펀앤핏 엑서보드 베이스 **DOCS**

최초작성 20230316 ㅅㅇㅎ

목차

설치

기능 소개 및 사용

- 간단한 문자열 포맷 이벤트 핸들러
- 씬 페이드 오브젝트
- 게임 빌드버전 디버그 로그 리포터
- 엑서 보드와의 호환 및 키맵
- 블루투스 연결 씬
- 엑서보드 연결신호 스크립트 수신
- 키스토어 제작 및 구글 마켓용 앱 번들 빌드
- 메인화면 플레이어 버튼 기능
- 랭킹화면 기능
- 랭킹화면 기능을 사용하는 다른 방법
- 랭킹화면의 생명주기와 코드 호출

문제 해결

- 프리징 문제 해결
- 앱 용량 150메가 해결
- 안드로이드 환경에서의 화면 회전 옵션

설치

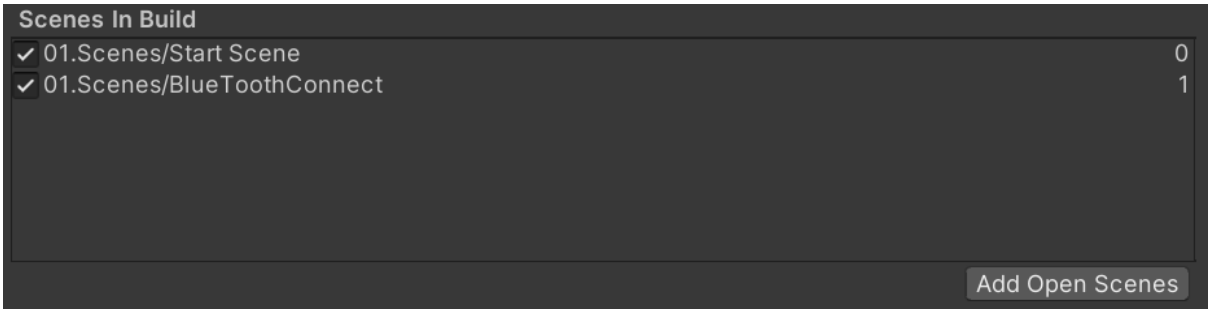
~~현 베이스 패키지의 유니티 버전은 2020.3.2f1을 기반으로 합니다.~~
구글 플레이스토어 업로드를 위해 버전을 2020.3.47f1버전으로 업그레이드했습니다.

먼저 새로운 프로젝트를 생성합니다.
package manager 탭에서 Unity registry의 input system을 설치합니다.

엑서 보드의 입력 세팅을 위해 유니티 인풋시스템 원본 파일을 찾습니다.
원본 파일 중 수정이 필요한 파일은 AndroidGameController.cs입니다.
2023.03.08 기준 해당 파일의 위치는 다음과 같습니다.
C:\Users\유저이름\AppData\Local\Unity\cache\packages\packages.unity.com\com.unity.inputsystem@1.0.2\InputSystem\Plugins\Android

AndroidGameController.cs에서 leftStickPress, rightStickPress를 검색하여 해당 코드 라인의 밑줄 내용을 다음과 같이 수정합니다.
[InputControl(name = "leftStickPress", bit = (uint)AndroidKeyCode.ButtonL2, ~
[InputControl(name = "rightStickPress", bit = (uint)AndroidKeyCode.ButtonR2, ~

베이스 패키지를 설치한 후, 씬을 다음 순서에 맞게 배치합니다.



게임 콘텐츠 등 다른 씬은 2번부터 시작하게 됩니다.

해당 설치 과정을 올바르게 진행하였다면 전술한 기능을 사용할 수 있습니다.

기능 소개 및 사용

간단한 문자열 포맷 이벤트 핸들러
MonoBehaviour의 역할을 하는 컴포넌트를 대상으로 합니다.
MonoBehaviour 대신 BasicBroadcastHandler를 상속시킵니다.
컴포넌트명 *: BasicBroadcastHandler*

public static BasicBroadcastHandler StartListen(BasicBroadcastHandler handler)
이벤트를 받기 시작합니다.
```
Start
{
  StartListen(this);
}
```

public static void StopListen(BasicBroadcastHandler handler)
이벤트를 더이상 받지 않습니다.
```
OnDestroy()
{
   StopListen(this);
}
```

protected abstract void ListenEvent(List<string> commands)
이벤트를 수신할 때의 동작을 정의합니다.
```
protected override void ListenEvent(List<string> commands)
{
  동작
}
```

public bool listening
현재 컴포넌트가 이벤트를 수신하는지 확인합니다.
if(listening) { }

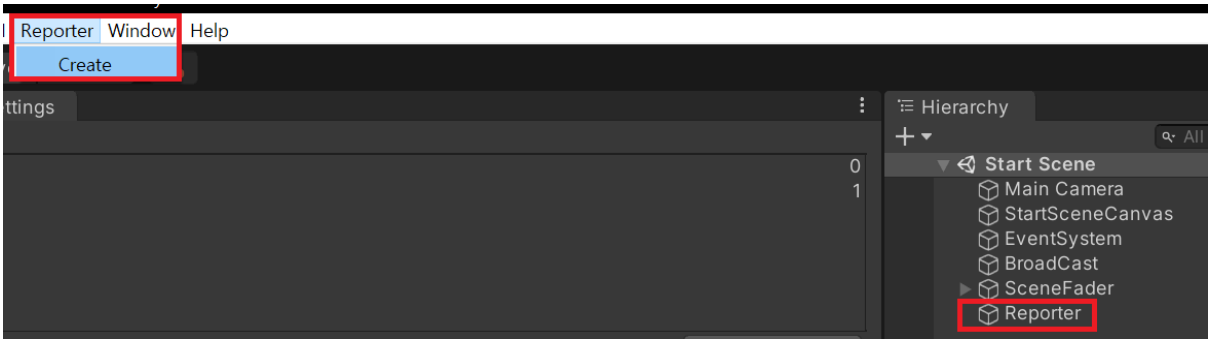
public static void Broadcast(List<string> commands)
이벤트를 송신합니다.
List<string> commands = 명령어*;*
Broadcast(commands);

public string oneCommand
이벤트를 string 1개로 송신합니다.
*// Broadcast()*에 *"*명령어*"*를 넣은것과 동일
oneCommand = *"*명령어*"*

씬 페이드 오브젝트
0번 씬에서 SceneFader가 존재한다면, 다음 코드를 통해 씬의 이동이 가능합니다.
List<string> command = new List<string>();
command.Add(Commands.GoToScene.ToString());
command.Add("이동 씬 번호");
command.Add(FadeType.옵션.ToString());
BasicBroadcastHandler.Broadcast(command);

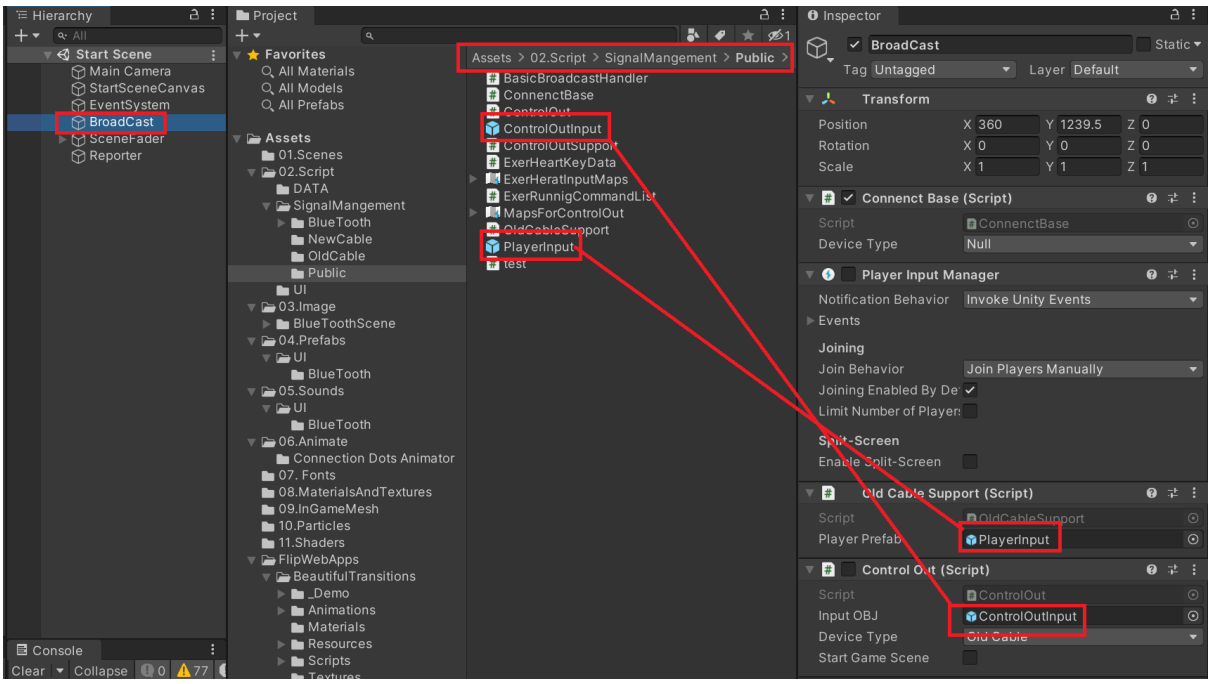
FadeType에서의 옵션은 다음과 같습니다.
Black
화면을 검은색으로 칠한 상태로 이동
CaptureToTransparent
이동 전 화면이 천천히 이동 후 화면으로 변경

게임 빌드버전 디버그 로그 리포터

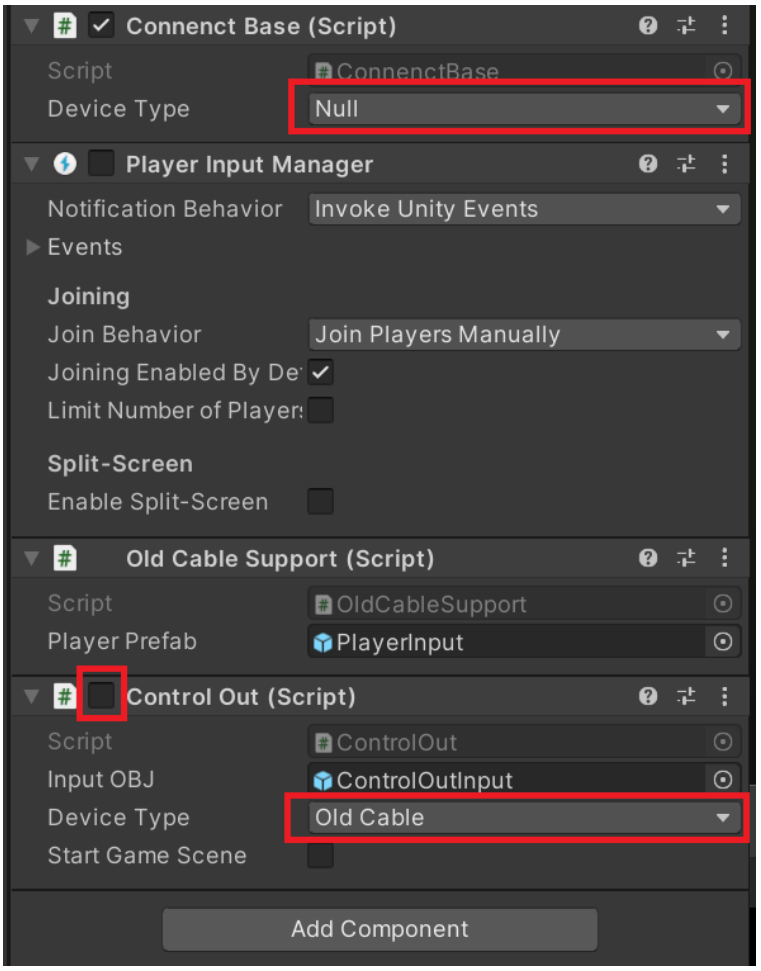


0번 씬에서 상단바 > Reporter > Create를 눌러 씬 하이어라키에 Reporter 오브젝트가 생성된 것을 확인합니다.
씬이 게임 실행 시 동작하게끔 할 땐 리포터 오브젝트의 Load on Start 옵션을 체크합니다.
게임 도중 로그를 확인할 때, 마우스를 반시계방향 원으로 드래그하면 됩니다.
단, 마우스 컨트롤은 모바일에서 적용되지 않고, 유니티 레지스트리 인풋 시스템 호환하여 사용할 때는 Load on Start 옵션이 체크되어있어야 합니다..

엑서 보드와의 호환 및 키맵



0번 씬에서 인풋 시스템은 다음과 같이 구성되어 있습니다.



플레이어의 인풋을 확인하려면 Connect Base의 디바이스 타입을 결정한 후, ControlOut의 디바이스 타입을 Connect Base의 것과 동일하게 맞추고 컴포넌트를 활성화시키면, 키보드 숫자패드와 wasd로 동작을 확인할 수 있게 됩니다.

Connect Base는 입력을 확인하기 위한 컴포넌트이므로, 실제 빌드시 해당 컴포넌트를 비활성화 해야 합니다.

디바이스 타입은 2023.03.08 기준 OldCable이 케이블 엑서보드, Bluetooth가 블루투스 엑서보드에 호환됩니다.
새로운 입력 체계가 등장한다면, NewCable과 같은 다른 스크립트를 추가로 작성하고 연결하면 됩니다.

엑서보드 연결신호 스크립트 수신
엑서보드의 키코드를 확인하려면 다음 조건이 선행되어야 합니다.

- BasicBroadcastHandler를 상속받은 컴포넌트
- ExerHeartKeyData 인스턴스를 변수로 선언

엑서보드에 대한 이전 단계의 설정들이 완료되었다면, BasicBroadcastHandler의 서브클래스는 ListenEvent에서 엑서보드에 대한 입력신호를 전달받을 수 있습니다.
전달되는 커맨드는 다음과 같습니다.

- commands[0] : 엑서보드 입력신호의 경우 *Commands.Input.ToString()*과 같음
- commands[1] : 플레이어 번호(1p, 2p 등)
- commands[2] : 입력 값

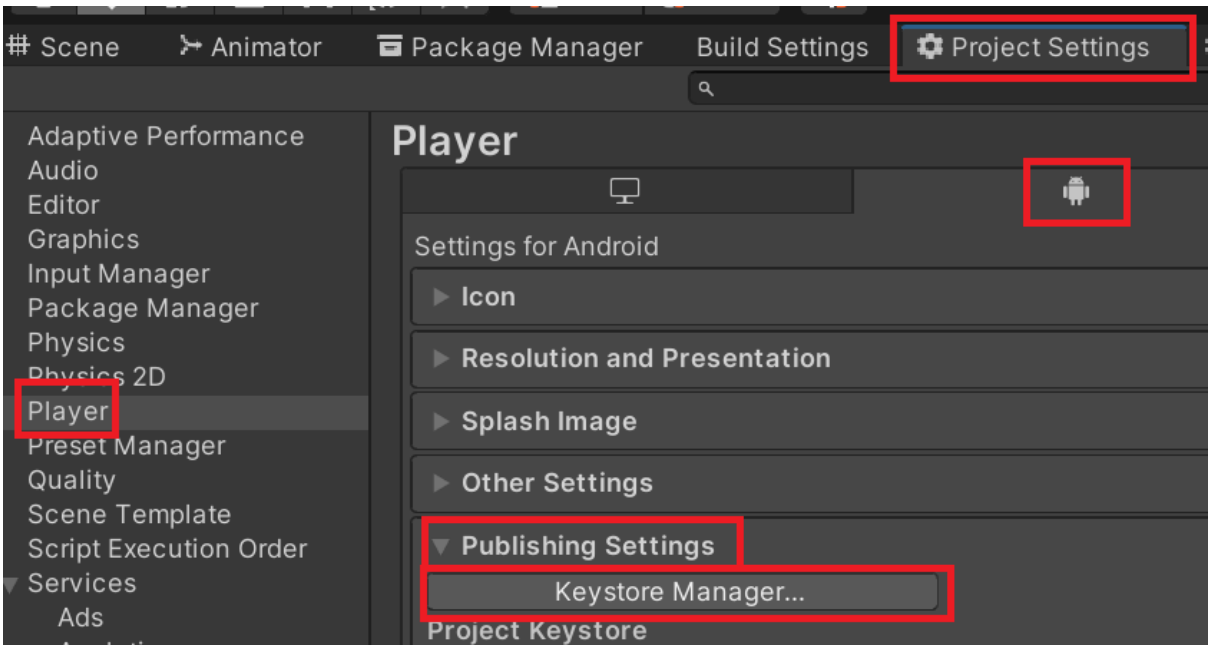
입력 신호에 대한 판별 이후, 입력값을 ExerHeartKeyData 인스턴스에 다음의 메서드로 해석을 시도하고, 해석된 값을 인스턴스에 저장합니다.
인스턴스명*.SetKeyData(commands[2]);*

입력값이 저장된 ExerHeartKeyData 인스턴스에서 자료는 다음과 같이 동작합니다.

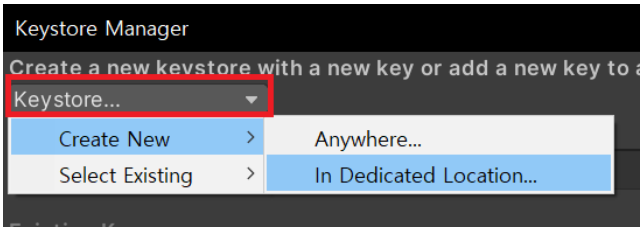
- 인스턴스명.keyPadNum : int, 입력버튼 좌상단부터 우하단까지 0 ~ 9의 값
- 인스턴스명.isPress : bool, true시 ButtonDown, false시 ButtonUp과 동일

키스토어 제작 및 구글 마켓용 앱 번들 빌드
구글의 안드로이드 업데이트 이후로, 구글 마켓에서의 빌드는 .apk 파일 대신 .aab 파일로 전환되었습니다.
따라서, 구글마켓에서는 앱 번들로서의 빌드가 필요하고, 몇몇 세팅이 선행되어야 합니다.

먼저, 구글 마켓의 앱을 개시할 때는 키스토어 인증을 거쳐야 합니다.
Project Settings > Player > 안드로이드 > Publishing Settings에서 키스토어 매니저로 이동합니다.

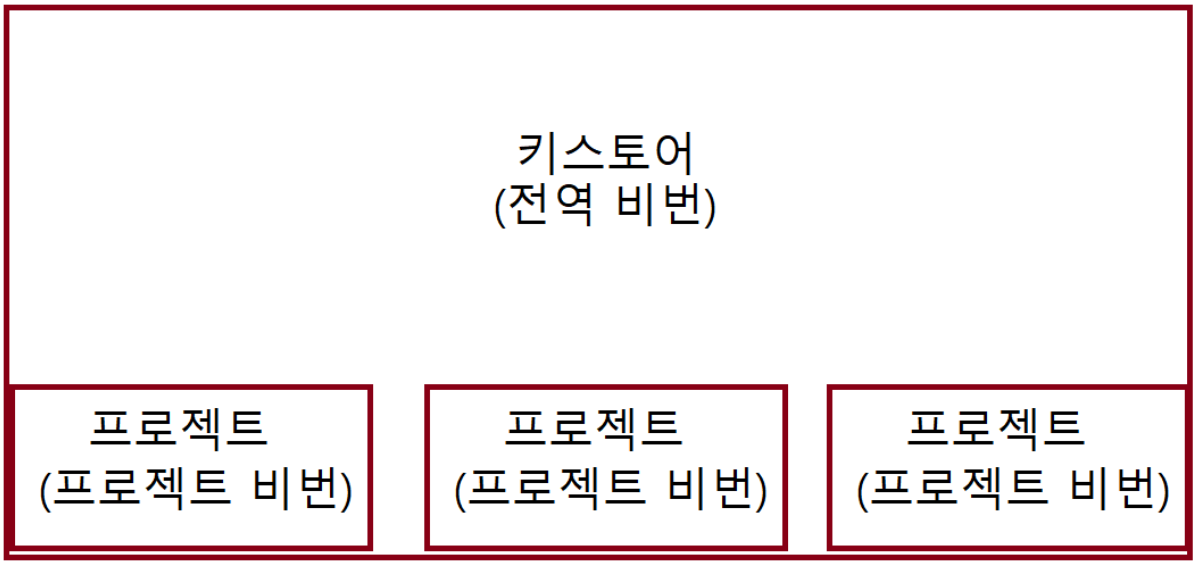


키스토어 매니저에서, Keystore > Create New > In Dedicated Location을 선택해서 새로운 키스토어 파일을 생성할 수 있습니다.

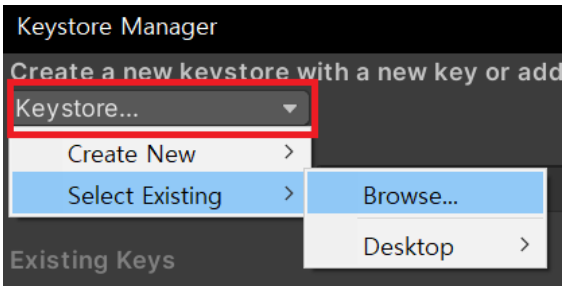


키스토어 작성시 필요한 내용은 다음과 같습니다.

키스토어의 경우, 하나의 키스토어 파일을 생성한 이후에는 완전히 다른 회사끼리의 프로젝트도 하나의 키스토어 파일에서 관리가 가능한 구조입니다.
새 프로젝트에 키스토어 인증이 필요할 경우에도 같은 키스토어 파일에 새로운 프로젝트를 등록시키는 것으로 키스토어 관련 인증을 해결할 수 있습니다.

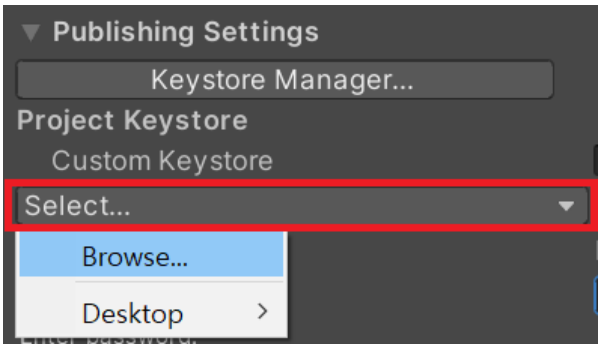


이미 존재하는 키스토어 파일에 새로운 프로젝트를 등록하기 위해 이미 존재하는 키스토어 파일을 불러오는 방법입니다.

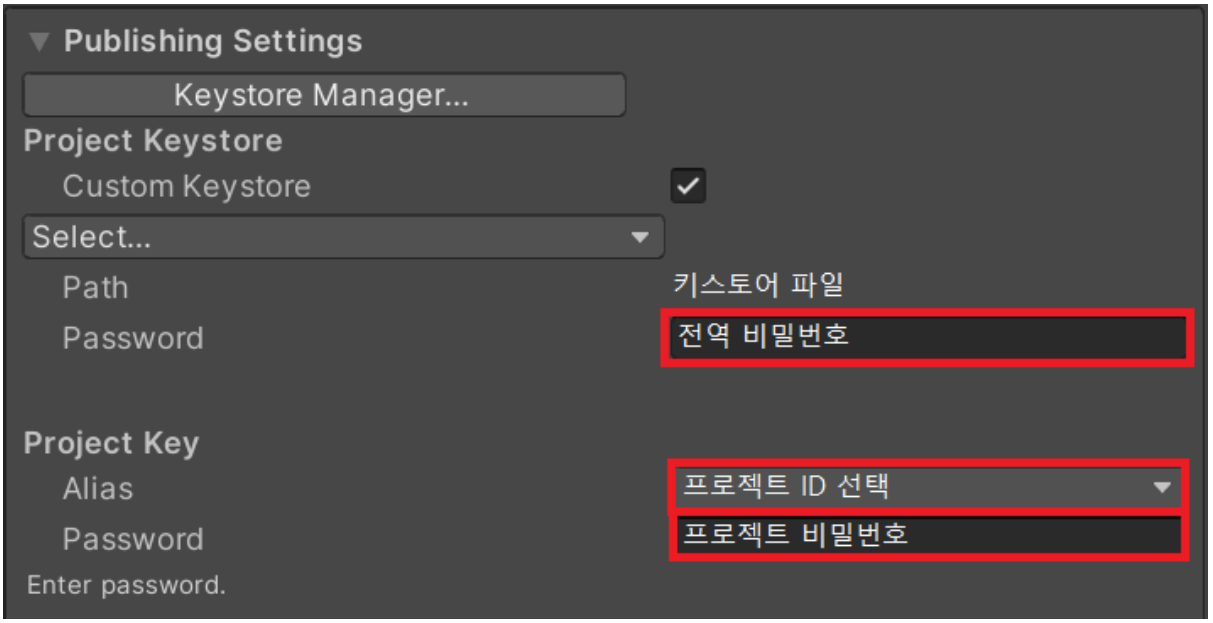


불러온 키스토어는 이미 정해진 전역 비밀번호를 확인한 후, 다른 프로젝트 인증을 진행할 수 있습니다.

키스토어에 프로젝트까지 연결을 하고 난 후에는 키스토어 비밀번호를 인증해야 합니다.
키스토어 비밀번호는 유니티 프로젝트를 열 때마다 다시 인증절차가 필요하게 됩니다.
먼저, Publishing Settings에서 키스토어 파일을 찾습니다.



이후 다음의 내용에서 인증이 완료된다면 구글 플레이스토어용 앱 번들 빌드가 가능합니다.

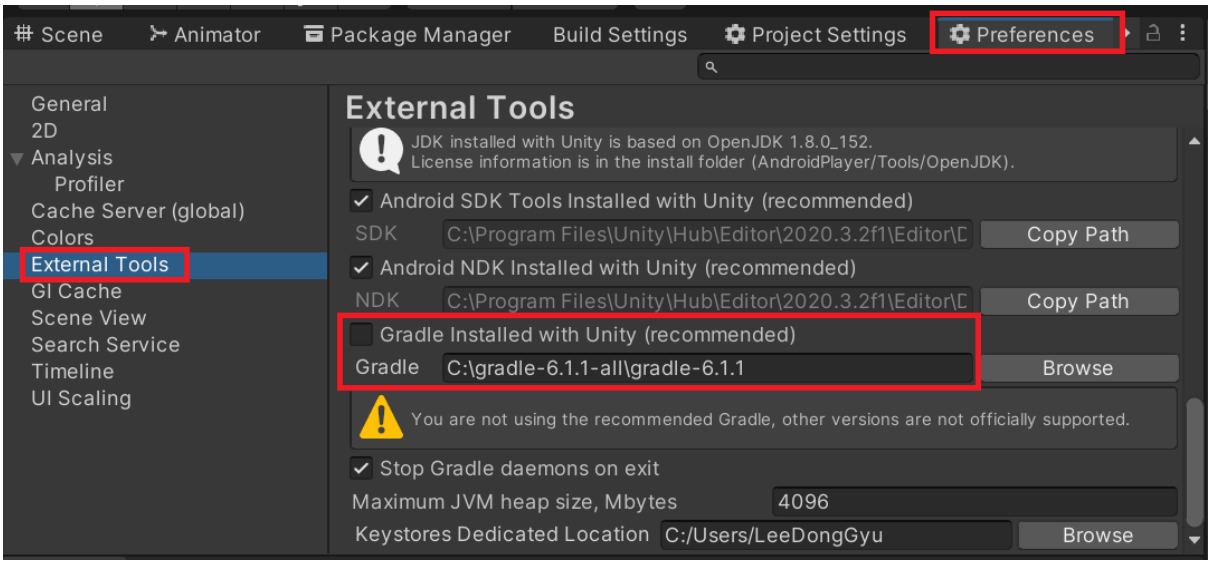


이제부터 후술될 내용은 구글 개발자 계정에 앱 번들을 업로드하고, 앱을 개시하는 과정에서 생길 수 있는 문제들에 대한 해결 방법입니다. 후술할 내용들은 예방적 조치로서도 유효하니, 빌드 이전에 확인해보는 것이 권장됩니다.

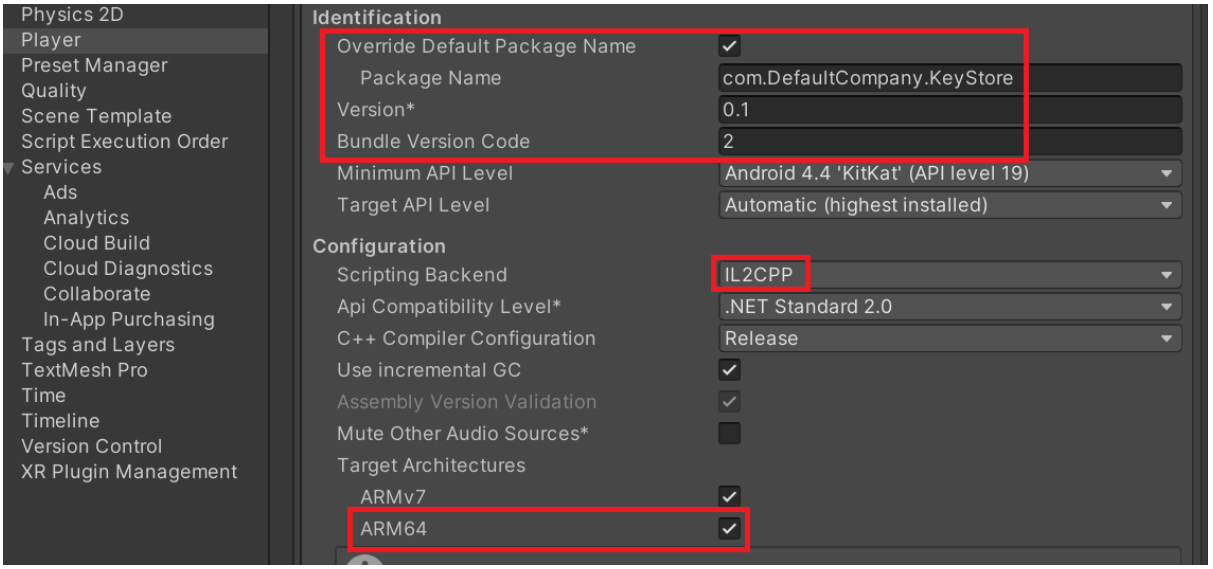
~~먼저, Preferences > External Tools > Gradle 버전을 맞춰줍니다.~~
~~각 유니티 프로젝트별로 호환되는 버전은 다음 두 링크를 참조합시다.~~
(2020.3.47f1 업그레이드 이후 엔진 기본 버전을 사용하므로 넘어갑시다**.)**

https://docs.unity3d.com/Manual/android-gradle-overview.html

https://developer.android.com/studio/releases/gradle-plugin?buildsystem=ndk-build&hl=ko#updating-gradle



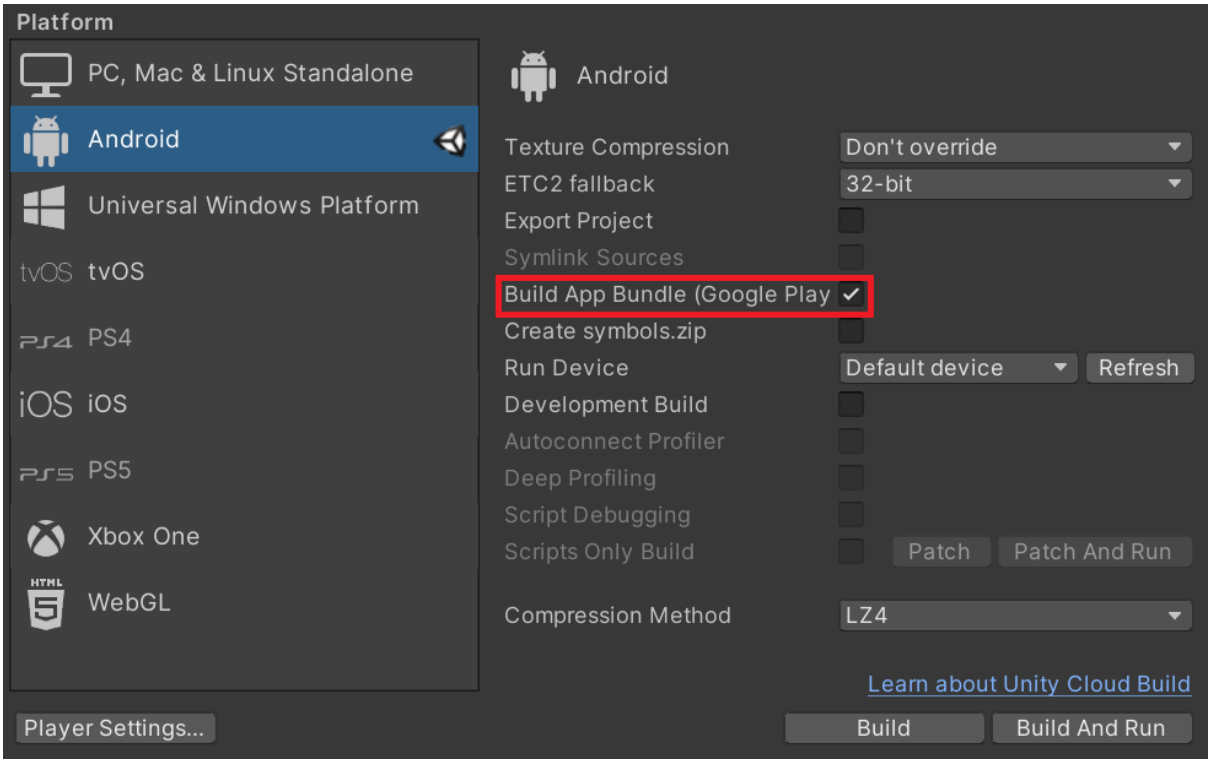
다음 구글플레이 빌드 세팅입니다.
Project Settings > Player > 안드로이드 > Publishing Settings로 이동합니다.



패키지명 오버라이딩을 체크해서 패키지명을 직접 지정하고, 같은 프로젝트 상에서도 서로 다른 패키지명은 다른 빌드 파일로 취급되는 점을 기억해주세요.

구글 플레이스토어는 앱 번들 테스트시 내부 테스트로도 패키지명을 출시한 것으로 판단하니 테스트용 패키지명을 별도로 작성하는 게 좋고, 번들 버전 코드는 새로 올릴때마다 증가하여야 합니다.

구글 플레이스토어에서 64비트 지원 이슈가 발생할 경우, Configuration 탭의 Scripting Backend를 IL2CPP로 바꾸고, ARM64를 체크하면 해결됩니다.
단, IL2CPP 환경에서의 보안 이슈는 현 독스에서는 고려되지 않습니다.

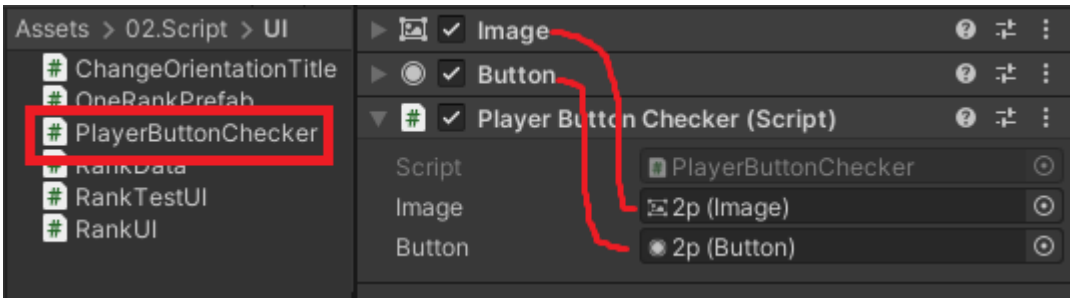


전술한 기능 체크 이후 빌드 세팅에서 앱 번들 빌드를 체크하고 빌드하면 앱 번들 빌드가 완료됩니다.

~~만약 Gradle 버전 이슈로 제대로 된 번들 빌드가 되지 않는다면, 프로젝트 폴더에서 .aab 파일을 검색하여 임시 앱 번들을 사용해볼 수 있습니다.(권장되지 않음)~~

메인화면 플레이어 버튼 기능
23년 6월 부터 버튼 기능 동작이 통일됨에 따라, 베이스 프로젝트에도 버튼 기능이 추가되었습니다.

23.06.19 현재 버튼 기능의 사용 가능 조건은 다음과 같습니다.
- 게임을 플레이하는 최대 인원수가 2인 이하
- 1인 플레이와 2인 플레이 가능 여부에 따라 버튼의 존재 유무가 바뀜
- 케이블 연결과 블루투스 연결이 되는 보드의 존재
- 버튼 ui에는 버튼 컴포넌트와 이미지 컴포넌트가 존재할 것



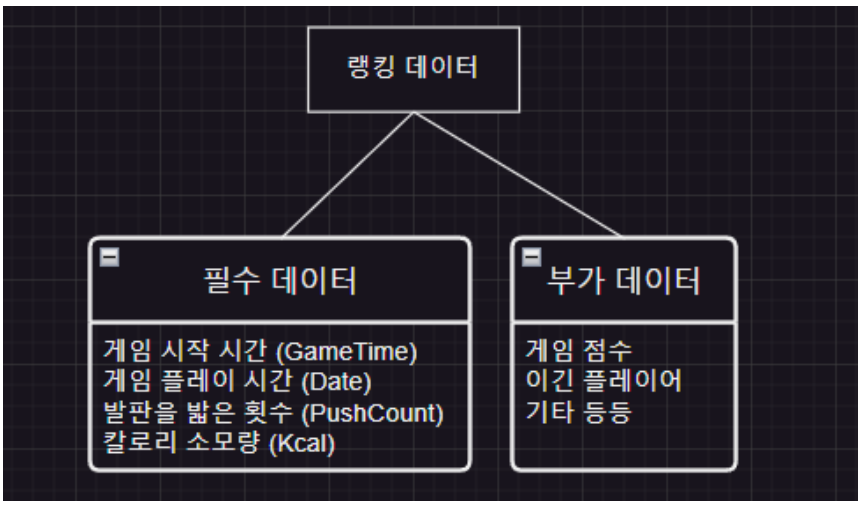
Assets > Script > UI 폴더에서 PlayerButtonChecker 컴포넌트를 사용하고, 해당 컴포넌트의 Button, Image 변수에 버튼의 각 컴포넌트를 연결해주면 됩니다.

랭킹화면 기능
랭킹화면의 기능을 설명하기에 앞서, 랭킹시스템의 기본 개념부터 설명하도록 하겠습니다.

타사와의 연계 혹은 보험사 등 서버 및 자료관리 측면에서 자사의 게임이 보존해야 하는 필수적인 데이터가 있고(필수 데이터), 각각의 게임에서만 적용되는 부가적인 데이터가 존재합니다(부가 데이터).

필수 데이터로는 게임 시작 시간, 게임 플레이 시간, 발판을 밟은 횟수, 칼로리 소모량이 존재합니다.

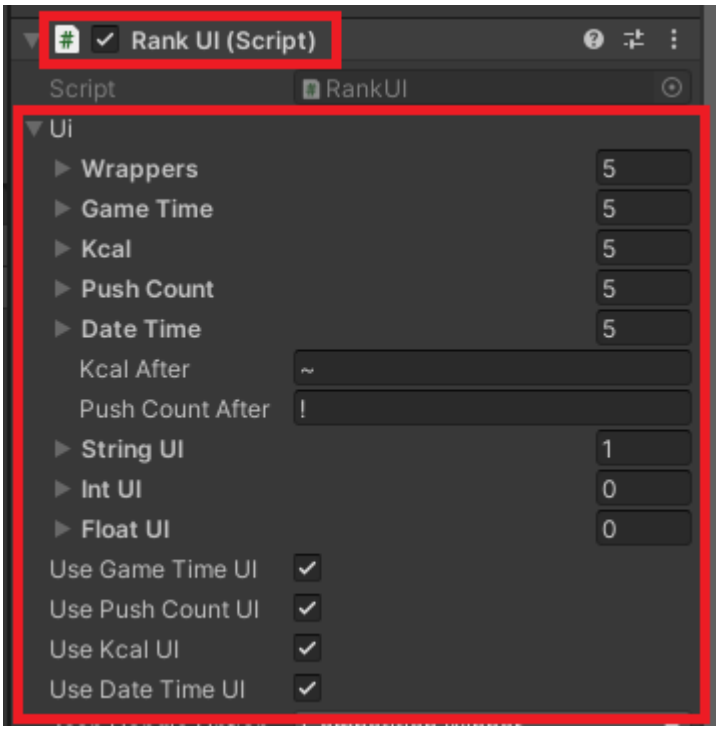
부가 데이터로는 게임 점수, 어느 플레이어가 승리했는지 등 다양하게 존재할 수 있습니다.



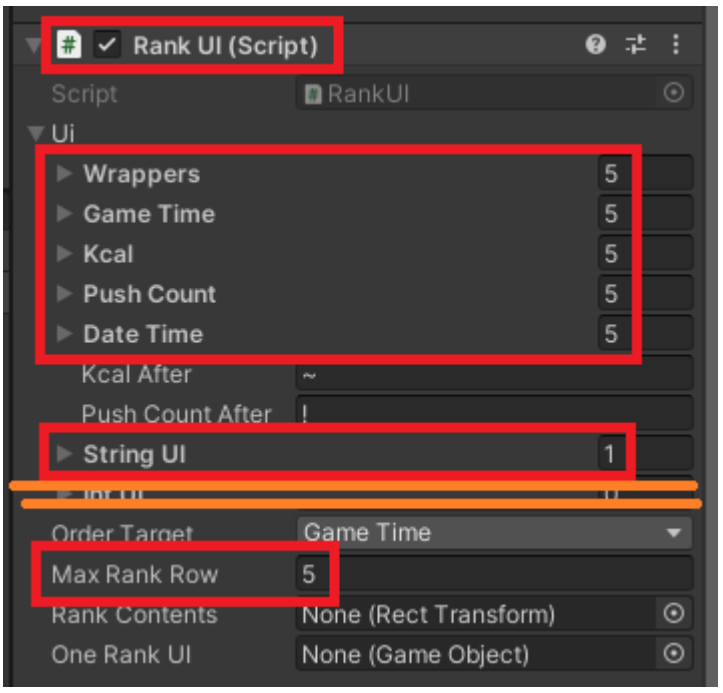
랭킹화면 구성은 23.06.19 현재 2가지의 사용 가능 방법이 존재합니다.
1. 랭킹 화면을 전체 구성하고, 각 행을 UI 컨트롤러에 수동으로 연결
2. 랭킹 화면의 한 행에 해당하는 내용을 하나의 프리팹으로 묶고, 이를 통해 자동으로 랭킹 화면 구성

1번과 2번 모두 랭킹화면의 틀이 존재한다고 가정하고 작동하기에, 랭킹화면 전체 레이아웃의 디자인은 별도로 필요한점을 참고하면 됩니다.

1번의 방법으로 랭킹화면을 구성하도록 하겠습니다.

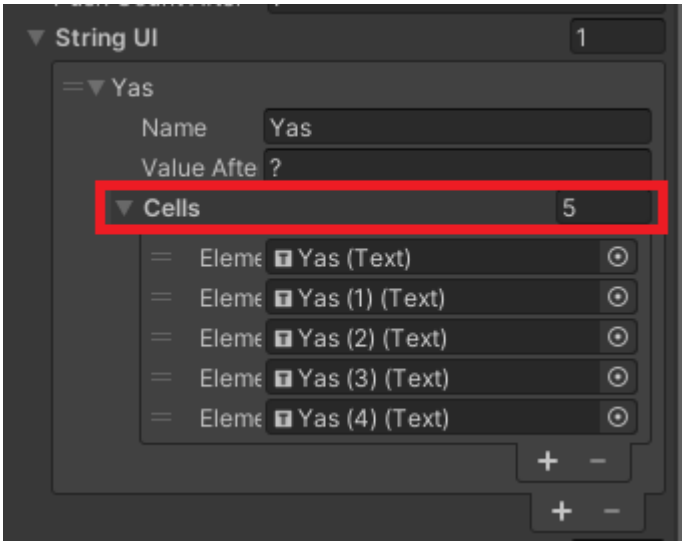


Assets > Script > UI의 RankUI 스크립트를 사용하면 다음과 같은 항목이 존재합니다.
GameTime, Kcal, PushCount, DateTime은 모두 이전에 설명한 필수 데이터입니다.
StringUI, IntUI, FloatUI는 부가 데이터를 포함시키기 위한 string, int, float 형식의 항목입니다.
Wrappers는 필수 데이터 셀과 부가 데이터 셀을 묶는 하나의 RectTransform입니다.



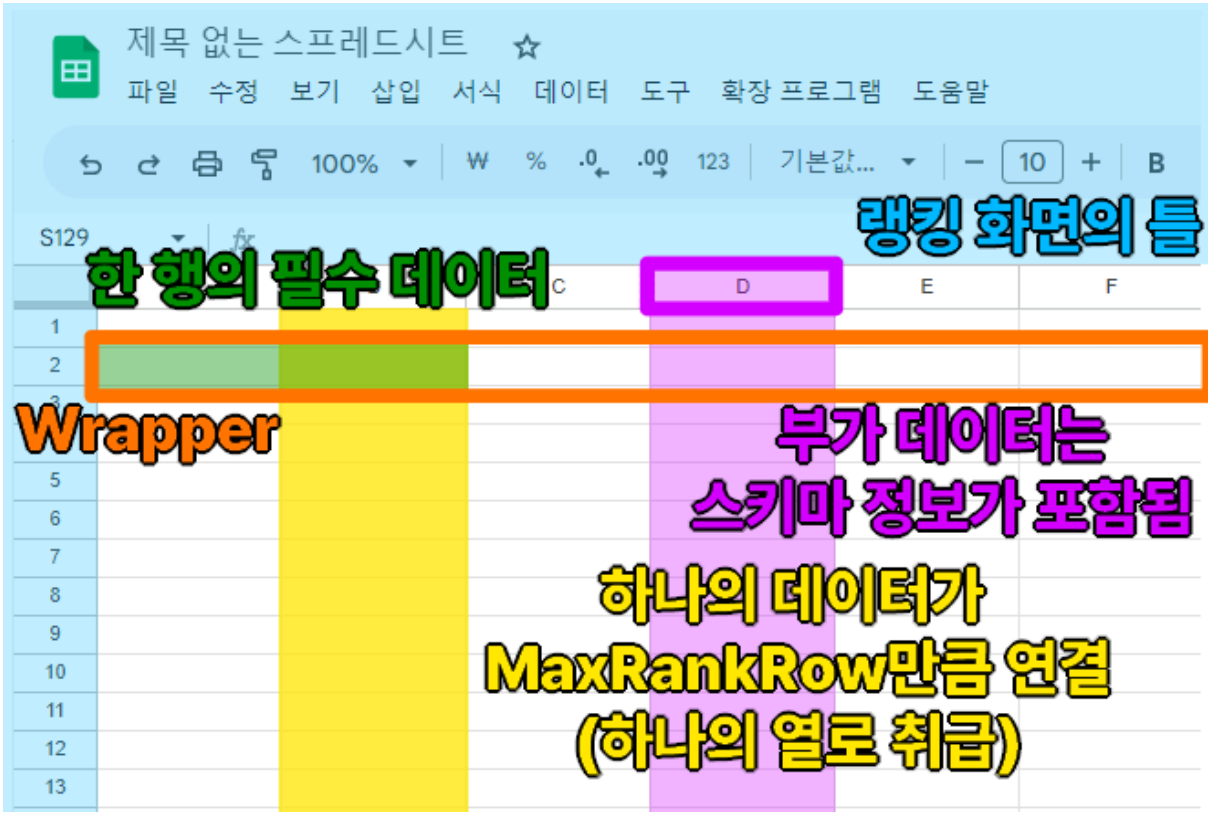
컴포넌트의 MaxRankRow에 대한 숫자는 랭킹에 표시되는 최대 행의 개수를 지정하며, 필수 데이터와 래퍼는 한 열에 해당하는 셀의 수가 맞게 들어간 것을 확인할 수 있습니다.

부가 데이터의 경우 MaxRankRow의 숫자와는 맞지 않은 모습을 보이는데, 이는 부가 데이터 항목이 이중 리스트로, 즉 각각 열의 리스트로 관리되기 때문입니다.

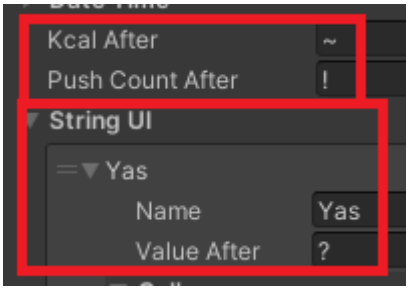


부가 데이터 항목의 리스트를 펼쳐 보면 MaxRankRow의 숫자에 맞게 ui가 등록되어 있는것을 확인할 수 있습니다.

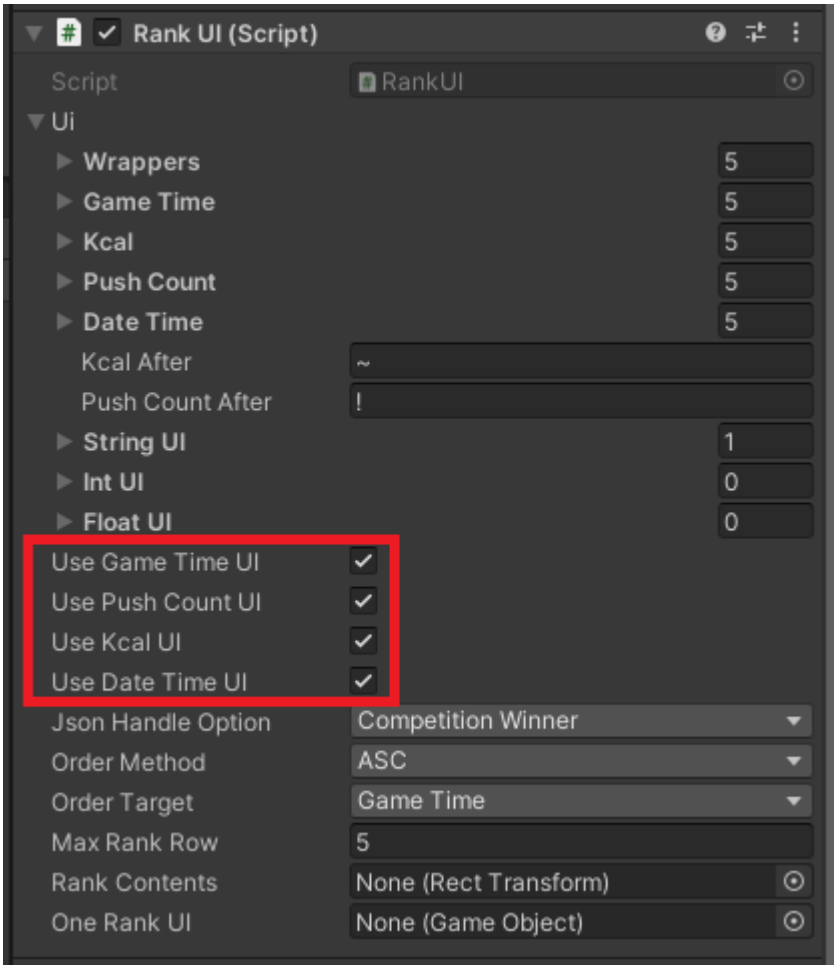
랭크 화면을 엑셀 테이블과 비교하면 다음과 같습니다.



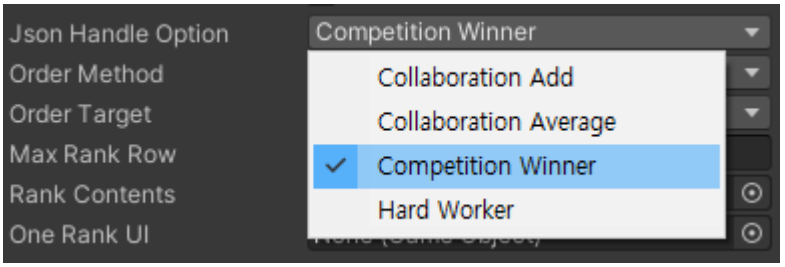
해당 사진에서 녹색과 같이 필수 데이터가 한 묶음으로 처리되지만, UI의 연결 상에서는 필수 데이터도 각각의 데이터마다 노란 색처럼 하나의 열로서 연결됩니다.



일부 필수 데이터와 부가 데이터는 스키마 정보를 가지고 있습니다.
스키마 정보에는 해당 정보를 저장하기 위한 키워드로 Name 값을 가지고 있고, 각 셀의 정보를 표시한 뒤에 붙는 ValueAfter 값이 존재합니다.
ValueAfter가 이미지처럼 되어있다면, Kcal 값의 뒤에는 항상 '~'기호가 붙습니다.

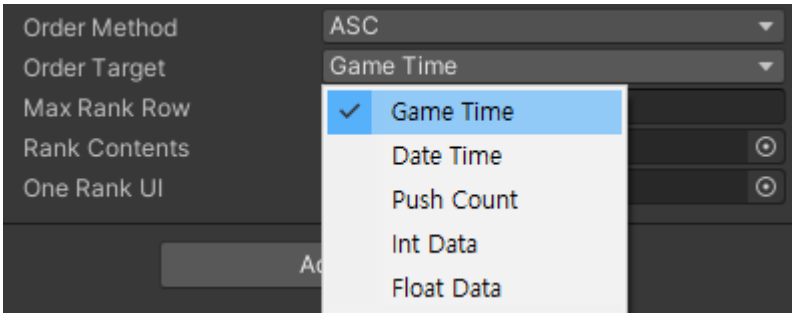


만약 필수 데이터를 사용하지 않고 싶다면, 해당 위치의 Use 옵션을 체크 해제하고 사용하지 않을 필수데이터 셀을 각각 리스트에서 제거하면 됩니다.
Use 옵션이 해제된 상태라면, 랭킹창 시스템은 해당 셀이 존재하지 않는것으로 간주하므로, 별도의 조치를 취하지 않아도 기능에는 문제가 없습니다.



JsonHandleOption은 타사와의 통신을 위해 JSON 파일을 추출하기 위한 옵션입니다.
JSON파일의 경우 랭킹화면의 생명주기에서 자동으로 생성되며, 해당 옵션은 JSON 파일의 산출 방식을 결정합니다.

옵션 설명은 다음과 같습니다.
- CollaborationAdd : 모든 플레이어의 결과를 더합니다.
- CollaborationAverage : 모든 플레이어의 결과의 평균을 구합니다.
- CompetitionWinner : 대결 형식의 게임일 때, 이긴 플레이어의 결과를 사용합니다.
- HardWorker : 시간을 고려하지 않고 가장 활동량이 많은 플레이어의 결과를 사용합니다.



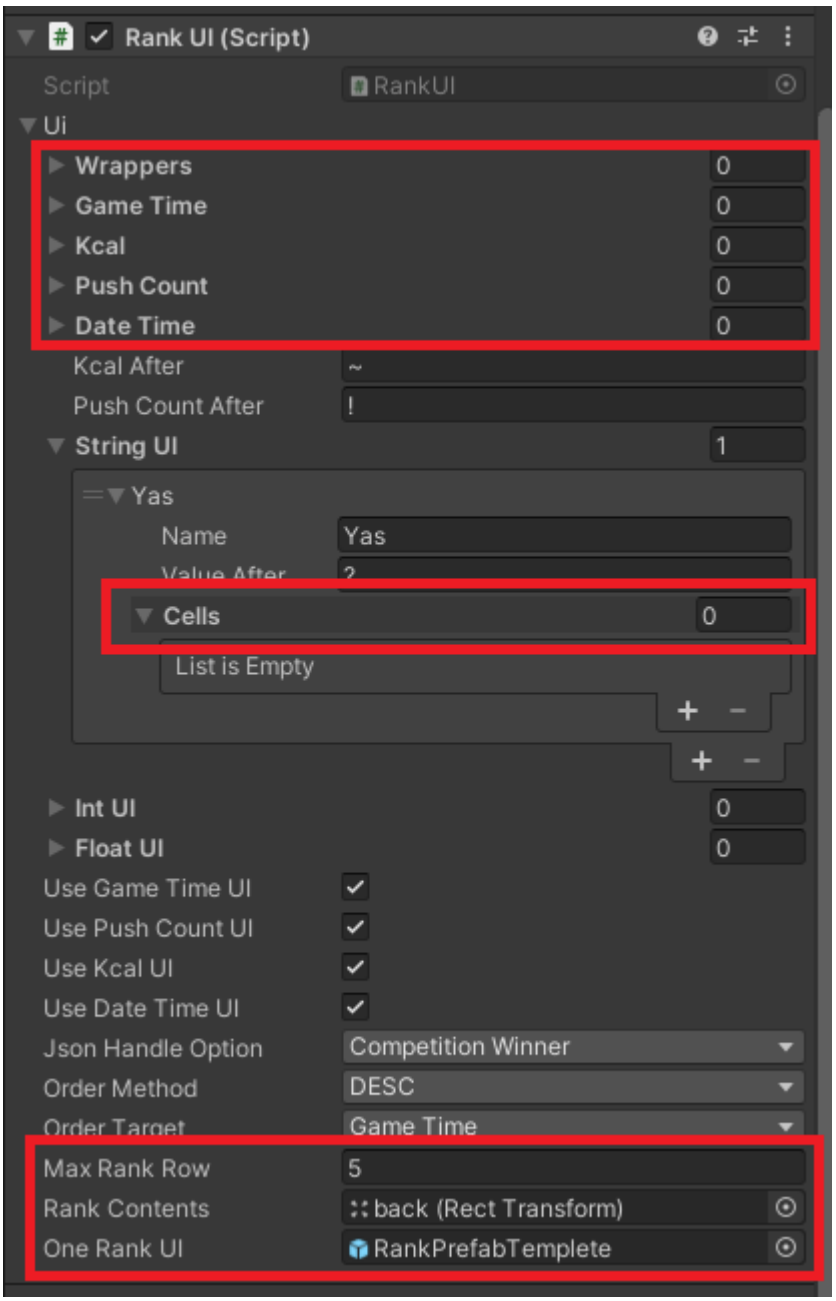
랭킹화면이 동작할 때, 각 랭킹 데이터를 어떻게 정렬할 것인지 결정하기 위해 OrderTarget, OrderMethod 옵션을 수정할 수 있습니다.

OrderTarget 옵션은 필수 데이터와 부가 데이터 중 하나를 선택할 수 있습니다.
부가 데이터를 정렬 기준으로 삼을 때, string 형식의 데이터는 사용할 수 없으며, int 및 float 형식의 데이터는 첫 번째로 등록된 자료를 기준으로 정렬합니다.

필수 데이터와 부가 데이터는 실제 UI에 배치할 때, 순서가 정해져 있지는 않습니다.
정렬 대상의 열을 우측 끝에 배치하더라도 연결시 첫 순서이기만 하면 됩니다.

정렬 대상을 정하고 난 뒤에는 OrderMethod를 통해서 오름차순 및 내림차순 정렬을 선택할 수 있습니다.

랭킹화면 기능을 사용하는 다른 방법
랭킹화면의 각 UI 컴포넌트 연결이 번거로울 경우, 프리팹을 활용하여 조금 더 단순한 방법으로 랭킹화면을 사용할 수 있습니다.

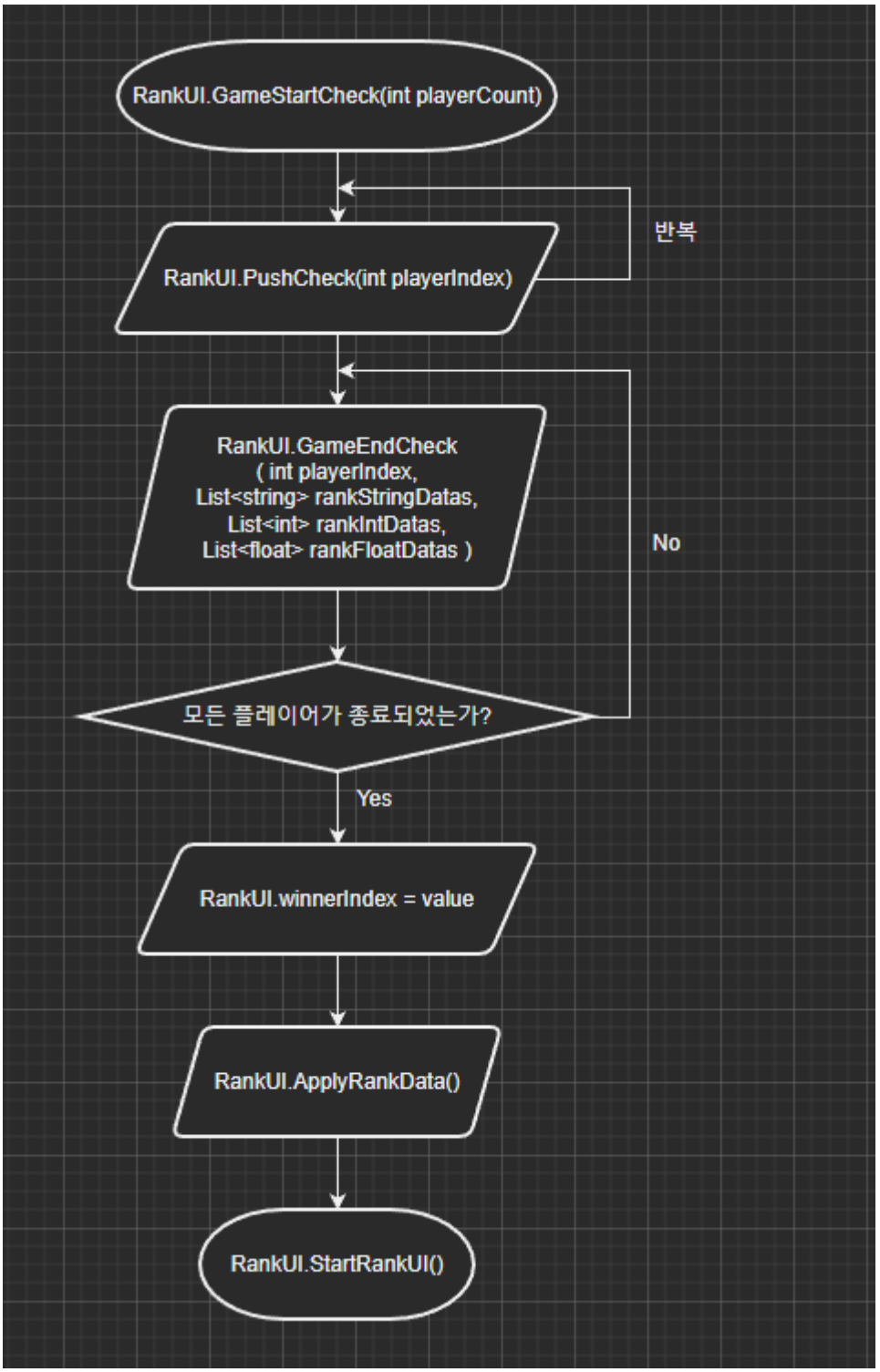


이 화면은 프리팹을 이용한 랭킹화면 세팅 화면입니다.
기본적으로 이전의 연결 세팅과 차이점은 없지만, 부가 데이터의 스키마를 제외하고 다른 UI들의 리스트에 MaxRankRow 숫자만큼의 UI들이 연결되어있지 않습니다.

다음과 같이 설정이 끝났을 때, 프리팹을 활용한 기능이 동작하려면 각각의 행이 위치할 RankContents를 RectTransform으로 지정해야 하며, 한 행에 대한 프리팹을 OneRankUI에 연결해야 합니다.

랭킹화면의 생명주기와 코드 호출
JSON 파일 추출과 랭킹화면 표시를 위해서 UI에 일정한 생명주기가 필요합니다.

생명주기는 다음과 같습니다.



랭킹화면을 시작시키기 전에, 각종 정보를 측정하기 위해 다음 메서드를 실행시킵니다.
RankUI.GameStartCheck(int playerCount)
playerCount의 경우 인원수를 나타냅니다.

이후 발판을 누를 때마다 RankUI.PushCheck(int playerIndex) 를 연결시킵니다.
해당 메서드는 한번 호출시마다 발판을 1회 누른것으로 간주하는 코드입니다.

playerIndex는 어떤 플레이어가 해당 메서드를 호출하였는지 결정하며, 해당 int 값은 Dictionary 형식으로 관리됩니다.
playerIndex의 값은, 이후의 다른 메서드에도 사용되므로, 통일성있게 호출하면 됩니다.

게임이 종료되는 시점에는 RankUI.GameEndCheck 메서드를 사용합니다.
협동 게임뿐만 아니라 경쟁 게임도 지원하기 위해, 해당 메서드는 각 플레이어별로 호출하여 최종적으로 모든 플레이어의 게임이 종료되어야 다음 단계로 넘어갈 수 있습니다.
RankUI.GameEndCheck 메서드의 매개변수는 다음과 같습니다.
int playerIndex >> 플레이어 값
List<string> rankStringDatas >> 랭킹의 string 형식 부가 데이터(게임 1회분)
List<int> rankIntDatas >> 랭킹의 int 형식 부가 데이터(게임 1회분)
List<float> rankFloatDatas >> 랭킹의 float 형식 부가 데이터(게임 1회분)

게임이 모두 종료된 이후 혹은 특정 플레이어의 게임 승리 이후 RankUI.winnerIndex 변수에 playerIndex에 해당하는 값을 대입할 수 있습니다. 협동 게임 등 게임 모드에 따라 해당 값은 사용되지 않을 수 있으며, 경쟁 게임에서 해당 값의 대입에 오류가 생길 경우 JSON 파일의 추출에 문제가 생길 수 있습니다.

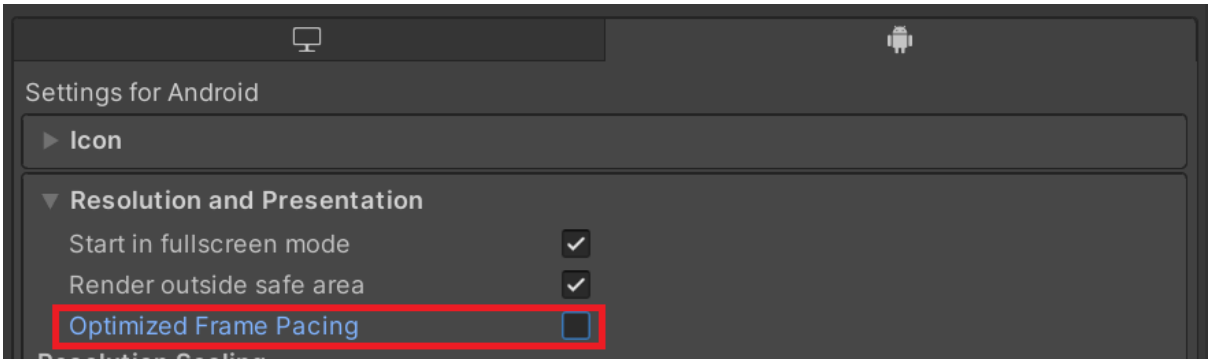
이전의 모든 절차가 완료된 뒤, RankUI.ApplyRankData() 메서드를 호출합니다.
해당 메서드는 JSON 파일 추출 및 현재 진행한 게임 데이터를 랭킹에 저장합니다.
게임 데이터의 저장은 23.06.19 시점 PlayerPrefs 클래스를 활용하여 저장됩니다.

마지막으로, RankUI.StartRankUI() 메서드를 호출하여 저장된 랭킹 데이터를 표시합니다.
이전의 RankUI.ApplyRankData() 데이터가 호출되지 않았을경우, RankUI.StartRankUI() 메서드 호출시 현재 진행한 게임의 랭킹 데이터가 표시되지 않을 수 있습니다.

만약 1인 플레이의 랭킹과 2인 플레이어 랭킹을 분리시키는 등의 작업이 필요하다면, 랭킹의 생명과정을 실행시키기에 앞서 RankUI.rankMode = int value로 랭킹 모드를 변동시켜볼 수 있습니다.
해당 값에 따라, 랭킹의 데이터가 분리되어 저장됩니다.

문제 해결

프리징 문제 해결
안드로이드 환경에서 특정 유니티 버전의 빌드는 랜덤한 시간 이후에 프리징(멈춤) 현상이 존재합니다(주로 빌드 2020.3~에서 발생).
이 문제는 Project Settings > Player > 안드로이드 > Resolution and Presentation에서 다음의 옵션을 체크 해제하는것으로 해결할 수 있습니다.



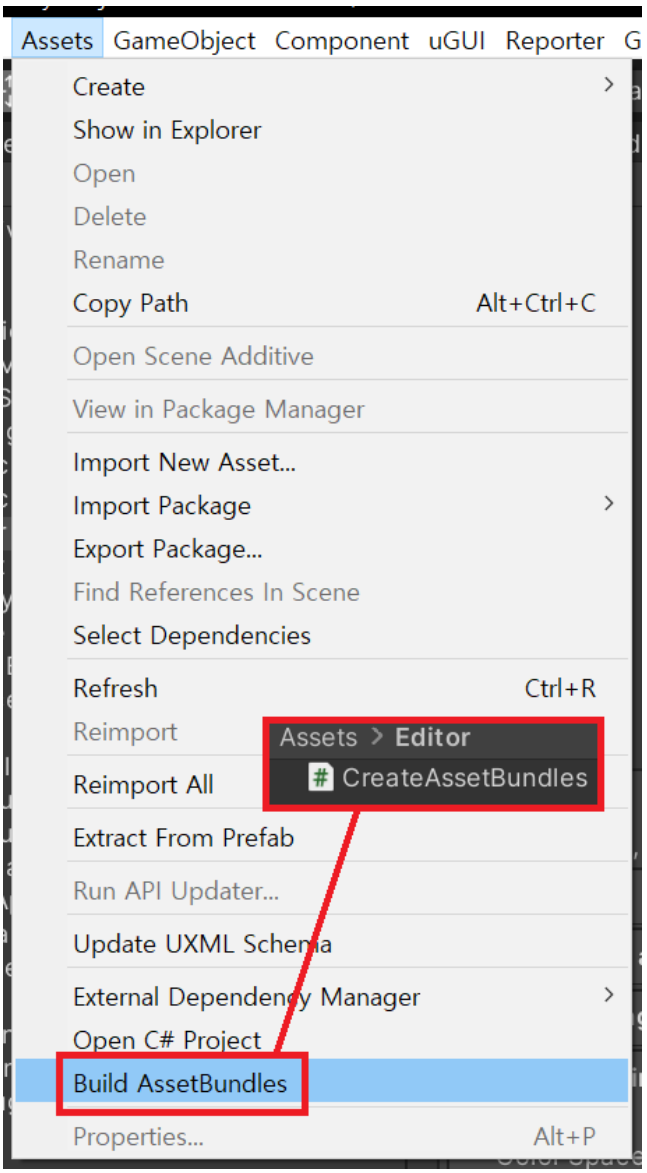
앱 용량 **150**메가 해결
구글 플레이스토어에서는 일반적으로 150mb 이상의 앱을 지원하지 않습니다.
앱 용량 제한에 대한 대응법으로는 두 가지 방법이 존재합니다.

첫째로는 게임 제작 시기부터 에셋 번들링을 사용하는 것입니다.
다만, -에셋 번들링을 어떻게 받을 수 있게 할 것인가, -에셋 번들을 로드 및 언로드하는 관리 코드가 필요한 점 등의 문제가 존재합니다.

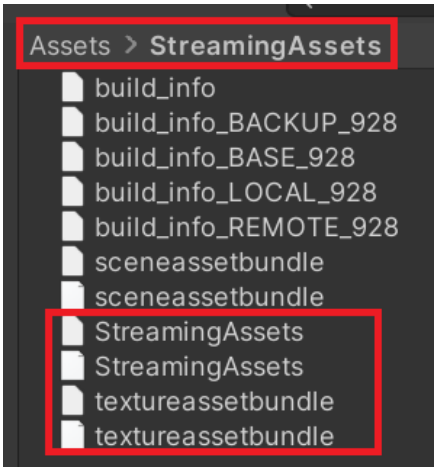
두번째로는 구글 플레이스토어의 Play Asset Delivery를 사용하는 겁니다.
해당 방법은 기존 apk 빌드본의 크기가 1Gb를 초과하지 않는 경우 사용 가능한 방법입니다.
해당 기능은 구글측에서의 하단 링크를 통해 더 알아볼 수 있습니다.
엑서 베이스에 설치된 pad 패키지는 1.8.2 버전이며, Assets 폴더에 내장되어 있습니다.

https://developer.android.com/guide/playcore/asset-delivery?hl=ko

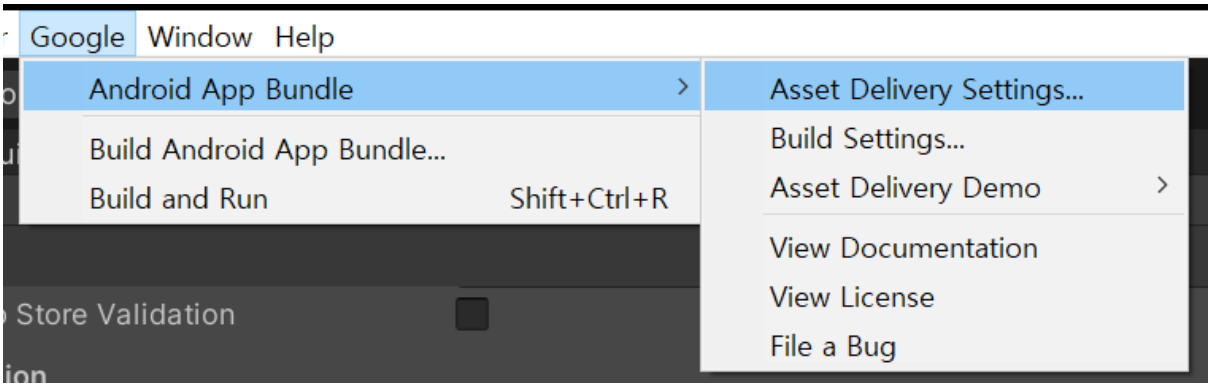
23.05.15 추가, 안드로이드 12 버전 충돌 이슈로 인하여 패키지 버전을 1.7.0 버전으로 다운그레이드 하였으며, 하단 사이트에서 설치할 수 있습니다.
https://developers.google.com/unity/archive?hl=ko#external_dependency_manager_for_unity



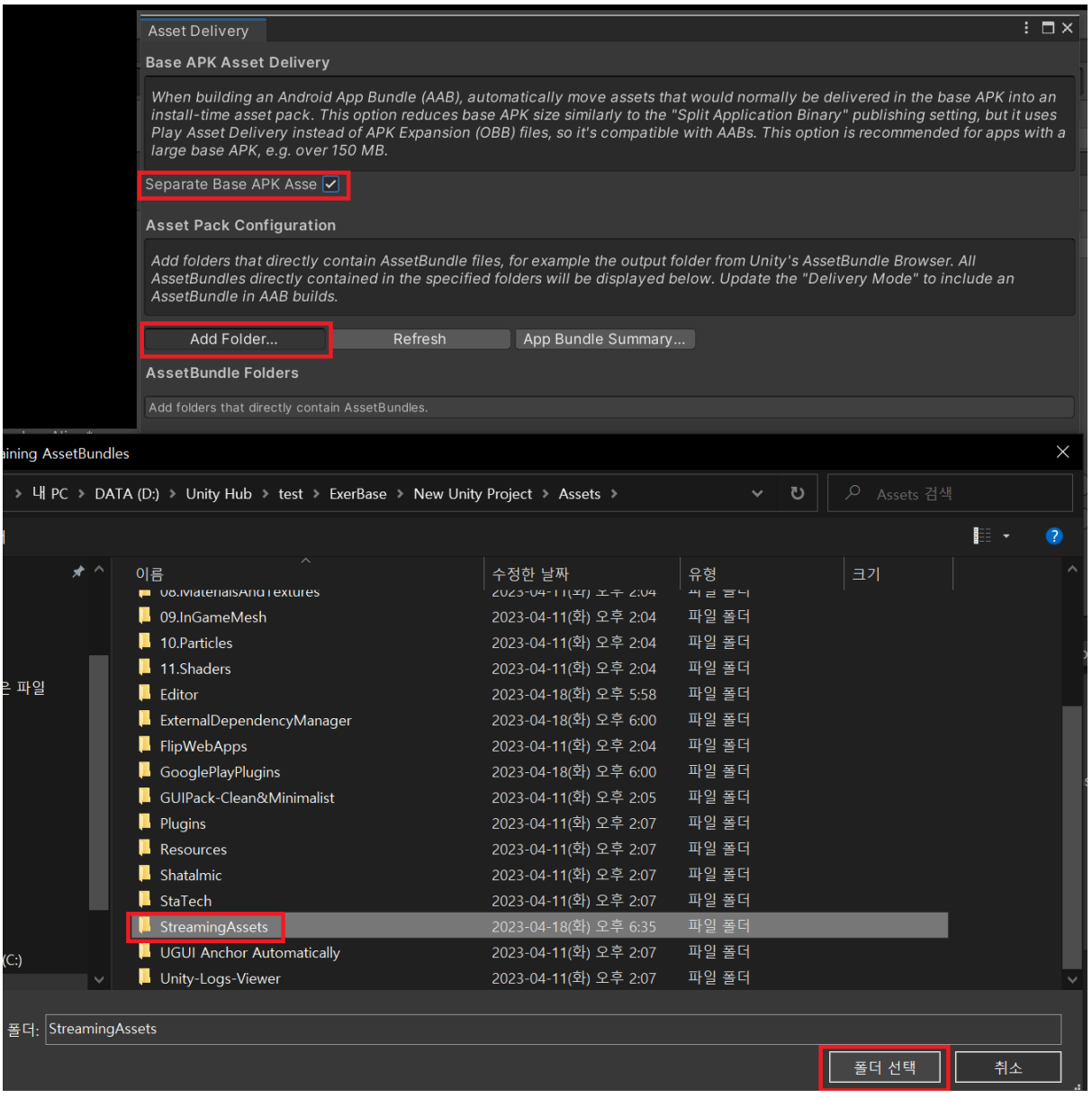
CreateAssetBundles 스크립트 기능으로 Assets > Build AssetBundles를 실행시킵니다.



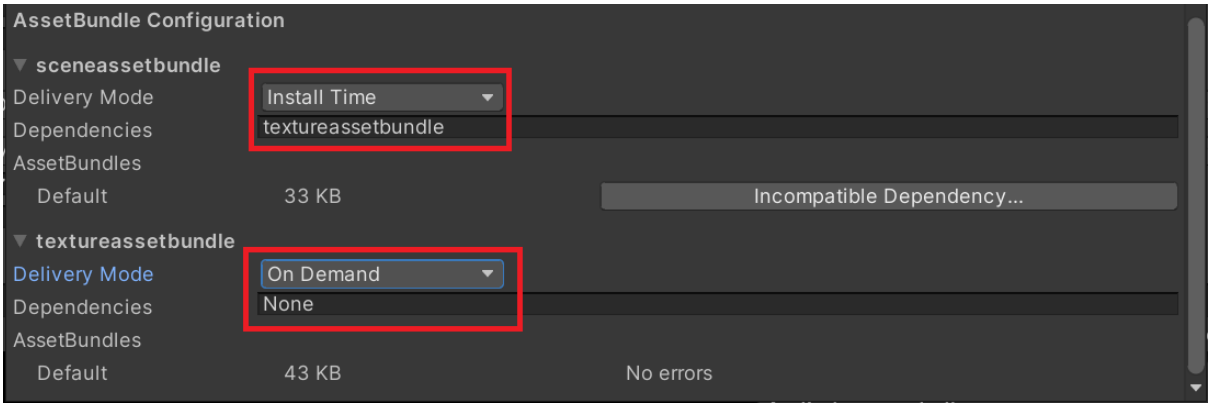
Assets > StreamingAssets에서 해당 파일들을 확인합니다.



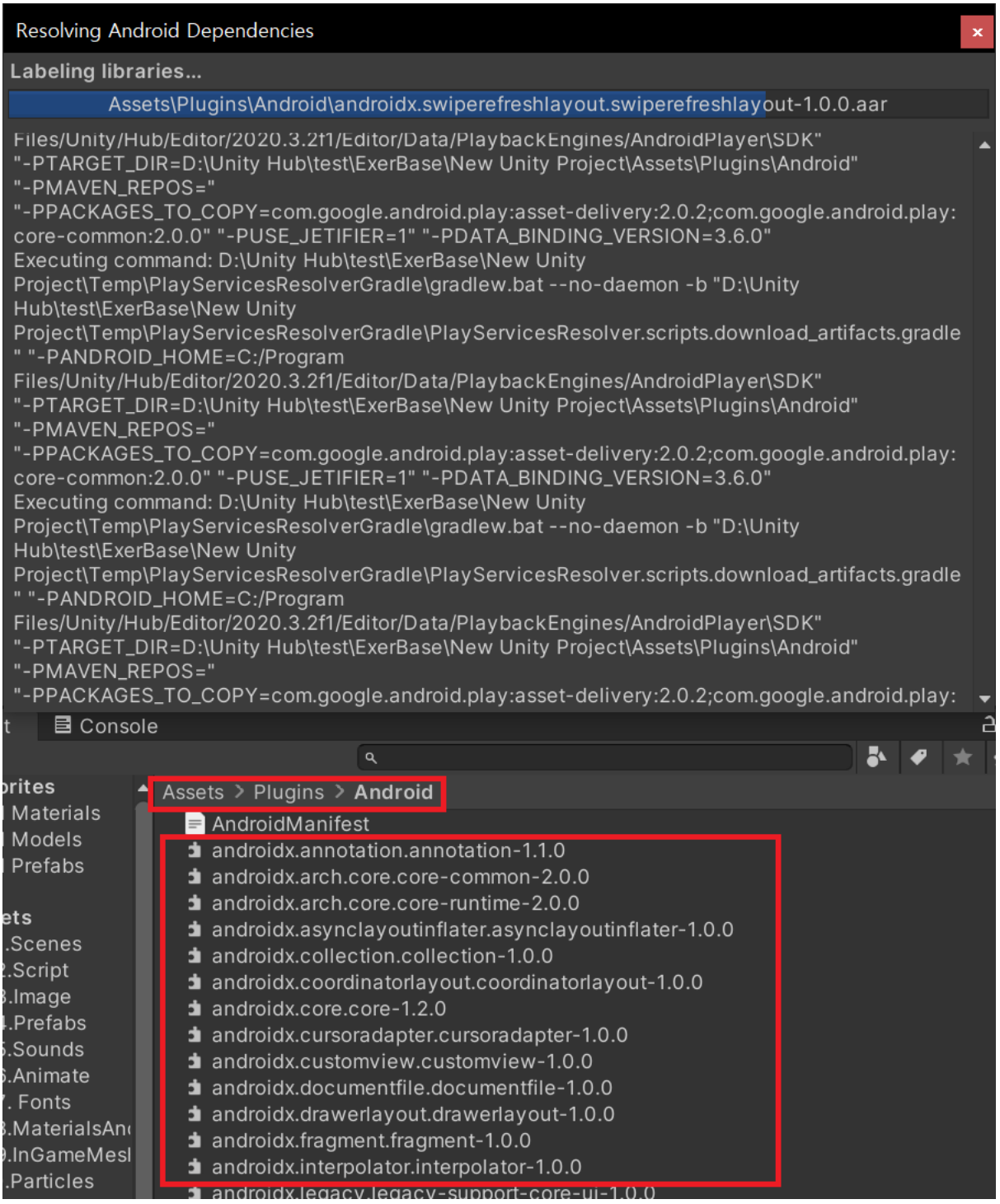
Google > Android App Bundle > Asset Delivery Settings를 선택합니다.



Separate Base APK Asset에 체크한 후, Add Folder > StreamingAssets 폴더를 선택합니다.



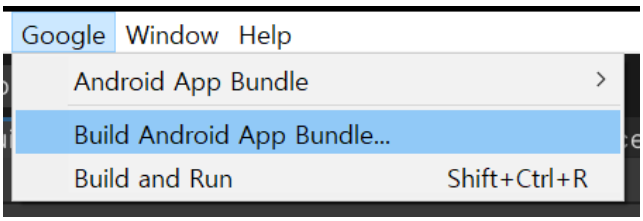
sceneassetbundle의 옵션을 Install Time, textureassetbundle의 옵션을 OnDemand로 설정합니다.



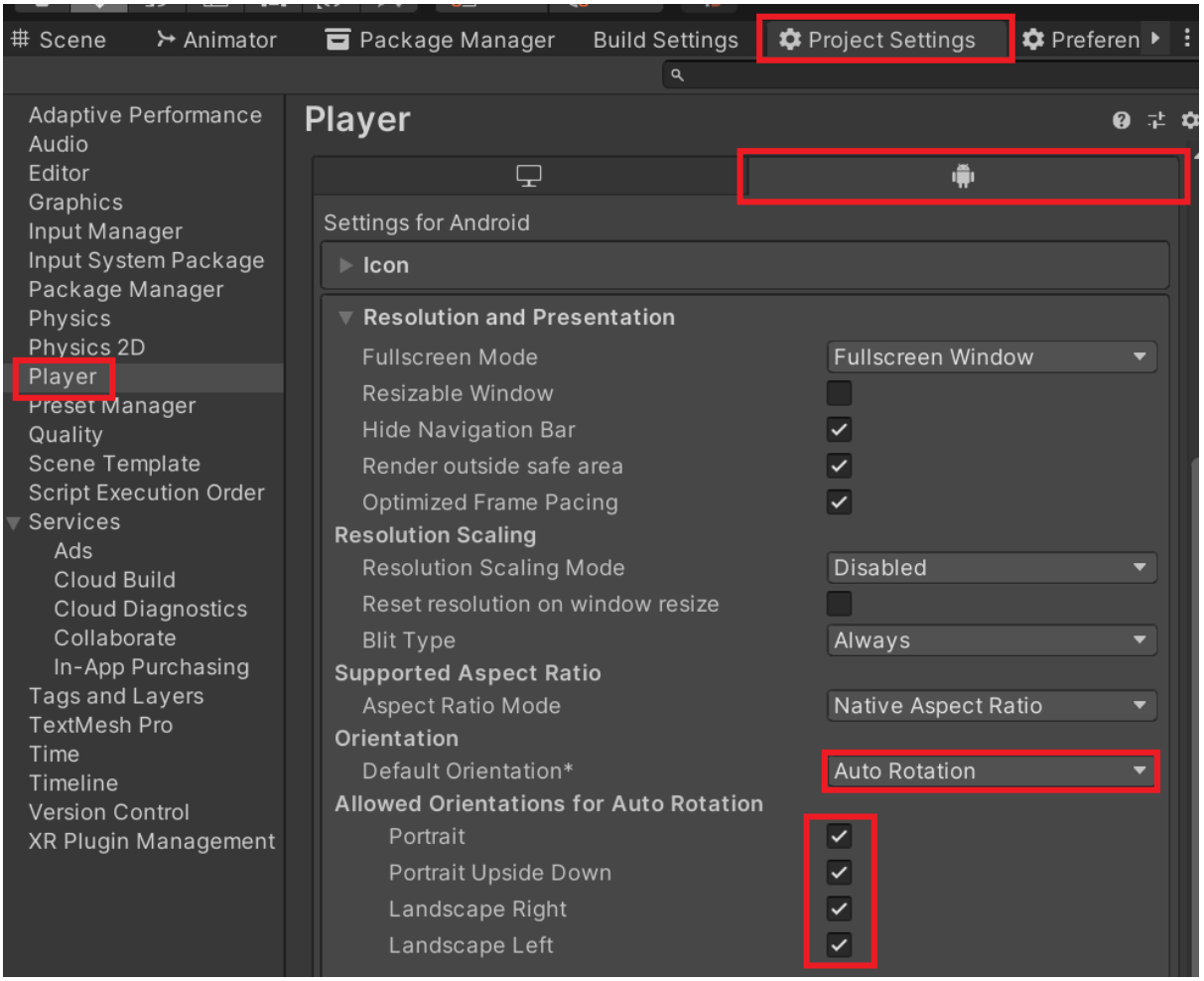
Auto Resolution 안내확인창에서 Enable 선택 이후, Assets > Plugins > Android 폴더에서 jar 및 aar파일이 생성되는것을 확인합니다(버전마다 생성되는 라이브러리에는 차이가 있습니다).

만약 해당 화면에서 JAVA_HOME 관련 오류가 발생할 경우, 자바 설치, 환경변수 > 시스템변수 등록, 재부팅의 순서로 진행하면 해결됩니다.

이렇게 적용이 완료된 pad 세팅은 Build Setting에서 앱 번들을 빌드하던 기존 절차와는 달리, 상단바 Google > Build Android App Bundle을 통해 빌드가 가능합니다.



안드로이드 환경에서의 화면 회전 옵션



안드로이드의 경우, 다음 위치의 세팅에서 회전 값을 변경할 수 있지만, 모든 프로젝트에서 동일한 옵션을 사용하지는 않을 수 있고, 참고만 하시면 됩니다.