# Pong Application

The completed pong central Android application can be installed on an Android phone using the Pong\_Central.apk file. The App Inventor source project can be uploaded to App Inventor using Pong\_Central.aia.

If you want to try creating the app yourself, the following may help:

# MIT App Inventor Getting Started

1. Visit <https://appinventor.mit.edu/>
2. Click on the "[Get Started](http://appinventor.mit.edu/explore/get-started)" button
   1. Click on "[Setup Instructions](http://appinventor.mit.edu/explore/ai2/setup)" and follow option 1 to install "MIT AI2 Companion" on your Android or iOS device.
   2. Go back to "Get Started" and click on "[Designer and Blocks Editor Overview](http://appinventor.mit.edu/explore/designer-blocks)"
      1. Designer is used to design the user interface
      2. Blocks Editor is used to implement the functionality
3. Go back to "Get Started" and click on "[Beginner Tutorials](http://appinventor.mit.edu/explore/ai2/beginner-videos)"
   1. Try out one or more tutorials:
      1. Click on "[Create Apps!](http://ai2.appinventor.mit.edu/)" and sign in to a Google account to launch the tool.
      2. Follow the instructions under Get Started > Setup Instructions > Option 1 [Instructions](http://appinventor.mit.edu/explore/ai2/setup-device-wifi) to connect your phone to App Inventor.
      3. Follow the tutorial instructions.
4. To learn how to package the app into an executable .apk file, go back to "Get Started" and click on "[Packaging and Sharing Apps](http://appinventor.mit.edu/explore/ai2/share)"

# App Inventor Pong Game

There are several tutorials available for creating a self-contained pong game using App Inventor. Starting with one of those and adding the Bluetooth capability is one way to create the Central side of our pong game.

Here is the one that I used:

<https://appinventor.mit.edu/explore/teach/pong>

It is a bit out of date so some of the instructions are not exactly right, but it is easy enough to follow.

I skipped the sections that keep track of the score and those that implemented the Start and Reset buttons since I didn’t need them. I just created the simple pong functionality with a canvas and then added Bluetooth® to it.

I set the canvas dimensions to be the same as the TFT screen dimensions (320x240) to keep things simple even though it looks a bit strange.

A black and white rectangle with black lines

Description automatically generated

# Using App Inventor with Bluetooth® LE

Once I had the basic pong game working on my Android phone, I added Bluetooth® functionality and tied that into what should happen when the ball hits the top wall. Specifically:

When the ball hits the top wall, disable the ball and hide it on the Android screen, then send a message to the peripheral with the ball information.

Once that’s done, just wait for a notification for when the ball comes back. When it does, enable it, make it visible again, move it to the correct starting screen position and set its heading and speed based on the Bluetooth® message.

To learn how to install the Bluetooth® LE extension and create a basic Bluetooth® LE app that will scan/connect, visit:

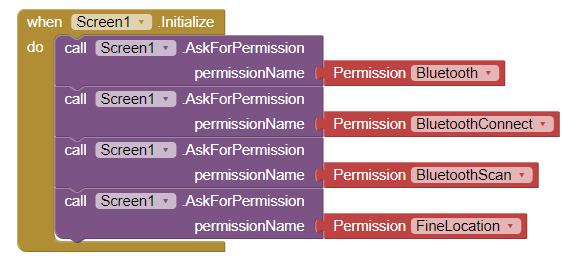
<http://iot.appinventor.mit.edu/assets/tutorials/MIT_App_Inventor_Basic_Connection.pdf>

**You must use the latest version of the Bluetooth® LE extension. The link shown in the PDF file is not the latest.** You should use to go this site and download the AIX file for Bluetooth® LE:

<http://mit-cml.github.io/extensions/>

A few additional instructions:

1. When a connection is made, hide the list of devices and un-hide the canvas to show the pong game playing field.
2. The PSoC application moves the ball by specifying X speed and Y speed while the ball object in App Inventor uses heading and speed. Therefore, you will need to convert cartesian coordinates to polar coordinates and vice versa when the ball jumps between the Android app and the PSoC peripheral.
3. Scan and Connect
   1. **You must add an additional block for scanning and connecting to work** – BLE on Android 10 and above requires Bluetooth® and fine location permissions so you must request them:



1. Writing a Characteristic

Once you are connected, here is an example of how you might write to a characteristic to send the ball information to the PSoC Peripheral when the ball reaches the top edge of the screen.

Note: App Inventor uses UUIDs to specify the service and characteristic. The UUIDs can be set using global variables in App Inventor. You can copy/paste the values from the Bluetooth® configurator in ModusToolbox™ for your peripheral into the global variables that you set up in App Inventor. Your values will not be the same as the ones shown here.

A close-up of a number

Description automatically generated

A screenshot of a computer

Description automatically generated

Note: This example uses a call BluetoothLE1.WriteShorts block 1to write a list of 16-bit signed values since that is what the PSoC Pong Peripheral uses. There are other functions available such as WriteBytes, WriteIntegers and WriteFloats for other data types.

1. Reading a Characteristic

Note: manually reading a characteristic isn’t needed for the Pong Central but is included for completeness.

To read a characteristic, you need to do two things:

* 1. Initiate the read with an appropriate "call" block. In our case, that would be:

call BluetoothLE1.ReadShorts

* 1. Add the corresponding "when" block which will be executed when values are received:

when BluetoothLE1.ShortsReceived

You should use "if" constructs to check the UUID to make sure the value you received is the one you expected. You can even have multiple "if" constructs to handle multiple read values in a single block.

1. Enabling Notifications

To use notifications, you need to do two things:

* 1. Register for notifications with an appropriate "call" block such as:

call BluetoothLE.RegisterForShorts

This can be done as soon as the connection is established.

A puzzle pieces with text

Description automatically generated

* 1. Add a corresponding "when" block which will be executed when values are received. This is identical to what is done for a manual read and in fact the same block will be executed either for data from an explicit read or from a notification:

when BluetoothLE1.ShortsReceived

A screenshot of a computer

Description automatically generated

Note that the UUID that comes back from the ShortsRecieved block is in lower case, so it must be converted using upcase if your global variable is in upper case (which it will be if you copy/pasted the value from the Bluetooth® configurator.

Note: In addition to what is shown above, the ShortsRecieved block will need to set the correct ball location along the top edge and calculate the correct ball heading a speed based on the Bluetooth® message you received.

Here’s what my finished game looks like when scanning for devices and playing:

A screenshot of a black screen

Description automatically generated A black and white rectangle with a black dot

Description automatically generated