

# 우주의 마음

김홍규

슬라이드 쇼(ppt의 경우) 혹은 전체화면(pdf의 경우)로 시청해주세요.

---

# Outline

자기소개

게임소개

코드리뷰

부록

# 자기소개

**게임만들기가 취미였던 중학생**

RPG Maker XP로 제작. (부록 참고)

**연세대학교 학사 졸업**

컴퓨터과학과, 철학과 복수전공

March 2011 - Present

**모바일 게임 『우주의마음』 출시**

Unity로 제작, 구글 플레이스토어 출시

June 2018 - November 2018

저는 코딩의 “고독”함과  
동료와의 “인연”을 모두 즐기는 사람입니다.

- 견<sup>고</sup>성 : 예외를 예방하는 코드

저는 코딩의 “독”함과  
동료와의 “인연”을 모두 즐기는 사람입니다.

- 견<sup>고</sup>성 : 예외를 예방하는 코드

- 독<sup>립</sup>성 : 코드간의 결합도가 낮은 코드

저는 코딩의 “ ”함과  
동료와의 “인연”을 모두 즐기는 사람입니다.

- 견<sup>고</sup>성 : 예외를 예방하는 코드

- 독<sup>립</sup>성 : 코드간의 결합도가 낮은 코드

저는 코딩의 “ ”함과  
동료와의 “ 연”을 모두 즐기는 사람입니다.

- 클<sup>린</sup>성 : 동료가 이해하기 쉬운 코드

- 견<sup>고</sup>성 : 예외를 예방하는 코드

- 독<sup>립</sup>성 : 코드간의 결합도가 낮은 코드

저는 코딩의 “ ”함과  
동료와의 “ ”을 모두 즐기는 사람입니다.

- 클<sup>린</sup>성 : 동료가 이해하기 쉬운 코드

- 유<sup>연</sup>성 : 변화에 대처하기 쉬운 코드



- 견<sup>고</sup>성 : 예외를 예방하는 코드

- 독<sup>립</sup>성 : 코드간의 결합도가 낮은 코드

4가지를 중시하는 프로그래머이기도 합니다.

- 클<sup>린</sup>성 : 동료가 이해하기 쉬운 코드

- 유<sup>연</sup>성 : 변화에 대처하기 쉬운 코드

# 게임소개

## 우주의 마음

- 제작인원 : 1인
- 게임장르 : 캐주얼
- 제작기간 : 6개월 (6월 2018 - 11월 2018)
- 사용엔진 : Unity 3D
- 사용언어 : C#
- 출시여부 : 유 ([구글플레이 스토어 링크](#))
- 목표플랫폼 : 안드로이드 모바일
- 총 다운로드수 : 2,000건

# 게임 소개 영상

(60초 영상)



## 채박사

[ 그 순간 제 마음속에 어지러움을 한번에 정리해준 목소리가 그 때 들렸던 거죠! ]

스킨

상단 그림을 클릭하면 Youtube 링크로 이동합니다.

영상이 재생되지 않는다면, <https://blog.naver.com/tommovie/221536483344> 링크를 이용해주세요.

# 코드리뷰

## 빙글머이 게임으로 코드 리뷰

- 게임소개 영상
- 코드리뷰 : 필드아이템인 별 구현
- 코드리뷰 : 게임시스템 구현

**빙글멍이 게임 소개 영상** (90초 영상)

www.BANDICAM.COM

# 우주의 마음

우주의 마음

빙글머이 게임소개

마음보기

자러가기

상단 그림을 클릭하면 Youtube 링크로 이동합니다.

영상이 재생되지 않는다면, <https://blog.naver.com/tommovie/221536487198> 링크를 이용해주세요.

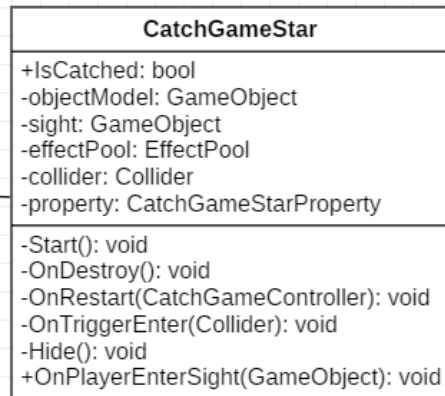
# 코드 리뷰

필드아이템인 별 구현

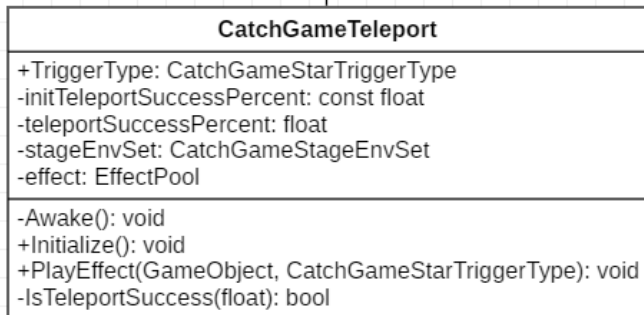


# UML

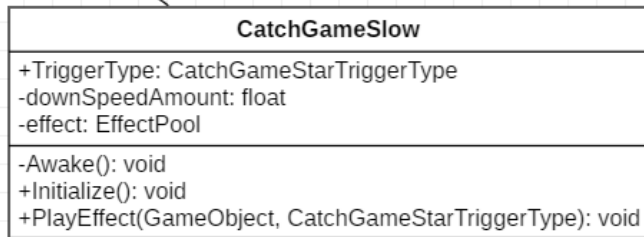
// 별에 속성(특수능력)을 부여한다.



// 멤버변수 property의 값에 따라  
별의 속성(특수능력)이 결정된다.



//플레이어가 근처에 다가가면  
특정확률로 순간이동



//플레이어가 별을 습득하면 이동속도가 느려짐

# 상세 코드

```
using System.Collections.Generic;
using UnityEngine;

// [유연성] 전략패턴 (42)
[RequireComponent(typeof(CatchGameStarProperty))]
public class CatchGameStar : MonoBehaviour {

    [Header("Child Component")]
    [SerializeField]
    GameObject _objectModel; // 오브젝트 모델 (캐릭터의 (배우)물, 물체(배우의 입체)에서 어떤 오브젝트 모델로 보일지)
    [SerializeField]
    GameObject _light; // 리어 오브젝트 (오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트)
    [SerializeField]
    AudioSource _audio; // 음향 오브젝트 (오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트)
    [SerializeField]
    float _radius; // 오브젝트의 반지름 (오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트)

    public void OnCollisionEnter(Collision collision) { // 오브젝트와 충돌할 때 호출되는 이벤트
        Collider _collider; // 오브젝트의 collider (오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트)
    }

    // [유연성] 전략패턴 (42)
    CatchGameStarProperty _property; // 오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트

    // [유연성] 전략패턴 (42)
    // 오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트
    // 오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트
    void Start() {
        _collider = GetComponent<Collider>();
        _property = GetComponent<CatchGameStarProperty>();
        CatchGameStarPublisher instance RestartGame -> OnRestart; // 오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트
    }

    // [유연성] 전략패턴 (42)
    // 오브젝트, 물체(배우의 입체)를 통해 사용자에게 보여지게 할 오브젝트와 함께 표시할 오브젝트
    void OnRestart() {
        CatchGameStarPublisher instance RestartGame -> OnRestart;
    }
}
```

## UML에서 현재 리뷰중인 부분 (빨간테두리로 표시)

### [유연성] 전략패턴 (42)

- property 멤버변수에 어떤 값을 할당해주느냐에 따라 다른 특수능력을 가진 별이 된다.

- 새로운 특수능력을 가진 별을 만들고 싶다면...

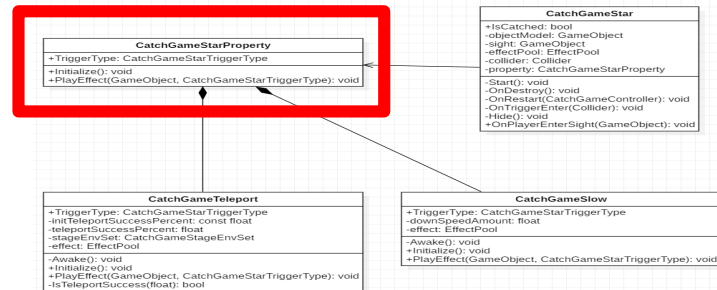
## 코드리뷰

- (1) CatchGameStarProperty 인터페이스를 구성하는 클래스를 만들고
- (2) 오브젝트에 새로 만든 클래스 컴포넌트를 추가해주면 된다.
- 중요한 점은 이 과정에서 기존의 코드를 수정해야하는 작업이 필요없기 때문에 변화에 유연하게 대처할 수 있다.

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.AI;
5
6 /// 필드아이템인 별에 특수능력이 언제 발동될지를 명시
7 public enum CatchGameStarTriggerType { OnEnter = 0, OnSight, NumOfTypes, };
8
9 /* [독립성] 구현보다는 구성 ...
10
11 <summary>
12 /// 필드아이템인 별에 속성(특수능력)을 부여해 주기 위한 클래스를 만들기 위해선,
13 /// 반드시 이 인터페이스를 상속받아야한다.
14 /// </summary>
15
16 public interface CatchGameStarProperty {
17
18     CatchGameStarTriggerType TriggerType { get; }
19     void Initialize();
20     void PlayEffect(GameObject target, CatchGameStarTriggerType type);
21 }

```



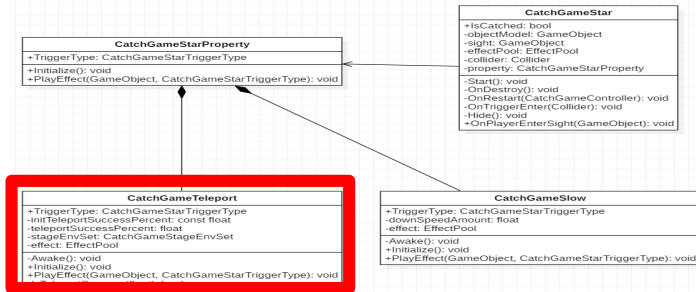
## [독립성] 구현보다는 구성 (21)

- 부모클래스를 상속하는 것보다 인터페이스를 통해 구성한다.
- 상속은 결합도를 높이기 때문이다.
- 그래서 책 “HeadFirst Design Pattern”에서 구현보다는 구성을 이용하라는 것을 제1원칙으로 제시한다.
- 또한 상속은 엄밀한 의미에서 캡슐화를 위반하기 때문이다.
- 그래서 책 “Effective C++”에서 protected 변수 선언 금지원칙을 제시한다.

```

6  /// <summary>
7  /// 플레이어가 근처에 다가가면 특정확률로 순간이동하는 특수능력을 필드아이템인 별에 부여한다.
8  /// </summary>
9  public class CatchGameTeleport : MonoBehaviour, CatchGameStarProperty {
10
11  [SerializeField]
12  EffectPool _effect;           /// 특수능력(순간이동) 발동시 나타나는 이펙트 (오브젝트 풀링을 이용)
13  /* 오브젝트 풀링 ...
14  public CatchGameStarTriggerType TriggerType { get; private set; }   /// 별의 특수능력이 발동되는 순간
15  const float _initTeleportSuccessPercent = 70.0f;                 /// 순간이동 초기성공확률 (순간이동할수록 성공확률이 낮아진다.)
16  float _teleportSuccessPercent;                                     /// 현재 순간이동 성공확률
17  CatchGameStageEnvSet _stageEnvSet;                                /// 현재 맵 정보
18
19  /// <summary>
20  /// 게임시작시, 단 한번만 초기화해줘도 되는 부분을 이 함수에서 처리한다.
21  /// </summary>
22  void Awake()
23  {
24      TriggerType = CatchGameStarTriggerType.OnSight; /// 별의 시야에 플레이어가 들어오면, 특수능력(순간이동)이 발동된다.
25      _teleportSuccessPercent = _initTeleportSuccessPercent;
26      _stageEnvSet = GameObject.FindWithTag("StageEnv").GetComponent<CatchGameStageEnvSet>;
27  }
28
29  /// <summary>
30  /// 플레이어가 게임에 실패하여 재도전할때마다, 초기화해줘야 하는 부분을 이 함수에서 처리한다.
31  /// </summary>
32  public void Initialize()
33  {
34      _teleportSuccessPercent = _initTeleportSuccessPercent;
35  }
36
37  /* [독립성] 두번째 매개변수로 CatchGameStarTriggerType을 둔 이유 ...
38  /// <summary>
39  /// 특수능력(순간이동)을 발동처리하는 함수이다.
40  /// </summary>
41  /// <param name="target"> 특수능력을 적용시키는 대상 </param>
42  /// <param name="type"> 유효성 검사 </param>
43  public void PlayEffect(GameObject target, CatchGameStarTriggerType type)
44  {
45      if (type != TriggerType) return;
46      if (!IsTeleportSuccess(_teleportSuccessPercent))
47      {
48          Vector3 posBeforeTeleport = transform.position;
49          transform.position = _stageEnvSet.GetRandomPointOnGenGround(); /// 오브젝트가 켄될 수 있는 위치를 랜덤으로 반환한다.
50          _effect.ShowEffect(posBeforeTeleport + Vector3.up * 2.0f);
51          AudioManager.instance.PlaySoundEffect("Teleport");
52      }
53      _teleportSuccessPercent -= 20.0f;
54  }
55  }

```



## 오브젝트 풀링 (11)

- 최대한 오브젝트를 생성하는 작업(=비싼작업)을 게임중간에 하지 않도록 한다.

## [독립성] 유효성 검사 (52)

- 최대한 함수 내부에서 유효성 검사
- 그러면 함수를 호출할 때 별도의 유효성 검사를 하지 않아도 된다.
- 이는 이 함수와 함수를 호출하는 객체(caller)사이의 결합도를 느슨하게 해준다.

```
63 /* [클린성] 함수화 ...
```

```
67 /* [유연성] 재활용 ...
```

```
72 /// <summary>
```

```
73 /// 순간이동 성공여부를 반환한다.
```

```
74 /// </summary>
```

```
75 /// <param name="teleportSuccessPercent">순간이동 성공확률</param>
```

```
76 /// <returns>순간이동 성공여부</returns>
```

```
77 bool IsTeleportSuccess(float teleportSuccessPercent)
```

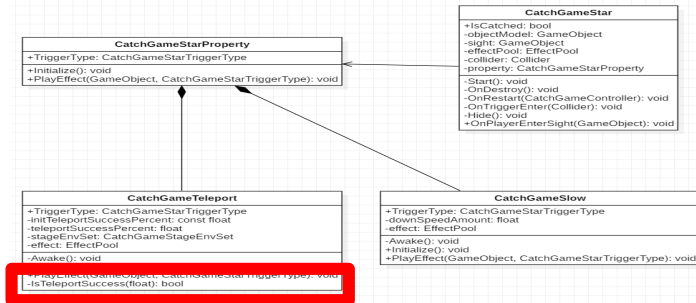
```
78 {
```

```
79     float teleportSuccessCriteria = Random.Range(0.0f, 100.0f);
```

```
80     return teleportSuccessCriteria < _teleportSuccessPercent;
```

```
81 }
```

```
82 }
```



## [클린성] 함수화 (77)

- 함수화는 가독성을 향상시켜준다.
- `if(Random.Range(0.0f, 100.0f) < _teleportSuccessPercent)`
- `if(IsTeleportSuccess(_teleportSuccessPercent))`
- 함수화한 후자의 코드가 훨씬 가독성이 좋을 수 있다.

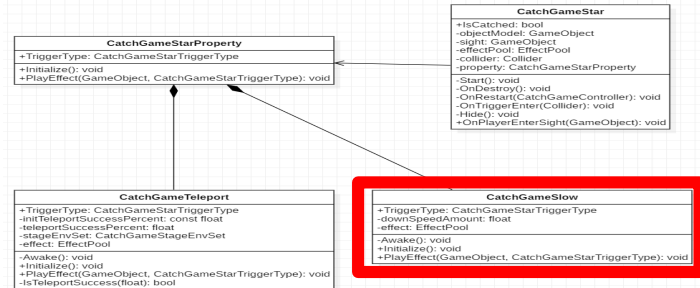
## [유연성] 재활용 (77)

- 뿐만 아니라 함수화를 해놓으면 쉽게 재활용이 가능하다.
- 또한 순간이동 성공여부와 관련된 수정사항이 있는 경우 이 함수 내부만 수정해주면 된다. (변화에 쉽게 대처)

```

1 using System.Collections.Generic;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CatchGameSlow : MonoBehaviour, CatchGameStarProperty {
6
7     [SerializeField]
8     EffectPool _effect;           /// 특수능력(순간이동) 발동시 나타나는 이펙트 (오브젝트 풀링을 이용)
9
10    public CatchGameStarTriggerType TriggerType { get; private set; }   /// 별의 속성(특수능력)이 발동되는 순간
11    const float _downSpeedAmount = 1.0f;           /// 플레이어의 속도를 얼마나 낮출지
12
13    /// <summary>
14    /// 게임시작시, 단 한번만 초기화해줘도 되는 부분을 이 함수에서 처리한다.
15    /// </summary>
16    void Awake()
17    {
18        TriggerType = CatchGameStarTriggerType.OnEnter; /// 플레이어가 별과 접촉하면 특수능력이 발동된다.
19    }
20
21    /// <summary>
22    /// 플레이어가 게임에 실패하여 재도전할때마다, 초기화해줘야 하는 부분을 이 함수에서 처리한다.
23    /// 별다르게 초기화해줘야하는 부분이 없다면 비워둔다. (함수 자체를 없애면 CatchGameStarProperty 인터페이스를 상속받을 수 없다.)
24    /// </summary>
25    public void Initialize()
26    {
27    }
28
29
30    /// <summary>
31    /// 특수능력(플레이어의 속도감소)를 처리하는 함수이다.
32    /// </summary>
33    /// <param name="target"> 특수능력을 적용할 대상 (플레이어) </param>
34    /// <param name="triggerType"> 유효성 검사 </param>
35    public void PlayEffect(GameObject target, CatchGameStarTriggerType triggerType)
36    {
37        if (triggerType != TriggerType) return;
38        target.GetComponent<DogPlayerController>().OnEnterSlowDownStar(_downSpeedAmount);
39        _effect.ShowEffect(transform.position + Vector3.up * 2.0f);
40    }
41

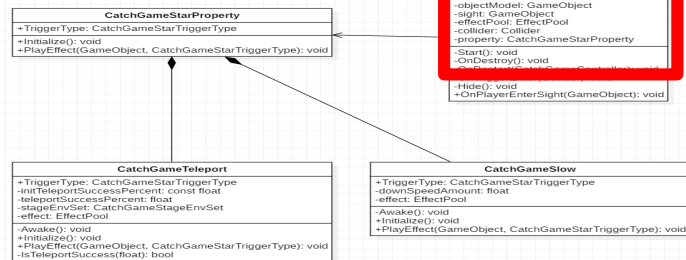
```



```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 /* [견고성] 제약조건 걸기 ...
6 [RequireComponent(typeof(CatchGameStarProperty))]
7
8 public class CatchGameStar : MonoBehaviour {
9
10     [Header("Child Component")]
11     [SerializeField]
12     GameObject _objectModel; /// 별의 3D 모델 (이 오브젝트만 Off해주면, 플레이어의 입장에서 별이 사라진 것처럼 보인다.)
13     [SerializeField]
14     GameObject _sight;      /// 시야 오브젝트 (오브젝트가 플레이어와 접촉여부를 통해 시야내에 플레이어가 들어왔는지 판별)
15     [SerializeField]
16     EffectPool _getEffect;   /// 플레이어가 별을 획득했을때 나타나는 이펙트
17
18     public bool IsCaught { get; private set; } /// 플레이어가 이 별을 획득했는지의 여부
19     Collider _collider;      /// 별의 collider 컴포넌트(충돌판정, 별과 플레이어가 접촉했는지를 판정)
20
21 /* [유연성] 전략패턴 (Strategy Pattern) ...
22 CatchGameStarProperty _property; /// 별의 속성(특수능력)을 참조하는 객체
23
24 /* [클린성] 선언한 순서대로 초기화 ...
25
26 /// <summary>
27 /// 게임시작시 단 한번만 초기화해줘야하는 부분을 여기서 초기화해준다.
28 /// </summary>
29 void Start()
30 {
31     _collider = GetComponent<Collider>();
32     _property = GetComponent<CatchGameStarProperty>();
33     CatchGameEventPublisher.instance.RestartEvent += OnRestart; /// 플레이어가 게임 재도전시 발생하는 이벤트를 수신한다.
34 }
35
36 /// <summary>
37 /// 컴포넌트가 소멸될기전에 호출되는 함수, 이벤트를 연결해제해준다.
38 /// </summary>
39 void OnDestroy()
40 {
41     CatchGameEventPublisher.instance.RestartEvent -= OnRestart;
42 }
43 }

```



## [견고성] 제약조건 걸기 (14)

- 발생할 수 있는 오류나 실수를 방지하기 위해서 제약조건을 두는 것이 견고성을 확보하는 지름길이다.
- 이 컴포넌트를 추가하려면 CatchGameStarProperty 컴포넌트가 그 전에 추가되어 있어야 하는 제약조건을 만들었다.
- 이 컴포넌트가 없다면 Start()함수에서 필히 오류가 발생하기 때문이다.

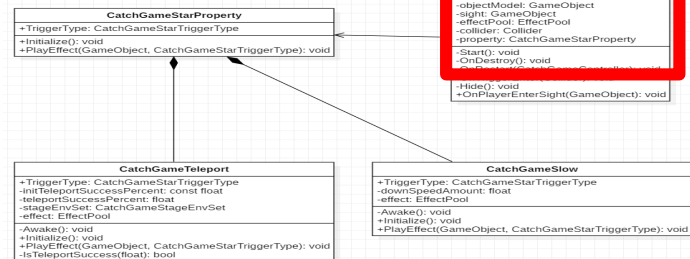
## [클린성] 선언한 순서로 초기화 (50)

- 가독성을 위해서 멤버변수를 선언한 순서대로 초기화하였다.

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 /* [견고성] 제약조건 걸기 ...
6 [RequireComponent(typeof(CatchGameStarProperty))]
7
8 public class CatchGameStar : MonoBehaviour {
9
10     [Header("Child Component")]
11     [SerializeField]
12     GameObject _objectModel; /// 별의 3D 모델 (이 오브젝트만 Off해주면, 플레이어의 입장에서 별이 사라진 것처럼 보인다.)
13     [SerializeField]
14     GameObject _sight;      /// 시야 오브젝트 (오브젝트가 플레이어와 접촉여부를 통해 시야내에 플레이어가 들어왔는지 판별)
15     [SerializeField]
16     EffectPool _getEffect;   /// 플레이어가 별을 획득했을때 나타나는 이펙트
17
18     public bool IsCaught { get; private set; } /// 플레이어가 이 별을 획득했는지의 여부
19     Collider _collider;      /// 별의 collider 컴포넌트(충돌판정, 별과 플레이어가 접촉했는지를 판정)
20
21 /* [유연성] 전략패턴 (Strategy Pattern) ...
22 CatchGameStarProperty _property; /// 별의 속성(특수능력)을 참조하는 객체
23
24 /* [클린성] 선언한 순서대로 초기화 ...
25
26 /// <summary>
27 /// 게임시작시 단 한번만 초기화해줘야하는 부분을 여기서 초기화해준다.
28 /// </summary>
29 void Start()
30 {
31     _collider = GetComponent<Collider>();
32     _property = GetComponent<CatchGameStarProperty>();
33     CatchGameEventPublisher.instance.RestartEvent += OnRestart; /// 플레이어가 게임 재도전시 발생하는 이벤트를 수신한다.
34 }
35
36 /// <summary>
37 /// 컴포넌트가 소멸될기전에 호출되는 함수, 이벤트를 연결해제해준다.
38 /// </summary>
39 void OnDestroy()
40 {
41     CatchGameEventPublisher.instance.RestartEvent -= OnRestart;
42 }

```



## [유연성] 전략패턴 (42)

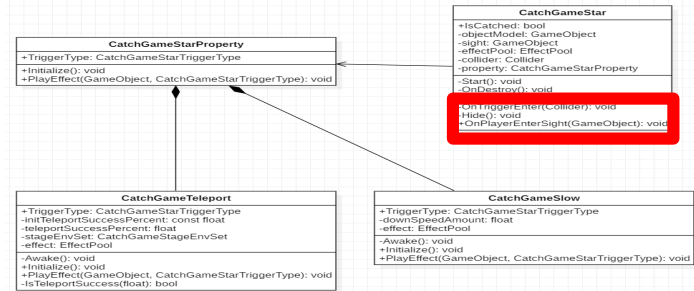
- property 멤버변수에 어떤 값을 할당해주느냐에 따라 다른 특수능력을 가진 별이 된다.
- 새로운 특수능력을 가진 별을 만들고 싶다면...
- (1) CatchGameStarProperty 인터페이스를 구성하는 클래스를 만들고
- (2) 오브젝트에 새로 만든 클래스 컴포넌트를 추가해주면 된다.
- 중요한 점은 이 과정에서 기존의 코드를 수정해야하는 작업이 필요없기 때문에 변화에 유연하게 대처할 수 있다.



```

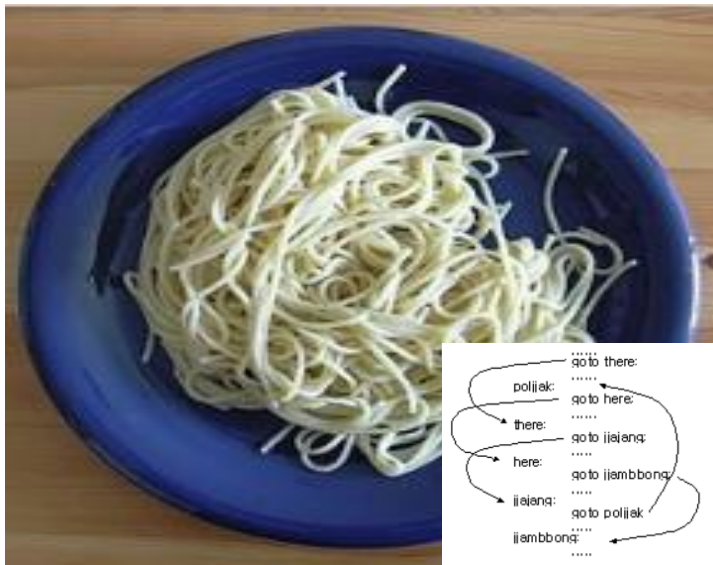
66 /// <summary>
67 /// 플레이어가 게임을 재도전할 시 호출되는 함수.
68 /// 게임을 재시작할때마다 초기화해줘야하는 부분을 여기서 초기화해준다.
69 /// </summary>
70 /// <param name="sender"></param>
71 void OnRestart(CatchGameController sender)
72 {
73     _objectModel.SetActive(true);
74     _sight.SetActive(true);
75     IsCaught = false;
76     _collider.enabled = true;
77     _property.Initialize();
78 }
79 /// <summary>
80 /// 어떤 오브젝트가 별에 접촉하면 호출되는 함수
81 /// </summary>
82 /// <param name="other"></param>
83 void OnTriggerEnter(Collider other)
84 {
85     if (other.CompareTag("Player")) /// 접촉한 대상이 플레이어인 경우
86     {
87         IsCaught = true;
88         _property.PlayEffect(other.gameObject, CatchGameStarTriggerType.OnEnter);
89         CatchGameEventPublisher.instance.DoGetStar(this);
90         _getEffect.ShowEffect(transform.position + Vector3.up * 2.0f);
91         Hide();
92     }
93 }
94 /// <summary>
95 /// 별을 플레이어로부터 숨긴다.
96 /// gameobject 전체를 Off시키는 것이 아니라, 별의 외형만 사라지게 만든다.
97 /// gameobject 전체를 Off시켜버리면, 이벤트(플레이어가 게임 재시작시 발생하는 이벤트)를 수신받을 수 없고 그에 따른 대처도 불가능하다.
98 /// </summary>
99 void Hide()
100 {
101     _objectModel.SetActive(false);
102     _sight.SetActive(false);
103     _collider.enabled = false;
104 }
105
106 /// <summary>
107 /// 플레이어가 별의 시야에 들어오면 호출되는 함수.
108 /// </summary>
109 /// <param name="player"> 플레이어를 참조하는 객체 </param>
110 public void OnPlayerEnterSight(GameObject player)
111 {
112     _property.PlayEffect(player, CatchGameStarTriggerType.OnSight);
113 }

```



# 코드를 짤수록 스파게티 코드가 되어갈때.

— — —



## 내가 겪은 문제점

- 코드를 짤수록 코드 간의 의존도가 높아져서 상당히 복잡해졌다.
- 이 부분을 수정하면 저 부분도 수정해야 되는 경우가 많았고, 이렇게 높은 결합도는 잦은 오류를 발생시켰다.

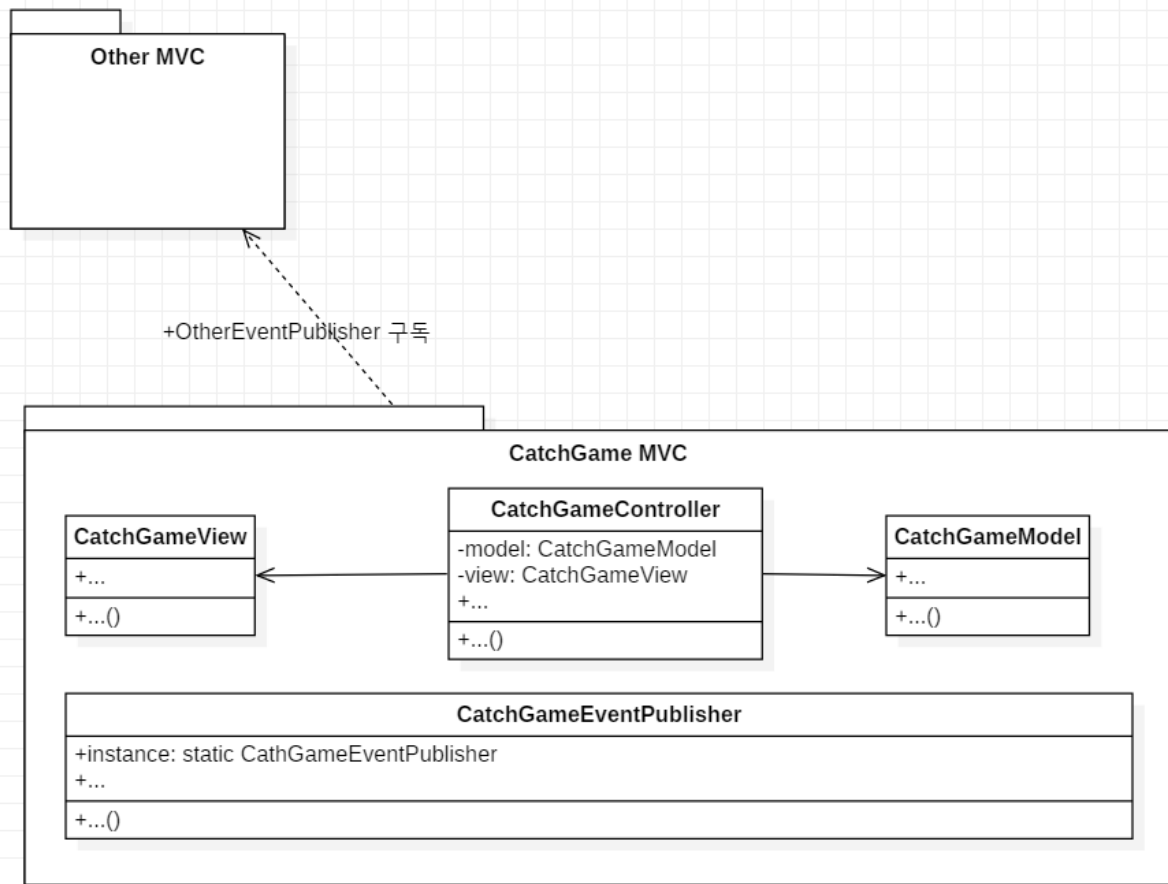
## 나의 해결책

- 읽었던 디자인 패턴 교재를 다시 읽는다.
- 어려움을 겪고 난 뒤에 책을 보면 코드간의 결합도를 낮추려는 패턴들의 의도와 원리가 좀 더 체감된다.
- 전략패턴은 내가 정말 그 필요성을 마음으로 느꼈던 디자인패턴이었으며, 내가 가장 자주 애용하는 디자인패턴이다.

# 코드 리뷰

전반적인 빙글머리 게임 시스템 구현

# UML



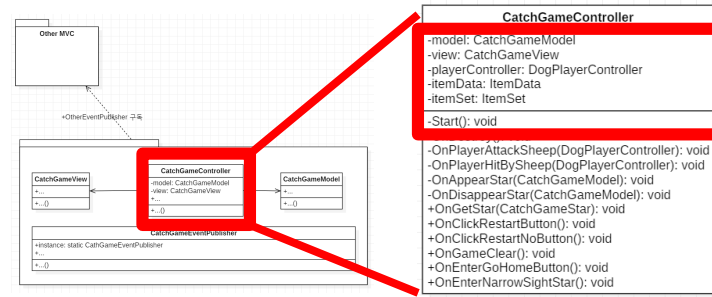
## MVC 패턴 사용

- CatchGame**Controller**에서 사용자의 입력을 받아서 CatchGame**Model**과 CatchGame**View**를 업데이트한다.
- CatchGame**Model**에서 게임관련 변수를 업데이트한다.
- CatchGame**View**에서 게임관련 변수를 참조해서 UI를 사용자에게 보여준다.

## 이벤트로 통신

- 서로 다른 MVC 모델은 이벤트를 통해 통신하여 모델간의 결합도를 낮춘다.
- CatchGame**EventPublisher**에서 이벤트를 발행한다.

```
5 using UnityEngine.SceneManagement;
6
7 public class CatchGameController : MonoBehaviour {
8     /* [클린성] 네이밍 ...
9
10
11     /* [건강성] 캡슐화 ...
12
13     [Header("Refer")]
14     [SerializeField]
15     CatchGameModel _model;           /// MVC 패턴에서 Model
16     [SerializeField]
17     CatchGameView _view;             /// MVC 패턴에서 View
18
19     DogPlayerController _playerController;   ///플레이어를 참조하는 객체 (플레이어에게 게임 이벤트 발생시 알림)
20     ItemData _itemData;                     /// 인벤토리 정보
21     ItemSet _itemSet;                       /// 아이템 정보
22
23     /* [독립성] 이벤트 사용 ...
24     /// <summary>
25     /// 변수를 초기화하고, 이벤트를 연결한다.
26     /// </summary>
27     void Start()
28     {
29         _playerController = GameObject.FindWithTag("Player").GetComponent<DogPlayerController>();
30         _itemData = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemData>();
31         _itemSet = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemSet>();
32         _playerController.OnUpdateSheepIndicator(_model.GetNearestSheepPlayerCanHit());
33         _view.UpdateSheepCountPanel(_model.CurrentSheepCount);
34
35         CatchGameEventPublisher.instance.PlayerAttackSheepEvent += OnPlayerAttackSheep; /// 싱글턴 패턴 사용
36         CatchGameEventPublisher.instance.PlayerHitBySheepEvent += OnPlayerHitBySheep;
37         CatchGameEventPublisher.instance.AppearStarEvent += OnAppearStar;
38         CatchGameEventPublisher.instance.DisappearStarEvent += OnDisappearStar;
39         CatchGameEventPublisher.instance.GetStarEvent += OnGetStar;
40     }
41 }
```



## [클린성] 네이밍

- 읽는이의 코드이해를 돕기 위해서 일관성있게 네이밍한다.
- 기본적으로 네이밍은 '낙타 표기법' 을 사용하였다.
- 변수는 소문자로 시작하고, 함수는 대문자로 시작한다.
- 변수는 명사로 하고, 함수는 동사로 시작한다.
- private 멤버변수는 접두어로 `_`을 붙이고, public 속성은 대문자로 시작한다.

```
using UnityEngine.SceneManagement;

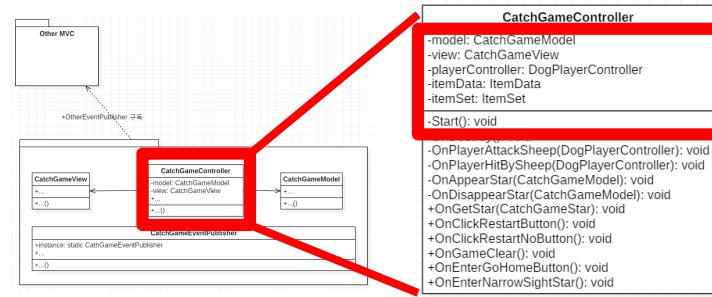
public class CatchGameController : MonoBehaviour {
    /* [클린성] 네이밍 ... */
    /* [견고성] 캡슐화 ... */

    [Header("Refer")]
    [SerializeField]
    CatchGameModel _model;          /// MVC 패턴에서 Model
    [SerializeField]
    CatchGameView _view;           /// MVC 패턴에서 View

    DogPlayerController _playerController;  /// 플레이어를 참조하는 객체 (플레이어에게 게임 이벤트 발생시 알림)
    ItemData _itemData;                /// 인벤토리 정보
    ItemSet _itemSet;                  /// 아이템 정보

    /* [독립성] 이벤트 사용 ... */
    /// <summary>
    /// 변수를 초기화하고, 이벤트를 연결한다.
    /// </summary>
    void Start()
    {
        _playerController = GameObject.FindWithTag("Player").GetComponent<DogPlayerController>();
        _itemData = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemData>();
        _itemSet = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemSet>();
        _playerController.OnUpdateSheepIndicator(_model.GetNearestSheepPlayerCanHit());
        _view.UpdateSheepCountPanel(_model.CurrentSheepCount);

        CatchGameEventPublisher.instance.PlayerAttackSheepEvent += OnPlayerAttackSheep; /// 싱글턴 패턴 사용
        CatchGameEventPublisher.instance.PlayerHitBySheepEvent += OnPlayerHitBySheep;
        CatchGameEventPublisher.instance.AppearStarEvent += OnAppearStar;
        CatchGameEventPublisher.instance.DisappearStarEvent += OnDisappearStar;
        CatchGameEventPublisher.instance.GetStarEvent += OnGetStar;
    }
}
```



```
-model: CatchGameModel
-view: CatchGameView
-playerController: DogPlayerController
-itemData: ItemData
-itemSet: ItemSet
.Start(): void
.OnPlayerAttackSheep(DogPlayerController): void
.OnPlayerHitBySheep(DogPlayerController): void
.OnAppearStar(CatchGameModel): void
.OnDisappearStar(CatchGameModel): void
.OnGetStar(CatchGameStar): void
.OnClickRestartButton(): void
.OnClickRestartNoButton(): void
.OnGameClear(): void
.OnEnterGoHomeButton(): void
.OnEnterNarrowSightStar(): void
```

## [클린성] 코드 자체를 주석화

- 되도록 필요한 주석만 달고, 코드 자체가 주석이 되도록 코딩한다.
- 주석도 유지보수의 대상이므로, 과도한 주석은 프로젝트 진전을 어렵게 만들거나 오히려 코드이해에 독이 될 수도 있기 때문이다.
- 네이밍할때, 변수는 명사화하고, 함수는 동사로 시작하면, 코드 자체가 주석화되는 효과를 누릴 수 있다.
- 예를 들어, 46번 line을 보면 UI를 담당하는 view 컴포넌트에서 양(sheep)의 수를 보여주는 패널을 업데이트한다는 것을 코드자체가 설명해준다.

```
using UnityEngine.SceneManagement;

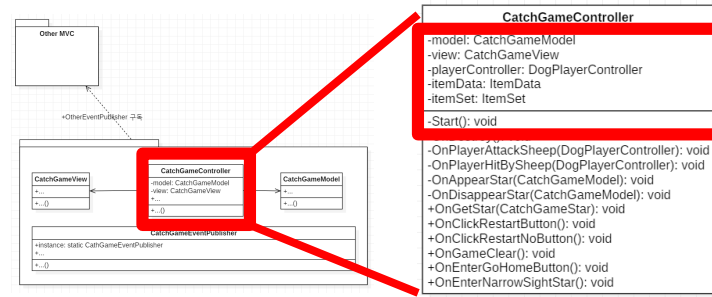
public class CatchGameController : MonoBehaviour {
    /* [클린성] 네이밍 */
    /* [견고성] 캡슐화 ... */

    [Header("Refer")]
    [SerializeField]
    CatchGameModel _model;        /// MVC 패턴에서 Model
    [SerializeField]
    CatchGameView _view;          /// MVC 패턴에서 View

    DogPlayerController _playerController;    ///플레이어를 참조하는 객체 (플레이어에게 게임 이벤트 발생시 알림)
    ItemData _itemData;                      /// 인벤토리 정보
    ItemSet _itemSet;                        /// 아이템 정보

    /* [독립성] 이벤트 사용 ... */
    /// <summary>
    /// 변수를 초기화하고, 이벤트를 연결한다.
    /// </summary>
    void Start()
    {
        _playerController = GameObject.FindWithTag("Player").GetComponent<DogPlayerController>();
        _itemData = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemData>();
        _itemSet = GameObject.FindWithTag("Managers").GetComponentInChildren<ItemSet>();
        _playerController.OnUpdateSheepIndicator(_model.GetNearestSheepPlayerCanHit());
        _view.UpdateSheepCountPanel(_model.CurrentSheepCount);

        CatchGameEventPublisher.instance.PlayerAttackSheepEvent += OnPlayerAttackSheep; /// 싱글턴 패턴 사용
        CatchGameEventPublisher.instance.PlayerHitBySheepEvent += OnPlayerHitBySheep;
        CatchGameEventPublisher.instance.AppearStarEvent += OnAppearStar;
        CatchGameEventPublisher.instance.DisappearStarEvent += OnDisappearStar;
        CatchGameEventPublisher.instance.GetStarEvent += OnGetStar;
    }
}
```



## [독립성] 이벤트 활용 (48~52)

- C#의 이벤트는 옵저버패턴을 언어적으로 제공해준다.
- 코드간의 결합을 느슨하게 만들 수 있기 때문에 이벤트를 자주 활용한다.
- 서로를 직접 참조하지 않아도(결합도를 낮추면서) 상대객체의 멤버함수를 호출하는 효과를 누릴 수 있기 때문이다.

## [견고성] 캡슐화

- A클래스의 멤버변수는 A클래스의 멤버함수에 의해서만 수정할 수 있도록 한다.
- 캡슐화가 완벽히 이루어지면, 예외가 발생했을 때, 그 원인이 되는 코드를 쉽게 예측할 수 있다.

```

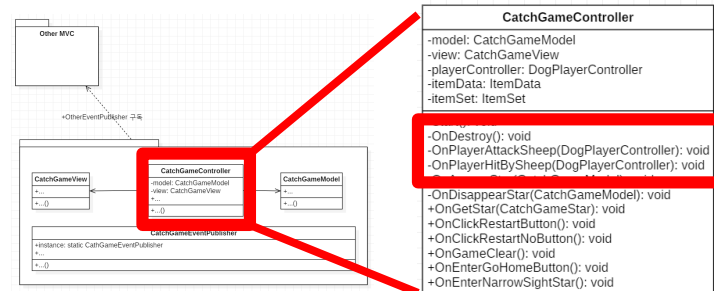
55 56 61 62 63 64 65 66 67 68 69 70 71 72 73 78 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
/* [견고성] 쌍으로 뒤야하는 것 ...
/// <summary>
/// 이 객체가 소멸되기 전에, 이벤트를 해제한다.
/// </summary>
void OnDestroy()
{
    CatchGameEventPublisher.instance.PlayerAttackSheepEvent -= OnPlayerAttackSheep;
    CatchGameEventPublisher.instance.PlayerHitBySheepEvent -= OnPlayerHitBySheep;
    CatchGameEventPublisher.instance.AppearStarEvent -= OnAppearStar;
    CatchGameEventPublisher.instance.DisappearStarEvent -= OnDisappearStar;
    CatchGameEventPublisher.instance.GetStarEvent -= OnGetStar;
}

/* [견고성] 테스트 단위 함수 분리 ...
/* [고난] 게임은 함께 만드는 것이다. ...
/// <summary>
/// 플레이어가 필드몬스터인 양을 공격할때, 관련변수와 UI(view)를 업데이트하고, 모든 양을 잡으면 게임 클리어시킨다.
/// </summary>
/// <param name="sender"></param>
void OnPlayerAttackSheep(DogPlayerController sender)
{
    _model.DecreaseSheepCount(1);
    if(!_model.IsAppearStar())
        _model.AppearStar();
    _view.UpdateSheepCountPanel(_model.CurrentSheepCount);
    _playerController.OnUpdateSheepIndicator(_model.GetNearestSheepPlayerCanHit());
    AudioManager.instance.PlaySoundEffect("Kick");

    if (_model.CurrentSheepCount <= 0)
        OnGameClear();
}

/// <summary>
/// 플레이어가 필드몬스터인 양에게 공격받았다면, 미션실패이며 재도전할지를 물어본다.
/// </summary>
/// <param name="sender"></param>
void OnPlayerHitBySheep(DogPlayerController sender)
{
    _model.DeactivateAllSheeps();
    _view.SwitchUI(_view.AskRestartPanel.gameObject, true);
    AudioManager.instance.PlaySoundAndResumeBGM("Fail");
}

```



## [견고성] 습관 = 견고성 (66~70)

- 평소에 코드를 짜는 습관으로 코드의 오류를 미리 방지할 수 있다는 생각을 품고 있다.
- 그 중에 하나로, 이벤트를 등록하기 위한 코드를 쓸 때, 해제할 시점에 해제를 위한 코드도 동시에 미리 써놓는다.
- C++의 경우, new - delete도 이와 같이 코드를 쓴다.



```

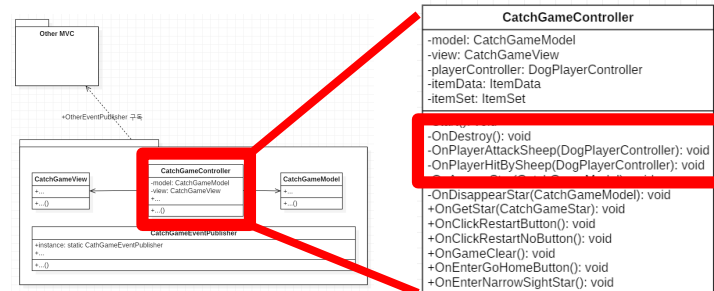
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
/* [견고성] 쌍으로 뒤야하는 것 ...
/// <summary>
/// 이 객체가 소멸되기 전에, 이벤트를 해제한다.
/// </summary>
void OnDestroy()
{
    CatchGameEventPublisher.instance.PlayerAttackSheepEvent -= OnPlayerAttackSheep;
    CatchGameEventPublisher.instance.PlayerHitBySheepEvent -= OnPlayerHitBySheep;
    CatchGameEventPublisher.instance.AppearStarEvent -= OnAppearStar;
    CatchGameEventPublisher.instance.DisappearStarEvent -= OnDisappearStar;
    CatchGameEventPublisher.instance.GetStarEvent -= OnGetStar;
}

/* [견고성] 테스트 단위 함수 분리 ...
/* [고난] 게임은 함께 만드는 것이다. ...
/// <summary>
/// 플레이어가 필드몬스터인 양을 공격할때, 관련변수와 UI(view)를 업데이트하고, 모든 양을 잡으면 게임 클리어시킨다.
/// </summary>
/// <param name="sender"></param>
void OnPlayerAttackSheep(DogPlayerController sender)
{
    _model.DecreaseSheepCount(1);
    if(_model.IsAppearStar())
        _model.AppearStar();
    _view.UpdateSheepCountPanel(_model.CurrentSheepCount);
    _playerController.OnUpdateSheepIndicator(_model.GetNearestSheepPlayerCanHit());
    AudioManager.instance.PlaySoundEffect("Kick");

    if (_model.CurrentSheepCount <= 0)
        OnGameClear();
}

/// <summary>
/// 플레이어가 필드몬스터인 양에게 공격받았다면, 미션실패이며 재도전할지를 물어본다.
/// </summary>
/// <param name="sender"></param>
void OnPlayerHitBySheep(DogPlayerController sender)
{
    _model.DeactivateAllSheeps();
    _view.SwitchUI(_view.AskRestartPanel.gameObject, true);
    AudioManager.instance.PlaySoundAndResumeBGM("Fail");
}

```



## [견고성] 테스트 단위 함수 분리

- 예측가능한 디버깅을 위해서 테스트 단위로 함수를 쪼갬다.
- 이를 위한 방법으로 함수 하나에는 하나의 기능(행동)만을 담당하도록 노력해서 코드를 짰다.
- 어떤 기능에 문제가 생기면, 그 기능을 담당하는 함수를 바로 보면 되기 때문에 예외의 원인을 빠르게 예측할 수 있다.

# 게임은 함께 만드는 것.

## 내가 겪은 문제점

- 이 프로젝트는 혼자 만들었지만, 이전에 팀원들과 같이 게임을 만들었던 경험을 바탕으로 내가 겪었던 문제를 적어본다.
- 게임을 함께 만든다는 사실은, 팀원 서로가 짠 코드를 봐야할 상황이 많다는 것을 의미한다.
- 내가 코드를 짜는 시간보다 동료가 짠 코드를 보는 시간이 훨씬 더 많을 때가 비일비재했다.

## 나의 해결책

- 단기적으로, 마이크로소프트 코딩 스타일을 참조하여 코딩 스타일을 통일했다.
- 중기적으로, 일주일동안 서로가 짠 코드를 브리핑하는 회의시간을 가졌다.
- 장기적으로, 『클린코드, 로버트 C 마틴』 『C# 코딩의 기술, 가와마타아키라』와 같은 책을 읽으며 어떻게 하면 코드를 클린하게 짤 수 있을지를 항상 고민했다.



/\* [견고성] 초기화(혹은 업데이트) 순서는 명확히! ...

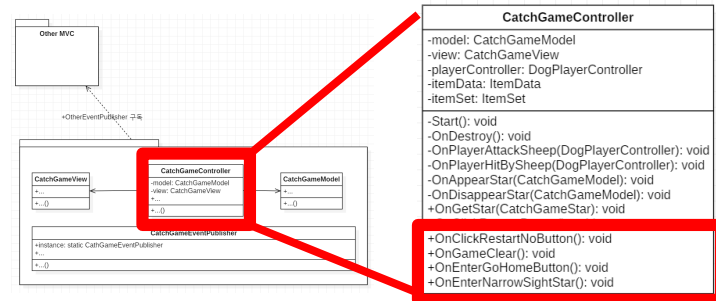


- 여러 컴포넌트가 상호작용하는 게임 프로젝트에서는 무엇보다도 컴포넌트 사이의 초기화(혹은 업데이트) 순서가 명확해야 예기치 않은 오류를 방지할 수 있다.
- 169~174 line을 예로 들면, 나는 MVC 모델에서 controller → model → view 순으로 업데이트했다.
- 컴포넌트 사이의 초기화(혹은 업데이트) 순서가 상관없는 경우에도, 명확하게 정해두어야 디버깅하기도 쉬워진다.

```

178 }
179 /// <summary>
180 /// 플레이어가 게임 재도전 안함 버튼을 누를때 호출되는 함수
181 /// </summary>
182 public void OnClickRestartNoButton()
183 {
184     _model.GoHomeWithFail();
185 }
186
187 /// <summary>
188 /// 게임 클리어시 호출되는 함수
189 /// </summary>
190 public void OnGameClear()
191 {
192     var rewardItemInfo = _model.GetRewardItemInfo(_itemSet);
193     int rewardItemCount = _model.GetRewardItemCount();
194     _view.UpdateGameClearPanel(_model.CurrentStarCount, rewardItemInfo, rewardItemCount);
195     _view.SwitchUI(_view.GameClearPanel.gameObject, true);
196     AudioMgr.instance.PlaySoundAndResumeBGM("Clear");
197 }
198
199 /// <summary>
200 /// 홈으로 돌아가기 버튼을 눌렀을때 호출되는 함수
201 /// </summary>
202 public void OnEnterGoHomeButton()
203 {
204     _model.GoHome(_itemData, _itemSet);
205 }
206
207 /// <summary>
208 /// 시야를 가리는 별을 먹었을때 호출되는 함수
209 /// </summary>
210 public void OnEnterNarrowSightStar()
211 {
212     _view.SwitchUI(_view.NarrowSightPanel, true);
213 }
214 }
215

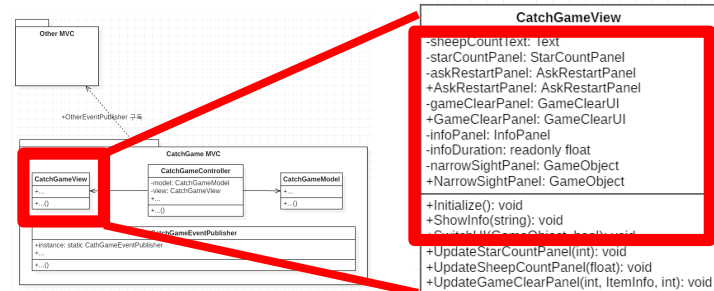
```



```

public class CatchGameView : MonoBehaviour {
    7
    8
    16 /* [견고성] 클래스의 역할을 세분화 ...
    [Header("Sheep Count Panel")]
    [SerializeField]
    17 Text sheepCountText;
    18
    19
    20 [Header("Star Count Panel")]
    [SerializeField]
    21 StarCountPanel _starCountPanel;
    22
    23
    24 [Header("Ask Restart Panel")]
    [SerializeField]
    25 AskRestartPanel _askRestartPanel;
    26
    27
    28
    29 [Header("Game Clear Panel")]
    [SerializeField]
    30 GameClearUI _gameClearPanel;
    31
    32
    33
    34 [Header("Show Info Panel")]
    [SerializeField]
    35 InfoPanel _infoPanel;
    36
    37
    38
    39 [Header("Narrow Sight Panel")]
    [SerializeField]
    40 private GameObject _narrowSightPanel;
    41
    42
    43
    44
    45
    46
    47
    48
    49
    50
    51
    52
    53
    54
    55
    56
    57
    58
    59
    60

```



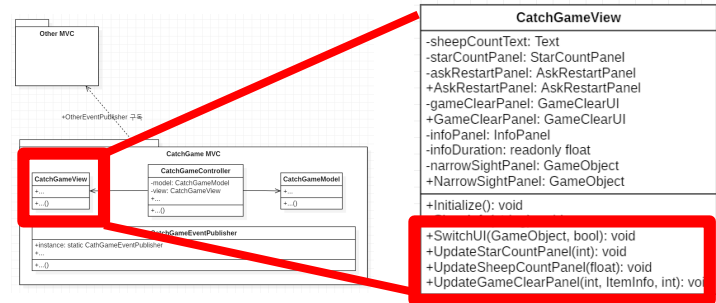
## [견고성] 클래스의 역할을 세분화

- 클래스의 역할을 적당히 세분화할수록 견고성이 높아진다.
- 책임이 명확해져 디버깅하기 쉬워지기 때문이다.
- 이 클래스를 예로들어 설명하면, 게임을 구성하는 각 UI를 현재 클래스에서 모두 처리하는 것이 아니다.
- 각 UI를 전담하는 클래스를 각각 만들어주고 이 클래스의 멤버변수가 이들을 각각 참조한다. (16~42 line)
- 각 UI를 전담하는 이 클래스들을 중재하는 역할을 현재 클래스가 담당한다.

```

61 // <summary>
62 /// 비를 플레이어에게 보여주거나 사라지게 만든다.
63 // </summary>
64 /// <param name="UI">UI컴포넌트</param>
65 /// <param name="flag">플레이어에게 보여줄지의 여부</param>
66 public void SwitchUI(GameObject UI, bool flag)
67 {
68     UI.SetActive(flag);
69 }
70
71 // <summary>
72 /// 현재 습득한 별의 수를 보여주는 비를 업데이트한다.
73 // </summary>
74 /// <param name="currentStarCount">현재 습득한 별의 수</param>
75 public void UpdateStarCountPanel(int currentStarCount)
76 {
77     _starCountPanel.UpdateUI(currentStarCount);
78 }
79
80 // <summary>
81 /// 현재 필드위에 있는 양의 수를 보여주는 비를 업데이트한다.
82 // </summary>
83 /// <param name="currentSheepCount">현재 필드위에 있는 양의 수</param>
84 public void UpdateSheepCountPanel(float currentSheepCount)
85 {
86     sheepCountText.text = currentSheepCount + "마리";
87 }
88
89 // <summary>
90 /// 게임 클리어시 클리어 정보를 보여주는 비를 업데이트한다.
91 // </summary>
92 /// <param name="currentStarCount">현재 습득한 별의 수</param>
93 /// <param name="rewardItemInfo">보상아이템 정보</param>
94 /// <param name="rewardItemCount">보상아이템 개수</param>
95 public void UpdateGameClearPanel(int currentStarCount, ItemInfo rewardItemInfo, int rewardItemCount)
96 {
97     _gameClearPanel.UpdateUI(currentStarCount, rewardItemInfo, rewardItemCount);
98 }
99
100

```



```

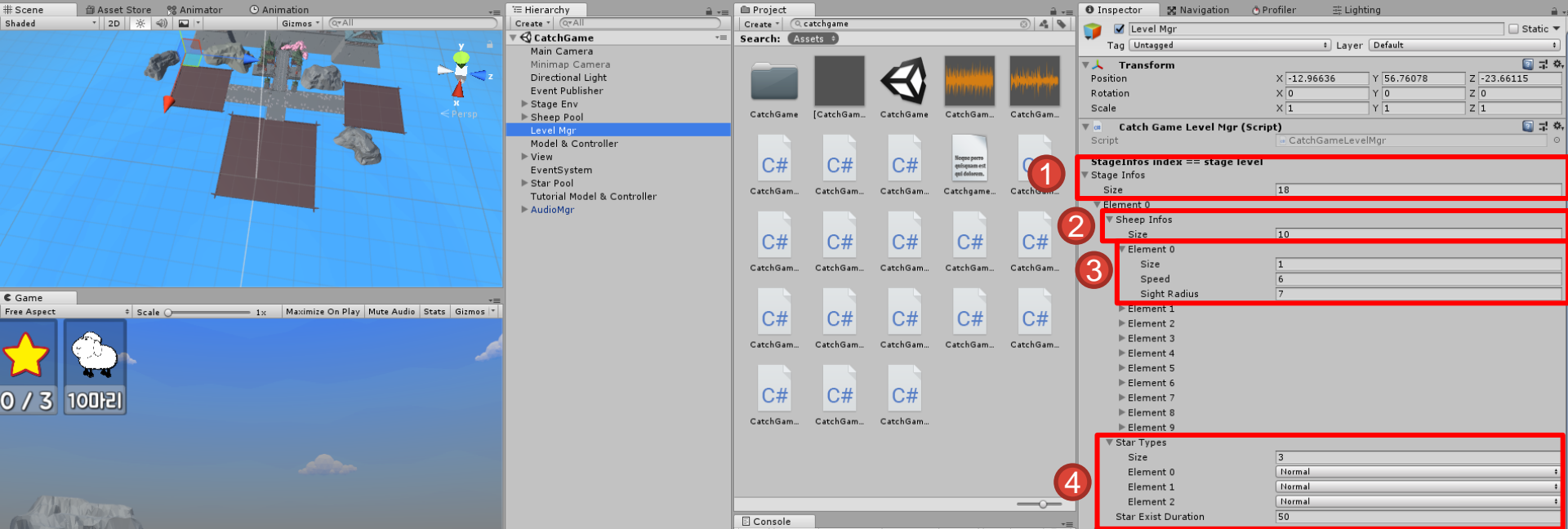
8 public class CatchGameModel : MonoBehaviour {
9
10     /* [견고성] 애초에 기존코드 수정방지 ...
11     [Header("Info")]
12     [SerializeField]
13     ItemSet.ItemNameType _rewardItemNameType; /// 게임 클리어시 보상아이템 정보
14
15     [Header("Refer")]
16     [SerializeField]
17     CatchGameStageEnvSet _stageEnvSet;          /// 스테이지 환경(맵) 정보
18     [SerializeField]
19     CatchGameLevelMgr _levelMgr;                /// 난이도 정보
20     [SerializeField]
21     ObjectPool_sheepPool;                      /// 오브젝트 풀링 -> 필드몬스터 양
22     [SerializeField]
23     CatchGameStarPool _starPool;               /// 오브젝트 풀링 -> 필드아이템 별
24
25     DogPlayerController _playerController;     /// 플레이어 컨트롤러 (게임시작과 함께 플레이어 초기화 용도)
26     MinGameData _data;                        /// 게임 세이브 관련 클래스
27
28     public int CurrentStageLevel { get; private set; }    /// 현재 스테이지 난이도
29
30     /// (필드몬스터) 양 관련 변수
31     public int CurrentSheepCount { get; private set; }    /// 생존해있는 양(필드몬스터)의 수
32     int _initSheepCount;                                /// 처음 환경(맵)에 생성한 양의 수
33     GameObject[] _sheeps;                               /// 환경에 생성된 양들을 참조하는 오브젝트
34
35     /// (필드아이템) 별 관련 변수
36     public int CurrentStarCount { get; private set; }    /// 플레이어가 획득한 별의 수
37     GameObject[] _stars;                                /// 환경에 생성된 별들을 참조하는 오브젝트
38     int _createdStarCount = 0;                          /// 현재까지 생성된 별의 수
39     const int _maxStarCount = 3;                        /// 환경에 생성할 수 있는 별의 총 개수
40
41     /* 초기화를 두개의 함수로 분리 ...
42     /* [클린성] 가독성을 위해서 Enumeration 이용 ...
43     /// <summary>
44     /// 게임 입장시, "단 한번만" 초기화해도 되는 부분을 이 함수에서 초기화한다.
45     /// </summary>
46     void Awake()
47     {
48         _playerController = GameObject.FindWithTag("Player").GetComponent<DogPlayerController>();
49         var minGameDataSet = GameObject.FindWithTag("Managers").GetComponentInChildren<MinGameDataSet>();
50         _data = minGameDataSet.GetMinGameData(MingameType.Type.CatchGame);
51         Initialize();
52     }

```



## [견고성] 애초에 기존코드 수정방지

- 애초에 기존코드를 수정하지 않도록 하는 것이 오류를 막는 지름길이다.
- 오래된 코드를 수정하는 작업은 예기치 못한 오류를 불러올 공산이 매우 크기 때문이다.
- 이를 위해 유니티의 인스펙터 뷰를 적극 활용하였다.
- 예를 들어 게임난이도를 인스펙터 뷰에서 수정할 수 있으며, 이 과정에서 기존 코드를 직접 보거나 건드릴 필요가 없다. line 25에 명시된 CatchGameLevelMgr 클래스가 이를 담당한다.



## [견고성] 애초에 기존코드 수정방지

- 게임의 난이도를 수정할 때, 기존의 코드를 볼 필요없이 유니티의 인스펙터 뷰에서 수정하면 되도록 코드를 짰다.
- 게임의 난이도와 같이 수정이 잦은 사항은 외부 윈도우에서 쉽게 수정할 수 있도록 하여 편의성과 코드의 견고성을 보장하도록 했다.

- 1 스테이지의 수
- 2 필드몬스터인 양의 수
- 3 양의 크기, 속도, 시야 설정
- 4 필드아이템인 별의 수, 각 별의 특수능력 설정



```

67 66 /* [클린성] 함수의 배치순서 ...
72 71 /// <summary>
73 70 /// 게임 재도전을 할때마다, 초기화해야되는 부분을 이 함수에서 초기화한다.
74 69 /// </summary>
75 68 public void Initialize()
76 67 {
77 66 StopAllCoroutines();
78 65 CurrentStageLevel = _data.CurrentStageLevel;
79 64 var currentStageEnv = GetCurrentStageEnv(CurrentStageLevel); // 현재 환경(맵) 정보
80 63 var currentStageInfo = GetCurrentStageInfo(CurrentStageLevel); // 현재 난이도 정보
81
82 62 _sheepPool.DeactivateAllObejcts();
83 61 _stageEnvSet.ActivateCurrentStageEnv(CurrentStageLevel);
84
85 60 InitializePlayer(currentStageEnv);
86 59 InitializeSheep(currentStageEnv, currentStageInfo);
87 58 InitializeStar(currentStageInfo);
88 57 }
89
90 56 /* [클린성] 함수 하나당 10줄 이하 ...
97 55 /// <summary>
98 54 /// 현재 환경(맵) 정보를 반환한다.
99 53 /// </summary>
100 52 /// <param name="stageLevel">스테이지 난이도</param>
101 51 /// <returns></returns>
102 50 public CatchGameStageEnv GetCurrentStageEnv(int stageLevel)
103 49 {
104 48 return _stageEnvSet.StageEnvs[stageLevel];
105 47 }
106
107 46 /// <summary>
108 45 /// 현재 난이도 정보를 반환한다.
109 44 /// </summary>
110 43 /// <param name="stageLevel">스테이지 난이도</param>
111 42 /// <returns></returns>
112 41 public CatchGameLevelMgr.StageInfo GetCurrentStageInfo(int stageLevel)
113 40 {
114 39 return _levelMgr.StageInfos[stageLevel];
115 38 }

```



## [클린성] 함수의 배치순서 (75)

- 클래스내 함수의 배치순서는 호출순서대로 배치하였다.
- 예를 들어 A함수에서 B함수와 C함수 순서대로 호출한다면, 함수의 배치순서는 A-B-C가 된다.
- 이러한 배치는 코드를 위에서 아래로 내려가면서 읽을때 내용전반을 자연스럽게 이해하도록 해준다.

## [클린성] 함수내 코드는 10줄 이내

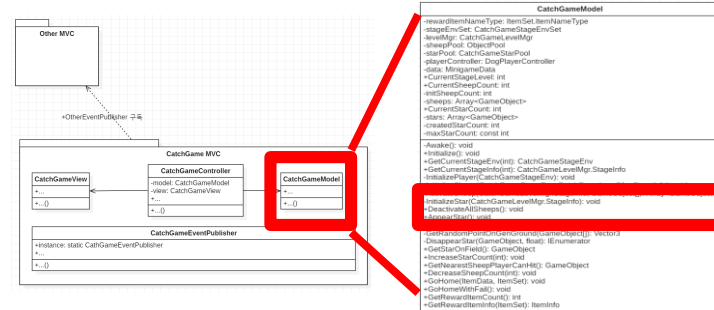
- 길이가 짧은 코드일수록 읽기 편하기 때문이다.
- 더 중요한 이유는 이러한 제약이 함수가 한가지 행동만을 강제하도록 할 수 있기 때문이다.



```

166 /// <summary>
167 /// 필드아이템인 별을 생성 및 초기화한다.
168 /// </summary>
169 /// <param name="currentStageInfo"> 난이도 정보 </param>
170 void InitializeStar(CatchGameLevelMgr.StageInfo currentStageInfo)
171 {
172     _starPool.DeactivateAllStars();
173     CurrentStarCount = 0;
174     _createdStarCount = 0;
175     _stars = _starPool.GetStars(currentStageInfo.StarTypes);
176     AppearStar();
177 }
178
179 /// <summary>
180 /// 필드에 있는 모든 양을 사라지게 한다.
181 /// </summary>
182 public void DeactivateAllSheeps()
183 {
184     _sheepPool.DeactivateAllObjects();
185 }
186
187 /* [독립성] 서로가 서로를 참조하는 경우 방지 ...
188
189 /// <summary>
190 /// 필드아이템인 별이 생성될 조건을 만족하면 필드 위에 생성하고, 특정 시간이 지나면 사라지게 만든다.
191 /// </summary>
192 public void AppearStar()
193 {
194     if (IsAppearStar() == false) return;
195
196     var currentStageEnv = GetCurrentStageEnv(CurrentStageLevel); // 현재 환경(맵) 정보
197     var currentStageInfo = GetCurrentStageInfo(CurrentStageLevel); // 현재 난이도 정보
198     var currentStar = _stars[_createdStarCount];
199     currentStar.SetActive(true);
200     currentStar.transform.position = GetRandomPointOnGenGround(currentStageEnv.GenGrounds);
201     StartCoroutine(DisappearStar(currentStar, currentStageInfo.StarExistDuration));
202     ++_createdStarCount;
203     CatchGameEventPublisher.instance.DoAppearStar(this);
204 }

```



## [견고성] 상호참조 방지 (207)

- 두 객체가 서로를 참조(상호참조)하는 경우 예기치 않은 예외가 발생할 확률이 굉장히 높다.
- 대표적인 예로, 초기화되기 이전에 다른 객체의 멤버함수를 호출해버릴 수 있다.
- 애초에 두 객체가 상호참조하지 않도록 설계해야 되겠지만, 어쩔 수 없이 그래야 하는 경우에는 이벤트를 이용하였다.

CatchGameModel 클래스의 나머지 코드는 생략한다.

# 부록

게임만들기가 취미였던 중학생 시절의 습작

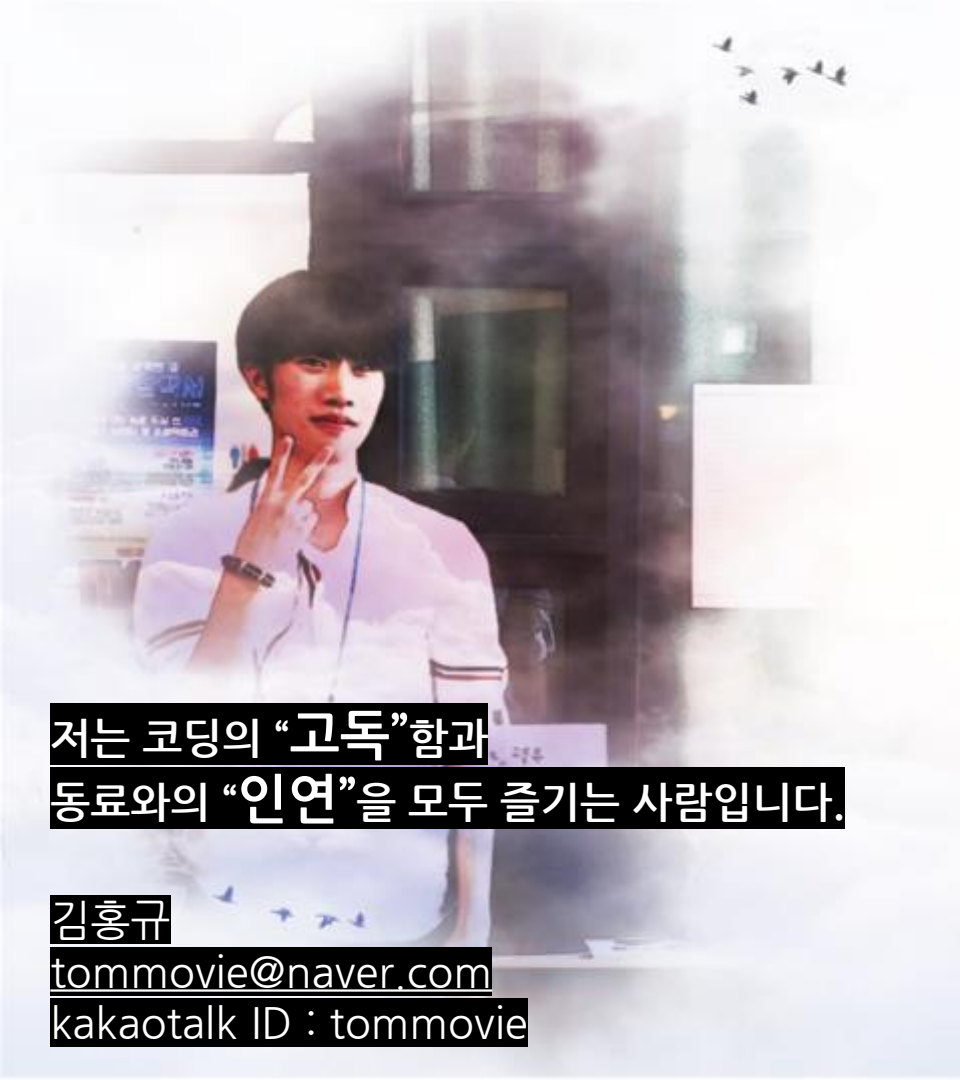
— — —

# 게임만들기가 취미였던 중학생 시절의 습작

— — —



# Contact

A man with dark hair, wearing a white long-sleeved shirt with a blue lanyard, is making a peace sign with his right hand. He is standing in front of a blurred background that appears to be an outdoor setting with a building and some foliage. There are some birds flying in the sky in the background.

저는 코딩의 “고독”함과  
동료와의 “인연”을 모두 즐기는 사람입니다.

김흥규  
[tommovie@naver.com](mailto:tommovie@naver.com)  
kakaotalk ID : tommovie