

TextRPG

WINDOWS 11조

내일배움캠프
Unity 7기

목차

1 팀원 소개

2 시연

3 구상

- 와이어프레임

4 메인 로직

5 핵심기능

- 핵심 기능의 동작원리와 포인트

6 소감

- 팀 프로젝트를 진행하면서 느낀 점

1 팀원 소개



신소현

- 인벤토리 담당
- 포션사용 담당



김지환

- 플레이어 담당



이상훈

- 상점 담당
- 보상 담당
- 저장 담당



김지현

- 메인 시스템 로직 담당
- 퀘스트 담당



박성주

- 던전 담당
- 전투 담당



02

영상 시연

WINDOW 11 조의 TEXT RPG 시연영상 입니다!

상훈님 편집 감사합니다!

D:\code\WcsWTeamSparta\WW X + v

스파르타 던전에 오신 여러분 환영합니다.
원하시는 이름을 설정해주세요
>>

게임 시작 화면

X

1.

게임 시작 화면

2.

로비 화면

*

3.

전투 구현 - 치명타 / 스킬

4.

저장 / 불러오기

X

5.

던전 진행도

6.

인벤토리 - 장착

7.

퀘스트

7.

상점 - 퀘스트 아이템 판매

+

8.

상점 - 일반 아이템 구매/판매

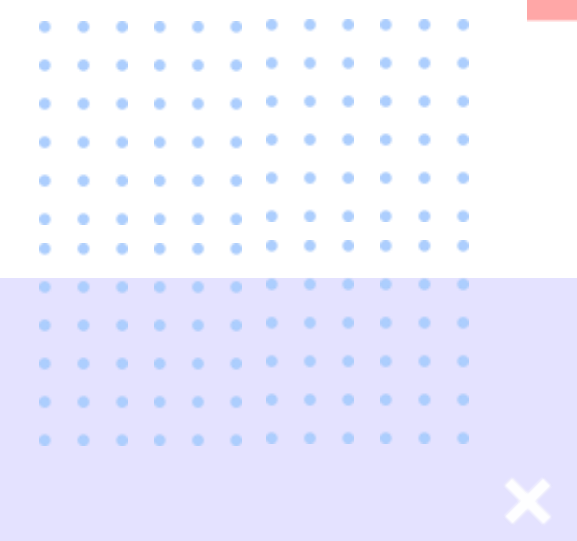
X



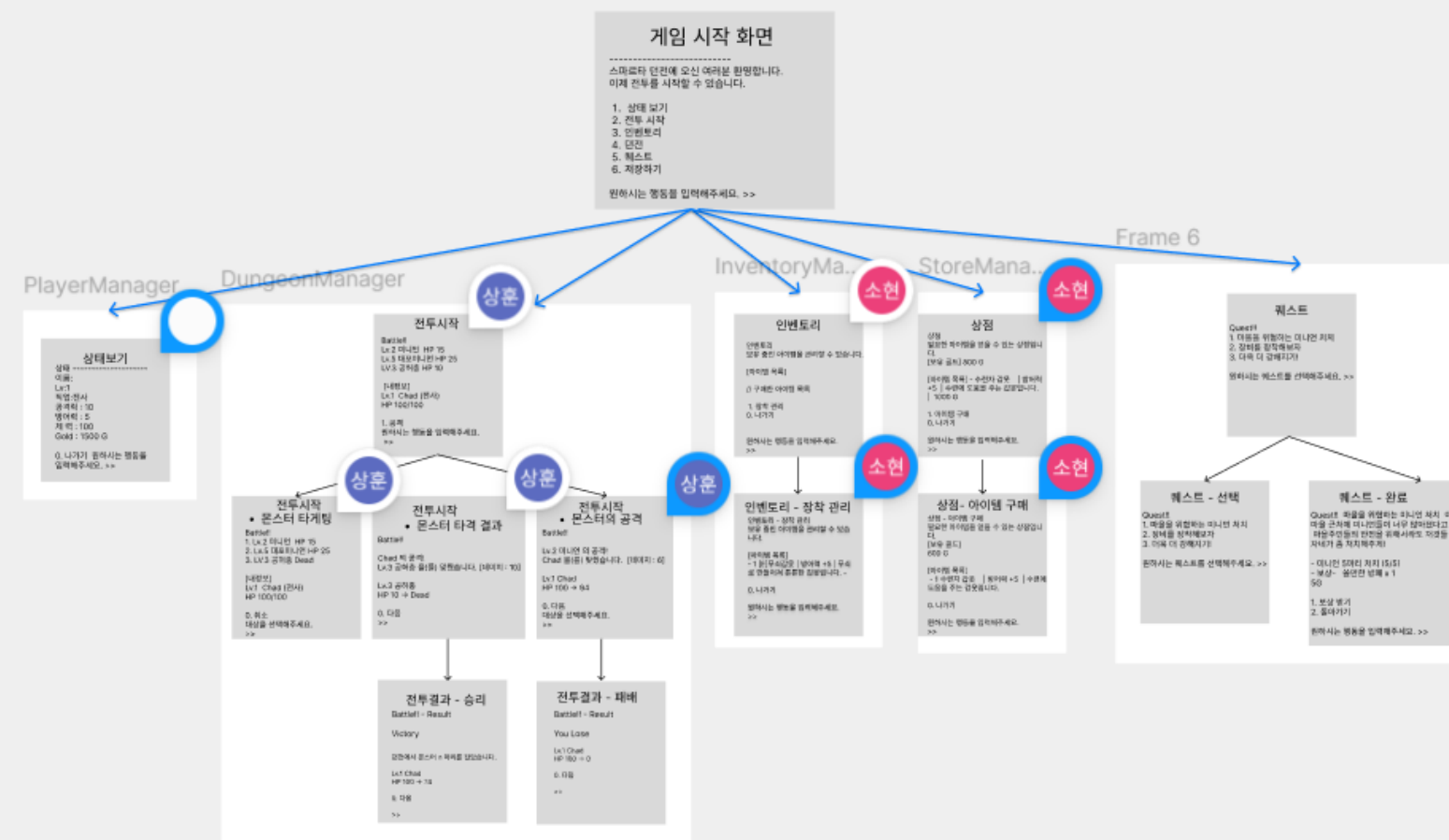
03

와이어프레임

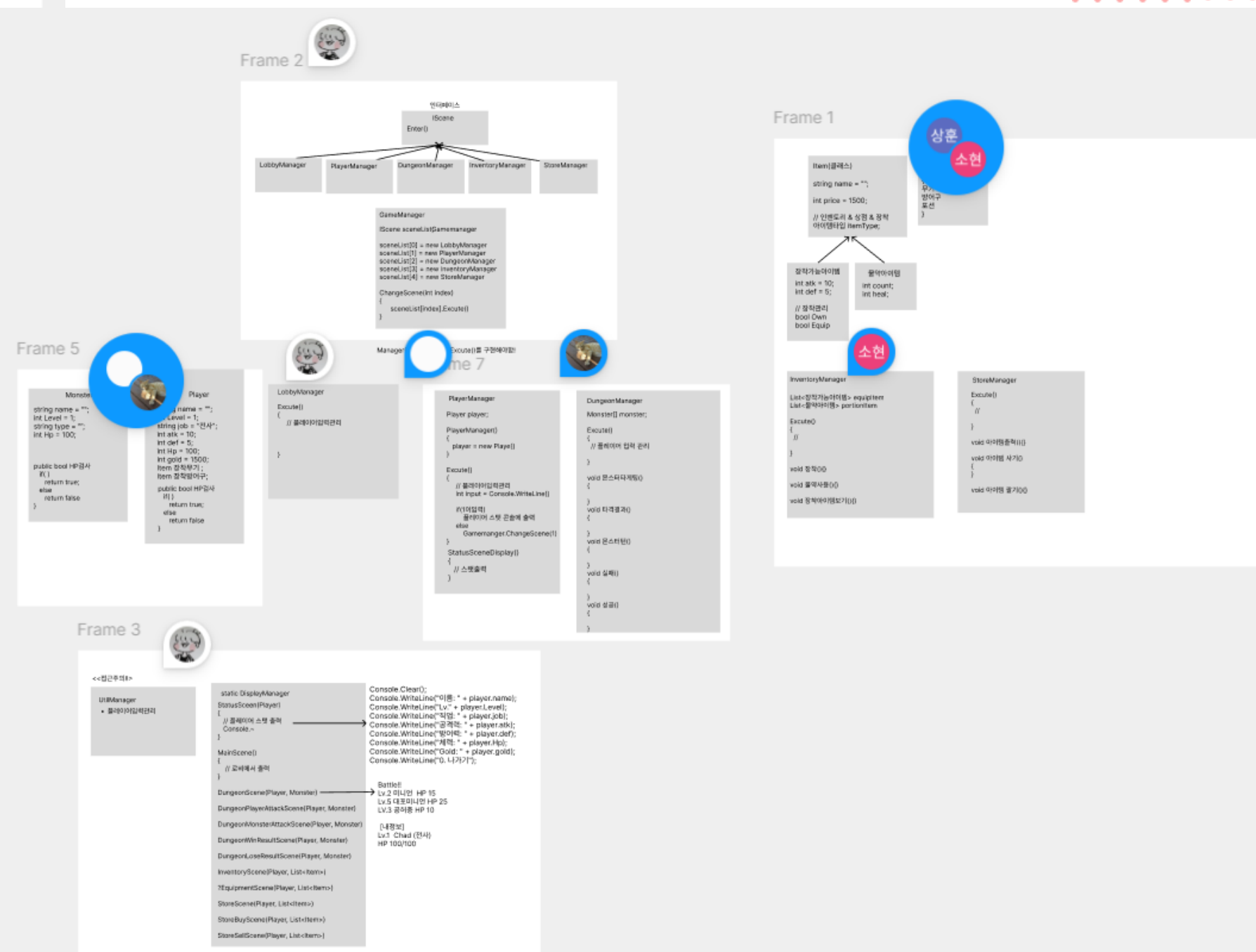
피그마를 사용하여 와이어프레임을 제작하였습니다.



<와이어프레임>



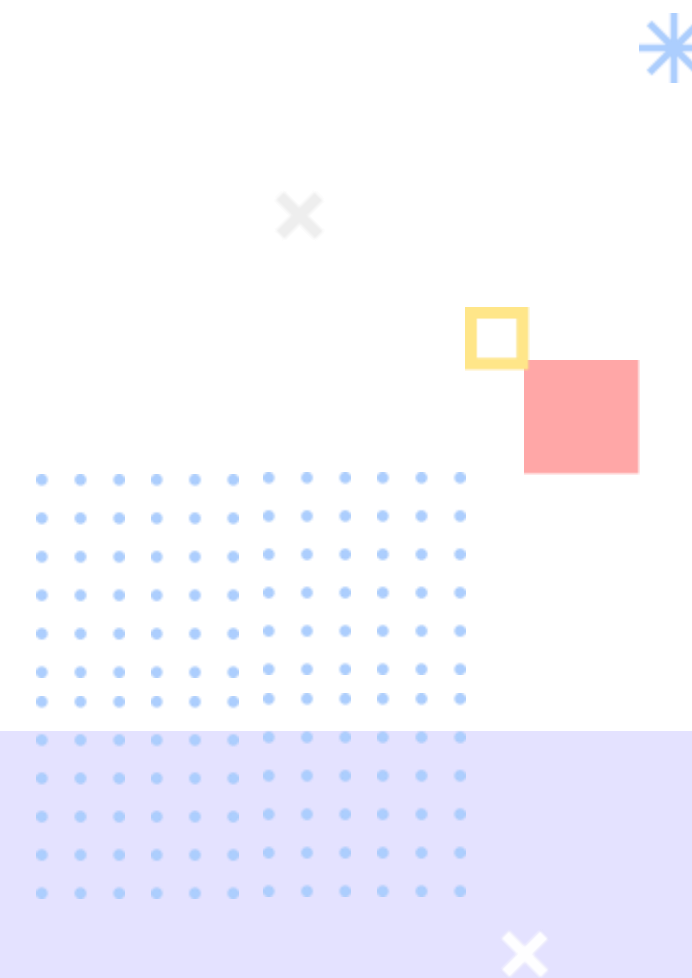
<클래스 다이어그램>



04

메인로직

코드가 전체적으로 흘러가는 구조입니다.

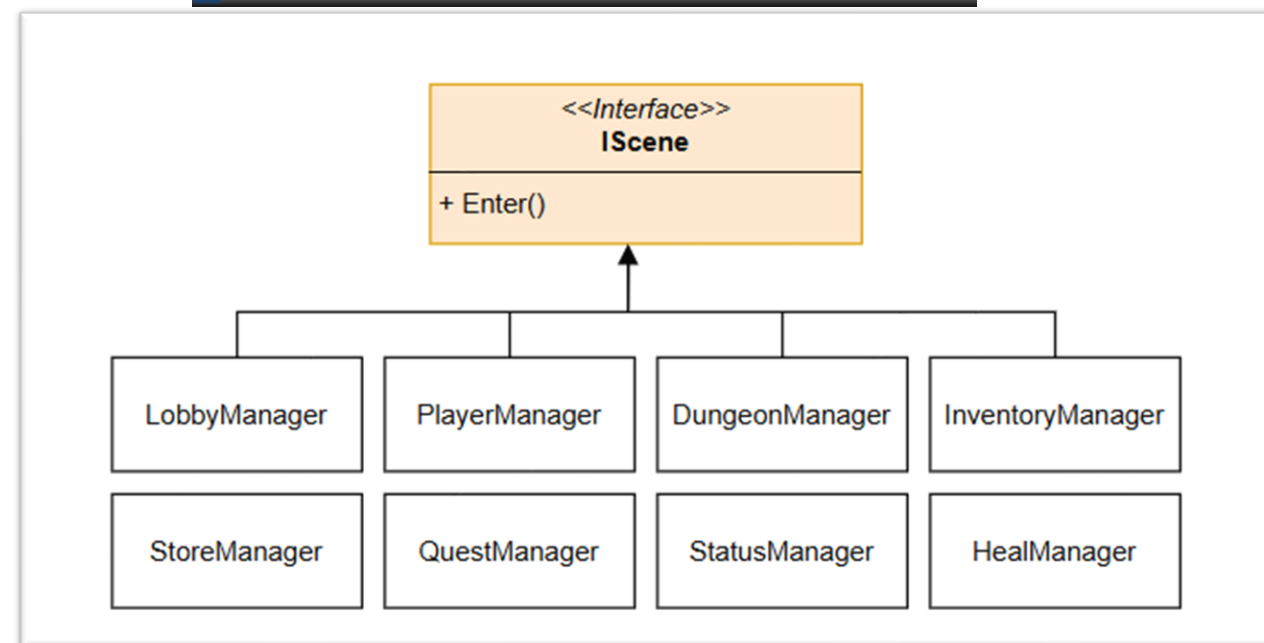


인터페이스를 사용한 클래스 관리

01

참조 11개

```
internal interface IScene
{
    참조 10개
    public void Enter();
}
```



Isceine 인터페이스

Enter()을 필수적으로 구현해야함

02

Isceine 타입의 배열 선언

```
// Iscene 리스트
private IScene[] ISceneList;
private IScene currScene;

// '영양사'
참조 1개
public GameManager()
{
    // Scene 리스트 초기화
    ISceneList = new IScene[Enum.GetNames(typeof(SceneState)).Length];

    ISceneList[(int)SceneState.LobbyManager] = LobbyManager.Instance;
    ISceneList[(int)SceneState.PlayerManager] = PlayerManager.Instance;
    ISceneList[(int)SceneState.DungeonManager] = DungeonManager.Instance;
    ISceneList[(int)SceneState.InventoryManager] = InventoryManager.Instance;
    ISceneList[(int)SceneState.StoreManager] = StoreManager.Instance;
    ISceneList[(int)SceneState.QuestManager] = QuestManager.Instance;
    ISceneList[(int)SceneState.StatusManager] = StatusManager.Instance;
    ISceneList[(int)SceneState.HealManager] = HealManager.Instance;
}
```

GameManger에서 Iscene타입의 배열을 선언 후 Iscene을 구현하는 클래스를 인스턴스화

03

클래스 전환

```
// 씬 (manager) 변화
참조 19개
public void ChangeScene(SceneState _state)
{
    currScene = ISceneList[(int)_state];

    if (currScene != null)
    {
        // enter 실행
        currScene.Enter();
    }
}
```

SceneType을 매개변수로 받는 GameManager의 메서드 ChangeScene()을 실행

공통으로 사용하는 static 클래스

01 DisplayManager : 화면을 출력할 때 사용됨

```
internal static class DisplayManager
{
    // 모든 Scene에서 화면을 출력할 때 사용되는 메니저

    참조 1개
    public static void StatusScene(Player player) 화면출력
    {
        Clear();
        ColorText("[상태보기]", 255, 165, 0);
        AddBlankLine(2);

        Console.WriteLine("이름: " + player.name);
        Console.WriteLine("Lv." + player.level);
        Console.WriteLine("직업: " + player.job);
        Console.WriteLine("공격력: " + player.atk);
        Console.WriteLine("방어력: " + player.def);
        Console.WriteLine("체력: " + player.hp);
        Console.WriteLine("Gold: " + player.gold);

        AddBlankLine(2);
        Console.WriteLine("0. 나가기");

        InputInduction();
    }

    참조 20개
    public static void ColorText(string message, int r = 0, int g = 0, int b = 0, bool lineChange = true)
    {
        // ANSI TrueColor
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        Console.Write($" \u001b[38;2;{r};{g};{b}m");
        Console.Write(message);
        Console.Write($" \u001b[0m");
        if (lineChange)
            Console.WriteLine("");
    }
}
```

색상코드 변경

02 색상코드를 enum과 구조체로 으로 관리

```
public enum TEXTCOLOR
{
    BLACK = ConsoleColor.Black,
    BLUE = ConsoleColor.Blue,
    CYAN = ConsoleColor.Cyan,
    DBLUE = ConsoleColor.DarkBlue,
    DCYAN = ConsoleColor.DarkCyan,
    DGRAY = ConsoleColor.DarkGray,
    DGREEN = ConsoleColor.DarkGreen,
    DMAGENTA = ConsoleColor.DarkMagenta,
    DRED = ConsoleColor.DarkRed,
    DYELLOW = ConsoleColor.DarkYellow,
    GRAY = ConsoleColor.Gray,
    GREEN = ConsoleColor.Green,
    MAGENTA = ConsoleColor.Magenta,
    RED = ConsoleColor.Red,
    WHITE = ConsoleColor.White,
    YELLOW = ConsoleColor.Yellow,
}
```

```
참조 5개
public struct COLOR
{
    public int r;
    public int g;
    public int b;

    참조 2개
    public COLOR(int r = 0, int g = 0, int b = 0)
    {
        this.r = r;
        this.g = g;
        this.b = b;
    }
}
```

```
// 지정 색상 사용
참조 0개
public static void ColorText(string message, TEXTCOLOR color, bool lineChange = true)
{
    Console.ForegroundColor = (ConsoleColor)color;
    Console.Write(message);
    Console.ResetColor();
    if (lineChange)
        Console.WriteLine("");
}

// 추가 색상 사용
참조 3개
public static void ColorText(string message, COLOR color, bool lineChange = true)
{
    ColorText(message, color.r, color.g, color.b, lineChange);
}
```

03

Enum과 구조체 값을 받아
텍스트 색상 변경

공통으로 사용하는 static 클래스

UtilManager

01

```
internal static class UtilManager
{
    // 전반적으로 공통 되는 부분을 작성하는 static 클래스

    // 플레이어 input
    참조 22개
    public static int PlayerInput(int min, int max)
    {
        // 매개변수 :
        // min, max 포함되게

        int input;
        while (true)
        {
            Console.WriteLine($"{min} ~ {max} 숫자를 입력하세요 >>> ");
            if (int.TryParse(Console.ReadLine(), out input)
                && input >= min
                && input <= max)
            {
                // 성공적으로 숫자를 입력받으면 종료
                // 범위내에 입력하면 종료

                break;
            }
            Console.WriteLine("올바른 숫자를 입력해주세요.");
        }

        return input;
    }

    //소수점 올림
    참조 12개
    public static int GetCeiling(double number)
    {
        return (int)Math.Ceiling(number);
    }
}
```

공통으로 사용되는
메서드 관리

: 플레이어 입력처리 메서드

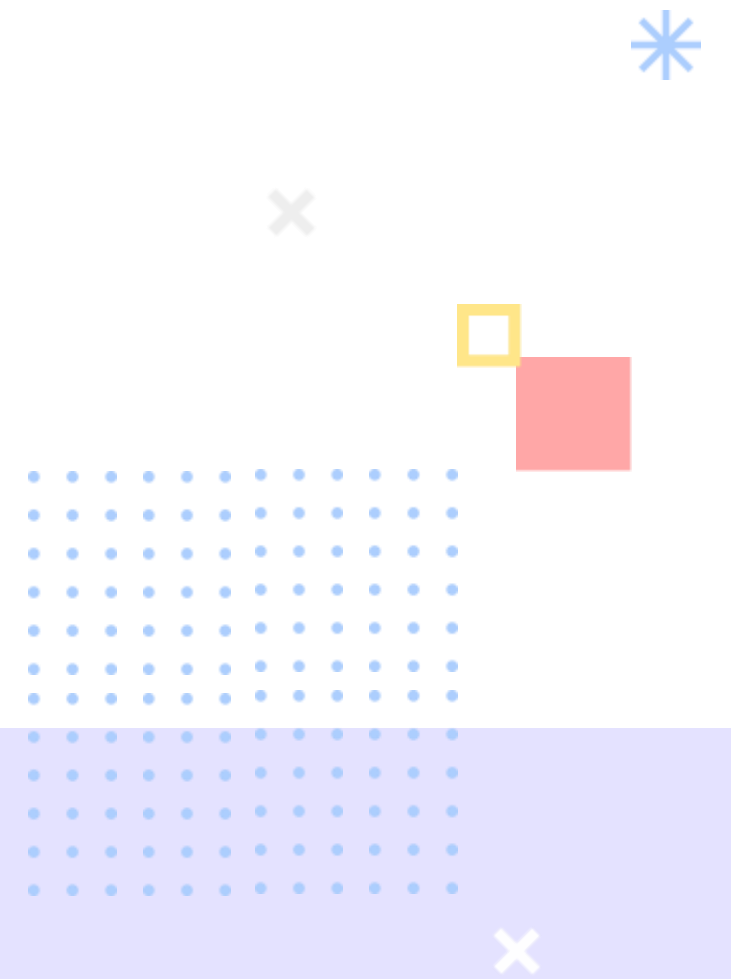
: N초 대기 메서드

: 소수점 올림 메서드

05

핵심기능

클래스 별 핵심기능



플레이어

01 Player 클래스 관리

```

public class PlayerManager : IScene
{
    참조 1개
    private PlayerManager()
    {
        player = new Player();
    }
    private static PlayerManager? instance;
    참조 16개
    public static PlayerManager Instance
    {
        get
        {
            if (instance == null)
            {
                instance = new PlayerManager();
            }
            return instance;
        }
    }
    Player player;

```

```

public class Player : IMove
{
    참조 15개
    public string name { get; set; }
    참조 15개
    public int level { get; set; }
    참조 8개
    public string job { get; set; }
    참조 15개
    public float atk { get; set; }
    참조 6개
    public float def { get; set; }

    참조 13개
    public int maxhp { get; set; }
    참조 25개
    public int hp { get; set; }

```

02 직업선택

```

public void Enter()
{
    DisplayManager.ChooseNameScene(player);
    string input = Console.ReadLine().ToString();
    DisplayManager.ChooseJobScene(player);

    //직업정하기

    int result = UtilManager.PlayerInput(1, 4);
    switch (result)
    {
        case 1:
            player = new Player("전사", input, 130, 10);
            break;

        case 2:
            player = new Player("마법사", input, 90, 17);
            break;

        case 3:
            player = new Player("도적", input, 100, 15);
            break;

        case 4:
            player = new Player("궁수", input, 80, 20);
            break;
        default:
            break;
    }
    Console.WriteLine(player.name);
    Console.WriteLine(player.job);

    GameManager.Instance.ChangeScene(SceneState.LobbyManager);
}

```

퀘스트

01

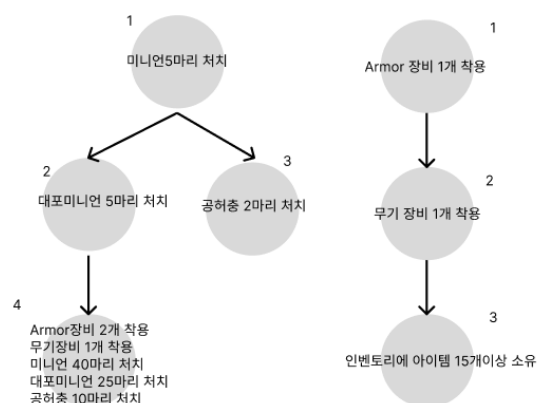
연계 퀘스트 관리

```
// 참조 25개
public abstract class Quest
{
    // 필드
    protected string questName;
    protected string questStory;
    protected int rewardGold;
    protected string questPerform;
    protected QuestState questState;

    // 트리구조
    protected Quest parentQuest;
    protected List<Quest> childQuest;

    // child 자식 대입
    참조 5개
    public void AddChild(Quest child)
    {
        child.parentQuest = this;
        childQuest.Add(child);
    }
}
```

연계퀘스트



연계 퀘스트 저장할
컨테이너 생성

02

```
// 트리 연결하기
// 연계퀘스트 내역은 눈 피그마 확인해주세요!
try
{
    killQuest1.AddChild(killQuest2);
    killQuest1.AddChild(killQuest3);
    killQuest2.AddChild(killQuest4);

    equipQuest1.AddChild(equipQuest2);
    equipQuest2.AddChild(equipQuest3);
}
catch (Exception ex) { Console.WriteLine(ex); }
```

QuestManager.cs
: 연계 퀘스트 저장

퀘스트 완료 시

03

완료한 퀘스트의
연계 퀘스트 가져오기
수행가능한 퀘스트
목록에 넣기

```
private List<Quest> performableQuests; // 수행가능 퀘스트
private List<Quest> doneQuest; // 아예 끝난 퀘스트

참조 1개
private void RemovePerformListAndAddToChild()
{
    // 현재 퀘스트의 state를 done으로
    // 퀘스트 수락 -> 현재 퀘스트를 accept 로 바꾸기
    currQuest.ChangeState(QuestState.done);

    // currQuest와 같은 quest 반환
    var temp = performableQuests.Find( quest => quest.Equals(currQuest));

    // 수행가능 리스트에서 삭제
    if (temp != null)
    {
        performableQuests.Remove(temp);
    }

    // done 리스트에 추가
    doneQuest.Add(currQuest);

    // 현재 퀘스트의 child리스트에 접근해서 가능한 퀘스트리스트에 넣어야 함
    for (int i = 0; i < currQuest.ChildQuest.Count; i++)
    {
        performableQuests.Add(currQuest.ChildQuest[i]);
    }
}
```


던전

01

참조 2개

```
public void Enter()
{
    Player enterPlayer = PlayerManager.Instance._Player;
    int playerHpBeforeEnter = enterPlayer.hp;
    int playerMpBeforeEnter = enterPlayer.mp;
    SetMonsters(enterPlayer);
    while (!enterPlayer.Hpcheck() && GetMonsterDieCount() != monsters.Count)
    {
        EnterDungeon(enterPlayer);
    }
    if (enterPlayer.Hpcheck()) //플레이어의 체력이 0일때
    {
        Lose(enterPlayer, playerHpBeforeEnter, playerMpBeforeEnter);
    }
    else if (GetMonsterDieCount() == monsters.Count) //몬스터를 전부 처리할 때
    {
        Victory(enterPlayer, playerHpBeforeEnter, playerMpBeforeEnter);
    }
    switch (UtilManager.PlayerInput(0,0))
    {
        case 0:
            ClearMonsters();
            GameManager.Instance.ChangeScene(SceneState.LobbyManager);
            break;
    }
}
```

: While() 문을 사용해서 Player-Monster 턴 제 구현

플레이어 체력이 0 일 때

L o s e () 실행

잡은 몬스터 수와 던전의 몬스터
수가 같으면

V i c t o r y () 실행

스킬

01

Func<Player, Monster, int>를 가지는 Skill 클래스

```

참조 24개
public class Skill
{
    참조 12개
    public string name { get; set; }
    참조 4개
    public string description { get; set; }
    참조 9개
    public int mp { get; set; }
    참조 8개
    public int type { get; set; }
    참조 4개
    Func<Player, Monster, int> action { get; set; }

    // action 포함 생성자
    참조 8개
    public Skill(string name, string description, int mp, int type, Func<Player, Monster, int> action)
    {
        this.name = name;
        this.description = description;
        this.mp = mp;
        this.type = type;
        this.action = action;
    }
}

```

직업별로 스킬 저장

```

public void SetSkills(string job)
{
    skills = new List<Skill>();

    switch (job)
    {
        case "전사":
            skills.Add(new Skill("알파 스트라이크", "공격력 * 3 로 하나의 적을 공격합니다.", 10, (int)DungeonManager.SkillType.one, SkillManager.AlphaStrike));
            skills.Add(new Skill("더블 스트라이크", "공격력 * 1.5 로 2명의 적을 랜덤으로 공격합니다.", 15, (int)DungeonManager.SkillType.random2, SkillManager.DoubleStrike));
            break;
        case "마법사":
            skills.Add(new Skill("파이어볼", "공격력 * 2 로 하나의 적을 공격합니다.", 15, (int)DungeonManager.SkillType.one, SkillManager.FireBall));
            skills.Add(new Skill("메테오", "공격력 * 3 로 모든 적을 공격합니다.", 30, (int)DungeonManager.SkillType.all, SkillManager.Meteor));
            break;
        case "도적":
            skills.Add(new Skill("라이프 스틸", "공격력만큼 하나의 적의 체력을 훔칩니다.", 15, (int)DungeonManager.SkillType.one, SkillManager.LifeSteal));
            skills.Add(new Skill("골드 어택", "공격력 + (현재골드 / 100)만큼 하나의 적을 공격합니다.", 15, (int)DungeonManager.SkillType.one, SkillManager.GoldAttack));
            break;
        case "궁수":
            skills.Add(new Skill("파워샷", "공격력 * 2 로 하나의 적을 공격합니다.", 15, (int)DungeonManager.SkillType.one, SkillManager.PowerShot));
            skills.Add(new Skill("멀티샷", "(공격력 * 1.5) - 대상 수만큼 모든 적을 공격합니다.", 20, (int)DungeonManager.SkillType.all, SkillManager.MultiShot));
            break;
    }
}

```

구체적인 스킬은 SkillManager에서 구현!

```

참조 8개
public static class SkillManager
{
    참조 1개
    public static int AlphaStrike(Player player, Monster monster)
    {
        int damage = UtilManager.GetCeiling(player.atk) * 2;
        monster.hp = UtilManager.CalcDamage(monster.hp, damage);
        return damage;
    }

    참조 1개
    public static int DoubleStrike(Player player, Monster monster)
    {
        int damage = UtilManager.GetCeiling(player.atk * 1.5);
        monster.hp = UtilManager.CalcDamage(monster.hp, damage);
        return damage;
    }

    참조 1개
    public static int FireBall(Player player, Monster monster)
    {
        int damage = UtilManager.GetCeiling(player.atk) * 2;
        monster.hp = UtilManager.CalcDamage(monster.hp, damage);
        return damage;
    }
}

```

SkillManager의 메서드를 매개변수로!

인벤토리

Item 클래스

장착 관리 메서드

02

01

```
참조 32개
public enum ITEMTYPE
{
    WEAPON,
    ARMOR,
    POTION,
}
참조 8개
public abstract class Item
{
    protected string? name;
    protected string? description;
    protected int price;
    protected ITEMTYPE type;

    참조 29개
    public string Name{get { return name; }set { name = value; }}
    참조 26개
    public string Description{get { return description; }set { description = value; }}
    참조 25개
    public int Price{get { return price; }set { price = value; }}
    참조 27개
    public ITEMTYPE Type{get { return type; }set { type = value; }}
}

// 장착 가능한 아이템
참조 67개
public class MountableItem : Item
{
    private int attack;
    private int defense;
    private bool own;
    private bool equip;

    참조 30개
    public int Attack{get { return attack; }set { attack = value; }}
    참조 26개
    public int Defense{get { return defense; }set { defense = value; }}
    참조 29개
    public bool Own{get { return own; }set { own = value; }}
    참조 30개
    public bool Equip{get { return equip; }set { equip = value; }}
}
```

ITEM이 WEAPON인지,
ARMOR인지,
POTION인지 나타내는 타입

소유 여부에 대한 bool 변수
장착 여부에 대한 bool 변수

```
// 장착 (타입에 따라 중복 장착이 안 되도록)
참조 17개
public void Equip(int input)
{
    MountableItem select = ownItems[input - 1];
    MountableItem equipped = null;

    if (select.Type == ITEMTYPE.WEAPON)
    {
        for (int i = 0; i < ownWeapons.Count; i++)
        {
            if (ownWeapons[i].Equip)
            {
                equipped = ownWeapons[i];
                break;
            }
        }

        if (equipped != null && equipped == select)
        {
            Unequip(equipped);
        }
        else if (equipped != null && equipped != select)
        {
            Unequip(equipped);
            select.Equip = true;
            player.atk += select.Attack;
        }
        else
        {
            select.Equip = true;
            player.atk += select.Attack;
        }
    }
    else if (select.Type == ITEMTYPE.ARMOR)
    {
        for (int i = 0; i < ownArmors.Count; i++)
        {
            if (ownArmors[i].Equip)
            {
                // < 고른 게 웨폰이라면 장착된 게 있는지 확인 후 장착>
                // weapon만 있는 리스트에서 equip이 true인 것 찾기
            }
        }
    }
}

// 장착 관리 창으로 이동
참조 2개
public void MoveEquipmentScene(List<MountableItem> items)
{
    items = items.Distinct().ToList();
    DisplayManager.EquipmentScene(player, items);
    int input = UtilManager.PlayerInput(0, items.Count);

    if (input == 0)
    {
        MoveInventoryScene();
    }
    else if (input > 0 && input <= items.Count)
    {
        Equip(input);
        MoveEquipmentScene(items);
    }
}
```

03 Item이 중복되는 문제점

Distinct()로 해결!

상점 / 보상

01

퀘스트 수행 / 던전 클리어시에만
획득할 수 있는 특수보상

```
rewardArmor.Add(new MountableItem()
{
    Name = "코튼엑스트라임",
    Description = "코튼 100%",
    Price = 150,
    Type = ITEMTYPE.ARMOR,

    Attack = 0,
    Defense = 30,
    Own = false,
    Equip = false
});
rewardArmor.Add(new MountableItem()
{
    Name = "플러피안후리스폴집 재킷",
    Description = "이불을 덮은 듯한 따뜻함",
    Price = 200,
    Type = ITEMTYPE.ARMOR,

    Attack = 0,
    Defense = 35,
    Own = false,
    Equip = false
});
```

02

보상아이템을 획득하면 인벤토리에서
사라지는 현상

참조 2개

```
T rewardItem<T>(List<T> reward, List<T> result) where T : MountableItem
{
    T item;
    do
    {
        item = reward[ranFunc(0, reward.Count)];
        item.Own = true;

        // 무한 반복 방지
        int count = 0;
        foreach (T item2 in reward)
        {
            if (item2.Own)
            {
                count++;
            }
        }
        if (count == reward.Count)
            break;

        // 아이템 중복 검사. 중복이면 한번 더
        while (result.Contains(item));

        // 보상아이템 인벤토리로 추가
        result.Add(item);

        return item;
    }
}
```

보상의 중복검사를 통해서
중복되지 않는 아이템만
인벤토리에 추가

03

보상아이템 : 판매가능 / 구매불가

참조 1개

```
private void SellItem(int idx)
{
    List<MountableItem> items = InventoryManager.Instance.mountableItems;
    Player player = PlayerManager.Instance._Player;

    MountableItem item = items[idx - 1];
    // 판매 불가
    if (!item.Own)
    {
        Console.WriteLine("소유하지 않은 아이템입니다.");
        UtilManager.DelayForSecond(1);
    }
    // 판매 가능
    else
    {
        // 장착 중이라면
        if (item.Equip)
        {
            // 아이템 장착 해제 및 성능 만큼 캐릭터 성능 하향
            InventoryManager.Instance.Unequip(item);
        }
        player.gold += (int)((float)item.Price * 0.85f);

        // 보상아이템이라면
        if (InventoryManager.Instance.RewardInstnace.rewardItems.Contains(item))
        {
            // 리스트에서 삭제
            items.Remove(item);
        }

        Console.WriteLine("판매를 완료했습니다.");
        UtilManager.DelayForSecond(1);
    }
}
```

아이템 판매 메서드

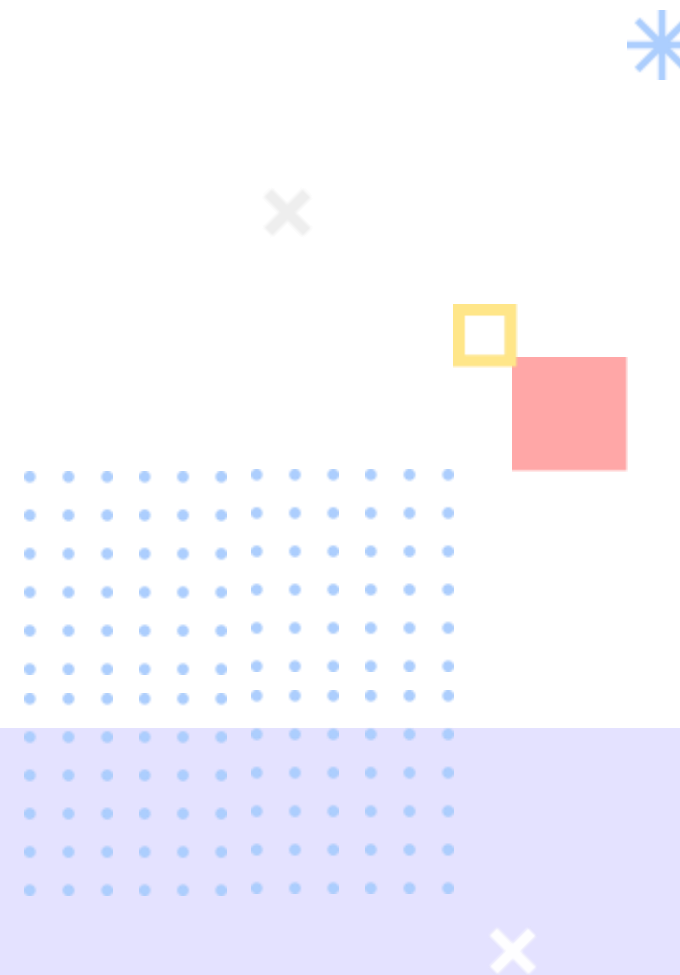
보상아이템이라면?

삭제

06

소감

팀 소감 / 개인 소감



팀 소감

01

깃허브 컨벤션



Commit을 할 때 컨벤션을
잘 지켜서 커밋함

Merge branch 'develope' into SSH_힐매니저
S-SoHyun • 11 hours ago

[Feat] 포션사용 기능 추가
S-SoHyun • 11 hours ago

Merge branch 'KJH_Quest' into develope
김지현 • 11 hours ago

[feat]QuestManger에서 보상받은 후 리스트에 추가삭...
김지현 • 11 hours ago

[feat]Quest-EquiptQuest관련 type에 따라 return하는...
김지현 • 11 hours ago

[feat] 터미널 색상 추가
Lee Sanghun • 11 hours ago

[feat]던전승리시 경험치따른 레벨업과 출력
Naring • 12 hours ago

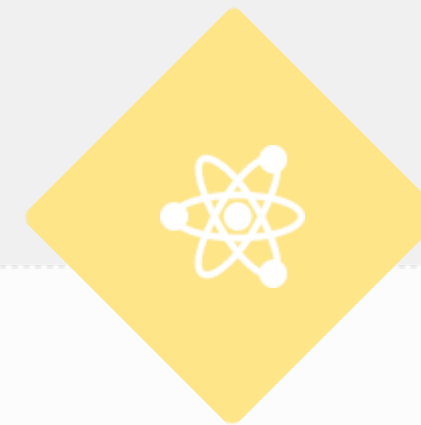
[feat] 스킬 객체 작성
Lee Sanghun • 12 hours ago

[Feat] 스킬 매니저를 활용한 스킬 구현
ProgramCnt • 13 hours ago

[feat]QueseState추가, Quest컨테이너 추가, Quest 추가
김지현 • 14 hours ago

02

깃허브 브랜치



용도에 따라 브랜치 생성 후
작업

Merge branch 'dev...' into develope

Merge branch 'psj_DungeonMana...

[Feat] DisplayManager 던전씬 부...

Merge branch 'develope' into LS

[fix] 꼬인파일 정리... Lee Sanghun

[feat] 저장 기능 구조 작성... Lee

[feat] Monster Attack 반환값 Na

Merge branch 'develope' into KJ

Merge branch 'psj_DungeonM...

[Feat] 던전매니저 몬스터 세팅...

[Feat] 던전 전투시작, 플레이어 ..

Merge branch 'LSH_상점매니저'

[fix] StroeScene 전환 적용... Lee

[perf] 싱글톤 생성 규칙 제작... L

Merge branch 'KJH_Lobby' int...

[add] LobbyManager 생성 / E

개인 후기

팀원 | 김지현

자료구조를 퀘스트 연계 기능 구현에 실제로 적용해볼 수 있어서 즐거운 경험이었습니다. 다들 수고하셨습니다.

이상훈 | 팀원

깃허브를 통해 각자의 노력이 차곡차곡 쌓여가며, 하나의 세계가 완성되어가는 기분 좋은 과정이었습니다.

박성주 | 팀원

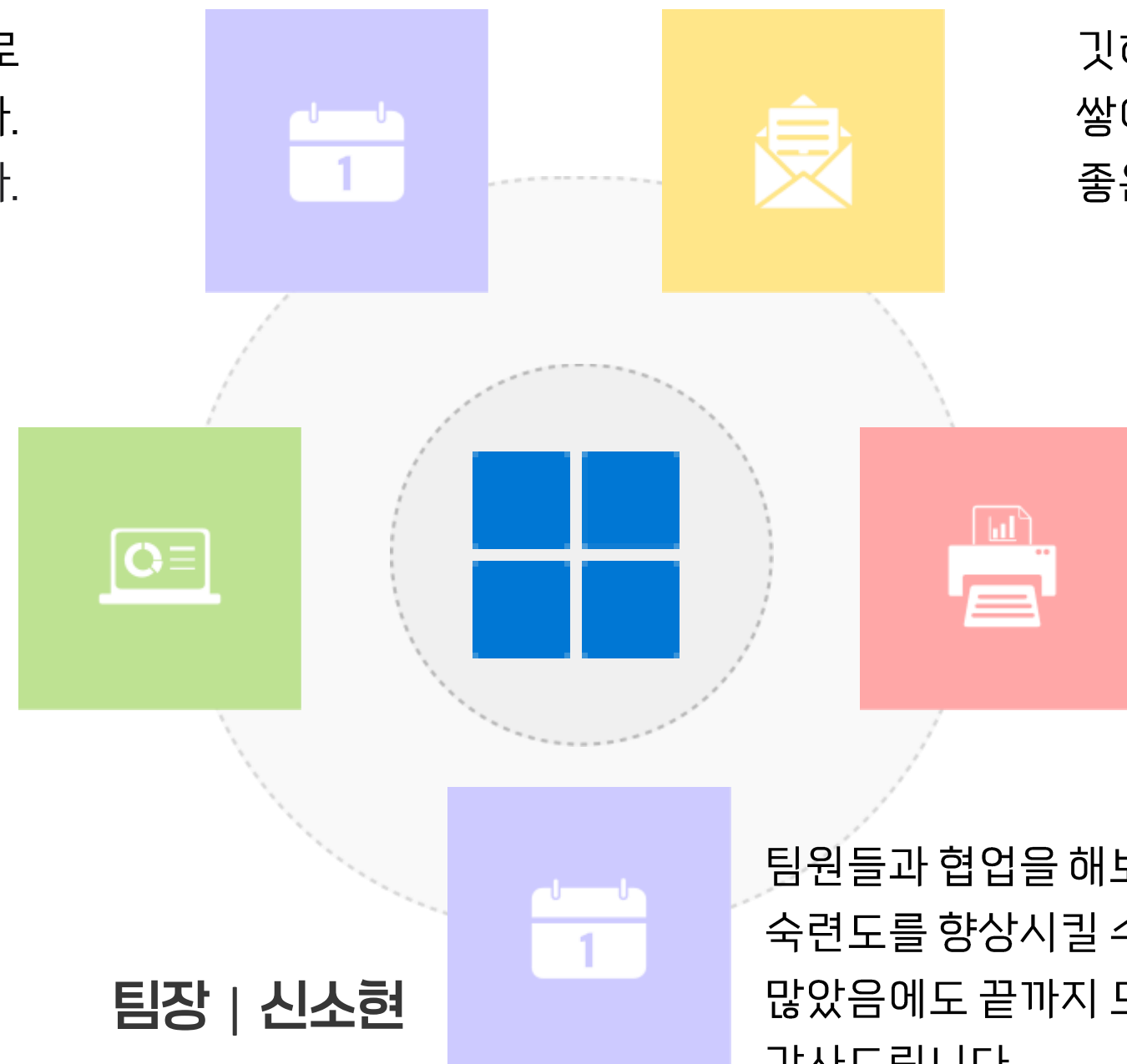
시간이 부족하지 않은 줄 알았으나 생각보다 빠르게 지나가서 해당 작업 전에 작업이 얼마나 걸릴지를 먼저 생각하고 작업하는 것을 좀 더 길러야겠다고 느껴졌습니다.

팀원 | 김지환

팀 프로젝트를 하며 분업의 중요성을 크게 느꼈으며, 규칙을 잘 지키며 코드를 작성하는 습관을 가져야겠다고 느꼈습니다.

팀장 | 신소현

팀원들과 협업을 해보면서 C#과 Github에 대한 숙련도를 향상시킬 수 있었습니다. 미숙한 부분이 많았음에도 끝까지 도와주신 팀원분들께 정말 감사드립니다.





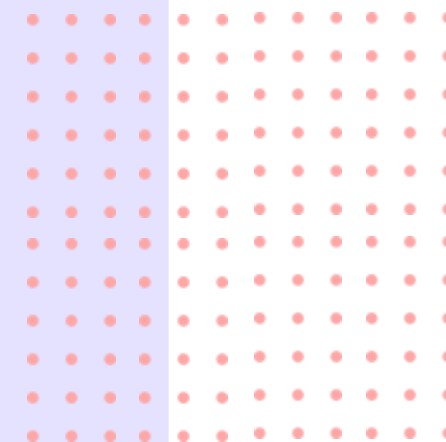
WINDOW 11 조

THANK YOU!

팀장 | 신소현

팀원 | 김지환 이상훈 김지현 박성주

내일배움캠프 Unity 7기



https://github.com/S-SoHyun/Window11_TextRPG