Look out, Duke! Part 2: Control a Java game with Raspberry Pi and a joystick

It takes only a little hardware, and only a little software, to attach a single-board computer to a Java game.

by Frank Delporte

July 23, 2021

Controlling an arcade game with a keyboard is fine—but for the real arcade experience, you'll want a joystick. In this article, I'll show you how to extend the simple game built in “Look out, Duke! Part 1: How to build a Java game with JavaFX and the FXGL library” with a real joystick to control the game by adding the Pi4J object-oriented I/O API to the mix, along with some inexpensive hardware and other components. For this project, I'll use a Raspberry Pi 4, a very inexpensive single-board computer with a lot of power.

The shoot-em-up game uses JavaFX and the FXGL library. With four classes and not too much coding, you can create a fully functional little game to illustrate the possibilities of this great open source project.

Here is a small video of the game running on the Raspberry Pi. The movement of the game entities needs some further fine-tuning, but I'll leave that to you, because the performance will depend on the type of board you are running the game on. And let's agree that I'm a better programmer than game player, as you can see based on the scoreboard.

Oracle Java Magazine - FXGL - Part 2
from Frank Delporte
The finished project is available on GitHub. The main branch contains the code from part 1 of this article; you can run that code on any PC.

The full code to run on the Raspberry Pi is in the pi4j branch. Because this is Java, this version will run on a PC, but it will throw some errors where the Pi4J methods are called.

There are a few changes between the code of the previous post and this branch, as follows:

- In the assets directory, two helper scripts make it easy to start the game on the Raspberry Pi.
- In pom.xml, you’ll find extra dependencies and build steps.
- Pi4JFactory.java is the code to link the general-purpose input/output (GPIO) interface to the game controls.
- There are a few other small changes in the previous code to adapt the animations to the performance of the Raspberry Pi.

These changes are described in this article.

Introducing the Pi4J I/O API

The Pi4J project, launched by Robert Savage in 2012, unites Java programming with electronics on the Raspberry Pi. In the first half of 2021, the project was updated twice.

- Version 1.3 (02/2021) added support for the newest Raspberry Pi boards on Java 8.
- Version 1.4 (03/2021) targeted Java 11 with initial rework to simplify the code.

Currently the Pi4J team is working on a major V.2 release, which will refactor the code to bring modules and an improved architecture to make the project easier to maintain, test, and release. For this article, I am using an early build of this update.

Important caveat: The Pi4J team emphasizes that V.2 is still experimental: “While many parts of the project are working, there are still a number of areas that require further development and certain APIs are subject to change without notice. A significant portion of the code is presently undocumented and hardware integration testing is incomplete. It is not recommended to use Pi4J V.2 in any production workload at this time.”

By adding the Pi4J dependency to your code, you can control electronic components connected to the GPIO pins of the Raspberry Pi computer (see Figure 1). Once they are connected, they become objects that Java can control and program, the same as you can do with other Java objects. This way, you don’t need to be fully aware of all the behind-the-scenes activity that relates to the hardware communication; you can work at a higher level of abstraction.
In case you are wondering, Pi4J controls the GPIO pins with Java Native Interface (JNI) and the pigpio native library.

The hardware components

To control the game with a physical joystick and button, extend the Raspberry Pi with a Picade Hardware Attached on Top (HAT) circuit board (shown in Figure 2) and an arcade parts kit (shown in Figure 3). The HAT board is optional, but having one greatly simplifies attaching the components because it clearly shows the connections to be used.
Figure 3. The arcade parts kit

On pinout.xyz, you can find an interactive map of the GPIO BCM (the Broadcom SOC channel) numbers to use for each HAT connection. For example, “joystick up” is linked to physical pin 32, which is BCM 12. You can learn more about GPIO wiring, including the differences between BCM numbers and physical board pins, here. See Figure 4 for the pinout. Figure 5 shows how everything looks when it’s connected. It’s not pretty, but it works.

<table>
<thead>
<tr>
<th>3v3 Power</th>
<th>1</th>
<th>2</th>
<th>5v Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO 2</td>
<td></td>
<td></td>
<td>5v Power</td>
</tr>
<tr>
<td>GPIO 3</td>
<td></td>
<td>6</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO 4</td>
<td></td>
<td>5</td>
<td>GPIO 14 (UART TX)</td>
</tr>
<tr>
<td>Ground</td>
<td></td>
<td>10</td>
<td>GPIO 15 (UART RX)</td>
</tr>
<tr>
<td>GPIO 17</td>
<td>11</td>
<td>12</td>
<td>GPIO 18 (I2S Clock)</td>
</tr>
<tr>
<td>GPIO 27</td>
<td>13</td>
<td>14</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO 22</td>
<td>15</td>
<td>16</td>
<td>GPIO 23 (Coin)</td>
</tr>
<tr>
<td>3v3 Power</td>
<td>17</td>
<td>18</td>
<td>GPIO 24 (Start)</td>
</tr>
<tr>
<td>GPIO 10</td>
<td>19</td>
<td>20</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO 9</td>
<td>21</td>
<td>22</td>
<td>GPIO 25 (Button 4)</td>
</tr>
<tr>
<td>GPIO 11</td>
<td>23</td>
<td>24</td>
<td>GPIO 8 (Button 3)</td>
</tr>
<tr>
<td>Ground</td>
<td>25</td>
<td>26</td>
<td>GPIO 7 (SPI CE1)</td>
</tr>
<tr>
<td>GPIO 0</td>
<td>27</td>
<td>28</td>
<td>GPIO 1 (EEPROM SCL)</td>
</tr>
<tr>
<td>GPIO 5</td>
<td>29</td>
<td>30</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO 6</td>
<td>31</td>
<td>32</td>
<td>GPIO 12 (Up)</td>
</tr>
<tr>
<td>GPIO 13</td>
<td>33</td>
<td>34</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO 19</td>
<td>35</td>
<td>36</td>
<td>GPIO 16 (Right)</td>
</tr>
<tr>
<td>GPIO 26</td>
<td>37</td>
<td>38</td>
<td>GPIO 20 (Left)</td>
</tr>
<tr>
<td>Ground</td>
<td>39</td>
<td>40</td>
<td>GPIO 21 (I2S Data)</td>
</tr>
</tbody>
</table>

Figure 4. The pinout diagram
The extended pom.xml

For the PC-based game, you only needed to specify the FXGL library dependency in pom.xml. You need to add a few more dependencies to add the logging framework and the modules of the Pi4J library, as follows:

```xml
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>${slf4j.version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>${slf4j.version}</version>
</dependency>
<dependency>
  <groupId>com.pi4j</groupId>
  <artifactId>pi4j-core</artifactId>
  <version>${pi4j.version}</version>
</dependency>
<dependency>
  <groupId>com.pi4j</groupId>
  <artifactId>pi4j-plugin-raspberrypi</artifactId>
  <version>${pi4j.version}</version>
</dependency>
<dependency>
  <groupId>com.pi4j</groupId>
  <artifactId>pi4j-plugin-pigpio</artifactId>
  <version>${pi4j.version}</version>
</dependency>
```

A few additional plugins are used to build a modular Java project to run on the Raspberry Pi. One important configuration is needed for the Maven dependency plugin, which will be discussed later. You will add dependencies for all the modules (such as Pi4J, FXGL, and SLF4J) to be copied to the target/distribution directory, except the JavaFX modules. Why? Because this project will use JavaFX modules provided by Gluon that are optimized for the Raspberry Pi.
Add this line to the pom.xml file:

```xml
<excludeGroupIds>org.openjfx</excludeGroupIds>
```

## Coding the joystick and push-button switch

As many as six buttons can be connected to the Picade HAT board. The joystick uses four of them; each direction is signaled when the stick’s movement presses one of the controller’s buttons. A fifth HAT button connection will be used for a physical push-button switch, which is used to tell Duke to shoot a bullet.

Button handling is implemented in the `Pi4JFactory.java` class. First, define the pin numbers by which the buttons are connected to the Picade HAT board. You can use `Console` to help; it’s a log output helper provided by Pi4J. `Context` holds all the GPIO-related objects.

```java
public class Pi4JFactory {
    // BCM numbers of the connected components
    // Full list can be found on https://pinout.xy
    private static final int PIN_JOYSTICK_UP = 12;
    private static final int PIN_JOYSTICK_DOWN = 6;
    private static final int PIN_JOYSTICK_LEFT = 2;
    private static final int PIN_JOYSTICK_RIGHT = 14;
    private static final int PIN_BUTTON_1 = 5;

    private final Console console;
    private Context pi4j;

    public Pi4JFactory() {
        console = new Console();
        try {
            pi4j = Pi4J.newAutoContext();
            initInputGpios();
        } catch (Exception ex) {
            console.println("Error while initializ
        }
    }

    public Console getConsole() {
        return console;
    }
}
```

For each input button, you need to define the input pin and the event listener. By passing all these parameters to one `Pi4JFactory` method, you get clean, short code. The link with FXGL is made in the `DigitalStateChangeListener` of the Pi4J `DigitalInput`. By calling the FXGL executor, you can trigger a key press, just as you would by playing the game on a PC keyboard. The benefit is that you don’t need to change the game code at all. You simply extend the code to add the extra hardware functionality. You can even avoid thread issues, thanks to the FXGL `startAsyncFX` executor method.

```java
/**
 * Initialize Pi4J and the connected components
 */
```
Changes to the existing game code

Because the performance of the Raspberry Pi is lower than that of a developer PC, you should make two minor adjustments to the code provided in part 1 of this article. The `onUpdate` methods of both Duke and the clouds need to take the frame rate into account for a smooth animation.

In `PlayerComponent.java`, change the following code

```java
private static final double ROTATION_CHANGE = 0.01;

@Override
public void onUpdate(double tpf) {
    entity.translate(direction.multiply(tpf * 30));
    checkForBounds();
}
```

to

```java
private static final double ROTATION_CHANGE = 0.5;

@Override
public void onUpdate(double tpf) {
    entity.translate(direction.multiply(tpf * 30));
    checkForBounds();
}
```
Those new values turned out to be perfect for my Raspberry Pi 4 board with 2 GB of RAM, but you may need to fine-tune them for your specific board. Play with the values and see what different numbers do to the game.

The module-info.java file

Because a Pi4J project builds as a modular project, you need to provide a module-info.java file. Use open module to enable FXGL to access the images in the resources directory. See this information from the FXGL wiki for a deeper description.

```java
open module be.webtechie.fxgl {
    requires com.pi4j;
    requires com.pi4j.plugin.pigpio;
    requires org.slf4j;
    requires org.slf4j.simple;
    requires com.almasb.fxgl.all;

    uses com.pi4j.extension.Extension;
    uses com.pi4j.provider.Provider;

    exports be.webtechie.fxgl to com.almasb.fxgl;
}
```

Running the game on a Raspberry Pi board

Install Raspberry Pi OS. In the installer, select Raspberry Pi OS Full (32-bit), because that Linux port includes OpenJDK 11. See Figure 6.

![Raspberry Pi OS installer](image)

**Figure 6.** Choose the full Linux option in the Raspberry Pi OS installer.

To benefit from the improvements done by Gluon and the community to provide the best performance of JavaFX on embedded systems, you should install JavaFX 17. (As of early July 2021, it’s the early access version.)

Open a terminal window and use the following commands to download the package, unzip it, and move it to the correct directory:

```
$ wget -O openjfx.zip https://gluonhq.com/download
$ unzip openjfx.zip
```
Install Maven. Maven lets you build the game as modules, which allows you to include the Pi4J dependencies but not the JavaFX ones. That's because the JavaFX dependencies are now in the /opt/javafx-sdk-17/ directory, thanks to the download you performed in the previous step.

```
$ sudo apt install maven
$ mvn -v
```

Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 11.0.11, vendor: Raspbian, runtime: Java version: 11.0.11, vendor: Raspbian, runtime:
Default locale: en_GB, platform encoding: UTF-8 Default locale: en_GB, platform encoding: UTF-8
OS name: "linux", version: "5.10.17-v7l+", arch: OS name: "linux", version: "5.10.17-v7l+", arch:

Build and run the game. Download the source code, build the project, and run it using five lines in the terminal, as follows:

```
$ git clone -b pi4j https://github.com/FDelporte/JavaMagazineFXGL
$ cd target/distribution
$ sudo bash run.sh
```

Run the game in kiosk mode. Next to the run.sh script, I've provided an additional script, run-kiosk.sh, which starts the game in kiosk mode. This disables the default window manager and gives all resources to the JavaFX application in Direct Rendering Mode (DRM) to directly use the underlying (hardware) framework.

To learn more about this approach see my article "JavaFX running in kiosk mode on the Raspberry Pi."

Conclusion

In my previous article, you saw how to easily create an interactive game with very little code. This time, you made the game experience more exciting with a hardware game controller and cheap components connected to a Raspberry Pi single-board computer, and you also saw how to use Java (and JavaFX) in an embedded system.

Dig deeper

- Look out, Duke! Part 1: How to build a Java game with JavaFX and the FXGL library
- How to build a Raspberry Pi webcam, plus the surprising links between Pi and Oracle Cloud Infrastructure
- Getting started with JavaFX on Raspberry Pi
- JavaFX running in kiosk mode on the Raspberry Pi
- Oracle Java ME Embedded getting started guide for the reference platform (Raspberry Pi)
- Set up a Raspberry Pi for connecting to Oracle IoT Cloud Service
- Using a Raspberry Pi to deploy JavaFX applications
Frank Delporte

Frank Delporte is the author of *Getting Started with Java on Raspberry Pi* and works at toadi.com. He is also lead coach of CoderDojo Belgium in Ieper.