UE4 안드로이드 프로그래밍

신광섭
Developer Relations Manager/Programmer
Epic Games Korea

목차

- UE4 빌드 시스템
- 네이티브 코드 연동
- 안드로이드 빌드 및 테스트
- 안드로이드 디버깅

UNREAL ENGINE
UNREAL SUMMIT 2015

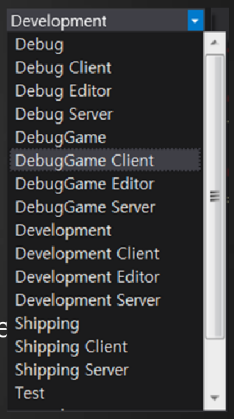
UE4 빌드 시스템

UE4 빌드 시스템

- 언리얼 엔진 4는 모든 플랫폼을 VS를 기반으로 개발
- 그 핵심은 UE4 빌드 프로그램인 UnrealBuildTool
- UnrealBuildTool이 Solution Platforms에서 선택한 플랫폼에 맞는 설치된 SDK와 기타 플랫폼 관련 설정 내용을 확인해서 빌드를 해줌

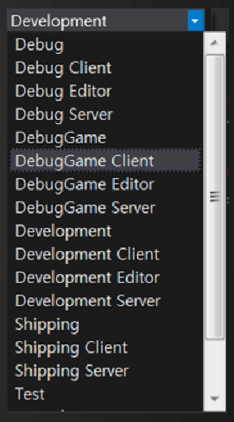
UE4 빌드 시스템

- VS에서 선택가능한 Solution Configurations 살펴보기
  - 기본적으로 중요한 Solution Configurations는
    Debug / Development
    DebugGame
    Debug Editor / Development Editor
    Shipping
  - Debug / Development
    - 엔진코드와 게임 코드 모두가 같은 환경으로 빌드됨
    - Monolithic 빌드로 빌드된 바이너리 파일은 프로젝트/Binarie
      에 생성됨
    - 실행에는 쿠킹된 컨텐츠가 필요



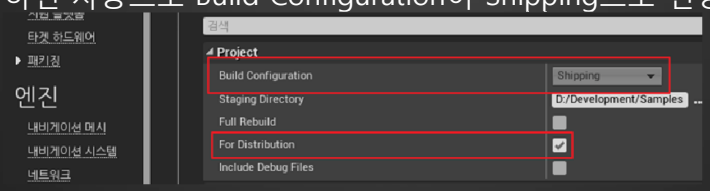
UE4 빌드 시스템

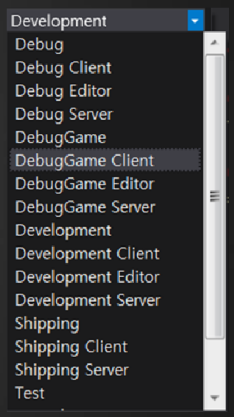
  - Debug Editor / Development Editor
    - 엔진 모듈들과 게임모듈들이 DLL 모듈들로 빌드 생성됨
    - Debug / Development 와는 다르게
      Engine₩Binaries₩Win64₩UE4Editor.exe (Development설정)
      가 실행되어 모듈들을 로딩하는 방식
    - 쿠킹되지 않은 원본 uasset들이 사용됨
  - DebugGame / DebugGame Editor
    - 개발을 더 빠르게 하기 위해서 엔진은 Development로
      게임만 Debug로 빌드되는 설정
    - 엔진 디버깅은 필요 없이 게임만 디버깅 할 경우 추천



UE4 빌드 시스템

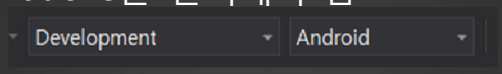
- Shipping
  - Debug / Development 와 같은 Monolithic 빌드
  - 실제 게임 출시용 바이너리이기에 로그나 기타 불필요한 모든 소스코드와 기능이 제거되어 만들어짐
  - 가장 작고 빠른 바이너리 파일
  - 프로젝트 세팅에서 패키징에 For Distribution을 true로 선택하면 자동으로 Build Configuration이 Shipping으로 변경됨





안드로이드 빌드

- 안드로이드 빌드는 Editor가 없는 Debug나 Development Solution Configurations를 선택해야 함



- VS에서 빌드를 하면 APK까지 생성이 됨
  - APK 생성은 UEDeployAndroid.cs에 MakeAPK 함수에서 생성됨
  - 4.7 에서는 APK 생성시 Java 빌드 관련 로그가 완전히 출력되지 않은 이슈가 있고, 이 부분은 UEDeployAndroid.cs에서

```
RunCommandLineProgramAndThrowOnError(UE4BuildPath, "cmd.exe", "/c ₩"" + GetAntPath() + "₩" " +
AntBuildType, "Making .apk with Ant... (note: it's safe to ignore javac obsolete warnings)");
```

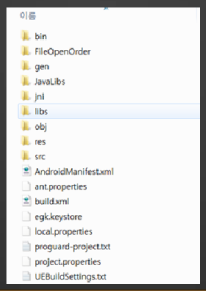
위와 같이 -quiet 옵션을 제거하면 완전한 로그를 볼 수 있음! 4.8에서는 수정됨

안드로이드 빌드

- 안드로이드 빌드까지 되는 것이기 때문에 안드로이드 프로젝트가 필요하고, 안드로이드를 빌드하면, 그 안드로이드 프로젝트는

  게임프로젝트₩Intermediate₩Android₩APK

  폴더에서 찾을 수 있음



안드로이드 빌드

- 이 안드로이드 프로젝트는 어떻게 생성되는 것일까?
  - 당연히 UnrealBuildTool이 처리해주고, 이것도 UEDeployAndroid.cs에서 처리됨
- 그렇더라도 기본적은 Java 코드 파일과 AndroidManifest.xml 등등은 어디서 오는지?
- 기본적인 안드로이드 Java 파일과 리소스들은

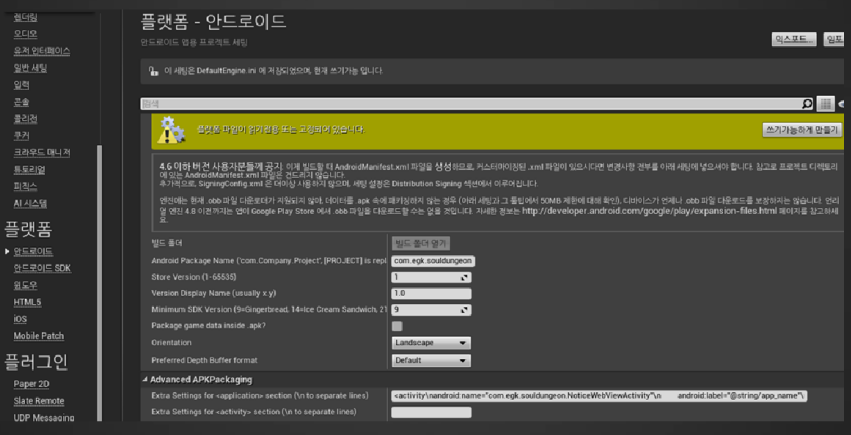
  Engine₩Build₩Android₩Java

  폴더에 있는 내용들을 APK 폴더로 복사해서 세팅됨

안드로이드 빌드

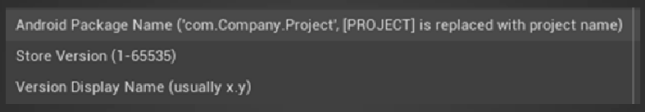
- 기본적인 Activity 클래스는 Engine₩Build₩Android₩Java₩src₩com₩epicgames₩ue4₩GameActivity.java
- 그런데 해당 폴더를 봐도 AndroidManifest.xml 파일은 없음
- AndroidManifest.xml 파일은 프로젝트 세팅과 UEDeployAndroid.cs 에 GenerateManifest 함수 내부에 코드를 기준으로 생성됨

안드로이드 빌드



안드로이드 빌드

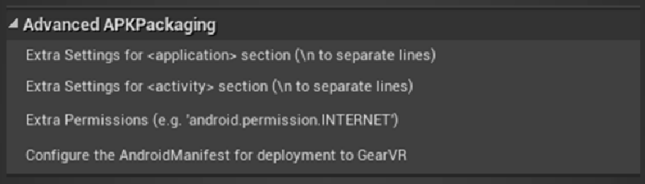
- 안드로이드 - 플랫폼 항목 세팅들



  - Android Package Name: 유니크한 패키지 내용 설정
    - AndroidManifest에 package= 항목
  - Store Version: 내부적으로 사용하는 앱 버전 세팅
    - AndroidManifest에 android:versionCode 항목
  - Version Display Name: 유저에게 보이는 버전 이름
    - AndroidManifest에 android:versionName 항목

안드로이드 빌드

- 안드로이드 - 플랫폼 항목 세팅들



  - Extra Settings for <application> section
    - AndroidManifest에 <application> 항목 안에 들어가는 내용으로 일반적으로 추가적인 Activity를 넣을 때 사용됨
  - Extra Settings for <activity> section
    - 기본적인 Activity에 추가적으로 변경하고 싶은 옵션이 있는 경우 여기에 추가

안드로이드 빌드

- 안드로이드 - 플랫폼 항목 세팅들
  - Extra Permissions
    - 기본적으로 아래 Permission들은 추가됨. 따로 추가 필요 없음

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="com.android.vending.CHECK_LICENSE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="com.android.vending.BILLING"/>
```


네이티브 코드 연동

네이티브 코드 연동

- UE4는 네이티브 코드로 이루어져 있기 때문에 안드로이드 Java 코드와 서로 통신하기 위해서 JNI 사용이 필수
- 기본적은 GameActivity.java의 함수들은 AndroidJNI.cpp에서 JNI 관련한 설정이 되어 있고, 호출용 함수들을 볼 수 있음
- 그렇다면 내가 추가해야 하는 Java 코드와 네이티브 코드는 어떻게 연동?
- 좀 더 쉬운 JNI 사용을 위한 클래스 지원 FJavaClassObject
- JNI를 위한 Java 함수 찾기 및 호출 하기를 위한 여러가지 편리한 함수들을 제공

네이티브 코드 연동 - FJavaClassObject

- 예제: AndroidJavaMessageBox.h 와 MessageBox01.java
- Java 코드와 FJavaClassObject를 상속받은 네이티브 클래스가 한쌍
  class MessageBox01 implements View.OnClickListener
  class FJavaAndroidMessageBox : public FJavaClassObject
- Java 코드에서 JNI로 호출되는 함수들은 전부 public 으로 선언해야함
  - public void setCaption(String text)

네이티브 코드 연동 - FJavaClassObject

- FJavaClassObject를 상속받은 네이티브 클래스에 (FJavaAndroidMessageBox) Java 함수 호출을 위한 FJavaClassMethod 변수 선언

```
public void setCaption(String text)
```

을 위해서

```
FJavaClassMethod SetCaptionMethod;
```

선언

네이티브 코드 연동 - FJavaClassObject

- FJavaAndroidMessageBox 생성자에서 연결되는 Java 클래스 설정과 FJavaClassMethod들 설정

```
FJavaAndroidMessageBox::FJavaAndroidMessageBox()
    : FJavaClassObject(GetClassName(), "()V")
    , SetCaptionMethod(GetClassMethod("setCaption", "(Ljava/lang/String;)V"))
    , SetTextMethod(GetClassMethod("setText", "(Ljava/lang/String;)V"))
    , AddButtonMethod(GetClassMethod("addButton", "(Ljava/lang/String;)V"))
    , ClearMethod(GetClassMethod("clear", "()V"))
    , ShowMethod(GetClassMethod("show", "()I"))
    {
    }
```

네이티브 코드 연동 - FJavaClassObject

- FJavaAndroidMessageBox 생성자에서 연결되는 Java 클래스 설정을 위한 GetClassName 함수

```
FName FJavaAndroidMessageBox::GetClassName()
{
	if (FAndroidMisc::GetAndroidBuildVersion() >= 1)
	{
		return FName("com/epicgames/ue4/MessageBox01");
	}
	else
	{
		return FName("");
	}
}
```


네이티브 코드 연동 - FJavaClassObject

- 게임 내에 네이티브 코드에서 호출 할 수 있는 네이티브 함수들 추가하고 해당 함수에서 각각의 JNI 함수 호출

```
void FJavaAndroidMessageBox::SetCaption(const FString & Text)
{
	BindObjectToThread();
	CallMethod<void>(SetCaptionMethod, GetJString(Text));
}
```

- 이제 SetCaption 함수를 네이티브 코드에서 호출하면 MessageBox01.java 에 setCaption 함수 호출됨! 쉽다(?) 편하다!!

네이티브 코드 연동 – Java -> 네이티브 코드

- FJavaClassObject를 통해서 좀 더 편하게 네이티브 코드에서 Java 함수 호출이 가능
- 그렇다면 역으로 Java에서 네이티브 코드 호출은?

네이티브 코드 연동 – Java -> 네이티브 코드

- 예제: OnlineStoreInterfaceGooglePlay.cpp와 GooglePlayStoreHelper.java
- Java 코드에서는 native public 함수를 선언

```
public native void nativePurchaseComplete(boolean bSuccess, String ProductId, String ReceiptData);
```

- 네이티브 코드에서는

```
extern "C" void Java_com_epicgames_ue4_GooglePlayStoreHelper_
nativePurchaseComplete
(JNIEnv* jenv, jobject thiz, jboolean bSuccess, jstring productId, jstring receiptData)
```

네이티브 코드 연동 – Java -> 네이티브 코드

- 네이티브 함수 선언 구조는

extern "C" void Java_com_epicgames_ue4_GooglePlayStoreHelper_nativePurchaseComplete(JNIEnv* jenv, jobject thiz, jboolean bSuccess, jstring productId, jstring receiptData)

- 기본: extern “C” void Java_
- 클래스 패키지 이름: com_epicgames_ue4_
- 클래스 이름: GooglePlayStoreHelper_
- 함수 이름: nativePurchaseComplete
- 필수 기본 파라미터: JNIEnv* jenv, jobject thiz
- 나머지 파라미터의 선언한 함수의 파라미터

네이티브 코드 연동 – Java -> 네이티브 코드

- 호출을 받아야 하는 클래스는 Singleton 구조를 통해서 호출 받도록 구조를 잡아주면 됨
- 그래서 추가한 Java와 네이티브 코드가 서로 통신을 해야 할 때 FJavaClassObject를 상속받은 네이티브 코드를 만들고, Java에서 네이티브 코드 호출이 필요한 부분은 추가한 FJavaClassObject 클래스의 cpp 파일에 Java가 호출하는 네이티브 코드를 추가
- 이렇게 하면 완벽하가 Java <-> 네이티브 코드 구조 완성!

네이티브 코드 연동 – Java -> 네이티브 코드

- Java -> 네이티브 코드 호출시에 조심해야 하는 것은 네이티브 코드 쪽으로 데이터를 넘기거나 함수를 호출 시에 Java 스레드에서 호출 되기 때문에 thread safe 하지 않아서 크래쉬가 발생할 수 있음
- 그래서 Java -> 네이티브 코드 함수에서 넘어온 호출이 게임스레드에서 안전하게 호출하게 해주어야 함

네이티브 코드 연동 – Java -> 네이티브 코드

- Java_com_epicgames_ue4_GooglePlayStoreHelper_nativePurchaseComplete 함수는 좋은 예제
- FSimpleDelegateGraphTask::CreateAndDispatchWhenReady(..., ENamedThreads::GameThread) 함수호출을 통해서 GameThread에서 안전하게 호출되게 할 수 있음

```
FSimpleDelegateGraphTask::CreateAndDispatchWhenReady(
    FSimpleDelegateGraphTask::FDelegate::CreateLambda([=](){
        FPlatformMisc::LowLevelOutputDebugStringf(TEXT("In-App Purchase was completed  %s\n"), bSuccess ? TEXT("successfully") : TEXT("unsuccessfully"));
        if (IOnlineSubsystem* const OnlineSub = IOnlineSubsystem::Get())
        {
            FPlatformMisc::LowLevelOutputDebugStringf(TEXT("2... ProductId: %s, ReceiptData: %s\n"), *ProductId, *ReceiptData);
            // call store implementation to process query results.
            if (FOnlineStoreGooglePlay* StoreInterface = (FOnlineStoreGooglePlay*)OnlineSub->GetStoreInterface().Get())
            {
                StoreInterface->ProcessPurchaseResult(bSuccess, ProductId, ReceiptData);
            }
        }
    }),
    GET_STATID(STAT_FSimpleDelegateGraphTask_ProcessIapResult),
    nullptr,
    ENamedThreads::GameThread
);
```

네이티브 코드 연동

- 어? 그런데 내가 추가한 Java 파일은 어떻게 APK에 포함되게 하지???
- 엔진 코드 수정이 필요 없는 아주 편리한 인터페이스 지원!
- 게임프로젝트₩Build₩Android 있는 폴더들은 전부 APK가 만들어지는 폴더로 복사됨

네이티브 코드 연동

- 즉, 소스코드의 경우
  게임프로젝트₩Build₩Android₩src₩com₩test₩testproject₩TestJava.java
  이렇게 추가해주면 APK에 포함되어서 빌드됨
- 여기서 추가적으로 UE4도 proguard를 사용해서 코드 난독화를 하는데 JNI함수들은 난독화 되면 안되기때문에
  게임프로젝트₩Build₩Android₩proguard-project.txt
  파일을 열어서 예외 추가
  예를들면:

```
-keep class com.test.testproject.TestJava {
   public *;
}
```

외부 라이브러리 추가하기

- 외부 라이브러리 추가도
  게임프로젝트₩Build₩Android
  폴더를 사용하면 전과 같이 엔진 코드 수정없이도 추가가 간편함!

- Lib 파일만 있는 라이브러리의 경우
  게임프로젝트₩Build₩Android₩libs
  에 .jar 파일 넣어두면 APK에 자동으로 추가됨

- 프로젝트 단위로 추가되는 외부 라이브러리의 경우는
  게임프로젝트₩Build₩Android₩JavaLibs
  폴더에 그 라이브러리 프로젝트 폴더를 복사해두면 추가와 빌드 및
  레퍼런스도 자동으로 됨. Engine₩Build₩Android₩Java₩JavaLibs 참고

안드로이드 빌드 및 테스트

안드로이드 빌드 및 테스트

- 네이티브 코드를 수정 했거나 Java 코드를 수정했다면 빌드가 되어야 하고, 설치 및 테스트가 필요
- 안드로이드 앱은 쿠킹된 리소스가 필수로 필요하기 때문에 에디터에서 "실행"이나 프로젝트 런쳐를 통해서 쿠킹 및 앱 설치가 필수
- 혹시나 TADP를 사용하지 않으신다면, TADP를 설치 및 사용 강력 추천!
  - 추천 버전은 tadp-3.0r4-windows 으로 https://developer.nvidia.com/tegra-android-development-pack 가입해서 받으실 수 있음.
  - 더 최신 버전 4.x 버전은 호환성의 문제가 있는 것으로 보여서 아직은 비추

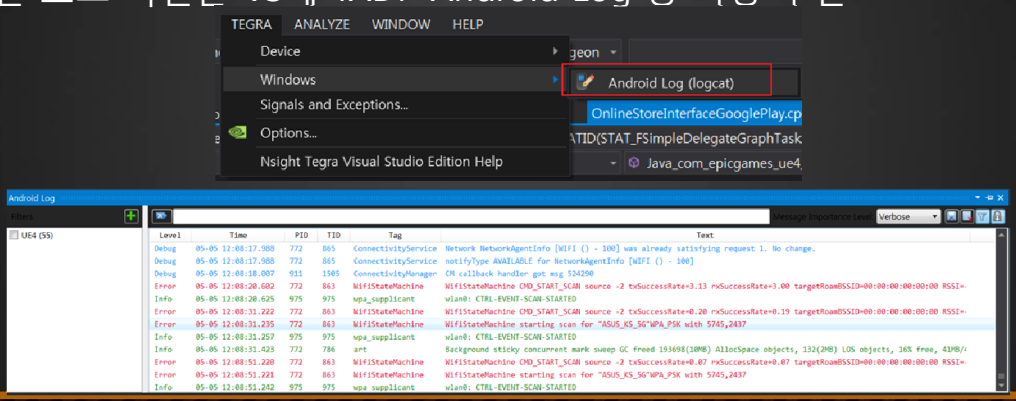
안드로이드 빌드 및 테스트

- 안드로이드 코드 수정과 컨텐츠 수정이 같이 있는 경우는 에디터를 열어둔 상태에서 안드로이드 코드 및 컨텐츠 수정 후에 "실행" 으로 테스트 추천
  - "실행"이 코드 빌드를 하기 때문에 VS에서 빌드는 불필요
- 컨텐츠 수정이 없고, 코드 수정만 하는 경우라면 VS만 이용을 추천
- 코드 수정 후에 Ctrl+F5 를 하면 빌드, APK 생성 설치 및 실행까지 전부 수행됨
  - 그나마 빠른 iteration 타임

UNREAL ENGINE UNREAL SUMMIT 2015

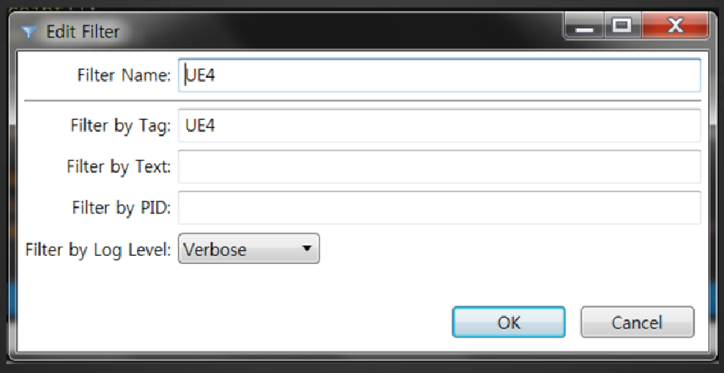
안드로이드 빌드 및 테스트

- 실행 시에 크래쉬가 난 경우 콜스택 확인이나 정상 동작 여부를 위한 로그 확인은 VS에 TADP Android Log 창 사용 추천



안드로이드 빌드 및 테스트

- Android Log창에서 UE4 로그만 필터링 해서 볼 수 있도록 UE4 필터 추가 강력 추천!



안드로이드 빌드 및 테스트

- 크래쉬 콜스택의 경우 현재 기본 logcat으로 출력하는 것이 가장 확실한 크래쉬 정보를 보여주기 때문에 UE4 필터로는 정확한 콜스택을 보실 수 없음
- 콜스택이 표시되는 DEBUG Tag 필터를 추가해서 크래쉬 콜스택을 쉽게 확인 할 수 있도록 하는 것도 추천 드리는 방법

안드로이드 빌드 및 테스트

- 디바이스에도 로그가 기록됨
  - 디바이스 내부저장소에 게임프로젝트₩게임프로젝트₩Saved₩Logs 폴더에 로그가 있음
  - 단, UE4 Tag의 로그만 기록이 됨
  - 4.7에서는 크래쉬시에 전체 로그가 출력되지 않은 상태에서 로그 출력이 멈추는 이슈가 있고, 4.8에서 해결됨

UNREAL ENGINE UNREAL SUMMIT 2015

안드로이드 디버깅

안드로이드 디버깅

- TADP의 힘을 이용해서 가능!
- 단, 안드로이드 디바이스가 OS버전이나 기타 디바이스별로 케바케로 되고 안되고 함
- 가장 확실한 지원 디바이스는 역시나 Tegra 디바이스
- 그 이외에 넥서스 시리즈가 그나마 믿을 만 하고, 종종 다른 디바이스도 디버깅 가능 디바이스들이 있음

안드로이드 디버깅

- 필요한 기본 설정들
  - VS 2013
  - tadp-3.0r4-windows 버전
    - tadp 재설치시에는 전에 버전을 uninstall하고 전에 버전 폴더를 완전히 삭제
    - tadp 다시 설치 후에는 GenerateProjectFiles.bat등으로 프로젝트 파일 재생성 필요
  - 혹시나 JDK를 따로 설치하셨다면 전부 uninstall해주셔야함
    - Tadp와 설치되는 JDK만 사용되어야 됨
  - 디버깅에 사용할 프로젝트는 패스에 공백이 있으면 안됨
    - 즉, D:₩Unreal Projects₩AndroidProject 이렇게 Unresl Projects와 같은 공백이 있으면 안 됨
  - Monitor.bat나 logcat등 다른 adb 사용 툴이 실행되고 있으면 안됨
  - 게임 실행에 필요한 기본 컨텐츠들 "실행"등으로 설치되어 있어야 함
  - Debug Android 빌드 설정을 추천

안드로이드 디버깅

- 기본 설정 후에 Debug Android에서 F5를 눌러서 디버깅을 시작하면 디버깅 시도를 하게 됨
- Tegra 디바이스를 제외하고 디버깅 시작시에 exception이 처음에 한 번 나는데 그것은 정상 continue를 눌러주면 됨
- 아무래도 디버깅이 VS와 GDB등등과의 연동이기에 디버깅시에 VS 크래쉬가 발생하는 경우가 종종 있을 수 있음
- 로그 디버깅과 적절히 섞어서 디버깅을 하시기를 추천!

감사합니다!
Q/A