2D Color_RPG

작성자 : 채용운

작성일 : 2019년 12월 08일

목차

2D Color_RPG

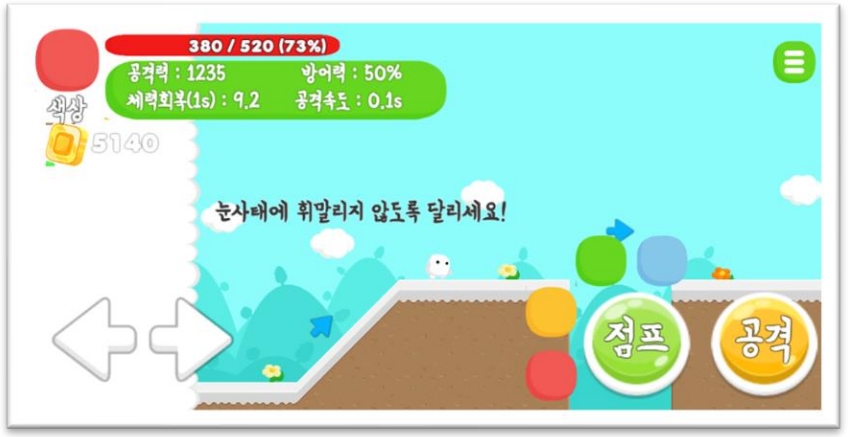
1. 프로젝트 설명

게임 장르	2D 슈팅 RPG
사용언어	C#
플랫폼	PC : Windows / Mobile : Android
게임특징	몬스터와 같은 색상의 공격만 피해를 입힘
깃허브	https://github.com/chaeyongwoon/2D_COLOR_RPG
시연영상	https://youtu.be/k6XDMN8dm6M

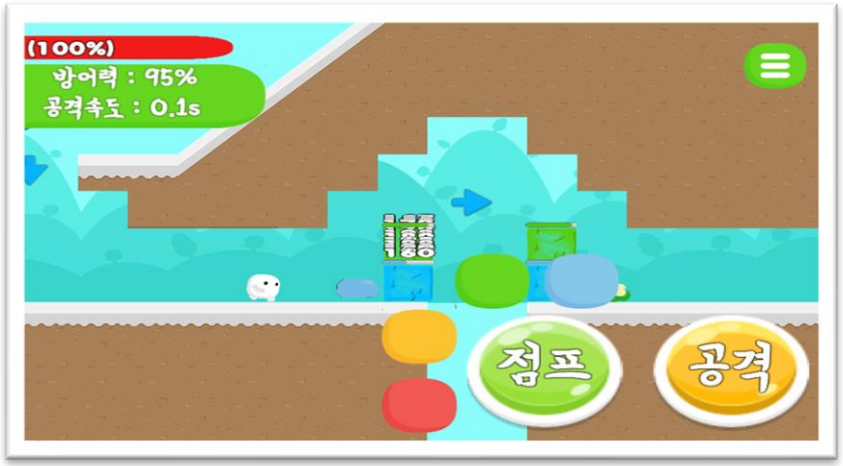
2D Color_RPG

1. 프로젝트 설명

< 캐릭터 능력치 강화 >



< 서로 다른 컨셉의 스테이지 >



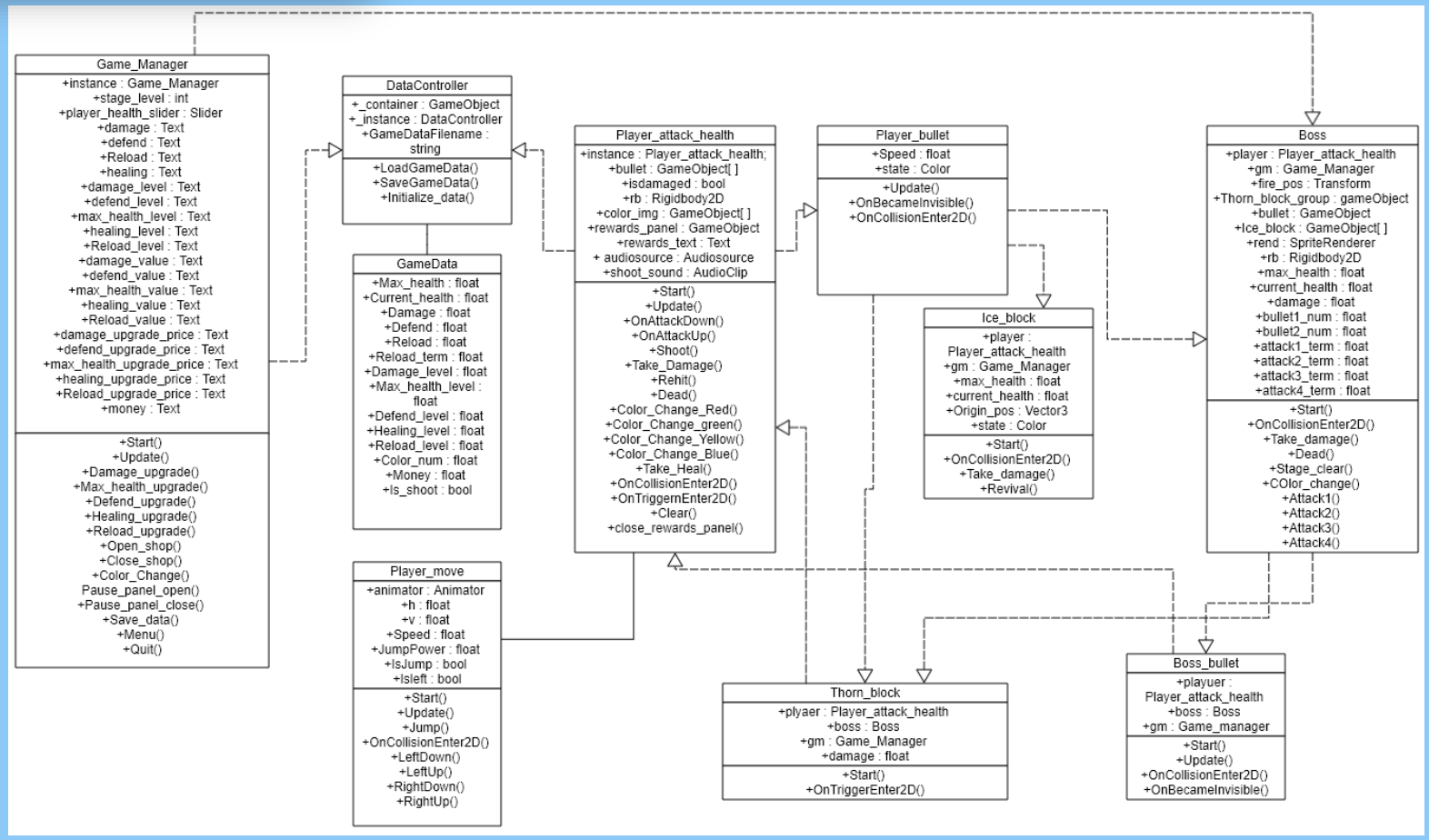
< 신나는 타격감 >

< 난이도 조절 가능한 보스스테이지 >

2D Color_RPG

2. 프로젝트 설계

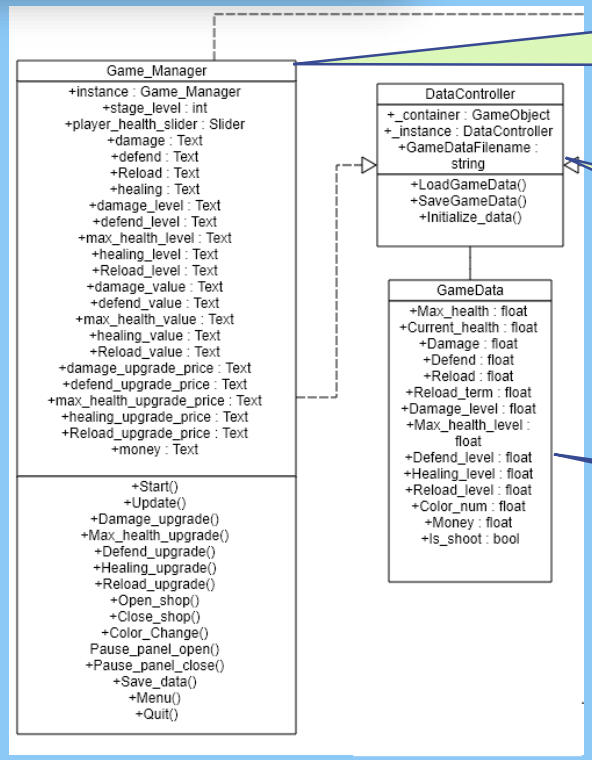
클래스 다이어그램



2D Color_RPG

2. 프로젝트 설계

클래스 다이어그램



Game_Manager
- 게임 일시정지메뉴, 상점,능력치 업그레이드, UI 변수를 관리

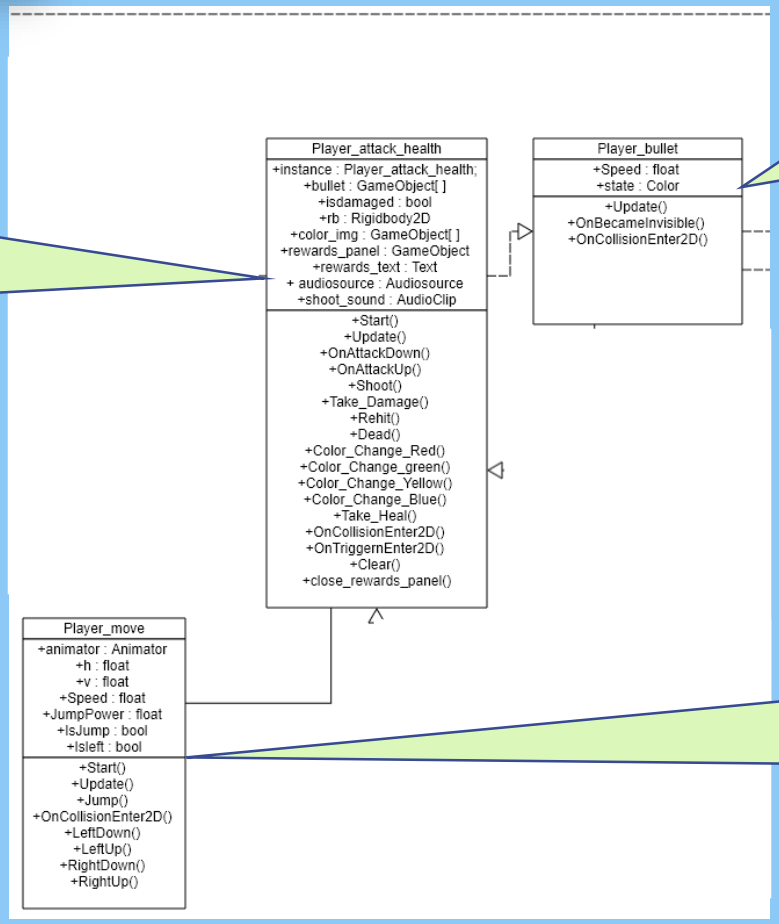
DataController
- 게임 데이터 저장 및 불러오기 담당

GameData
- 플레이어에게 사용되는 데이터들을 보관

2D Color_RPG

2. 프로젝트 설계

클래스 다이어그램



Player_attack_health
- 공격,피격,색상변경 등 직접 플레이어의 캐릭터를 제어한다

Player_bullet
- 플레이어의 공격으로 생성되며 적에게 피해를 준다

Player_move
- 플레이어의 이동과 점프를 담당하며 PC,모바일 두 플랫폼에서 모두 작동한다

2D Color_RPG

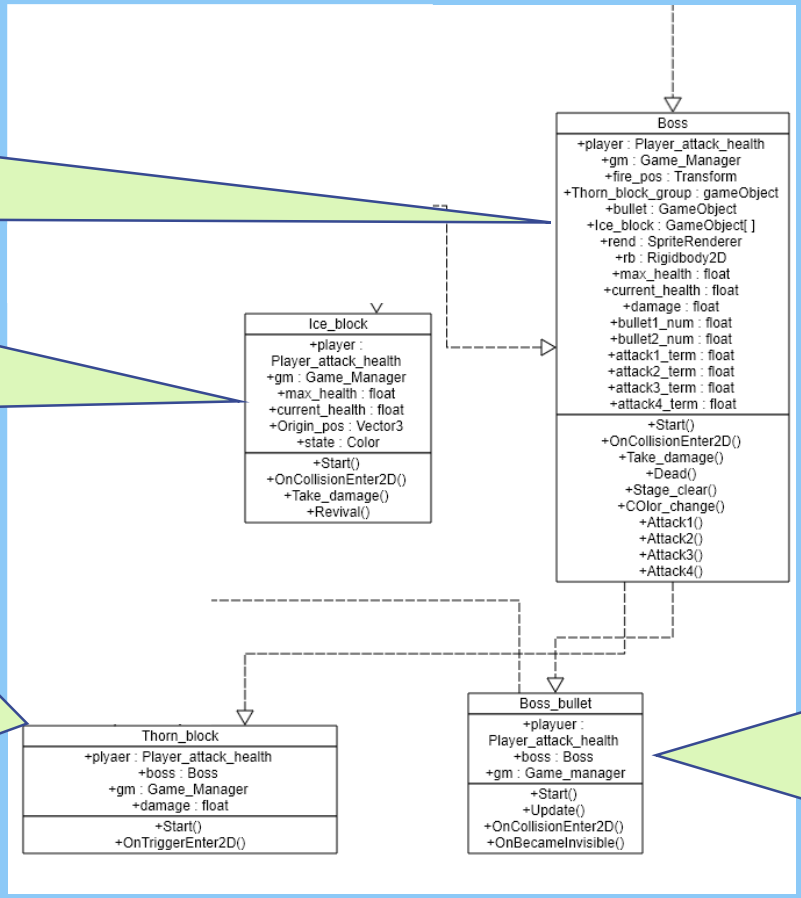
2. 프로젝트 설계

클래스 다이어그램

Boss
- 무한난이도 스테이지에서 등장하는 보스몬스터의 상태를 담당. Game_Manager로 부터 난이도를 값을 받아와 체력과 공격력을 산정한다.

Ice_block
- 플레이어의 이동과 공격을 막는 방해물. 보스몬스터와 마찬가지로 난이도가 높아질수록 체력이 증가한다

Thorn_block
- 보스몬스터의 공격력을 받아와 플레이어와 충돌시 피해를 준다.

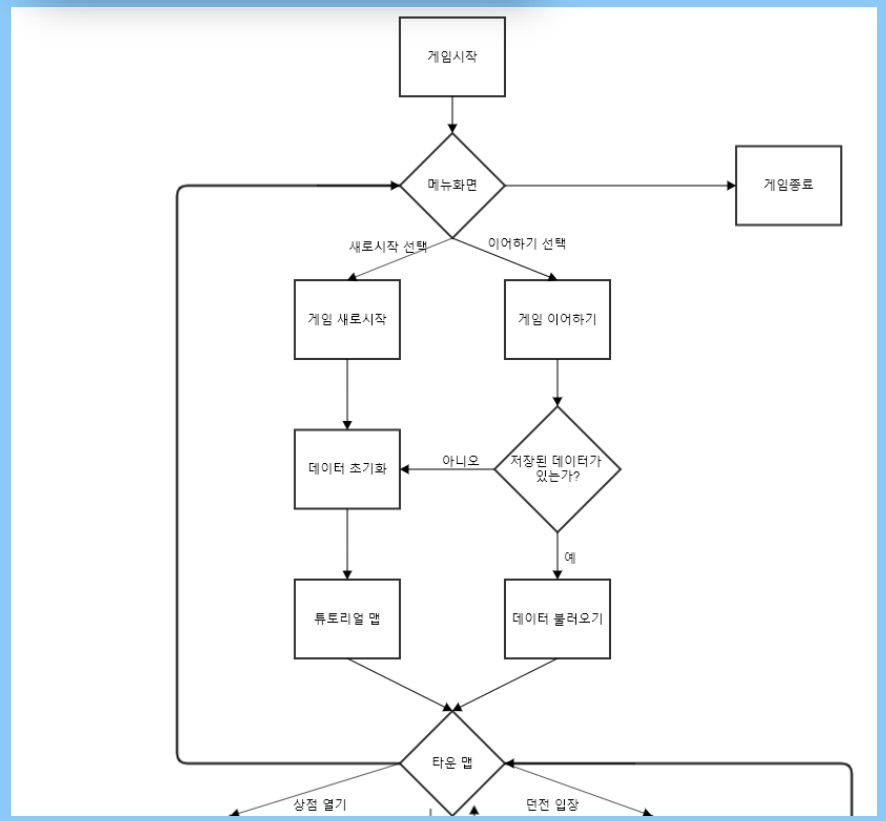


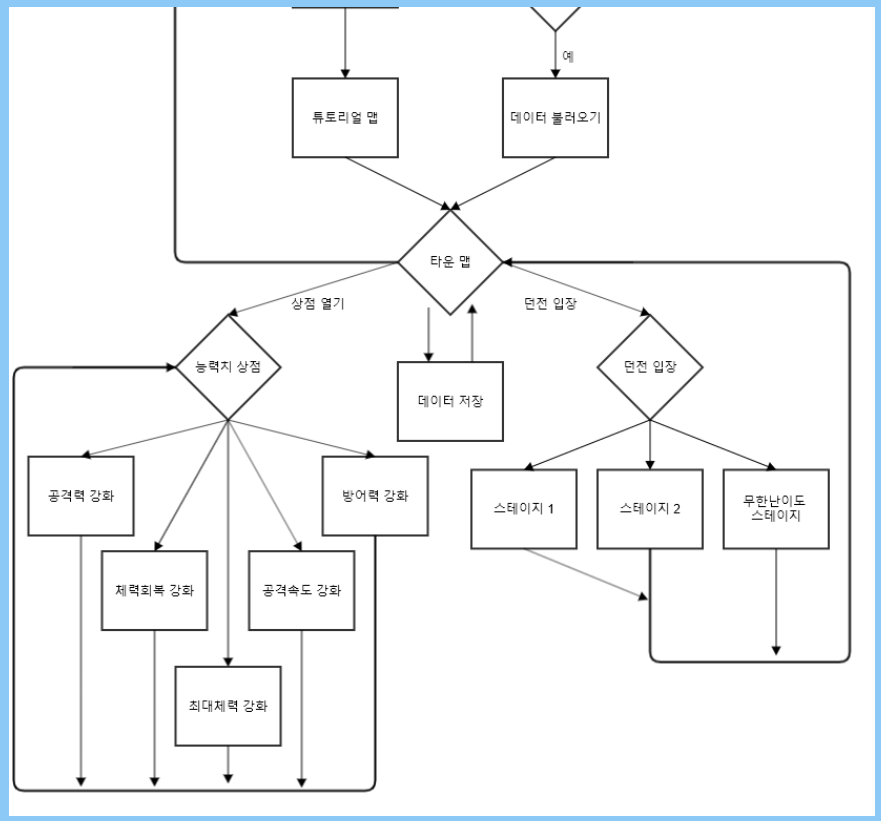
Boss_bullet- 보스몬스터의 공격력을 받아와 플레이어와 충돌시 피해를 준다.

2D Color_RPG

2. 프로젝트 설계

게임 흐름도





2D Color_RPG

3. 주요구현부

개발내용

보스몬스터 Coroutine 사용

Update()를 사용하지 않고 Coroutine을 사용하여 공격 패턴 구현

```
// 4가지의 공격패턴을 각각 설정된 시간마다 사용
StartCoroutine(Attack1());
StartCoroutine(Attack2());
StartCoroutine(Attack3());
StartCoroutine(Attack4());
```

```
public IEnumerator Attack1() // 1번 공격패턴 함수. 총알 부채꼴 발사
{
    while (true)
    {
        yield return new WaitForSeconds(attack1_term);

        Quaternion angle;
        for (int i = 1; i < bullet1_num; i++)
        {
            angle = Quaternion.Euler(new Vector3(0, 0, -90 + 180 / bullet1_num * i));   // 부채꼴 모양으로 각도 계산
            Instantiate(bullet, fire_pos.position, angle);
        }
    }
}
참조 1개
public IEnumerator Attack2()// 2번 공격패턴 함수. 가시블록 생성
{
    while (true)
    {
        yield return new WaitForSeconds(attack2_term);

        Thorn_block_group.GetComponent<Rigidbody2D>().constraints
            = RigidbodyConstraints2D.FreezePositionY;
        Thorn_block_group.transform.position = new Vector3(-10, 30, 0);
        Thorn_block_group.GetComponent<Rigidbody2D>().constraints
            = RigidbodyConstraints2D.None;


    }
}
참조 1개
public IEnumerator Attack3()// 3번 공격패턴 함수. 총알 나선형 발사
{
    while (true)
    {
        yield return new WaitForSeconds(attack3_term);

        int i = 1;
        Quaternion angle;
        while (i < bullet2_num)
        {
            angle = Quaternion.Euler(new Vector3(0, 0, -90 + 180 / bullet2_num * i));  // 부채꼴 모양으로 각도 계산
            Instantiate(bullet, fire_pos.position, angle);
            i++;
            yield return new WaitForSeconds(0.1f); // 0.1초의 딜레이를 주어 나선형으로 발사
        }
    }
}
```

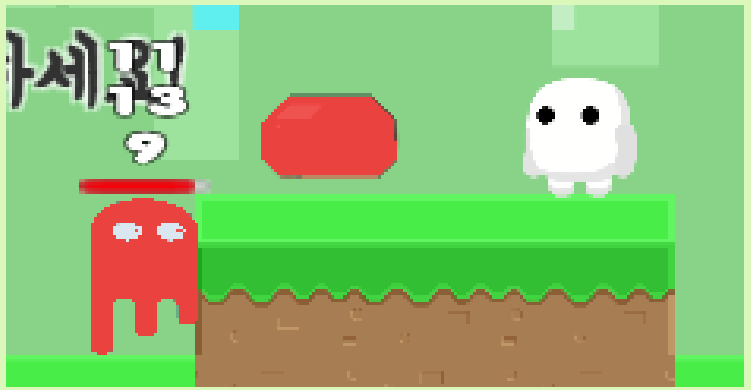
2D Color_RPG

3. 주요구현부

개발내용

같은 색상만 공격가능

Color 변수 State를 ToString()으로 비교하여 몬스터와 총알의 색상이 서로 같을 경우에 피해를 입힘



```
public enum Color
{
    red,
    yellow,
    green,
    blue
}

public Color state;
```

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.transform.CompareTag("bullet"))
    {
        if (state.ToString() == collision.gameObject.GetComponent<player_bullet>().state.ToString())
        {
            Take_damage(DataController.instance.gameData.Damage);
        }
    }
```

2D Color_RPG

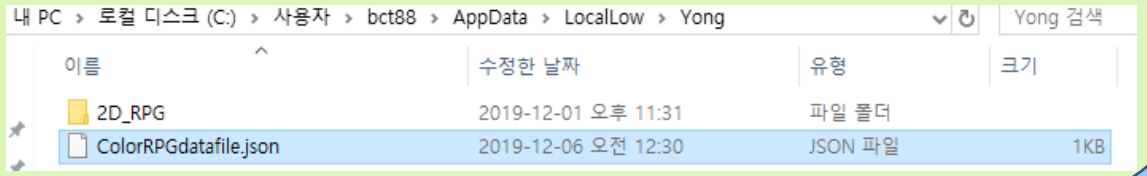
3. 주요구현부

게임 데이터 관리

Json을 이용한 데이터관리

Json유틸리티를 이용해 게임 데이터를 Json 파일로 저장하고 필요할 때 로드하여 사용

```
public void LoadGameData() // 게임데이터 로드
{
    string filePath = Application.persistentDataPath + GameDataFileName;
    if (File.Exists(filePath))
    {
        Debug.Log("불러오기 성공!");
        Debug.Log(filePath);
        string FromJsonData = File.ReadAllText(filePath);           // 거
        _gameData = JsonUtility.FromJson<GameData>(FromJsonData);   // J
    }
    else
    {
        Debug.Log("새로운 파일 생성");
        _gameData = new GameData();
        Initialize_Data();
    }
}
참조 2개
public void SaveGameData() // 게임데이터 저장
{
    string ToJsonData = JsonUtility.ToJson(gameData);
    string filePath = Application.persistentDataPath + GameDataFileName;
    File.WriteAllText(filePath, ToJsonData);
    Debug.Log("저장 완료");
}
```



2D Color_RPG

3. 주요구현부

최적화 작업

1. String.Format

String은 +연산자를 사용할 때마다 새 메모리를 할당하여 string생성비용 & 가비지를 생성하기 때문에 string.format을 사용

```
player_health_slider.value = (DataController.instance.gameData.Current_heal
damage.text = string.Format("공격력 : {0}", DataController.instance.gameDat
health.text = string.Format(" {0} / {1}", DataController.instance.gameData.
defend.text = string.Format("방어력 : {0}%", DataController.instance.gameDa
healing.text = string.Format("체력회복(1s) : {0}",DataController.instance.g
Reload.text = string.Format("공격속도 : {0}s", DataController.instance.game
```

2. GC.Collet()

각각의 스테이지 씬에서 타운 씬으로 전환시 가비지컬렉터 실행

```
System.GC.Collect(); // 던전 씬에서 마을로 이동시 가비지컬렉터 사용
SceneManager.LoadScene("2.town");
```

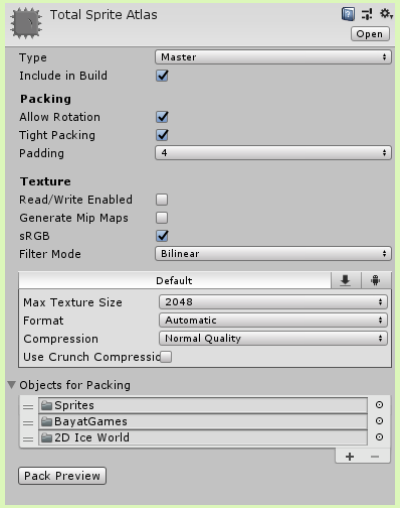
2D Color_RPG

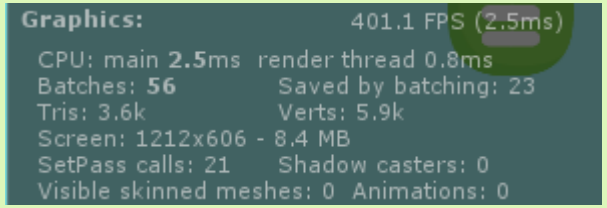
3. 주요구현부

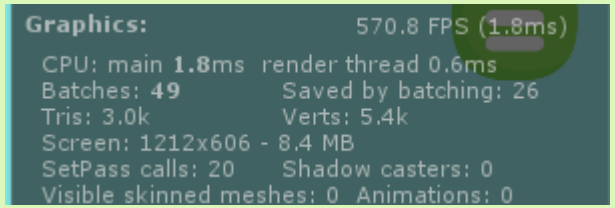
최적화 작업

3. 스프라이트 아틀라스

여러개의 스프라이트를 한장의 큰 텍스쳐로 모아 관리한다. Batches ,Verts가 소폭 감소







4. 오브젝트 풀링

새로 생성하지 않고 풀링하여 재사용

```
public IEnumerator revival() // 오브젝트를 새로 생성하지 않고 풀링하여 재사용
{
    yield return new WaitForSeconds(revival_term);

    transform.position = orignal_position;
    current_health = max_health;
    hp_slider.value = current_health / max_health;    // 체력,포지션,콜라이더를 초기상태로 재설정
    cap_col.isTrigger = false;
    isdead = false;
    rb.constraints = RigidbodyConstraints2D.None | RigidbodyConstraints2D.FreezeRotation;
    rend.enabled = true;
    hp_slider.gameObject.SetActive(true);          // 새로 생성하지 않고 오브젝트를 풀링하여 재사용

}
```

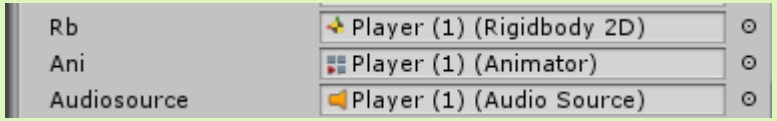
2D Color_RPG

3. 주요구현부

최적화 작업

5. 캐싱

참조할 컴포넌트를 미리 캐싱하여 사용하고 캐싱이 되지 않은 경우 스크립트를 통해 한번만 캐싱하여 사용




```
void Start()
{
    if (!rb)
    {
        rb = GetComponent<Rigidbody2D>();
    }
    if (!ani)
    {
        ani = GetComponent<Animator>();
    }
    if (!audiosource)
    {
        audiosource = GetComponent<AudioSource>();
    }
```

감사합니다!