ANDROID NDK FOR DEVELOPMENT IN C/C++

By: José Aarón López García

Introduction

Android applications are commonly developed in Java using the Android Virtual Machine in DEX bytecode format. However there may cases where developing in native C/C++ code is needed or is optimally preferable.

Some of the features we can enjoy when using C/C++ are the following:

- More control of the hardware and OS: all the standard Linux syscalls plus some Android syscalls are either only available for native code or you can fully customize them without restrictions that apply to Java code.

- Legacy software support: some libraries, algorithms and software written in native code can be used to perform various tasks that would be complicated or impossible using Java. This includes advanced networking utilities such as Ping or Traceroute as well as emulators, games or domain-specific algorithms code written in C/C++ that can be ported to Android without requiring a full rewrite.

- Faster and lighter execution: though the Virtual Machine featured in the Android OS is extremely well optimized for fast code execution, nothing ever beats running native code without having an interpreter or a JIT compiler running behind your code. This is very important for tasks and softwares where speed is extremely important (games, emulators, complicated algorithms, etc).

- Multiplatforming: though at first sight it might seem like the opposite, C/C++ code written for Android can be easily ported/compiled to run (also natively) in other Desktop Linux OS such as Ubuntu with little to no modifications needed. This is thanks to the fact that Android shares all the standard Linux syscalls and C libraries such as OpenGL or SDL2. No emulation or Virtual Device is needed to achieve this.

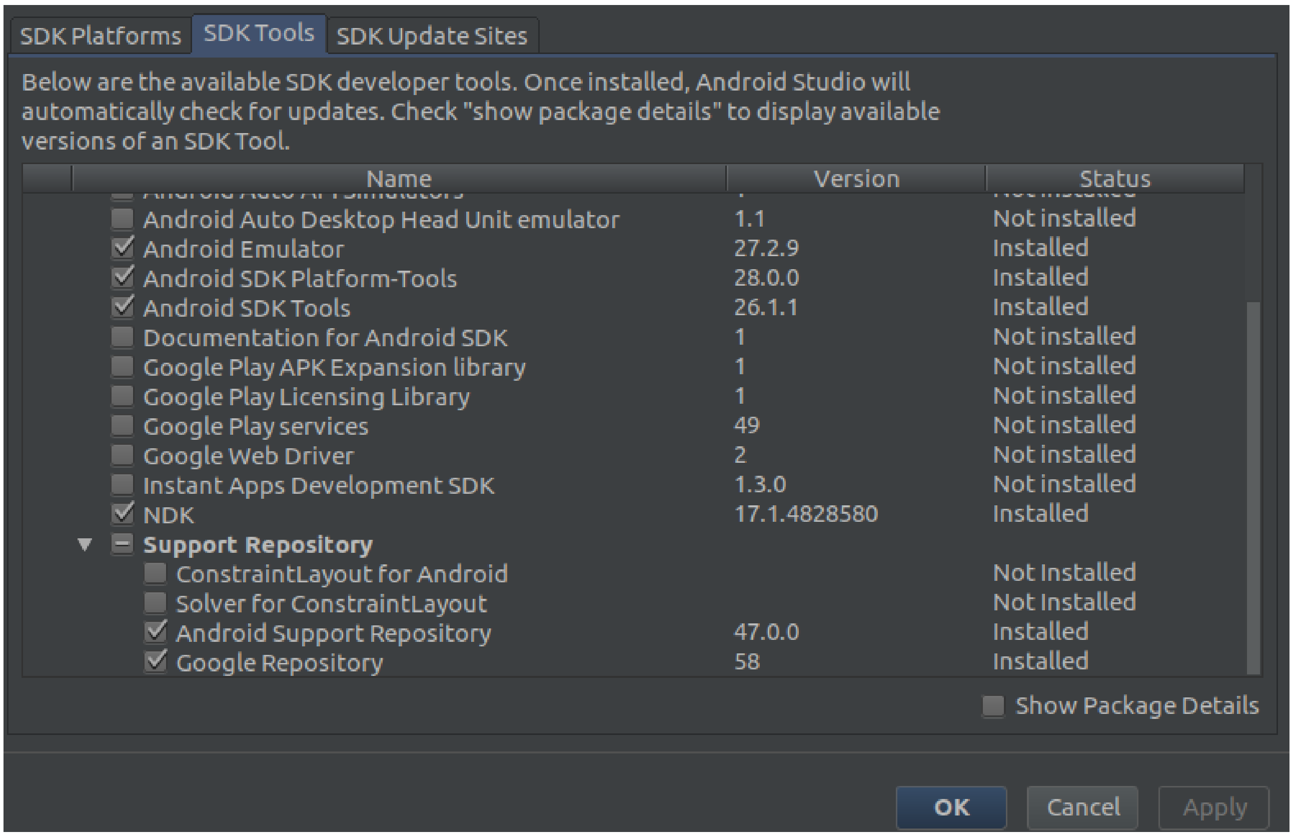
- Compatibility with Java libraries: thanks to the Java Native Interface (JNI), communication between C/C++ code and Java code is bidirectional, so your C/C++ code can call Android Java code and activities to make full use of the Android RunTime (ART). You don't loose the Android's Java capabilities and features!

Being able to develop applications in C/C++ for Android without loosing the ability to use JVM/ART opens the gate for a whole new world of features that enrich your coding experience.

<u>Native Development Kit</u>

The standard SDK to develop C/C++ applications in Android is called the Android NDK (Native Development Kit), and it makes use of the Java Native Interface (JNI), a library/feature of Java that allows to execute C/C++ code, which in term can also access Java code and libraries (including classes and methods).

To install the NDK we simply open the SDK Manager using Android Studio and navigate to the SDK Tools tab.



Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
Android Auto API Simulators	1	Not installed
☐ Android Auto Desktop Head Unit emulator	1.1	Not installed
✓ Android Emulator	27.2.9	Installed
✓ Android SDK Platform-Tools	28.0.0	Installed
✓ Android SDK Tools	26.1.1	Installed
☐ Documentation for Android SDK	1	Not installed
☐ Google Play APK Expansion library	1	Not installed
☐ Google Play Licensing Library	1	Not installed
☐ Google Play services	49	Not installed
☐ Google Web Driver	2	Not installed
☐ Instant Apps Development SDK	1.3.0	Not installed
✓ NDK	17.1.4828580	Installed
▼ − **Support Repository**		
☐ ConstraintLayout for Android		Not Installed
☐ Solver for ConstraintLayout		Not Installed
✓ Android Support Repository	47.0.0	Installed
✓ Google Repository	58	Installed

☐ Show Package Details

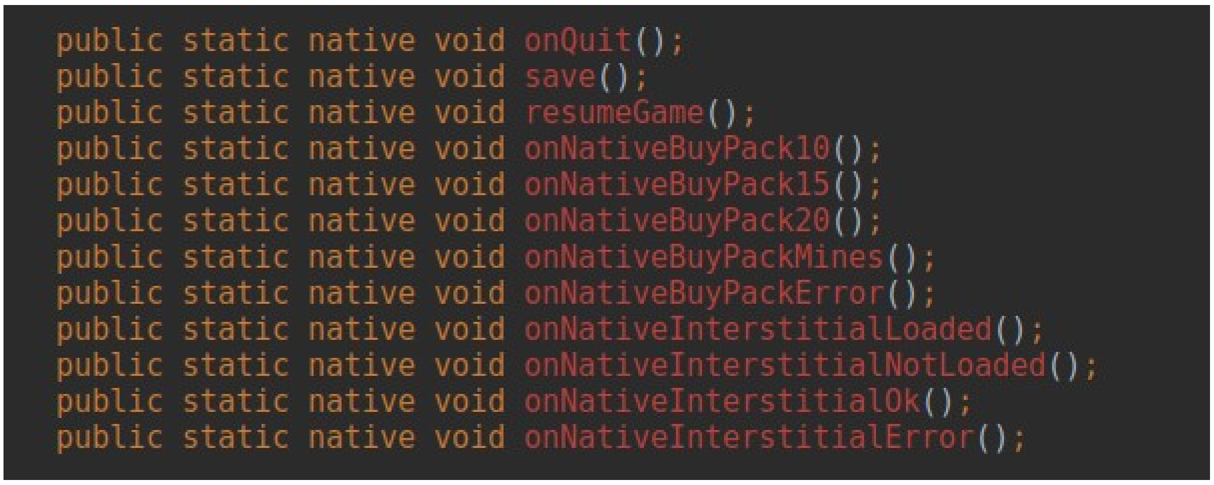
The NDK includes a C/C++ compiler (clang), development files (headers and precompiled shared libraries) and the necessary tools to generate full functioning C/C++ code for Android.

Installation is straightforward and independent on the Android SDK version you use; code written and compiled for the NDK will work with all Android SDK versions! So no need to install a new NDK each time you install an SDK for a given Android version.

Java Native Interface

The JNI allows us to issue calls between Java and C++ code.

In your Java class you can declare member methods to be implemented in C/C++ using the “native” keyword and issue calls to them as you normally would to Java methods.

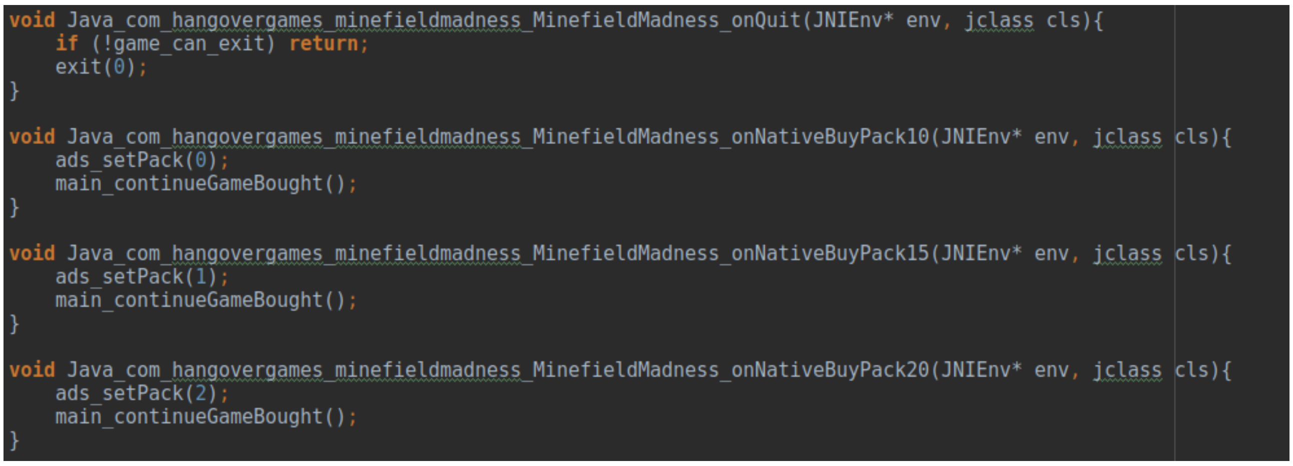


```java
public static native void onQuit();
public static native void save();
public static native void resumeGame();
public static native void onNativeBuyPack10();
public static native void onNativeBuyPack15();
public static native void onNativeBuyPack20();
public static native void onNativeBuyPackMines();
public static native void onNativeBuyPackError();
public static native void onNativeInterstitialLoaded();
public static native void onNativeInterstitialNotLoaded();
public static native void onNativeInterstitialOk();
public static native void onNativeInterstitialError();
```

In your C/C++ code you must implement each of these function using the following criteria:

- The function name must be Java_package_name_ClassName_methodName.

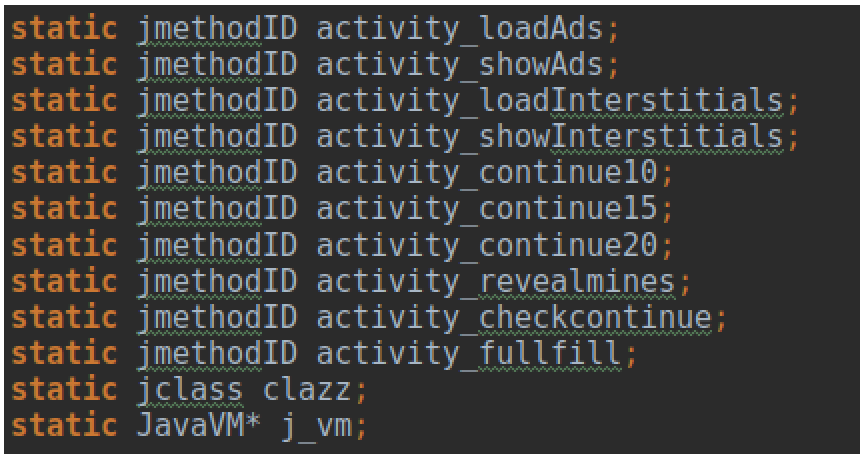
- The function will have two mandatory arguments: JNIEnv* and jclass.



```c
void Java_com_hangovergames_minefieldmadness_MinefieldMadness_onQuit(JNIEnv* env, jclass cls){
    if (!game_can_exit) return;
    exit(0);
}

void Java_com_hangovergames_minefieldmadness_MinefieldMadness_onNativeBuyPack10(JNIEnv* env, jclass cls){
    ads_setPack(0);
    main_continueGameBought();
}

void Java_com_hangovergames_minefieldmadness_MinefieldMadness_onNativeBuyPack15(JNIEnv* env, jclass cls){
    ads_setPack(1);
    main_continueGameBought();
}

void Java_com_hangovergames_minefieldmadness_MinefieldMadness_onNativeBuyPack20(JNIEnv* env, jclass cls){
    ads_setPack(2);
    main_continueGameBought();
}
```

The package name is the same as the class’ package that declares the native methods, but the dots (.) must be replaced with underscore (_), so the native function onQuit() declared in class MinefieldMadness of package com.hangovergames.minefieldmadness will be named Java_com_hangovergames_minefieldmadness_MinefieldMadness_onQuit() in the C/C++ source code.

Issuing calls to Java code from C/C++ code can be done by accessing the Java Virtual Machine and obtaining the instance for the class and methods you wish to call.
For that we use the C data types JavaVM*, jclass and jmethodID.



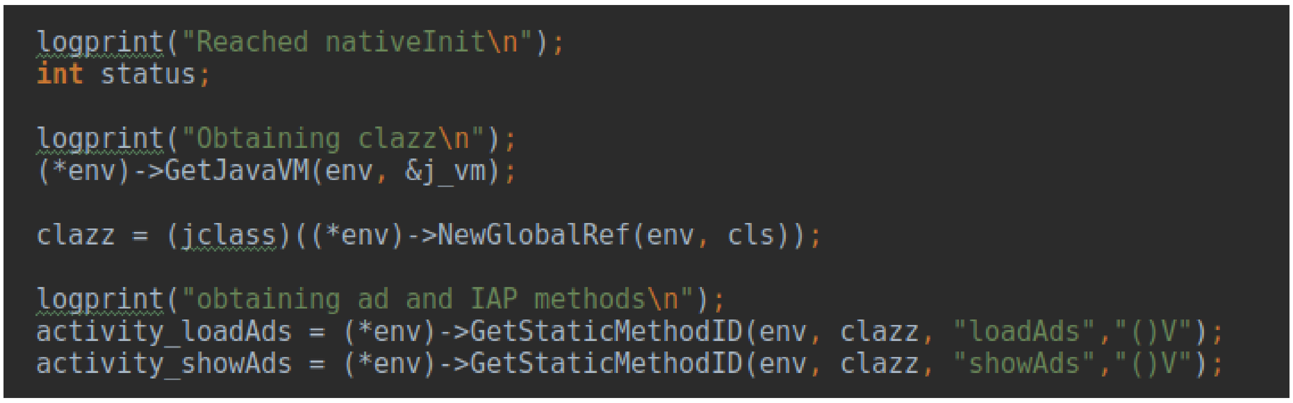
```
static jmethodID activity_loadAds;
static jmethodID activity_showAds;
static jmethodID activity_loadInterstitials;
static jmethodID activity_showInterstitials;
static jmethodID activity_continue10;
static jmethodID activity_continue15;
static jmethodID activity_continue20;
static jmethodID activity_revealmines;
static jmethodID activity_checkcontinue;
static jmethodID activity_fullfill;
static jclass clazz;
static JavaVM* j_vm;
```

- JavaVM: the pointer to the instance of the Java VM being used by your app’s process.

- jclass: data type that represents a specific class from the VM.

- jmethodID: data type that identifies the method to be executed.

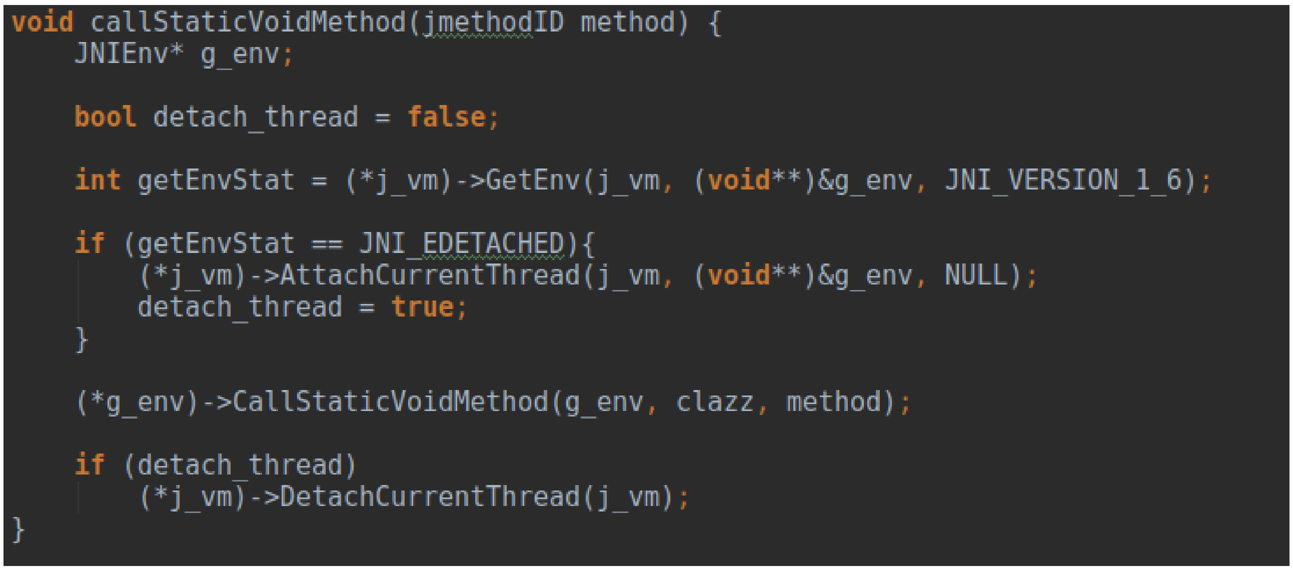
Obtaining such information can be done with the following code:



```
logprint("Reached nativeInit\n");
int status;

logprint("Obtaining clazz\n");
(*env)->GetJavaVM(env, &j_vm);

clazz = (jclass)((*env)->NewGlobalRef(env, cls));

logprint("obtaining ad and IAP methods\n");
activity_loadAds = (*env)->GetStaticMethodID(env, clazz, "loadAds","()V");
activity_showAds = (*env)->GetStaticMethodID(env, clazz, "showAds","()V");
```

Note: we use “clazz” as the variable name to prevent conflict with C++’s reserved keyword “class”.

Issuing calls to the Java method requires to register the current thread with the JVM so that it can be handled correctly by Java. It may already be registered, so we have to check that too.
Once we’ve registered our thread, we can issue a call to CallStaticVoidMethod in the jenv variable (obtained through the JVM variable).



```
void callStaticVoidMethod(jmethodID method) {
    JNIEnv* g_env;

    bool detach_thread = false;

    int getEnvStat = (*j_vm)->GetEnv(j_vm, (void**)&g_env, JNI_VERSION_1_6);

    if (getEnvStat == JNI_EDETACHED){
        (*j_vm)->AttachCurrentThread(j_vm, (void**)&g_env, NULL);
        detach_thread = true;
    }

    (*g_env)->CallStaticVoidMethod(g_env, clazz, method);

    if (detach_thread)
        (*j_vm)->DetachCurrentThread(j_vm);
}
```

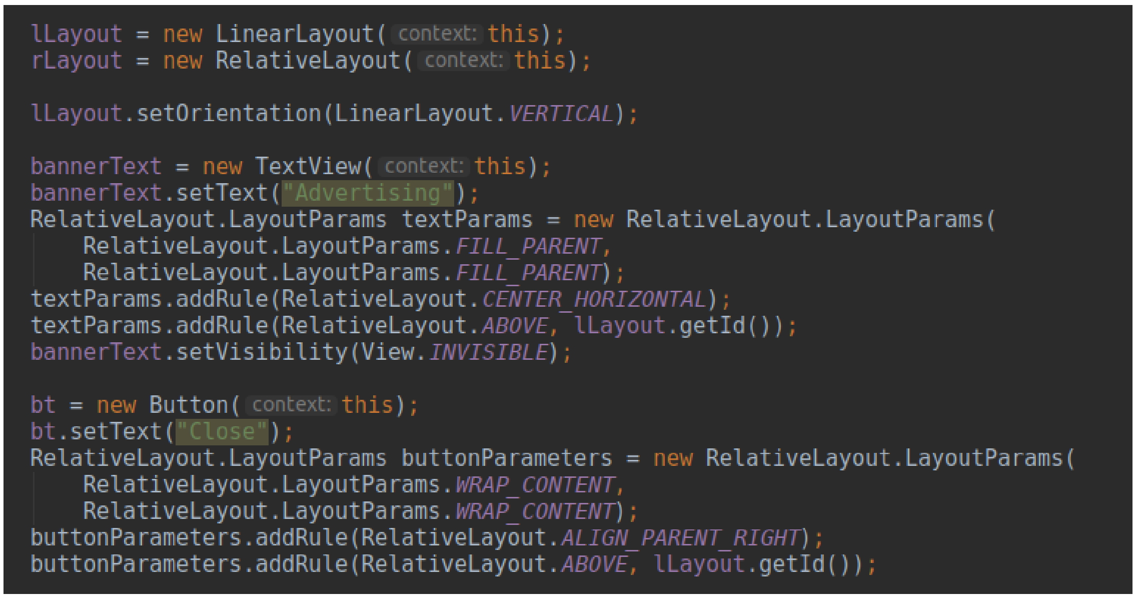
Once finished, our thread can be safely deregistered from the JVM.

Graphics and UI

OpenGL is the most common graphics library, being supported by a wide range of hardware and software.
Due to its low level complexity, many other libraries are built on top of OpenGL to ease development. Most notably SDL2; a 2D graphics library compatible with most common operating systems such as Windows, Linux or Android.

For more information on SDL2 you can visit the following page:
http://lazyfoo.net/tutorials/SDL/

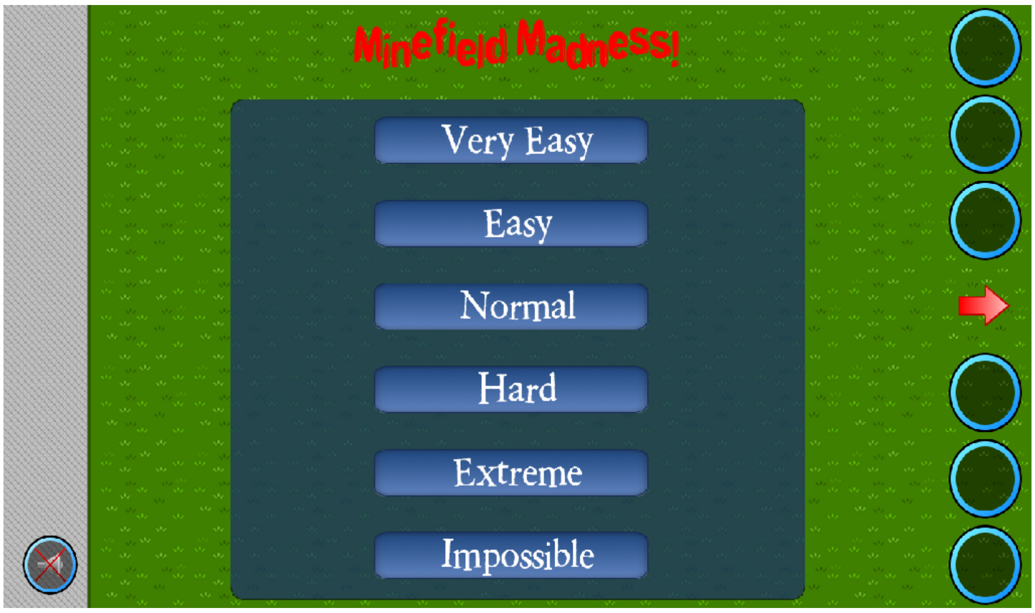
However doing graphics and UI on C/C++ is unnecessary, as we can always use the easier and more powerful Android UI system, either by means of XML files or by programmatically creating the UI.

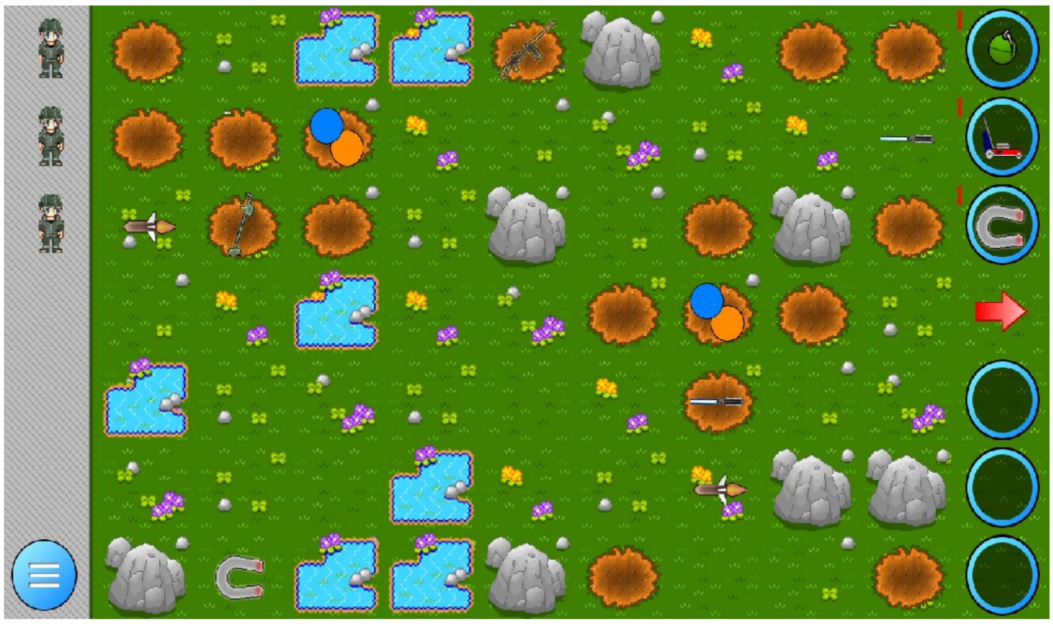


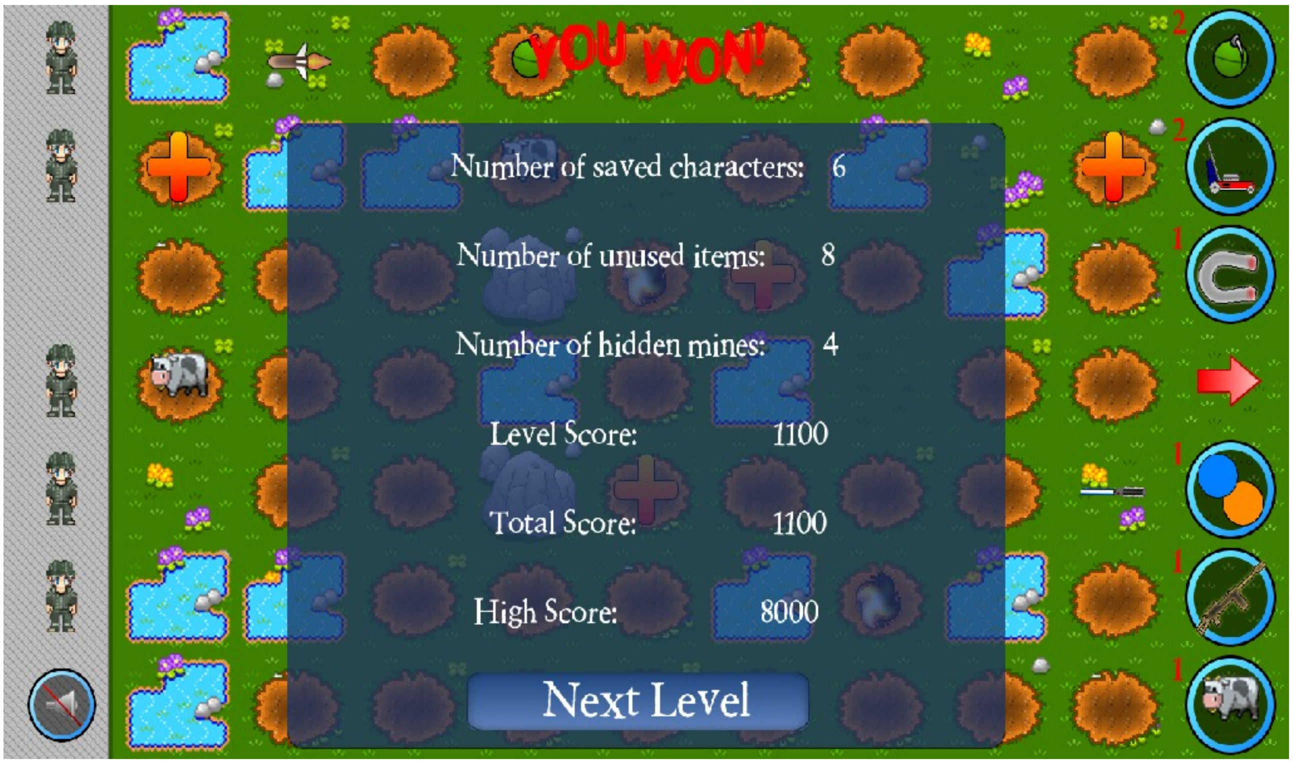
```
lLayout = new LinearLayout( context: this);
rLayout = new RelativeLayout( context: this);

lLayout.setOrientation(LinearLayout.VERTICAL);

bannerText = new TextView( context: this);
bannerText.setText("Advertising");
RelativeLayout.LayoutParams textParams = new RelativeLayout.LayoutParams(
    RelativeLayout.LayoutParams.FILL_PARENT,
    RelativeLayout.LayoutParams.FILL_PARENT);
textParams.addRule(RelativeLayout.CENTER_HORIZONTAL);
textParams.addRule(RelativeLayout.ABOVE, lLayout.getId());
bannerText.setVisibility(View.INVISIBLE);

bt = new Button( context: this);
bt.setText("Close");
RelativeLayout.LayoutParams buttonParameters = new RelativeLayout.LayoutParams(
    RelativeLayout.LayoutParams.WRAP_CONTENT,
    RelativeLayout.LayoutParams.WRAP_CONTENT);
buttonParameters.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
buttonParameters.addRule(RelativeLayout.ABOVE, lLayout.getId());
```

Sample Project

The proposed sample project to showcase C/C++ development for Android is a simple 2D-based game using SDL2.
The game is compiled with the NDK and makes use of Google's advertisement and In-App Purchases libraries to demonstrate the JNI's ability to communicate with our native C/C++ code.
The game is also compiled for Linux with little to no modifications done to the source code.







Number of saved characters: 6

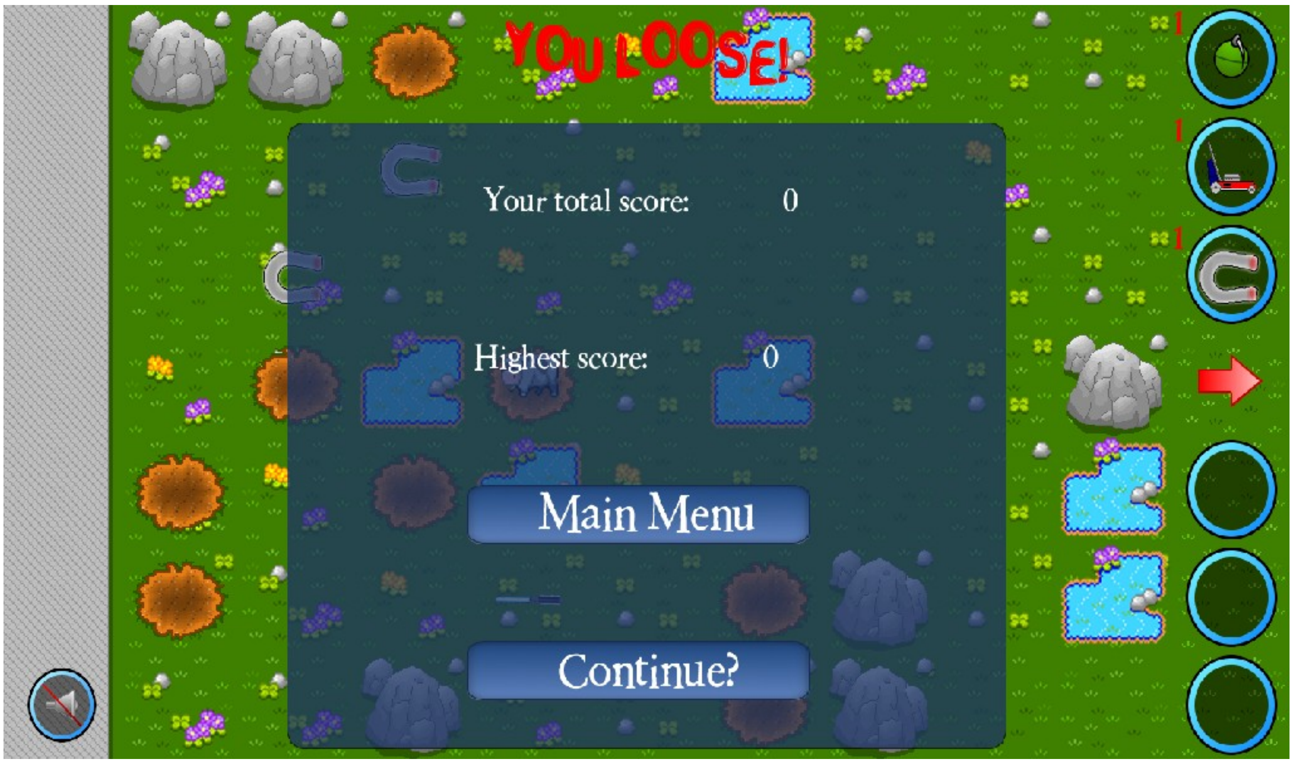
Number of unused items: 8

Number of hidden mines: 4

Level Score: 1100

Total Score: 1100

High Score: 8000



Your total score: 0

Highest score: 0

__Video Presentation__

A video presentation of the project can be found in the following URL:

https://youtu.be/Mok8A0RHBKE