
[PORTFOLIO]

이름 : 권오현

번호 : 010.2719.5330

이메일 : ohhyun5442@gmail.com

목차

NGUI	3
UGUI	4
CustomUI	5
가상 키보드	7
신규 게임 모드	8
사운드	10
런처	12
그래픽 환경 툴	13
이사만루 콘텐츠 추가	14
ToolTip	16
AssetBundle	17
데이터 뷰어 툴	18
타이머	19
WorldUI	20
Camera Stack	21
구글 광고	22
Jenkins 자동 빌드	23

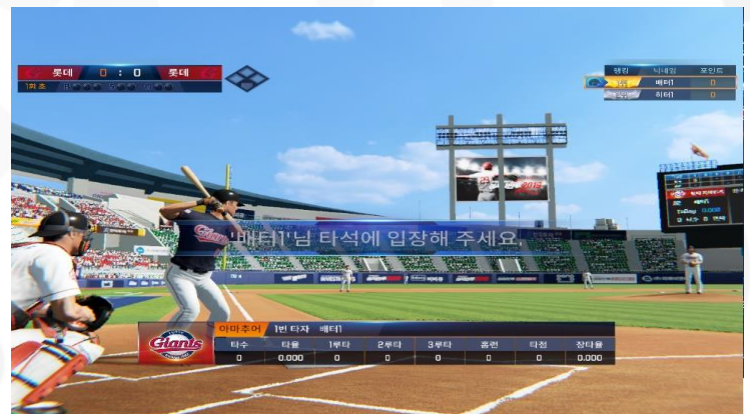
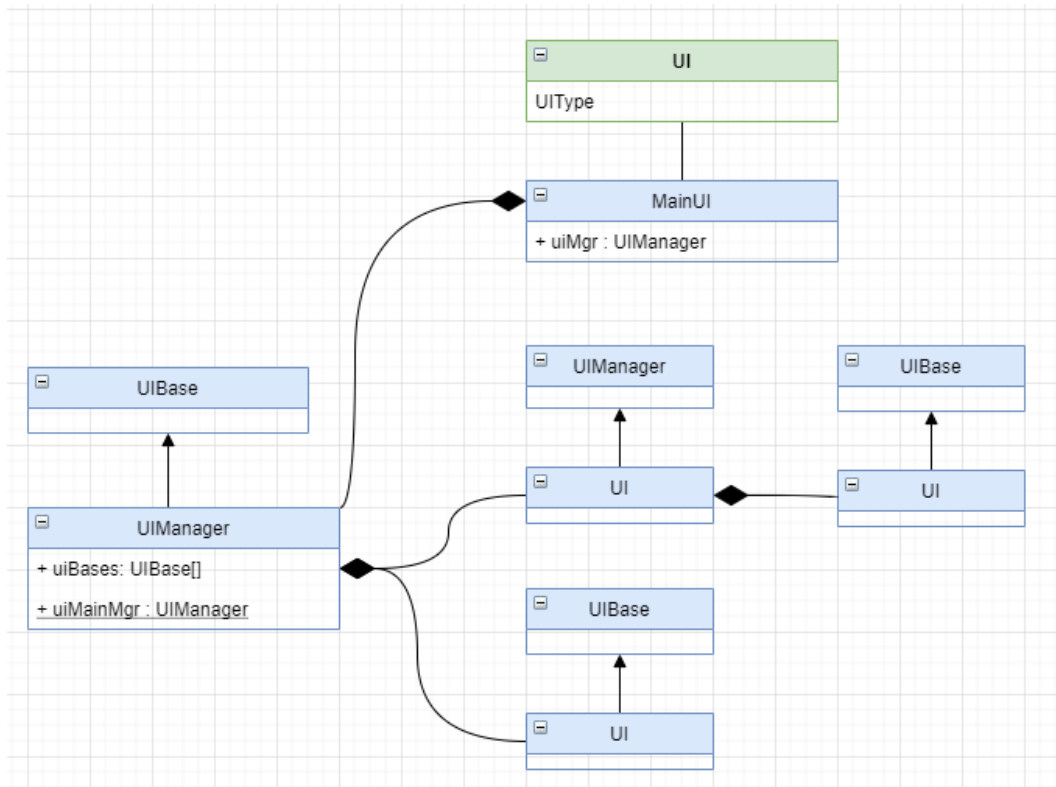
NGUI

사용 목적

유니티 프로젝트에서 UI를 표현하여 유저들에게 게임 정보를 표현하기 위하여 사용했습니다.

콘텐츠 설명

유니티 프로젝트에서 제공하는 UI로 Panel의 Depth를 이용하여 UI의 Depth를 관리했습니다. 그리고 Text, Image, Button, Toggle, Scroll 등의 컴포넌트를 사용하여 UI를 표현했습니다.



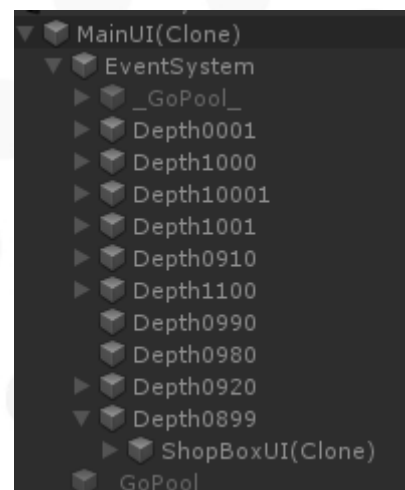
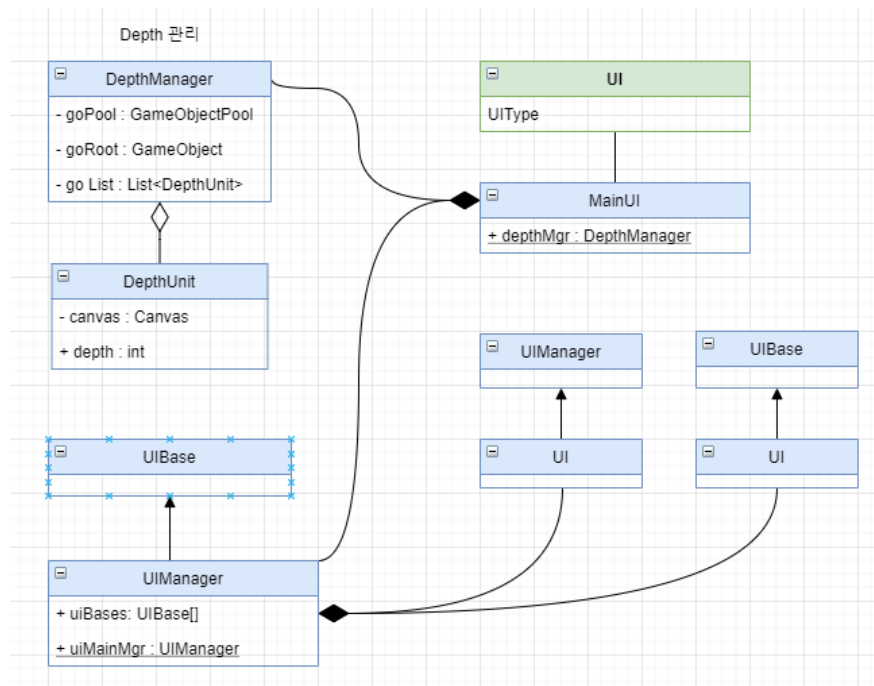
UGUI

사용 목적

유니티 프로젝트에서 UI를 표현하여 유저들에게 게임 정보를 표현하기 위하여 사용했습니다.

콘텐츠 설명

유니티 프로젝트에서 제공하는 UI로 Depth 관리를 Main Canvas밑에 자식 Canvas들을 설정하여 Sort Order를 부여하여 Depth를 관리했습니다. 그리고 NGUI와 마찬가지로 Image, Text, Button 등 다양한 UI 컴포넌트를 사용하였습니다.



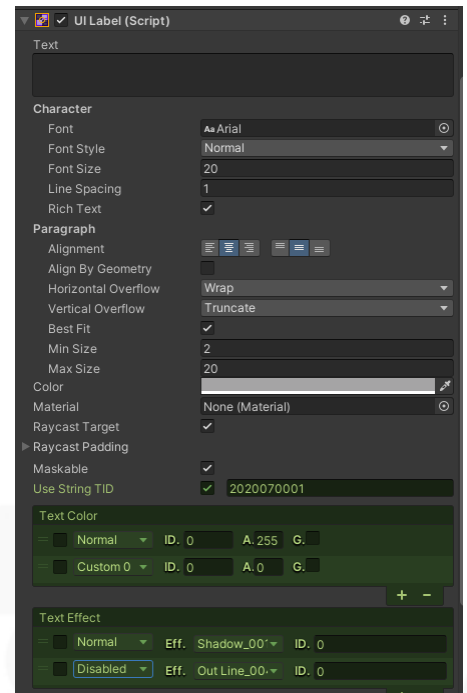
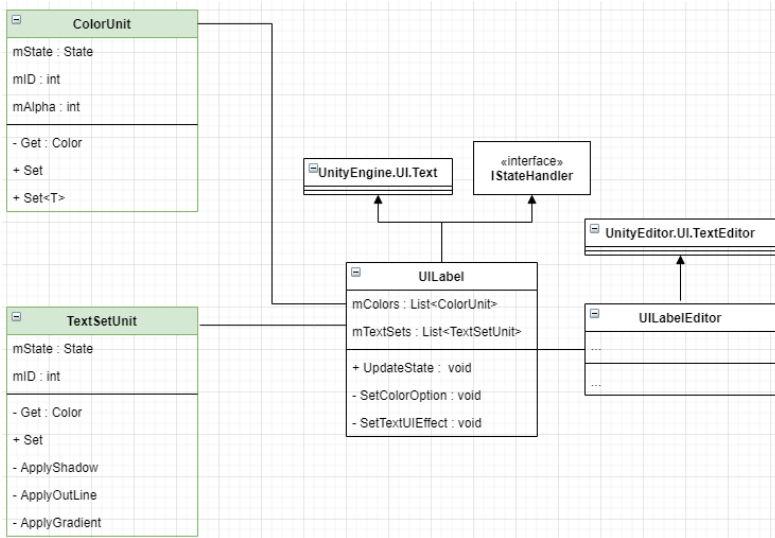
CustomUI

개발 목적

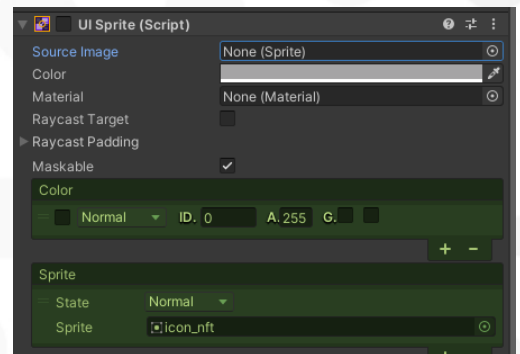
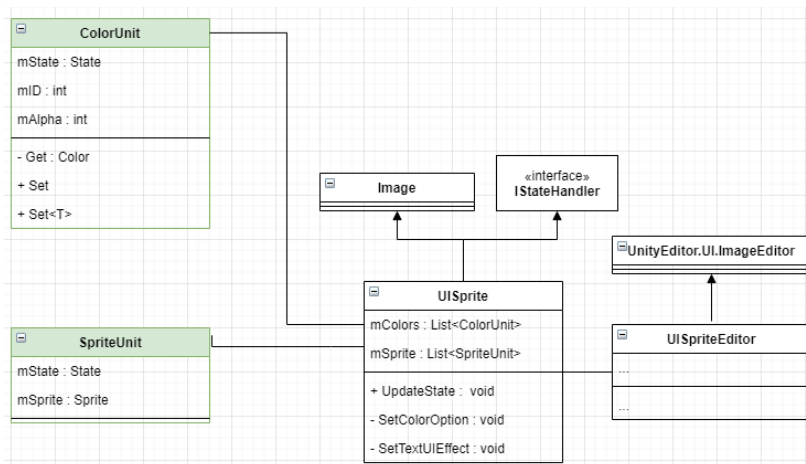
유니티에서 제공하는 UI들에 필요한 기능들을 추가하고 확장하기 위해 개발했습니다.

콘텐츠 설명

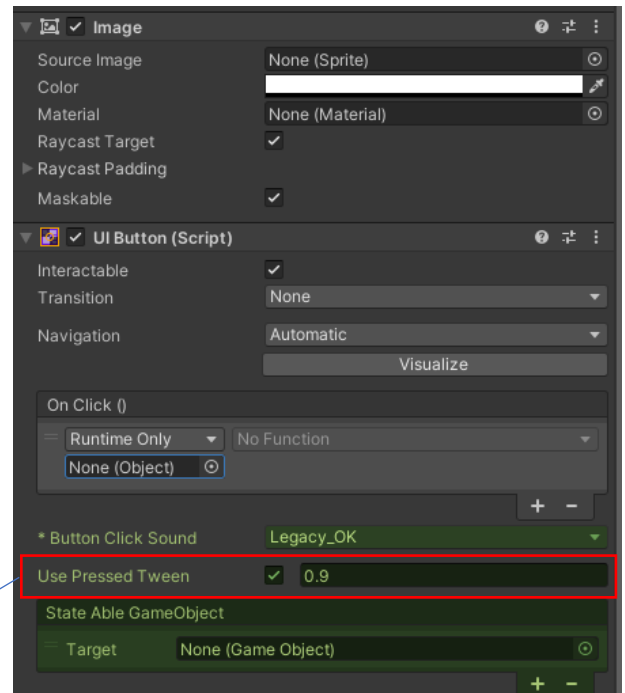
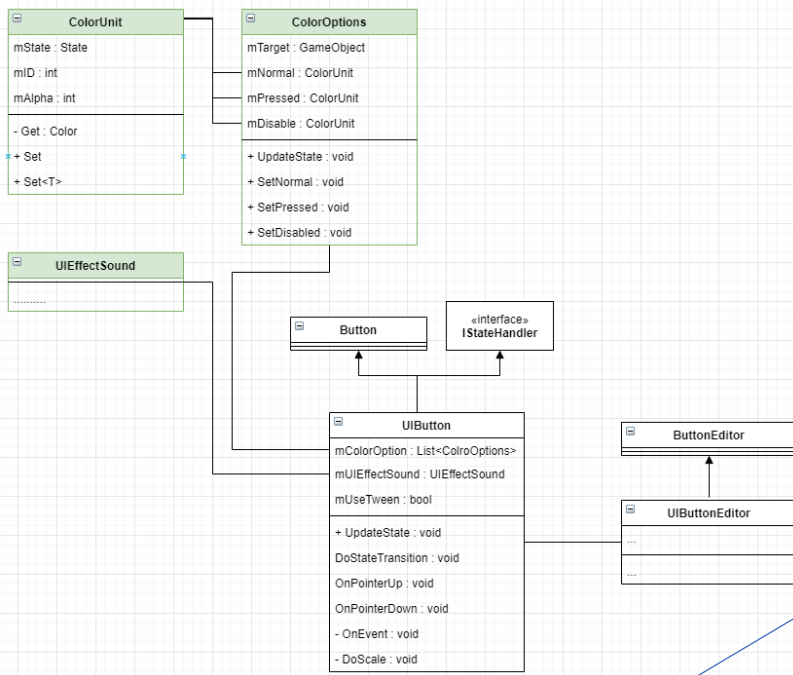
유니티에서 제공되는 UI들인 Image, Text, Button등을 확장하였으며 CustomEditor를 통해 인스펙터에서 컴포넌트들이 표현될 때 필요한 기능들을 사용할 수 있게 개발했습니다. 그리고 Unit화 하여 필요한 기능들을 결합하여 사용할 수 있도록 개발했습니다.



UILabel은 Text를 커스텀 한 것으로 State에 따라 컬러와 알파값을 조절하고 쉐도우, 아웃라인, 그라디언트 효과를 줄 수 있도록 개발했습니다.



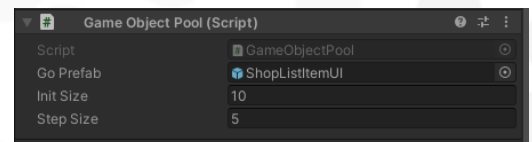
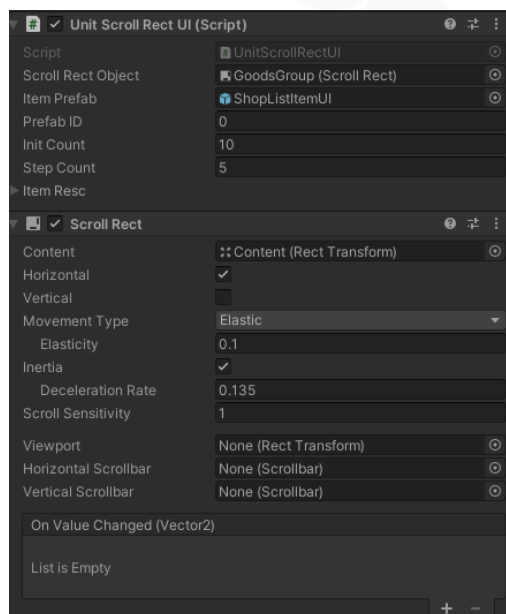
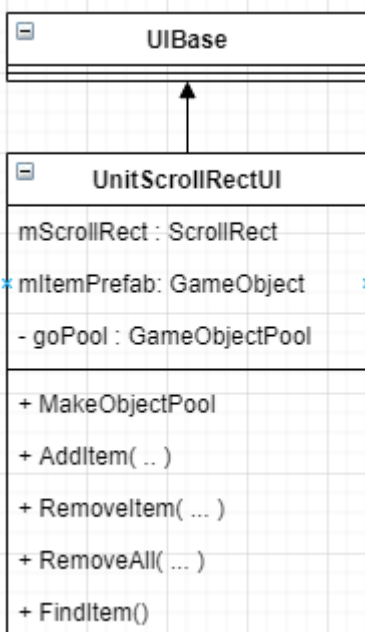
UISprite의 경우에는 컬러 값과 스프라이트 이미지를 상태에 따라 바꿀 수 있도록 개발했습니다.



```

private void DoScale( float scale )
{
    if( mUseTween == true )
    {
        transform.DOScale( new Vector3( scale, scale, 1f ), 0.1f ).SetEase( Ease.OutBack );
    }
}
  
```

UIButton의 경우에는 클릭 이벤트 시에 발생하는 사운드와 DOTween을 사용하여 스케일을 조정하는 기능을 추가했으며 버튼의 상태에 따라 설정한 UI의 State에 따라 개발한 CustomUI들이 바뀌도록 개발했습니다.



Scroll을 사용하려 할 때 오브젝트풀들과 자식 Prefab들을 인스펙터에서 지정해서 AddItem만 하면 되도록 개발했습니다.

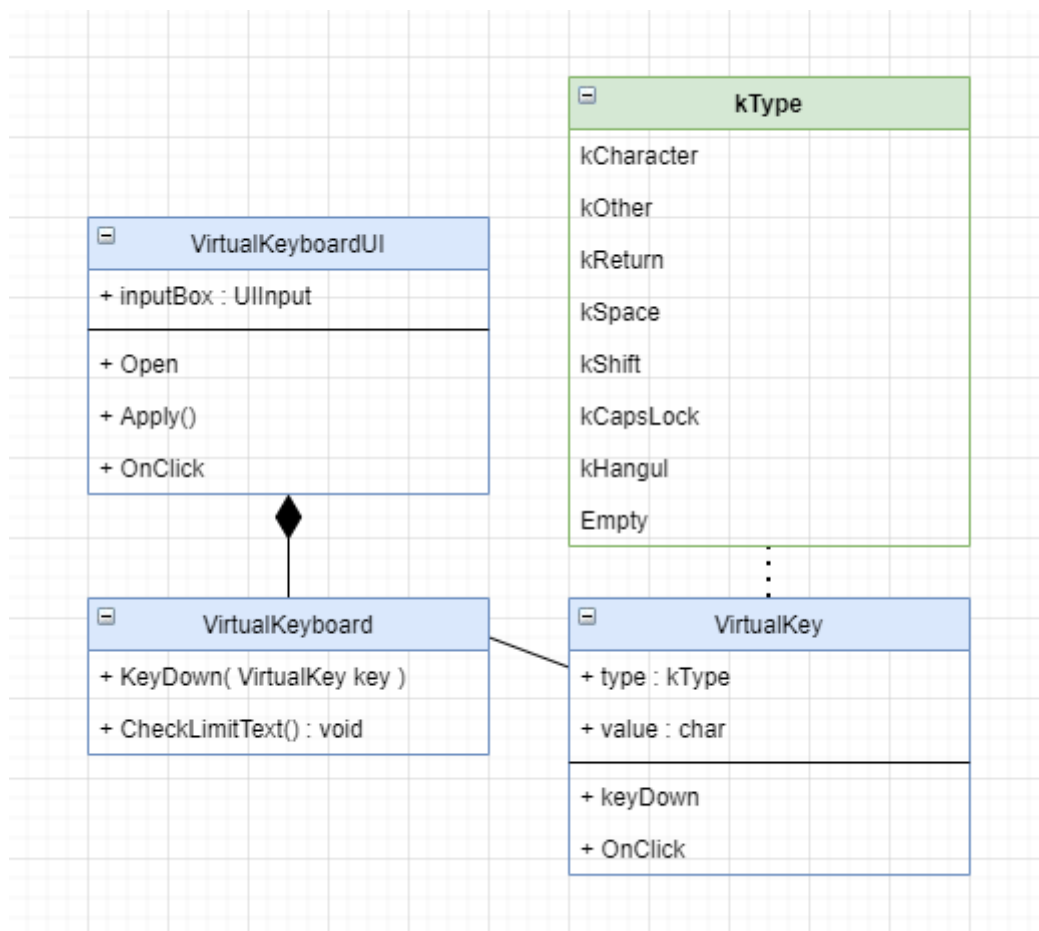
가상 키보드

개발 목적

스크린 야구 개발 시에 키오스크 터치 스크린을 통해 키입력을 하기 위해 개발했습니다.

콘텐츠 설명

키 타입을 만들어 각 키 마다 타입을 부여했으며 버튼 클릭 시 해당 키가 입력되며 UI상에 표현되도록 개발했습니다. 또한 델리게이트를 통해 엔터 클릭 시 해당 스트링을 금치어 체크를 하고 반환하도록 개발했습니다.



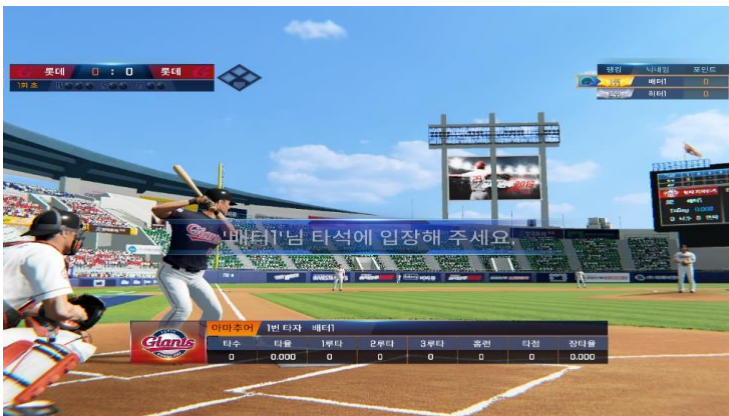
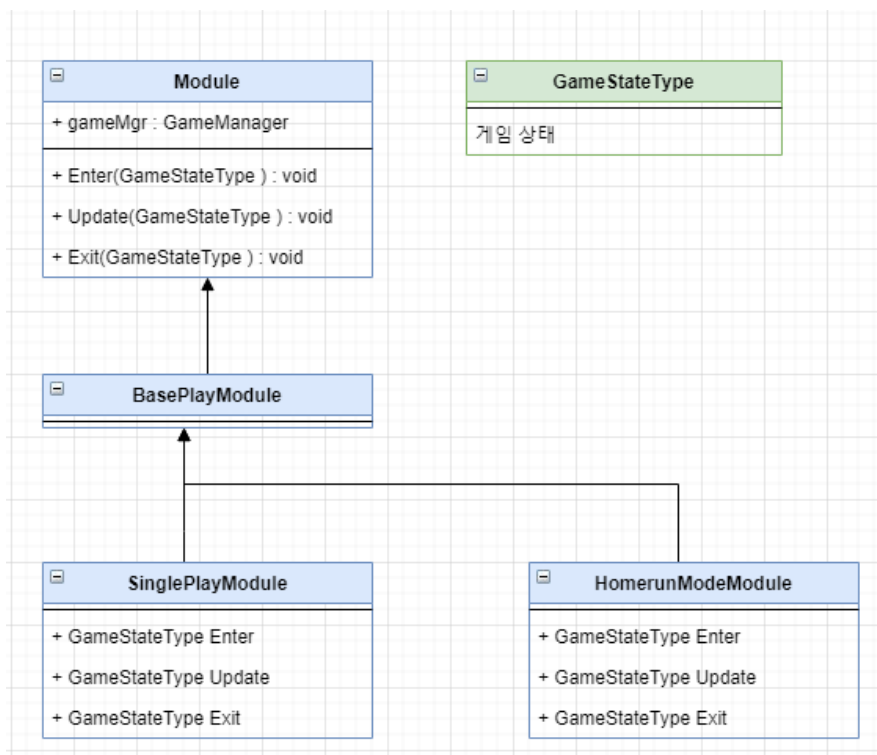
신규 게임 모드

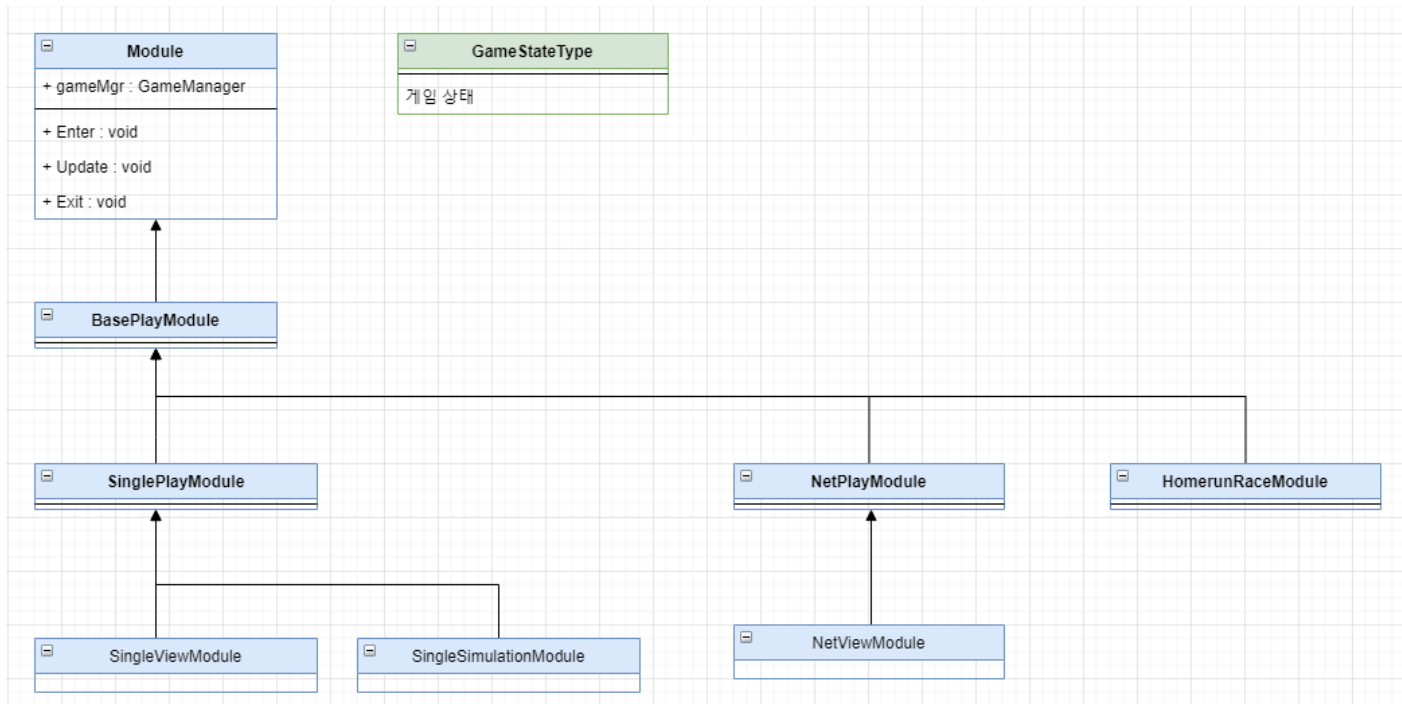
개발 목적

게임 내에서 유저들이 플레이 할 수 있는 여러 야구 게임 모드를 추가하기 위해 개발했습니다.

콘텐츠 설명

스크린 야구 프로젝트에서 싱글 모드, 홈런 모드를 개발하였고 이사만루 프로젝트에서 시뮬레이션 모드와 즉시 완료모드를 개발했습니다. 각 게임 상태를 지정해두고 Enter에서는 각 상태에서 필요한 것들을 초기화를 진행하고 Update에서는 게임의 시간을 유지하거나 진행하도록 하고 Exit에서는 다음 상태로 넘어가거나 데이터를 상태에서 초기화가 필요한 것은 초기화하고 데이터를 정리하도록 했습니다.





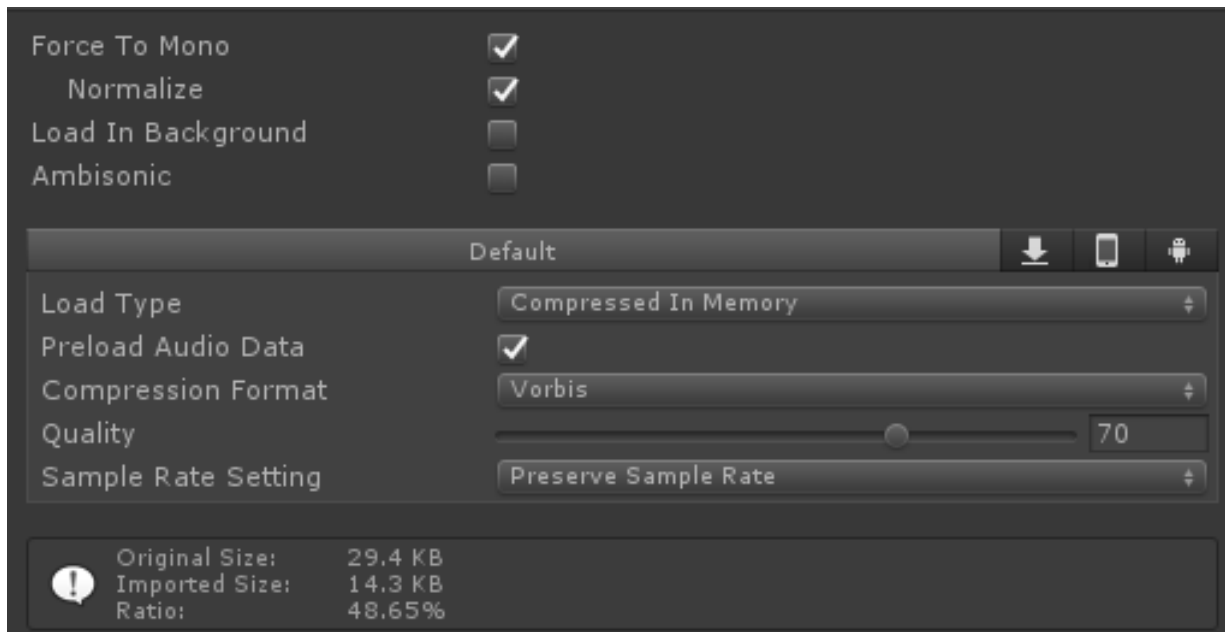
사운드

개발 목적

게임 내에서 재생되는 사운드들을 최적화하고 현재 상황을 체크하고 상황에 맞는 코멘터리나 응원 곡이 나오도록 개발했습니다.

콘텐츠 설명

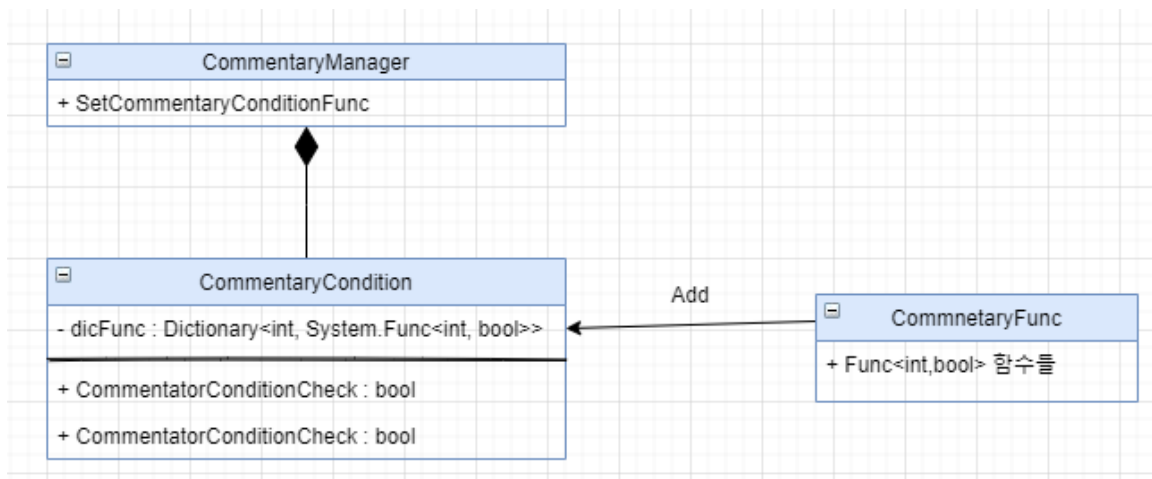
유니티에서 사운드 파일들을 Mono설정을 하고 Load Type을 사이즈가 큰 파일은 Compressed In Memory 설정을 하고 사이즈가 작은 파일은 Decompress On Load설정을 했습니다. 그리고 압축형식을 Vorbis방식을 사용하여 메모리 최적화를 했습니다. 그리고 체크 함수들을 Dictionary에 저장을 하고 상황을 체크하여 상황에 맞는 코멘터리와 응원가가 재생되도록 개발했습니다.



Textures: 1035 / 0.83 GB
 Meshes: 181 / 35.9 MB
 Materials: 4141 / 9.0 MB
 AnimationClips: 1841 / 0.53 GB
 AudioClips: 388 / 28.4 MB
 Assets: 41150
 GameObjects in Scene: 954
 Total Objects in Scene: 28404
 Total Object Count: 69554
 GC Allocations per Frame: 42237 / 5.0 MB

Textures: 1137 / 0.85 GB
 Meshes: 179 / 35.8 MB
 Materials: 3989 / 8.7 MB
 AnimationClips: 1841 / 0.53 GB
 AudioClips: 377 / 16.2 MB
 Assets: 40017
 GameObjects in Scene: 958
 Total Objects in Scene: 27849
 Total Object Count: 67866
 GC Allocations per Frame: 43846 / 4.8 MB

사운드 최적화를 해서 오디오 클립 메모리를 12MB 정도 최적화를 했습니다. 10회 반복하였을 때 평균적으로 11MB정도의 메모리를 최적화했습니다.



```

public virtual void SetCommentaryConditionFunc()
{
    condition.SetConditionFunc( (int)CommentatorFuncType.Random, > > > > CommentaryFunc.ConditionRandom );
    condition.SetConditionFunc( (int)CommentatorFuncType.MatchType, > > > > CommentaryFunc.ConditionMatchType );
    condition.SetConditionFunc( (int)CommentatorFuncType.PreInningScored, > > > > CommentaryFunc.ConditionPreInningScored );
    condition.SetConditionFunc( (int)CommentatorFuncType.OverTime, > > > > CommentaryFunc.ConditionOverTime );
    condition.SetConditionFunc( (int)CommentatorFuncType.GameResult, > > > > CommentaryFunc.ConditionGameResult );
    condition.SetConditionFunc( (int)CommentatorFuncType.OutCount, > > > > CommentaryFunc.ConditionOutCount );
    condition.SetConditionFunc( (int)CommentatorFuncType.Runner, > > > > CommentaryFunc.ConditionRunner );
}

```

```

public partial class CommentaryFunc
{
    > static public bool ConditionRandom( int param )
    > {
    >     return MatchUtility.CheckRandomPer100( param );
    > }
    > static private bool RoundScore( int a, int b )...
    > static public bool ConditionMatchType( int param )...
    > static public bool ConditionStadium( int param )...
    > static public bool ConditionPreInningScored( int param )...
    > static public bool ConditionOverTime( int param )...
    > static public bool ConditionGameResult( int param )...
    > static public bool ConditionOutCount( int param )...
    > static public bool ConditionRunner( int param )...
    > static public bool ConditionBatterAdmission( int param )...
    > static public bool ConditionThreeBA_OrMore( int param )...
    > static public bool ConditionHomeRunBefore( int param )...
}

```

조건 함수들을 저장해두고 사운드 재생할 때 마다 체크하여 원하는 코멘터리와 응원가가 재생되도록 개발했습니다.

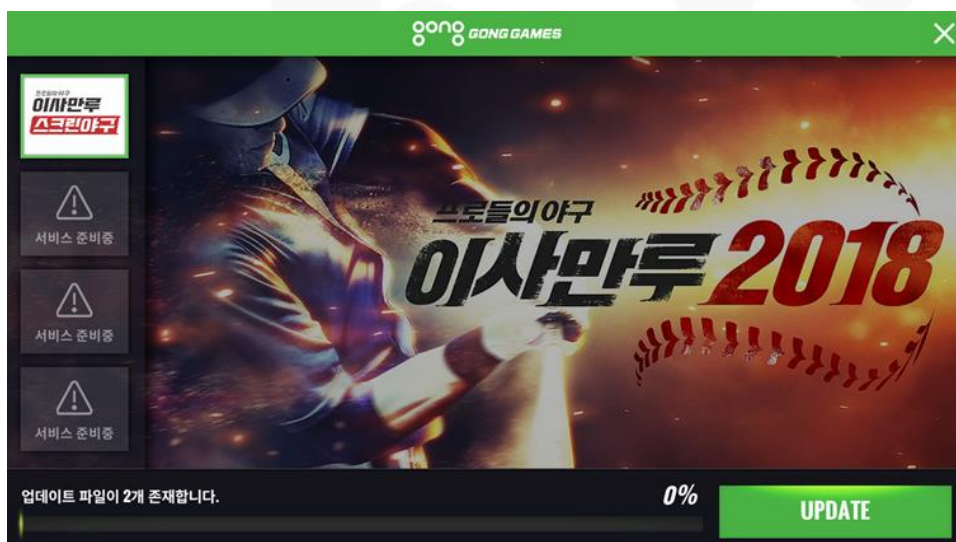
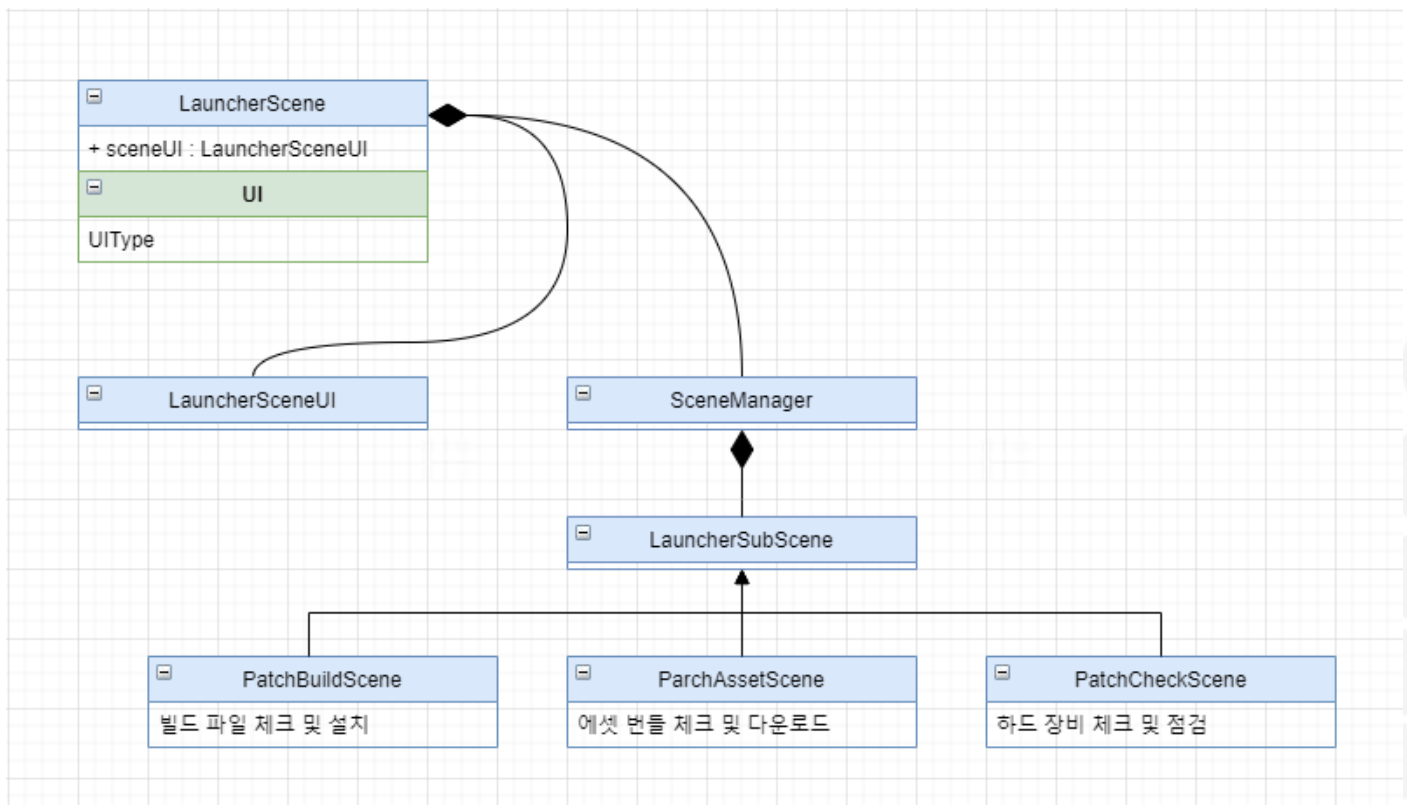
런처

개발 목적

스크린 야구 프로젝트에서 매장별로 실행파일과 번들을 관리하고 하드웨어 장비들을 점검하기 위해 개발했습니다. .

콘텐츠 설명

유니티로 개발을 했으며 전체화면이 아닌 창화면으로 개발을 했으며 매장별로 DB에 정한 버전에 있는 실행파일과 에셋번들을 다운로드 받고 센서와 피칭 머신의 신호를 체크하여 고장 여부와 정상적으로 실행되었는지 체크하도록 했습니다.



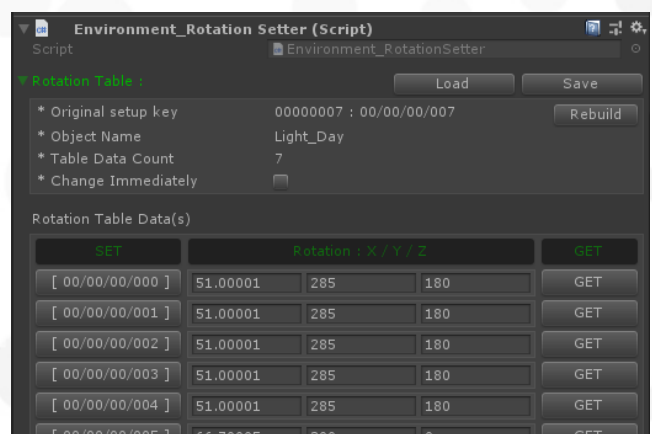
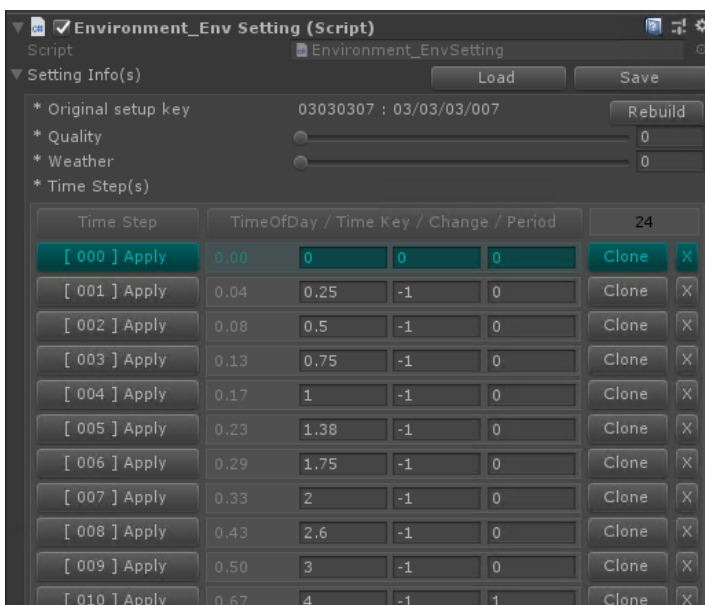
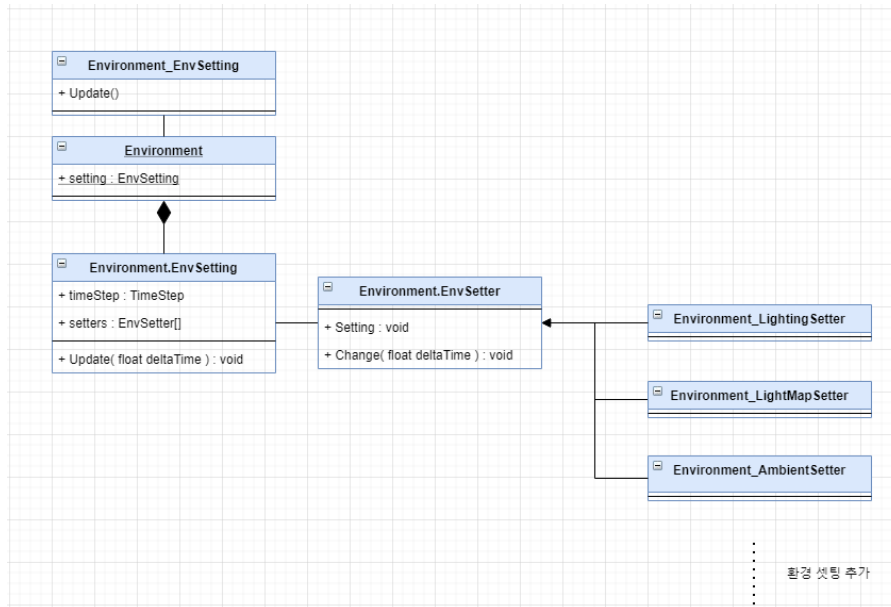
그래픽 환경 툴

개발 목적

시간의 흐름(야구의 이닝)에 따라 구장의 환경정보들이 바뀌는 것을 목표로 개발했습니다.

콘텐츠 설명

구장 별로 이닝의 흐름에 따라 구장의 환경설정(Light, Shadow, Ambient, Emission, Bloom, Skybox) 등을 이닝 별로 설정한 값으로 바뀌도록 했고 오브젝트의 액티브로 켜지고 꺼지도록 개발을 했습니다.



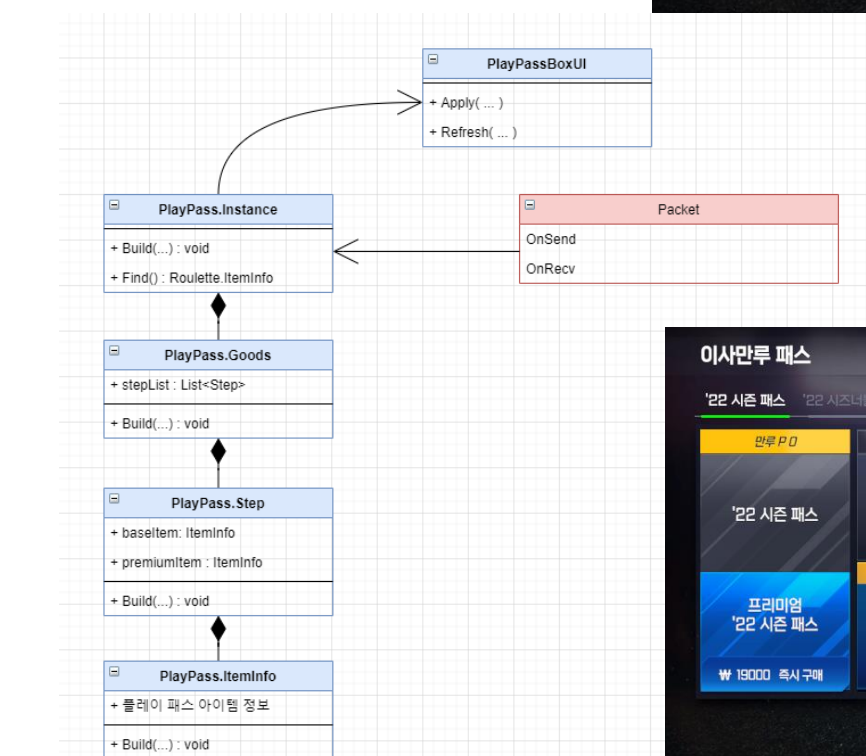
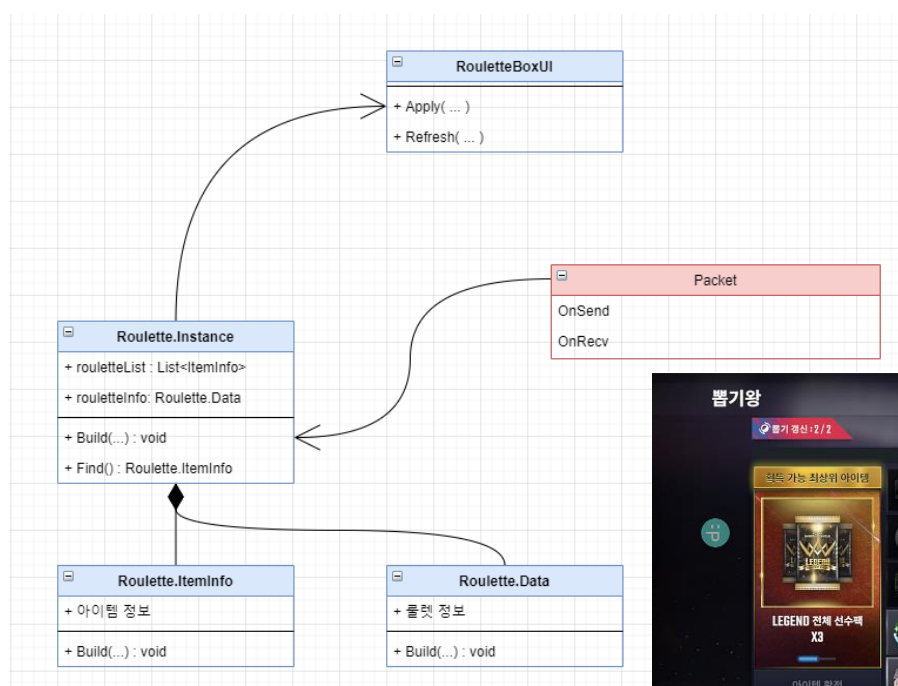
이사만루 콘텐츠 추가

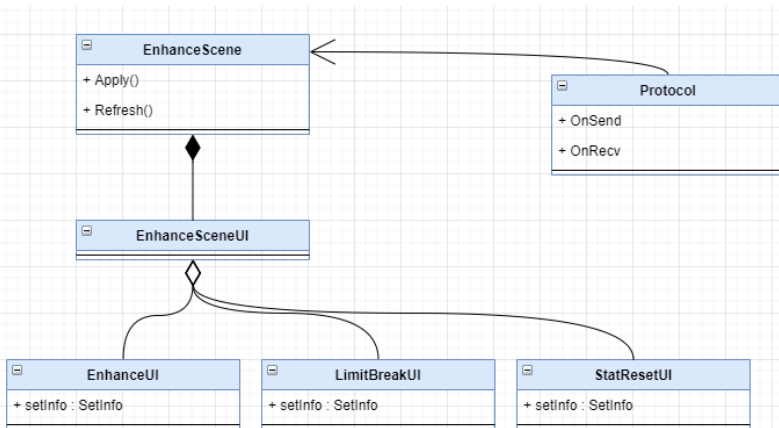
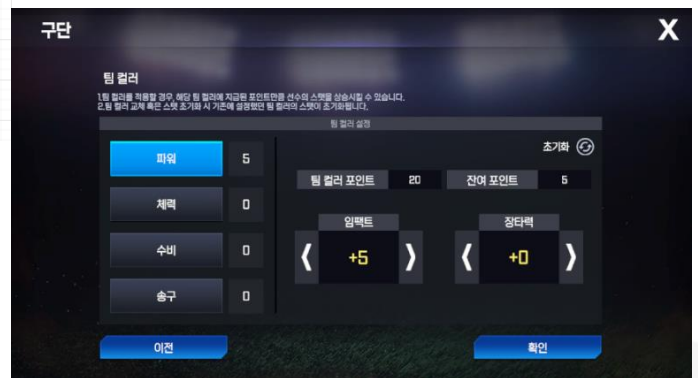
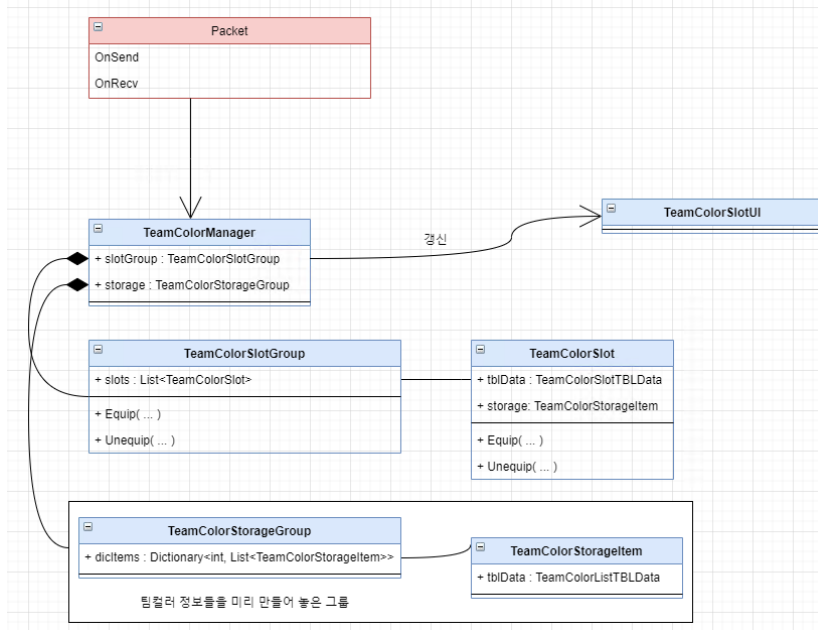
개발 목적

이사만루 프로젝트의 유저들이 플레이하고 보상을 얻을 수 있는 콘텐츠들을 추가하기 위해 개발했습니다.

콘텐츠 설명

이사만루에서 룰렛, 플레이 패스, 타이틀 랭커 등 유저들이 보상을 얻을 수 있는 콘텐츠를 개발하였고 유저들의 편의성을 위한 라인업 프리셋을 개발하였고 유저들이 게임의 재미를 위한 선수 강화, 추가 스텟, 한계 돌파 등을 개발을 했습니다.





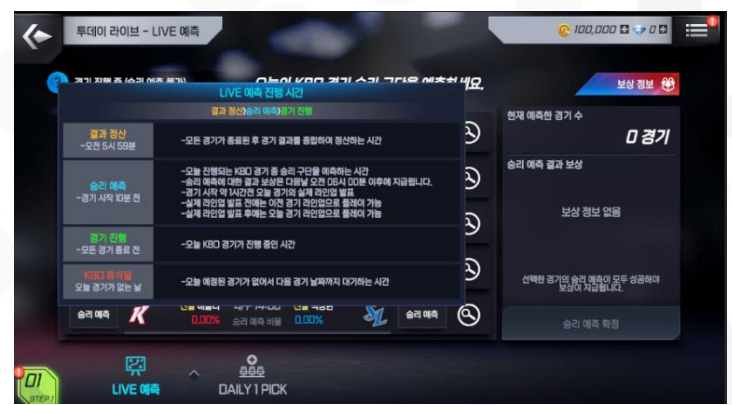
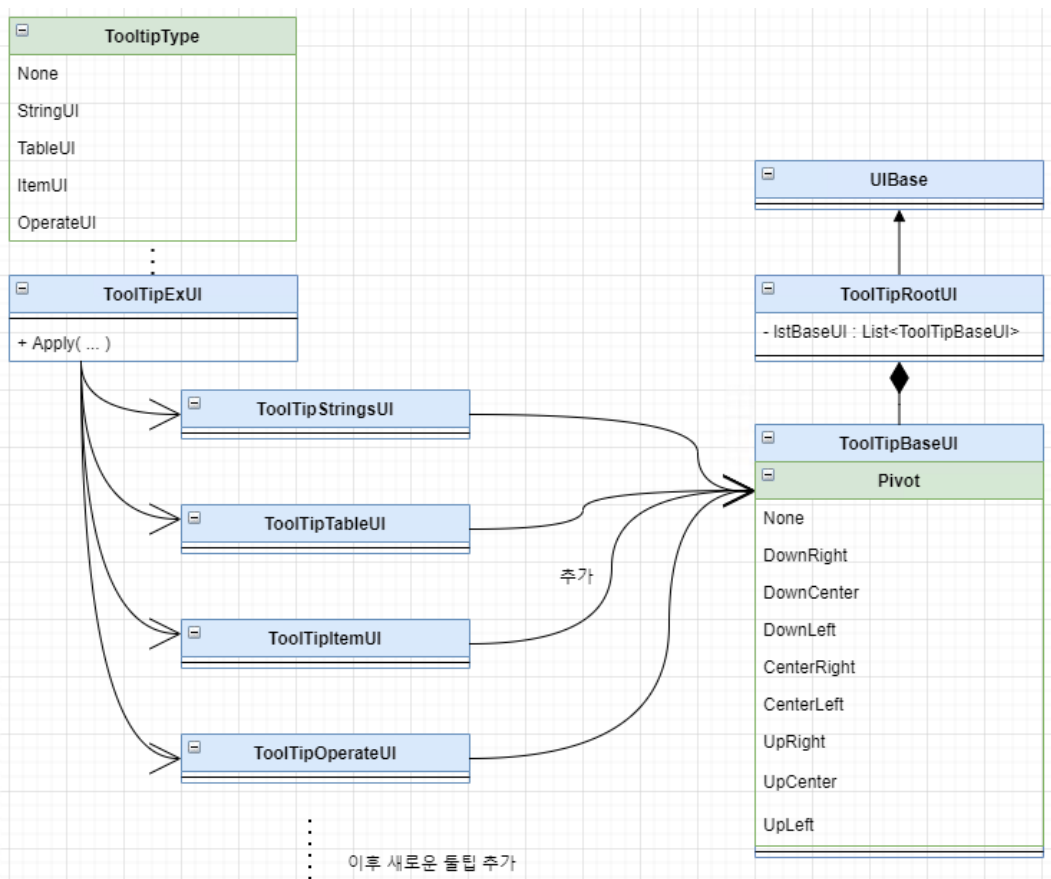
ToolTip

개발 목적

이사만루에서 툴 팁을 다양하게 표현을 할 수 있도록 Component화 했습니다.

콘텐츠 설명

ToolTipExUI 를 사용하여 원하는 ToolTip의 Type을 만들어 높은 Depth를 가진 ToolTipRootUI에 생성하여 ToolTip을 표현하고 위치를 스크린 좌표를 기준으로 계산하여 원하는 Pivot을 설정하여 표현하도록 개발 했습니다.



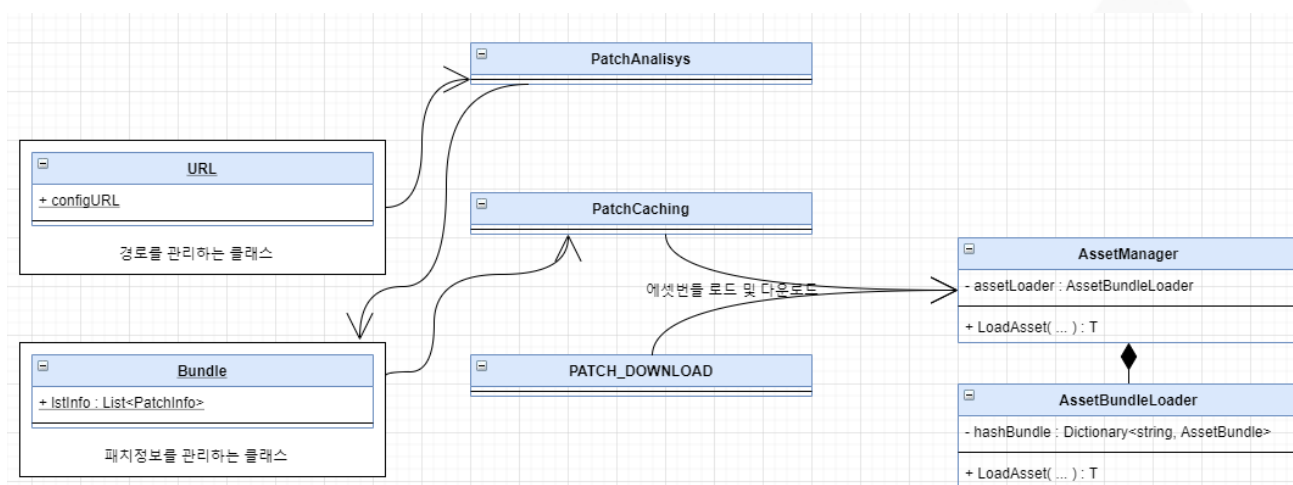
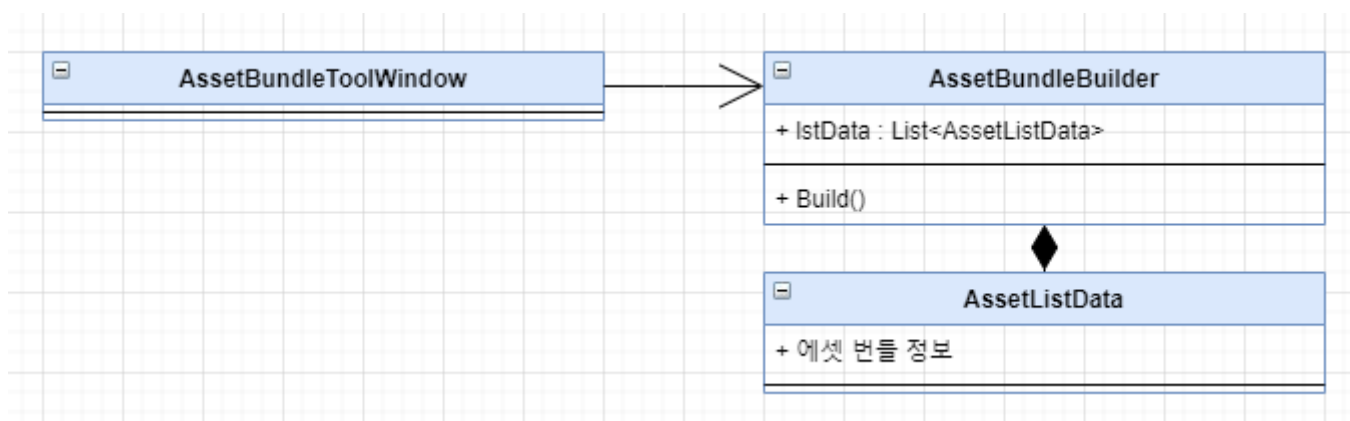
AssetBundle

개발 목적

빌드 사이즈를 줄이고 리소스들을 에셋 번들로 관리하여 패치가 가능하도록 개발했습니다.

콘텐츠 설명

정한 경로의 리소스들을 에셋 번들을 리스트하고 해당 리스트들을 CRC체크와 Hash체크를 하여 패치를 하여 번들 패치를 가능하도록 개발했습니다. 또한 에셋 번들을 FTP/CDN서버에 업로드하여 리소스를 관리되도록 했습니다.



Tool					
인덱스	타입	isStatic	경로		
✓1	1 2 3	✓	Assets/Resources/AdMob	admob	
✓2	✓1 2 3		Assets/Resources/Player/Animator/Directing	player_animat	
✓3	✓1 2 3		Assets/Resources/Player/Animator/Directing01	player_animat	
✓4	✓1 2 3		Assets/Resources/Player/Animator/Directing02	player_animat	
✓5	✓1 2 3		Assets/Resources/Player/Animator/Directing03	player_animat	
✓6	✓1 2 3		Assets/Resources/Player/Animator/Directing04	player_animat	
✓7	✓1 2 3		Assets/Resources/Player/Animator/DirectingMesh	player_animat	
✓8	✓1 2 3		Assets/Resources/Player/Animator/PlayerCommon/Coach	player_animat	

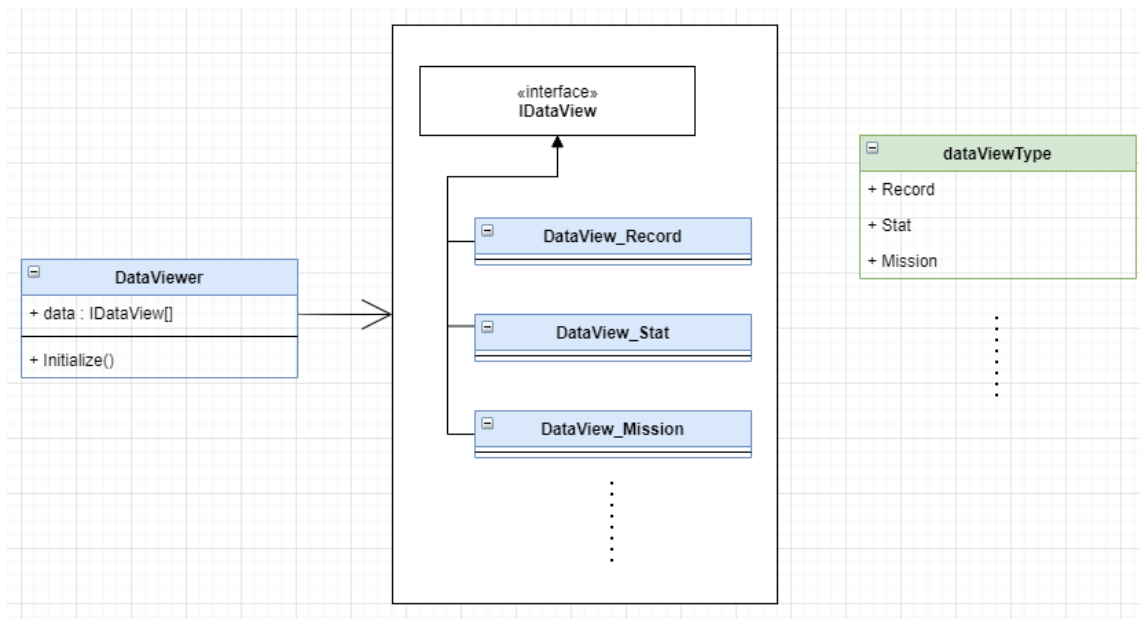
데이터 뷰어 툴

개발 목적

게임 QA를 하거나 필요한 정보를 확인해야할 때 편하게 개발을 하기위해 개발했습니다.

콘텐츠 설명

유니티에서 제공하는 CustomEditor를 사용하여 Menu에 툴을 제작하였습니다. DataView 클래스에서 선택한 툴들이 보여지고 제작한 툴은 선수 기록 툴, 선수 스렛툴, 미션툴, 사운드 툴 등 여러 툴을 만들었습니다.

[illegible]

	Player	Status	HOME	AWAY	BATTER	PITCHER
1	정윤환					선박하기
2	최명훈					선박하기
3	박크현					선박하기
4	노시환					선박하기
5	려주석					선박하기
6	김태현					선박하기
7	이범근					선박하기
8	김지승					선박하기
9	노수영					선박하기

타이머

개발 목적

게임 내 시간의 흐름을 체크하고 표현하기 위한 데이터를 추가했습니다.

콘텐츠 설명

시간을 설정하여 체크하는 구조체를 만들어 활성화하면 현재 남은 시간이나 날짜를 확인할 수 있도록 했습니다. 그리고 타이머가 작동했을 때와 종료했을 때를 코루틴을 활용하여 RemainTimer 클래스를 개발했습니다.

```
public struct ExpireTimer
{
    > private DateTime _expireTime;
    > public DateTime expireTime => { get { return _expireTime; } }
    > public TimeSpan remainTime => { get { return (_expireTime - DateTime.Now); } }
    > public bool isExpire => { get { return (_expireTime <= DateTime.Now) ? true : false; } }

    > public int days => { get { return (isExpire == true) ? 0 : remainTime.Days; } }
    > public int hours => { get { return (isExpire == true) ? 0 : remainTime.Hours; } }
    > public int minutes => { get { return (isExpire == true) ? 0 : remainTime.Minutes; } }
    > public int seconds => { get { return (isExpire == true) ? 0 : remainTime.Seconds; } }
    > public double totalDays => { get { return remainTime.TotalDays; } }
    > public double totalHours => { get { return remainTime.TotalHours; } }
    > public double totalMinutes => { get { return remainTime.TotalMinutes; } }
    > public double totalSeconds => { get { return remainTime.TotalSeconds; } }

    > public ExpireTimer(int seconds) => { _expireTime = System.DateTime.Now + new TimeSpan(0, 0, seconds); }
    > public ExpireTimer(float seconds) => { _expireTime = System.DateTime.Now + new TimeSpan((long)(seconds * 10000000)); }
    > public ExpireTimer(TimeSpan span) => { _expireTime = System.DateTime.Now + span; }
    > public ExpireTimer(double endDate) => { _expireTime = System.DateTime.FromOADate(endDate); }
}
```

```
public class RemainTimer
{
    > private ExpireTimer expireTimer;
    > private System.Action onExpire = null;
    > private System.Action<ExpireTimer> onUpdate = null;

    > private IEnumerator coRemainTimer = null;
    > private MonoBehaviour mono = null;
    > private WaitForSeconds waitTime = new WaitForSeconds(0.5f);
}
```

```
public void Apply(MonoBehaviour behaviour, ExpireTimer timer)
{
    this.mono = behaviour;
    this.expireTimer = timer;
    this.onExpire = _onExpire;
    this.onUpdate = _onUpdate;

    if (null != coRemainTimer)
    {
        mono.StopCoroutine(coRemainTimer);
        coRemainTimer = null;
    }

    if (expireTimer.isExpire)
    {
        return;
    }

    coRemainTimer = CoRemainTimer();
    mono.StartCoroutine(coRemainTimer);
}
```

```
IEnumerator CoRemainTimer()
{
    while(true)
    {
        if(expireTimer.isExpire)
        {
            onExpire?.Invoke();
            yield break;
        }

        onUpdate?.Invoke(expireTimer);

        yield return waitTime;
    }
}
```

WorldUI

개발 목적

이사만루에서 월드에 존재하는 선수들의 머리 위에 UI를 표기하기 위해 개발했습니다.

콘텐츠 설명

선수들의 월드 좌표가 카메라 뷰에 있으면 활성화되도록 했으며 해당 좌표를 스크린 좌표로 바꾸어 표현하도록 개발했습니다.

```
Vector3 position = target.playerObject.GetNameCardPosition();

if( false == Match.Instance.gameMgr.gameCamera.isActiveName
    || Match.GameUtil.IsInGameCamera(position) )
{
    → SetActive(false);
    → return;
}

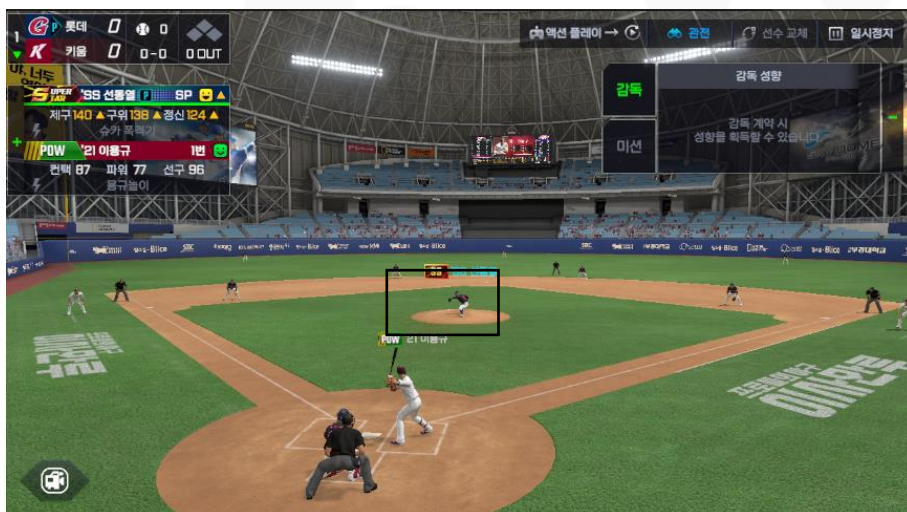
SetActive(true);

Vector3 uiPosition = Match.GameUtil.GameWorldToScreen(position);

obj.transform.position = uiPosition;
```

```
static public Vector2 GameWorldToScreen(Vector3 position)
{
    → if( gameCam == null )
    →     return Vector2.zero;

    → return gameCam.WorldToScreenPoint(position);
}
```



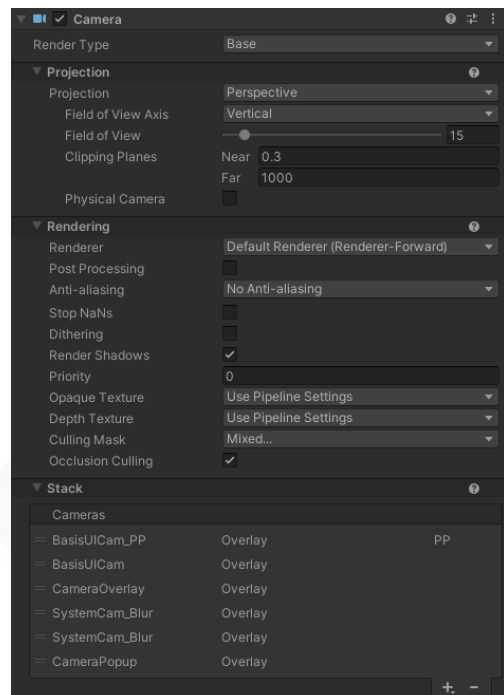
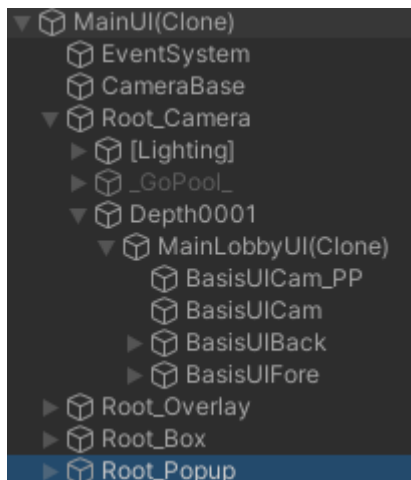
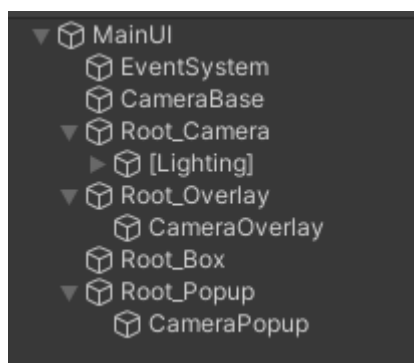
Camera Stack

개발 목적

카메라마다 특정 그래픽 효과를 주기 위해 여러 카메라를 사용하기 위해 개발했습니다.

콘텐츠 설명

유니티의 새로운 렌더링 파이프라인인 URP를 사용했습니다. 캔버스에다가 카메라를 계층화 하여 여러 카메라를 나누어 사용을 하고 해당 카메라에 원하는 Lighting 효과를 주거나 Bloom 효과를 주기도 했습니다. 프리팹을 인스턴싱 할 때 내부에 카메라가 있으면 체크해서 카메라 스택에 특정 카메라를 제외하고는 누적되도록 개발했습니다.



```
static public void AddCameraStack( eUIRootType rmType, Camera cam, int index )
{
    if( null == cam || cam.GetComponent<UniversalAdditionalCameraData>().renderType != CameraRenderType.Overlay )
    {
        return;
    }

    UniversalAdditionalCameraData camData = baseCamera.GetComponent<UniversalAdditionalCameraData>();
    camData.cameraStack.Remove( cam );
    if( rmType == eUIRootType.Popup )
    {
        camData.cameraStack.Insert( camData.cameraStack.Count - 1, cam );
    }
    else if( rmType == eUIRootType.Camera || rmType == eUIRootType.Box )
    {
        if( false == cam.name.Contains( "_PP" ) )
        {
            cam.orthographicSize = 360;
            for( int i = 0; i < camData.cameraStack.Count; ++i )
            {
                Camera camera = camData.cameraStack[ camData.cameraStack.Count - 1 - i ];
                if( null != camera && camera.name.Contains( "Overlay" ) )
                {
                    if( rmType == eUIRootType.Camera )
                    {
                        camData.cameraStack.Insert( camData.cameraStack.Count - 1 - i, cam );
                    }
                    else
                    {
                        camData.cameraStack.Insert( camData.cameraStack.Count - i + index, cam );
                        break;
                    }
                }
            }
        }
    }
}
```

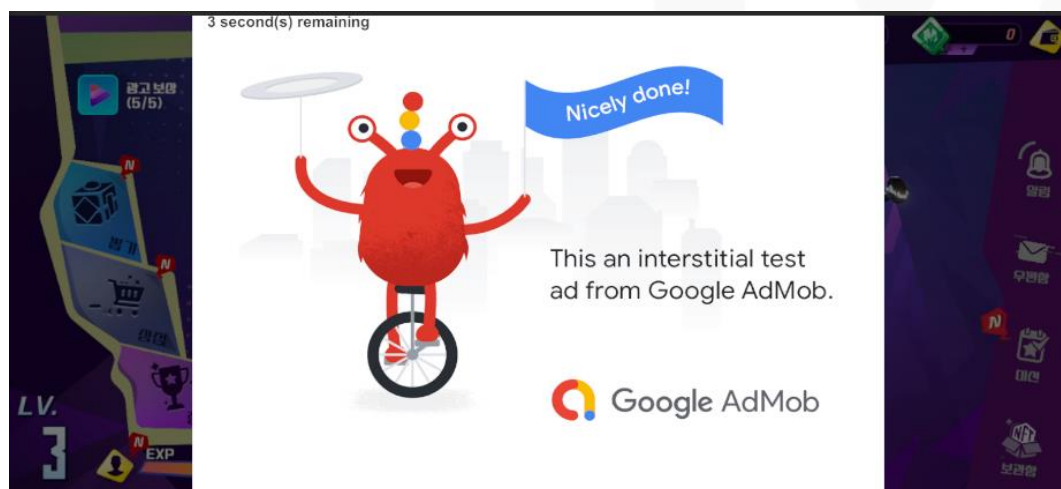
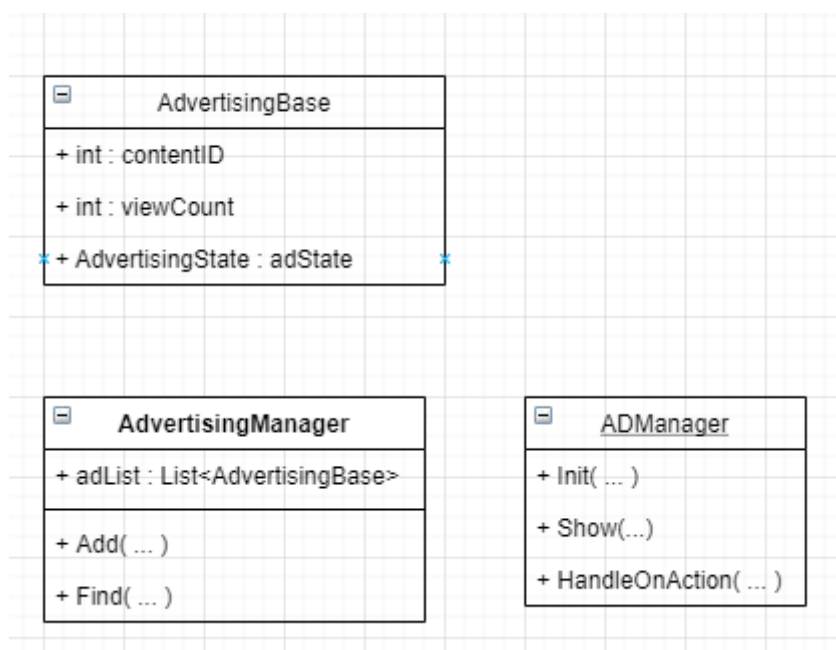

구글 광고

개발 목적

광고를 통해 게임의 매출을 조금 더 올리기 위해 개발을 했습니다.

콘텐츠 설명

구글에서 제공하는 GoogleMobileAds SDK플러그인을 추가했으며 광고들마다 쿼타임과 이벤트들을 관리하기 위해 ADManager를 개발했습니다. 플랫폼마다 광고 Key를 설정하고 해당 광고를 보게 나면 보상을 획득하도록 개발했습니다.
























Jenkins 자동 빌드

개발 목적

빌드를 조금 더 효율적으로 하기 위해 구축했으며 젠킨스를 사용하여 유니티 AOS, IOS빌드를 하기 위해 개발했습니다.

콘텐츠 설명

Jenkins를 이용하여 빌드를 특정 시간마다 되거나 명령으로 유니티에서 빌드를 하고 보안 모듈을 묶는 배치 파일을 실행하도록 구축했습니다. 여기서는 Shell Scripts와 Python을 사용하였으며 플랫폼별로 잡을 생성하여 관리했습니다. .

All	+					
S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	
		Branch Server Restart	16 hr - #658	1 mo 12 days - #547	41 sec	
		KBO3_ALL-IN-ONE_BUILD	8 hr 44 min - #351 ▼	2 mo 6 days - #279	17 sec	
		KBO3_AOS_TRUNK	14 days - #579	6 days 21 hr - #587	19 min	
		KBO3_AOS_TRUNK-pipeline	8 hr 44 min - #40	—	12 min	
		KBO3_ONE_TRUNK	11 days - #73	7 days 8 hr - #75	27 min	
		KBO3_ONE_TRUNK-pipeline	8 hr 44 min - #134	—		
		Trunk Server Restart	16 hr - #1148	11 days -		
		UNIT_TEST	1 mo 19 days - #157	18 hr - #1		

[illegible]

감사합니다.

이름 : 권오현

번호 : 010.2719.5330

이메일 : ohhyun5442@gmail.com
