestInfo()
    {
        return "앞에 있는 벽을 부숴주세요";
    }

    public override void updateQuest()
    {
        qs++;
        QuestCompleted.Invoke(); // 퀘스트 완료 이벤트 호출
        Debug.Log("벽 부수기 퀘스트 완료!");
    }
}
```

표 13. QuestData를 통해 구현된 DestroyBlock class

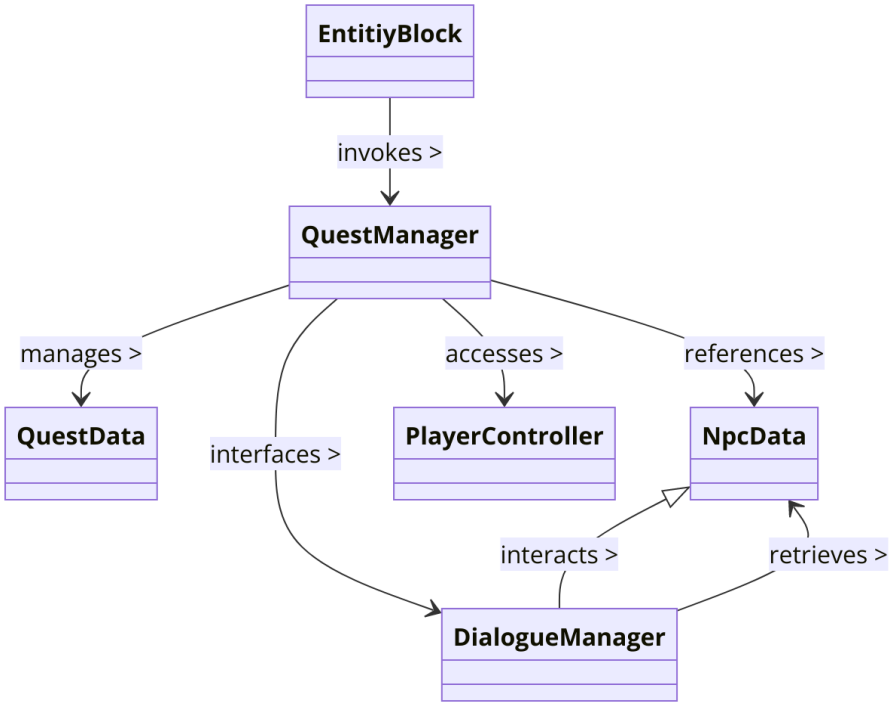
EntitiyBlock

```
public class EntitiyBlock : EntityInfo
{
    [Header("EntityInfo")]
    [SerializeField] private int EHp;
```

```
public bool playerInRange;
public UnityEvent BlockDead;
public static EntitiyBlock Instance;
public NpcData npc;
private void Awake()
{
    EHp = 100;
    Instance = this;
    playerInRange = false;
    if (BlockDead == null)
    {
        BlockDead = new UnityEvent();
    }
}

private void FixedUpdate()
{
    if (playerInRange )
    {
        if (PlayerController.GetInstance().GetInteractPressed())
        {
            Damaged(50);
        }
    }
}

public static EntitiyBlock GetInstance()
{
    return Instance;
}

private void OnTriggerEnter2D(Collider2D collider)
{
    playerInRange = true;
    if (collider.tag == "Player")
    {
        npc = collider.gameObject.GetComponent<NpcData>();
    }

}

public override void Damaged(int damage)
{
    if (QuestManager.GetInstance().CheckState(0) == QuestState.IN_PROGRESS)
    {
        if (EHp - damage > 0)
        {
            EHp -= damage;
            DebugEx.Log("Damage : " + damage + "!");
        }
        else
        {
            EHp = 0;
```

```
            DebugEx.Log(EHp);
            Debug.Log(QuestManager.GetInstance().CheckState(0));
            if (QuestManager.GetInstance().CheckState(0) == QuestState.IN_PROGRESS)
            {
                Destroy(this.gameObject);
                BlockDead.Invoke(); // 엔티티가 죽을 때 이벤트 호출
            }
        }
    }

  }

}
```

표 14. Entity가 죽으면 Unity event가 호출되며 QuestState가 업데이트



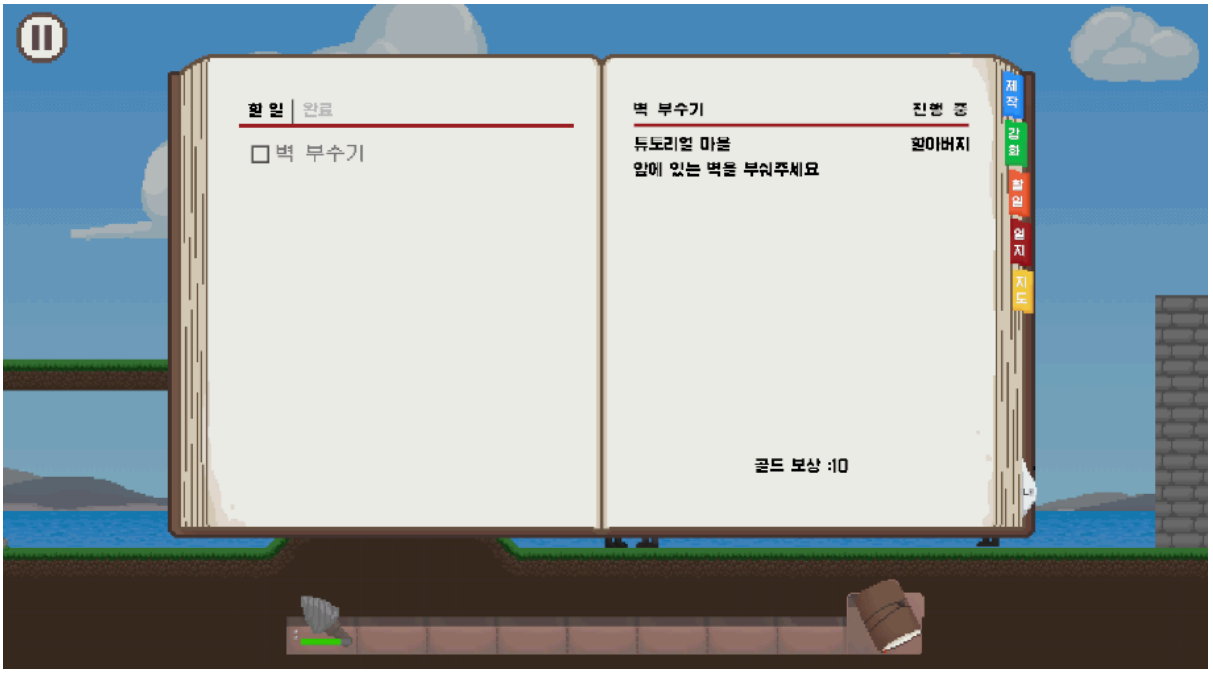


그림20. Quest 관계도 및 실제 퀘스트 화면

[Repute Manager]

Repute Manager은 평판을 관리해주는 매니저로 퀘스트와 플레이어 선택에 따라 플레이어의 평판이 달라지게 해주며 이에 따라 Repute State도 달라지게 된다.

Repute Manager

```
public class ReputeManager
{
    //평판관리 시스템

    //평판 점수
    [SerializeField] private int ReputeScore;
    [SerializeField] private ReputeState ReputeState;


    public void Init()        //제일 초기에는 평판이 50
    {
        ReputeScore = 50;
        ReputeState = SetReputeState(ReputeScore);
        DebugEx.Log("현재 평판 수치 : " + ReputeScore + " 평판 " +
GetReputeState(ReputeState));

    }

    public ReputeState SetReputeState(int score)
    {
        if (score >= 60)
        {

            return ReputeState.Good;
        }
```

```
        else if (score <= 60 && score >= 40)
        {
            return ReputeState.Normal;
        }
        else
        {
            return ReputeState.Bad;
        }

    }
    public string GetReputeState(ReputeState reputeState)
    {
        switch (reputeState)
        {
            case ReputeState.Good:
                return "선함";
            case ReputeState.Normal:
                return "보통";
            case ReputeState.Bad:
                return "악명";
            default:
                return "";

        }

    }

    public ReputeState GetRepute()
    {
        return this.ReputeState;
    }


    public void SetRepute(int score)
    {
        ReputeScore = score;
    }

    public void AddRepute(int score)
    {
        ReputeScore = Mathf.Min(ReputeScore + score, 100);
        ReputeState = SetReputeState(ReputeScore);
        PlayerController player =
Managers.Game.GetPlayer().GetComponent<PlayerController>();
        player.RState = ReputeState;
        if (ReputeState == ReputeState.Good)
        {
            DialogueManager.GetInstance().setvisibleNpc();
        }
        DebugEx.Log("현재 평판 점수 : " + ReputeScore + "평판도 " +
GetReputeState(ReputeState));
    }

    public void DecreaseRepute(int score)
```

```
    {
        ReputeScore = Mathf.Max(ReputeScore - score, 0);
        ReputeState = SetReputeState(ReputeScore);
        DebugEx.Log("현재 평판 점수 : " + ReputeScore + "평판도 " +
GetReputeState(ReputeState));
    }
}

    public void AddRepute(int score)
    {
        ReputeScore = Mathf.Min(ReputeScore + score, 100);
        ReputeState = SetReputeState(ReputeScore);
        DebugEx.Log("현재 평판 점수 : " + ReputeScore + "평판도 " +
GetReputeState(ReputeState));
    }

    public void DecreaseRepute(int score)
    {
        ReputeScore = Mathf.Max(ReputeScore - score, 0);
        ReputeState = SetReputeState(ReputeScore);
        DebugEx.Log("현재 평판 점수 : " + ReputeScore + "평판도 " +
GetReputeState(ReputeState));
    }
}
```

표 15. ReputeManager

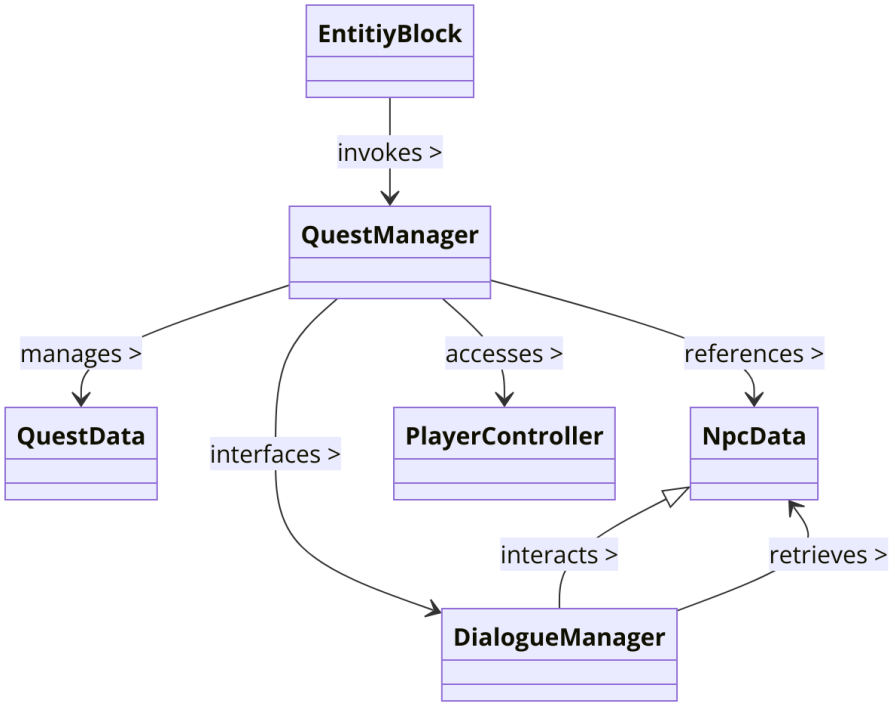


그림 21. 퀘스트 전체 흐름도

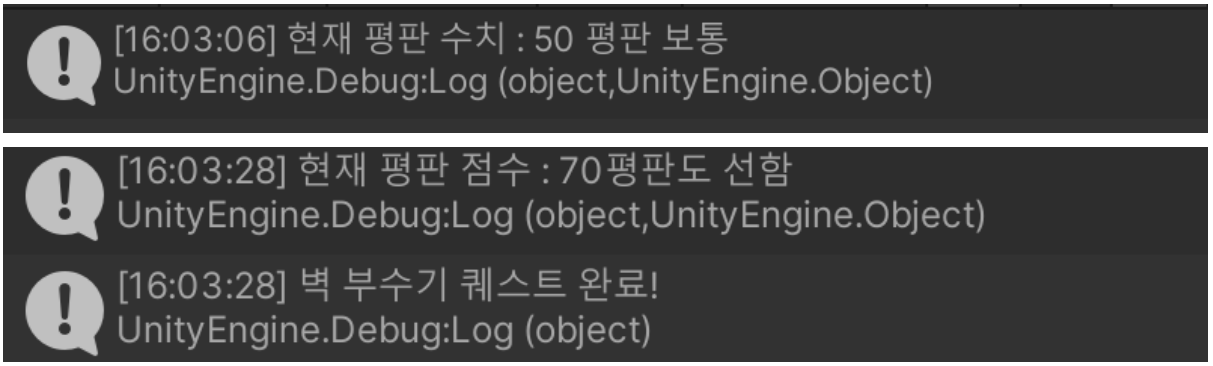


그림 22. 퀘스트 진행에 따른 평판 변화

[상점**]**

상점은 평판에 따라 우호상점, 평판 상점으로 사용할 수 있는 것이 나눠져 있다. 만약 평판이 좋다면 우호 상점을 이용할 수 있다. ,팝업 형태로 구현되어있으며 사용자의 인벤토리를 같이
볼 수 있다.

```
public class UI_Merchant_PlayerInven : UI_Popup, ICatcher
{
    private const int numberOfItemSlots = 30;  // 기본 아이템 슬롯 수
    [SerializeField] private List<UI_Merchat_Player_Inven_Slot> itemSlots;  // 인벤토리
슬롯 리스트
    [SerializeField] private List<UI_Inven_Item> visualizedItems = new
List<UI_Inven_Item>();  // 시각화된 아이템 리스트
    private List<ItemData> _itemDataList = new List<ItemData>();  // 아이템 데이터 리스트
    [SerializeField] public GameObject VisualizedLayer;  // 시각화 레이어
    [SerializeField] public Confirm_Player ConfirmBuy;  // 구매 확인 팝업
    private UI_Inven_ItemTooltip tooltip;  // 아이템 툴팁
    public UI_Merchant merchant;  // 상점 객체
    public bool isBuyActive = false;  // 구매 활성화 상태
    public UI_Inven_Item CatchedItem { get; set; }  // 잡힌 아이템

    private void OnEnable()
    {
        // 데이터 저장 이벤트 등록
        Managers.Data.OnClose -= ExportInventoryData;
        Managers.Data.OnClose += ExportInventoryData;
    }

    public void InitInventory(GameObject visualizedLayer, UI_Inven_ItemTooltip itemTooltip)
    {
        // 인벤토리 초기화 및 시각화
    }

    public void VisualizeItemsInTheGrid(bool initialize = false)
    {
        // 아이템을 그리드에 시각화
    }
```

```
    private void GetItemDataFromDataManager()
    {
        // 데이터 매니저에서 아이템 데이터 가져오기
    }

    public void ExportInventoryData()
    {
        // 인벤토리 데이터를 저장
    }

    private ItemData ConvertToItemData(UI_Inven_Item itemUI)
    {
        // UI 아이템을 데이터로 변환
    }

    public override void ClosePopupUI(PointerEventData action)
    {
        // 팝업 닫기 및 데이터 저장
    }

    private void EnablePickupItem()
    {
        // 아이템 줍기 기능 활성화
    }

    public void ActivateConfirmBuy()
    {
        // 구매 확인 팝업 활성화
    }

    public void DeactivateConfirmBuy()
    {
        // 구매 확인 팝업 비활성화
    }
}
```

그림23.우호상점(좌) 및 단풍 상점(우)

4.5. 업무 분담

업무 분담	인원
기획	고희수, 김현찬, 배정훈, 이경현
개발	김현찬, 이경현, 배정훈
아트	고희수

표16. 업무 분담표

4.6. 개인별 개발내용

기획

배경 : 제 3차 세계대전 후 15~20년후 포스트 아포칼립스

전체 스토리 흐름

1. 전쟁 발생 - 할아버지와 손자가 전쟁이 일어나자 섬 마을로 피난
2. 전쟁 끝 - 핵폭발
3. 섬마을
    1. 10대 후반인 손자가 할아버지와 지내다가 할아버지의 심부름 때문에 항구 도시로 이동
4. 항구 도시
    1. 심부름을 하러 갔던 항구 도시에서 도시의 일을 도와주면서 부모님의 흔적을 발견하게 된다. 흔적을 따라 가다 보니 다른 도시로 가야 한다는 것을 알게 되지만 막혀있어 그 길을 뚫고 가려고 한다.
    2. 길을 뚫기 위해 사람들을 도우면서 재료를 모으던 도중 조직(단풍)에 대해 알게 된다. 길을 뚫으러 가지만 그 길에 오염된 곰이 등장, 이 곰을 물리치고 다음 도시로 가는 길을 뚫게 된다. 철 도시까지는 마을에서 알게 된 나그네의 오토바이를 타고 이동

챕터1 스토리 개발 완료

챕터2 스토리 40% 완료

시나리오

Chap 0.전쟁 전

- #Scene1 첫시작 (플레이어블)
- #Scene2 피난길 컷신
- #Scene3 항구 도시 속 친구 집- Playable

Chap 1. 전쟁 후 섬 마을

- #Scene5 집
- #Scene6 고물 처리장
- #Scene 7: 시장
- #Scene8: 산
- #Scene9: 할아버지 공방 ~ 집

Chap 2. 항구도시

- Scene10: 항구도시
- #Scene11 병원
- #Scene12: 시청

Chap 0.전쟁 전

#Scene1 첫시작 (플레이어블)

[한밤중, 집안을 비추고 있는 화면]

[사이렌이 울리고 소란스러운 소리가 들린다.]

[할아버지 캐릭터가 놀라서 잠에서 깨 창문을 열어 바깥을 본다.]

[산 넘어 미사일이 폭발하며 보이는 빛이 보인다]

[티비에선 공습경보라는 뉴스 속보가 계속 나온다.]

TV: 현 시간부로 전국에 공습경보를 발령합니다. 이 방송은 훈련 상황이 아닌 실제 상황입니다.

[아이가 울면서 깬다.]

[==Playable Moment==(할아버지 플레이어가 손자에게 가서 상호작용 키를 누르도록)]

할아버지: (손자를 안으며)이게 대체...

[==Playable Moment==(할아버지 플레이어가 테이블 위의 전화기에 가서 상호작용 키를 누르도록)]

[전화기를 보니 여러 통의 부재중 전화와 여러 통의 문자가 와있었는데 그중에 친구에게서 온 다급한 문자 한 통을 보았다.]

친구의 문자: 문자를 보거든 얼른 내 집으로 와!

[할아버지는 아이를 안고 비상시를 대비해 챙겨둔 가방을 등에 멘 뒤 집을 나선다.]

→ 일러스트 출력

그림24.시나리오 작성현황

주인공이 포스트 아포칼립스 세계에서 다양한 사람들과의 만남과 선택에 따라 변하게 될 수 있도록 스토리의 분기점과 다양한 선택을 고려하여 시나리오를 작성했다.

아이템 스텟

아이템 스텟은 각 능력치에서 상호작용할 수 있는 개체 중 제일 기본이 되는 것(나무, 바위, 폐 전자기기)을 2번 안에 상호작용을 하여 재료를 얻을 수 있도록 만들었다.

도구	ID	벌목	분해	채광	공격	주 능력치
돌 곡괭이	**128**	**10**	**5**	**15**	**7**	채광
돌 도끼	**129**	**15**	**5**	**10**	**10**	벌목
돌 망치	**130**	**1**	**10**	**5**	**5**	분해
돌 드라이버	**131**	**1**	**10**	**1**	**3**	분해
돌 단검	**132**	**5**	**1**	**1**	**15**	공격
활	**133**	**1**	**1**	**1**	**10**	공격
나무 몽둥이	**134**	**1**	**1**	**5**	**10**	공격
철 곡괭이	**135**	**15**	**10**	**20**	**10**	채광
철 도끼	**136**	**20**	**10**	**15**	**13**	벌목
철 망치	**137**	**1**	**10**	**15**	**8**	벌목
철 드라이버	**138**	**1**	**15**	**1**	**5**	분해
철 단검	**139**	**7**	**1**	**1**	**20**	공격
철 단검	**140**	**30**	**30**	**30**	**30**	메카닉

Entity 스탯

개체명	체력	필요 능력치
나무	**20**	벌목
바위	**20**	채광
철광석	**30**	채광
폐 자동차	**30**	분해
폐 전자기기	**20**	분해
사이버트럭	**40**	분해

개발

아트

작업환경과 인게임 환경에서 해상도 차이로 인해 의도한 크기로 나오지 않는 문제가 발생하여 이를 해결하기 위한 자료이다.

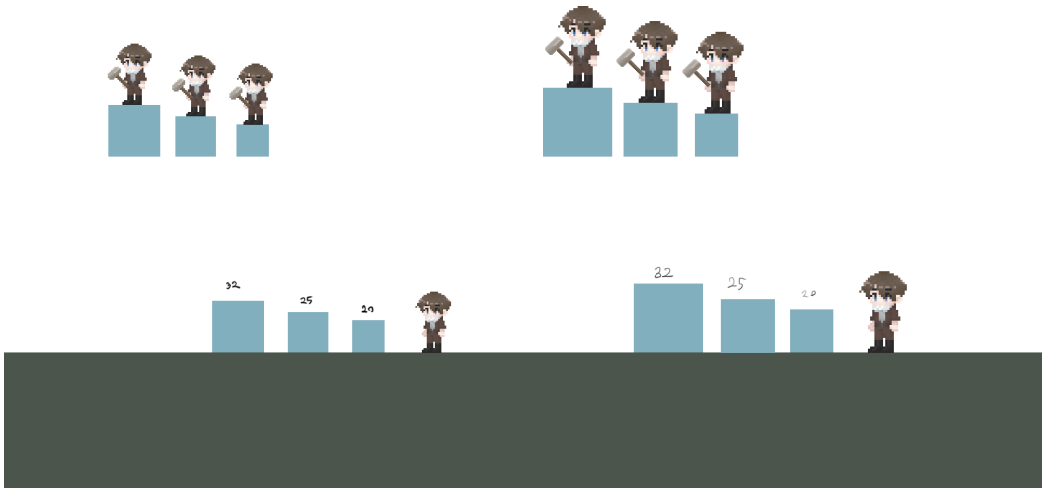


그림 25. 타일, 해상도 조정을 위해 제작했던 자료

타일크기는 32px에서 작업하고, 작업환경에서 해상도를 조정하여 넘기는 것으로 해결했다.

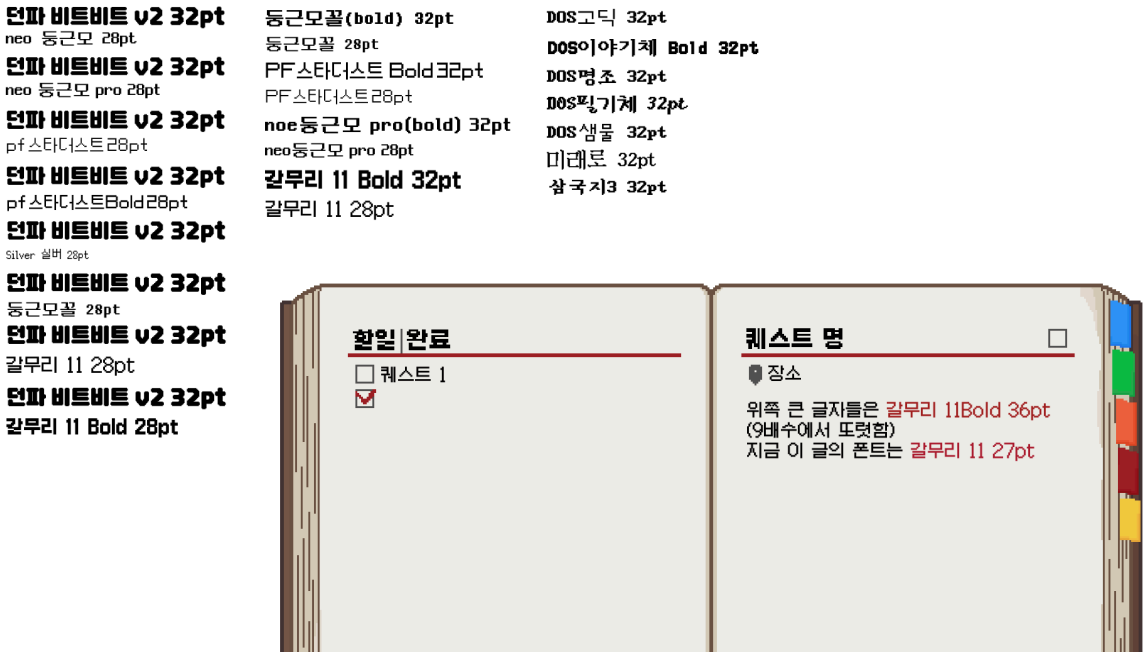


그림26. 폰트 후보군

일반 설명글이나 대화 등에 들어갈 폰트와 주요 메뉴를 표기할 때 사용할 두꺼운 폰트의 괴리가 적은 폰트들을 선정하기 위해 제작한 자료이며, 최종적으로 갈무리 11을 사용하기로 결정했다.

타이틀 로고를 제작하기 위해 폰트를 추려 제작했다. 할아버지의 일지가 플레이어에게 세상을 알려주며 이야기를 진행하는 요소 중 하나이기 때문에 로고에 일지의 가름끈을 바람에 흩날리는 형상으로 넣었다.

그림 27. 배경 시안과 최종 배경

횡스크롤 게임임을 감안하여 각 객체별 레이어를 분리하여 작업한 후 각 레이어별 파일을 제공했다. 전쟁의 화마가 닿지 않은 섬이므로 평화로운 느낌을 강조하고자 시안에 비해 전반적인 채도를 높여 작업했다.

위 배경에 맞춰 타일맵과 자원을 획득할 수 있는 객체들을 제작했다.

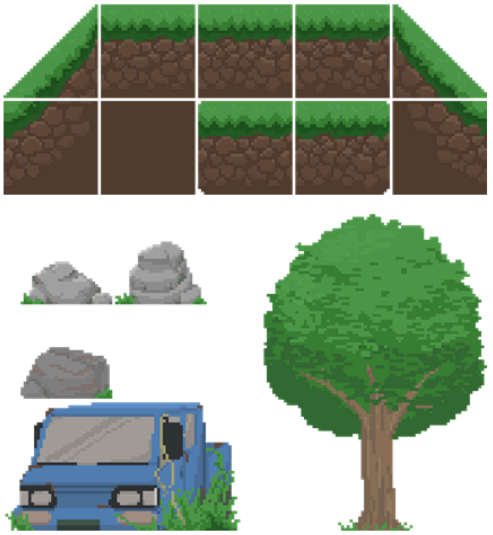


그림 28. 타일맵

플랫포머의 특성상 공중에 떠 있는 플랫폼이 존재할 수 있기 때문에 여러 형태의 타일을 제작했고, 곡괭이로 얻을 수 있는 광물인 돌과 철 각각 다른 형태로 작업했다.

할아버지의 일지를 테마로 잡고 전반적인 ui를 제작했다.

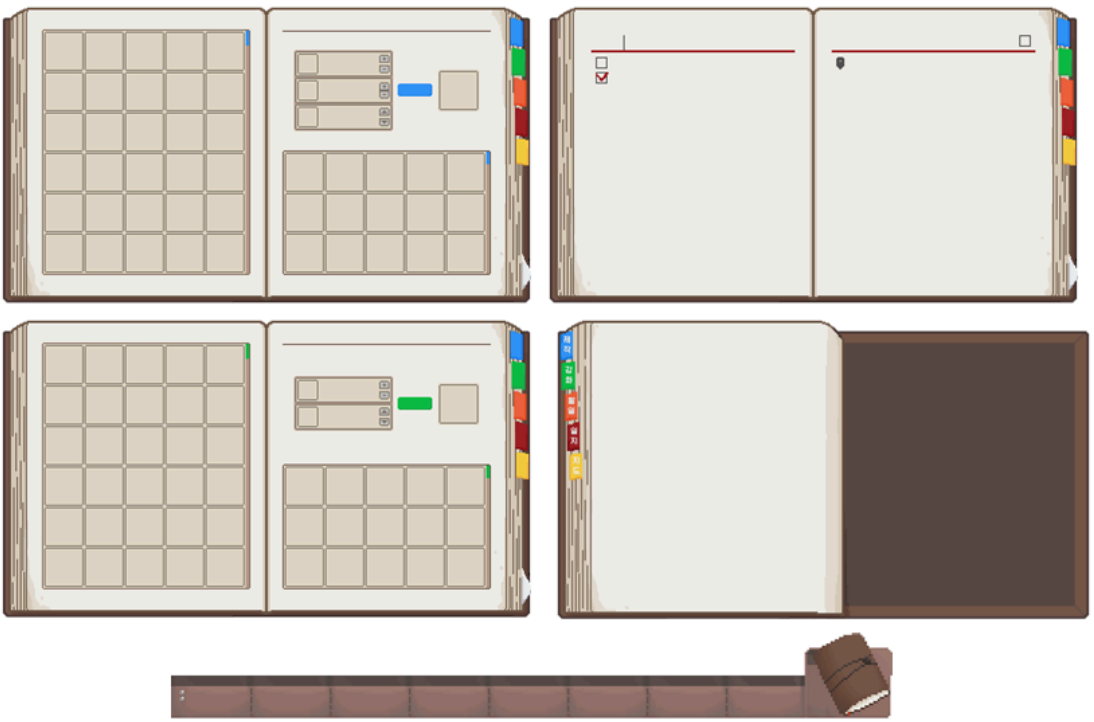


그림 29. 메뉴와 퀵슬롯

메인메뉴인 수첩에서 그 테마가 가장 잘 드러나며, 빛바랜 종이와 종이의 얼룩이 오래된 일지의 느낌을 살린다. 퀵슬롯은 일지가 들어간 가방과, 도구 벨트를 테마로 작업했다.

그림 30. 도구와 아이템

도구는 소재별로 색을 다르게 만들어 다른 도구라는 것을 표현했고, 아이템은 막대나 판 같은 소재별로 공통적으로 가지는 재료들은 고정된 틀을 두고 소재 별로 색을 바꿔 표현했다.

그림 31. 플레이어 시안과 초상화

플레이어의 외관과 옷 디자인은 여러개의 시안 중에 골라 제작했다. 주인공이 아직 어리고, 성장해 나갈것임을 강조하기 위해 설정상의 실제 나이보다 더 어린 모습으로 작업했다. 가장 많이 보고, 조작해야하는 캐릭터이기 때문에 사람들에게 호불호가 덜할 머리색과 눈 색을 선택했다.

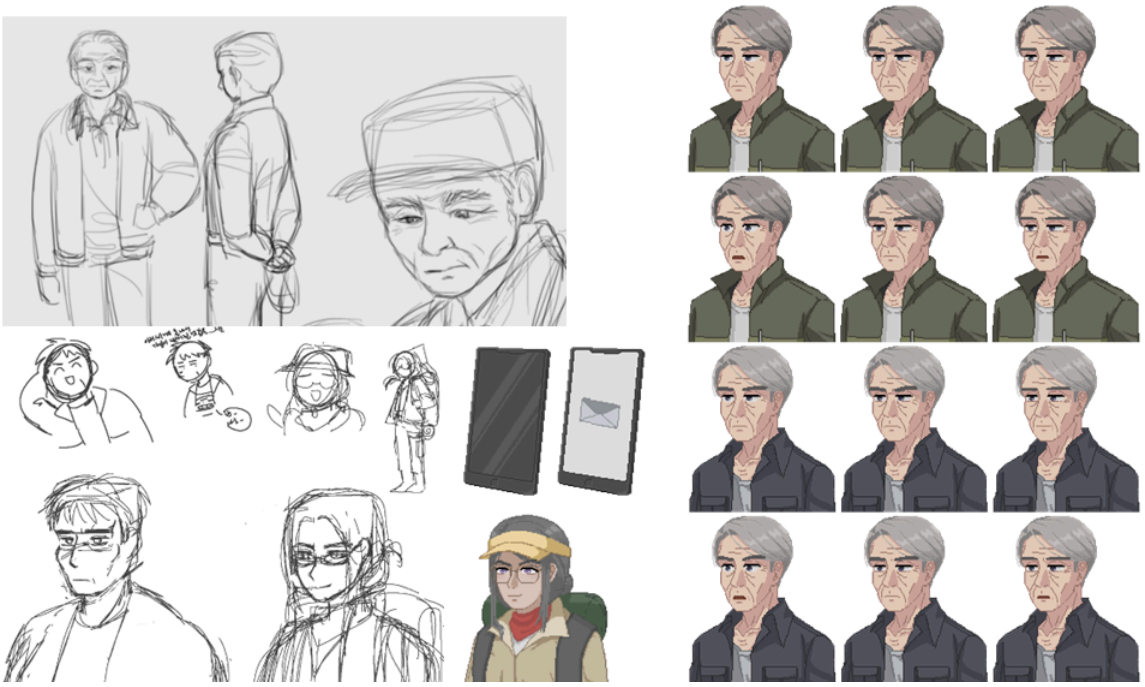


그림 32. NPC 시안과 작업물

주변 NPC들은 확실하게 어른으로 보여 주인공과 대조되도록 제작했다. 특히 할아버지의 경우 성격을 반영하기 위해 시안에 비해 인상이 많이 바뀌었다.

그림 33. 인게임 도트

팔다리가 긴 버전과, 짧은 버전의 뼈대를 제작했다. 아기자기한 도트게임 특유의 분위기를 살리기 위해 팔다리가 짧은 버전의 뼈대를 두고 캐릭터 도트를 제작했다.

몬스터는 야생동물이 모티브이지만 주변 그래픽이 단순한 편이므로 보호색을 띄는 특유의 털 색을 묘사하기 보다는 최대한 단색으로 표현해 묘사를 덜어 제작했다.

그림 34. 보스 몬스터와 일반 필드 몬스터

보스몬스터는 반달가슴곰을 모티브로 삼았다. 연구실에서 탈출한 곰이라는 설정을 가지고 있기 때문에 앞발과 뒷발에 구속을 당했던 흔적이 있고, 도망치면서 다친 상처도 표현했다. 다른 일반 필드 몬스터는 각각 우리나라에서 자주 관찰할 수 있는 고라니와 너구리로, 게임 초반부이기 때문에 아주 험악하지 않은 인상으로 제작했다.

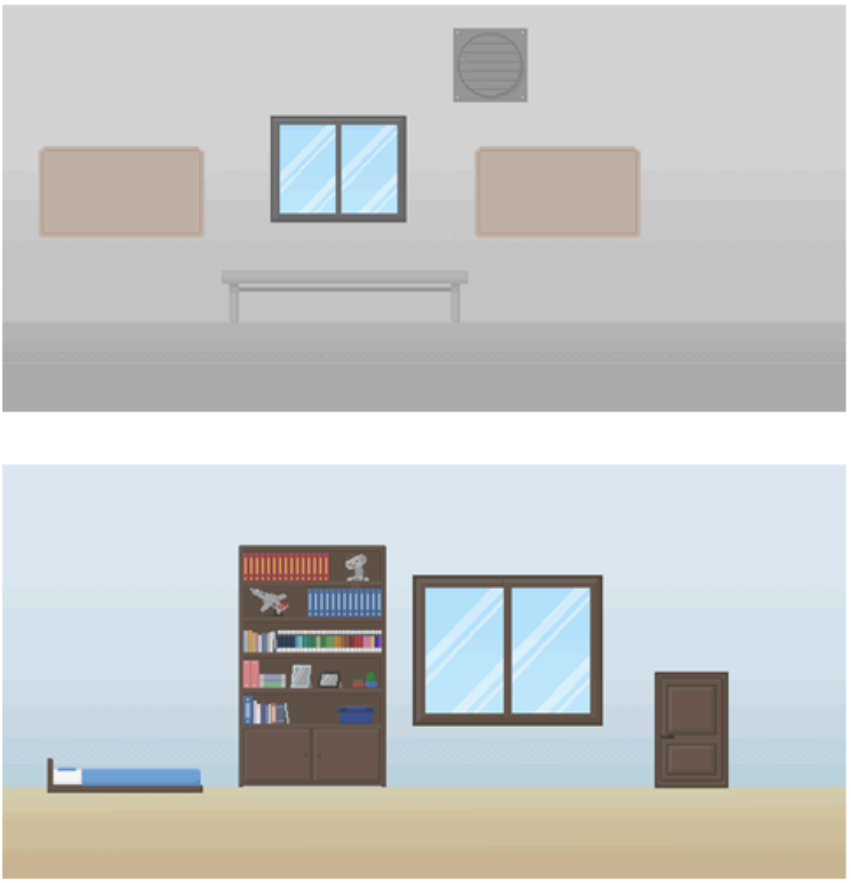


그림 35. 할아버지 작업실과 플레이어의 방

스토리 진행에 필요한 물체와 배경을 작업했다. 플레이어의 방은 할아버지의 생활방식이 녹아있는 책장과, 세월은 느껴지지만 잘 정돈된 가구의 느낌을 살리고자 했다. 할아버지의 작업실은 단정하고 잘 정돈된 모습을 보여주고자 했다.

5. 개발결과 - 구현 결과

본 프로젝트를 24년도 1학기 동안 진행한 결과로, 앞서 언급한 기능이 구현되어있어 이를 직접 체험해볼 수 있는 Demo 버전의 게임을 개발했다.

Demo 버전의 게임 특성 상 게임의 많은 부분을 보여주지는 못하지만, 전반적인 게임의 기반 시스템과 간단한 컨텐츠를 보여줄 수 있어 짧은 시간안에 게임에 대해 소개하고자하는 목적에 충분히 부합하다고 판단했다.

본 Demo 버전의 게임에서 등장하지 못한 나머지 스토리와 컨텐츠, 그리고 게임이 앞으로 나아가야 할 방향이 남아있다. 이는 현재 프로젝트 이후의 계획으로 추가 개발을 통해 더욱 높은 완성도와 많은 볼륨으로 게임을 보완한 후에, 온라인 전자 소프트웨어 유통망인 Steam에 출시하는 것을 목표로 하고 있다.

6. 결론

6.1. 기대 효과 및 활용방안

이번학기에 개발한 Demo 버전의 여러 기능과 구상한 여러 시나리오들을 활용하여 추가적인 개발을 통해 완성도 있는 게임을 출시한다는 목표를 달성했을 경우, 수익창출의 가능성 또한 존재한다. 또, 그 이전에 스토리가 포함된 소규모 컨텐츠로 클라우드 펀딩 이나 투자 유치 또한 받을 수 있을 것이다.

새바람에서 취하고 있는 멀티엔딩 방식은 유저의 선택에 따라 달라지는 다양한 엔딩과 속에 담겨있는 여러 사람의 이야기를 통해 게임의 내용을 좀 더 입체적으로 전달할 수 있다. 이에 따라 플레이어들의 몰입도를 높여 새로운 경험을 제공할 수 있다.

프로젝트 새바람에서 팀원들은 그동안 학과에서 배운 전공 지식을 동원해 여러 기능을 개발하며 기술적 노하우를 쌓았다. 이렇게 강화된 역량은 미래 다른 프로젝트 등에도 활용될 수 있으며, 좀 더 완성도 높은 버전을 출시한다는 목표를 달성하기에도 큰 도움이 될 것이다.