GET 6 CHALIES

1. 게임 기획하기

- 평소에 포켓몬스터나 스타듀밸리, 바람의나라 같은 게임을 선호하여 2D RPG 게임을 만들고자 함.
- 이를 위해 학기 초부터 교내 도서관에 있는 유니티 관련 책을 많이 참고하고 유튜브 강좌나 유니티 도큐먼트를 보면서 유니티 엔진 자체에 대해 공부하려고 노력했고, 학교 각 과제에 그를 적용해보면서 유니티 기본기를 다짐.
- 지난 겨울의 동계 올림픽을 재밌게 보아서 쇼트트랙 게임을 제작하고자 생각하고 곽윤기 선수 캐릭터를 도트로 찍었으나, 빙판 얼음 처리 대각선 이동 그림 등의 효과가 어렵고 정보가 부족하여 새롭게 기획하게 됨.
- '히트맨'이란 3D 게임에서 영감을 받아 2D 버전 히트맨을 생각함. 따라서 도둑이 아이템을 훔쳐서 도망가는 게임을 생각하고, 아이템을 훔친 것에 대해 순찰을 보는 경찰 Enemy들이 캐릭터를 쫓아오는 게임을 기획.
- 맵 기획에 있어서는 경찰들이 도둑을 쫓아온다는 점이 '팩맨'이 플레이어를 따라오는 것과 비슷하다고 생각해서 '팩맨'의 맵 구성을 참고해 유니티 Tilemap의 Collision을 배치.
- 게임을 만드는 것은 스크립트를 쓰는 것이 영화 제작과 비슷하다고 생각해 스토리가 있으면 좋겠다고 생각함. 따라서 Player가 미션을 받고 수행하는 게임을 만들고자 함.
- 히트맨에서 누군가를 포획하거나 Kill하는 미션을 구현하긴 어렵다고 생각해서 맵 전체를 탐험할 수 있도록 6개의 성배를 배치해 성배를 구해오는 미션을 생각함. 성배인 이유는 게임에서 많이 사용되는 아이템으로 에셋을 구하기 쉬웠고, 마을의 제사를 위해 등 스토리를 엮기 쉽다고 생각했기 때문임.
- 탐험 후 다른 맵으로 이동해 이후의 스토리에 대한 엔딩을 넣고자 함.
- Player가 처음부터 게임의 키를 알지 못하기 때문에 튜토리얼로 기본 동작을 알려주고자 함.

- 따라서, 게임 시작 화면 -> 튜토리얼 씬 -> 주요 게임화면(성 안) -> 미션 성공 후(동글) 이러한 순서로 기획함.
- 여러 게임 오브젝트들 파괴할 수 있는 아이템과 사용할 수 있는 아이템 등을 sprite만 있다면 쉽게 확장 할 수 있도록 처음에 오래 걸리더라도 꼼꼼하게 작성하고자 하였고, 게임을 만들어 나가면서 추가하려고 함.
- Enemy가 특정 지역을 순찰하고 있거나 그 자리를 지키고 있다가 특정 Range 안에 들어가면 Player를 추격하도록 함. Enemy와 Player간 공격하고 HP가 소모되게 함과 같은 내용들은 게임을 하는 사람이 직관적으로 알 수 있기에 만들면서 여러 요소를 생각함.

2. 사용한 오브젝트

- Unity Tilemap 배경 제작
  : Overworld, rowboat, cave, Castle2, tiles 등, assets-art-worlds에 있는 것들 것, Inner 폴더 (집 내부용),
  + 모두 사용하진 않음(추후 맵 수정이나 제작을 위해 미리 넣어 놓은 것들과 사용한 것들)
- 캐릭터
  : 주인공 캐릭터(이미 존재하는 Sprite를 받아온 것. 게임 완성 후 도둑 느낌으로 수정하려 했으나 중간에서 알파버전으로 임시 완성하고 제출해서 수정하지 못함), Enemy 경찰 캐릭터(주인공 캐릭터를 포토샵으로 불러와 원하는 색을 팔레트화 시켜놓고 도트 찍음), NPC 2명
  * 3d 에셋에 비해 2d 에셋은 적고, 4way asset의 경우 특히 부족함
- 캐릭터 체력(하트), 깰 수 있는 아이템(항아리), 먹을 수 있는 아이템(하트), 집을 수 있는 아이템(성배), 푯말 등 -> Objects 폴더
- 게임 시작 화면 배경 및 시작 화면 버튼 : 포토샵을 통해 성배 이미지를 따서 다시그려(화질때문) 직접 제작

3. 각 게임 오브젝트에 설정된 컴포넌트(Transform은 생략)

*** BoxCollider2D가 2개인데 is Trigger가 체크된 것과 체크되지 않은 것이 함께 들어가 있는 이유는 isTrigger 체크는 충돌 여부를 넓게 잡기 위함이고 체크 되지 않은 것은 그 오브젝트 자체의 영역으로 지나가지 못하게 하기 위함.**

- Player: Sprite Renderer, Rigidbody 2D, BoxCollider2D(is Trigger check), BoxCollider2D(is Trigger not check), Animator, PlayerMovement, ContextClue, SignalListener
- Enemy(경찰) : Sprite Renderer, Rigidbody2D, BoxCollider2D(is Trigger check), BOxCollider2D(is Trigger not check), Knockback,
  PoliceManEnemy/PatrolPoliceMan
- Sign(푯말), npc
- 집 입구, 동글 들어가는 입구 등 empty Object를 포함하여 Scene을 전환하는 것 : BoxCollider2D(is Trigger check), Dungeon Transition/
- 하트(체력)= HeartHolder + HeartContainers + Heart
  : HeartHolder의 경우 enemy를 kill 할 때마다 경험치를 쌓게 해서 레벨업 시 공격력을 증가시키는 방향으로 할려고 Signal Listener(스크립터블 오브젝트인 Signal을 인지하는 스크립트)를 달았지만 사실상 현재 제출에선 사용하지 않음 + Horizontal LayoutGroup, HeartManager
- 성배(Chalices) : Sprite Renderer, BoxCollider2D(is Trigger check), BoxCollider2D(is Trigger not check), Interactable, PhysicalInventoryItem
- * 아래쪽으로는 UI 관련 요소들 (Canvas는 제외하고 그 안으로 달린 것을 Panel 기준으로 나눠서 설명 * (Canvas Renderer, Image 등 생략)
- Inventory Panel = Inventory Background + ScrollView + Item Description
  : InventoryManager = Image, Text + Image, Scroll Rect + Image, text, button(컴포넌트는 아니지만 밑으로 달린 오브젝트들을 함께 적음)
- ChalicesLack Panel , Lack Text = Image, Text
- GameOver Panel = GameOver Text + RestartButton

= Image, Text, Image, Button 등

- Pot : SpriteRenderer, BoxCollider2D, Animator, Pot
- MainCamera : 기본 설정 + AudioSource, CinemachineBrain
- FadeFromWhite/ FadeToWhite : Canvas로 만들어 그 하위의 Panel에 Animator를 넣어 씬 전환시 화면 깜빡하는 효과
- EndingCredit : Ending, BoxCollider2D + 하위로 Canvas와 Panel(크레딧 위에 Text 작성을 위함)
- Title 씬에 있는 캔버스 등은 정적으로 이미지 제작해서 넣는 것으로 하고 위에서 계속 사용한 Canvas, Panel, Button의 구조를 이용함

4. 작성한 스크립트 (코드 삽입하고 코드에 대한 설명 추가하여 제출)

*** 후반 부에 작성한 코드일수록 급하게 동작하게 하기 위해 같이 작동 시킬 수 있는 것도 스크립트를 추가해서 넣었기 때문에 비효율적인 코드가 많습니다. Enemy에 오브젝트 풀링 등으로 최적화 작업도 마찬가지로 되어있지 않습니다.**

1) Player 관련 스크립트

- PlayerMovement

```
참조 11개
public enum PlayerState
{
    walk,
    attack,
    interact,
    stagger,
    idle
}
```

Animation 처리 및 상태전이를 위해, 보편적으로 사용하는 FSM(유한상태머신)패턴을 참고함.
https://ozt88.tistory.com/8

```
14  public class PlayerMovement : MonoBehaviour
15  {
16      public PlayerState currentState; // 위에서 Enum으로 나열한 현재 상태
17      public float speed; // 캐릭터의 walk speed
18      public Rigidbody2D myRigidbody; // 물리처리를 위한 리지드바디
19      private Vector3 change; //
20      private Animator animator; //캐릭터 좌우상하, 공격 동작을 위한 애니매이터
21      // FloatValue 타입은 사용자 지정 타입(뒤쪽 스크립트 설명 통해 설명)
22      // 현재의 HP를 담는 변수
23      public FloatValue currentHealth;
24      // Player의 HP와 관련해서 Hearts container에게 알리기 위한 변수
25      public Signal playerHealthSignal;
26      // 특정 맵에서 캐릭터의 시작 위치를 정하는 변수(VectorValue타입은 사용자 지정 타입)
27      public VectorValue startingPosition;
28      // *구더기 코드
29      //  player의 currentHealth로 게임 오버를 체크하기 위한 bool 변수
30      public bool gameOver;
31
32
33      // 한 번만 불려지는 함수이므로, 주된 일이 위의 변수들의 초기화 담당
        Unity 메시지 | 참조 0개
34      void Start()
35      {
36          currentState = PlayerState.walk;
37          animator = GetComponent<Animator>();
38          myRigidbody = GetComponent<Rigidbody2D>();
39          animator.SetFloat("moveX", 0);
40          animator.SetFloat("moveY", -1); //Player 가 시작시 앞방향을 보도록
41          // 스타트 위치 초기화
42          this.transform.position = startingPosition.initialValue;
43          gameOver = false;
44      }
```

함수 (update 제외)

// update함수에서 어떻게 동작하는 지

```
46      // Update is called once per frame
        Unity 메시지 | 참조 0개
47      void Update()
48      {
49          change = Vector3.zero;
50          change.x = Input.GetAxisRaw("Horizontal"); // x축 방향 입력
51          change.y = Input.GetAxisRaw("Vertical"); // y축 방향 입력
52          if (Input.GetButtonDown("attack")&& currentState!=PlayerState.attack
53              && currentState != PlayerState.stagger) // left ctrl + 상태 전이 등 check , Attack은 미리 ctrl키로 연결해놓음
54          {
55              StartCoroutine(AttackCo()); // ctrl 시 공격
56          }
57          else if (currentState == PlayerState.walk || currentState == PlayerState.idle)
58          {
59              UpdateAnimationAndMove(); //평상시는 그저 이동
60          }
61
62
63      }
```

```
    참조 1개
65  private IEnumerator AttackCo() // Animation과 싱크를 맞추기 위한 공격 코루틴함수
66  {
67      animator.SetBool("attacking", true); //애니메이션 동작 설정
68      currentState = PlayerState.attack; //상태전이
69      yield return null;
70      animator.SetBool("attacking", false); //동작
71      yield return new WaitForSeconds(0.3f); //싱크용 코루틴
72      currentState = PlayerState.walk; // 상태전이
73  }
74
    참조 1개
75  void UpdateAnimationAndMove() // 움직이는 방향과, 걸음에 관한 애니매이션
76  {
77      if (change != Vector3.zero)
78      {
79          MoveCharacter();
80          animator.SetFloat("moveX", change.x);
81          animator.SetFloat("moveY", change.y);
82          animator.SetBool("walking", true);
83      }
84      else
85      {
86          animator.SetBool("walking", false);
87      }
88  }
```

```
void MoveCharacter() // 캐릭터의 실질적 이동
{
    change.Normalize();
    myRigidbody.MovePosition(this.transform.position + change * speed * Time.deltaTime);
}

//  플레이어가 Enemy로부터 공격받았을 시 hp감소 및 동작을 제어하기 위한 함수
// 체력 미달 시 player 를 setactive(false)로 감춤 (Destory 함수보다 메모리 효율적)
참조 1개
```

```
96      //  플레이어가 Enemy로부터 공격받았을 시 hp감소 및 동작을 제어하기 위한 함수
97      // 체력 미달 시 player 를 setactive(false)로 감춤 (Destory 함수보다 메모리 효율적)
        참조 1개
98      public void Knock(float knockTime, float damage)
99      {
100         currentHealth.RuntimeValue -= damage;
101         playerHealthSignal.Raise(); // HeartsContainer로 Signal 보냄
102         if (currentHealth.RuntimeValue > 0)
103         {
104             gameOver = false;
105             StartCoroutine(KnockCo(knockTime));
106         }
107         else
108         {
109             gameOver = true;
110             this.gameObject.SetActive(false);
111         }
112     }
113
        참조 1개
114     private IEnumerator KnockCo(float knockTime) // 애니메이션과 상태전이 동작 맞추기 위한 코루틴
115     {
116         if (myRigidbody != null)
117         {
118             yield return new WaitForSeconds(knockTime);
119             myRigidbody.velocity = Vector2.zero;
120             currentState = PlayerState.idle;
121             myRigidbody.velocity = Vector2.zero;
122         }
123     }
124 }
```

- PlayerHit

```
    Unity 스크립트(자산 참조 4개)|참조 0개
5   public class PlayerHit : MonoBehaviour
6   {
7
8       // Enemy가 아닌 일반 object들 을 부시기 위한 것.
9       // 현재 게임에는 pot 오브젝트만 breakable Tag를 달아 부실 수 잇음
10      // pot 외에도 tag를 이용해 다른 오브젝트를 추가할 수 있음
        Unity 메시지|참조 0개
11      private void OnTriggerEnter2D(Collider2D other)
12      {
13          if (other.CompareTag("breakable"))
14          {
15              other.GetComponent<pot>().Smash();
16          }
17      }
18  }
19
```

- Knockback

```
Unity 스크립트(자산 참조 7개)|참조 0개
public class Knockback : MonoBehaviour
{
    // breakable Item인 pot과 공격을 받을 수 있는 player와 enemy에 대해
    // 공격 시 상태전이 처리

    public float thrust;
    public float knockTime;
    public float damage;

    Unity 메시지|참조 0개
```

위에는 변수 부분

아래는 실질적인 함수

```
    Unity 메시지|참조 0개
    private void OnTriggerEnter2D(Collider2D other) // 접촉 상태시
    {
        if (other.gameObject.CompareTag("breakable") && this.gameObject.CompareTag("Player"))
        {
            other.GetComponent<pot>().Smash(); //상대방이 breakable이고, 현재가 player이면 smash
            // 즉 enemy말고 player만 pot을 break할 수 잇음
        }

        if (other.gameObject.CompareTag("Enemy") || other.gameObject.CompareTag("Player"))
        {
            // enemy와 player는 서로 공격할 수 있는 유일한 대상임 그에 대한 상호간 처리
            Rigidbody2D hit = other.GetComponent<Rigidbody2D>();
            if(hit != null)
            {
                // 공격되는 방향을 계산해 공격받으면 공격받은 대상이 뒤족으로 밀리는 처리
                Vector2 difference = hit.transform.position - this.transform.position;
                difference = difference.normalized * thrust;
                hit.AddForce(difference, ForceMode2D.Impulse);

                // Enemy인지 Player인지에 따라 따로 처리
                if (other.gameObject.CompareTag("Enemy") && other.isTrigger)
                {
                    hit.GetComponent<Enemy>().currentState = EnemyState.stagger;
                    other.GetComponent<Enemy>().Knock(hit, knockTime,damage);
                }
                if (other.gameObject.CompareTag("Player"))
                {
                    if (other.GetComponent<PlayerMovement>().currentState != PlayerState.stagger)
                    {
                        hit.GetComponent<PlayerMovement>().currentState = PlayerState.stagger;
                        other.GetComponent<PlayerMovement>().Knock(knockTime, damage);
                    }
                }

            }
```

2) **Enemy(경찰) 관련 스크립트**

- Enemy (경찰 Enemy 제외하고 추가적인 Enemy가 생길 시, 이 Enemy라는 클래스를 상속받아서 사용하면 됨)

상태전이를 위한 Enum 열거

```
4
        참조 17개
5       public enum EnemyState
6       {
7           idle,
8           walk,
9           attack,
10          stagger
11      }
12
```

전체적으로 Player와 Enemy의 움직임 자체가 유사하기 때문에 Player와 비슷한 코드

```
        Unity 메시지 | 참조 0개
24      private void Awake() // 초기화 // Start 함수보다 먼저 실행됨
25      {
26          health = maxHealth.initialValue;
27      }
28      // Player가 공격시 받아 쓰게할 damage를 받았을 시의 변수
        참조 1개
29      private void TakeDamage(float damage)
30      {
31          health -= damage;
32          if(health <= 0)
33          {
34              DeathEffect();
35              Destroy(this.gameObject); // destory보단 setactive(false)로 변경할것
36          }
37      }
38       // 죽음 효과 함수 (enemy의 투명도를 조절함)
        참조 1개
39      private void DeathEffect()
40      {
41          if(deathEffect != null)
42          {
43              GameObject effect = Instantiate(deathEffect,this.transform.position, Quaternion.identity);
44              Destroy(effect, 1f);
45          }
46      }
```

```
47      // 공격을 받았을 시 수행되게 하는 함수
        참조 1개
48      public void Knock(Rigidbody2D myRigidbody, float knockTime, float damage)
49      {
50          StartCoroutine(KnockCo(myRigidbody, knockTime));
51          TakeDamage(damage);
52      }
53      // 위의 함수의 실질적인 함수로 코루틴을 통해 싱크를 맞춤
        참조 1개
54      private IEnumerator KnockCo(Rigidbody2D myRigidbody, float knockTime)
55      {
56          if (myRigidbody != null )
57          {
58              yield return new WaitForSeconds(knockTime);
59              myRigidbody.velocity = Vector2.zero;
60              currentState = EnemyState.idle;
61              myRigidbody.velocity = Vector2.zero;
62          }
63      }
64  }
65
```

- PoliceManEnemy

: Enemy 클래스를 상속받아서 죽음과 Knock 처리는 필요 없음.

이 스크립트를 컴포넌트로 가진 Enemy는 자기 자리를 지키다가 Range 이내로 Player가 들어오면 추격

변수 및 초기화

```
Unity 스크립트(자산 참조 2개)|참조 1개
public class PoliceManEnemy : Enemy
{
    // PoliceManEnemy는 Enemy 클래스를 상속받아서 죽음과 Knock 처리는 필요 없음
    // 이 클래스를 받은 Enemy는 자기 자리를 지키다가 Range이내로 player가 들어오면 추격
    public Rigidbody2D myRigidbody;
    public Transform target;
    public float chaseRadius; // 추격 반지름
    public float attackRadius;
    public Transform homePosition; // 자기 고정 위치
    public Animator anim;

    // 초기화
    Unity 메시지|참조 0개
    void Start()
    {
        currentState = EnemyState.idle;
        myRigidbody = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        target = GameObject.FindWithTag("Player").transform;
    }
    // 프레임 단위 upate가 아닌 규칙적인 간격으로 update 되는 FixedUpdate를 사용
    Unity 메시지|참조 0개
    void FixedUpdate()
    {
        CheckDistance();
    }

```

FixedUpdate

```
    }
    // 프레임 단위 upate가 아닌 규칙적인 간격으로 update 되는 FixedUpdate를 사용
    Unity 메시지|참조 0개
    void FixedUpdate()
    {
        CheckDistance();
    }

    참조 2개
```

함수들

```
public virtual void CheckDistance() // target인 Player와의 거리를 체크해 추격, 공격 여부 적용
{
    if(Vector3.Distance(target.position, this.transform.position) <= chaseRadius) // 추격범위 내이고
    {
        if(Vector3.Distance(target.position, this.transform.position) > attackRadius) // 공격 범위 밖인 경우 쫓아와
        {
            if (currentState == EnemyState.idle || currentState == EnemyState.walk || currentState == EnemyState.attack && currentState != EnemyState.stagger)
            {
                Vector3 temp = Vector3.MoveTowards(this.transform.position, target.position, moveSpeed * Time.deltaTime);

                changeAnim(temp - transform.position);
                myRigidbody.MovePosition(temp);

                ChangeState(EnemyState.walk);
                anim.SetBool("walking", true);
                anim.SetBool("attacking", false);
            }
        }
        else // 공격범위 내인 경우
        {
            Vector3 temp = Vector3.MoveTowards(this.transform.position, target.position, moveSpeed * Time.deltaTime);

            changeAnim(temp - transform.position);
            myRigidbody.MovePosition(temp);

            ChangeState(EnemyState.attack);
            anim.SetBool("attacking", true);
        }
    }
    else if (Vector3.Distance(target.position, this.transform.position) > chaseRadius) // 추격 범위 밖인 경우
    {

        anim.SetBool("walking", false);
        anim.SetBool("attacking", false);
    }
}
```

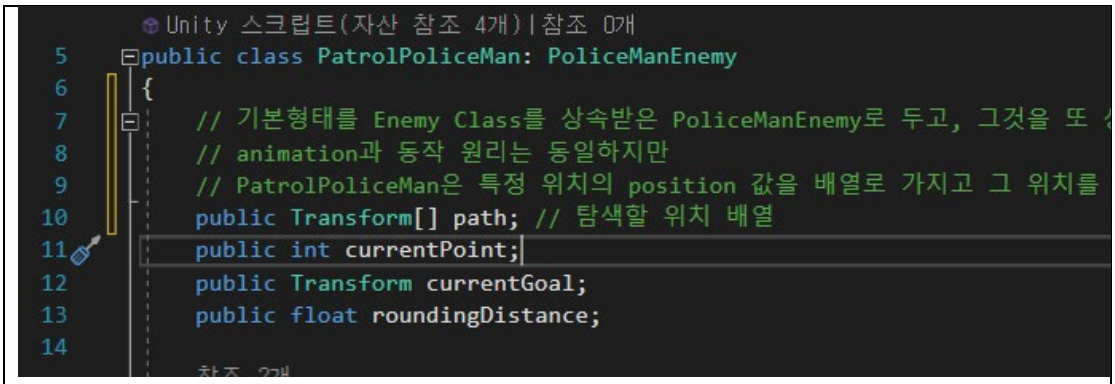
```
// Player와 비슷하게 x축은 좌우  y축은 상하로 값 설정
참조 4개
private void SetAnimFloat(Vector2 setVector)
{
    anim.SetFloat("moveX", setVector.x);
    anim.SetFloat("moveY", setVector.y);
}
```

```
75      // Player의 위치에 따라 방향을 조정해서 이동하게 함
        참조 5개
76      public void changeAnim(Vector2 direction)
77      {
78          if (Mathf.Abs(direction.x) > Mathf.Abs(direction.y)
79          {
80              if (direction.x > 0)
81              {
82                  SetAnimFloat(Vector2.right);
83              }
84              else if (direction.x < 0)
85              {
86                  SetAnimFloat(Vector2.left);
87              }
88          }
89          else if (Mathf.Abs(direction.x) < Mathf.Abs(directi
90          {
91              if(direction.y > 0)
92              {
93                  SetAnimFloat(Vector2.up);
94              }
95              else if(direction.y < 0)
96              {
97                  SetAnimFloat(Vector2.down);
98              }
99          }
100     }
```

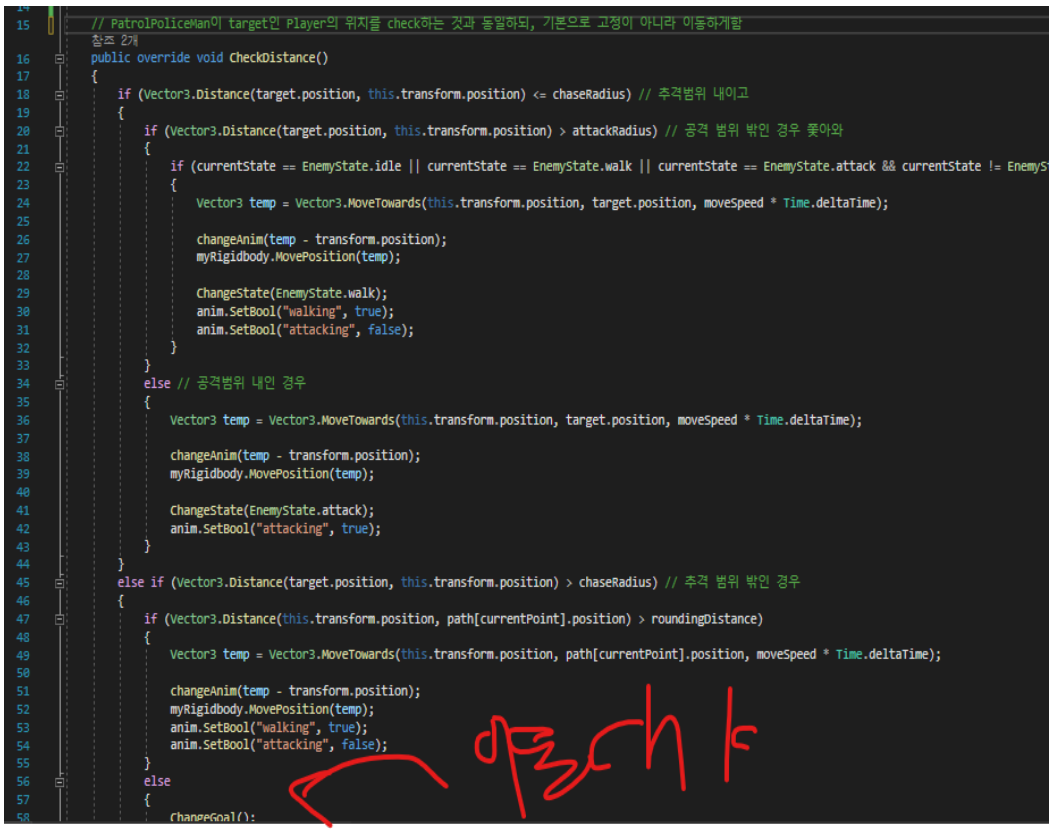
```
102     // 상태변화 함수 부모 클래스인 Enemy의 enum을 이용
        참조 4개
103     public void ChangeState(EnemyState newState)
104     {
105         if(currentState != newState)
106         {
107             currentState = newState;
108         }
109     }
110 }
111
```

- PatrolPoliceMan

: 기본형태를 Enemy class를 상속받은 PoliceMan Enemy로 두고, 그것을 상속 받은 클래스로 animation과 동작 원리는 PoliceManEnemy와 동일하지만 PatrolPoliceMan은 고정위치 순찰이 아니라 특정 위치를 동적으로 순찰하고 있음.



```
Unity 스크립트(자산 참조 4개)|참조 0개
public class PatrolPoliceMan: PoliceManEnemy
{
    // 기본형태를 Enemy Class를 상속받은 PoliceManEnemy로 두고, 그것을 또 
    // animation과 동작 원리는 동일하지만
    // PatrolPoliceMan은 특정 위치의 position 값을 배열로 가지고 그 위치를
    public Transform[] path; // 탐색할 위치 배열
    public int currentPoint;
    public Transform currentGoal;
    public float roundingDistance;

    참조 2개
```



```
// PatrolPoliceMan이 target인 Player의 위치를 check하는 것과 동일하되, 기본으로 고정이 아니라 이동하게함
참조 2개
public override void CheckDistance()
{
    if (Vector3.Distance(target.position, this.transform.position) <= chaseRadius) // 추격범위 내이고
    {
        if (Vector3.Distance(target.position, this.transform.position) > attackRadius) // 공격 범위 밖인 경우 쫓아와
        {
            if (currentState == EnemyState.idle || currentState == EnemyState.walk || currentState == EnemyState.attack && currentState != EnemyS
            {
                Vector3 temp = Vector3.MoveTowards(this.transform.position, target.position, moveSpeed * Time.deltaTime);

                changeAnim(temp - transform.position);
                myRigidbody.MovePosition(temp);

                ChangeState(EnemyState.walk);
                anim.SetBool("walking", true);
                anim.SetBool("attacking", false);
            }
        }
        else // 공격범위 내인 경우
        {
            Vector3 temp = Vector3.MoveTowards(this.transform.position, target.position, moveSpeed * Time.deltaTime);

            changeAnim(temp - transform.position);
            myRigidbody.MovePosition(temp);

            ChangeState(EnemyState.attack);
            anim.SetBool("attacking", true);
        }
    }
    else if (Vector3.Distance(target.position, this.transform.position) > chaseRadius) // 추격 범위 밖인 경우
    {
        if (Vector3.Distance(this.transform.position, path[currentPoint].position) > roundingDistance)
        {
            Vector3 temp = Vector3.MoveTowards(this.transform.position, path[currentPoint].position, moveSpeed * Time.deltaTime);

            changeAnim(temp - transform.position);
            myRigidbody.MovePosition(temp);
            anim.SetBool("walking", true);
            anim.SetBool("attacking", false);
        }
        else
        {
            ChangeGoal();
```

```
63      // 이동할 위치를 배열 변수로 받았기 때문에 현재 위치가 몇인지 chec
64      // 다음 이동할 target 위치로 이동하는 것을 처리하는 함수
        참조 1개
65      private void ChangeGoal()
66      {
67          if(currentPoint == path.Length - 1)
68          {
69              currentPoint = 0;
70              currentGoal = path[0];
71          }
72          else
73          {
74              currentPoint++;
75              currentGoal = path[currentPoint];
76          }
77      }
78  }
```

3) Scene 전환 관련 스크립트

- DungeonTransition

```
6   public class DungeonTransition : MonoBehaviour
7   {
8       public string sceneToLoad; // Loading할 씬 이름을 컴포넌트 창에서 받음
9       // room의 경우 들어왔다 나가는 것을 고려해서 현 위치를 저장함
10      public Vector2 playerPosition;
11      public VectorValue playerStorage;
12      // 씬 전환시 자연스러움을 인해 fadeIn, FadeOut 처리 Panel
13      // 및 씬 로드시 필요한 시간을 위한 처리
14      public GameObject fadeInPanel;
15      public GameObject fadeOutPanel;
16      public float fadeWait;
17      // GameManager가 성배의 수를 chk 하므로 필요
18      public GameManager theGM;
19      public GameObject chalicesLackPanel;
20
```

```
22      // 초기화
        Unity 메시지 | 참조 0개
23      private void Awake()
24      {
25          if (fadeInPanel != null)
26          {
27              GameObject panel = Instantiate(fadeInPanel, Vector3.zero, Quaternion.identity) as GameObject;
28              Destroy(panel, 1);
29          }
30          //FindObjectOfType는 메모리에 별로 좋지 않다고 해서 수정필요
31          theGM = FindObjectOfType<GameManager>();
32          chalicesLackPanel.SetActive(false); //기본적으로 안보이게
33      }
34
```

```
35      // BoxCollider 2D 에 들어오면 씬 전환 혹은 성배개수 부족에 대해 안내하게 하는 함수
        Unity 메시지 | 참조 0개
36      public void OnTriggerEnter2D(Collider2D other)
37      {
38          // 성배개수에 대한 체크는 GameManager의 cahlices 변수를 통해서 함 (public 처리)
39          if (other.CompareTag("Player") && !other.isTrigger && theGM.chalices == 6)
40          {
41              playerStorage.initialValue = playerPosition;
42              StartCoroutine(FadeCo());
43              SceneManager.LoadScene(sceneToLoad);
44          }
45          else if (other.CompareTag("Player") && !other.isTrigger &&theGM.chalices < 6)
46          {
47              chalicesLackPanel.SetActive(true);
48          }
49      }
```

```
49      }
50      // 멀어지면 성배개수 부족 Panel 끄기 위함
        Unity 메시지 | 참조 0개
51      public void OnTriggerExit2D(Collider2D other)
52      {
53          if (other.CompareTag("Player"))
54          {
55              chalicesLackPanel.SetActive(false);
56          }
57      }
58      // Fade Out과 Fade In 효과를 위한 코루틴 함수.
```

```
58      // Fade Out과 Fade In 효과를 위한 코루틴 함수.
        참조 1개
59      public IEnumerator FadeCo()
60      {
61          if (fadeOutPanel != null)
62          {
63              Instantiate(fadeOutPanel, Vector3.zero, Quaternion.identity);
64          }
65          yield return new WaitForSeconds(fadeWait);
66
67          AsyncOperation asyncOperation = SceneManager.LoadSceneAsync(sceneToLoad);
68          while (!asyncOperation.isDone)
69          {
70              yield return null;
71          }
72      }
73  }
```

- SceneTransition

: Dungeon Transition과 사실상 동일함 성배 체크 등이 없는 것만 다를 뿐

임.

```
Unity 스크립트(자산 참조 1개)|참조 0개
6   public class SceneTransition : MonoBehaviour
7   {
8
9       public string sceneToLoad; // 이동할 씬 이름
10      // 돌아올 때 대비용 플레이어 위치 저장
11      public Vector2 playerPosition;
12      public VectorValue playerStorage;
13      // FadeOut-FadeIn, 그 시간 코루틴 처리를 위함
14      // awake나 start와 같은 처리를 할 때 걸리는 시간을 위해
15      // fadeWait로 그 시간을 확보하고
16      // FadeIn-Out 효과로 갑자기 빡! 하고 나타나는 것이 아닌
17      // 자연스러운 씬 전환 처리를 할 수 있음
18      public GameObject fadeInPanel;
19      public GameObject fadeOutPanel;
20      public float fadeWait;
21
```

```
Unity 메시지|참조 0개
22  private void Awake()
23  {
24      if (fadeInPanel != null) // 일종의 Panel에 대한 싱글톤 처리
25      {
26          GameObject panel = Instantiate(fadeInPanel, Vector3.zero, Quaternion.identity) as GameObject;
27          Destroy(panel, 1);
28      }
29  }
30
```

```
31      // Player가 그 위치로 왔을 시 플레이어의 현 위치를 저장하고
32      // 다른 씬을 로드
        Unity 메시지|참조 0개
33      public void OnTriggerEnter2D(Collider2D other)
34      {
35          if (other.CompareTag("Player")&& !other.isTrigger)
36          {
37              playerStorage.initialValue = playerPosition;
38              StartCoroutine(FadeCo()); // Fade관련 코루틴에서 실질적으로 씬 불러옴
39              //SceneManager.LoadScene(sceneToLoad);
40          }
41      }
```

```
43      // Fade-out 처리
        참조 1개
44      public IEnumerator FadeCo()
45      {
46          if (fadeOutPanel != null) // 하나만 있게 하기 위함, 일종의 싱글톤 처
47          {
48              Instantiate(fadeOutPanel, Vector3.zero, Quaternion.identity);
49          }
50          yield return new WaitForSeconds(fadeWait); // 시간 확보
51
52          // 씬 로드 (싱크를 맞추기 위함)
53          AsyncOperation asyncOperation = SceneManager.LoadSceneAsync(scene
54          while (!asyncOperation.isDone)
55          {
56              yield return null;
57          }
58      }
```

- RoomMove

: CamearMovement와 함께 카메라에 대한 처리를 할 때, 씬 전환이 아닌 해당 씬 내에서 구역을 나누어서 구역에 대한 카메라 처리를 할 때 카메라에 바운드를 주며 이동시키기 위해 필요했던 거였는데 씨네머신 처리로 인해서 사실상 필요 없어진 스크립트

```
        Unity 스크립트(사산 참조 1개)|참조 0개
6       public class RoomMove : MonoBehaviour
7       {
8           public Vector2 cameraChange; // 카메라의 변하를 위함
9           public Vector3 playerChange; //플레이어의 위치 값을 가지고 있음
10          private CameraMovement cam; // 카메라 무브먼트 클래스로 이동 처리를 위함
11
12          // 변화된 장소의 이름 등에 대한 처리
13          // Canvas에 TExt를 붙여서 그 이름이 뜨게 함
14          public bool needText;
15          public string placeName;
16          public GameObject text;
17          public Text placeText;
18
```

```
19          void Start()
20          {
21              // 초기화
22              cam = Camera.main.GetComponent<CameraMovement>();
23          }
```

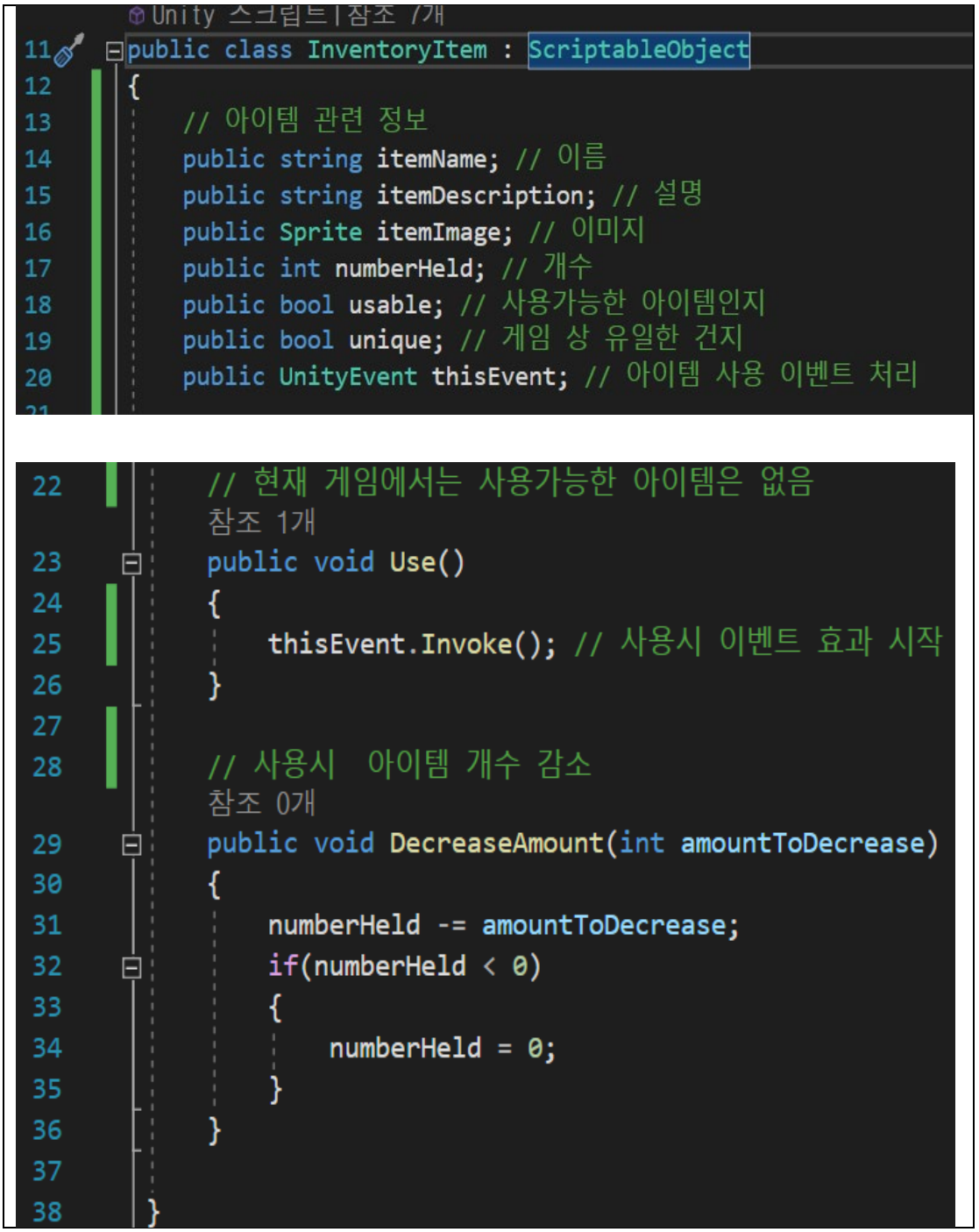
```
Unity 메시지 | 참조 0개
25  private void OnTriggerEnter2D(Collider2D other)
26  {
27      //  한 씬 내에서 여러 room으로 이루어졌을 때 그 룸의 경계에
28      // 각각의 방 크기에 맞게 카메라 값 처리를 함.
29      if (other.CompareTag("Player") && !other.isTrigger)
30      {
31          cam.minPosition += cameraChange;
32          cam.maxPosition += cameraChange;
33          other.transform.position += playerChange;
34          if (needText)
35          {
36              StartCoroutine(placeNameCoroutine());
37          }
38      }
```

```
40  // 룸 변화시 그 이름이 화면에 뜨도록 하기 위한 처리
    참조 1개
41  private IEnumerator placeNameCoroutine()
42  {
43      text.SetActive(true);
44      placeText.text = placeName;
45      yield return new WaitForSeconds(1.5f);
46      text.SetActive(false);
47  }
48  }
```

4) **Inventory 관련 스크립트**

- InventoryItem

```
6   // Unity에 마치 기본으로 있는 것처럼
7   // NewItem이란 것 하위로 Inventory/Items를 넣어서
8   // 구현을 신경 쓰지 않고 할 수 있게 함.
9   [CreateAssetMenu(fileName ="New Item", menuName ="Inventory/Items")]
10
```



```
Unity 스크립트 | 참조 7개
public class InventoryItem : ScriptableObject
{
    // 아이템 관련 정보
    public string itemName; // 이름
    public string itemDescription; // 설명
    public Sprite itemImage; // 이미지
    public int numberHeld; // 개수
    public bool usable; // 사용가능한 아이템인지
    public bool unique; // 게임 상 유일한 건지
    public UnityEvent thisEvent; // 아이템 사용 이벤트 처리

    // 현재 게임에서는 사용가능한 아이템은 없음
    참조 1개
    public void Use()
    {
        thisEvent.Invoke(); // 사용시 이벤트 효과 시작
    }

    // 사용시  아이템 개수 감소
    참조 0개
    public void DecreaseAmount(int amountToDecrease)
    {
        numberHeld -= amountToDecrease;
        if(numberHeld < 0)
        {
            numberHeld = 0;
        }
    }

}
```

- InventorySlot

```
Unity 스크립트(자산 참조 1개)|참조 2개
public class InventorySlot : MonoBehaviour
{
    // Header : 유니티 컴포넌트에 가따되면 설명이 뜸
    [Header("UI stuff to change")]
    // private로 외부에서 사용하지 못하게 막고
    // serialziedField로 직렬화 처리(public처럼 컴포넌트에서 보이게)
    [SerializeField] private Text itemNumberText;
    [SerializeField] private Image itemImage;

    // Header로 설명
    [Header("Variables from the item")]
    public InventoryItem thisItem; // 현재는 슬롯이므로 아이템 처리
    public InventoryManager thisManager;  //메니저로 대부분 처리

    // 슬롯에 새로운 아이템이 추가 될 때, 업데이트를 위한 처리
    참조 1개
```

```
    // 슬롯에 새로운 아이템이 추가 될 때, 업데이트를 위한 처리
    참조 1개
    public void Setup(InventoryItem newItem, InventoryManager newManager)
    {
        thisItem = newItem;
        thisManager = newManager;
        if(thisItem)
        {
            itemImage.sprite = thisItem.itemImage;
            itemNumberText.text = "" +  thisItem.numberHeld;
        }

    }
```

```
    // 클릭시 아이템에 대한 설명이 화면에 뜨도록 하기 위함
    // 구현은 Manager 클래스에 있음
    참조 0개
    public void ClickedOn()
    {
        if (thisItem)
        {
            thisManager.SetupDescriptionAndButton(thisItem.itemDescription,thisItem.usable, thisItem);
        }
    }
```

- PhysicalInventoryItem

: 대표적인 구더기 코드의 하나로, GameManager에서 성배를 count할 필요가 없는데 들어간 부분

```
Unity 스크립트(자산 참조 1개)|참조 0개
public class PhysicalInventoryItem : MonoBehaviour
{
    // 컴포넌트로 보이게는 하고 싶고
    // 외부에서 연결은 못하게 하고 싶음.
    // 따라서 직렬화(Serialzie처리)
    [SerializeField] private PlayerInventory playerInventory;
    [SerializeField] private InventoryItem thisItem;
    public bool leftShiftClicked; // 좌 shift 클릭시 Item(성배)를 get하기 위함
    public GameManager theGM;

    // 필요한 것 초기화
    Unity 메시지|참조 0개
    void Start()
    {
        leftShiftClicked = false; // 처음엔 눌려있지 않음
        theGM = FindObjectOfType<GameManager>();
    }
```

```
    Unity 메시지|참조 0개
    void Update()
    {
        // LeftShift키를 누르거나 땔 때 bool 변화
        if (Input.GetKeyDown(KeyCode.LeftShift))
        {
            leftShiftClicked = true;
        }
        else if (Input.GetKeyUp(KeyCode.LeftShift))
        {
            leftShiftClicked = false;
        }
    }
```

```
    // enter로 처리하게 되면 그 순간과 shift가 함께 처리되야하므로
    // stay를 사용해야함.
    Unity 메시지|참조 0개
    private void OnTriggerStay2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Player") && leftShiftClicked
            && this.gameObject.GetComponent<Interactable>().playerInRange)
        {
            AddItemToInventory();
            Destroy(this.gameObject);
        }
    }
```

```
44      // Item을 인벤토리로 추가하는 함수.
        참조 1개
45      void AddItemToInventory()
46      {
47          if (playerInventory && thisItem)
48          {
49              if (playerInventory.myInventory.Contains(thisItem))
50              {
51                  thisItem.numberHeld += 1; // Item 자체의 수 처리
52                  theGM.chalices++; // gamemanager에서 미션 여부를 체크하기 위함
53              }
54              else
55              {
56                  playerInventory.myInventory.Add(thisItem);
57                  thisItem.numberHeld += 1;
58                  theGM.chalices++;
59              }
```

- PlayerInventory

```
2   using System.Collections.Generic;
3   using UnityEngine;
4
5
6   // 유니티에 기본으로 있는 것처럼 New Inventory라는 상위 폴더 내로 하위로 menu를 넣음
7   [CreateAssetMenu(fileName ="New Inventory", menuName = "Inventory/Palyer Inventory") ]
8
9   // ScriptableObject는 유니티에서 제공하는 대량 데이터 저장을 위한 데이터 통(?)이다.
10  // 사본 생성을 방지해 메모리를 줄일 수 있다.
11  // 메모리에 스크럽터블 오브젝트의 데이터 사본을 저장해 이를 반복해서 참조하게 되므로
12  // 메모리를 줄일 수 잇게 되는 것이다.
    Unity 스크립트 | 참조 2개
13  public class PlayerInventory : ScriptableObject
14  {
15      // List로 실제로 플레이어가 가진 인벤토리의 아이템들을 처리함
16      public List<InventoryItem> myInventory = new List<InventoryItem>();
17  }
18
```

5) **Manager 스크립트**

- InventoryManager

```
Unity 스크립트(자산 참조 1개)|참조 2개
public class InventoryManager : MonoBehaviour
{
    // 컴포넌트에서 정보를 확인할 수 있도록
    [Header("Inventory Information")]
    // PlayerInventory에 담는 것을 List로 관리되는 것
    public PlayerInventory playerInventory;
    // 직렬화(앞서 계쏙 적엇으므로 생략)
    [SerializeField] private GameObject blankInventorySlot;
    [SerializeField] private GameObject inventoryPanel;
    [SerializeField] private Text descriptionText;
    [SerializeField] private GameObject useButton;
    public InventoryItem currentItem;
```

```
    // 아이템 클릭시 text를 그 아이템에 설명을 하단으로 뜨게하고
    // 그 아이템이 사용가능한 아이템(usable = true)이라면 button이 보임
    참조 2개
    public void SetTextAndButton(string description, bool buttonActive)
    {
        descriptionText.text = description;
        if (buttonActive) // true
        {
            useButton.SetActive(true);
        }
        else
        {
            useButton.SetActive(false);
        }
    }
```

```
    // Player가 Item을 가지게 되면 그 아이템 슬롯을 추가해서 보이게 하는 것을 계속 check함.
    // 우리에게 시각적으로 보이는 창에대한 실질적인 관리
    참조 2개
    void MakeInventorySlots()
    {
        if (playerInventory)
        {
            for(int i = 0; i<playerInventory.myInventory.Count; i++)
            {
                if (playerInventory.myInventory[i].numberHeld > 0 || playerInventory.myInventory[i].itemName == "Bottle")
                {
                    GameObject temp
                        = Instantiate(blankInventorySlot, inventoryPanel.transform.position, Quaternion.identity);
                    temp.transform.SetParent(inventoryPanel.transform);

                    InventorySlot newSlot = temp.GetComponent<InventorySlot>();

                    if (newSlot)
                    {
                        newSlot.transform.SetParent(inventoryPanel.transform);
                        newSlot.Setup(playerInventory.myInventory[i], this);
                    }
```

```
62      // start나 awake는 한 번만 호출되는데
63      // onEnable은 활성화 될 때마다 호출됨
64      // 인벤토리 창을 여러번 Shift를 눌러서 업데이트 하면서 아이템 변화를
65      // 매번 반영해주게 하기 위함
        Unity 메시지 | 참조 0개
66      void OnEnable()
67      {
68          ClearInventorySlots();
69          MakeInventorySlots();
70          SetTextAndButton("", false);
71      }
72
```

```
74      // 아이템에 대한 설명이 들어가고 아이템에 따라 버튼 여부를 활성화
        참조 1개
75      public void SetupDescriptionAndButton(string newDescriptionString, bool isButtonUsable, InventoryItem newItem)
76      {
77          currentItem = newItem;
78          descriptionText.text = newDescriptionString;
79          useButton.SetActive(isButtonUsable);
80      }
81
82      // onEnable 될 때마다 그 슬롯에 대한 check를 하면서 쫙 지우고 다시하도록
        참조 2개
83      void ClearInventorySlots()
84      {
85          for(int i = 0; i < inventoryPanel.transform.childCount; i++)
86          {
87              Destroy(inventoryPanel.transform.GetChild(i).gameObject);
88          }
89      }
90
```

```
91      // 사용가능한 아이템일 경우 버튼이 눌리게 되면 아이템 개수 줄이는 것
        참조 0개
92      public void UseButtonPressed()
93      {
94          if (currentItem)
95          {
96              currentItem.Use();
97              // Clear all of the inventory slots
98              ClearInventorySlots();
99              // Refill all slots with new numbers
100             MakeInventorySlots();
101             if (currentItem.numberHeld == 0)
102             {
103                 SetTextAndButton("", false);
104             }
105         }
106     }
```

- AudioManager

: 현재는 제일 처음의 게임 타이틀이 들어간 씬에서만 마우스 클릭시 소리

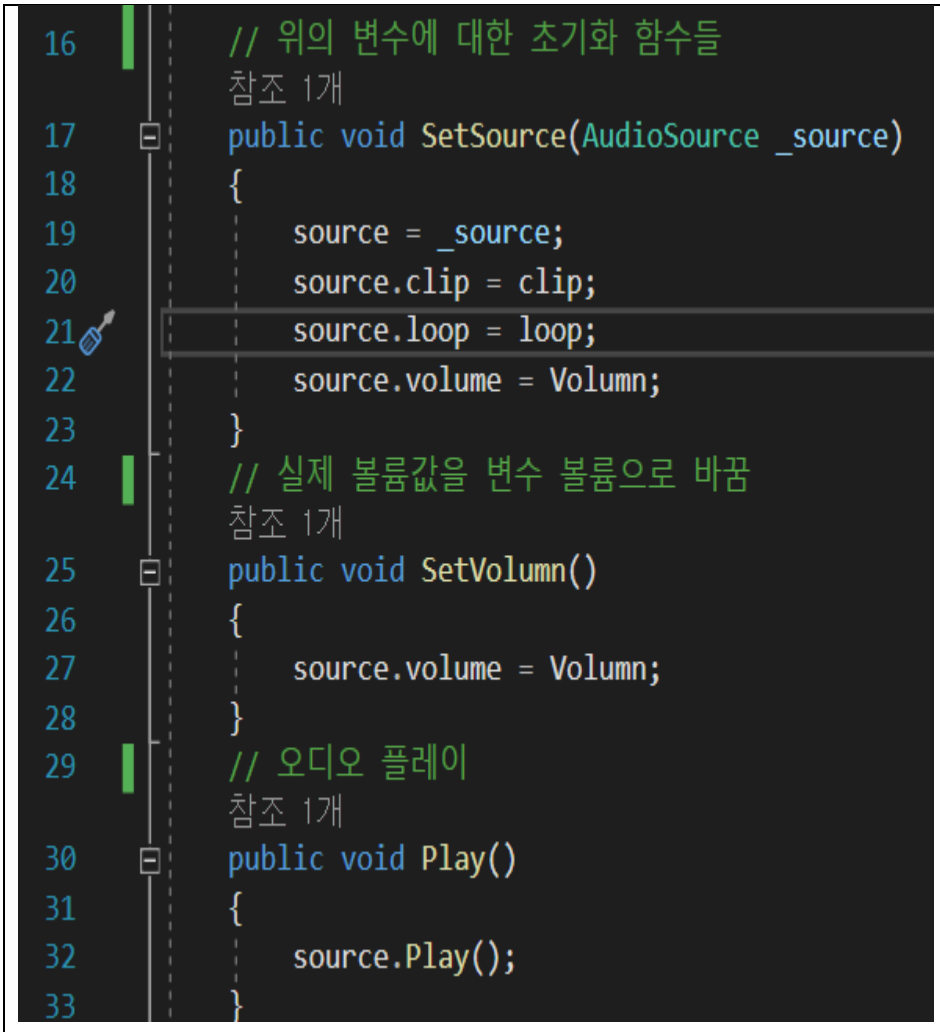
가 나는 정도로 사용되고 있는데, 아이템 pot 등을 부셨을 때 나는 효과음이나 공격 모션시 효과음 추가가 필요함. *유튜버 케이디님 코드 참고

+ Manager 클래스와 Sound 클래스를 함께 적음

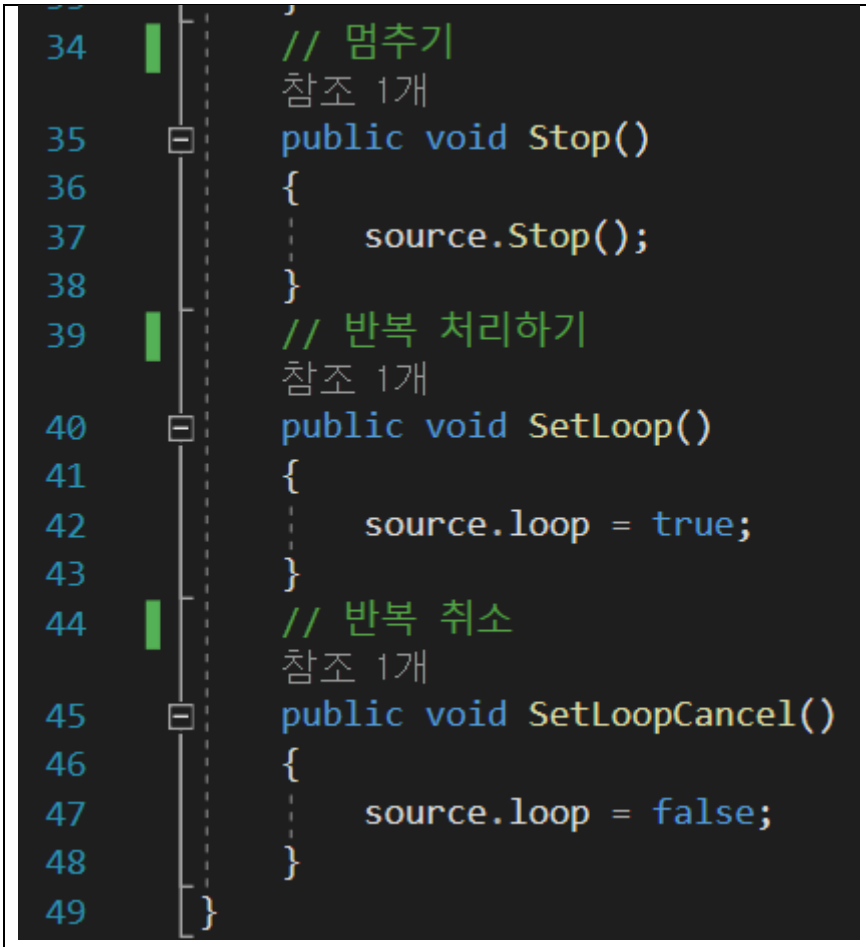
사운드 클래스

이 자체를 serializeable 화 (직렬화 함)

```
5       [System.Serializable]
        참조 1개
6       public class Sound
7       {
8           public string name; // 사운드의 이름.
9
10          public AudioClip clip; // 사운드 파일 (파일은 카세트 테이프)
11          private AudioSource source; // 사운드 플레이어 (오디오)
12
13          public float Volumn; // 볼륨 소리
14          public bool loop; // loop 시킬지 말지
15
16          // 위의 변수에 대한 초기화 함수들
            참조 1개
```



```
16      // 위의 변수에 대한 초기화 함수들
        참조 1개
17      public void SetSource(AudioSource _source)
18      {
19          source = _source;
20          source.clip = clip;
21          source.loop = loop;
22          source.volume = Volumn;
23      }
24      // 실제 볼륨값을 변수 볼륨으로 바꿈
        참조 1개
25      public void SetVolumn()
26      {
27          source.volume = Volumn;
28      }
29      // 오디오 플레이
        참조 1개
30      public void Play()
31      {
32          source.Play();
33      }
```

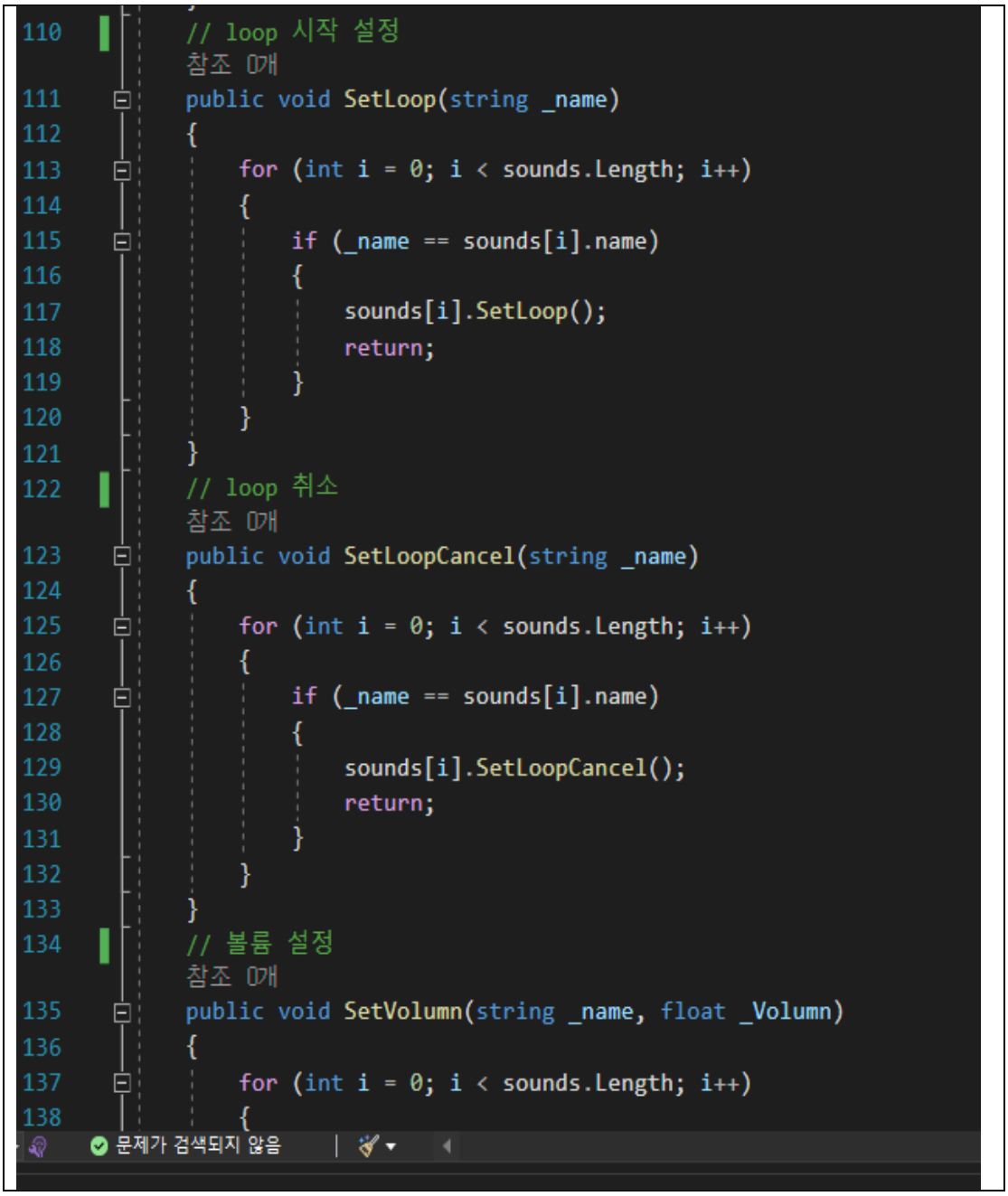


```
34      // 멈추기
        참조 1개
35      public void Stop()
36      {
37          source.Stop();
38      }
39      // 반복 처리하기
        참조 1개
40      public void SetLoop()
41      {
42          source.loop = true;
43      }
44      // 반복 취소
        참조 1개
45      public void SetLoopCancel()
46      {
47          source.loop = false;
48      }
49  }
```

```
51  // 실제 메니저
    Unity 스크립트(자산 참조 1개)|참조 3개
52  public class AudioManager : MonoBehaviour
53  {
54      // 매니저 클래스이므로 싱글톤 처리를 위한 변수
55      // 싱글톤이란 이 해당 클래스가 전체 게임에서 1개만 잇게 하기 위함
56      static public AudioManager instance;
57
58      [SerializeField]
59      public Sound[] sounds; //클립 여러개
60
        Unity 메시지|참조 0개
61      private void Awake() // 싱글톤 처리
62      {
63          if (instance != null)
64          {
65              Destroy(this.gameObject);
66          }
67          else
68          {
69              DontDestroyOnLoad(this.gameObject);
70              instance = this;
71          }
72      }
73
```

```
74      // 초기화
        Unity 메시지|참조 0개
75      void Start()
76      {
77          // 사운드파일의 이름을 위에서 만든 사운드 관련 컴포넌트처럼 있는 것에서 미리 넣어놓음
78          // 그거를 반복해서 돌리면서 실제 audioSource로 넣음
79          for (int i = 0; i < sounds.Length; i++)
80          {
81              GameObject soundObject = new GameObject("사운드 파일 이름 : " + i + " = " + sounds[i].name);
82              sounds[i].SetSource(soundObject.AddComponent<AudioSource>());
83              soundObject.transform.SetParent(this.transform);
84          }
85      }
```

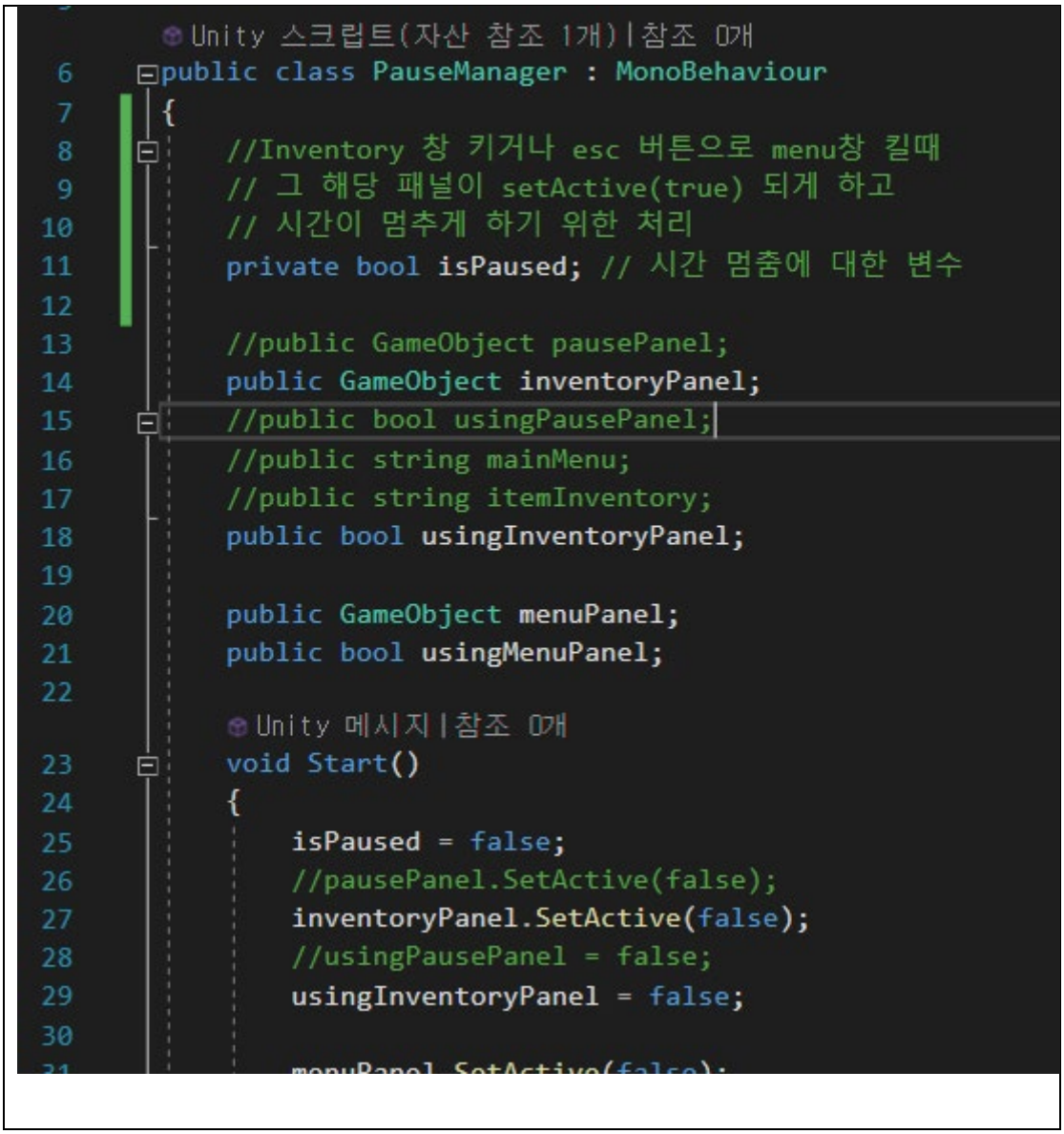
```
86      // 위에서 반복한 것을 이름을 체크해서 list로 돌리면서 p
        참조 2개
87      public void Play(string _name)
88      {
89          for (int i = 0; i < sounds.Length; i++)
90          {
91              if (_name == sounds[i].name)
92              {
93                  sounds[i].Play();
94                  return;
95              }
96          }
97      }
98      // 멈추는 것
        참조 0개
99      public void Stop(string _name)
100     {
101         for (int i = 0; i < sounds.Length; i++)
102         {
103             if (_name == sounds[i].name)
104             {
105                 sounds[i].Stop();
106                 return;
107             }
108         }
109     }
```



```
110     // loop 시작 설정
        참조 0개
111     public void SetLoop(string _name)
112     {
113         for (int i = 0; i < sounds.Length; i++)
114         {
115             if (_name == sounds[i].name)
116             {
117                 sounds[i].SetLoop();
118                 return;
119             }
120         }
121     }
122     // loop 취소
        참조 0개
123     public void SetLoopCancel(string _name)
124     {
125         for (int i = 0; i < sounds.Length; i++)
126         {
127             if (_name == sounds[i].name)
128             {
129                 sounds[i].SetLoopCancel();
130                 return;
131             }
132         }
133     }
134     // 볼륨 설정
        참조 0개
135     public void SetVolumn(string _name, float _Volumn)
136     {
137         for (int i = 0; i < sounds.Length; i++)
138         {
```

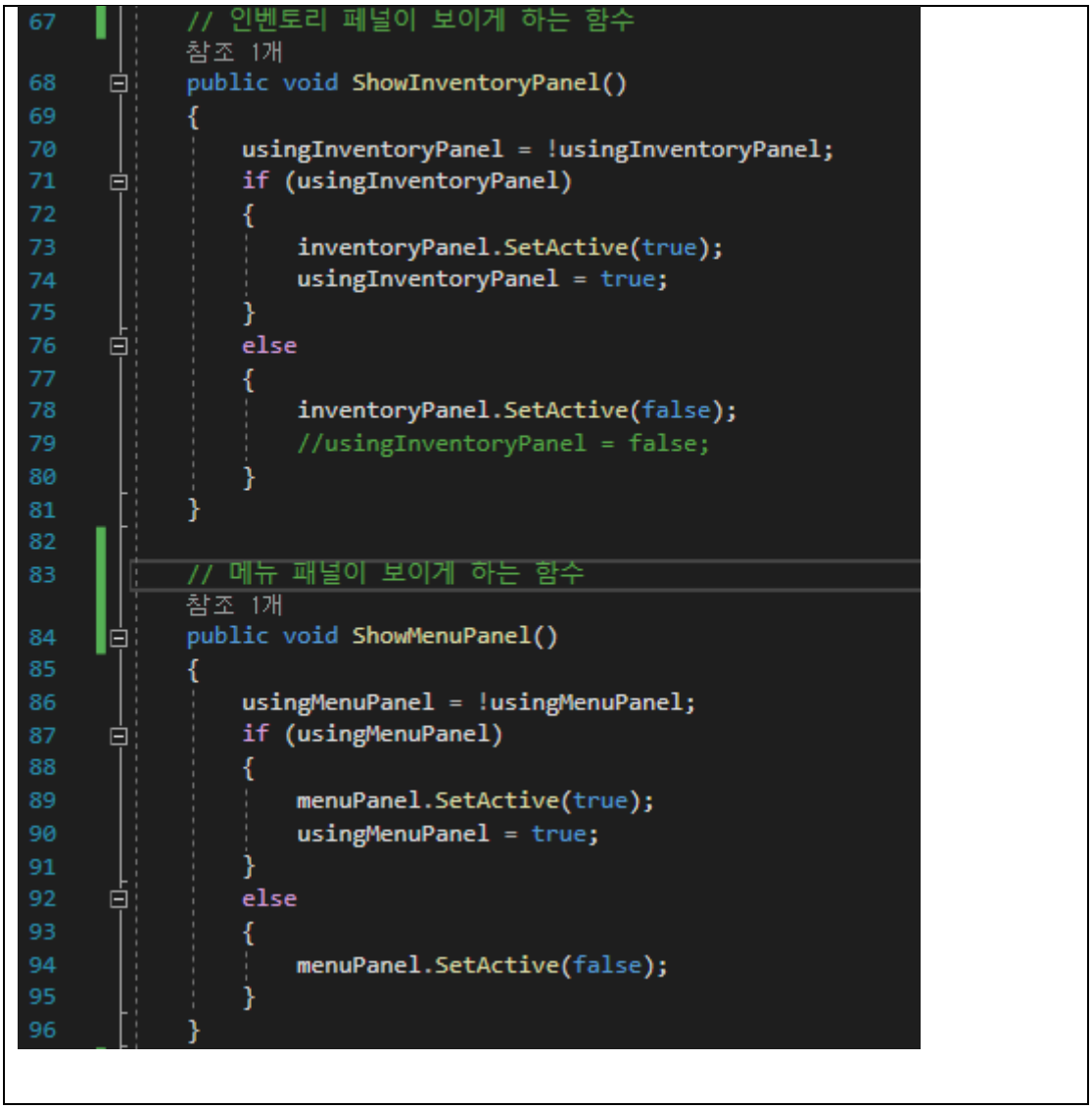
문제가 검색되지 않음

- PauseManager



```
Unity 스크립트(자산 참조 1개)|참조 0개
public class PauseManager : MonoBehaviour
{
    //Inventory 창 키거나 esc 버튼으로 menu창 킬때
    // 그 해당 패널이 setActive(true) 되게 하고
    // 시간이 멈추게 하기 위한 처리
    private bool isPaused; // 시간 멈춤에 대한 변수

    //public GameObject pausePanel;
    public GameObject inventoryPanel;
    //public bool usingPausePanel;
    //public string mainMenu;
    //public string itemInventory;
    public bool usingInventoryPanel;

    public GameObject menuPanel;
    public bool usingMenuPanel;

    Unity 메시지|참조 0개
    void Start()
    {
        isPaused = false;
        //pausePanel.SetActive(false);
        inventoryPanel.SetActive(false);
        //usingPausePanel = false;
        usingInventoryPanel = false;

        menuPanel.SetActive(false);
```

```
36      // Z키 누르면 시간이 멈추고 인벤토리 창이 켜짐
        Unity 메시지 | 참조 0개
37      void Update()
38      {
39          if (Input.GetKeyDown(KeyCode.Z))
40          {
41              ChangePause();
42              ShowInventoryPanel();
43          }
44
45          if (Input.GetKeyDown(KeyCode.Escape))
46          {
47              ChangePause();
48              ShowMenuPanel();
49          }
50      }
51      // 게임 전체의 시간을 멈추고 풀고 하게 하는 함수
        참조 2개
52      public void ChangePause()
53      {
54          isPaused = !isPaused;
55          if (isPaused)
56          {
57              //pausePanel.SetActive(true);
58              Time.timeScale = 0f;
59              //usingPausePanel = true;
60          }
61          else
62          {
63              //pausePanel.SetActive(false);
64              Time.timeScale = 1f;
65          }
66      }
```



```
67      // 인벤토리 패널이 보이게 하는 함수
        참조 1개
68      public void ShowInventoryPanel()
69      {
70          usingInventoryPanel = !usingInventoryPanel;
71          if (usingInventoryPanel)
72          {
73              inventoryPanel.SetActive(true);
74              usingInventoryPanel = true;
75          }
76          else
77          {
78              inventoryPanel.SetActive(false);
79              //usingInventoryPanel = false;
80          }
81      }
82
83      // 메뉴 패널이 보이게 하는 함수
        참조 1개
84      public void ShowMenuPanel()
85      {
86          usingMenuPanel = !usingMenuPanel;
87          if (usingMenuPanel)
88          {
89              menuPanel.SetActive(true);
90              usingMenuPanel = true;
91          }
92          else
93          {
94              menuPanel.SetActive(false);
95          }
96      }
```

- Title

```
Unity 스크립트(자산 참조 1개)|참조 0개
public class Title : MonoBehaviour
{
    // AudioManager를 이용해 click_sound 효과 내기
    private AudioManager theAudio;
    public string click_sound;
    // 메인도 씬 전환시 Fade 효과를 내야 자연스럽고
    // 로딩되는 것의 코루틴 처리도 할 수 있음
    public GameObject fadeInPanel;
    public GameObject fadeOutPanel;
    public float fadeWait;
    // 게임 키 동작에 대한 설명을 위한 패널
    public GameObject explainPanel;

    Unity 메시지|참조 0개
    private void Start()
    {
        theAudio = FindObjectOfType<AudioManager>();
        explainPanel.SetActive(false);
    }
```

```
    // 게임 시작시 첫 시작 되는 씬을 부름
    참조 0개
    public void PlayGame()
    {
        StartCoroutine(PlayGameCoroutine());
    }
    // 코루틴 처리
    참조 1개
    IEnumerator PlayGameCoroutine()
    {
        theAudio.Play(click_sound);
        yield return new WaitForSeconds(2f);
        //SceneManager.LoadScene("StartScene");
        StartCoroutine(FadeCo());
    }
    // 설명 버튼 클릭시 exPlainPanel 활성화 및 비활성화
    참조 0개
    public void explainButtonClicked()
    {
        explainPanel.SetActive(true);
    }
    참조 0개
    public void explainCloseClicked()
    {
        explainPanel.SetActive(false);
    }
    // 게임 종료
    참조 0개
    public void ExitGame()
    {
        theAudio.Play(click_sound);
        Application.Quit();
    }
```

- GameoverManager

: 윗부분은 start와 변수이므로 생략

```
    // 게임 오버 는 플레이어의 hp를 이용해서 gameOver이란 bool 변수를
    //변경해서 판단함
    // 게임 오버 여부에 따라서 gameOverPanel 활성화 및 비활성화
    Unity 메시지 | 참조 0개
    void Update()
    {
        if (player.GetComponent<PlayerMovement>().gameOver)
        {
            gameoverPanel.SetActive(true);
        }
        else
        {
            gameoverPanel.SetActive(false);
        }
    }
    // restart 버튼 클릭시 메인 게임씬(성안)
    // 부터 다시 시작하게 함
    참조 0개
    public void restartBtnClick()
    {
        SceneManager.LoadScene(SceneName);
        player.GetComponent<PlayerMovement>().currentHealth.RuntimeValue =
player.GetComponent<PlayerMovement>().currentHealth.initialValue;
    }
}
```

- GameManager

: 성배에 대한 개수를 체크하기 위한 역할만 하고 있는 구더기 코드인데 위쪽의 GameOverManager와 합쳐져서 일해야하는 스크립트임. *수정이 매우 많이 필요한 스크립트* 그래서 현재는 주석 처리로 막아놓은 것이 많음.

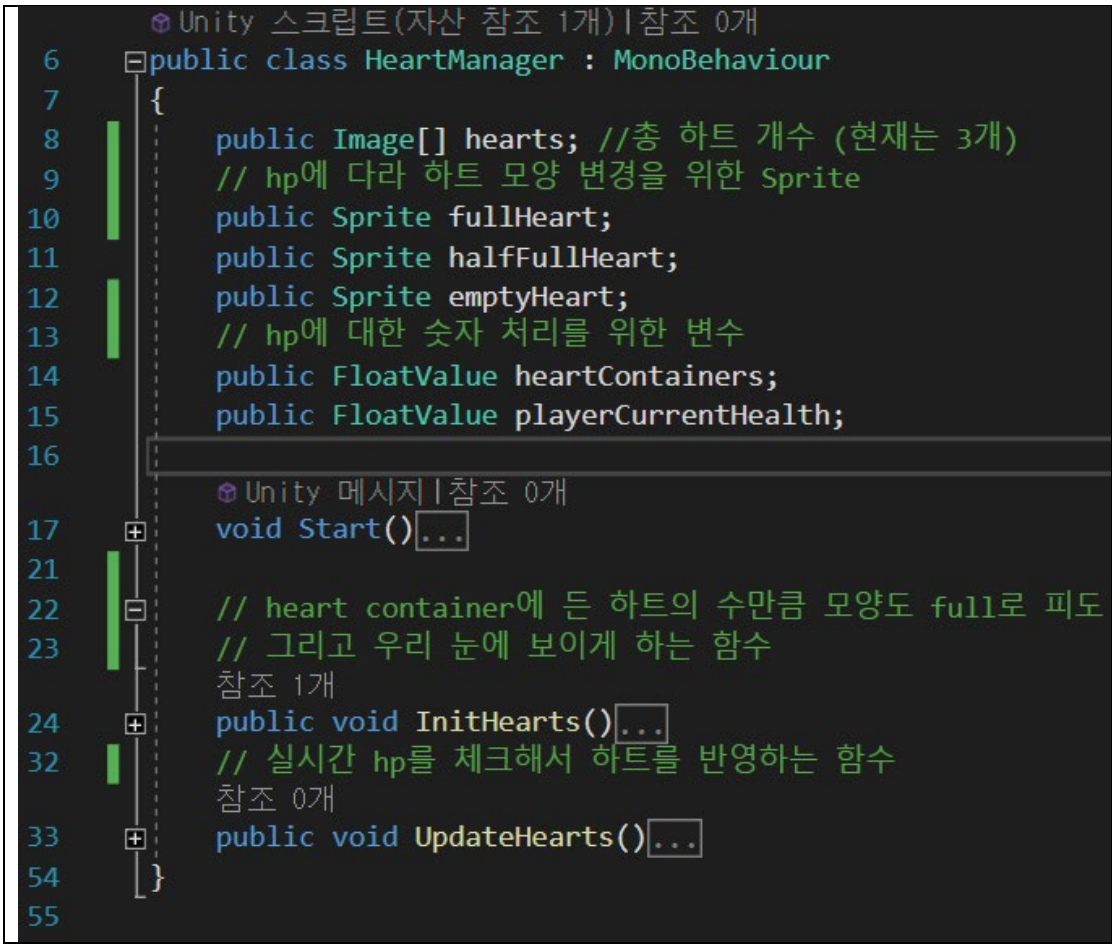
```
Unity 스크립트(자산 참조 2개)|참조 4개
public class GameManager : MonoBehaviour
{
    public int chalices = 0;
    //private FadeManager theFade;

    Unity 메시지|참조 0개
    private void OnEnable()
    {
        //theFade = FindObjectOfType<FadeManager>();
    }


    참조 0개
    public void LoadStart(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    참조 0개
    IEnumerator LoadWaitCoroutine()
    {
        yield return new WaitForSeconds(0.5f);
        //theFade = FindObjectOfType<FadeManager>();

        //theFade.FadeIn();
    }

    참조 0개
    public void Exit()
    {
        Application.Quit();
    }
}
```

- CameraMovement

: RoomMove와 함께 카메라에 대한 처리를 위해 Camera가 캐릭터를 따라가게 하기 위한 것이었는데 씨네머신 처리로 인해 사실상 필요 없는 클래스 (너무 길어졌고, 사용하지 않으므로 생략합니다.....)

```
// Pass
```

- HeartManager



```
Unity 스크립트(자산 참조 1개)|참조 0개
public class HeartManager : MonoBehaviour
{
    public Image[] hearts; //총 하트 개수 (현재는 3개)
    // hp에 다라 하트 모양 변경을 위한 Sprite
    public Sprite fullHeart;
    public Sprite halfFullHeart;
    public Sprite emptyHeart;
    // hp에 대한 숫자 처리를 위한 변수
    public FloatValue heartContainers;
    public FloatValue playerCurrentHealth;

    Unity 메시지|참조 0개
    void Start()...

    // heart container에 든 하트의 수만큼 모양도 full로 피도
    // 그리고 우리 눈에 보이게 하는 함수
    참조 1개
    public void InitHearts()...
    // 실시간 hp를 체크해서 하트를 반영하는 함수
    참조 0개
    public void UpdateHearts()...
}
```

6) **Object 관련 스크립트**

- Interactable

```
5   public class Interactable : MonoBehaviour
6   {
7       public Signal context; //palyer에 달려있는 컨텍스트 클루 setAci
8       public bool playerInRange; // player와의 범위 체크
9
10      // 가까워 질때 컨텍스트 클루 보이도록
            Unity 메시지 | 참조 0개
11      private void OnTriggerEnter2D(Collider2D other)
12      {
13          if (other.CompareTag("Player") && !other.isTrigger)
14          {
15              context.Raise();
16              playerInRange = true;
17          }
18      }
19      // 나갈때 아이템을 받을 수 없고 컨텍스트 없애기
            Unity 메시지 | 참조 0개
20      private void OnTriggerExit2D(Collider2D other)
21      {
22          if (other.CompareTag("Player") && !other.isTrigger)
23          {
24              context.Raise();
25              playerInRange = false;
26          }
27      }
28  }
```

- Heart

```
        Unity 스크립트(자산 참조 1개) | 참조 0개
5   public class Heart : Powerup
6   {
7       // Powerup이란 스크립트를 상속 받아서
8       // heart를 가까워지게 하면 powerup을 이용함
9       // (Powerup은 피 뿐만아니라 여러 가지를 증가시킬 수 있게 변형 가
10      public FloatValue playerHealth;
11      public FloatValue heartContainers;
12      public float amountToIncrease;
13
14      // 하트와 가까워지고 그것이 Player라면 별도의 처리 없이
15      // 하트를 get하게 되고 hp가 증가되고 그 모양에 대한 update도 이
            Unity 메시지 | 참조 0개
16      public void OnTriggerEnter2D(Collider2D other)...
29  }
30
```

- Sign

```
Unity 스크립트(자산 참조 6개)|참조 0개
6   public class Sign : Interactable
7   {
8       // 이미있는 다이알로그 Panel을 두기
9       public GameObject dialogBox;
10      public Text dialogText;
11      public string dialog;
12
        Unity 메시지|참조 0개
13      void Update()
14      {
15          // Sign에서 space bar 누르면, 다얄로그 활성화
16          // Sign에 등록된 string 변수를 그 내용으로 함
17          if (Input.GetKeyDown(KeyCode.Space) && playerInRang
29      }
30      // 멀어지면 비활성화
        Unity 메시지|참조 0개
31      private void OnTriggerExit2D(Collider2D other)...
39
40  }
41
```

- Pot

```
14      //smash 시 깨지는 애니매이션 효과
        참조 2개
15      public void Smash()
16      {
17          anim.SetBool("smash", true);
18          StartCoroutine(breakCo());
19      }
20      // 깨지면서 더이상 보이지 않게 처리
        참조 1개
21      IEnumerator breakCo()
22      {
23          yield return new WaitForSeconds(0.3f);
24          this.gameObject.SetActive(false);
25      }
26  }
```

- Powerup

: Signal에 대한 정보만 받고 있고 이를 상속 받아서 PowerUp 되게 하는 Item을 처리할 수 있게 했기 때문에 변수 하나만 들고 있는 상속용 스크립

트 클래스

```
public class Powerup : MonoBehaviour
{
    public Signal powerupSignal;

}
```

7) **ScriptableOjbects 관련 스크립트**

*** https://everyday-devup.tistory.com/88 유니티의 다양한 직렬화에 대한 설명 블로그 링크 (국내용 블로그로 택하여 실제 참고와는 조금 다름)**

*** 가장 핵심은 ISerializationCallbackReceiver는 class가 아닌 Interface로 다중 상속첨 사용할 수 있게 하고 이를 통해 코드 작성자가 직렬화가 가능한 클래스를 만들 수 있다는 것이 포인트**

- FloatValue

```
[CreateAssetMenu]
Unity 스크립트 | 참조 6개
public class FloatValue : ScriptableObject, ISerializationCallbackR
{
    // 인스펙터 창에 보이게하고
    // 이것으로 플레이어의 좌표 등을 설정
    public float initialValue;

    // runTime에서 돌아가는 값은 숨김
    [HideInInspector]
    public float RuntimeValue;

    // ISerializationCallbackReciver를 상속받았기에
    // 사용하지 않는 OnBeforeSerialize()도 일단 함수로 있어야함
    참조 0개
    public void OnAfterDeserialize() {
        RuntimeValue = initialValue;
    }
    참조 0개
    public void OnBeforeSerialize() { }

}
```

- Signal

```
// 스크립트, 폴더 등 생성 시 마우스 우클릭으로 생성하는 것처럼
// Asset Menu에 마치 유니티에 기본적으로 있는 것처럼 create하는 것
[CreateAssetMenu]
Unity 스크립트 | 참조 4개
public class Signal : ScriptableObject
{
    // Signal에 대한 정보를 listen할 SignalListener List 생성
    public List<SignalListener> listeners = new List<SignalLister

    //받아놓은 Signal들을 발생(Raise) 시키도록 하는 함수
    참조 4개
    public void Raise()
    {
        for(int i = listeners.Count - 1; i >= 0; i--)
        {
            listeners[i].OnSignalRaised();
        }
    }

    // List에 Signal을 받을 SignalListener 추가
    참조 1개
    public void RegisterListener(SignalListener listener)
    {
        listeners.Add(listener);
    }
    // 삭제
    참조 1개
    public void DeRegisterListener(SignalListener listener)
    {
        listeners.Remove(listener);
    }
}
```

- VectorValue

: FloatValue와 마찬가지이므로 생략

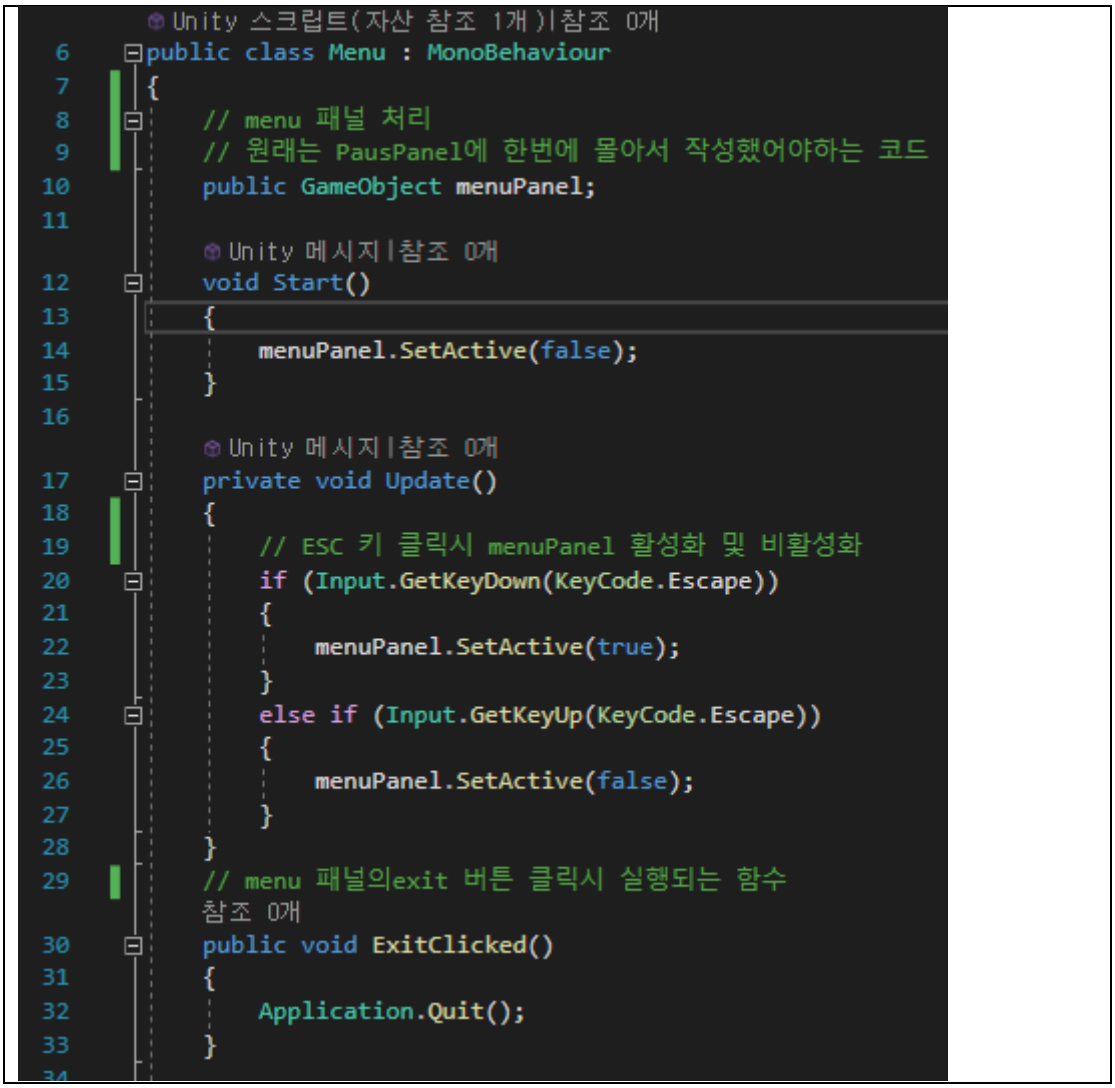
Pass (그 값이 Float이 아니고 Vector일 뿐임)

- SignalListener

```
public class SignalListener : MonoBehaviour
{
    //SignalListener가 받을 signal
    public Signal signal;
    public UnityEvent signalEvent; //Unity 내장 Event관련 Action을 우

    // Invoke 자극하다 실행하게 하는 함수
    참조 1개
    public void OnSignalRaised()
    {
        signalEvent.Invoke();
    }

    // signal 스크립트에 만든 RegisterListener을 이용해
    // 등록
    Unity 메시지 | 참조 0개
    private void OnEnable()
    {
        signal.RegisterListener(this);
    }
    //등록 해제
    Unity 메시지 | 참조 0개
    private void OnDisable()
    {
        signal.DeRegisterListener(this);
    }
}
```

8) **기타 (엔딩크레딧, Menu 등)**

- Menu



```
Unity 스크립트(자산 참조 1개)|참조 0개
public class Menu : MonoBehaviour
{
    // menu 패널 처리
    // 원래는 PausPanel에 한번에 몰아서 작성했어야하는 코드
    public GameObject menuPanel;

    Unity 메시지|참조 0개
    void Start()
    {
        menuPanel.SetActive(false);
    }

    Unity 메시지|참조 0개
    private void Update()
    {
        // ESC 키 클릭시 menuPanel 활성화 및 비활성화
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            menuPanel.SetActive(true);
        }
        else if (Input.GetKeyUp(KeyCode.Escape))
        {
            menuPanel.SetActive(false);
        }
    }
    // menu 패널의exit 버튼 클릭시 실행되는 함수
    참조 0개
    public void ExitClicked()
    {
        Application.Quit();
    }
```

- Ending

```
5   public class Ending : MonoBehaviour
6   {
7       public GameObject go;
8
        Unity 메시지 | 참조 0개
9       void Start()
10      {
11          go.SetActive(false);
12      }
13
14      // palyer가 들어와서 머물고 있는 상태이면
15      // go라는 ending 크레딧으로 받은 것을 보이게함
16      // 기본적으로 애니매이션 효과를 인스펙터를 통해 넣었으므로
17      // active 되면 크레딧 실행됨
        Unity 메시지 | 참조 0개
18      private void OnTriggerStay2D(Collider2D collision)
19      {
20
21          go.SetActive(true);
22      }
23  }
24
```

5. 프로젝트 작업 시 느낀 점

유니티 강좌를 통해 따라 만드는 것 말고, 직접 프로젝트를 기획부터 구현까지 하는 것은 '내가 얼만큼 구현할 수 있는 지'를 확신할 수 없어서 어려운 일이었다. 그래도 학기 초부터 여러 유튜브와 책들을 찾아보며 기본기를 다진 것을 나의 프로젝트를 위해 수행함으로써 배운 것을 직접적으로 익힐 수 있는 좋은 경험이었다. 또한 대부분의 RPG 게임은 지속적으로 Player와 상호작용할 수 있는 Objects를 업데이트 하는 경우가 많은데, 왜 오랜 시간이 걸리는 지 알 수 있었다. RPG 게임의 경우 Objects와의 접촉 및 접촉 후 Player에게 어떤 상호작용을 할 지 등에 대한 처리가 정말 작은 것이어도 세세하게 많이 필요했다.

초등학생 시절 컴퓨터실에서 몰래한 '병든 공 튀기기 게임'같은 것이 얼마나 아이디어 측면에서 단순하고도 재미있는 지 알 게 되었다. 그리고 구현도 비교적 간단한 게임인지 다시 한번 느끼게 되었다. 다른 학생들의 프로젝트를 시청하면서 다양한 아이디어에 자극받았고, 간단하고 깔끔한 게임은 무엇인 지 생각해보는

계기가 되었다.

2D게임은 3D게임에 비해 몰입도가 적고 RPG게임은 완성이 없는 게임이나 마찬가지여서 어느 정도의 규모가 있어야 완성도 있는 게임처럼 느껴진다. 그래서 단순히 학교 과제를 넘어서 개인 프로젝트로 진행해볼 예정이다. 특히 급한 제출을 위해서 동작만 가능한 코드들(일명 구더기 코드)을 급하게 작성하였다. Enemy에 대한 처리를 Objects Pooling 기법 처리로 바꾸고 destroy 대신 setactive(false)를 활용해 runtime에 있는 메모리를 적게 소모할 수 있게 하면서 빠르고 안정감있는 게임이 되도록 해야겠다.