PowerA MOGA FAQ

Version	Date	Author	Description
1.0	14-Feb-2012	SAM	Initial version.
1.1	24-Apr-2012	CE	Modifications and additions.
1.3.0	23-Jan-2013	MD & IB	MOGA Pro modifications and additions. General additional modifications.

1 SUMMARY

This document describes common questions from Android developers who wish to use the PowerA MOGA controller in their applications.

Additional Questions: email devsupport@PowerA.com

2 CONTENTS

3 QUESTIONS

3.1 Q: HOW DO I INSTALL THE ANDROID DEVELOPER TOOLS?

A: From the Android Developer website: http://developer.android.com/ you can link to the Android SDK http://developer.android.com/sdk/ and then the installation instructions and "Hello World" http://developer.android.com/sdk/installing.html.

3.2 Q: WHAT IS THE JDK?

A: The Java Development Kit is the developer edition of the Java Runtime Environment. It lets you develop Java applications and not just run them. To compile Android source code you will need to install this on your development machine. Typically this means you are using a Java tool to make a Java application. http://www.oracle.com/technetwork/java/javase/downloads/index.html is where you can download the JDK, you will need it.

3.3 Q: WHAT IS ECLIPSE, WHERE DO I GET IT AND WHICH VERSION DO I NEED?

A: Eclipse is an IDE and is the preferred development tool for making Android Apps.

You get it from http://developer.android.com/sdk/index.html#download

If you are running a Windows machine, click the "Download the SDK button" else, there are other platforms in the "DOWNLOAD FOR OTHER PLATFORMS". Following the instructions on the Android website will guide you through this installation and show you how to create your first android app in eclipse.

3.4 Q: CAN I USE THE ANDROID EMULATOR TO DEVELOP?

A: As the MOGA controller is a Bluetooth based controller you should use a real Android Phone or Tablet that has Bluetooth support for development and testing.

The controller library communicates to controller using a service. For end-users this service is installed by the Launcher application. But during development it is possible that the service is not running on the target device (phone or tablet).
To install the service, copy `com.bda.controller.service.apk` from the `redist` folder of this SDK to your phone/tablet, and use an android file manager to execute the program which will install the service.

You may need to install the USB drivers for your Android product to connect it to your developer tools and download any programs you are developing.

3.5 Q: WHAT IS THE LAUNCHER APPLICATION?

A: To help players connect the MOGA controller we have supplied a Launcher App that you can install. It contains the service program that handles Bluetooth communications with the controller and prompts players to enable Bluetooth and pair the MOGA for the first time.

3.6 Q: WHAT'S THE DIFFERENCE BETWEEN THE DIFFERENT SAMPLE PROJECTS PROVIDED?

A: The DEMO projects use only Java to demonstrate the controllers' features. These show best use of the controller. They show how to implement a controller using the polling and listener models. They also show how to map controls for both the MOGA (Pocket) and MOGA Pro depending on what version of the controller is connected.

The JNI projects use a mixture of Java and C/C++. For additional tutorials on how to use JNI please see http://developer.android.com/training/articles/perf-jni.html

The NATIVEACTIVITY use only C/C++.

Be aware you will need the Android NDK and the CDT plugin in eclipse to run the native activity and JNI projects.

3.7 Q: THE SAMPLE PROJECTS WILL NOT COMPILE IN ECLIPSE… WHAT IS WRONG?

Firstly examine the "Problems" tab in Eclipse, and address errors listed there (such as selecting Android Project / Fix project properties). If the "Problems" tab is not visible, enable it by going to the toolbar and clicking:

Window → Show View → Problems.

If you are using the JNI or NATIVEACTIVITY samples you may need to re-create the BUILDER command that compiles C/C++ code:

1. Bring up the project properties by right click on the package name and selecting ***PROPERTIES***
2. Select ***BUILDERS***.
3. Remove any invalid build tools.
4. Select ***NEW…*** and then ***PROGRAM***.
5. Rename ***NEW_BUILDER*** to ***C Make***.
6. Using the ***BROWSE FILE SYSTEM*** button, set ***LOCATION*** to be the location of `NDK-BUILD.CMD` from your NDK installation.
7. Use ***BROWSE WORKSPACE*** to set the ***WORKING DIRECTORY*** to be root of your project.
8. On the ***REFRESH*** tab, specify the `LIBS` directory refreshed.
9. On the ***BUILD OPTIONS*** tab, check ***DURING AUTO BUILDS***, and the ***SPECIFY WORKING SET*** to be the `JNI` folder.
10. The ***UP*** button to ensure ***C Make*** is before ***Android Package Builder***.
11. Select ***PROJECT / BUILD ALL.***

3.8 Q: THERE ARE TWO WAYS TO READ THE CONTROLLER, WHICH DO I USE TO ADD THE CONTROLLER TECHNOLOGY TO MY GAME APPLICATION?

A: If your project is a traditional game or console type application that has worked with a controller before; you can use the "***polling***" method to read the state of the MOGA controller once a game frame.

If you are developing an event driven game we have an event driven interface (“***listen***” mode). You can, instead use a controller listener which waits for various changes of button and joystick values before invoking code.

The library supports the use of both modes at the same time if your game requires it.

3.9 Q: WHAT IS A GOOD GAME TO DO A TEST INTEGRATION WITH?

A: Lunar Lander is a game that you can find in the samples folder within the android SDK. It can typically be found <android-sdk>\samples\android-8\LunarLander. If you don’t have an android sdk with APIs 8 or later, follow these instructions to download it http://developer.android.com/sdk/installing/adding-packages.html . It is a fairly typical multithreading example of a game using a thread for rendering and a main thread for the Activity that launches the game.

3.10 Q: HOW DO I MAKE A DEMO PROJECT FROM AN ANDROID DEMO EXAMPLE?

A: This link is helpful, http://developer.android.com/resources/samples/get.html. The text below is taken from it:

...if you are developing in Eclipse with the ADT Plugin, you can create a project for the "API Demos" sample app by starting a new Android Project, selecting "Create project from existing source", and then browsing to the <sdk>/samples/android-<level>/ApiDemos directory (the samples directory for the platform version you are using)...

3.11 Q: WHAT IS THE FIRST THING I DO TO GET THE MOGA CONTROLLER LIBRARY INTO MY APPLICATION?

A: You add the supplied com.bda.controller.jar file to your project; follow the instructions in the API documentation to do this before you start adding code.

Once the library is added, Eclipse automatically offers to add include statements when you reference the new MOGA controller library. You will see the new code in red. Click on it and accept the inclusion of the library into your source file.

3.12 Q: HOW DO I MODIFY LUNAR LANDER TO USE THE GAMEPAD?

1. Include the controller library (`redist\com.bda.controller.jar`) into the Lunar Lander project.

   If you are developing with Eclipse the following method is recommended:

   a. Create a “`libs`” folder in the project root.
   b. Copy `redist\com.bda.controller.jar` into `libs`.
   c. In Eclipse, right click on **LunarLander** in the **Project Explorer** window and select “**Refresh**”. `com.bda.controller.jar` should appear in the `libs` folder.

2. In **LunarLander.java:**

   a. Add the follow line to the imports section to include the Gamepad controller library:

```
import  com.bda.controller.Controller;
```

   b. After the line "`public class LunarLander extends Activity {`" insert the follow line of code to create a member variable for the game controller:

```
Controller mController = null;
```

   c. In **onCreate(Bundle savedInstanceState)**, after "`setContentView (R.layout.lunar_layout);`" Insert these two lines to initialize the controller object:

```
mController = Controller.getInstance(this);
mController.init();
```

   d. At the end of **onPause()** insert the follow code to handle when the activity pauses:

```
if(mController != null) {
        mController.onPause();
}
```

   e. Add the following two housekeeping functions to handle the controller objects destruction when the app is closed and its resumption when the app is opened from memory *after* the onPause() function:

```
@Override
protected void onDestroy() {
        if(mController != null) {
                mController.exit();
        }
        super.onDestroy();
}

@Override
protected void onResume() {
        super.onResume();
        if(mController != null) {
                mController.onResume();
        }
}
```

   f. In **onCreate()**, after "`mLunarThread = mLunarView.getThread()`" add this line so that later on we can add a function to pass the controller state to the activity:

```
mLunarThread.setController(mController);
```

3. In **LunarView.java**:

   a. Add the follow line to the imports section to include the Gamepad controller library:

```
import  com.bda.controller.Controller;
```

   b. Add the following code to the **`LunarThread`** class.

```
Controller Pad = null;

public void setController(Controller mController){
        synchronized (mSurfaceHolder) {
                Pad = mController;
        }
}
```

This is so there is a local **Controller** object to receive the **setController** reference from the main Actvity class call in **LunarLander.java.**
Now you can access the Controller from your View thread.

c. **Reading the controller status:**
The `updatePhysics()` function in LunarView.java is a great place to check the controller. You can cut and paste code in here, or to keep it neater you can make a call to a new function where you can add your code to read the controller.
Inside the function `updatePhysics()` Insert a function call:

```
checkController();
```

Then in the area just above the function definition insert the new function:

```
private void checkController() {
        if(Pad == null)
                return;

        if(Pad.getState(Controller.STATE_CONNECTION) != Controller.ACTION_CONNECTED)
                return;

        if(Pad.getKeyCode(Controller.KEYCODE_BUTTON_A) == Controller.ACTION_DOWN)
                setFiring(true);
        else
                setFiring(false);

        mRotating = 0;
        if(Pad.getKeyCode(Controller.KEYCODE_DPAD_LEFT) == Controller.ACTION_DOWN)
                mRotating = -1;
        if(Pad.getKeyCode(Controller.KEYCODE_DPAD_RIGHT) == Controller.ACTION_DOWN)
                mRotating = +1;

        if(Pad.getKeyCode(Controller.KEYCODE_BUTTON_START) == Controller.ACTION_DOWN)
                pause();

}
```

This was based on the example code in the MOGA Controller API and reads four buttons and calls existing functions or makes simple variable changes in the existing architecture.

It does not replace the old user interface it just makes simple use of Left, Right, A button and the Start button. In this example "Start" pauses / un-pauses. For more subtle control try using the Analogue values to affect rotation.

3.13 Q: ON MY ANDROID DEVICE I CAN'T START THE LUNAR LANDER GAME?

A: The in App menus are not accessible on some Android devices, typically tablets running 3.X OS. We have found you can modify the manifest.xml file in your Lunar Lander test App:

(1) Double click on your AndroidManifest.xml file in the Package Explorer of your Lunar Lander project
(2) When LunarLander Manifest opens in your edit window, click on the Tab at the bottom entitled "`Android.Manifest.xml`"
(3) Look for the text:

“android:theme=”@android:style/Theme.NoTitleBar”
and delete this from the file, save it using the File->Save option from the menus. Be careful to remove or alter no other text.

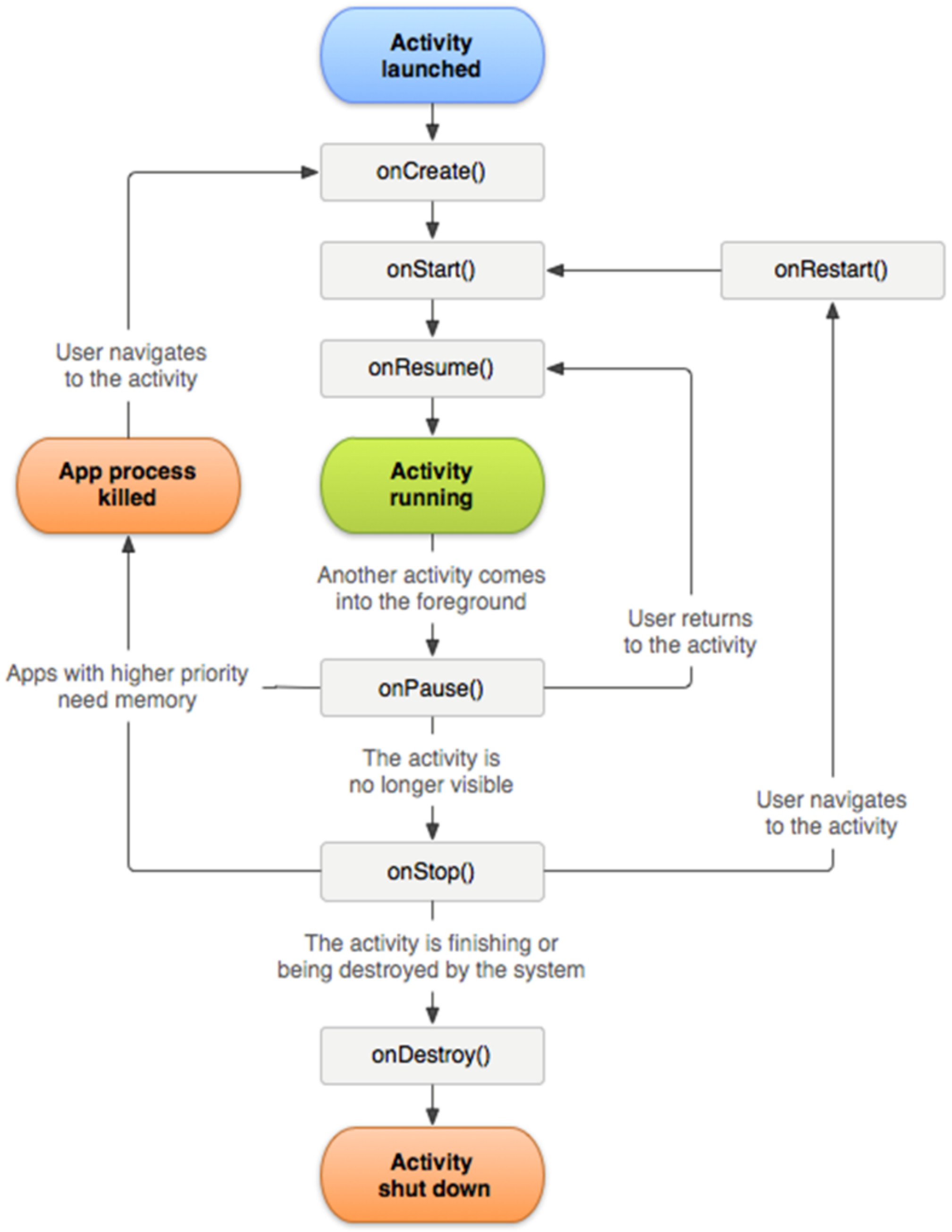
(4) Re-install your App and the Title Bar that contains the menu icon is now enabled. This lets you access the in-game menu in certain versions of Android OS (3.x) where it may have been moved to a virtual icon in the top right of the Title Bar.

3.14 Q: WHAT ARE THE ONCREATE(), ONDESTROY(), ONPAUSE() AND ONRESUME() CODE INSERTIONS FOR?

A: For those new to Android development we suggest you look at the developer.android.com online documentation and http://developer.android.com/reference/android/app/Activity.html

From this section we have taken the below Activity Lifecycle image helping show how the main Activity in an Android App is expected to work.

http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

This shows you how an Android Activity works. Here you can see where you are asked to add our configuration and maintenance calls in the standard functions where android expects them to be.

`onCreate()` : This is the first thing that happens when an Android App is started by the system, it only happens here and does not keep running whilst your App runs, it is just for initialization.

`onResume()` **:** This is the last initialization phase that you know will always get called when you return to an App that was launched previously and you are returning to.

`onPause()` **:** The is the first thing that is called when an App is about to be suspended, it is important to protect your system state here if you need to preserve it.

`onDestroy()` **:** The last thing to be called before your App is totally exited, any final clean-up happens here.

As is usual in object oriented programming the classes inherit the special properties of the “super class”, in this case the Activity class offers the App developer access to these to add their own setup and clean-up functions and we use them now to help with the controller’s proper “housekeeping”.

3.15 Q: MY GAME RUNS BUT CANNOT BE CONTROLLED. WHAT IS WRONG?

3.15.1 THE CONTROLLER PIVOT APP IS NOT INSTALLED.

The Gamepad controller uses a service to communicate between your game and the actual controller.

For end-users this service is installed by the Launcher application. But during development it is possible that the service is not running on the target device.

To install the service, copy `moga.pivot.sdk.apk` from the `redist` folder of this SDK to your phone\tablet, and use your phones’ file manager to execute the program which will install the service.

3.15.2 I AM USING PRO GUARD

If your Pro Guard file is aggressive this could have an adverse effect on the controller library code. There have been previous issues reported to us that adding The following ProGuard directive may help:

```
-keep public class com.bda.controller.* {public *;}
```

3.16 Q: THERE IS A LAUNCHER AVAILABLE FOR HANDLING THE BLUETOOTH AND PAIRING OF A GAMEPAD FOR ME. HOW DO I ADD MY APPLICATION TO IT, DURING TESTING, SO I CAN LAUNCH MY APP FROM INSIDE?

A: You need an .xml file called “**gameconfig.xml**” copied into the **“/sdcard/gamepad/**” path of your Android device.

Here is the file we used to test the “Lunar Lander” App when we added the controller code. *Note: This approach is to only work with the developer version of the Launcher App.*

```
<android_games>
               <android
                      name = "Lunar Lander"
                      icon = "game_icon_need4speed.png"
                      description = "Test App"
                      package_name = "com.example.android.lunarlander"
                      class_name ="com.example.android.lunarlander.LunarLander"
                      where_to_buy="http://google.com"
                      price="$99.99"
                      rating="5"
                      genre="simulation"
                      is_popular = "true"
                      featured = "true"
               />

               <!-- Multiple Android games -->
</android_games>
```

i.e.

(i) Make an empty file in your notepad application or source code editor and save it out as "**gameconfig.xml**" with the text above inserted.

(ii) Copy that file into **/sdcard/gamepad/** on your Android device and then run the Launcher App.

If you want to test Apps other than Lunar Lander:

(iii) Replace the **name**, **package_name** and **class_name** with your own application.

(iv) You can add extra <android> entries inside the XML file if you want to test more Apps at once.

3.17 Q: WHY DOES MY PHONE GO TO SLEEP WHEN PLAYING WITH THE GAMEPAD CONTROLLER?

A: Previously your game relied on the user pressing the touch screen to keep the phone awake. When a user plays the game using Game Pad, the screen is never touched, so Android puts the phone to sleep, initially dimming the LCD backlight, before going in to full sleep mode.

You need to add code to request a `PowerManager WakeLock` from Android to prevent the phone sleeping while when using the Game Pad.

3.18 Q: HOW CAN I DETECT THE CONTROLLER MODEL?

A: This involves calling `Controller.getState()` using the `Controller.STATE_CURRENT_PRODUCT_VERSION` argument. If the return value is `Controller.ACTION_VERSION_MOGA`, then the connected game controller is a MOGA Controller, if the return value is `Controller.ACTION_VERSION_MOGAPRO`, then the connected game controller is a MOGA Pro Controller.

3.19 Q: HOW CAN I SUPPORT MULTIPLE CONTROLLER MODELS?

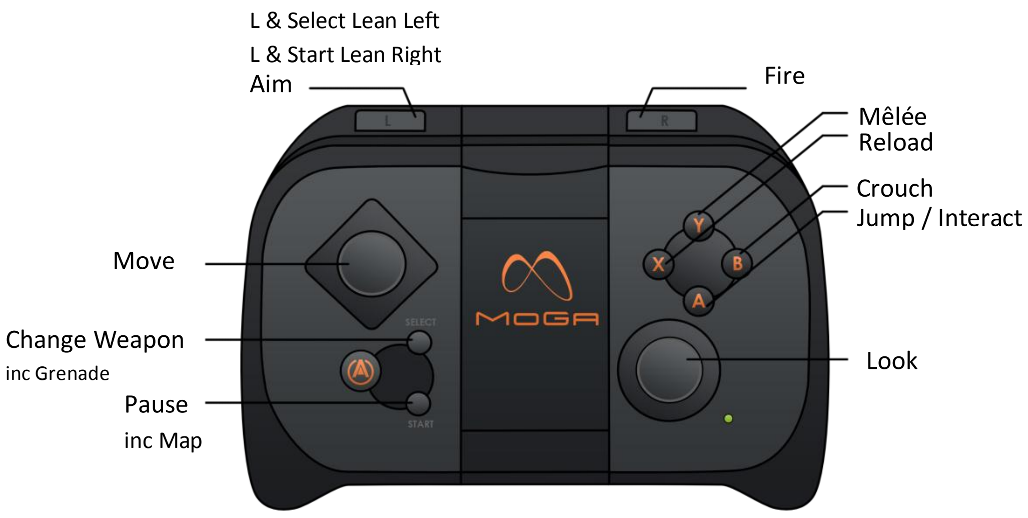
A: The Controller key mapping is done at the developer's discretion, although we recommend you support both the MOGA and MOGA Pro Controllers in order to maximize your userbase.

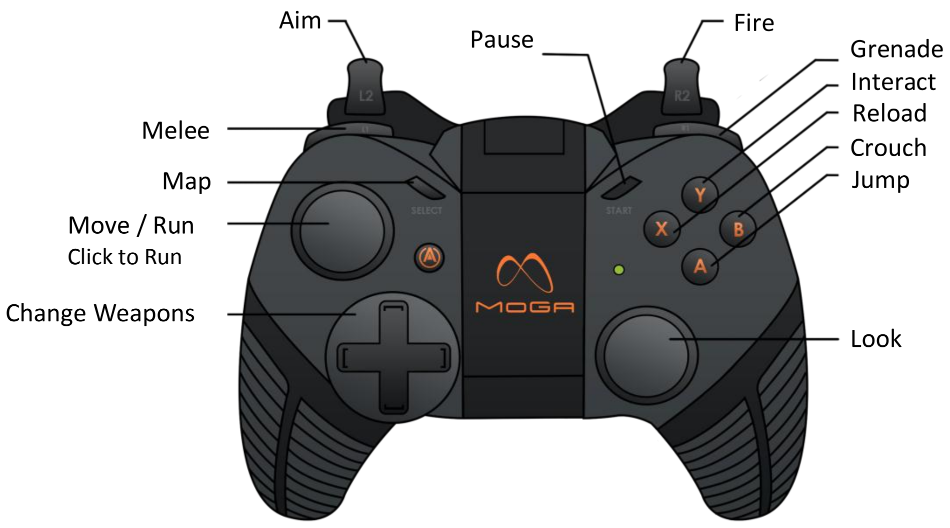
In order to support the entire range of MOGA Controllers you should consider how to map your controls in accordance with the number of buttons available whilst retaining a user friendly controller scheme.

The MOGA PRO features an additional d-Pad, two extra trigger buttons and two clickable nubs, over the MOGA Controller.

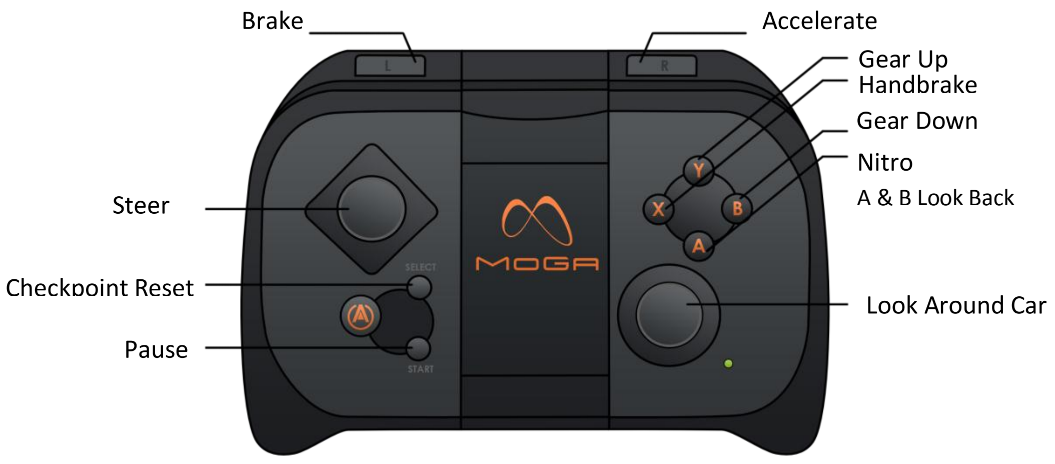
Below are some examples showing how button mapping can support both configurations across popular genres.

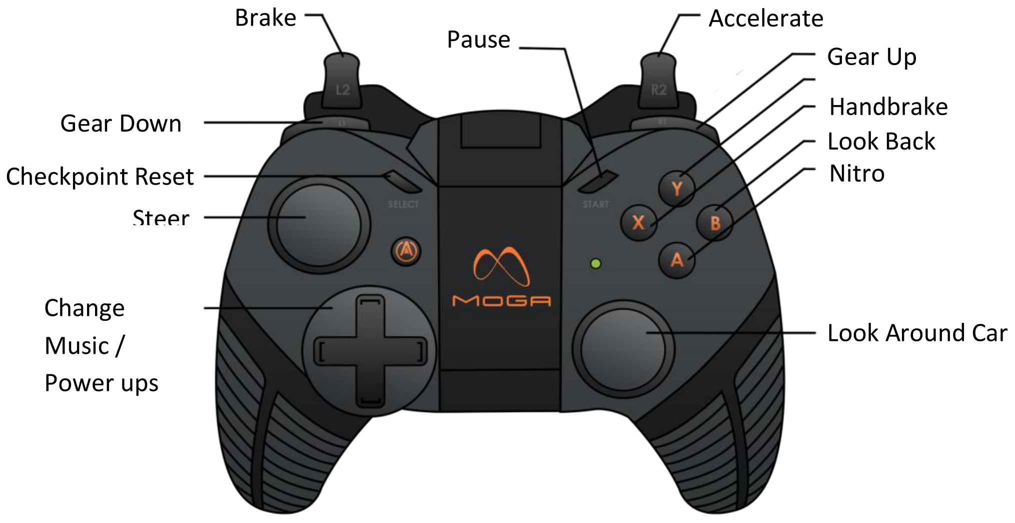
First Person Shooter





Racing


Brake
Accelerate
Gear Up
Handbrake
Gear Down
Nitro
A & B Look Back
Steer
Checkpoint Reset
Pause
Look Around Car
L
R
Y
X
B
A
SELECT
START
MOGA


Brake
Pause
Accelerate
Gear Up
Handbrake
Look Back
Nitro
Gear Down
Checkpoint Reset
Steer
Change Music / Power ups
Look Around Car
L2
R2
Y
X
B
A
SELECT
START
MOGA