

Programuj gry wykorzystujące wszystkie zalety tabletów!



Android

Programowanie gier na tablety

Jeremy Kerfs

Apress®



Tytuł oryginału: Beginning Android Tablet Games Programming

Tłumaczenie: Rafał Szpoton

ISBN: 978-83-246-5004-0

Original edition copyright © 2011 by Jeremy Kerfs.
All rights reserved.

Polish edition copyright 2012 by HELION SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/androt.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/androt>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to!» Nasza społeczność](#)

Spis treści

O autorze	7
O redaktorze technicznym	8
Podziękowania	9
Rozdział 1. Konfiguracja środowiska programistycznego Java dla systemu Android 3.0	11
Czym jest system Android?	11
Początki systemu Android	11
Główne cechy systemu Android 3.0	13
Czego potrzeba do tworzenia gier w systemie Android?	14
Co należy wiedzieć?	14
Środowisko programistyczne	15
Konfiguracja środowiska programistycznego	16
Instalacja pakietu Java JDK	16
Instalacja środowiska Eclipse	17
Instalacja pakietu SDK dla systemu Android	20
Konfiguracja narzędzi Androida oraz urządzenia wirtualnego w środowisku Eclipse	23
Sprawdzanie działania narzędzi programistycznych	26
Tworzenie projektu dla systemu Android	27
Projekt programu dla systemu Android w Eclipse	29
Tworzenie wirtualnego urządzenia z Androidem	31
Uruchamianie aplikacji	33
Pierwsze zmiany w aplikacji	33
Podsumowanie	35
Rozdział 2. Tworzenie prostych gier z użyciem ruchomych sprajtów	37
Praca z obrazami	37
Tworzenie podłoża do wyświetlania obrazów	38
Rysowanie obrazu	42
Używanie sprajtów	44
Uruchomienie gry	49

	Nadawanie grze profesjonalnego wyglądu	51
	Implementacja zarządzania czasem oraz złożonym ruchem	52
	Wykrywanie kolizji	53
	Podsumowanie	54
Rozdział 3.	Pobieranie danych od użytkownika	55
	Sposoby pobierania danych wejściowych	55
	Pobieranie danych wejściowych w tablecie	57
	Reagowanie na dotyk	59
	Reagowanie na gesty	61
	Korzystanie z kolejek wejścia	64
	Reagowanie na dane pochodzące z czujników	70
	Korzystanie z danych z czujnika	73
	Podsumowanie	75
Rozdział 4.	Dodawanie efektów dźwiękowych, muzyki oraz sekwencji filmowych	77
	Przygotowanie do odtwarzania dźwięków	78
	Szukanie oraz dodawanie efektów dźwiękowych	78
	Odtwarzanie efektów dźwiękowych	79
	Odtwarzanie wielu efektów dźwiękowych	80
	Dopasowanie efektów dźwiękowych do zdarzeń	84
	Dodawanie muzyki	85
	Dodawanie sekwencji filmowych	86
	Zarządzanie obsługą muzyki	87
	Podsumowanie	94
Rozdział 5.	Tworzenie jednoosobowej gry z utrudnieniami	95
	Planowanie gry jednoosobowej — AllTogether	95
	Tworzenie gry jednoosobowej	96
	Ulepszanie sprajtów gry	97
	Dodawanie nagrody za ukończenie gry	100
	Śledzenie stanu sprajtów	101
	Podsumowanie	109
Rozdział 6.	Gra w odbijaną piłkę	111
	Początki	111
	Gromadzenie zasobów używanych w grze	112
	Tworzenie nowego projektu	113
	Przygotowanie środowiska gry	114
	Modyfikacja pliku SpriteObject.java	114
	Modyfikacja pliku GameView.java	114
	Dodawanie wykrywania kolizji oraz obsługi zdarzeń	117
	Dodawanie obsługi dotyku, dźwięku oraz nagród	121
	Dodawanie dotykowego sterowania rakieta	121
	Dodawanie dźwięków	122
	Inicjalizacja bloków	123
	Usuwanie nieaktywnych bloków	125
	Podsumowanie	126

Rozdział 7.	Tworzenie gry dwuosobowej	127
	Podstawy gier wieloosobowych	127
	Gry wieloosobowe wykorzystujące serwer gier	128
	Gry wieloosobowe z połączeniami równorzędnymi	128
	Wybór metody rozgrywki wieloosobowej	129
	Gra dwuosobowa z połączeniami równorzędnymi	130
	Dodawanie połączeń Bluetooth	130
	Zarządzanie połączeniami Bluetooth	134
	Modyfikacja kodu gry dla dwóch graczy	140
	Testowanie gry	141
	Podsumowanie	142
Rozdział 8.	Jednoosobowa gra strategiczna. Część I. Tworzenie gry	143
	Wprowadzenie do obrony portu	144
	Składanie elementów gry	144
	Tworzenie falochronu	145
	Dodawanie gruntu oraz zamku	148
	Tworzenie łodzi	149
	Dodawanie dział	151
	Dodawanie obrazów	151
	Testowanie gry	152
	Podsumowanie	154
Rozdział 9.	Jednoosobowa gra strategiczna. Część II. Programowanie gry	155
	Rozszerzenie sprajtów używanych w grze	156
	Projektowanie sterowania grą	157
	Rozmieszczanie elementów na ekranie	162
	Dodawanie łodzi oraz sterowanie nimi	163
	Strzelanie z dział	164
	Wynik działania gry	167
	Analiza gry	168
	Podsumowanie	169
Rozdział 10.	Publikacja gry	171
	Poprawianie aplikacji	171
	Dodawanie ekranu początkowego	171
	Reakcja na wciśnięcie przycisku	174
	Opakowywanie gry	175
	Rozpowszechnianie gry	176
	Otwieranie konta w usłudze Google Play	179
	Wysyłanie aplikacji do sklepu Google Play	180
	Reklamowanie gry	180
	Podsumowanie	181
Dodatek A	Testowanie gier dla systemu Android na prawdziwym urządzeniu	183
	Skorowidz	187

ROZDZIAŁ 7

Tworzenie gry dwuosobowej

Tworząc gry na tablety z systemem Android, wykonaliśmy już do tej chwili całkiem sporo fantastycznej pracy. Teraz jesteśmy gotowi, aby do naszego dzieła dodać kolejny poziom zaawansowania, umożliwiając graczowi współzawodnictwo z innymi osobami znajdującymi się w pobliżu. Będzie to następny, mający duże konsekwencje krok milowy w programowaniu gier. Jeżeli przyjrzymy się wielu popularnym obecnie grom, przekonamy się, że większość z nich jest wybierana przez graczy głównie ze względu na możliwość przeprowadzania z własnego domu rozgrywek z przyjaciółmi oraz zupełnie obcymi osobami. Dodawanie funkcji łączenia wielu urządzeń jest całkiem skomplikowanym procesem. Na szczęście dokumentacja systemu Android zawiera przykłady, które można dostosować do swoich potrzeb w taki sposób, aby uzyskać pożądaný efekt. Musimy więc jedynie zrozumieć sposób działania kodu, a następnie dołączyć go do własnych gier.

W tym rozdziale zajmujemy się szeregiem różnych zagadnień związanych z grami wieloosobowymi, omawiając jednocześnie wiele różnych ich typów oraz implementacji. Następnie skoncentrujemy się na systemie Android. Pod koniec rozdziału Czytelnik będzie już rozumiał, w jaki sposób zmodyfikować własne gry, aby umożliwić w nich rozgrywkę wieloosobową. Zanim jednak zagłębimy się w ten temat, przyjrzymy się różnym rodzajom trybów wieloosobowych oraz ich typowej implementacji.

■ **Uwaga!** Jeżeli jakkolwiek umieszczony w tym rozdziale fragment kodu będzie w pierwszej chwili dla Czytelnika niezrozumiały, powinien on czytać dalej, a po pewnym czasie wszystko ułoży się w spójną całość. Jeśli jednak pewne fragmenty wciąż pozostaną dla niego niezrozumiałe, powinien sprawdzić rozwiązania dostępne w internecie lub też uruchomić programy, modyfikując jedynie ich niezbędne elementy. Dobrym miejscem do rozpoczęcia poszukiwań jest zawsze dokumentacja systemu Android, dostępna pod adresem: <http://developer.android.com/guide/index.html>. Jeżeli tylko Czytelnik zrozumie, w jaki sposób działa przykładowy kod, bardzo często nie będzie musiał nawet umieć odtwarzać go od podstaw.

Podstawy gier wieloosobowych

Czy Czytelnik grał kiedykolwiek na komputerze lub konsoli gier wideo z innymi graczami w strzelankę typu FPS? Tego rodzaju gry przynoszą swoim twórcom corocznie setki milionów dolarów zysku, ponieważ ich główną cechą jest włączanie do rozgrywki innych graczy, a nie tylko postaci kreowanych przez komputer.

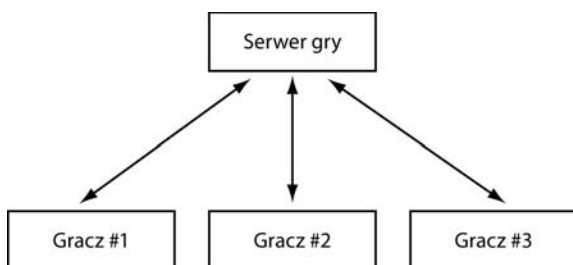
Bardzo popularne są również gry sieciowe udostępniające całe światy (spójrzmy na przykład na grę World of Warcraft). Także tablety oraz telefony powoli wkraczają do tego świata połączeń.

Prawdopodobnie najnowszym rodzajem gier wieloosobowych są gry społecznościowe. Aplikacje w rodzaju Farmville, Mafia Wars oraz wiele innych łączą się z witrynami społecznościowymi (głównie ze stroną Facebook) i przesyłają tam informację o postępie gracza oraz informują o nim przyjaciół tego gracza.

Gry wieloosobowe wykorzystujące serwer gier

Wszystkie gry wspomniane powyżej używają do łączenia graczy serwera pośredniczącego. Oznacza to, że ani urządzenia, ani gracze nie są połączeni ze sobą bezpośrednio, ale raczej poprzez dodatkową warstwę. W rzeczywistości strony internetowe wykorzystują tę samą metodę. Użytkownik (klient) pobiera ze strony informację umieszczoną na serwerze WWW (serwer).

Na rysunku 7.1 przedstawiony jest prosty schemat ilustrujący sytuację, w której kilku graczy łączy się z serwerem, aby grać w grę wieloosobową.



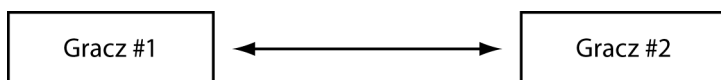
Rysunek 7.1. Grupa graczy łączy się z centralnym serwerem z różnych miejsc, co pozwala im grać ze sobą

Zanim jednak przyjrzymy się wadom i zaletom gier wieloosobowych korzystających z serwera gier, pomocne może okazać się porównanie tej metody z inną. Spójrzmy więc teraz na metodę połączeń równorzędnych (ang. *peer-to-peer*).

Gry wieloosobowe z połączeniami równorzędnymi

W sytuacji gdy gracze łączą się ze sobą bezpośrednio, używają sieci połączeń równorzędnych (ang. *P2P network*). Gry umożliwiające tego rodzaju połączenia, używane przez graczy znajdujących się blisko siebie, wykorzystują zazwyczaj połączenia typu Bluetooth. Obsługa tego standardu dostępna jest w większości tabletów z systemem Android. Połączenia tego rodzaju nie mają żadnej warstwy kontrolującej proces komunikacji. Jeżeli Czytelnik używał w przeszłości do wymiany plików sieci P2P (na przykład pobierając duże pliki z sieci torrent), wie, że był wtedy połączony z innymi komputerami w sposób równorzędny. Oznacza to, że nie był mu potrzebny żaden duży serwer łączący wszystkich użytkowników. Wiele dużych gier wideo przeznaczonych na konsole gier nie używa połączeń równorzędnych, ponieważ gdyby ich używały, umożliwiałyby rozgrywkę jedynie kilku graczom jednocześnie.

Różnica pomiędzy grą wykorzystującą połączenia klient-serwer a grą używającą połączeń równorzędnych przedstawiona jest na rysunku 7.2.



Rysunek 7.2. Dwóch graczy w celu przeprowadzenia rozgrywki łączy się bezpośrednio ze sobą

Oczywiście te dwie strategie gier wieloosobowych w sposób znaczący różnią się od siebie. Czytelnik może się zastanawiać, która z nich jest lepsza. Na to pytanie nie ma jednoznacznej odpowiedzi. Natomiast istnieją przykłady sytuacji, w których jedna metoda ma przewagę nad drugą.

Wybór metody rozgrywki wieloosobowej

W tabelach 7.1 oraz 7.2 przedstawionych jest kilka głównych zalet oraz wad każdej z dwóch metod rozgrywek wieloosobowych. Nie jest to oficjalna lista (a niektórzy z czytelników mogą mieć nawet własne zdanie, czy pewne cechy wymienione w tabelach są zaletami, czy w istocie wadami), jednak daje ona pewne bardzo ważne podstawy wyboru właściwego rozwiązania.

Tabela 7.1. Wady oraz zalety gier wieloosobowych z udziałem serwera

Zaleta	Wada
Umożliwia jednoczesny dostęp do gry wielu graczom.	Wymaga dodatkowego sprzętu i prawdopodobnie opłat za dostęp do serwera.
Zmniejsza obciążenie obliczeniowe poszczególnych urządzeń.	Awaria serwera dotyka wszystkich graczy.
Ułatwia dystrybucję uaktualnień oraz poprawek.	Programista musi utworzyć dodatkowy kod obsługujący operacje na serwerze.
Gracze mogą być rozmieszczeni na całym świecie.	Gracze nie mogą się łatwo ze sobą porozumiewać, o ile nie używają komunikatora.

Tabela 7.2. Wady oraz zalety gier wieloosobowych z użyciem połączeń równorzędnych

Zaleta	Wada
Programista nie musi tworzyć zbyt wiele kodu.	W przypadku wykrycia błędu każdy z graczy osobno musi pobrać uaktualnienie.
Usterka pojedynczego urządzenia nie powstrzymuje działania gry na innych urządzeniach.	Liczba graczy jest zazwyczaj ograniczona.
Nie jest konieczny dodatkowy serwer (wszystkie urządzenia zawierają niezbędne technologie).	Urządzenia graczy muszą same wykonywać wszystkie niezbędne obliczenia.
Podczas gry użytkownicy znajdują się zazwyczaj blisko siebie i mogą komunikować się nawzajem.	Prawie niemożliwe jest prowadzenie rozgrywek z graczami na całym świecie.

Jeżeli Czytelnik przyjrzał się dokładnie powyższym tabelom, powinien zauważyć, że kolumna opisująca zalety metody z użyciem serwera jest podobna do kolumny opisującej wady metody z wykorzystaniem połączeń równorzędnych, podobnie jest w przypadku drugiej pary kolumn. Niemniej zgromadzenie w jednym miejscu wad oraz zalet każdej z metod nie prowadzi wcale do podjęcia właściwego wyboru. To Czytelnik musi mieć świadomość tego, jaki efekt chce osiągnąć, a następnie musi wybrać tę metodę, która w największym stopniu mu to umożliwi.

W dalszej części tego rozdziału dostosujemy grę *TabletPaddle*, utworzoną przez nas w rozdziale 6., do rozgrywki dwuosobowej. Każdy z graczy będzie kontrolował na swoim tablecie jedną z rakietek. Ponieważ temat programowania gier wieloosobowych może być dość złożony, w tym rozdziale zajmujemy się przedstawieniem jedynie jego głównych zagadnień. Pełny kod aplikacji dostępny jest pod adresem: <http://code.google.com/p/android-tablet-games/>.

Ponieważ chcemy umożliwić rozgrywkę jedynie dwóm graczom jednocześnie, a przy tym użyć w tym celu najbardziej efektywnego z dostępnych sposobów, wykorzystamy metodę połączeń równorzędnych. Dodatkowo zamiast łączyć graczy poprzez połączenie internetowe w sieci 3G lub WIFI, połączymy ich urządzenia bezpośrednio za pomocą interfejsu Bluetooth, dostępnego w większości urządzeń z systemem Android. W ten sposób oszczędzimy sobie znaczną ilość czasu, który musielibyśmy poświęcić na konfigurację architektury serwera oraz zapewnienie poprawnych połączeń za jego pośrednictwem.

- **Wskazówka** Początkujący programiści gier powinni trzymać się z dala od gier wieloosobowych typu klient-serwer, ponieważ prawie zawsze są one bardziej złożone. Nie powinno to jednak zniechęcać Czytelnika, ponieważ i tak będzie w stanie utworzyć wiele wspaniałych gier używających połączeń Bluetooth. W tym przypadku dodatkową zaletą dla graczy będzie to, że będą przebywać w niewielkiej odległości od siebie. Dzięki temu będą w stanie wymieniać między sobą komentarze na temat skomplikowanych poziomów lub też oddawać się swobodnej rozmowie.

Gra dwuosobowa z połączeniami równorzędnymi

Czytelnik może być słusznie przekonany, że większość tabletów z systemem Android obsługuje standard Bluetooth¹. Prawie wszystkie nowoczesne telefony używają go do połączeń z bezprzewodowymi zestawami słuchawkowymi. Również tablety implementują tę technologię, umożliwiając zastosowanie tych samych zestawów, jak również klawiatur i wielu innych urządzeń zewnętrznych.

Chociaż niektórzy ludzie używają terminu Bluetooth do określenia zestawów słuchawkowych oraz urządzeń służących do podłączenia do telefonów, to jednak w rzeczywistości Bluetooth jest systemem transmisji radiowej, używanym przez różnego rodzaju urządzenia do wymiany zdjęć, muzyki, wideo, a także prawie każdego innego typu danych. Największą zaletą tego standardu jest jednak jego niesamowita szybkość². Jeżeli może on być wykorzystywany do prowadzenia ciągłych rozmów telefonicznych przy użyciu słuchawek bezprzewodowych, możemy być pewni, że będzie się nadawać również do większości typów gier³.

W kolejnych podrozdziałach dostosujemy grę TabletPaddle z poprzedniego rozdziału do rozgrywki dwuosobowej. Najpierw dodamy kod umożliwiający połączenie dwóch tabletów z Androidem przy użyciu ich wbudowanych nadajników Bluetooth, a następnie dołączymy dodatkową raketkę oraz kod pozwalający graczom na współzawodnictwo o kontrolę nad piłką.

Zacznijmy więc od utworzenia nowego projektu w środowisku Eclipse i nazwania go *TwoPlayerPaddleGame*.

Dodawanie połączeń Bluetooth

Ponieważ łączenie wielu urządzeń jest zadaniem złożonym, kod obsługujący tego rodzaju interakcje na tabletach z Androidem będzie znacznie trudniej objaśnić. Fragmenty kodu umieszczone w tym przykładzie stanowią część dostarczonego z przykładami Androida większego projektu o nazwie *BluetoothChat*, obsługującego standard Bluetooth. W tym miejscu użyjemy ich do omówienia głównych zagadnień. Nie wszystkie zmienne z kodu zostały już wcześniej zainicjalizowane, niemniej wciąż mogą posłużyć do omówienia istoty zagadnienia. Zanim jednak zajmiemy się przedstawieniem przykładu, spójrzmy na główne elementy aplikacji korzystającej ze standardu Bluetooth.

Na początek musimy w tablecie zainicjalizować łącze do urządzenia Bluetooth. W tym celu wykonajmy następujące czynności:

¹ Tanie urządzenia spotykane na polskim rynku w cenie ok. 500 zł prawie zawsze pozbawione są interfejsu Bluetooth. Można założyć, że stwierdzenie autora jest w większości przypadków prawdziwe dla urządzeń w cenie od 1000 zł wzwyż — *przyp. tłum.*

² Autor ma na myśli ostatnie standardy Bluetooth 3.X oferujące teoretyczne transfery na poziomie kilkunastu Mb/s. Należy jednak pamiętać, że poprzednie wersje standardu Bluetooth oferowały transfery o rząd wielkości niższe i o ile nadawały się do transmisji głosowej, o tyle transmisja danych była dla nich wyzwaniem. Protokoły używane do transmisji danych głosowych nie wymagają aż tak dużej przepustowości. Dane głosowe poddają się dość dobrej kompresji i mogą być kompresowane stratnie. Natomiast dane aplikacji muszą być zawsze przesyłane bezstratnie, co też stanowi pewne wyzwanie dla medium używanego do transmisji — *przyp. tłum.*

³ Istotną wadą standardu Bluetooth jest jego podatność na złamanie i podsłuchanie. Co prawda wadę tę ogranicza niewielki zasięg transmisji, niemniej decydując się na wykorzystanie standardu Bluetooth do transmisji poufnych danych należy zaimplementować dodatkową warstwę szyfrowania — *przyp. tłum.*

1. Do funkcji `onCreate()` w pliku `MainActivity.java` dodajmy kod przedstawiony na listingu 7.1.

Listing 7.1. Funkcja `onCreate()`

```
BlueAdapter = BluetoothAdapter.getDefaultAdapter();

if (BlueAdapter == null) {
    Toast.makeText(this, "Interfejs Bluetooth nie jest dostępny", Toast.LENGTH_LONG).show();
    return;
}
```

Obiekt `BlueAdapter` stanie się uchwytym do funkcji urządzenia Bluetooth. Instrukcja `if` wykorzystana została do sprawdzenia dostępności interfejsu w danym urządzeniu. W przypadku jego nieobecności użytkownik jest powiadamiany stosownym komunikatem o braku możliwości użycia programu.

2. Kolejny fragment inicjalizujący umieszczony jest w metodzie, z którą nie mieliśmy dotychczas do czynienia — w funkcji `onState()` znajdującej się w pliku `MainActivity.java` tuż po metodzie `onCreate()`. Jej zawartość przedstawiona jest na listingu 7.2. Do poprawnego działania kodu potrzebne jest również dołączenie klasy `android.intent.Intent`, umożliwiającej wysyłanie komunikatów.

Listing 7.2. Metoda `onStart()`

```
@Override
public void onStart() {
    super.onStart();

    if (!BluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    }
    else {
        if (game_running == null) startgame();
    }
}
```

Kod na listingu 7.2 sprawdza najpierw, czy urządzenie Bluetooth jest włączone, czy wyłączone. W przypadku gdy jest niedostępne, zostaje utworzona aktywność z żądaniem włączenia urządzenia (niedługo dowiemy się, co ta aktywność wykonuje). Jeżeli urządzenie jest włączone, następuje sprawdzenie stanu gry. W przypadku gdy gra nie została uruchomiona, wywoływana jest funkcja inicjalizująca nową grę. Zauważmy, jak duża ilość kodu wykorzystywana jest najpierw do sprawdzania dostępności poprawnego połączenia Bluetooth.

3. Kod na listingu 7.3 używany jest po wysłaniu przez aktywność żądania.

Listing 7.3. Metoda `onActivityResult()`

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:

            if (resultCode == Activity.RESULT_OK) {

                String address = data.getExtras()
                    .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);

                BluetoothDevice device = BlueAdapter.getRemoteDevice(address);
            }
    }
}
```

```

        mGameView.connect(device);
    }
    break;

case REQUEST_ENABLE_BT:

    if (resultCode == Activity.RESULT_OK) {

        startgame();
    } else {

        Toast.makeText(this, "Błąd inicjalizacji interfejsu Bluetooth",
↳Toast.LENGTH_SHORT).show();
        finish();
    }
}
}
}

```

Kod umieszczony powyżej wykonuje dwie proste czynności. Jeżeli zostanie wywołany z żądaniem połączenia do innego urządzenia, kod pobierze jego adres i utworzy link do jego interfejsu Bluetooth. Następnie wywoływana jest funkcja obiektu `mGameView` łącząca ze sobą dwa urządzenia.

4. A teraz przysłała pora na bardzo krótką i prostą funkcję `startgame()`. Listing 7.4 przedstawia sposób uruchomienia gry.

Listing 7.4. *Metoda startgame()*

```

private void startgame() {

    mGameView = new GameView(this, mHandler);
    setContentView(mGameView);

}

```

Powyższa metoda nie jest zbyt ciekawa. Istotne jest jednak, aby zauważyć, w jaki sposób do konstruktora klasy `GameView` przekazywany jest nowy argument. Uchwyt do procedury obsługi komunikatów pozwala na przesłanie do gry danych z kanału Bluetooth. Zrozumienie sposobu działania tej procedury jest prawdopodobnie najważniejszym aspektem programowania tego interfejsu.

5. Kod na listingu 7.5 zajmuje się obsługą uchwytu, używanego w różnych zadaniach wysyłania oraz odbierania danych.

Listing 7.5. *Obsługa uchwytu do procedury obsługi komunikatów*

```

private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MESSAGE_STATE_CHANGE:
                switch (msg.arg1) {
                    case BluetoothChatService.STATE_CONNECTED:
                        break;
                    case BluetoothChatService.STATE_CONNECTING:
                        Toast.makeText(this, "Łączenie z Bluetooth", Toast.LENGTH_SHORT).show();
                        break;
                    case BluetoothChatService.STATE_LISTEN:

```

```

        case BluetoothChatService.STATE_NONE:
            Toast.makeText(this, "Brak połączenia z Bluetooth",
                ↳Toast.LENGTH_SHORT).show();
            break;
        }
        break;

    case SEND_DATA:
        byte[] writeBuf = (byte[]) msg.obj;

        String writeMessage = new String(writeBuf);

        break;
    case RECEIVE_DATA:
        byte[] readBuf = (byte[]) msg.obj;

        String readMessage = new String(readBuf, 0, msg.arg1);

        break;
    case MESSAGE_DEVICE_NAME:

        mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
        Toast.makeText(getApplicationContext(), "Podłączony do "
            + mConnectedDeviceName, Toast.LENGTH_SHORT).show();

        break;
    case MESSAGE_TOAST:
        Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
            Toast.LENGTH_SHORT).show();

        break;
    }
}
};

```

Ponieważ w trakcie zaprezentowanej powyżej inicjalizacji uchwytu wykonywanych jest wiele czynności, poniżej przedstawiona zostanie lista różnych wykorzystywanych aktywności. Powrócimy do niej, kiedy będziemy tworzyć własny rzeczywisty projekt. Mówiąc najogólniej, do uchwytu przekazywany jest określony komunikat lub zdarzenie, które musi on obsłużyć albo zignorować. Lista odpowiedzi na zdarzenia wymagająca dodatkowego kodowania jest długa. Należy pamiętać również, że zdarzenia są wysyłane z wnętrza klasy `GameView`.

- `MESSAGE_STATE_CHANGE`: Pierwsza instrukcja `case` sprawdza zmianę stanu połączenia Bluetooth. W większości przypadków konieczne jest poinformowanie użytkownika o zmianie stanu na brak połączenia. Na przykład możemy poinformować użytkownika o tym, że urządzenie próbuje nawiązać połączenie, a następnie, w przypadku gdy ta próba zakończyła się niepowodzeniem, wyjaśnić użytkownikowi przyczynę tej sytuacji. Takie założenie jest przydatne podczas testowania programu.
- `SEND_DATA`: Kolejne zdarzenie obsługuje wysyłanie danych do innego urządzenia. W tym miejscu tworzony jest bufor danych i następuje przygotowanie do ich wysłania. W rzeczywistości w tym momencie żadne dane nie są jeszcze wysyłane. Ta funkcja zostanie dodana w tym miejscu w późniejszym czasie.
- `RECEIVE_DATA`: Analogicznie do fragmentu obsługującego wysyłanie komunikatu występuje tu również część kodu odbierająca dane nadchodzące z innego urządzenia. Podobnie jak poprzednio więcej kodu do tego fragmentu zostanie dodane później, gdy będziemy już wiedzieli, co dokładnie chcemy osiągnąć.

- MESSAGE_DEVICE_NAME: Przedostatni komunikat jest wywołaniem informującym użytkownika o podłączeniu do konkretnego urządzenia. O tym fakcie użytkownik jest informowany za pomocą małego okna dialogowego.
- MESSAGE_TOAST: Ostatecznie natrafiamy na ogólny kod wysyłający do użytkownika komunikat z wnętrza klasy GameView.

Zarządzanie połączeniami Bluetooth

A teraz wrócimy na bardziej znane terytorium i dodamy kilka zmian do pliku *GameView.java*. Pamiętajmy, że większość umieszczonego w tym podrozdziale kodu jest niezbędna, ponieważ to w tej klasie będziemy mogli zmienić położenie sprajtów na podstawie danych przesyłanych tam i z powrotem pomiędzy tabletami.

Na listingach 7.6, 7.7 oraz 7.8 przedstawiony jest kod trzech miniwątków, który musi zostać dodany do klasy GameView w celu zapewnienia obsługi różnych operacji Bluetooth zachodzących w trakcie interakcji pomiędzy dwoma graczami. Należą do nich będą wątki AcceptThread, ConnectThread oraz ConnectedThread. Wątek AcceptThread obsługuje początkowe połączenie, ConnectThread obsługuje niuanse związane z parowaniem urządzeń, a ConnectedThread jest zwykłą procedurą obsługiwaną w chwili połączenia urządzeń ze sobą.

Listing 7.6. Wątek AcceptThread

```
private class AcceptThread extends Thread {
    // Gniazdo serwera lokalnego
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        BluetoothServerSocket tmp = null;

        // Utwórz nowe nasłuchujące gniazdo serwera
        try {
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "błąd funkcji listen()", e);
        }
        mmServerSocket = tmp;
    }

    public void run() {
        if (D) Log.d(TAG, "START mAcceptThread" + this);
        setName("AcceptThread");
        BluetoothSocket socket = null;

        // Dopóki brak jest połączenia, nasłuchuj na gnieździe sieciowym
        while (mState != STATE_CONNECTED) {
            try {
                // To jest wywołanie blokujące, które zwróci wartość jedynie w przypadku
                // nawiązania poprawnego połączenia lub wystąpienia wyjątku
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                Log.e(TAG, "błąd funkcji accept()", e);
                break;
            }

            // Po zaakceptowaniu połączenia
            if (socket != null) {
```

```

synchronized (BluetoothChatService.this) {
    switch (mState) {
        case STATE_LISTEN:
        case STATE_CONNECTING:
            // Sytuacja poprawna. Uruchom wątek połączenia
            connected(socket, socket.getRemoteDevice());
            break;
        case STATE_NONE:
        case STATE_CONNECTED:
            // Połączenie aktywne lub brak gotowości. Usuń nowe gniazdo
            try {
                socket.close();
            } catch (IOException e) {
                Log.e(TAG, "Błąd zamykania zbędnego gniazda ", e);
            }
            break;
    }
}
}
}
if (D) Log.i(TAG, "KONIEC mAcceptThread");
}

public void cancel() {
    if (D) Log.d(TAG, "anulowanie " + this);
    try {
        mmServerSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "błąd funkcji close()", e);
    }
}
}
}

```

Klasa `AcceptThread` jest złożonym fragmentem kodu, jednak w rzeczywistości oczekuje ona jedynie na zaakceptowanie połączenia. Zwróćmy uwagę na to, jak często pojawia się słowo kluczowe `socket` (gniazdo). Gniazda są standardowym sposobem nawiązywania połączeń pomiędzy urządzeniami i umożliwiają przesyłanie informacji. Ten kod został zaczerpnięty z jednego z przykładów z dokumentacji systemu Android. Kilkanaście zawartych w nich metod oraz bloków kodu jest, jak się okazało, niesamowicie wydajnie napisanych i nie wymagało żadnych poprawek.

Listing 7.7. Wątek `ConnectThread`

```

private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        mmDevice = device;
        BluetoothSocket tmp = null;

        // Pobierz gniazdo Bluetooth dla połączenia
        // z danym urządzeniem Bluetooth
        try {
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "błąd funkcji create()", e);
        }
        mmSocket = tmp;
    }
}

```

```

}

public void run() {
    Log.i(TAG, "START mConnectThread");
    setName("ConnectThread");

    // Należy zawsze unikać fazy discovery, ponieważ zdecydowanie spowalnia ona połączenie
    mAdapter.cancelDiscovery();

    // Nawiąż połączenie z gniazdem Bluetooth
    try {
        // To jest wywołanie blokujące, które zwróci wartość jedynie w przypadku
        // nawiązania poprawnego połączenia lub wystąpienia wyjątku
        mmSocket.connect();
    } catch (IOException e) {
        connectionFailed();
        // Zamknij gniazdo
        try {
            mmSocket.close();
        } catch (IOException e2) {
            Log.e(TAG, "błąd zamykania gniazda spowodowany błędem połączenia", e2);
        }
        // Ponownie uruchom usługę w trybie nasłuchu
        GameView.this.start();
        return;
    }

    // Wyzeruj zmienną mConnectThread
    synchronized (BluetoothChatService.this) {
        mConnectThread = null;
    }

    // Uruchom wątek połączenia
    connected(mmSocket, mmDevice);
}

public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "błąd funkcji close()", e);
    }
}
}

```

Pod względem obsługi próby połączenia z innym urządzeniem ten wątek przypomina poprzedni.

Ten fragment kodu także zawarty był w przykładach Androida, dlatego został on pozostawiony bez zmian. Jeżeli takie szczegóły interesują Czytelnika, na początku kod podejmuje jedną próbę nawiązania połączenia z innym urządzeniem. W przypadku błędu kolejne próby mogą być kontynuowane w bloku try, w którym wystąpienie błędu spowoduje powtórne przejście w tryb nasłuchu.

Na całe szczęście jesteśmy jedynie zainteresowani wysyłaniem danych tam i z powrotem i nie musimy zmieniać sposobu nawiązywania połączenia.

Klasa ConnectedThread wykonuje nadzwyczajną ilość pracy. Ten kod uruchamiany jest zawsze wtedy, gdy urządzenia są w stanie połączenia. Zwróćmy uwagę, że na początku pobierane są strumienie wejściowy oraz wyjściowy, aby możliwe było odbieranie danych z innego urządzenia lub wysyłanie własnych informacji.

Listing 7.8. Wątek *ConnectedThread*

```

private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        Log.d(TAG, "utwórz ConnectedThread");
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Pobierz strumienie wejściowe i wyjściowe gniazda
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "błąd tworzenia gniazd tymczasowych", e);
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        Log.i(TAG, "START mConnectedThread");
        byte[] buffer = new byte[1024];
        int bytes;

        // Dopóki jest połączenie, nasłuchuj strumień wejściowy
        while (true) {
            try {
                // Czytaj ze strumienia wejściowego
                bytes = mmInStream.read(buffer);

                // Wyślij otrzymane bajty do głównej aktywności
                mHandler.obtainMessage(MainActivity.MESSAGE_READ, bytes, -1, buffer)
                    .sendToTarget();
            } catch (IOException e) {
                Log.e(TAG, "rozłączony", e);
                connectionLost();
                break;
            }
        }
    }

    /**
     * Zapisz do podłączonego strumienia OutStream.
     * @param buffer Bajty do zapisu
     */
    public void write(byte[] buffer) {
        try {
            mmOutStream.write(buffer);

            // Udostępnij wysłany komunikat głównej aktywności
            mHandler.obtainMessage(MainActivity.MESSAGE_WRITE, -1, -1, buffer)

```

```

        .sendToTarget();
    } catch (IOException e) {
        Log.e(TAG, "Wyjątek podczas zapisu ", e);
    }
}

public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "błąd funkcji close()", e);
    }
}
}

```

W dalszej kolejności w metodzie `run()` umieszczona jest pętla, w której następuje stałe sprawdzanie obecności nowych danych do przetworzenia. Większość danych przesyłana jest w postaci liczb całkowitych, jednak użycie łańcuchów znakowych w charakterze łącznika pomiędzy urządzeniami ma też pewne zalety. Po pierwsze, w złożonych grach może występować wiele liczb koniecznych do przesłania (np. poziom życia lub amunicji, położenie czy wyposażenie). Wysyłanie samych liczb nie jest zbyt zrozumiałe. Natomiast łańcuch w postaci "a:10" może zostać bardzo szybko odnaleziony i rozdzielony na wartość liczbową oraz jej opis przed znakiem dwukropka. Na tej podstawie można szybko podjąć decyzję, czy konieczna jest zmiana.

Za pętlą występuje metoda wysyłająca komunikat w buforze do innego urządzenia. Metoda jest łatwa do zrozumienia i wysyła komunikat w takiej postaci jak w buforze.

Przed dodaniem wątków konieczne jest jeszcze zdefiniowanie pewnych metod używanych do wysyłania danych i wywoływania tych wątków podczas wykonywania określonych akcji. Pamiętajmy przy tym, że wątki nie zostały jeszcze w żaden sposób zainicjalizowane ani użyte. Wszystkie wymienione metody przedstawione są na listingu 7.9.

Listing 7.9. Połączenie z urządzeniem Bluetooth

```

public synchronized void start() {
    if (D) Log.d(TAG, "start");

    // Usuń dowolny wątek próbujący nawiązać połączenie
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Usuń dowolny wątek obsługujący połączenie
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Uruchom wątek nasłuchujący na gnieździe Bluetooth
    if (mAcceptThread == null) {
        mAcceptThread = new AcceptThread();
        mAcceptThread.start();
    }
    setState(STATE_LISTEN);
}

public synchronized void connect(BluetoothDevice device) {
    if (D) Log.d(TAG, "połącz z: " + device);

    // Usuń dowolny wątek próbujący nawiązać połączenie
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }
}

```

```

// Usuń dowolny wątek obsługujący połączenie
if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

// Uruchom wątek w celu nawiązania połączenia z danym urządzeniem
mConnectThread = new ConnectThread(device);
mConnectThread.start();
setState(STATE_CONNECTING);
}

public synchronized void connected(BluetoothSocket socket, BluetoothDevice device) {
    if (D) Log.d(TAG, "połączony");

    // Usuń wątek, który ukończył połączenie
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Usuń dowolny wątek obsługujący połączenie
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}

    // Usuń wątek AcceptThread, ponieważ chcemy połączyć się tylko z jednym urządzeniem
    if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}

    // Uruchom wątek w celu zarządzania połączeniem i transmisją
    mConnectedThread = new ConnectedThread(socket);
    mConnectedThread.start();

    Message msg = mHandler.obtainMessage(MainActivity.MESSAGE_DEVICE_NAME);
    Bundle bundle = new Bundle();
    bundle.putString(BluetoothChat.DEVICE_NAME, device.getName());
    msg.setData(bundle);
    mHandler.sendMessage(msg);

    setState(STATE_CONNECTED);
}

public synchronized void stop() {
    if (D) Log.d(TAG, "stop");
    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    if (mConnectedThread != null) {mConnectedThread.cancel(); mConnectedThread = null;}
    if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread = null;}
    setState(STATE_NONE);
}

public void write(byte[] out) {
    // Utwórz obiekt tymczasowy
    ConnectedThread r;
    // Synchronizuj kopię wątku ConnectedThread
    synchronized (this) {
        if (mState != STATE_CONNECTED) return;
        r = mConnectedThread;
    }
    // Wykonaj niesynchronizowany zapis
    r.write(out);
}

private void connectionFailed() {
    setState(STATE_LISTEN);
}

```

```

// Wyślij komunikat o błędzie z powrotem do aktywności
Message msg = mHandler.obtainMessage(MainActivity.MESSAGE_TOAST);
Bundle bundle = new Bundle();
bundle.putString(BluetoothChat.TOAST, "Błąd połączenia z urządzeniem");
msg.setData(bundle);
mHandler.sendMessage(msg);
}

private void connectionLost() {
    setState(STATE_LISTEN);

    // Wyślij komunikat o błędzie z powrotem do aktywności
    Message msg = mHandler.obtainMessage(MainActivity.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(MainActivity.TOAST, "Utracono połączenie z urządzeniem");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
}

```

Ponieważ wcześniej widzieliśmy już kod wątków, łatwo zrozumiemy, że głównym zadaniem tych funkcji jest uruchomienie wątków. Pierwsze trzy funkcje uruchamiają trzy wątki (AcceptThread, ConnectThread oraz ConnectedThread). Kiedy gra dobiega końca (to znaczy postać umiera), wywoływana jest funkcja stop(), zapewniająca, że żaden z wątków nie będzie działał nieskończenie. W przypadku gdybyśmy chcieli wysłać coś do innego urządzenia, użyjemy metody write().

Na koniec dwie inne metody wykorzystują uchwyt do wyświetlania komunikatów w chwili utraty połączenia lub wystąpienia błędu.

Modyfikacja kodu gry dla dwóch graczy

Dysponujemy już większością kodu do nawiązywania oraz utrzymywania połączenia. Teraz musimy zastanowić się, w jaki sposób gra będzie współpracować z interfejsem Bluetooth. Cały kod dla tej prostej gry okazał się zbyt duży, aby zmieścić się na stronach tej książki. Można go jednak pobrać ze strony: <http://code.google.com/p/android-tablet-games/>. Cały dodatkowy kod pominięty w tym miejscu obsługuje wybór urządzenia do podłączenia (co nie jest istotne w tym momencie).

Kontynuując już bez zbędnej zwłoki nasz opis, podczas gry chcielibyśmy mieć na ekranie dwie rakiетки: jedną na górze ekranu i jedną na dole. Cały ważny kod z metody update() w klasie GameView umieszczony jest na listingu 7.10. Zauważmy, że w istniejących funkcjach konieczne jest zainicjalizowanie sprajta o nazwie paddle_other, a także dodanie go do funkcji draw(). Sprajt ten zostanie umieszczony u góry ekranu i będzie używać tego samego obrazka co druga rakiетка.

Listing 7.10. Dodawanie rakiетки, wykrywanie kolizji oraz uaktualnianie stanu gry

```

//dane wejściowe dla rakiетки
int val=0;
for (int i=latest_input.length-1, j = 0; i >= 0; i--,j++)
    {
        val += (latest_input[i] & 0xff) << (8*j);
    }
paddle_other.setX(val);

//obsługa kolizji obiektu paddle_other
int paddle_other_bottom = paddle_other.getBitmap().getHeight();
if(paddle_other.collide(ball)){
    if(ball_y < paddle_other_bottom && ball_y < paddle_other_bottom + 20){
        ball.setMoveY(-ball.getMoveY());
    }
}

```

```

    }
}

//dane wyjściowe dla rakiетки
byte[] paddle_output;
ByteBuffer bb = ByteBuffer.allocate(4);
bb.putInt((int)paddle.getX());
paddle_output = bb.array();
write(paddle_output);

```

Kod umieszczony na listingu 7.10 wykonuje trzy rzeczy. Najpierw na podstawie informacji wejściowych uzyskanych z urządzenia kontrolującego obiekt `paddle_other` przesuwa ten obiekt do położenia docelowego. Po drugie, kod ten wykrywa kolizje. Po trzecie, kod wysyła dane o położeniu rakiетки kontrolowanej przez to urządzenie do drugiego urządzenia, tak aby przeciwnik mógł zobaczyć ostatni ruch gracza.

Wchodząc odrobinę w szczegóły, pętla `for` konwertuje tablicę bajtów otrzymaną jako dane wejściowe na wartości całkowite, używane później do przesuwania rakiетки. Nie jest na szczęście konieczna konwersja tablicy `byte[]` na bardziej złożone wartości. Wykrywanie kolizji przypomina kod wykorzystywany dla drugiej rakiетки, jednak kierunki są odwrócone, ponieważ w tym przypadku interesuje nas jedynie wykrywanie kolizji z dołem rakiетки, a nie z jej górą. Jeżeli Czytelnik chce, może dodać kod powodujący zerowanie lub zakończenie gry w przypadku, gdy piłka dotknie góry drugiej paletki, aby w ten sposób ukarać też drugiego gracza.

Na koniec aktualne położenie rakiетки jest konwertowane na tablicę bajtów i wysyłane do funkcji `write()`, która z kolei wysyła je do wątku `connectedThread` obsługującego transmisję danych.

Testowanie gry

Testowanie gry wieloosobowej używającej połączenia Bluetooth może być skomplikowane. Jeżeli Czytelnik dysponuje dwoma tabletami z systemem Android, może ustanowić między nimi połączenie. Następnie może wczytać program do obu urządzeń. Jeżeli jednak nie ma dwóch urządzeń lub gra ma korzystać z większej liczby tabletów, wtedy należy podejść do testowania w inny sposób. Oczywiście brakujący tablet można również spróbować pożyczyć od kogoś innego. Zwróćmy uwagę, że w celu zainstalowania oprogramowania na innym tablecie należy najpierw wykonać czynności opisane w dodatku A. Zanim Czytelnik zacznie eksperymentować z czymś urządzeniem, powinien również upewnić się, że jego właściciel będzie o tym poinformowany.

Kuszące może wydawać się podłączenie interfejsu Bluetooth do komputera i oczekiwanie poprawnej obsługi w emulatorze. Niestety w obecnej chwili emulator nie ma możliwości obsługi interfejsu Bluetooth. Do czasu dodania tej funkcji do emulatora w celu przetestowania aplikacji konieczne będzie zatem użycie prawdziwych urządzeń⁴.

⁴ W rzeczywistości do chwili obecnej emulator Androida pozbawiony jest opcji emulacji interfejsu Bluetooth, co jest jednym z bardziej frustrujących braków. Istnieje jednak kilka sposobów na obejście tego ograniczenia. Jednym z nich jest zainstalowanie na komputerze PC portu systemu Android dla systemów x86. Taki obraz, dostępny na przykład pod adresem: <http://www.android-x86.org/>, należy zainstalować w środowisku wirtualnym (np. VirtualBox), a następnie po uruchomieniu przekierować obsługę interfejsu Bluetooth do zvirtualizowanego Androida. Taki system powinien też być widoczny w środowisku Eclipse. Ponieważ jednak ta metoda nie jest oficjalnie wspierana przez firmę Google, a także ma pewne dodatkowe ograniczenia, najbardziej zdeteminowanych Czytelników odsyłam do lektury zasobów dostępnych w internecie, np. <http://androiddevnotes.com/2011/03/08/1299521520000.html> — *przyyp. tłum.*

Podsumowanie

Czytelnikowi ponownie należą się gratulacje. W tym rozdziale przeanalizowaliśmy pewne interesujące aspekty wykorzystania interfejsu Bluetooth do tworzenia gier wieloosobowych. To zagadnienie będzie z pewnością jednym z najbardziej skomplikowanych, na jakie Czytelnik może się natknąć podczas programowania gier. W tym momencie jesteśmy już gotowi do stworzenia dużej gry, co uczynimy pod koniec tej książki. Przygotujmy się na więcej sprajtów i dźwięków, a także na znacznie większą ilość kodu.

Skorowidz

A

- akcelerometr, 56
- algorytmy wykrywania kolizji, 119
- analiza gry, 168
- Android, 11
- Android 3.0, 13
- AVD, Android Virtual Device, 31

B

- barometr, 56
- błąd w programie, 153

C

- cechy systemu Android 3.0, 13
- czujnik
 - oświetlenia, 56
 - zbliżeniowy, 56

D

- dane z akcelerometru, 72
- dodawanie
 - dział, 151
 - dźwięku, 78, 122
 - ekranu początkowego, 171
 - grafiki do sprajtów, 100
 - łodzi, 163
 - muzyki, 85
 - obrazów, 151
 - połączeń Bluetooth, 130
 - rakietek, 140
 - sekwencji filmowych, 86
 - sprajtów, 99

E

- edytor
 - graficzny, 34
 - typu WYSIWYG, 34
- efekty dźwiękowe, 84
- ekrany
 - opornościowe, 55
 - pojemnościowe, 55
- elementy
 - falochronu, 146
 - sterujące, 163
- emulator Androida, 141

F

- FLAC, Free Lossless Audio Codec, 78
- funkcja
 - collide(), 125
 - draw(), 49, 125
 - getOrientation(), 75
 - getState(), 102
 - onCreate(), 131, 164
 - onDraw(), 50, 116, 162
 - playsound(), 84
 - setState(), 102
 - surfaceCreated(), 116
 - update(), 49, 99, 119, 123, 157, 160
- funkcje
 - w klasie SpriteObject, 156
 - wykrywania kolizji, 98
- funkcjonalności dodane do gry, 155

G

- generator liczb pseudolosowych, 149
- generowanie klucza, 177
- GPS, 56

gra

- AllTogether, 95
- Breakout, 109
- Harbor Defender, 144
- Pong, 112

grafika rastrowa, 42

gry

- dwuosobowe, 130
- jednoosobowe, 95
- strategiczne, 143
- wielosobowe, 128

I

IDE, Integrated Development Environment, 16

ikonki dział, 158

implementacja zarządzania czasem, 52

informacje o programiście, 177

inicjalizacja

- bloków, 123
- obiektów czujników, 71
- obiektów w projekcie, 147

instalacja

- pakietu Java JDK, 16
- pakietu SDK dla systemu Android, 20
- sterowników, 184
- środowiska Eclipse, 17
- klasy GameView, 73

integracja Eclipse z SDK, 24

interfejs

- sprzętowy, 183
- wielodotkowy, multitouch, 55

J

JDK, Java Development Kit, 16

język

- Java, 14
- XML, 15

K

katalog

- Res, 33
- values, 33

klasa

- Asteroid, 89
- AudioManager, 83
- Explosion, 89
- GameLogic, 46, 49
- GameView, 43, 80, 105, 115, 133
- Gesture, 61
- InputObject, 65

JetBoyView, 89

JetPlayer, 87, 89

Main, 30

MainActivity, 174

MediaPlayer, 79

MotionEvent, 59

SpriteObject, 48, 53, 103

View, 40

kod wykrywający kolizje, 53

kolejki wejścia, 64

kolizje pomiędzy sprajtami, 97

komentarz z nagłówkiem, 51

komunikat o błędzie, 153

komunikaty, 133

konfiguracja

- emulatora, 62
- ikon, 158
- łodzi, 163
- narzędzi Androida, 23
- środowiska programistycznego, 16

konstruktor klasy GameView, 115

kontrola ruchu, 97

M

MESSAGE_DEVICE_NAME, 134

MESSAGE_STATE_CHANGE, 133

MESSAGE_TOAST, 134

metoda

- createBullet(), 167
- draw(), 49, 125
- initializeJetPlayer, 89
- load(), 83
- onActivityResult(), 131
- onCreate(), 131, 164
- onDraw, 50, 116, 162
- onPause(), 71
- onResume(), 71
- onSensorChanged, 72
- onStart(), 131
- onTouchEvent, 67
- processMotionEvent(), 100
- queueJetSegment(), 92
- reset(), 167
- startgame(), 132
- update(), 49, 99, 119, 123, 157, 160
- updateGameState(), 93

metody

- do obsługi czujników, 71
- klasy GameLogic, 47
- obsługujące kolizję, 54
- tworzące pule obiektów, 67

mikrofon, 56

modyfikacja
 GameView.java, 114
 SpriteObject.java, 114
 muzyka sterowana zdarzeniami, 89

N

nagroda dla gracza, 101

O

obiekt
 BlueAdapter, 131
 Canvas, 42
 inputObjectPool, 66
 obiekty klasy SpriteObject, 158
 obliczanie momentu wystrzelenia kul, 166
 obsługa
 dotyku, 121
 dźwięku, 121
 komunikatów, 132
 kul, 164
 muzyki, 87
 nagród, 121
 poleceń użytkownika, 160
 zdarzeń, 117
 zmian w kulach, 165
 odtwarzanie dźwięków, 78–80
 okno tworzenia projektu, 39
 opakowywanie gry, 175
 osie współrzędnych, 75

P

P2P, peer-to-peer, 128
 pakiet
 JDK, 16
 SDK, 16
 parametry queueJetSegment(), 92
 pętla gry, 58
 pętla gry programu JetBoy, 92
 plik
 AndroidManifest.xml, 175, 183
 GameLogic.java, 46
 GameView.java, 43, 45, 80, 105
 InputObject.java, 65
 JetBoy.zip, 89
 Level1.jtc, 89
 Main.java, 30
 main.xml, 34, 172, 174
 MainActivity.java, 174
 SpriteObject.java, 48, 103
 strings.xml, 33

pliki
 FLAC, 78
 JET, 87
 MIDI, 78
 mp3, 78
 XML, 123
 płótno, 42
 pobieranie danych, 55
 połączenia równorzędne, P2P network, 128
 połączenie z urządzeniem Bluetooth, 134, 138
 proces gry, 57
 program
 Eclipse, 16
 FirstApp, 33
 GIMP, 112
 JET Creator, 87
 LogCat, 153
 projekt
 programu w Eclipse, 29
 TabletPaddle, 113
 TwoPlayerPaddleGame, 130
 przetwarzanie
 danych, 60
 danych wejściowych, 68, 69
 danych z sensora, 73
 poleceń, 59
 zdarzeń, 68
 publikacja gry, 171

R

reagowanie na
 dane z czujników, 70
 dotyk, 59
 gesty, 61
 wciśnięcie przycisku, 174
 RECEIVE_DATA, 133
 reklamowanie gry, 180
 rozmieszczanie elementów, 162
 rozpowszechnianie gry, 176
 rozszerzenie sprajтів, 156
 rysowanie
 dział, 151
 falochronu, 148
 łodzi, 149
 obrazu, 42
 zamku oraz gruntu, 149

S

sekwencje filmowe, 77
 SEND_DATA, 133
 serwer gier, 128

sieć

- P2P, 128
- torrent, 128

sklep

- Android Market, 15
- App Store firmy Apple, 178
- Google Play, 178

sprajt, sprite, 37

sprajt star.png, 61

sprawdzanie zderzenia, 164

stała liczba klatek animacji, 52

stan początkowy, 103

stany sprajtów, 101, 102

sterowanie grą, 157

sterowanie rakietską, 121

sterowniki USB, 184

strzelanie z dział, 164

Ś

śledzenie stanu sprajtów, 101

środowisko gry, 114

środowisko programistyczne, 15

T

testowanie

- gry, 141, 152
- gry na fizycznym urządzeniu, 183
- narzędzi programistycznych, 27
- programów, 15
- projektu JetBoy, 90, 91

tworzenie

- falochronu, 145, 147
- gestu, 63
- gruntu oraz zamku, 148
- gry jednoosobowej, 96
- konta w Google Play, 179
- łodzi, 149
- obiektów nowych klas, 50
- obiektów wejściowych, 67
- pętli gry, 46
- projektu, 27
- projektu TabletPaddle, 113
- sprajta, 48
- wirtualnego urządzenia z Androidem, 31

U

uaktualnianie stanu gry, 140

ukrywanie paska zadań, 51

uruchamianie

- aplikacji, 33
- emulatora, 63
- gry, 49

USB Debugging, 184

usługa Google Play, 179

usuwanie bloków, 125

uwolnienie sprajta, 74

W

warstwy obrazu, 148

wątek, 46

AcceptThread, 134

ConnectedThread, 137

ConnectThread, 135

wielowątkowość, 46

wprowadzenie do gry, 172

wykrywanie kolizji, 53, 98, 117, 119, 140

wysyłanie aplikacji, 180

wysyłanie danych czujnika, 73

wyświetlanie

- grafiki, 42
- obrazów, 38
- sprajtów, 45

Z

zachowywanie wskazań użytkownika, 159

zarządzanie sprajtem, 60

zasoby, resources, 33

zdarzenia, 133

zdarzenia wejściowe, input events, 57

zdarzenie dotknięcia ekranu, 159

zerowanie stanu gry, 102, 164

zmiana

- kierunku poruszania, 150
- liczby dział, 151

zmiennie

- przechowujące wybór użytkownika, 159
- w klasie SpriteObject, 156

Ż

żyroskop, 56

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Tablety z Androidem na pokładzie zdobywają coraz większą popularność. Jednym z powodów tego zjawiska jest niezwykle korzystny stosunek jakości i możliwości do ceny. Czy są jakieś inne? Oczywiście — ogromny wybór przydatnych aplikacji oraz atrakcyjnych gier. Duże ekrany i mocne, często wielordzeniowe procesory dają programistom pole do popisu. Jeżeli chcesz dołączyć do tego grona i przygotować genialną grę na platformę Android, potrzebujesz tylko kilku rzeczy: czasu, chęci oraz tej książki!

Dzięki niej poznasz wszystkie niuanse tworzenia gier na tablety pracujące pod kontrolą systemu Android. Na początku dowiesz się, jak przygotować środowisko pracy oraz jakich narzędzi i bibliotek będziesz potrzebować. Po krótkim wstępie przejdiesz do sedna sprawy. Nauczysz się wykorzystywać ekrany dotykowe, moc wielordzeniowych procesorów oraz informacje z czujników położenia. Ponadto poznasz tajniki tworzenia sztucznej inteligencji oraz mechanizmów dla wielu graczy. O sukcesie gry decyduje atrakcyjna oprawa dźwiękowa i wizualna, dlatego autor kładzie duży nacisk właśnie na te aspekty tworzenia gier. Na koniec sprawdzisz, jak opublikować grę w Google Play Store (dawniej Android Market). Jest to idealna pozycja dla każdego pasjonata systemu Android.

Sięgnij po tę książkę, a następnie:

- zaprojektuj własną grę dla Androida
- przygotuj grę dla jednego i wielu graczy
- wykorzystaj moc współczesnych procesorów
- opublikuj swoje dzieło w Google Play Store

Twórz świetne gry na platformę Android!

helion.pl
księgarnia
internetowa

Nr katalogowy: 11464



Księgarnia internetowa
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

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

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-5004-0



9 788324 650040

Cena: 39,00 zł

Informatyka w najlepszym wydaniu