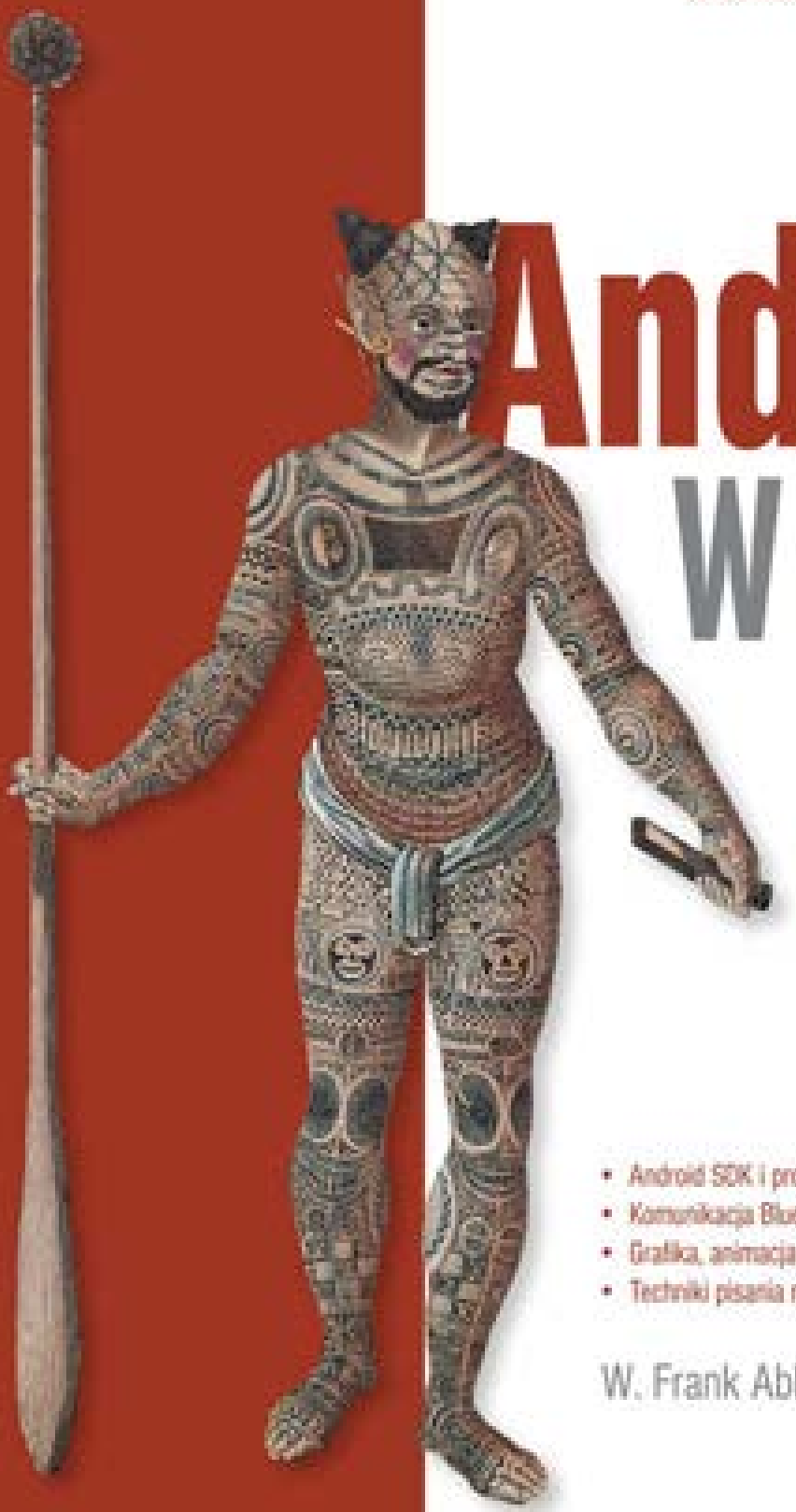
Rozbudowany podręcznik
tworzenia niesamowitych aplikacji
dla systemu Android!



Android W AKCJI

WYDANIE II

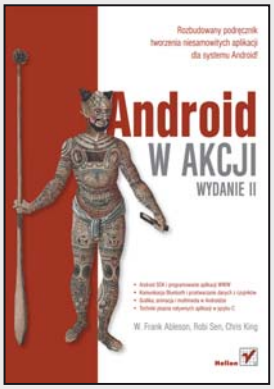
- Android SDK i programowanie aplikacji WWW
- Komunikacja Bluetooth i przetwarzanie danych z czujników
- Grafika, animacja i multimedia w Androidzie
- Techniki pisania natywnych aplikacji w języku C

W. Frank Ableson, Robi Sen, Chris King

Helion

Android w akcji. Wydanie II

Autorzy: W. Frank Ableson, Robi Sen, Chris King
Tłumaczenie: Paweł Gonera
ISBN: 978-83-246-3380-7
Tytuł oryginału: Android in Action
Format: 168×237, stron: 624



Rozbudowany podręcznik tworzenia niesamowitych aplikacji dla systemu Android!

- Android SDK i programowanie aplikacji WWW
- Komunikacja Bluetooth i przetwarzanie danych z czujników
- Grafika, animacja i multimedia w Androidzie
- Techniki pisania natywnych aplikacji w języku C

Skoro zwróciłeś uwagę właśnie na tę książkę, zapewne dobrze wiesz, czym jest Android i co potrafi – teraz przyszła pora, abyś sprawdził go także w akcji! Oto doskonała książka dla wszystkich programistów, którym marzy się tworzenie własnych aplikacji dla robiącego oszałamiającą karierę systemu. Choć ta książka nie jest przeznaczona dla początkujących, zawiera wszystkie informacje potrzebne osobom, dla których Android jest całkowicie nowym środowiskiem. Można w niej znaleźć instrukcje niezbędne do szybkiego zorientowania się w architekturze tej platformy oraz sposobie jej działania, co pozwoli sprawnie rozpocząć pracę w tym środowisku. Pozostałe rozdziały to już czysta frajda programowania!

Od czego zaczniesz zabawę z Androidem? Dowiesz się, jak budować aplikacje dla tego systemu od najmniejszych cegiełek aż po ekrany, dodawać funkcje telefoniczne i wykorzystywać bibliotekę OpenGL ES do tworzenia złożonej grafiki 2D oraz 3D. Następnie poznasz zasady tworzenia większych aplikacji oraz techniki pisania aplikacji w języku C, także z użyciem Android Native Development Kit. Opanujesz potężne narzędzie Android SDK oraz budowanie aplikacji dla WebKit z użyciem HTML 5, a nawet nauczysz się rozszerzać lub zastępować wbudowane funkcje Androida na podstawie użytecznych i intrygujących przykładów.

- Wprowadzenie do systemu Android
- Środowisko programowania
- Komponenty interfejsu użytkownika, w tym View i Layout
- Metody przechowywania i odczytywania lokalnych danych
- Sieci oraz usługi sieciowe, Bluetooth, sensory i widżety aplikacji
- Przegląd podstawowych zagadnień związanych z telefonią
- Powiadomienia i alarmy
- Grafika i animacja w Androidzie
- Korzystanie z funkcji multimedialnych Androida
- Usługi oparte na lokalizacji
- Integracja kontaktów z platformy społecznościowej
- Aplikacja wspomagająca serwisantów
- Budowanie aplikacji Android w języku C
- Tworzenie stron WWW dla systemu Android
- Strategie lokalizowania aplikacji
- Android Native Development Kit oraz korzystanie z SDK oraz AVD Manager

Wkrocz wreszcie do akcji i zacznij tworzyć własne aplikacje dla Androida!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

Spis treści

14 Bluetooth i sensory

W tym rozdziale:

- Dołączanie Bluetooth do urządzenia
- Interakcja z obiektem SensorManager
- Budowanie i uruchamianie aplikacji SenseBot

Większość materiału przedstawionego w tej książce dotyczy wykorzystania różnych możliwości Android SDK. Teraz jednak pokażemy, jak skorzystać z możliwości sprzętowych urządzeń Android. Zapoznamy się tu z podłączaniem urządzenia Android do zdalnych urządzeń za pomocą połączenia bezprzewodowego Bluetooth oraz odczytem i interpretacją wartości ze sprzętowego czujnika orientacji urządzenia. W rozdziale tym te dwa tematy związane ze sprzętem połączyliśmy w jednym przykładowym programie, który pozwala sterować robotem zbudowanym przy użyciu popularnego zestawu LEGO Mindstorms NXT. Robot Mindstorm NXT obsługuje protokół komunikacyjny znany pod nazwą „Direct Commands[1]", co pozwala na sterowanie nim z użyciem urządzenia zdalnego. Jest to rozdział, w którym konieczne jest użycie fizycznego urządzenia z zainstalowanym systemem Android w wersji 2 lub nowszej — sam symulator nie jest wystarczający do sprawdzania działania Bluetooth oraz sensorów.

Kod dołączony do tego rozdziału tworzy aplikację o nazwie SenseBot. Jest to średnio skomplikowany przykład użycia systemu Android do manipulowania zewnętrznymi obiektami. Czujniki orientacji urządzenia Android pozwalają użytkownikowi „jechać" robotem, przechylając telefon w określonym kierunku, w sposób podobny do używania Nintendo Wii lub innego zaawansowanego systemu gier. Przechylamy telefon do przodu, a robot jedzie do przodu. Przechylamy telefon do tyłu, a robot zmienia kierunek. Przechylanie w lewo lub w prawo powoduje skręt robota w odpowiednim kierunku. Po zinterpretowaniu wartości każdego czujnika ruchu aplikacja SenseBot wysyła polecenia do robota poprzez Bluetooth, powodując odpowiednie fizyczne działanie. LEGO NXT ma wbudowany zestaw poleceń pozwalających na operacje niskiego poziomu, takie jak bezpośrednie sterowanie silnikami. Ruchy urządzenia Android są interpretowane, konwertowane na polecenia i przesyłane za pomocą Bluetooth do robota.

Oprócz podstaw komunikacji Bluetooth oraz zarządzania sensorami kod demonstruje użycie dynamicznie tworzonych obiektów `BroadcastReceiver`, które obsługują zdarzenia związane z połączeniem Bluetooth.

Temat komunikacji Bluetooth jest znacznie szerszy i nie mamy możliwości opisać go w całości w jednym rozdziale. Na platformie Android dostępnych jest co najmniej sześć różnych sensorów, a w tym rozdziale demonstrujemy użycie tylko jednego. Jeżeli szukasz dokładnego opisu tych dwóch tematów, zachęcamy do odszukania dokumentacji w sieci lub być może innej książki na ten temat. Celem tego rozdziału jest przedstawienie funkcji Bluetooth oraz sensorów na platformie Android w kontekście działającej (i zabawnej) aplikacji. Jeżeli masz dostęp do robota LEGO Mindstorms NXT i zbudujesz tę aplikację, obiecujemy, że na długi czas nie będziesz się mógł oderwać od „jeżdżenia" robotem za pomocą telefonu. Jedna z wersji tej aplikacji jest dostępna do pobrania z Android Market.

[1] Więcej informacji na temat Direct Commands dla Lego Mindstorm można znaleźć pod adresem *http://mindstorms.lego.com/en-us/support/files/default.aspx*.

14.1. Przegląd możliwości Bluetooth w systemie Android

Pierwszym urządzeniem, jakie przychodzi na myśl, gdy wspomina się *Bluetooth*, jest bezprzewodowy zestaw słuchawkowy. W wielu krajach te cudeńka technologii bezprzewodowej są wymagane przez prawo w przypadku korzystania z telefonu w czasie prowadzenia samochodu. W rzeczywistości zestawy słuchawkowe to tylko jedno z wielu zastosowań technologii Bluetooth.

Bluetooth to technologia komunikacji bezprzewodowej podobna do Wi-Fi, ale ograniczona do komunikacji o niewielkim zasięgu, około 10 metrów. Oprócz zastosowania w bezprzewodowych słuchawkach i mikrofonie telefonu komórkowego Bluetooth pozwala również na realizację połączeń sieciowych między urządzeniami, wymianę obiektów, zastępowanie kabli oraz realizację zaawansowanych funkcji audiowizualnych.

Podobnie jak każdy inny standardowy protokół, tak i Bluetooth posiada własny „stos" protokołów, z których każdy implementuje osobne możliwości i funkcje protokołu. W rozdziale tym nie będziemy tracić czasu na prezentowanie tych warstw, ponieważ stos Bluetooth jest opisany w innych miejscach. Zamiast tego w rozdziale tym pokażemy możliwości nawiązania połączenia danych pomiędzy dwoma urządzeniami. Będziemy tu korzystać z „profilu" Bluetooth o nazwie RFCOMM[2], będącego profilem zastępowania kabla.

W tym podrozdziale pokażemy, jak nawiązać połączenie pomiędzy Androidem a zdalnym urządzeniem, korzystając z pakietu `android.bluetooth`. Ponieważ platforma Android pozwala tylko na połączenia szyfrowane, dwa urządzenia komunikacyjne muszą być wcześniej sparowane, a później mogą łączyć się ze sobą bez konieczności podawania hasła. Aby wiedzieć, czy aplikacja jest połączona z urządzeniem Bluetooth, należy następnie zarejestrować dwa zdarzenia: `ACTION_ACL_CONNECTED` oraz `ACTION_ACL_`
`↪DISCONNECTED`. Dodatkowo aplikacja Android musi posiadać uprawnienie BLUETOOTH zdefiniowane w pliku *AndroidManifest.xml*. Zaczynamy!

14.1.1. Zastępowanie kabli

Obecnie podłączanie się do internetu w celu sprawdzenia poczty lub przeglądania sieci WWW jest codzienną czynnością wielu użytkowników systemu Android. Przy użyciu naszego telefonu możemy połączyć się z komputerami znajdującymi się po drugiej stronie planety, ale jak komunikować się z czymś, co znajduje się w tym samym pokoju? W niezbyt odległej przeszłości programowaliśmy interfejsy pomiędzy komputerami połączonymi szeregowo kablem korzystającym z interfejsu RS232. Po kilku krótkich latach kable szeregowe RS232, zastąpione przez bardziej zaawansowane kable USB oraz przez profil Bluetooth Serial Port, stały się eksponatem muzealnym.

Tak jak USB może być używane przez wiele różnych aplikacji, tak i protokół bezprzewodowy Bluetooth może być wykorzystywany na wiele sposobów. W tym momencie interesuje nas możliwość użycia Bluetooth do zastąpienia kabla szeregowego za pomocą profilu Serial Port Profile (SPP), nazywanego czasami RFCOMM. RF pochodzi

[2] Więcej informacji na temat RFCOMM można znaleźć na stronie *http://www.bluetooth.com*.

od słów *radio frequency*, czyli częstotliwości radiowych. COMM pochodzi od portu komunikacyjnego, czyli antycznego już połączenia punkt-punkt, wykorzystującego protokół strumieniowy.

14.1.2. Rola podstawowa i podrzędna oraz gniazda

Protokół Bluetooth działa w sposób podobny do innych środowisk komunikacyjnych, w których urządzenie podstawowe inicjuje komunikację z jednym lub większą liczbą urządzeń podrzędnych. Android jest na tyle rozbudowany, że może być zarówno urządzeniem podstawowym, jak i podrzędnym w połączeniu Bluetooth.

Niezależnie od sposobu nawiązania połączenia — jako podstawowe lub drugorzędne urządzenie Bluetooth — aplikacja Android wymienia dane poprzez interfejs gniazd. To prawda — znany już mechanizm sieciowy korzystający z gniazd oraz skojarzonych z nimi strumieni wejściowych i wyjściowych jest również stosowany przy połączeniach Bluetooth. Jeżeli więc przejdziemy przez etap łączenia ze sobą dwóch urządzeń Bluetooth w celu utworzenia sesji komunikacyjnej, możemy mniej przejmować się szczegółami komunikacji i po prostu traktować urządzenie jako aplikację po drugiej stronie gniazda. Jest to podobne do relacji pomiędzy przeglądarką WWW i zdalnym serwerem, który wymienia dane poprzez gniazda TCP.

Aby użyć środowiska Bluetooth w urządzeniu Android, musimy skorzystać z pakietu `android.bluetooth`, który został dodany w wersji 2.0. Choć większość urządzeń Android w wersji niższej niż 2.0 mogła korzystać z zestawów słuchawkowych Bluetooth, to dopiero od wersji 2.0 aplikacje Android mogły wykorzystywać komunikację Bluetooth w sposób przedstawiony w tym rozdziale. W tabeli 14.1 pokazane są główne klasy Java używane w aplikacjach Android korzystających z komunikacji Bluetooth.

Tabela 14.1. Klasy Bluetooth

Klasa	Komentarz
`BluetoothAdapter`	Klasa ta reprezentuje sprzęt lokalnego urządzenia Bluetooth w systemie Android i pozwala na tworzenie interfejsu. Wszystko zaczyna się od `BluetoothAdapter`.
`BluetoothClass`	Klasa `BluetoothClass` zapewnia wygodne metody dostępu do stałych wartości związanych z komunikacją i działaniem Bluetooth.
`BluetoothDevice`	Każde urządzenie zdalne jest reprezentowane jako `BluetoothDevice`.
`BluetoothSocket`	Klasa `BluetoothSocket` jest używana przy wymianie danych. Urządzenie podstawowe inicjuje połączenie z gniazdem w urządzeniu podrzędnym, tworząc wcześniej `BluetoothSocket`. W rozdziale tym przedstawione są przykłady kodu demonstrujące tę technikę.
`BluetoothServerSocket`	Urządzenie podrzędne Bluetooth oczekuje połączeń z urządzeniem nadrzędnym za pomocą `BluetoothServerSocket` w taki sam sposób, jak serwer WWW oczekuje na połączenie przez gniazdo TCP z przeglądarki. Po zestawieniu połączenia tworzony jest obiekt `BluetoothSocket` zapewniający dalszą komunikację.

W rozdziale tym przedstawimy użycie klas `BluetoothAdapter`, `BluetoothDevice` oraz `BluetoothSocket`. W następnym punkcie pokażemy, w jaki sposób urządzenie Android podłącza się do innego urządzenia Bluetooth.

Uwaga

W przedstawionych tu przykładach urządzenie Android jest podstawowym, a kontroler LEGO Mindstorm NXT podrzędnym urządzeniem Bluetooth.

14.1.3. Urządzenia zaufane

Choć specyfikacja Bluetooth pozwala na korzystanie zarówno z połączeń szyfrowanych, jak i nieszyfrowanych, to jednak platforma Android pozwala wyłącznie na połączenia szyfrowane. W zasadzie oznacza to tylko, że dwa komunikujące się z sobą urządzenia muszą być wcześniej *sparowane* lub *związane*. Jest to nieco irytujący krok polegający na poinformowaniu każdego z urządzeń, że drugie jest urządzeniem zaufanym. Pomimo kłopotliwości tego procesu oraz faktu, że niemal wszystkie urządzenia Bluetooth na tej planecie korzystają z kodu zabezpieczeń 0000 lub 1234, jedynym zabezpieczeniem protokołu Bluetooth jest ten właśnie kod.

Urządzenia mogą być sparowane poprzez ich ekrany „ustawień" lub podczas nawiązywania pierwszego połączenia. W tym punkcie przedstawimy sposób parowania urządzenia Android[3] z modułem kontrolera robota LEGO.

Na rysunku 14.1 pokazana jest część ekranu ustawień Bluetooth z mojego telefonu Nexus One z systemem Android 2.2.

Rysunek 14.1. Ekran ustawień Bluetooth

Z tego ekranu możemy uzyskać następujące informacje:

- Bluetooth jest włączony.
- Urządzenie ma nazwę *Nexus One*.
- Urządzenie to nie jest obecnie wykrywalne. Oznacza to, że inne urządzenia Bluetooth nie widzą tego telefonu w czasie „skanowania". Praktycznie rzecz biorąc, oznacza to, że telefon ignoruje pakiety wykrywania. Znajduje się tu również

[3] Aby dowiedzieć się więcej na temat typów używanych układów w sprzęcie Android, warto skorzystać z forum Talk Android, dostępnego pod adresem *http://www.talkandroid.com/android-forums/android-hardware/*.

przycisk pozwalający na zainicjowanie wykrywania znajdujących się w pobliżu urządzeń.

- Aby rozpocząć skanowanie znajdujących się w zasięgu urządzeń Bluetooth, należy kliknąć przycisk *Wyszukaj urządzenia*.
- W telefonie były również wcześniej sparowane inne urządzenia, które nie są obecnie połączone:
  - NXT — robot LEGO.
  - Dwa urządzenia bezprzewodowe Samsung. To nie pomyłka — mamy tu dwa osobne urządzenia sparowane z telefonem. (Autor w ten sposób „rozwiązał" problem częstego gubienia zestawów słuchawkowych przez kupienie kilku sztuk poprzez eBay, stąd kilka sparowanych urządzeń).

Długie kliknięcie jednego z tych urządzeń Bluetooth na liście pozwala otworzyć ekran opcji z dodatkowymi operacjami, zależnymi od urządzenia. Na przykład wybranie jednego z urządzeń Samsung powoduje otwarcie okna pokazanego na rysunku 14.2.

Rysunek 14.2. Opcje sparowanego urządzenia

Aby sparować urządzenie, należy najpierw przeprowadzić skanowanie. Po dodaniu urządzenia do listy możemy je wybrać i zainicjować parowanie. Na rysunku 14.3 pokazany jest kontroler robota LEGO, na którym pokazane jest żądanie PIN w czasie parowania.

Rysunek 14.3. Kontroler LEGO żądający podania kodu PIN

Podany PIN jest porównywany z kodem wprowadzonym w telefonie, co jest pokazane na rysunku 14.4.

Rysunek 14.4. Parowanie z robotem LEGO

W tym momencie nasz telefon i kontroler robota LEGO są sparowane. Od tego momentu będziemy w stanie podłączać się do tego urządzenia bez żądania podania kodu PIN.

14.1.4. Podłączanie się do zdalnego urządzenia

Podłączanie się do sparowanego urządzenia wymaga wykonania dwóch kroków:

- Pobrania listy sparowanych urządzeń za pomocą stosu sprzętowo-programowego Bluetooth.
- Zainicjowania połączenia RFCOMM z urządzeniem docelowym. Na poniższym listingu pokazany jest sposób nawiązywania połączenia RFCOMM lub Serial Port Profile pomiędzy dwoma sparowanymi urządzeniami.

Listing 14.1. Inicjowanie połączenia z urządzeniem Bluetooth

```
public void findRobot(View v)
{
    try
    {
      btInterface = BluetoothAdapter.getDefaultAdapter();  ←1
      pairedDevices = btInterface.getBondedDevices();  ←2
      Iterator<BluetoothDevice> it = pairedDevices.iterator();
      while (it.hasNext())  ←3
      {
        BluetoothDevice bd = it.next();
        if (bd.getName().equalsIgnoreCase(ROBOTNAME)) {  ←4
          connectToRobot(bd);  ←5
          return;
        }
      }
    }
    catch (Exception e)
    {
      Log.e(tag,"Błąd w findRobot() " + e.getMessage());  ←6
    }
}
```

```
private void connectToRobot(BluetoothDevice bd)  ←❺
{
    try
    {
      socket = bd.createRfcommSocketToServiceRecord
(UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"));  ←❼
      socket.connect();  ←❽
    }
    catch (Exception e)
    {
      Log.e(tag,"Błąd komunikacji ze zdalnym urządzeniem [" + e.getMessage() + "]");
    }
}
```

Wszystkie operacje związane z Bluetooth[4] zaczynają się od `BluetoothAdapter` ❶. Przy użyciu referencji do adaptera można uzyskać listę sparowanych urządzeń ❷. Następnie przeglądamy listę ❸, szukając nazwy urządzenia ❹ odpowiadającej nazwie robota. Nazwa ta może być wpisana na stałe, jak jest to zrealizowane w przykładowej aplikacji, wprowadzana przez użytkownika w czasie uruchamiania lub nawet wybierana z bardziej zaawansowanej listy urządzeń. Niezależnie od metody naszym celem jest zidentyfikowanie obiektu `BluetoothDevice`, a następnie zainicjowanie połączenia, jak jest to pokazane w wywołaniu funkcji o nazwie `connectToRobot()` ❺. Dobrą praktyką jest przechwytywanie wyjątków ❻, szczególnie jeżeli korzystamy ze zdalnego urządzenia fizycznego, które może wyjść z zasięgu lub może mieć problemy z zasilaniem. Aby podłączyć się do zdalnego urządzenia za pomocą profilu Serial Port Profile, należy użyć metody `createRfComSocketToServiceRecord()` z klasy `BluetoothDevice`. Ciąg UUID pokazany w kodzie jest identyfikatorem profilu Serial Port Profile ❼. Gdy mamy dostępny obiekt `BluetoothSocket`, można wywołać metodę `connect()` ❽.

W tym momencie mamy odnalezione interesujące nas urządzenie i próbujemy wysłać żądanie połączenia. Czy to się udało? Jak możemy to sprawdzić? Można poczynić założenie na temat stanu urządzenia i ewentualnie czekać na błąd. Nie jest to jednak najlepsze podejście. Musi istnieć inna metoda — faktycznie jest taka metoda, ale wymaga wykorzystania obiektów `Intent`.

14.1.5. Przechwytywanie zdarzeń Bluetooth

Aby sprawdzić, czy aplikacja jest prawidłowo podłączona do `BluetoothDevice`, należy zarejestrować dwa zdarzenia związane z Bluetooth: `ACTION_ACL_CONNECTED` oraz `ACTION_`
`↪ACL_DISCONNECTED`. Gdy wystąpią te dwa zdarzenia, możemy być pewni, że mamy prawidłowe połączenie lub je utraciliśmy. W jaki sposób można użyć tych zdarzeń w połączeniu z utworzonym wcześniej gniazdem? Na poniższym listingu pokazana jest technika polegająca na tworzeniu obiektów `BroadcastReceiver` bezpośrednio w aktywności i rejestrowaniu w nich interesujących nas zdarzeń.

[4] Więcej informacji można znaleźć w dokumentacji Google na temat Bluetooth oraz Android dostępnej pod adresem *http://developer.android.com/guide/topics/wireless/bluetooth.html*.

Listing 14.2. Monitorowanie połączenia Bluetooth

```
private BroadcastReceiver btMonitor = null;                    ❶
private void setupBTMonitor() {                                ❷
    btMonitor = new BroadcastReceiver() {                      ❸
      @Override
      public void onReceive(Context context,Intent intent) {   ❹
        if (intent.getAction().equals(
"android.bluetooth.device.action.ACL_CONNECTED")) {
          handleConnected();                                   ❺
        }
        if (intent.getAction().equals(
"android.bluetooth.device.action.ACL_DISCONNECTED")) {
          handleDisconnected();                                ❻
        }
      }
    };
}
```

Aby monitorować specyficzne rozgłaszane zdarzenia, należy użyć obiektu `BroadcastReceiver`. Zwykle realizuje się to przy użyciu osobnej klasy, ale aplikacja ta wymaga bliższej integracji z interfejsem użytkownika, więc zastosowaliśmy alternatywne podejście. Zwykle obiekty `BroadcastReceiver` są definiowane w pliku *AndroidManifest.xml*, ale w tym przypadku chcemy otrzymywać powiadomienia w określonych przypadkach. W kodzie tym zdefiniowany jest obiekt `BroadcastReceiver` o nazwie `btMonitor` ❶. W metodzie `onCreate()` wywoływana jest metoda `setupBTMonitor()` ❷, w której tworzony jest obiekt `BroadcastReceiver` ❸ zawierający implementację metody `onReceive()` ❹. Za każdym razem, gdy dla tego obiektu `BroadcastReceiver` dostępny jest rozgłaszany obiekt `Intent`, wywoływana jest metoda `onReceive()`. W implementacji tej obsługujemy akcje podłączenia i rozłączenia urządzenia Bluetooth. Gdy urządzenie jest podłączone, wywoływana jest metoda `handleConnected()` ❺. Podobnie gdy zdalne urządzenie zostanie rozłączone, wywoływana jest metoda `handleDisconnected()` ❻, której zadaniem jest wykonanie operacji porządkowych.

Gdy urządzenie jest podłączone, musimy wykonać kilka operacji przygotowawczych, takich jak skonfigurowanie strumienia wejściowego oraz wyjściowego dla gniazda. Na następnym listingu przedstawiona jest skrócona wersja metody `handleConnected()` zawierająca kod pozwalający obsługiwać Bluetooth.

Listing 14.3. Metoda handleConnected

```
private void handleConnected() {
    try {
      is =
socket.getInputStream();                                       ❶
      os = socket.getOutputStream();
      bConnected = true;                                       ❷
      btnConnect.setVisibility(View.GONE);                     ❸
      btnDisconnect.setVisibility(View.VISIBLE);
    } catch (Exception e) {
      is = null;                                               ❹
      os = null;
      disconnectFromRobot(null);                               ❺
    }
}
```

Gdy wywołana zostaje metoda `handleConnection()`, zestawione jest prawidłowe połączenie gniazd Bluetooth, więc musimy tylko utworzyć strumienie wejścia i wyjścia ❶. Po utworzeniu tych strumieni może rozpocząć się komunikacja pomiędzy urządzeniem Android a robotem LEGO. Jak się okaże w dalszej części tego rozdziału, chcemy przetwarzać zdarzenia czujnika wyłącznie przy działającym połączeniu z robotem, więc ustawiamy znacznik ❷ informujący aplikację o stanie połączenia. W procedurze tej przełączamy również widoczność dwóch przycisków ❸ — jednego używanego do łączenia z robotem oraz drugiego pozwalającego zakończyć połączenie. W przypadku wystąpienia błędu w czasie wykonywania tych operacji musimy zamknąć strumienie ❹ i zainicjować żądanie rozłączenia ❺.

Kod pozwalający na rozłączenie gniazda jest bardzo prosty:

```
socket.close();
```

Aby można było wykonać większość operacji Bluetooth w systemie Android, nie należy zapominać o jeszcze jednym ważnym elemencie: uprawnieniach!

14.1.6. Uprawnienia Bluetooth

Korzystanie ze sparowanych urządzeń to nie jedyna czynność, do której wykonania konieczne są uprawnienia. Aby korzystać z Bluetooth API, aplikacja Android musi posiadać uprawnienie `BLUETOOTH` w pliku *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.BLUETOOTH">
</uses-permission>
```

Część kodu odpowiedzialnego za komunikację Bluetooth jest przedstawiona w trzecim podrozdziale, w którym przedstawiamy dokładniej kod aplikacji SenseBot. Zanim zajmiemy się kodowaniem i uruchamianiem aplikacji robota, przyjrzyjmy się klasie `SensorManager`, dzięki której będziemy mogli wykorzystać czujniki Androida do sterowania robotem.

14.2. Interakcja z obiektem SensorManager

Android udostępnia odczyty z czujników poprzez klasę o nazwie `SensorManager`. Klasa `SensorManager` jest podobna do `BluetoothAdapter`, ponieważ wszystkie związane z nią aktywności intensywnie korzystają z `SensorManager`. Klasa `SensorManager` wchodzi w skład pakietu `android.hardware`. W tym podrozdziale pokażemy, jak odczytywać informacje z sensora orientacji, co będzie nam potrzebne w aplikacji SenseBot.

W tabeli 14.2 wymienione są główne klasy skojarzone z `SensorManager`.

Korzystanie z klasy `SensorManager` jest bardzo proste. Na początek należy odczytać referencję do obiektu tej klasy:

```
SensorManager sManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);
```

Tabela 14.2. Klasy związane z sensorami

Klasa	Komentarz
`SensorManager`	Podstawowy interfejs do wszystkich zainstalowanych w urządzeniu sensorów.
`Sensor`	Reprezentuje pojedynczy sensor.
`SensorEvent`	Reprezentuje odczyt z sensora.
`SensorEventListener`	Interfejs ten jest używany do odbierania zdarzeń `SensorEvents` niemal w czasie rzeczywistym.

Po odczytaniu referencji można użyć tej zmiennej w całej aplikacji do realizowania interakcji z samymi sensorami. Na przykład aplikacja SenseBot korzysta z czujnika orientacji. Aby uzyskać referencję do obiektu tego czujnika, należy wywołać z klasy `SensorManager` metodę `getDefaultSensor()`:

```
Sensor orientationSensor =
    sManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

W aplikacji tej korzystamy tylko z czujnika orientacji, ale Android oferuje znacznie więcej. Przyjrzyjmy się liście czujników dostępnych w systemie Android 2.2.

14.2.1. Typy czujników

Android obsługuje typy czujników wymienione w tabeli 14.3.

Tabela 14.3. Czujniki systemu Android

`Sensor.TYPE_ACCELEROMETER`	Mierzy przyspieszenie w trzech wymiarach.
`Sensor.TYPE_GYROSCOPE`	Żyroskop.
`Sensor.TYPE_LIGHT`	Czujnik światła.
`Sensor.TYPE_MAGNETIC_FIELD`	Kompas mierzący pole magnetyczne.
`Sensor.TYPE_ORIENTATION`	Mierzy położenie w trzech wymiarach.
`Sensor.TYPE_PRESSURE`	Mierzy ciśnienie.
`Sensor.TYPE_PROXIMITY`	Mierzy odległość telefonu od innego obiektu, na przykład naszego ucha.
`Sensor.TYPE_TEMPERATURE`	Mierzy temperaturę otoczenia.

Każdy z obiektów czujnika może zawierać kilka przydatnych i interesujących atrybutów, takich jak:

- nazwa czujnika,
- zużycie prądu — w mA,
- rozdzielczość,
- maksymalny zakres,
- dostawca,
- wersja.

Czujnik orientacji w telefonie Nexus One ma następującą charakterystykę:

- Nazwa: *AK8973 Orientation Sensor*
- Pobór prądu: *7,0 mA*
- Rozdzielczość: *1,0 stopień*
- Maksymalny zakres: *360 stopni*

Teraz, gdy wiemy, jak uzyskać dostęp do czujnika poprzez `SensorManager`, zajmijmy się odczytem jego wartości.

14.2.2. Odczyt wartości czujnika

Wartości z czujników odczytujemy z użyciem obiektu implementującego interfejs `SensorEventInterface`. Do metody o nazwie `onSensorChanged()` wysyłane są obiekty `SensorEvent`. Klasa `SensorEvent` zawiera cztery pola wymienione w tabeli 14.4.

Tabela 14.4. Pola klasy SensorEvent

Pole	Komentarz
`accuracy`	Pole typu całkowitego reprezentujące dokładność odczytu szacowaną przez czujnik.
`Sensor`	Referencja do czujnika, który utworzył ten obiekt `SensorEvent`.
`timestamp`	Znacznik czasu z dokładnością nanosekundową, informujący o momencie utworzenia zdarzenia. Pole to może być przydatne przy korelowaniu wielu zdarzeń.
`values[3]`	Wartości z czujnika zapisane jako tablica liczb zmiennoprzecinkowych z trzema wartościami. Jednostki i dokładność tych trzech wartości jest zależna od czujnika.

Obiekt `SensorEventListener` otrzymuje zdarzenie za każdym razem, gdy zmieni się wartość sensora. Na poniższym listingu przedstawiona jest uproszczona wersja metody `onSensorChanged()` z aplikacji SenseBot.

Listing 14.4. Uproszczona wersja metody onSensorChanged()

```
public void onSensorChanged(SensorEvent event) {                    ❶
      try {
        if (bConnected == false) return;                             ❷
        StringBuilder sb = new StringBuilder();
        sb.append("[" + event.values[0] + "]");                      ❸
        sb.append("[" + event.values[1] + "]");
        sb.append("[" + event.values[2] + "]");

        readings.setText(sb.toString());                             ❹

        // przetworzenie tych danych z czujnika                      ❺
        // updateMotors();                                           ❻
      } catch (Exception e) {
        Log.e(tag,"Błąd w onSensorChanged ::" + e.getMessage());
      }
    }
```

Za każdym razem, gdy dostępny jest obiekt `SensorEvent` ❶, jest on przekazywany do metody `onSensorChanded()`. Pierwszą operacją w tym kodzie jest sprawdzenie, czy mamy działające połączenie z robotem ❷. Jeżeli nie ma połączenia, ignorujemy dane. Każda z trzech wartości jest odczytywana i formatowana ❸ do wyświetlenia w polu `TextView` ❹. Wartości są interpretowane ❺ i odpowiednie instrukcje są wysyłane do kontrolera sterującego silnikami robota ❻. Kod odpowiedzialny za interpretację i interakcję z robotem przedstawiono w dalszej części tego rozdziału.

Aplikacja musi zarejestrować swój obiekt `SensorEventListener`, aby otrzymywać te powiadomienia. W następnym punkcie przedstawimy zalecany sposób wykonywania procesu rejestracji.

14.2.3. Włączanie i wyłączanie czujników

Obiekt implementujący interfejs `SensorEventListener` otrzymuje komunikaty wyłącznie wtedy, gdy jest zarejestrowany. Klasa `SensorManager` zawiera dwie funkcje pozwalające aplikacji na rejestrowanie zdarzeń czujnika. W kontekście aplikacji SenseBot interesuje nas otrzymywanie zdarzeń z czujnika położenia wyłącznie wtedy, gdy urządzenie jest połączone z robotem poprzez Bluetooth. Dlatego kod rejestracji umieściliśmy w przedstawionej wcześniej metodzie `handleConnected()`. Na poniższym listingu zamieszczony jest nowy kod, dodany do metody `handleConnected()`.

Listing 14.5. Kod rejestracji czujnika

```
sManager.registerListener(SenseBot.this,                    ❶
      sManager.getDefaultSensor(
         Sensor.TYPE_ORIENTATION),                          ❷
      SensorManager.SENSOR_DELAY_UI);                       ❸
```

Metoda `registerListener()` z klasy `SensorManager` oczekuje trzech argumentów potrzebnych do przekazania danych z czujnika do aplikacji. Pierwszym argumentem jest obiekt implementujący interfejs `SensorEventListener`, którym w tym przypadku jest sama klasa `SenseBot.this` ❶. Drugim argumentem jest obiekt interesującego nas czujnika. W tym przypadku jesteśmy zainteresowani śledzeniem wartości z czujnika położenia ❷. Częstotliwość, z jaką są aktualizowane dane czujników, jest zmienna i definiowana przez programistę za pomocą trzeciego parametru. W tym przypadku użyliśmy stałej `Sensor ↪Manager.SENSOR_DELAY_UI` ❸, która jest dobrą, uniwersalną wartością. Dla gier i innych aplikacji czasu rzeczywistego należy użyć większych wartości.

Jak pamiętamy, czujnik położenia potrzebuje 7 mA. Aby wydłużyć czas pracy baterii, należy pamiętać o wyłączeniu czujnika, jeżeli nie jest on potrzebny. W aplikacji SenseBot istnieją dwa miejsca, w których ma to miejsce. Pierwszym jest metoda `handle ↪Disconnected()` — gdy utracimy połączenie z robotem, nie ma sensu próbować odczytywać dane z sensora. Innym miejscem, w którym należy dodać funkcję „wyrejestrowania", jest metoda cyklu życia aktywności, `onStop()`.

Niezależnie od miejsca, z którego wywołany jest kod, obiekt `SensorEventListener` jest wyrejestrowywany za pomocą prostej metody `unregisterListener()` z klasy `SensorManager`:

```
sManager.unregisterListener(SenseBot.this);
```

Trzeba pamiętać, że jeżeli aplikacja zarejestrowała więcej niż jeden typ czujnika, konieczne jest wyrejestrowanie wszystkich tych czujników.

Wiemy już, jak podłączyć się do robota i odczytywać wartości z czujnika położenia. Czas połączyć ze sobą te informacje i zbudować aplikację SenseBot!

14.3. Budowanie aplikacji SenseBot

Założenia aplikacji SenseBot są proste — chcemy sterować robotem LEGO Mindstorms NXT[5] przez zmianę orientacji telefonu Android. Nie korzystamy z przewodów — cała komunikacja jest realizowana poprzez Bluetooth, a orientacja telefonu pozwala określić sposób poruszania się robota. Choć robot LEGO jest programowalny, korzystamy tylko z wbudowanych możliwości manipulowania poszczególnymi silnikami. Zaletą takiego podejścia jest możliwość wykorzystywania tego programu z niemal każdym robotem LEGO, niezależnie od umiejętności programisty robota. Jedynym wymaganiem jest podłączenie silników do portów wyjściowych B oraz C, co jest częstą praktyką przy konstruowaniu robotów LEGO NXT. Na rysunku 14.5 pokazany jest robot z o prostej, dwusilnikowej konstrukcji.

Rysunek 14.5. Prosty robot LEGO NXT z silnikami podłączonymi do portów B oraz C

Robot może jechać w przód, w tył, skręcać w lewo oraz w prawo. Przechylenie telefonu do przodu lub do tyłu spowoduje jazdę robota, natomiast przechylenie go w lewo lub prawo powoduje, że robot skręca.

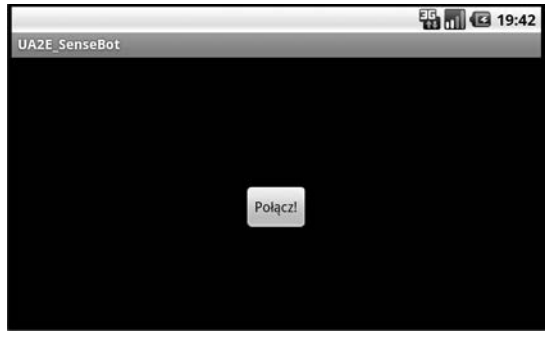
Choć robot jest kontrolowany wyłącznie przez ruchy telefonu, nadal musimy utworzyć przydatny i intuicyjny interfejs użytkownika. W rzeczywistości interfejs użytkownika w tej aplikacji jest bardzo ważny.

14.3.1. Interfejs użytkownika

Interfejs użytkownika tej aplikacji jest prosty, ale musi być również intuicyjny dla użytkownika. Chcemy pokazać użytkownikowi, co się dzieje, aby zapewnić mu informacje na temat sposobu używania aplikacji. Dodatkowo komunikujemy się z mechanicznym robotem, który niekoniecznie może prawidłowo działać. Robot może wykonać nie-

[5] Jeżeli masz w rodzinie przyszłego inżyniera robotyki, warto zapoznać się z ligą First Lego League: *http://www.firstlegoleague.org/*.

oczekiwaną akcję — dlatego pożądane jest, aby można było porównać ruchy robota z wizualnymi wskaźnikami wyświetlanymi w interfejsie użytkownika. Musimy więc informować użytkownika o stanie silników przez cały czas, gdy urządzenie Android jest podłączone do robota. Na rysunku 14.6 pokazany jest domyślny interfejs użytkownika przed podłączeniem robota.



Rysunek 14.6. Oczekiwanie na podłączenie robota

Kliknięcie przycisku *Połącz* inicjuje sekwencję łączenia, zawierającą metodę `findRobot()`, pokazaną wcześniej w punkcie „Podłączanie się do zdalnego urządzenia". Po podłączeniu do robota musimy ukryć przycisk *Połącz* i zapewnić możliwość odłączenia od robota za pomocą przycisku *Rozłącz*. Dodatkowo chcemy wyświetlić stan silników i wyświetlać odczyty z czujnika. Na rysunku 14.7 pokazana jest aplikacja po połączeniu, gdzie robot ma wyłączone silniki.

Rysunek 14.7. Podłączenie do robota z zatrzymanymi silnikami

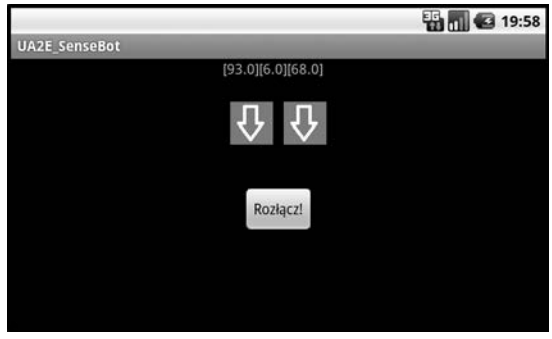
Uwaga

Wskaźniki silników na ekranie są wartościami zdefiniowanymi w aplikacji i skorelowanymi z instrukcjami sterowania silnikami, wysyłanymi do robota. Nie są to wartości odczytywane z robota.

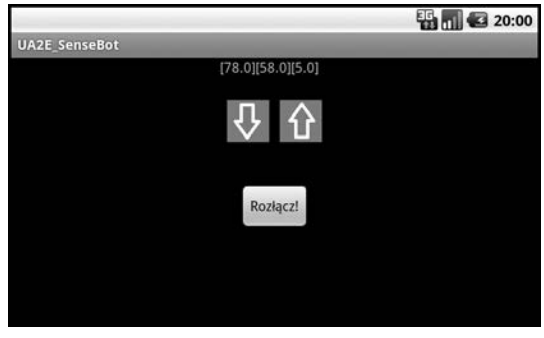
Jeżeli silniki robota pracują, a na ekranie pokazane jest, że oba są zatrzymane, występuje problem z wysłanym poleceniem lub z samym robotem.

Na rysunku 14.8 pokazany jest ekran aplikacji w przypadku wydania polecenia cofania.

Na rysunku 14.9 pokazana jest aplikacja wysyłająca komunikat skrętu w lewo. Aby wykonać taką operację, lewy silnik pracuje w tył, a prawy w przód.

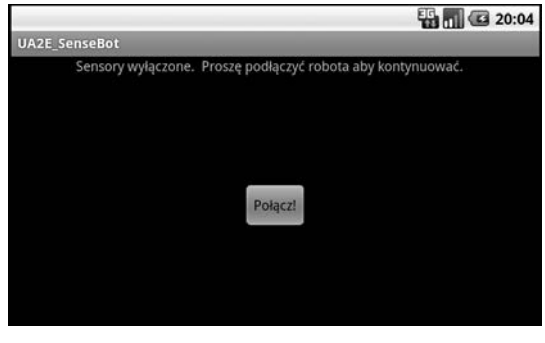


Rysunek 14.8. Oba silniki działają w tył



Rysunek 14.9. Skręt w lewo

Na koniec, gdy aplikacja odłączy się od robota (albo przez kliknięcie przycisku *Rozłącz*, albo przez wyłączenie robota), aplikacja wykrywa zdarzenie rozłączenia i wywołuje metodę `handleDisconnect()`, a interfejs użytkownika jest aktualizowany, jak jest to pokazane na rysunku 14.10.



Rysunek 14.10. Stan odłączenia, oczekiwanie na nowe połączenie

Interfejs użytkownika składa się z dwóch elementów View oraz trzech rysunków[6]: stop, w górę (w przód) oraz w dół (w tył). Na podstawie wartości z czujnika odpowiedni element `View` ma ustawiane odpowiednie tło.

[6] Warto pobrać aplikację wyświetlającą wszystkie zasoby znajdujące się w *android.R.drawable* urządzenia Android: *http://www.appbrain.com/app/android-drawables/aws.apps.androidDrawables*.

Aplikacja ta jest tak zależna od orientacji telefonu przy sterowaniu robotem, że nie możemy pozwolić na zmiany z układu pionowego na poziomy i odwrotnie, ponieważ zarówno powoduje to ponowne uruchomienie aktywności, co może wprowadzić sporo zamieszania, jak również zmienia orientację czujników. Aby spełnić to wymaganie, w pliku `AndroidManifest.xml` należy do znacznika `activity` dodać poniższy atrybut:

```
android:screenOrientation=landscape
```

Po skonfigurowaniu orientacji nie ma obawy o zmianę układu z poziomego na pionowy przy sterowaniu robotem. Uznaliśmy, że trzymanie telefonu poziomo pozwala na wygodne „kierowanie".

Aby zapewnić dokładne skoordynowanie interfejsu użytkownika z fizycznymi silnikami, konieczne jest przygotowanie mechanizmu sprzężenia, dzięki czemu można lepiej sterować robotem, jak i pomóc przy rozwiązywaniu problemów z anomaliami w czasie testowania tego projektu inżynierskiego.

Komunikacja jest gotowa do działania, a czujniki orientacji dostarczają danych; czas zająć się interpretowaniem wartości z czujników.

14.3.2. Interpretowanie wartości czujnika

Aby sterować robotem przy użyciu orientacji telefonu, należy zdefiniować „martwą strefę", której „środek" jest reprezentowany przez pozycję telefonu w poziomie, z niewielkim odchyleniem w tył. Po zdefiniowaniu tej pozycji centralnej dodajemy w wymiarach x i y wygodne w użyciu zakresy czułości. Dopóki orientacja telefonu nie przekroczy wartości czułości, silniki pozostają wyłączone. Zmienne o nazwach `xCenter`, `yCenter`, `xSensitivity` oraz `ySensitivity` pozwalają utworzyć „neutralny prostokąt".

Spójrzmy na metodę `onSensorChanged()`: w niej właśnie odbieramy obiekty `SensorEvent` zawierające wartości dla każdego z wymiarów, `x`, `y` oraz `z`. Na poniższym listingu pokazana jest kompletna implementacja tej metody wraz z analizą czujników oraz sugestiami ruchu.

Listing 14.6. Metoda onSensorChanged() interpretująca orientację

```
public void onSensorChanged(SensorEvent event) {
      try {
        if (bConnected == false) return;
        StringBuilder sb = new StringBuilder();
        sb.append("[" + event.values[0] + "]");
        sb.append("[" + event.values[1] + "]");
        sb.append("[" + event.values[2] + "]");
        readings.setText(sb.toString());
        // przetwarzanie danych z czujników
        movementMask = MOTOR_B_STOP + MOTOR_C_STOP;          ❶

        if (event.values[2] < (yCenter - ySensitivity)) {
          movementMask = MOTOR_B_FORWARD + MOTOR_C_FORWARD;
          motorPower = 75;                                   ❹
                                                             ❷
        } else if (event.values[2] > (yCenter + ySensitivity)) {
          movementMask = MOTOR_B_BACKWARD + MOTOR_C_BACKWARD;
          motorPower = 75;                                   ❹
```

```
      } else if (event.values[1] >(xCenter + xSensitivity)) {
        movementMask = MOTOR_B_BACKWARD + MOTOR_C_FORWARD;

        motorPower = 50;
      } else if (event.values[1] < (xCenter - xSensitivity)) {
        movementMask = MOTOR_B_FORWARD + MOTOR_C_BACKWARD;
        motorPower = 50;
      }
     updateMotors();
     } catch (Exception e) {
       Log.e(tag,"Błąd onSensorChanged ::" + e.getMessage());
     }
   }
```

Przy interpretacji wartości dla silników domyślnie mamy oba silniki zatrzymane ❶. Zwróć uwagę, że silniki B i C są zarządzane osobno. Na początek sprawdzamy, czy wartość czujnika *y* jest poza martwą strefą osi *y* ❷. Jeżeli wykryta wartość jest poza granicą „przechylony w przód", ruszamy robotem do przodu. Podobnie jeżeli odczytana wartość jest mniejsza od granicy wartości spoczynkowej, ruszamy robotem do tyłu, przez włączenie wstecznych obrotów obu silników. Jeżeli nie określiliśmy ruchu robota w przód lub w tył, sprawdzamy kolejne opcje, dla ruchu w lewo lub w prawo ❸. Jeżeli robot przesuwa się w przód lub w tył, szybkość jest ustawiona na `75%` ❹. Jeżeli robot skręca, jego szybkość jest ustawiona na `50%` ❺. Ostatnim krokiem jest przekształcenie tych masek ruchu na prawdziwe akcje przez zmodyfikowanie stanu silników ❻ i zaktualizowanie interfejsu, aby odzwierciedlał te polecenia.

Ponieważ metoda `onSensorChanged()` w pełni przetwarza dane `SensorEvent`, czas na uruchomienie silników robota i zaktualizowanie interfejsu użytkownika.

14.3.3. Jazda robotem

Jazda robotem jest bardzo prosta — i jednocześnie złożona — ponieważ polega tylko na włączeniu silników za pomocą serii poleceń. Sam protokół poleceń jest przedstawiony w następnym punkcie, a teraz skupimy się na metodzie `updateMotors()`, w której modyfikowany jest interfejs użytkownika oraz stan silników. Na poniższym listingu zamieszczona jest metoda `updateMotors()`.

Listing 14.7. Metoda updateMotors()

```
private void updateMotors() {
    try {
      if ((movementMask & MOTOR_B_FORWARD) == MOTOR_B_FORWARD) {
        motorB.setBackgroundResource(R.drawable.uparrow);
     MoveMotor(MOTOR_B,motorPower);

     } else if ((movementMask & MOTOR_B_BACKWARD) == MOTOR_B_BACKWARD) {
        motorB.setBackgroundResource(R.drawable.downarrow);
        MoveMotor(MOTOR_B,-motorPower);
     } else {
        motorB.setBackgroundResource(R.drawable.stop);
        MoveMotor(MOTOR_B,0);
      }
```

```
      if ((movementMask & MOTOR_C_FORWARD) == MOTOR_C_FORWARD) {
        motorC.setBackgroundResource(R.drawable.uparrow);
        MoveMotor(MOTOR_C,motorPower);

      } else if ((movementMask & MOTOR_C_BACKWARD) == MOTOR_C_BACKWARD) {
        motorC.setBackgroundResource(R.drawable.downarrow);
        MoveMotor(MOTOR_C,-motorPower);

      } else {
        motorC.setBackgroundResource(R.drawable.stop);
        MoveMotor(MOTOR_C,0);
      }
    } catch (Exception e) {
      Log.e(tag,"Błąd w updateMotors ::" + e.getMessage());
    }
}
```

W metodzie `updateMotors()` porównywane są żądania ruchu zdefiniowane w postaci zmiennej `movementMask`, osobnej dla każdego z silników ❶. Gdy zostanie znaleziona pasująca wartość — na przykład gdy ustawiony jest bit `MOTOR_B_FORWARD` — włączany jest dany silnik w zdefiniowanym kierunku i szybkości ❸. Kierunek ujemny oznacza jazdę w tył, a wartość szybkości jest skalowana do zakresu od 0 do 100. Dodatkowo aktualizowany jest interfejs użytkownika ❷ w połączeniu z samymi silnikami, co daje użytkownikowi możliwie dokładny obraz działania.

14.3.4. Komunikacja z robotem

Protokół komunikacji pozwalający na interakcję z robotem LEGO NXT składa się z poleceń strukturalnych z opcjonalnym protokołem odpowiedzi. Każdy pakiet danych jest umieszczony w kopercie opisującej jego rozmiar. Wewnątrz koperty każde polecenie protokołu Direct Command posiada standardowy nagłówek, po którym następują specyficzne parametry. W aplikacji tej potrzebujemy tylko jednego polecenia — powodującego ustawienie stanu działania silnika. Kod pozwalający na zbudowanie i wysłanie tych pakietów jest pokazany na poniższym listingu.

Listing 14.8. Metoda MoveMotor

```
private void MoveMotor(int motor,int speed)
{
    try
    {
      byte[] buffer = new byte[14];
      buffer[0] = (byte) (14-2); //długość lsb
      buffer[1] = 0; // długość msb
      buffer[2] =  0; // polecenie Direct Command (z odpowiedzią)
      buffer[3] = 0x04; // ustawienie stanu wyjścia
      buffer[4] = (byte) motor; // wyjście 0, 1, 2 (silniki A, B, C)
      buffer[5] = (byte) speed; // moc
      buffer[6] = 1 + 2; // włączenie silnika + hamulec pomiędzy PWM
      buffer[7] = 0; // regulacja
      buffer[8] = 0; // rotacja skrętu
      buffer[9] = 0x20; // stan działania
      buffer[10] = 0; // cztery bajty danych pozycji
```

```
        buffer[11] = 0; // ustawione na zero
        buffer[12] = 0;
        buffer[13] = 0;

        os.write(buffer);
        os.flush();
        byte response [] = ReadResponse(4);
      }
      catch (Exception e)
      {
        Log.e(tag,"Błąd w MoveForward(" + e.getMessage() + ")");
      }
  }
```

❷

❸

Kod ten realizuje prostą, choć precyzyjną operację formatowania polecenia wysyłanego do robota LEGO, które zapewnia bezpośrednią kontrolę nad silnikami. Na początek deklarujemy bufor o odpowiedniej wielkości ❶. Rozmiar tego bufora jest definiowany przez polecenie `SetOutputState`, które jest jednym z wielu poleceń obsługiwanych przez robota. Każda z informacji jest umieszczana w odpowiednim miejscu bufora ❷. Po sformatowaniu bufora polecenia jest on zapisywany do gniazda, a gniazdo opróżniane ❸. Kod odpowiedzi odczytywany przez metodę `ReadResponse()` jest znacznikiem prawidłowego odbioru polecenia. Jak się okazuje, oprócz specyficznego formatowania danych sterujących robotem, wysyłanie i odbieranie danych poprzez Bluetooth jest równie proste, co odczyt i zapis do bufora bajtów.

Na tym etapie czujniki działają, a urządzenie Android i robot LEGO komunikują się ze sobą. Z czasem, po nabraniu praktyki, możesz się stać doskonałym pilotem robota LEGO. Pełny kod źródłowy tej aplikacji jest dostępny do pobrania.

14.4. Podsumowanie

W tym rozdziale wprowadziliśmy dwie funkcje platformy Android zorientowane sprzętowo: Bluetooth i czujniki. Z tych dwóch pozornie niezwiązanych obszarów działania wyrosła aplikacja pozwalająca na operowanie robotem LEGO Mindstorms NXT. Pokazaliśmy tu najważniejsze kroki wymagane do połączenia urządzenia Android ze zdalnym partnerem obsługującym Bluetooth, z użyciem protokołu zastępującego połączenie kablowe, RFCOMM. Ten kanał komunikacyjny jest używany do wymiany zestawu poleceń znanych jako protokół Direct Command, udostępniany przez kontroler LEGO NXT. Dzięki tym poleceniom można manipulować silnikami robota, wprawiając go w ruch. Aby interfejs użytkownika był możliwie intuicyjny, skorzystaliśmy z czujnika orientacji, wbudowanego w większość telefonów z systemem Android, który pozwala na odczytywanie ruchów użytkownika. Położenie urządzenia jest interpretowane i odpowiadająca im seria poleceń jest wysyłana do robota. Sensory nie tylko są dobrą metodą sterowania robotem, ale również dają dużo zabawy!

Oprócz komunikacji poprzez Bluetooth i użycia czujników w rozdziale tym pokazaliśmy również techniki zapewniające intuicyjne informowanie użytkownika o operacjach wykorzystywanych przez aplikację. Na przykład gdy silniki są włączone, użytkownik widzi na ekranie kierunek obrotu każdego z nich. Podobnie ruchy użytkownika są przetwarzane wyłącznie w przypadku aktywnego połączenia Bluetooth. Ponieważ jest to

scenariusz sterowany zdarzeniami, aplikacja demonstruje nasłuch tych zdarzeń poprzez dynamicznie rejestrowany obiekt `BroadcastReceiver`, z odpowiednimi filtrami intencji.

Mamy nadzieję, że lekcja użycia komunikacji Bluetooth oraz czujników, przedstawiona w tym rozdziale, była ciekawa, a jeżeli masz dostęp do robota LEGO Mindstorm, zachęcamy do jazdy próbnej.

W następnym rozdziale pokażemy inne sposoby podłączania urządzenia Android do zewnętrznego świata — tym razem będziemy wykorzystywać możliwości platformy do synchronizowania danych z popularną biznesową witryną społecznościową LinkedIn.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Ż

Wkrocz wreszcie do akcji i zacznij tworzyć własne aplikacje dla Androida!

Skoro zwróciłeś uwagę właśnie na tę książkę, zapewne dobrze wiesz, czym jest Android i co potrafi — teraz przyszła pora, abyś sprawdził go także w akcji! Oto doskonała propozycja dla wszystkich programistów, którym marzy się tworzenie własnych aplikacji dla robiącego oszałamiającą karierę systemu. Choć ta książka nie jest przeznaczona dla początkujących, zawiera wszystkie informacje potrzebne osobom, dla których Android to całkowicie nowe środowisko. Można w niej znaleźć instrukcje niezbędne do szybkiego zorientowania się w architekturze tej platformy oraz sposobie jej działania, co pozwoli sprawnie rozpocząć pracę w tym środowisku. Pozostałe rozdziały to już czysta frajda programowania!

Od czego zaczniesz zabawę z Androidem? Dowiesz się, jak budować aplikacje dla tego systemu od najmniejszych cegiełek aż po ekrany, dodawać funkcje telefoniczne i wykorzystywać bibliotekę OpenGL ES do tworzenia złożonej grafiki 2D oraz 3D. Następnie poznasz zasady tworzenia większych aplikacji oraz techniki pisania aplikacji w języku C, także z użyciem Android Native Development Kit. Opanujesz potężne narzędzie Android SDK oraz budowanie aplikacji dla WebKit z wykorzystaniem HTML 5, a nawet nauczysz się rozszerzać lub zastępować wbudowane funkcje Androida na podstawie użytecznych i intrygujących przykładów.

- Wprowadzenie do systemu Android
- Środowisko programowania
- Komponenty interfejsu użytkownika, w tym View i Layout
- Metody przechowywania i odczytywania lokalnych danych
- Sieci oraz usługi sieciowe
- Przegląd podstawowych zagadnień związanych z telefonią
- Powiadomienia i alarmy
- Grafika i animacja w Androidzie
- Korzystanie z funkcji multimedialnych Androida
- Usługi oparte na lokalizacji
- Integracja kontaktów z platformy społecznościowej
- Aplikacja wspomagająca serwisantów
- Budowanie aplikacji na Androida w języku C
- Bluetooth, sensory i widżety aplikacji
- Tworzenie stron WWW dla systemu Android
- Strategie lokalizowania aplikacji
- Android Native Development Kit
- Korzystanie z SDK oraz AVD Manager

nr katalogowy 71122

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

helion.pl
helion sa
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Cena 99,00 zł
ISBN 978-83-246-3380-7