

WYDANIE III

Android™ 6

dla programistów

Techniki tworzenia aplikacji



Helion 

PAUL DEITEL • HARVEY DEITEL
ALEXANDER WALD

Tytuł oryginalny: Android 6 for Programmers: An App-Driven Approach (3rd Edition)

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-2578-4

Authorized translation from the English language edition, entitled: ANDROID 6 FOR PROGRAMMERS: AN APP-DRIVEN APPROACH, Third Edition; ISBN 0134289366; by Paul Deitel; and Harvey Deitel; and Alexander Wald; published by Pearson Education, Inc, publishing as Prentice Hall. Copyright © 2016 Pearson Education, Inc.

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 Pearson Education, Inc.

Polish language edition published by HELION SA. Copyright © 2016.

DEITEL, the double-thumbs-up bug and DIVE-INTO are registered trademarks of Deitel & Associates, Inc. Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. Google, Android, Google Play, Google Maps, Google Wallet, Nexus, YouTube, AdSense and AdMob are trademarks of Google, Inc.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screenshots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screenshots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.

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.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą 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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/and6p3>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Przedmowa	17
Zanim zaczniesz	27
Rozdział 1. Android — wprowadzenie	33
1.1. Wstęp	34
1.2. Android — najpopularniejszy mobilny system operacyjny na świecie	34
1.3. Cechy systemu Android	35
1.4. System operacyjny Android	38
1.4.1. Android 2.2 (Froyo)	39
1.4.2. Android 2.3 (Gingerbread)	39
1.4.3. Android 3.0 – 3.2 (Honeycomb)	40
1.4.4. Android 4.0 – 4.0.4 (Ice Cream Sandwich)	40
1.4.5. Android 4.1 – 4.3 (Jelly Bean)	42
1.4.6. Android 4.4 (KitKat)	43
1.4.7. Android 5.0 i 5.1 (Lollipop)	43
1.4.8. Android 6 (Marshmallow)	45
1.5. Pobieranie aplikacji z serwisu Google Play	47
1.6. Pakiety	47
1.7. Zestaw narzędzi Android Software Development Kit (SDK)	50
1.8. Programowanie obiektowe — krótkie przypomnienie	53
1.8.1. Samochód jako obiekt	53
1.8.2. Metody i klasy	53
1.8.3. Konkretyzacja	53
1.8.4. Ponowne wykorzystanie	54
1.8.5. Komunikaty i wywołania metod	54
1.8.6. Atrybuty i zmienne egzemplarzowe	54
1.8.7. Hermetyzacja	54
1.8.8. Dziedziczenie	54
1.8.9. Analiza i projektowanie zorientowane obiektowo (OOAD)	55
1.9. Sprawdzanie działania aplikacji Tip Calculator za pomocą urządzenia wirtualnego z systemem Android (AVD)	55
1.9.1. Otwarcie projektu Tip Calculator w środowisku Android Studio	55
1.9.2. Tworzenie wirtualnych urządzeń systemu Android	57

1.9.3. Uruchamianie aplikacji Tip Calculator na emulowanym smartfonie Nexus 6	60
1.9.4. Uruchamianie aplikacji Tip Calculator na realnym urządzeniu wyposażonym w system Android	63
1.10. Tworzenie doskonałych aplikacji przeznaczonych dla systemu Android	64
1.11. Dokumentacja przydatna podczas pracy nad aplikacjami systemu Android	66
1.12. Podsumowanie	68

Rozdział 2. Aplikacja powitalna **69**

Rozpoczęcie pracy w środowisku Android Studio — wizualne projektowanie graficznego interfejsu użytkownika, szablony, dostępność i internacjonalizacja

2.1. Wstęp	70
2.2. Przegląd technologii	71
2.2.1. Android Studio	71
2.2.2. LinearLayout, TextView i ImageView	71
2.2.3. Rozszerzalny język znaczników (XML)	71
2.2.4. Zasoby aplikacji	71
2.2.5. Dostępność	72
2.2.6. Internacjonalizacja	72
2.3. Tworzenie aplikacji	72
2.3.1. Uruchamianie środowiska Android Studio	72
2.3.2. Tworzenie nowego projektu	72
2.3.3. Tworzenie nowego projektu w oknie Create New Project	72
2.3.4. Wybór docelowych urządzeń	74
2.3.5. Dodaj aktywność	75
2.3.6. Personalizacja aktywności	77
2.4. Okno Android Studio	77
2.4.1. Okno Project	78
2.4.2. Okna edytora	79
2.4.3. Okno Component Tree	80
2.4.4. Pliki z zasobami aplikacji	80
2.4.5. Edytor rozkładu	80
2.4.6. Domyślny graficzny interfejs użytkownika	81
2.4.7. Kod XML domyślnego interfejsu użytkownika	81
2.5. Tworzenie graficznego interfejsu użytkownika aplikacji za pomocą edytora rozkładu	83
2.5.1. Dodawanie obrazu	83
2.5.2. Dodawanie ikony aplikacji	85
2.5.3. Zmiana rozkładu RelativeLayout na LinearLayout	86
2.5.4. Zmiana identyfikatora i orientacji rozkładu LinearLayout	87
2.5.5. Konfigurowanie właściwości id i text pola TextView	88
2.5.6. Konfiguracja parametru textSize pola TextView — skalowane piksele (Scaled Pixels) i piksele niezależne od gęstości (density independent pixel)	90
2.5.7. Definiowanie parametru textColor pola TextView	91
2.5.8. Definiowanie parametru gravity pola TextView	92
2.5.9. Definiowanie parametru layout:gravity pola TextView	93
2.5.10. Definiowanie parametru layout:weight pola TextView	94
2.5.11. Dodawanie pola obrazu ImageView	95
2.5.12. Podgląd projektu	97

2.6. Uruchamianie aplikacji Welcome	98
2.7. Ułatwienie dostępu do aplikacji	100
2.8. Internacjonalizacja aplikacji	101
2.8.1. Lokalizacja	101
2.8.2. Nazywanie folderów zlokalizowanych zasobów	101
2.8.3. Dodawanie przetłumaczonych łańcuchów do projektu aplikacji	102
2.8.4. Lokalizacja łańcuchów	102
2.8.5. Testowanie hiszpańskiej wersji językowej na urządzeniu wirtualnym	103
2.8.6. Testowanie aplikacji w języku hiszpańskim na urządzeniu	104
2.8.7. TalkBack i lokalizacja	105
2.8.8. Rzeczy, o których należy pamiętać podczas lokalizacji	105
2.8.9. Profesjonalne tłumaczenie	105
2.9. Podsumowanie	106

Rozdział 3. Aplikacja obliczająca napiwek 107

Wprowadzenie elementów takich jak `GridLayout`, `EditText`, `SeekBar`, obsługa zdarzeń, `NumberFormat`, personalizacja motywu aplikacji i definiowanie funkcjonalności aplikacji za pomocą kodu Java

3.1. Wstęp	108
3.2. Testowanie działania aplikacji Tip Calculator	109
3.3. Zastosowane rozwiązania	110
3.3.1. Klasa <code>Activity</code>	110
3.3.2. Metody cyklu roboczego klasy <code>Activity</code>	110
3.3.3. Biblioteka <code>AppCompat</code> i klasa <code>AppCompatActivity</code>	111
3.3.4. <code>GridLayout</code> i organizowanie widoków	112
3.3.5. Tworzenie graficznego interfejsu użytkownika w oknach <code>Layout Editor</code> , Component Tree i Properties	112
3.3.6. Formatowanie liczb na wartości walutowe zależne od lokalizacji i na łańcuchy procentowe	113
3.3.7. Implementacja interfejsu <code>TextWatcher</code> obsługującego obiekt <code>EditText</code> i zmiany tekstu	113
3.3.8. Implementacja interfejsu <code>OnSeekBarChangeListener</code> do obsługi zmian położenia suwaka <code>SeekBar</code>	113
3.3.9. Motywy <code>Material Themes</code>	113
3.3.10. Wytyczne <code>material design</code> : parametr <code>elevation</code> i cienie	114
3.3.11. Wytyczne <code>material design</code> dotyczące kolorów	114
3.3.12. <code>AndroidManifest.xml</code>	115
3.3.13. Wyszukiwanie w oknie <code>Properties</code>	115
3.4. Budowa graficznego interfejsu użytkownika	115
3.4.1. <code>GridLayout</code> — wprowadzenie	115
3.4.2. Tworzenie projektu <code>TipCalculator</code>	117
3.4.3. Zmiana rozkładu na <code>GridLayout</code>	117
3.4.4. Dodawanie obiektów <code>TextViews</code> , <code>EditText</code> i <code>SeekBar</code>	117
3.4.5. Personalizacja widoków	120
3.5. Domyślny motyw i personalizacja kolorów motywu	123
3.5.1. Motyw <code>parent</code>	123
3.5.2. Personalizacja kolorów motywu	124
3.5.3. Wspólne właściwości widoku jako <code>style</code>	126

3.6. Tworzenie logiki aplikacji	127
3.6.1. Instrukcje package i import	128
3.6.2. MainActivity — podklasa AppCompatActivity	129
3.6.3. Zmienne klasy i zmienne obiektowe	129
3.6.4. Przedefiniowanie metody onCreate klasy Activity	130
3.6.5. Metoda calculate klasy MainActivity	133
3.6.6. Anonimowa klasa wewnętrzna implementująca interfejs OnSeekBarChangeListener	133
3.6.7. Anonimowa klasa wewnętrzna implementująca interfejs TextWatcher	134
3.7. AndroidManifest.xml	135
3.7.1. Element manifest	136
3.7.2. Element application	136
3.7.3. Element activity	137
3.7.4. Element intent-filter	137
3.8. Podsumowanie	139

Rozdział 4. Aplikacja Flag Quiz

141

Fragmenty, menu, preferencje, jawne intencje, obiekt handler, `AssetManager`, animacje, obiekty animujące, obiekty `Toast`, listy stanów kolorów, rozkłady obsługujące wiele orientacji urządzenia, tworzenie logów zawierających komunikaty przydatne podczas debugowania

4.1. Wstęp	142
4.2. Testowanie aplikacji Flag Quiz	143
4.2.1. Konfiguracja quizu	143
4.2.2. Rozwiązywanie quizu	145
4.3. Omówienie technologii	148
4.3.1. Menu	148
4.3.2. Fragmenty	148
4.3.3. Metody cyklu życiowego fragmentu	149
4.3.4. Zarządzanie fragmentami	149
4.3.5. Preferencje	150
4.3.6. Folder assets	150
4.3.7. Foldery zasobów	150
4.3.8. Obsługa ekranów o różnych rozmiarach i rozdzielczościach	151
4.3.9. Określanie orientacji urządzenia	152
4.3.10. Wyświetlanie komunikatów za pomocą obiektów Toast	152
4.3.11. Korzystanie z klasy Handler w celu wykonania w przyszłości obiektu Runnable	152
4.3.12. Obiekty View i animacje	153
4.3.13. Korzystanie z klasy ViewAnimationUtils w celu stworzenia animacji okręgu odsłaniającego kolejny ekran	153
4.3.14. Określanie kolorów na podstawie stanu obiektu View za pośrednictwem listy kolorów stanów	153
4.3.15. Okno AlertDialog	153
4.3.16. Zapisywanie komunikatów wyjątków w pliku dziennika	154
4.3.17. Uruchamianie kolejnego obiektu Activity za pomocą jawnego komunikatu Intent	154
4.3.18. Java — struktury danych	155
4.3.19. Funkcje środowiska Java SE 7	155
4.3.20. AndroidManifest.xml	156

4.4. Tworzenie projektu, plików zasobów i dodatkowych klas	156
4.4.1. Tworzenie projektu	156
4.4.2. Rozkłady szablonu Blank Activity	157
4.4.3. Konfiguracja obsługi środowiska Java SE 7	158
4.4.4. Dodawanie obrazów flag do projektu	158
4.4.5. Plik strings.xml i sformatowane łańcuchy	158
4.4.6. arrays.xml	159
4.4.7. colors.xml	161
4.4.8. button_text_color.xml	162
4.4.9. Edycja pliku menu_main.xml	162
4.4.10. Tworzenie animacji trzęsącej się flagi	163
4.4.11. Definiowanie preferencji aplikacji za pomocą pliku preferences.xml	164
4.4.12. Dodawanie do projektu klas SettingsActivity i SettingsActivityFragment	166
4.5. Tworzenie graficznego interfejsu użytkownika aplikacji	167
4.5.1. Rozkład activity_main.xml dla urządzeń w orientacji pionowej	167
4.5.2. Projektowanie rozkładu fragment_main.xml	167
4.5.3. Pasek narzędzi graficznego edytora rozkładu	172
4.5.4. Rozkład content_main.xml i pozioma orientacja tabletu	173
4.6. Klasa MainActivity	175
4.6.1. Instrukcje package i import	175
4.6.2. Pola	176
4.6.3. Przedefiniowana metoda onCreate obiektu Activity	176
4.6.4. Przedefiniowana metoda onStart obiektu Activity	178
4.6.5. Przedefiniowana metoda onCreateOptionsMenu obiektu Activity	179
4.6.6. Przedefiniowana metoda onOptionsItemSelected obiektu Activity	180
4.6.7. Anonimowa klasa wewnętrzna implementująca obiekt OnSharedPreferenceChangeListener	180
4.7. Klasa MainActivityFragment	182
4.7.1. Instrukcje package i import	182
4.7.2. Pola	183
4.7.3. Przedefiniowywanie metody onCreateView obiektu Fragment	184
4.7.4. Metoda updateGuessRows	186
4.7.5. Metoda updateRegions	187
4.7.6. Metoda resetQuiz	187
4.7.7. Metoda loadNextFlag	189
4.7.8. Metoda getCountryName	191
4.7.9. Metoda animate	191
4.7.10. Anonimowa klasa wewnętrzna implementująca obiekt OnClickListener	193
4.7.11. Metoda disableButtons	195
4.8. Klasa SettingsActivity	196
4.9. Klasa SettingsActivityFragment	197
4.10. AndroidManifest.xml	197
4.11. Podsumowanie	199

Rozdział 5. Aplikacja Doodlz201

Grafika 2D, klasa Canvas, bitmapy, przyspieszeniometer, menedżer SensorManager, zdarzenia wielodotkowe, magazyn MediaStore, drukowanie, uprawnienia platformy Android 6.0, system Gradle

5.1. Wstęp	202
5.2. Testowanie działania aplikacji Doodlz za pomocą maszyny wirtualnej AVD	203
5.3. Technologie zastosowane w aplikacji	208
5.3.1. Metody cyklu życiowego obiektów Activity i Fragment	208
5.3.2. Personalizacja obiektów View	209
5.3.3. Korzystanie z obiektu SensorManager w celu nasłuchiwanie zdarzeń przyspieszeniometera	209
5.3.4. Spersonalizowane obiekty DialogFragment	209
5.3.5. Rysowanie za pomocą klas Canvas i Paint, bitmapy	210
5.3.6. Przetwarzanie wielu zdarzeń dotknięcia ekranu i zapisywanie linii jako obiekty Path	210
5.3.7. Zapisywanie obrazów w pamięci urządzenia	211
5.3.8. Drukowanie i klasa PrintHelper biblioteki Android Support Library	211
5.3.9. Nowy model zezwoleń systemu Android 6.0 (Marshmallow)	211
5.3.10. Dodawanie zależności za pomocą systemu Gradle Build	211
5.4. Tworzenie projektu i zasobów	212
5.4.1. Tworzenie projektu	212
5.4.2. Gradle — dodawanie do projektu biblioteki pomocniczej	212
5.4.3. strings.xml	213
5.4.4. Importowanie ikon elementów znajdujących się w menu aplikacji	214
5.4.5. Menu MainActivityFragment	214
5.4.6. Dodawanie zezwoleń do pliku AndroidManifest.xml	216
5.5. Budowa graficznego interfejsu użytkownika aplikacji	217
5.5.1. Rozkład content_main.xml aktywności MainActivity	217
5.5.2. Rozkład fragment_main.xml fragmentu MainActivityFragment	217
5.5.3. Rozkład fragment_color.xml fragmentu ColorDialogFragment	218
5.5.4. Rozkład fragment_line_width.xml fragmentu LineWidthDialogFragment	219
5.5.5. Dodawanie klasy EraseImageDialogFragment	223
5.6. Klasa MainActivity	223
5.7. Klasa MainActivityFragment	224
5.7.1. Pola i instrukcje package i import	224
5.7.2. Przedefiniowana metoda onCreateView obiektu Fragment	226
5.7.3. Metody onResume i enableAccelerometerListening	226
5.7.4. Metody onPause i disableAccelerometerListening	227
5.7.5. Anonimowa klasa wewnętrzna przetwarzająca zdarzenia przyspieszeniometera	228
5.7.6. Metoda confirmErase	229
5.7.7. Przedefiniowane metody obiektu Fragment: onCreateOptionsMenu i onOptionsItemSelected ...	230
5.7.8. Metoda saveImage	231
5.7.9. Przedefiniowana metoda onRequestPermissionsResult	233
5.7.10. Metody getDoodleView i setDialogOnScreen	233
5.8. Klasa DoodleView	234
5.8.1. Instrukcja package i instrukcje import	234
5.8.2. Zmienne statyczne i egzemplarzowe	235
5.8.3. Konstruktor	235
5.8.4. Przedefiniowana metoda onSizeChanged klasy View	236

5.8.5. Metody clear, setDrawingColor, getDrawingColor, setLineWidth i getLineWidth	237
5.8.6. Przewidywana metoda onDraw klasy View	238
5.8.7. Przewidywana metoda onTouchEvent klasy View	238
5.8.8. Metoda touchStarted	239
5.8.9. Metoda touchMoved	240
5.8.10. Metoda touchEnded	241
5.8.11. Metoda saveImage	242
5.8.12. Metoda printImage	243
5.9. Klasa ColorDialogFragment	243
5.9.1. Przewidywana metoda onCreateDialog klasy DialogFragment	244
5.9.2. Metoda getDoodleFragment	245
5.9.3. Przewidywane metody cyklu życia obiektu Fragment — onAttach i onDetach	246
5.9.4. Anonimowa klasa wewnętrzna, która reaguje na zdarzenia czterech suwaków SeekBar definiujących barwę	246
5.10. Klasa LineWidthDialogFragment	247
5.10.1. Metoda onCreateDialog	250
5.10.2. Anonimowa klasa wewnętrzna reagująca na zdarzenia paska widthSeekBar	250
5.11. Klasa EraseImageDialogFragment	250
5.12. Podsumowanie	252

Rozdział 6. Aplikacja Cannon Game 253

Ręczna animacja klatka po klatce, grafika, dźwięk, tworzenie wielu wątków, obiekty SurfaceView i SurfaceHolder, tryb pełnoekranowy

6.1. Wstęp	254
6.2. Testowanie aplikacji Cannon Game	255
6.3. Zastosowane rozwiązania	255
6.3.1. Korzystanie z folderu zasobów res/raw	255
6.3.2. Metody cyklu życia klas Activity i Fragment	255
6.3.3. Przewidywanie metody onTouchEvent klasy View	256
6.3.4. Dodawanie dźwięku za pomocą narzędzi SoundPool i AudioManager	256
6.3.5. Tworzenie kolejnych klatek animacji za pomocą wątków i obiektów SurfaceView i SurfaceHolder	256
6.3.6. Proste wykrywanie zderzeń	257
6.3.7. Tryb pełnoekranowy	257
6.4. Tworzenie graficznego interfejsu użytkownika i plików zasobów	257
6.4.1. Tworzenie projektu	257
6.4.2. Modyfikacja motywu umożliwiaująca usunięcie nazwy aplikacji i jej paska	258
6.4.3. strings.xml	258
6.4.4. Kolory	259
6.4.5. Dodawanie dźwięku	259
6.4.6. Dodawanie klasy MainActivityFragment	259
6.4.7. Edycja pliku activity_main.xml	259
6.4.8. Dodawanie obiektu CannonView do fragmentu fragment_main.xml	260
6.5. Przegląd klas aplikacji	260
6.6. MainActivity — podklasa klasy Activity	261
6.7. MainActivityFragment — podklasa klasy Fragment	262

6.8. Klasa <code>GameElement</code>	263
6.8.1. Zmienne egzemplarzowe i konstruktor	264
6.8.2. Metody <code>update</code> , <code>draw</code> i <code>playSound</code>	265
6.9. <code>Blocker</code> — podklasa klasy <code>GameElement</code>	265
6.10. <code>Target</code> — podklasa klasy <code>GameElement</code>	266
6.11. Klasa <code>Cannon</code>	266
6.11.1. Zmienne egzemplarzowe i konstruktor	266
6.11.2. Metoda <code>align</code>	267
6.11.3. Metoda <code>fireCannonball</code>	268
6.11.4. Metoda <code>draw</code>	268
6.11.5. Metody <code>getCannonball</code> i <code>removeCannonball</code>	269
6.12. <code>Cannonball</code> — podklasa klasy <code>GameElement</code>	270
6.12.1. Zmienne egzemplarzowe i konstruktor	270
6.12.2. Metody <code>getRadius</code> , <code>collidesWith</code> , <code>isOnScreen</code> i <code>reverseVelocityX</code>	270
6.12.3. Metoda <code>update</code>	271
6.12.4. Metoda <code>draw</code>	272
6.13. Klasa <code>CannonView</code> — podklasa <code>SurfaceView</code>	272
6.13.1. Instrukcje <code>package</code> i <code>import</code>	272
6.13.2. Stałe i zmienne egzemplarzowe	273
6.13.3. Konstruktor	274
6.13.4. Przedefiniowywanie metody <code>onSizeChanged</code> klasy <code>View</code>	276
6.13.5. Metody <code>getScreenWidth</code> , <code>getScreenHeight</code> i <code>playSound</code>	277
6.13.6. Metoda <code>newGame</code>	277
6.13.7. Metoda <code>updatePositions</code>	279
6.13.8. Metoda <code>alignAndFireCannonball</code>	280
6.13.9. Metoda <code>showGameOverDialog</code>	281
6.13.10. Metoda <code>drawGameElements</code>	282
6.13.11. Metoda <code>testForCollisions</code>	283
6.13.12. Metody <code>stopGame</code> i <code>releaseResources</code>	285
6.13.13. Implementacja metod interfejsu <code>SurfaceHolder.Callback</code>	285
6.13.14. Przedefiniowywanie metody <code>onTouchEvent</code> klasy <code>View</code>	286
6.13.15. Korzystanie z wątku <code>CannonThread</code> w celu utworzenia pętli gry	287
6.13.16. Metody <code>hideSystemBars</code> i <code>showSystemBars</code>	288
6.14. Podsumowanie	289

Rozdział 7. Aplikacja `WeatherViewer` 291

Usługi sieciowe REST, zdarzenie asynchroniczne `AsyncTask`, połączenie `URLConnection`, przetwarzanie danych JSON, `JSONObject`, `JSONArray`, `ListView`, `ArrayAdapter`, usługa `ViewHolder`

7.1. Wstęp	292
7.2. Testowanie aplikacji <code>WeatherViewer</code>	293
7.3. Zastosowane rozwiązania	294
7.3.1. Usługi sieciowe	294
7.3.2. JavaScript Object Notation (JSON) i pakiet <code>org.json</code>	295
7.3.3. Połączenie <code>URLConnection</code> i wywoływanie usługi sieciowej REST	297
7.3.4. Korzystanie z klasy <code>AsyncTask</code> w celu wygenerowania żądania sieciowego poza wątkiem graficznego interfejsu użytkownika	297

7.3.5. ListView, ArrayAdapter i wzorec ViewHolder	298
7.3.6. Przycisk FloatingActionButton	299
7.3.7. TextInputLayout	299
7.3.8. Snackbar	300
7.4. Tworzenie graficznego interfejsu użytkownika aplikacji oraz plików zasobów	300
7.4.1. Tworzenie projektu	300
7.4.2. AndroidManifest.xml	300
7.4.3. strings.xml	301
7.4.4. colors.xml	301
7.4.5. activity_main.xml	301
7.4.6. content_main.xml	302
7.4.7. list_item.xml	302
7.5. Klasa Weather	304
7.5.1. Zmienne egzemplarzowe oraz instrukcje import i package	305
7.5.2. Konstruktor	305
7.5.3. Metoda convertTimeStampToDay	306
7.6. Klasa WeatherArrayAdapter	307
7.6.1. Instrukcja package i instrukcje import	307
7.6.2. Zagnieżdżona klasa ViewHolder	308
7.6.3. Zmienne egzemplarzowe i konstruktor	308
7.6.4. Przekonfigurowana metoda getView adaptera ArrayAdapter	309
7.6.5. Podklasa AsyncTask służąca do pobierania obrazów w oddzielnym wątku	311
7.7. Klasa MainActivity	312
7.7.1. Instrukcja package i instrukcje import	312
7.7.2. Zmienne egzemplarzowe	313
7.7.3. Przekonfigurowana metoda onCreate klasy Activity	314
7.7.4. Metody dismissKeyboard i createURL	315
7.7.5. Podklasa AsyncTask służąca do wywoływania usługi sieciowej	316
7.7.6. Metoda convertJSONtoArrayList	318
7.8. Podsumowanie	319
Rozdział 8. Aplikacja Twitter Searches	321
Klasa SharedPreferences, podklasa SharedPreferences.Editor, niejawne intencje, wybór aktywności obsługującej intencję, klasa RecyclerView, podklasy RecyclerView.Adapter, RecyclerView.ViewHolder i RecyclerView.ItemDecoration	
8.1. Wstęp	322
8.2. Testowanie aplikacji Twitter Searches	323
8.2.1. Dodawanie ulubionych wyszukiwań	323
8.2.2. Przeglądanie wyników wyszukiwania wygenerowanych przez serwis Twitter	324
8.2.3. Edycja zapytania	326
8.2.4. Udostępnianie zapytania	327
8.2.5. Kasowanie zapytania	328
8.2.6. Przewijanie listy zapisanych zapytań	329
8.3. Zastosowane rozwiązania	329
8.3.1. Przechowywanie danych w postaci par klucz-wartość w pliku SharedPreferences	329
8.3.2. Niejawne intencje i wybór aplikacji obsługującej intencję	330
8.3.3. RecyclerView	331
8.3.4. Podklasy RecyclerView.Adapter i RecyclerView.ViewHolder	331

8.3.5. Podklasa RecyclerView.ItemDecoration	331
8.3.6. Wyświetlanie listy opcji w oknie AlertDialog	332
8.4. Tworzenie graficznego interfejsu użytkownika oraz plików zasobów	332
8.4.1. Tworzenie projektu	332
8.4.2. AndroidManifest.xml	332
8.4.3. Dodawanie biblioteki RecyclerView	332
8.4.4. colors.xml	333
8.4.5. strings.xml	333
8.4.6. arrays.xml	333
8.4.7. dimens.xml	334
8.4.8. Dodawanie ikony przycisku zapisywania	334
8.4.9. activity_main.xml	334
8.4.10. content_main.xml	335
8.4.11. Rozkład elementu RecyclerView — list_item.xml	337
8.5. Klasa MainActivity	338
8.5.1. Instrukcje package i import	338
8.5.2. Pola klasy MainActivity	339
8.5.3. Przedefiniowana metoda onCreate klasy Activity	340
8.5.4. Procedura obsługi zdarzeń TextWatcher i metoda updateSaveFAB	342
8.5.5. Interfejs OnClickListener nasłuchujący zdarzeń przycisku saveButton	343
8.5.6. Metoda addTaggedSearch	344
8.5.7. Anonimowa klasa wewnętrzna implementująca obiekt View.OnClickListener w celu wyświetlenia wyników wyszukiwania	345
8.5.8. Anonimowa klasa wewnętrzna, która implementuje interfejs View.OnLongClickListener umożliwiający udostępnienie, skasowanie i edycję zapisanego zapytania	346
8.5.9. Metoda shareSearch	348
8.5.10. Metoda deleteSearch	349
8.6. SearchesAdapter — podklasa RecyclerView.Adapter	350
8.6.1. Instrukcja package, instrukcje import, zmienne egzemplarzowe i konstruktor	350
8.6.2. Zagnieżdżona podklasa ViewHolder klasy RecyclerView.ViewHolder	351
8.6.3. Przedefiniowane metody klasy RecyclerView.Adapter	352
8.7. ItemDivider — podklasa klasy RecyclerView.ItemDecoration	353
8.8. Fabric — nowa platforma programistyczna serwisu Twitter	355
8.9. Podsumowanie	356

Rozdział 9. Aplikacja Address Book 357

Klasa FragmentTransaction i stos obiektów Fragment, SQLite, SQLiteDatabase, SQLiteOpenHelper, ContentProvider, ContentResolver, Loader, LoaderManager, Cursor i style graficznego interfejsu użytkownika

9.1. Wstęp	358
9.2. Testowanie aplikacji Address Book	360
9.2.1. Dodawanie kontaktu	360
9.2.2. Przeglądanie informacji przypisanych do kontaktu	361
9.2.3. Edycja kontaktu	361
9.2.4. Kasowanie kontaktu	361
9.3. Zastosowane rozwiązania	362
9.3.1. Wyświetlanie obiektów Fragment za pomocą klasy FragmentTransaction	362
9.3.2. Przesyłanie danych pomiędzy fragmentem a nadrzędną aktywnością	363

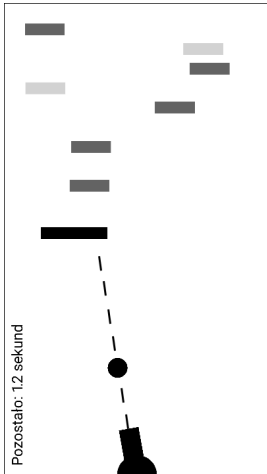
9.3.3. Obsługa bazy danych SQLite	363
9.3.4. Obiekty ContentProvider i ContentResolver	363
9.3.5. Asynchroniczny dostęp do bazy danych — obiekt Loader i menedżer LoaderManager	364
9.3.6. Definiowanie stylów i przypisywanie ich do komponentów graficznego interfejsu użytkownika	365
9.3.7. Określanie tła pola TextView	365
9.4. Tworzenie graficznego interfejsu użytkownika oraz plików zasobów	365
9.4.1. Tworzenie projektu	365
9.4.2. Tworzenie klas	366
9.4.3. Dodawanie ikon aplikacji	367
9.4.4. strings.xml	367
9.4.5. styles.xml	367
9.4.6. textview_border.xml	369
9.4.7. Rozkład aktywności MainActivity	369
9.4.8. Rozkład fragmentu ContactsFragment	371
9.4.9. Rozkład fragmentu DetailFragment	372
9.4.10. Rozkład fragmentu AddEditFragment	373
9.4.11. Menu fragmentu DetailFragment	375
9.5. Przegląd klas opisanych w dalszej części tego rozdziału	376
9.6. Klasa DatabaseDescription	377
9.6.1. Pola static	377
9.6.2. Zagnieżdżona klasa Contact	378
9.7. Klasa AddressBookDatabaseHelper	379
9.8. Klasa AddressBookContentProvider	380
9.8.1. Pola klasy AddressBookContentProvider	381
9.8.2. Przedefiniowane metody onCreate i getType	382
9.8.3. Przedefiniowana metoda query	383
9.8.4. Przedefiniowana metoda insert	385
9.8.5. Przedefiniowana metoda update	387
9.8.6. Przedefiniowana metoda delete	388
9.9. Klasa MainActivity	389
9.9.1. Klasa nadrzędna, zaimplementowane interfejsy i pola	389
9.9.2. Przedefiniowana metoda onCreate	390
9.9.3. Metody interfejsu ContactsFragment.ContactsFragmentListener	391
9.9.4. Metoda displayContact	392
9.9.5. Metoda displayAddEditFragment	393
9.9.6. Metody interfejsu DetailFragment.DetailFragmentListener	394
9.9.7. Metody interfejsu AddEditFragment.AddEditFragmentListener	394
9.10. Klasa ContactsFragment	395
9.10.1. Klasa nadrzędna i implementowany interfejs	395
9.10.2. Interfejs ContactsFragmentListener	396
9.10.3. Pola	396
9.10.4. Przedefiniowana metoda onCreateView	397
9.10.5. Przedefiniowane metody onAttach i onDetach obiektu Fragment	398
9.10.6. Przedefiniowana metoda onActivityCreated obiektu Fragment	398
9.10.7. Metoda updateContactList	399
9.10.8. Metody interfejsu LoaderManager.LoaderCallbacks<Cursor>	399
9.11. Klasa ContactsAdapter	401

9.12. Klasa AddEditFragment	404
9.12.1. Klasa nadrzędna i implementowany interfejs	404
9.12.2. Interfejs AddEditFragmentManager	404
9.12.3. Pola	405
9.12.4. Przedefiniowane metody obiektu Fragment — onAttach, onDetach i onCreateView	406
9.12.5. Obiekt nasłuchujący zdarzeń TextWatcher nameChangeListener i metoda updateSaveButtonFAB ...	407
9.12.6. Metoda saveContact i obiekt saveContactButtonClicked nasłuchujący zdarzeń View.OnClickListener	408
9.12.7. Metody interfejsu LoaderManager.LoaderCallbacks<Cursor>	410
9.13. Klasa DetailFragment	411
9.13.1. Klasa nadrzędna i implementowany interfejs	411
9.13.2. Zagnieżdżony interfejs DetailFragmentManager	412
9.13.3. Pola	412
9.13.4. Przedefiniowane metody onAttach, onDetach i onCreateView	413
9.13.5. Przedefiniowane metody onCreateOptionsMenu i onOptionsItemSelected	414
9.13.6. Metoda deleteContact i fragment DialogFragment confirmDelete	415
9.13.7. Metody interfejsu LoaderManager.LoaderCallback<Cursor>	416
9.14. Podsumowanie	417
Rozdział 10. Serwis Google Play i zagadnienia biznesowe związane z tworzeniem aplikacji	419
10.1. Wstęp	420
10.2. Przygotowywanie aplikacji do publikacji	420
10.2.1. Testowanie aplikacji	421
10.2.2. Umowa pomiędzy licencjodawcą a użytkownikiem końcowym	421
10.2.3. Ikony i etykiety	421
10.2.4. Oznaczanie aplikacji numerem wersji	422
10.2.5. Tworzenie licencji określających zasady uzyskiwania dostępu do płatnych aplikacji	422
10.2.6. Zaciemnianie kodu	423
10.2.7. Uzyskiwanie prywatnego klucza przeznaczonego do cyfrowego podpisywania aplikacji	423
10.2.8. Obraz promocyjny i zrzuty ekranu	423
10.2.9. Materiał wideo promujący aplikację	425
10.3. Wycena aplikacji — darmowa czy płatna?	425
10.3.1. Płatne aplikacje	426
10.3.2. Darmowe aplikacje	426
10.4. Zarabianie na reklamach wyświetlanych w aplikacji	428
10.5. Zarabianie na sprzedaży wirtualnych dóbr za pomocą mechanizmu płatności wbudowanego w aplikację	428
10.6. Rejestracja w serwisie Google Play	430
10.7. Konfigurowanie konta sprzedawcy w serwisie Google Payments	430
10.8. Ładowanie aplikacji do serwisu Google Play	431
10.9. Otwieranie sklepu z aplikacjami z poziomu aplikacji	433
10.10. Zarządzanie aplikacjami udostępnionymi w serwisie Google Play	434
10.11. Inne serwisy pośredniczące w sprzedaży aplikacji systemu Android	434
10.12. Inne platformy mobilne i przenoszenie na nie aplikacji	434
10.13. Rozreklamowanie aplikacji	435
10.14. Podsumowanie	439
Skorowidz	441

6

Aplikacja Cannon Game

Ręczna animacja klatka po klatce, grafika, dźwięk, tworzenie wielu wątków, obiekty `SurfaceView` i `SurfaceHolder`, tryb pełnoekranowy



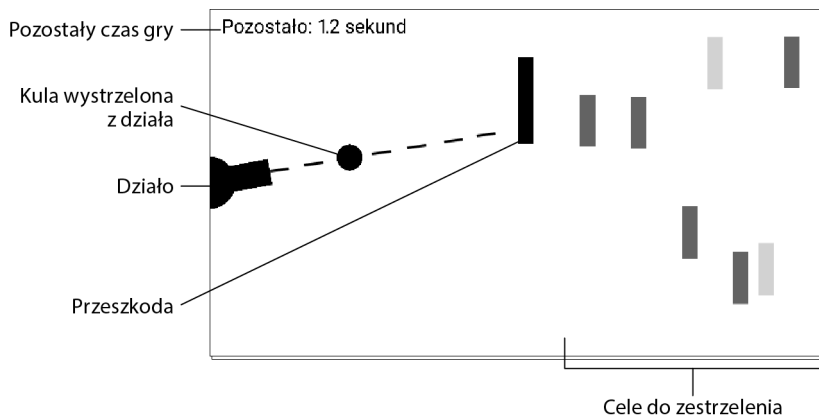
Tematyka

W tym rozdziale:

- Stworzysz prostą i dostarczającą wiele radości grę, której kod jest prosty.
- Utworzysz podklasy `SurfaceView`, które umożliwią wyświetlanie elementów graficznych gry za pomocą oddzielnego wątku.
- Wyświetlisz grafikę za pomocą obiektów `Paint` i `Canvas`.
- Przeprowadzisz metodę `onTouchEvent` klasy `View` tak, aby dotknięcie palcem do ekranu spowodowało wystrzelenie kuli.
- Opracujesz prosty mechanizm wykrywania zderzeń.
- Dodasz dźwięk do aplikacji za pomocą narzędzi `SoundPool` i `AudioManager`.
- Przeprowadzisz metodę `onDestroy` cyklu życia obiektu `Fragment`.
- Skorzystasz z trybu pełnoekranowego (*immersive mode*) — aplikacja zajmie całą dostępną powierzchnię ekranu, ale użytkownik wciąż będzie mógł uzyskać dostęp do pasków systemowych.

6.1. Wstęp

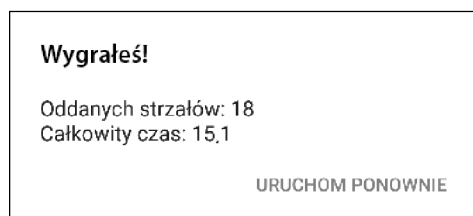
Aplikacja **Cannon Game**¹ jest grą, w której użytkownik ma za zadanie zestrzelić dziewięć celów przed upływem dziesięciosekundowego limitu czasu (patrz rysunek 6.1). Grafika gry składa się z czterech rodzajów komponentów — *działa*, wystrzelonej *kuli*, dziewięciu *celów* i *przeszkody*, która utrudnia strzelanie w kierunku celów. *Dotykając* ekranu, użytkownik celuje i oddaje strzał. Po dotknięciu ekranu przez użytkownika działo nakierowuje swoją lufę w stronę dotkniętego punktu i wyrzela kulę w tym kierunku (kula porusza się po linii prostej).



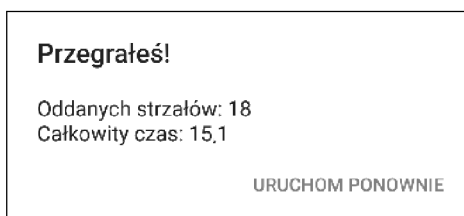
RYСУNEK 6.1. Aplikacja Cannon Game

Po trafieniu w cel gracz zostaje nagrodzony dodatkowymi trzema sekundami czasu gry, a przy każdym trafieniu w przeszkodę czas jest skracany o dwie sekundy. Gracz wygrywa, gdy zniszczy wszystkie dziewięć celów przed upływem czasu — gdy limit czasu wcześniej dojdzie do zera, gracz przegrywa. Na koniec aplikacja wyświetli okno `AlertDialog` informujące o przegranej lub wygranej, a także pokazujące liczbę wystrzałów i całkowity czas gry (patrz rysunek 6.2).

a) Okno `AlertDialog` wyświetlane, gdy użytkownik trafi we wszystkie dziewięć celów



b) Okno `AlertDialog` wyświetlane, gdy użytkownik nie trafi we wszystkie dziewięć celów przed upływem limitu czasu



RYСУNEK 6.2. Okna `AlertDialog` aplikacji Cannon Game wyświetlane w wyniku wygranej i przegranej

¹ Dziękujemy profesorowi Huguesowi Bersini — autorowi francuskojęzycznej książki dotyczącej programowania obiektowego napisanej dla Éditions Eyrolles, Secteur Informatique — za udzielenie nam wskazówek pozwalających na refaktoryzację kodu aplikacji *Cannon Game*. Wskazówki te stały się dla nas źródłem inspiracji i pozwoliły na opracowanie najnowszej wersji tej aplikacji opisaną w tej książce, a także w *iOS 8 for Programmers: An App-Driven Approach*.

Gra odtwarza *dźwięk wystrzału*, gdy z działa wystrzeliana jest kula. Gdy kula trafia w cel, odtwarzany jest *dźwięk tłuczonego szkła* i cel znika z ekranu. W przypadku trafienia kulą w przeszkodę odtwarzany jest dźwięk uderzenia i kula odbija się od przeszkody. Przeszkoda nie może zostać zniszczona. Cele oraz przeszkoda poruszają się z różnymi szybkościami w płaszczyźnie pionowej i zmieniają kierunek ruchu po dotknięciu do krawędzi ekranu.

Wskazówka: Emulator systemu Android działa wolno na niektórych komputerach. Aplikację najlepiej testować na prawdziwym urządzeniu. W przypadku wolnych emulatorów kula może czasem przelatywać przez przeszkodę i cele.

6.2. Testowanie aplikacji Cannon Game

Otwieranie i uruchamianie aplikacji

Uruchom środowisko Android Studio i otwórz grę *Cannon Game* znajdującą się w folderze *CannonGame*, a następnie uruchom ją na maszynie wirtualnej lub prawdziwym urządzeniu wyposażonym w system Android. Projekt zostanie zbudowany i uruchomiony.

Gra

Dotknij ekranu, aby wystrzelić kulę z działa. Kulę możesz wystrzelić tylko wtedy, gdy na ekranie nie znajduje się inna kula. Jeżeli korzystasz z emulatora, to rolę palca odgrywa mysz. Jak najszybciej zniszcz wszystkie cele — gra kończy się, jeżeli skończy Ci się czas lub trafisz we wszystkie cele.

6.3. Zastosowane rozwiązania

W sekcji tej przedstawimy nowe rozwiązania zastosowane w aplikacji *Cannon Game* w kolejności, w której zostaną one bardziej szczegółowo omówione w dalszej części rozdziału.

6.3.1. Korzystanie z folderu zasobów *res/raw*

Pliki multimediów, jak np. dźwięki używane przez aplikację *Cannon Game*, są umieszczane w folderze zasobów aplikacji *res/raw*. Tworzenie tego folderu opisano w sekcji 6.4.5. Po utworzeniu folderu będziesz mógł skopiować do niego pliki dźwiękowe.

6.3.2. Metody cyklu życia klas *Activity* i *Fragment*

W sekcji 5.3.1 wprowadziliśmy metod cyklu życia klas *Activity* i *Fragment*. Ta aplikacja korzysta z metody *onDestroy* klasy *Fragment*. Gdy aktywność jest zamykana, wywoływana jest jej metoda *onDestroy*, która wywołuje metody *onDestroy* wszystkich obiektów typu *Fragment* znajdujących się w danej aktywności. Korzystamy z tej metody w klasie *MainActivityFragment* w celu zwolnienia zasobów dźwiękowych obiektu *CannonView*.



Wskazówka zapobiegająca powstawaniu błędów 6.1

Nie ma gwarancji, że metoda *onDestroy* zostanie wywołana, a więc powinieneś jej używać tylko do zwalniania zasobów, a nie do zapisu danych. Dokumentacja platformy Android zaleca zapisywanie danych za pomocą metod *onPause* lub *onSaveInstanceState*.

6.3.3. Przedefiniowanie metody onTouchEvent klasy View

Użytkownik nawiązuje interakcję z aplikacją, dotykając ekranu. Działo obraca lufę w kierunku punktu dotknięcia, a następnie strzela kulą. W celu obsługi zdarzeń dotyku przeddefiniujemy metodę onTouchEvent klasy View (patrz sekcja 6.13.14). Typ zdarzenia zostanie określony za pomocą stałych klasy MotionEvent (pakiet *android.view*), a następnie odpowiednio przetworzony.

6.3.4. Dodawanie dźwięku za pomocą narzędzi SoundPool i AudioManager

Efektami dźwiękowymi aplikacji zarządza SoundPool (pakiet *android.media*). Narzędzie to przydaje się do *ładowania, odtwarzania i usuwania* dźwięków. Dźwięki są odtwarzane za pomocą jednego ze strumieni dźwiękowych systemu Android, służącego do *alarmowania, odtwarzania muzyki, powiadomiania, dzwonienia, odtwarzania dźwięków systemowych, obsługi połączeń telefonicznych* itd. Obiekt SoundPool stworzysz i skonfigurujesz za pomocą obiektu SoundPool.Builder. Ponadto będziesz korzystał z obiektu AudioAttributes.Builder w celu utworzenia obiektu AudioAttributes, który będzie powiązany z obiektem SoundPool. Metodę setUsage obiektu AudioAttributes wywołujemy w celu określenia tego, że dźwięk ma być dźwiękiem gry. Dokumentacja platformy Android zaleca, aby gry korzystały ze *strumienia muzyka-audio*, ponieważ głośność tego strumienia może być regulowana za pomocą przycisków znajdujących się na obudowie urządzenia. Ponadto będziemy korzystać z metody setVolumeControlStream klasy Activity, która umożliwia sterowanie głośnością dźwięku generowanego przez grę za pomocą przycisków głośności. Metoda ta otrzymuje stałą z klasy AudioManager (pakiet *android.media*), która umożliwia uzyskanie dostępu do sterowania głośnością urządzenia i dzwonka telefonu.

6.3.5. Tworzenie kolejnych klatek animacji za pomocą wątków i obiektów SurfaceView i SurfaceHolder

Aplikacja wykonuje animację, tworząc kolejne klatki w wyniku aktualizacji położenia elementów gry za pomocą zewnętrznego wątku. W tym celu korzystamy z podklasy Thread z metodą run, która umożliwia aktualizację położenia elementów gry za pomocą zmodyfikowanego obiektu CannonView. Metoda run tworzy *kolejne klatki animacji* — jest to tzw. **pętla gry**.

Wszystkie aktualizacje interfejsu użytkownika aplikacji muszą być wykonywane w wątku wykonywania graficznego interfejsu użytkownika, ponieważ aktualizowanie komponentów interfejsu użytkownika poza wspomnianym wątkiem może prowadzić do ich uszkodzenia. Wątki tego typu muszą zwykle wyświetlać elementy na ekranie. W takich przypadkach można skorzystać z klasy SurfaceView — podklasy View, która tworzy obszar dedykowany do bezpiecznego wyświetlania obiektów graficznych przez wątki.



Wskazówka dotycząca poprawy wydajności 6.1

Minimalizacja czynności wykonywanych w wątku graficznego interfejsu użytkownika jest bardzo ważna, ponieważ pozwala zachować responsywność interfejsu i uniknąć wyświetlania błędów „Aplikacja nie odpowiada”.

Operacje klasy SurfaceView są wykonywane za pośrednictwem obiektu klasy SurfaceHolder. Umożliwia on uzyskanie dostępu do obiektu Canvas, na którym możliwe jest wyświetlanie grafiki. Klasa SurfaceHolder jest również wyposażona w metody, które zapewniają wyłączny dostęp do obiektu Canvas dla danego wątku — elementy wyświetlane przez obiekt SurfaceView mogą być tworzone w danym momencie tylko przez jeden wątek. Każda podklasa SurfaceView powinna implementować interfejs SurfaceHolder.Callback, który zawiera metody wywoływane, gdy obiekt SurfaceView jest *tworzony, modyfikowany* (zmienia się np. jego rozmiar lub położenie) lub *niszczony*.

6.3.6. Proste wykrywanie zderzeń

Klasa `CannonView` wykonuje operacje prostego wykrywania zderzeń, które określają, czy kula zderzyła się z krawędzią obiektu `CannonView`, przeszkodą lub celem. Technika ta została szczegółowo opisana w sekcji 6.13.11.

Ramy projektowe przeznaczone do tworzenia gier zwykle dysponują bardziej wyrafinowanymi i dokładniejszymi technikami wykrywania zderzeń. Istnieje wiele ram projektowych (darmowych, a także płatnych) przeznaczonych do tworzenia prostych gier 2D, a także o wiele bardziej złożonych trójwymiarowych gier przeznaczonych dla konsol takich jak Sony PlayStation i Microsoft Xbox. W tabeli 6.1 przedstawiono listę kilku ram projektowych przeznaczonych do tworzenia gier. Wiele z nich obsługuje wiele platform (w tym między innymi systemy Android i iOS). Niektóre wymagają znajomości C++ lub innego języka programowania.

TABELA 6.1. Ramy projektowe przeznaczone do tworzenia gier

Ramy projektowe przeznaczone do tworzenia gier
AndEngine — http://www.andengine.org/
Cocos2D — http://code.google.com/p/cocos2d-android
GameMaker — http://www.yoyogames.com/studio
libgdx — https://libgdx.badlogicgames.com/
Unity — http://www.unity3d.com/
Unreal Engine — http://www.unrealengine.com/

6.3.7. Tryb pełnoekranowy

Aby użytkownikowi lepiej się grało, twórcy często korzystają z pełnoekranowych motywów, takich jak:

```
Theme.Material.Light.NoActionBar.Fullscreen.
```

Szablon ten wyświetla tylko dolny pasek systemowy. W przypadku telefonów w orientacji poziomej pasek ten pojawia się z prawej strony ekranu.

Firma Google wprowadziła obsługę trybu pełnoekranowego (*immersive mode*) w systemie Android 4.4 (KitKat) (patrz sekcja 6.13.16). Pozwala on aplikacjom na korzystanie z całej powierzchni ekranu. We wspomnianym trybie pasek systemowy jest chwilowo wyświetlany na ekranie dopiero wtedy, gdy użytkownik przeciągnie palcem po ekranie od góry do dołu. Jeżeli użytkownik nie nawiązuje interakcji z paskiem, to jest on ukrywany po kilku sekundach.

6.4. Tworzenie graficznego interfejsu użytkownika i plików zasobów

W tej sekcji stworzysz pliki zasobów aplikacji, pliki rozkładów graficznego interfejsu użytkownika i klasy.

6.4.1. Tworzenie projektu

W przypadku tej aplikacji będziesz musiał dodać obiekt `Fragment` do rozkładu ręcznie — większa część automatycznie wygenerowanego kodu szablonu *Blank Activity* jest zbędna. Utwórz nowy projekt za pomocą szablonu *Empty Activity*. W oknie *Create New Project* (utwórz nowy projekt), w kroku *New Project* (nowy projekt), zadeklaruj następujące parametry:

- Jako nazwę aplikacji (*Application name*) podaj `Cannon Game`.
- Jako domenę firmy (*Company Domain*) podaj `dei1.com` (lub podaj nazwę własnej domeny).

Skorzystaj z listy urządzeń widocznej w edytorze projektu i wybierz telefon *Nexus 6* (patrz rysunek 2.8). Kolejny raz będziemy opierać projekt na tym urządzeniu. Następnie usuń pole `TextView` zawierające napis *Hello world!* z `activity_main.xml`. Tak jak wcześniej dodaj ikonę aplikacji do projektu.

Przystosowanie aplikacji do pracy w orientacji poziomej

Gra *Cannon Game* jest zaprojektowana do pracy tylko w orientacji poziomej. Wykonaj czynności opisane w sekcji 3.7 w celu określenia orientacji ekranu, ale tym razem własności `android:screenOrientation` przypisz parametr `landscape`, a nie `portrait`.

6.4.2. Modyfikacja motywu umożliwiająca usunięcie nazwy aplikacji i jej paska

W sekcji 6.3.7 pisaliśmy o tym, że twórcy gier korzystają często z motywów pełnoekranowych, takich jak

```
Theme.Material.Light.NoActionBar.Fullscreen
```

Szablon ten wyświetla tylko dolny pasek systemowy. W przypadku telefonów w orientacji poziomej pasek ten pojawia się z prawej strony ekranu. Szablony `AppCompatActivity` domyślnie nie dołączają motywu pełnoekranowego, ale możesz samodzielnie zmodyfikować szablon aplikacji. W tym celu:

1. Otwórz plik `styles.xml`.
2. Do elementu `<style>` dodaj następujące linie kodu:

```
<item name="windowNoTitle">true</item>
<item name="windowActionBar">false</item>
<item name="android:windowFullscreen">true</item>
```

Pierwsza linia kodu informuje o tym, że tytuł (zwykle jest on nazwą aplikacji) nie powinien być wyświetlany. Druga linia kodu informuje o tym, że pasek aplikacji nie powinien być wyświetlany. Ostatnia linia kodu informuje o tym, że aplikacja powinna korzystać z pełnego ekranu.

6.4.3. strings.xml

W poprzednich rozdziałach tworzyłeś zasoby będące łańcuchami, a więc tym razem przedstawiamy Ci tylko tabelę zawierającą nazwy takich zasobów, a także przypisaną im treść (patrz tabela 6.2). Kliknij dwukrotnie plik `strings.xml` znajdujący się w folderze `res/values`, a następnie kliknij odwołanie *Open editor* (otwórz edytor) w celu otwarcia okna *Translations Editor* (edytor tłumaczeń) umożliwiającego tworzenie zasobów typu `String`.

TABELA 6.2. Łańcuchy aplikacji Cannon Game

Klucz	Wartość
<code>results_format</code>	Oddanych strzałów: %1\$d\nCałkowity czas: %2\$.1
<code>reset_game</code>	Uruchom ponownie
<code>win</code>	Wygrałeś!
<code>lose</code>	Przegrałeś!
<code>time_remaining_format</code>	Pozostało: %,1f sekund

6.4.4. Kolory

Cele są obiektami typu Canvas o dwóch różnych kolorach. W pliku *colors.xml* dodaliśmy dwa następujące kolory (ciemnoniebieski i żółty):

```
<color name="dark">#1976D2</color>
<color name="light">#FFE100</color>
```

6.4.5. Dodawanie dźwięku

Pisaliśmy wcześniej o tym, że pliki dźwiękowe są przechowywane w folderze *res/raw*. Opisywana aplikacja korzysta z trzech plików dźwiękowych: *blocker_hit.wav*, *target_hit.wav* i *cannon_fire.wav*, które umieściliśmy w folderze z przykładowymi aplikacjami w katalogu *sounds*. Aby dodać te pliki do projektu:

1. Prawym przyciskiem myszy kliknij folder *res*, a następnie wybierz *New/Android resource directory* (nowy/folder zasobu systemu Android) w celu otwarcia okna *New Resource Directory* (nowy folder zasobu).
2. W rozwijanym menu *Resource type* (typ zasobu) wybierz *raw*. Nazwa katalogu (*Directory name*) zostanie automatycznie zmieniona na *raw*.
3. Utwórz folder, klikając przycisk *OK*.
4. Skopiuj i wklej pliki dźwiękowe do folderu *res/raw*. W wyświetlonym oknie dialogowym *Copy* (kopiuj) kliknij przycisk *OK*.

6.4.6. Dodawanie klasy MainActivityFragment

Teraz czas na dodanie do projektu klasy *MainActivityFragment*.

1. W oknie *Project* kliknij prawym przyciskiem myszy węzeł *com.deitel.cannongame* i wybierz *New/Fragment/Fragment (Blank)*.
2. Jako nazwę fragmentu (pole *Fragment Name*) wpisz *MainActivityFragment*, a w polu *Fragment Layout Name* (nazwa rozkładu fragmentu) wpisz nazwę *fragment_main*.
3. Usuń zaznaczenie opcji *Include fragment factory methods?* i *Include interface callbacks?*.

Domyślnie *fragment_main.xml* zawiera rozkład *FrameLayout* wyświetlający obiekt *TextView*. Rozkład *FrameLayout* został zaprojektowany w celu wyświetlania jednego obiektu typu *View*, ale może być on również użyty w celu wyświetlenia kilku takich obiektów ułożonych warstwowo. Usuń obiekt *TextView* — w tej aplikacji rozkład *FrameLayout* będzie wyświetlał obiekt *CannonView*.

6.4.7. Edycja pliku activity_main.xml

W przypadku opisywanej aplikacji rozkład aktywności *MainActivity* wyświetla tylko obiekt *MainActivityFragment*. Edytuj obiekt, wykonując następujące czynności:

1. Otwórz plik *activity_main.xml* w edytorze rozkładu i wybierz zakładkę *Text* (tekst).
2. Zmień *RelativeLayout* na *fragment* i usuń własności wypełnienia, tak aby element *fragment* zajmował cały ekran.
3. Przejdź do zakładki *Design* (projekt), w polu *Component Tree* (drzewo komponentów) wybierz element *fragment* i przypisz mu identyfikator (*id*): *fragment*.
4. Podaj nazwę (*name*) *com.deitel.cannongame.MainActivityFragment*. Zamiast wpisywać ją ręcznie, kliknij przycisk wielokropka znajdujący się po prawej stronie pola wartości właściwości *name* i wybierz odpowiednią klasę z wyświetlonego okna *Fragments* (*fragmenty*).

Przypominamy, że widok Design edytora rozkładu może wyświetlać podgląd fragmentu wyświetlanego w danym rozkładzie. Jeżeli nie określisz fragmentu, który ma być wyświetlany jak podgląd rozkładu `MainActivity`, to edytor rozkładu wyświetli komunikat informujący o problemach z renderowaniem. W celu wybrania fragmentu wyświetlanego w podglądzie w polu *Component Tree* lub w trybie *Design* kliknij opcję *Choose Preview Layout...* (wybierz rozkład projektu), a następnie w oknie *Resources* (zasoby) wybierz nazwę fragmentu.

6.4.8. Dodawanie obiektu `CannonView` do fragmentu `fragment_main.xml`

Teraz dodasz obiekt `CannonView` do fragmentu `fragment_main.xml`. Najpierw musisz utworzyć plik `CannonView.java`, co pozwoli na wybór klasy `CannonView` podczas umieszczania obiektu `CustomView` w rozkładzie. W celu utworzenia pliku `CannonView.java` i dodania obiektu `CannonView` do fragmentu `fragment_main.xml` wykonaj następujące czynności:

1. W oknie *Project* (projekt) rozwiń folder *java*.
2. Prawym przyciskiem myszy kliknij folder pakietu *com.deitel.cannongame*, a następnie wybierz *New/Java Class* (nowa/klasa Javy).
3. W oknie *Create New Class* (utwórz nową klasę), w polu *Name*, wprowadź nazwę `CannonView` i kliknij przycisk *OK*. Spowoduje to automatyczne otwarcie pliku w edytorze.
4. W pliku `CannonView.java` zaznacz, że klasa `CannonView` rozszerza widok `SurfaceView`. Jeżeli nie widzisz instrukcji importu klasy `android.view.SurfaceView`, to umieść kursor na końcu nazwy klasy `SurfaceView`. Kliknij czerwoną żarówkę (🔦), która pojawi się nad początkiem linii kodu, i wybierz opcję *Import Class* (importuj klasę).
5. Jeżeli jeszcze tego nie zrobiłeś, to umieść kursor na końcu `SurfaceView`. Kliknij czerwoną żarówkę i wybierz opcję *Create constructor matching super* (dobór nadrzędnego konstruktora). Z listy *Choose Super Class Constructors* (wybierz konstruktor klasy nadrzędnej) wybierz konstruktor dwuargumentowy i kliknij przycisk *OK*. Środowisko programistyczne automatycznie doda do pliku wybrany przez Ciebie konstruktor.
6. Przejdź z powrotem do zakładki *Design* edytora rozkładu `fragment_main.xml`.
7. W sekcji *Custom* (personalizacja) menu *Palette* (paleta) kliknij obiekt `CustomView`.
8. W wyświetlonym oknie *View* (widok) wybierz `CannonView` (*com.deitel.cannongame*) i kliknij przycisk *OK*.
9. Poczekaj chwilę i kliknij element `FrameLayout` widoczny w oknie *Component Tree*. Powinieneś zobaczyć element *view* (`CustomView`) będący obiektem `CannonView` zlokalizowanym w rozkładzie `FrameLayout`.
10. Upewnij się, że w oknie *Component Tree* zaznaczono element *view* (`CustomView`). W oknie *Properties* własnościom `layout:width` i `layout:height` przypisz parametr `match_parent`.
11. Zmień identyfikator `id` z `view` na `cannonView` (okno *Properties*).
12. Zapisz zmiany w pliku `fragment_main.xml` i go zamknij.

6.5. Przegląd klas aplikacji

Aplikacja składa się z ośmiu klas:

- `MainActivity` (podklasa `Activity`; patrz sekcja 6.6) — zawiera `MainActivityFragment`.
- `MainActivityFragment` (patrz sekcja 6.7) — wyświetla klasę `CannonView`.

- `GameElement` (patrz sekcja 6.8) — nadrzędna klasa elementów poruszających się w górę i w dół ekranu (elementy `Blocker` i `Target`) oraz elementów poruszających się wzdłuż ekranu (`Cannonball`).
- `Blocker` (patrz sekcja 6.9) — reprezentuje przeszkodę utrudniającą zniszczenie celów.
- `Target` (patrz sekcja 6.10) — reprezentuje cel, który może zostać zniszczony w wyniku trafienia kulą.
- `Cannon` (patrz sekcja 6.11) — reprezentuje działo strzelające kulami w wyniku dotknięcia ekranu przez użytkownika.
- `Cannonball` (patrz sekcja 6.12) — reprezentuje kulę wystrzeliwaną przez działo po dotknięciu ekranu przez użytkownika.
- `CannonView` (patrz sekcja 6.13) — zawiera logikę gry i koordynuje zachowanie obiektów `Blocker`, `Target`, `Cannonball` i `Cannon`.

Musisz utworzyć klasy `GameElement`, `Blocker`, `Target`, `Cannonball` i `Cannon`. Tworzenie każdej klasy polega na kliknięciu prawym przyciskiem myszy folderu pakietu `com.deitel.cannongame` znajdującego się w podkatalogu `app/java` i wybraniu menu `New/Java Class`. W oknie `Create New Class` wpisz nazwę klasy w polu `Name` i kliknij przycisk `OK`.

6.6. MainActivity — podklasa klasy Activity

Klasa `MainActivity` (patrz listing 6.5) zawiera obiekt `MainActivityFragment`. W przypadku aplikacji `Cannon Game` musimy przededefiniować tylko metodę `onCreate` klasy `Activity`, która odpowiada za tworzenie graficznego interfejsu użytkownika. Skasowaliśmy automatycznie wygenerowane metody klasy `MainActivity`, które miały zarządzać menu tej aktywności (w opisywanej aplikacji nie korzystamy z tego menu).

LISTING 6.1. Klasa `MainActivity` wyświetlająca obiekt `MainActivityFragment`

```

1 // MainActivity.java
2 // Klasa MainActivity wyświetlająca obiekt MainActivityFragment
3 package com.deitel.cannongame;
4
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7
8 public class MainActivity extends AppCompatActivity {
9     // metoda wywoływana przy pierwszym uruchomieniu aplikacji
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.activity_main);
14    }
15 }

```

6.7. MainActivityFragment — podklasa klasy Fragment

Klasa MainActivityFragment (patrz listing 6.2) przeddefiniuje cztery metody klasy Fragment:

- onCreateView (linie 17 – 28) — w sekcji 4.3.3 dowiedziałeś się, że metoda ta jest wywoływana po metodzie onCreate klasy Fragment w celu zbudowania i zwrócenia obiektu View zawierającego graficzny interfejs użytkownika elementu Fragment. Kod znajdujący się w liniach 22 – 23 przygotowuje ten interfejs do wyświetlenia na ekranie. Kod znajdujący się w 26. linii uzyskuje odwołanie do obiektu CannonView należącego do MainActivityFragment, dzięki czemu możemy wywoływać jego metody.
- onActivityCreated (linie 31 – 37) — metoda ta jest wywoływana po utworzeniu obiektu Activity, w którym znajduje się element Fragment. Kod znajdujący się w 36. linii wywołuje metodę setVolumeControlStream klasy Activity w celu umożliwienia regulacji głośności dźwięku generowanego przez grę za pomocą przycisków znajdujących się na obudowie urządzenia. Stałe AudioManager identyfikują siedem strumieni dźwiękowych, ale w przypadku gier zaleca się korzystanie ze strumienia muzyki (AudioManager.STREAM_MUSIC), ponieważ głośność tego strumienia może być regulowana za pomocą przycisków.
- onPause (linie 40 – 44) — gdy aktywność MainActivity zostaje przeniesiona do tła (a tym samym wstrzymana), wykonywana jest metoda onPause klasy MainActivityFragment. Kod znajdujący się w linii 43. wywołuje metodę stopGame klasy CannonView (patrz sekcja 6.13.12), która zatrzymuje działanie pętli gry.
- onDestroy (linie 47 – 51) — gdy obiekt klasy MainActivity jest usuwany, uruchamia metodę onDestroy, która wywołuje metodę onDestroy obiektów MainActivityFragment. W linii 50. wywoływana jest metoda releaseResources klasy CannonView, która zwalnia zasoby dźwiękowe (patrz sekcja 6.13.12).

LISTING 6.2. Klasa MainActivityFragment tworzy obiekt CannonView i nim zarządza

```
1 // MainActivityFragment.java
2 // Klasa MainActivityFragment tworzy obiekt CannonView i zarządza nim
3 package com.deitel.cannongame;
4
5 import android.media.AudioManager;
6 import android.os.Bundle;
7 import android.support.v4.app.Fragment;
8 import android.view.LayoutInflater;
9 import android.view.View;
10 import android.view.ViewGroup;
11
12 public class MainActivityFragment extends Fragment {
13     private CannonView cannonView; // custom view to display the game
14
15     // metoda wywoływana w przypadku konieczności utworzenia widoku w obiekcie Fragment
16     @Override
17     public View onCreateView(LayoutInflater inflater, ViewGroup container,
18         Bundle savedInstanceState) {
19         super.onCreateView(inflater, container, savedInstanceState);
20
21         // przygotuj do wyświetlenia rozkład fragment_main.xml
22         View view =
23             inflater.inflate(R.layout.fragment_main, container, false);
```



```

24
25     // uzyskaj odwołanie do CannonView
26     cannonView = (CannonView) view.findViewById(R.id.cannonView);
27     return view;
28 }
29
30 // przygotuj możliwość sterowania głośnością po utworzeniu obiektu Activity
31 @Override
32 public void onActivityCreated(Bundle savedInstanceState) {
33     super.onActivityCreated(savedInstanceState);
34
35     // pozwól na regulację głośności gry za pomocą przycisków głośności
36     getActivity().setVolumeControlStream(AudioManager.STREAM_MUSIC);
37 }
38
39 // zatrzymaj grę w przypadku wstrzymania MainActivity
40 @Override
41 public void onPause() {
42     super.onPause();
43     cannonView.stopGame(); // zatrzymaj działanie gry
44 }
45
46 // w przypadku wstrzymania MainActivity MainActivityFragment zwalnia zasoby
47 @Override
48 public void onDestroy() {
49     super.onDestroy();
50     cannonView.releaseResources();
51 }
52 }

```

6.8. Klasa GameElement

Klasa `GameElement` (patrz listing 6.3) jest klasą nadrzędną obiektów `Blocker`, `Target` i `Cannonball` — zawiera dane wspólne obiektów poruszających się na ekranie gry *Cannon Game*, a także dane dotyczące ich funkcjonowania.

LISTING 6.3. Klasa `GameElement` opisuje elementy gry charakteryzujące się prostokątnymi granicami

```

1 // GameElement.java
2 // Opisuje elementy gry charakteryzujące się prostokątnymi granicami.
3 package com.deitel.cannongame;
4
5 import android.graphics.Canvas;
6 import android.graphics.Paint;
7 import android.graphics.Rect;
8
9 public class GameElement {
10     protected CannonView view; // widok, w którym znajduje się element GameElement
11     protected Paint paint = new Paint(); // obiekt Paint rysujący element GameElement
12     protected Rect shape; // prostokątne granice elementu GameElement
13     private float velocityY; // prędkość ruchu elementu GameElement w płaszczyźnie pionowej

```

```

14 private int soundId; // dźwięk powiązany z elementem GameElement
15
16 // publiczny konstruktor
17 public GameElement(CannonView view, int color, int soundId, int x,
18     int y, int width, int length, float velocityY) {
19     this.view = view;
20     paint.setColor(color);
21     shape = new Rect(x, y, x + width, y + length); // określ granice
22     this.soundId = soundId;
23     this.velocityY = velocityY;
24 }
25
26 // aktualizuj położenie elementu GameElement i sprawdź, czy nie doszło do zderzenia ze ścianą
27 public void update(double interval) {
28     // aktualizuje położenie w pionie
29     shape.offset(0, (int) (velocityY * interval));
30
31     // zmień kierunek ruchu elementu GameElement w przypadku zderzenia się go ze ścianą
32     if (shape.top < 0 && velocityY < 0 ||
33         shape.bottom > view.getScreenHeight() && velocityY > 0)
34         velocityY *= -1; // odwróć prędkość elementu GameElement
35 }
36
37 // rysuje element GameElement na danym obiekcie Canvas
38 public void draw(Canvas canvas) {
39     canvas.drawRect(shape, paint);
40 }
41
42 // odtwarza dźwięk właściwy dla danego elementu GameElement
43 public void playSound() {
44     view.playSound(soundId);
45 }
46 }

```

6.8.1. Zmienne egzemplarzowe i konstruktor

Konstruktor `GameElement` otrzymuje odwołanie do klasy `CannonView` (patrz sekcja 6.13), która implementuje logikę gry i wyświetla na ekranie jej elementy. Konstruktor otrzymuje wartość `int` określającą 32-bitowy kolor elementu `GameElement`, a także wartość `int` będącą identyfikatorem dźwięku związanego z tym elementem. Klasa `CannonView` przechowuje wszystkie dźwięki gry i dostarcza ich identyfikatory. Konstruktor otrzymuje ponadto:

- wartości typu `int` określające współrzędne `x` i `y` górnego lewego rogu elementu `GameElement`,
- wartości typu `int` określające szerokość (`width`) i wysokość (`height`),
- początkową szybkość przemieszczania się w płaszczyźnie pionowej (`velocityY`) elementu `GameElement`.

Kod znajdujący się w linii 20. określa kolor obiektu `paint` za pomocą wartości `int` przekazanej do konstruktora. Linia 21. oblicza granice elementu `GameElement` i zapisuje je w obiekcie `Rect`, który opisuje prostokąt.

6.8.2. Metody update, draw i playSound

W klasie `GameElement` znajdują się następujące metody:

- `update` (linie 27 – 35) — podczas każdej iteracji pętli gry metoda ta jest wywoływana w celu aktualizacji pozycji elementu `GameElement`. Kod znajdujący się w linii 29. aktualizuje położenie obiektu `shape` w płaszczyźnie poziomej na podstawie prędkości ruchu tego obiektu we wspomnianej płaszczyźnie (`velocityY`) i czasu, jaki upłynął pomiędzy poszczególnymi wywołaniami metody `update` (metoda otrzymuje dane dotyczące czasu w postaci parametru `interval`). Linie 32 – 34 sprawdzają, czy dany element `GameElement` nie zderza się z górną lub dolną krawędzią ekranu, a w razie kolizji odwracają zwrot prędkości tego elementu.
- `draw` (linie 38 – 40) — metoda ta jest wywoływana, gdy element `GameElement` musi zostać ponownie narysowany na ekranie. Metoda otrzymuje obiekt `Canvas` i rysuje dany element `GameElement` na ekranie jako prostokąt — metodę tę przeddefiniujemy w klasie `CannonBall` w celu rysowania koła. Zmienna egzemplarzowa `paint` klasy `GameElement` określa kolor prostokąta, a zmienna `shape` określa granice prostokąta wyświetlanego na ekranie.
- `playSound` (linie 43 – 45) — do każdego elementu gry przypisany jest dźwięk, który może zostać odtworzony poprzez wywołanie metody `playSound`. Metoda ta przekazuje wartość zmiennej egzemplarzowej `soundId` do metody `playSound` klasy `CannonView`. Klasa `CannonView` ładuje i utrzymuje odwołania do dźwięków gry.

6.9. Blocker — podklasa klasy GameElement

Klasa `Blocker` (patrz listing 6.4) jest podklasą klasy `GameElement` opisującą przeszkodę, która utrudnia graczowi trafienie w cele. Gdy kula trafi w przeszkodę, czas określany przez wartość `missPenalty` jest odejmowany od pozostałego czasu gry. Metoda `getMissPenalty` (linie 17 – 19) zwraca wartość `missPenalty` — metoda ta jest wywoływana przez metodę `testForCollisions` klasy `CannonView` — powoduje to pomniejszenie pozostałego czasu o wartość przypisaną zmiennej `missPenalty` (patrz sekcja 6.13.11). Konstruktor klasy `Blocker` (linie 9 – 14) przekazuje swoje argumenty i identyfikator dźwięku uderzenia w przeszkodę (`CannonView.BLOCKER_SOUND_ID`) do konstruktora klasy nadrzędnej (linia 11.), a następnie inicjuje zmienną `missPenalty`.

LISTING 6.4. Blocker — podklasa klasy `GameElement`

```
1 // Blocker.java
2 // Podklasa klasy GameElement obsługująca przeszkodę.
3 package com.deitel.cannongame;
4
5 public class Blocker extends GameElement {
6     private int missPenalty; // kara za trafienie w przeszkodę
7
8     // konstruktor
9     public Blocker(CannonView view, int color, int missPenalty, int x,
10         int y, int width, int length, float velocityY) {
11         super(view, color, CannonView.BLOCKER_SOUND_ID, x, y, width, length,
12             velocityY);
13         this.missPenalty = missPenalty;
14     }
15
16     // zwraca karę za trafienie w przeszkodę
17     public int getMissPenalty() {
18         return missPenalty;
19     }
20 }
```

6.10. Target — podklasa klasy GameElement

Klasa Target (patrz listing 6.5) jest podklasą klasy GameElement opisującą cel, który może zostać zniszczony przez gracza. Czas określany przez zmienną hitPenalty klasy Target jest dodawany do pozostałego czasu rozgrywki, jeżeli kula wystrzelona z działa trafi w cel. Metoda getHitReward (linie 17 – 18) zwraca obiekt hitReward. Metoda ta jest wywoływana przez metodę testForCollisions klasy CannonView podczas dodawania wartości hitPenalty do pozostałego czasu rozgrywki (patrz sekcja 6.13.11). Konstruktor klasy Target (linie 9 – 14) przekazuje swoje argumenty i identyfikator dźwięku trafienia w cel (CannonView.TARGET_SOUND_ID) konstruktorowi nadrzędnemu (linia 11.) i inicjuje zmienną hitReward.

LISTING 6.5. Target — podklasa klasy GameElement

```
1 // Target.java
2 // Podklasa GameElement przeznaczona do obsługi celów.
3 package com.deitel.cannongame;
4
5 public class Target extends GameElement {
6     private int hitReward; // nagroda za trafienie w ten cel
7
8     // konstruktor
9     public Target(CannonView view, int color, int hitReward, int x, int y,
10         int width, int length, float velocityY) {
11         super(view, color, CannonView.TARGET_SOUND_ID, x, y, width, length,
12             velocityY);
13         this.hitReward = hitReward;
14     }
15
16     // zwraca nagrodę za trafienie w ten cel
17     public int getHitReward() {
18         return hitReward;
19     }
20 }
```

6.11. Klasa Cannon

Klasa Cannon (patrz listingi 6.6 – 6.10) przedstawia działo z gry *Cannon Game*. Działo składa się z podstawy i lufy, z której można wystrzelić kulę.

6.11.1. Zmienne egzemplarzowe i konstruktor

Do konstruktora klasy Cannon (patrz listing 6.6) przekazywane są cztery parametry:

- obiekt CannonView, w którym znajduje się dany obiekt Cannon (view),
- promień podstawy danego obiektu Cannon (baseRadius),
- długość lufy danego obiektu Cannon (barrelLength),
- szerokość lufy danego obiektu Cannon (barrelWidth).

```

1 // Cannon.java
2 // Metoda przedstawiająca działo i proces wystrzału kuli.
3 package com.deitel.cannongame;
4
5 import android.graphics.Canvas;
6 import android.graphics.Color;
7 import android.graphics.Paint;
8 import android.graphics.Point;
9
10 public class Cannon {
11     private int baseRadius; // promień podstawy działła
12     private int barrelLength; // długość lufy działła
13     private Point barrelEnd = new Point(); // końcowy punkt lufy
14     private double barrelAngle; // kąt ustawienia lufy
15     private Cannonball cannonball; // kula wystrzeliana z działła
16     private Paint paint = new Paint(); // obiekt Paint używany do rysowania działła
17     private CannonView view; // obiekt view zawierający działło
18
19     // konstruktor
20     public Cannon(CannonView view, int baseRadius, int barrelLength,
21         int barrelWidth) {
22         this.view = view;
23         this.baseRadius = baseRadius;
24         this.barrelLength = barrelLength;
25         paint.setStrokeWidth(barrelWidth); // określ szerokość lufy
26         paint.setColor(Color.BLACK); // działło jest koloru czarnego
27         align(Math.PI / 2); // lufa jest zwrócona na wprost, w prawo
28     }
29

```

Kod znajdujący się w linii numer 25. określa szerokość obiektu Paint tak, aby lufa działła była wyświetlana z grubością określoną przez barrelWidth. W 27. linii kodu lufa działła jest ustawiana tak, aby początkowo była równoległa do górnej i dolnej krawędzi ekranu. Klasa Cannon zawiera obiekty: Point barrelEnd, który jest używany do rysowania lufy na ekranie, barrelAngle, który przechowuje obecny kąt działła, i cannonball przechowujący dane dotyczące ostatnio wystrzelonej kuli (o ile kula ta jest wciąż wyświetlana na ekranie).

6.11.2. Metoda align

Metoda align (patrz listing 6.7) celuje lufą działła. Do tej metody przekazywany jest w postaci argumentu kąt ustawienia działła wyrażony w radianach. Współrzędne x i y położenia końca lufy są obliczane na podstawie wartości cannonLength i barrelAngle i przypisywane obiektowi barrelEnd, który jest używany w celu narysowania linii łączącej środek podstawy działła położony po lewej stronie ekranu z punktem określającym położenie końca lufy. Kod znajdujący się w linii numer 32 zachowuje wartość barrelAngle, aby kula mogła zostać później wystrzelona pod właściwym kątem (angle).

LISTING 6.7. Metoda align klasy Cannon

```
30 // ustawia lufę działa pod określonym kątem
31 public void align(double barrelAngle) {
32     this.barrelAngle = barrelAngle;
33     barrelEnd.x = (int) (barrelLength * Math.sin(barrelAngle));
34     barrelEnd.y = (int) (-barrelLength * Math.cos(barrelAngle)) +
35         view.getScreenHeight() / 2;
36 }
37
```

6.11.3. Metoda fireCannonball

Metoda `fireCannonball` (patrz listing 6.8) wystrzeliwuje kulę przemieszczającą się po ekranie według bieżącej trajektorii `barrelAngle`. Kod znajdujący się w liniach 41 – 46 oblicza poziome i pionowe składowe prędkości kuli. W liniach 49 – 50 obliczany jest promień kuli, który jest ułamkiem `CannonView.CANNONBALL_RADIUS_PERCENT` wysokości ekranu. Kod znajdujący się w liniach 53 – 56 „ładuje dział” — konstruuje nową kulę i umieszcza ją w działle. Na koniec odtwarzany jest dźwięk wystrzału (linia nr 58).

LISTING 6.8. Metoda `fireCannonball` klasy Cannon

```
38 // metoda tworząca kulę i wystrzeliwująca ją w kierunku wskazywanym przez lufę
39 public void fireCannonball() {
40     // oblicz komponent x prędkości ruchu kuli
41     int velocityX = (int) (CannonView.CANNONBALL_SPEED_PERCENT *
42         view.getScreenWidth() * Math.sin(barrelAngle));
43
44     // oblicz komponent y prędkości ruchu kuli
45     int velocityY = (int) (CannonView.CANNONBALL_SPEED_PERCENT *
46         view.getScreenWidth() * -Math.cos(barrelAngle));
47
48     // oblicz promień kuli
49     int radius = (int) (view.getScreenHeight() *
50         CannonView.CANNONBALL_RADIUS_PERCENT);
51
52     // utwórz kulę i umieść ją wewnątrz działa
53     cannonball = new Cannonball(view, Color.BLACK,
54         CannonView.CANNON_SOUND_ID, -radius,
55         view.getScreenHeight() / 2 - radius, radius, velocityX,
56         velocityY);
57
58     cannonball.playSound(); // odtwarza dźwięk wystrzału
59 }
60
```

6.11.4. Metoda draw

Metoda `draw` (patrz listing 6.9) rysuje działło na ekranie. Działło wyświetlane na ekranie składa się z dwóch komponentów — najpierw rysujemy jego lufę, a później podstawę.

LISTING 6.9. Metoda draw klasy Cannon

```
61 // rysuje działo na obiekcie Canvas
62 public void draw(Canvas canvas) {
63     // rysuje lufę działa
64     canvas.drawLine(0, view.getScreenHeight() / 2, barrelEnd.x,
65         barrelEnd.y, paint);
66
67     // rysuje podstawę działa
68     canvas.drawCircle(0, (int) view.getScreenHeight() / 2,
69         (int) baseRadius, paint);
70 }
71
```

Rysowanie lufy działa za pomocą metody drawLine klasy Canvas

Lufę działa wyświetlamy za pomocą metody drawLine klasy Canvas (linie 64 – 65). Do metody tej przekazywanych jest pięć parametrów — pierwsze cztery określają współrzędne x i y początku i końca linii, a ostatni parametr jest obiektem Paint określającym charakterystyki linii, takie jak np. jej grubość. Przypominamy, że obiekt paint został skonfigurowany w celu rysowania lufy, której grubość jest określana przez konstruktor (patrz listing 6.6, linia nr 25).

Rysowanie podstawy działa za pomocą metody drawCircle klasy Canvas

Kod znajdujący się w liniach 68 – 69 rysuje półokrągłą podstawę działa za pomocą metody drawCircle klasy Canvas. Podstawa działa powstaje w wyniku narysowania okręgu, którego środek leży na lewej krawędzi ekranu. Umieszczenie okręgu zależy od współrzędnych jego środka, a więc połowa tego okręgu zostanie narysowana poza lewą granicą obiektu SurfaceView.

6.11.5. Metody getCannonball i removeCannonball

Listing 6.10 przedstawia metody getCannonball i removeCannonball. Metoda getCannonball (linie 73 – 75) zwraca bieżący egzemplarz obiektu Cannonball znajdujący się w obiekcie Cannon. Przypisanie wartości null zmiennej cannonball oznacza, że na ekranie gry nie ma żadnej kuli. Metoda ta jest używana przez klasę CannonView w celu uniemożliwienia wystrzelenia kolejnej kuli, jeżeli na ekranie znajduje się inna kula (patrz sekcja 6.13.8, listing 6.22). Metoda removeCannonball (linie 78 – 80 listingu 6.10) usuwa kulę z gry, przypisując wartość null zmiennej cannonball. Klasa CannonView korzysta z tej metody w celu usunięcia kuli z gry, gdy ta trafi w cel lub wyleci poza ekran (patrz sekcja 6.13.11, listing 6.25).

LISTING 6.10. Metody getCannonball i removeCannonball klasy CannonView

```
72 // zwraca kulę wystrzeloną przez działo
73 public Cannonball getCannonball() {
74     return cannonball;
75 }
76
77 // usuwa kulę z gry
78 public void removeCannonball() {
79     cannonball = null;
80 }
81 }
```

6.12. Cannonball — podklasa klasy GameElement

Klasa Cannonball, będąca podklasą klasy GameElement (patrz sekcje 6.12.1 – 6.12.4), opisuje kulę wystrzeloną z działa.

6.12.1. Zmienne egzemplarzowe i konstruktor

Do konstruktora klasy Cannonball (patrz listing 6.11) przekazywany jest promień kuli (zmienna radius) — zmienna ta nie była przekazywana do konstruktora klasy GameElement (konstruktor ten otrzymywał zmienne informujące o szerokości (width) i wysokości (height)). Konstruktor otrzymuje również wartość velocityX informującą o szybkości ruchu kuli w płaszczyźnie poziomej, a także wartość velocityY informującą o szybkości w płaszczyźnie pionowej. Kod znajdujący się w linii numer 18 przypisuje zmiennej onScreen wartość logiczną true, ponieważ kula jest właśnie wprowadzana na ekran.

LISTING 6.11. Zmienne egzemplarzowe i konstruktor klasy Cannonball

```
1 // Cannonball.java
2 // Klasa opisująca kulę wystrzelianą przez działo.
3 package com.deitel.cannongame;
4
5 import android.graphics.Canvas;
6 import android.graphics.Rect;
7
8 public class Cannonball extends GameElement {
9     private float velocityX;
10    private boolean onScreen;
11
12    // konstruktor
13    public Cannonball(CannonView view, int color, int soundId, int x,
14        int y, int radius, float velocityX, float velocityY) {
15        super(view, color, soundId, x, y,
16            2 * radius, 2 * radius, velocityY);
17        this.velocityX = velocityX;
18        onScreen = true;
19    }
20
```

6.12.2. Metody getRadius, collidesWith, isOnScreen i reverseVelocityX

Metoda getRadius (patrz listing 6.12, linie 22 – 24) zwraca promień kuli po określeniu połowy odległości pomiędzy granicami kuli (shape.right i shape.left). Metoda isOnScreen (linie 32 – 34) zwraca true, jeżeli kula znajduje się na ekranie.

LISTING 6.12. Metody getRadius, collidesWith, isOnScreen i reverseVelocityX

```
21 // ustal promień kuli
22 private int getRadius() {
23     return (shape.right - shape.left) / 2;
24 }
25
26 // sprawdź, czy kula uderza w dany element GameElement
27 public boolean collidesWith(GameElement element) {
28     return (Rect.intersects(shape, element.shape) && velocityX > 0);
29 }
```



```

29 }
30
31 // metoda zwraca wartość logiczną true, jeżeli kula znajduje się na ekranie
32 public boolean isOnScreen() {
33     return onScreen;
34 }
35
36 // odwróć zwrot kierunku ruchu kuli w płaszczyźnie poziomej
37 public void reverseVelocityX() {
38     velocityX *= -1;
39 }
40

```

Sprawdzanie za pomocą metody `collidesWith`, czy kula zderzyła się z innym elementem `GameElement`

Metoda `collidesWith` (linie 27 – 29) sprawdza, czy kula zderzyła się z danym elementem `GameElement`. Dokonujemy prostej *detekcji zderzenia* na podstawie prostokątnej krawędzi obiektu `Cannonball`. W przypadku zderzenia obiektu `Cannonball` z obiektem `GameElement` muszą wystąpić dwa warunki:

- Granice obiektu `Cannonball`, które są przechowywane w shape `Rect`, muszą przeciąć granice shape danego obiektu `GameElement`. Warunek ten jest sprawdzany przez metodę `intersects` obiektu `Rect`.
- Obiekt `Cannonball` musi przemieszczać się w płaszczyźnie poziomej w kierunku danego obiektu `GameElement`. O ile obiekt `Cannonball` nie uderzy w przeszkodę, to porusza się od lewej do prawej. Jeżeli prędkość pozioma (`velocityX`) jest wartością dodatnią, to znaczy, że kula porusza się w prawo, w kierunku danego obiektu `GameElement`.

Odwracanie kierunku ruchu kuli za pomocą metody `reverseVelocityX`

Metoda `reverseVelocityX` odwraca zwrot kierunku ruchu obiektu `Cannonball`, mnożąc wartość `velocityX` przez `-1`. Jeżeli metoda `collidesWith` zwróci wartość `true`, to metoda `testForCollisions` klasy `CannonView` wywoła metodę `reverseVelocityX`, która odwróci wartość poziomej prędkości — kula zostanie odbita z powrotem w kierunku działa (patrz sekcja 6.13.11).

6.12.3. Metoda `update`

Metoda `update` (patrz listing 6.13) najpierw wywołuje metodę `update` klasy nadrzędnej (linia 44.) w celu zaktualizowania pionowej prędkości obiektu `Cannonball`, a następnie sprawdza, czy doszło do zderzenia w płaszczyźnie pionowej. W 47. linii kodu metoda `offset` obiektu `Rect` jest używana do przesunięcia granic obiektu `Cannonball` w płaszczyźnie poziomej. Pozioma prędkość tego obiektu (`velocityX`) jest mnożona przez ilość czasu, który upłynął (`interval`), w celu określenia, o jaką odległość obiekt ma zostać przesunięty. Kod znajdujący się w liniach 50 – 53 przypisuje zmiennej `onScreen` wartość `false`, jeżeli obiekt `Cannonball` uderzy w którąś z krawędzi ekranu.

LISTING 6.13. Metoda `update` klasy `GameElement`

```

41 // aktualizuje pozycję kuli
42 @Override
43 public void update(double interval) {
44     super.update(interval); // aktualizuje położenie kuli w płaszczyźnie pionowej
45
46     // aktualizuje położenie w płaszczyźnie poziomej
47     shape.offset((int) (velocityX * interval), 0);

```

```
48
49 // jeżeli kula znajdzie się poza ekranem
50 if (shape.top < 0 || shape.left < 0 ||
51     shape.bottom > view.getScreenHeight() ||
52     shape.right > view.getScreenWidth())
53     onScreen = false; // przypisz wartość pozwalającą na usunięcie kuli
54 }
55
```

6.12.4. Metoda draw

Metoda draw (patrz listing 6.14) zdefiniowana w klasie `GameElement` korzysta z metody `drawCircle` obiektu `Canvas` w celu narysowania obiektu `Cannonball` w obecnym położeniu. Pierwsze dwa argumenty to współrzędne środka okręgu. Trzeci argument definiuje promień okręgu. Ostatni argument jest obiektem `Paint` określającym charakterystyki wyświetlania okręgu.

LISTING 6.14. Przeźdefiniowana metoda draw klasy `GameElement`

```
56 // rysuje obiekt Cannonball na danym obiekcie canvas
57 @Override
58 public void draw(Canvas canvas) {
59     canvas.drawCircle(shape.left + getRadius(),
60                      shape.top + getRadius(), getRadius(), paint);
61 }
62 }
```

6.13. Klasa CannonView — podklasa SurfaceView

Klasa `CannonView` (patrz listingi 6.15 – 6.29) jest zmodyfikowaną podklasą klasy `View`, która implementuje logikę gry *Cannon Game* i odpowiada za wyświetlanie na ekranie jej elementów.

6.13.1. Instrukcje package i import

Listing 6.15 zawiera instrukcje `package` i `import` klasy `CannonView`. W sekcji 6.3 opisano najważniejsze nowe klasy i interfejsy używane przez klasę `CannonView`. Zostały one wyróżnione w listingu 6.15.

LISTING 6.15. Instrukcje package i import klasy `CannonView`

```
1 // CannonView.java
2 // Wyświetla elementy gry Cannon Game i decyduje o ich zachowaniu.
3 package com.deitel.cannongame;
4
5 import android.app.Activity;
6 import android.app.AlertDialog;
7 import android.app.Dialog;
8 import android.app.DialogFragment;
9 import android.content.Context;
10 import android.content.DialogInterface;
11 import android.graphics.Canvas;
12 import android.graphics.Color;
13 import android.graphics.Paint;
14 import android.graphics.Point;
```

```

15 import android.media.AudioAttributes;
16 import android.media.SoundPool;
17 import android.os.Build;
18 import android.os.Bundle;
19 import android.util.AttributeSet;
20 import android.util.Log;
21 import android.util.SparseIntArray;
22 import android.view.MotionEvent;
23 import android.view.SurfaceHolder;
24 import android.view.SurfaceView;
25 import android.view.View;
26
27 import java.util.ArrayList;
28 import java.util.Random;
29
30 public class CannonView extends SurfaceView
31     implements SurfaceHolder.Callback {
32

```

6.13.2. Stałe i zmienne egzemplarzowe

Listing 6.16 przedstawia obszerny zbiór stałych i zmiennych egzemplarzowych klasy CannonView. Zostaną one opisane w dalszej części tego rozdziału, gdy będziemy z nich korzystać. Wiele stałych jest używanych w obliczeniach związanych ze skalowaniem rozmiaru elementów gry na podstawie wymiarów ekranu.

LISTING 6.16. Zmienne statyczne i egzemplarzowe klasy CannonView

```

33 private static final String TAG = "CannonView"; // do logowania błędów
34
35 // stałe rozgrywki
36 public static final int MISS_PENALTY = 2; // liczba sekund odejmowana za trafienie w przeszkodę
37 public static final int HIT_REWARD = 3; // liczba sekund odejmowana za trafienie w cel
38
39 // stałe działa
40 public static final double CANNON_BASE_RADIUS_PERCENT = 3.0 / 40;
41 public static final double CANNON_BARREL_WIDTH_PERCENT = 3.0 / 40;
42 public static final double CANNON_BARREL_LENGTH_PERCENT = 1.0 / 10;
43
44 // stałe kuli
45 public static final double CANNONBALL_RADIUS_PERCENT = 3.0 / 80;
46 public static final double CANNONBALL_SPEED_PERCENT = 3.0 / 2;
47
48 // stałe celów
49 public static final double TARGET_WIDTH_PERCENT = 1.0 / 40;
50 public static final double TARGET_LENGTH_PERCENT = 3.0 / 20;
51 public static final double TARGET_FIRST_X_PERCENT = 3.0 / 5;
52 public static final double TARGET_SPACING_PERCENT = 1.0 / 60;
53 public static final double TARGET_PIECES = 9;
54 public static final double TARGET_MIN_SPEED_PERCENT = 3.0 / 4;
55 public static final double TARGET_MAX_SPEED_PERCENT = 6.0 / 4;
56
57 // stałe przeszkody
58 public static final double BLOCKER_WIDTH_PERCENT = 1.0 / 40;
59 public static final double BLOCKER_LENGTH_PERCENT = 1.0 / 4;

```

```

60 public static final double BLOCKER_X_PERCENT = 1.0 / 2;
61 public static final double BLOCKER_SPEED_PERCENT = 1.0;
62
63 // tekst o rozmiarze 1/18 szerokości ekranu
64 public static final double TEXT_SIZE_PERCENT = 1.0 / 18;
65
66 private CannonThread cannonThread; // steruje pętlą gry
67 private Activity activity; // do wyświetlenia komunikatu kończącego grę w wątku graficznego interfejsu użytkownika
68 private boolean dialogIsDisplayed = false;
69
70 // obiekty gry
71 private Cannon cannon;
72 private Blocker blocker;
73 private ArrayList<Target> targets;
74
75 // zmienne określające wymiary
76 private int screenWidth;
77 private int screenHeight;
78
79 // zmienne pętli gry i zmienne przeznaczone do śledzenia statystyk gry
80 private boolean gameOver; // Czy gra się skończyła?
81 private double timeLeft; // pozostały czas gry wyrażony w sekundach
82 private int shotsFired; // liczba wykonanych wystrzałów
83 private double totalElapsedTime; // wyrażony w sekundach czas, który upłynął od rozpoczęcia gry
84
85 // stałe i zmienne zarządzające dźwiękami
86 public static final int TARGET_SOUND_ID = 0;
87 public static final int CANNON_SOUND_ID = 1;
88 public static final int BLOCKER_SOUND_ID = 2;
89 private SoundPool soundPool; // odtwarza efekty dźwiękowe
90 private SparseIntArray soundMap; // mapuje identyfikatory ID do obiektu SoundPool
91
92 // zmienne Paint używane do wyświetlania każdego elementu na ekranie
93 private Paint textPaint; // zmienna Paint używana do wyświetlenia tekstu
94 private Paint backgroundPaint; // zmienna Paint używana do czyszczenia obszaru wyświetlania
95

```

6.13.3. Konstruktor

Listing 6.17 przedstawia kod konstruktora klasy CannonView. Gdy obiekt View jest przygotowywany do wyświetlenia, wywoływany jest jego konstruktor, a jako argumenty przekazywane są do niego obiekty Context i AttributeSet. Obiekt Context jest obiektem Activity wyświetlającym obiekt MainActivityFragment zawierający obiekty CannonView i AttributeSet (pakiet *android.util*). Ostatni z tych obiektów zawiera wartości atrybutów CannonView, które są definiowane w dokumencie XML rozkładu. Argumenty te są przekazywane do konstruktora klasy nadrzędnej (linia 96.) w celu zapewnienia właściwej konfiguracji zmodyfikowanego obiektu View parametrami standardowego obiektu View zdefiniowanymi w kodzie XML. Kod znajdujący się w linii 99. przechowuje odwołanie do klasy MainActivity, dzięki czemu możemy z niej skorzystać na koniec gry w celu wyświetlenia okna AlertDialog za pomocą wątku graficznego interfejsu użytkownika. Zdecydowaliśmy się na przechowywanie odwołania do obiektu Activity, a więc możemy zawsze uzyskać do niego dostęp za pomocą metody getContext dziedziczonej z klasy View.

LISTING 6.17. Konstruktor klasy CannonView

```
96 // konstruktor
97 public CannonView(Context context, AttributeSet attrs) {
98     super(context, attrs); // wywołaj konstruktor klasy nadrzędnej
99     activity = (Activity) context; // zachowaj odwołanie do MainActivity
100
101     // zarejestruj obiekt nasłuchujący SurfaceHolder.Callback
102     getHolder().addCallback(this);
103
104     // skonfiguruj atrybuty dźwięku pod kątem dźwięków odtwarzanych przez grę
105     AudioAttributes.Builder attrBuilder = new AudioAttributes.Builder();
106     attrBuilder.setUsage(AudioAttributes.USAGE_GAME);
107
108     // zainicjuj obiekt SoundPool w celu odtwarzania trzech dźwięków aplikacji
109     SoundPool.Builder builder = new SoundPool.Builder();
110     builder.setMaxStreams(1);
111     builder.setAudioAttributes(attrBuilder.build());
112     soundPool = builder.build();
113
114     // stwórz obiekt Map zawierający dźwięki i załaduj dźwięki do tego obiektu
115     soundMap = new SparseIntArray(3); // tworzy nową tablicę SparseIntArray
116     soundMap.put(TARGET_SOUND_ID,
117         soundPool.load(context, R.raw.target_hit, 1));
118     soundMap.put(CANNON_SOUND_ID,
119         soundPool.load(context, R.raw.cannon_fire, 1));
120     soundMap.put(BLOCKER_SOUND_ID,
121         soundPool.load(context, R.raw.blocker_hit, 1));
122
123     textPaint = new Paint();
124     backgroundPaint = new Paint();
125     backgroundPaint.setColor(Color.WHITE);
126 }
127
```

Rejestrowanie obiektu nasłuchującego SurfaceHolder.Callback

Kod znajdujący się w linii 102 rejestruje obiekt `this` (tj. obiekt `CannonView`) jako `SurfaceHolder.Callback` — obiekt, do którego kierowane są wywołania metody, gdy obiekt `SurfaceView` jest *tworzony*, *aktualizowany* lub *usuwany*. Metoda `getHolder` odziedziczona po obiekcie `SurfaceView` zwraca obiekt `SurfaceHolder` przeznaczony do zarządzania obiektem `SurfaceView`, a metoda `addCallback` obiektu `SurfaceHolder` przechowuje obiekt, który implementuje interfejs `SurfaceHolder.Callback`.

Konfiguracja obiektu SoundPool i ładowanie dźwięków

Kod znajdujący się w liniach o numerach 105 – 121 konfiguruje dźwięki używane przez aplikację. Najpierw tworzymy obiekt `AudioAttributes.Builder` (linia 105.), a następnie wywołujemy metodę `setUsage` (linia 106.), do której przekazywana jest stała informująca o przeznaczeniu dźwięku. W tej aplikacji korzystamy ze stałej `AudioAttributes.USAGE_GAME`, która informuje, że dźwięk jest używany przez grę. Następnie tworzymy obiekt `SoundPool.Builder` (linia 109.), który pozwoli na stworzenie obiektu `SoundPool`, dzięki któremu załadujemy i odtworzymy dźwięki aplikacji. Następnie wywołujemy metodę `setMaxStreams` obiektu `SoundPool.Builder` (linia 110.), która przyjmuje argument określający maksymalną liczbę jednocześnie odtwarzanych strumieni dźwiękowych. Nie będziemy w tym samym czasie odtwarzali więcej niż jednego dźwięku,

a więc przekazujemy wartość 1. Niektóre bardziej rozbudowane gry mogą odtwarzać jednocześnie wiele dźwięków. Na koniec wywołujemy metodę `setAudioAttributes` obiektu `AudioAttributes.Builder` (linia 111.), która pozwoli na użycie atrybutów dźwięku przez utworzony w przyszłości obiekt `SoundPool`.

W linii 115. tworzona jest tablica `SparseIntArray` (`soundMap`), która mapuje klucze będące wartościami całkowitoliczbowymi do wartości całkowitoliczbowych. Tablica `SparseIntArray` jest podobna do tablicy `HashMap<Integer, Integer>`, ale w przypadku małej liczby par klucz-wartość jest ona bardziej wydajna. Mapujemy klucze dźwięków (zdefiniowane w liniach 86 – 88 listingu 6.16) do identyfikatorów załadowanych dźwięków, które są określone za pomocą wartości zwróconych przez metodę `load` obiektu `SoundPool` wywołaną w liniach o numerach 117, 119 i 121 listingu 6.17. Identyfikator każdego z dźwięków może być użyty do jego *odtworzenia*, a później również do zwrócenia tego zasobu do systemu. Metoda `load` obiektu `SoundPool` otrzymuje trzy argumenty — kontekst aplikacji (`Context`), identyfikator zasobu odwołujący się do pliku dźwiękowego, który należy załadować, i priorytet tego dźwięku. Zgodnie z dokumentacją tej metody ostatni argument nie jest obecnie do niczego stosowany i powinien przyjmować wartość 1.

Tworzenie obiektów `Paint` używanych do wyświetlania tła i tekstu czasomierza

Kod znajdujący się w liniach 123 – 124 tworzy obiekty `Paint` używane do wyświetlania tła gry i tekstu informującego gracza o ilości czasu do zakończenia gry. Domyślnym kolorem tekstu jest kolor czarny, a linia 125. ustawia biały kolor tła.

6.13.4. Przepdefiniowywanie metody `onSizeChanged` klasy `View`

Kod znajdujący się w listingu 6.18 przepdefiniowuje metodę `onSizeChanged` klasy `View`, która jest wywoływana za każdym razem, gdy zmieni się rozmiar obiektu `View`. Jest ona wywoływana również wtedy, gdy obiekt `View` jest dodawany do hierarchii obiektów tego typu podczas przygotowywania rozkładu do wyświetlenia na ekranie. Opisująca aplikacja zawsze działa w orientacji poziomej, a więc metoda `onSizeChanged` jest wywoływana tylko wtedy, gdy metoda `onCreate` aktywności przygotowuje do wyświetlenia graficzny interfejs użytkownika. Metoda ta otrzymuje nową szerokość i wysokość obiektu `View`, a także jego poprzednią wysokość i szerokość. Gdy metoda ta jest wywoływana po raz pierwszy, poprzednia wysokość i szerokość wynoszą 0. Kod znajdujący się w liniach 138 – 139 konfiguruje obiekt `textPaint`, który jest używany do wyświetlania tekstu informującego gracza, ile czasu pozostało do końca gry. Linia 138. definiuje rozmiar tekstu na ułamek `TEXT_SIZE_PERCENT` wysokości ekranu (`screenHeight`). Wartość `TEXT_SIZE_PERCENT`, a także inne współczynniki skalowania znajdujące się w listingu 6.16 zostały określone metodą prób i błędów, tak aby elementy gry wyglądały ładnie i czytelnie po wyświetleniu na ekranie.

LISTING 6.18. Przepdefiniowywanie metody `onSizeChanged` klasy `View`

```
128 // metoda wywoływana przy zmianie rozmiaru obiektu SurfaceView,
129 // do zmiany takiej dochodzi, gdy jest on dodawany po raz pierwszy do hierarchii obiektów View
130 @Override
131 protected void onSizeChanged(int w, int h, int oldw, int oldh) {
132     super.onSizeChanged(w, h, oldw, oldh);
133
134     screenWidth = w; // zapisz szerokość obiektu CannonView
135     screenHeight = h; // zapisz wysokość obiektu CannonView
136
137     // konfiguruj właściwości tekstu
138     textPaint.setTextSize((int) (TEXT_SIZE_PERCENT * screenHeight));
139     textPaint.setAntiAlias(true); // wygładza tekst
140 }
141
```

6.13.5. Metody getWidth, getHeight i playSound

Metody getWidth i getHeight zwracają szerokość i wysokość ekranu (patrz listing 6.19), które są aktualizowane przez metodę onSizeChanged (patrz listing 6.18). Metoda playSound korzystająca z metody play obiektu soundPool (linie 153 – 155) odtwarza dźwięk obiektu soundMap o danym identyfikatorze soundId, który został skojarzony z dźwiękiem w momencie tworzenia obiektu soundMap (patrz linie 113 – 119 listingu 6.17). Identyfikator soundId jest używany jako klucz soundMap służący do zlokalizowania identyfikatora dźwięku w obiekcie SoundPool. Obiekt klasy GameElement może wywołać metodę playSound w celu odtworzenia skojarzonego z nim dźwięku.

LISTING 6.19. Metody getWidth, getHeight i playSound klasy CannonView

```
142 // ustal szerokość ekranu gry
143 public int getWidth() {
144     return screenWidth;
145 }
146
147 // ustal wysokość ekranu gry
148 public int getHeight() {
149     return screenHeight;
150 }
151
152 // odtwarza dźwięk o identyfikatorze soundId w obiekcie soundMap
153 public void playSound(int soundId) {
154     soundPool.play(soundMap.get(soundId), 1, 1, 1, 0, 1f);
155 }
156
```

6.13.6. Metoda newGame

Metoda newGame (patrz listing 6.20) przywraca pierwotne wartości zmiennym, które są używane do sterowania grą. Kod znajdujący się w liniach 160 – 163 tworzy nowe działo charakteryzujące się:

- podstawą o średnicy będącej ułamkiem CANNON_BASE_RADIUS_PERCENT wysokości ekranu,
- lufą o długości będącej ułamkiem CANNON_BARREL_LENGTH_PERCENT szerokości ekranu,
- lufą o szerokości będącej ułamkiem CANNON_BARREL_WIDTH_PERCENT wysokości ekranu.

LISTING 6.20. Metoda newGame klasy CannonView

```
157 // przywraca początkowe wartości wszystkim obiektom widocznym na ekranie i uruchamia nową grę
158 public void newGame() {
159     // skonstruuj nowe działo
160     cannon = new Cannon(this,
161         (int) (CANNON_BASE_RADIUS_PERCENT * screenHeight),
162         (int) (CANNON_BARREL_LENGTH_PERCENT * screenWidth),
163         (int) (CANNON_BARREL_WIDTH_PERCENT * screenHeight));
164
165     Random random = new Random(); // do określania losowych prędkości
166     targets = new ArrayList<>(); // utwórz nową listę celów
167
168     // inicjuj współrzędną targetX dla pierwszego celu, licząc od lewej
169     int targetX = (int) (TARGET_FIRST_X_PERCENT * screenWidth);
170
```

```

171 // oblicz współrzędną y celów
172 int targetY = (int) ((0.5 - TARGET_LENGTH_PERCENT / 2) *
173     screenHeight);
174
175 // dodaj cele TARGET_PIECES do listy celów
176 for (int n = 0; n < TARGET_PIECES; n++) {
177
178     // określ losową prędkość celu
179     // znajdującą się w podanym zakresie
180     double velocity = screenHeight * (random.nextDouble() *
181         (TARGET_MAX_SPEED_PERCENT - TARGET_MIN_SPEED_PERCENT) +
182         TARGET_MIN_SPEED_PERCENT);
183
184     // zmieniaj kolory celów — przełączaj pomiędzy kolorem jasnym i ciemnym
185     int color = (n % 2 == 0) ?
186         getResources().getColor(R.color.dark,
187             getContext().getTheme()) :
188         getResources().getColor(R.color.light,
189             getContext().getTheme());
190
191     velocity *= -1; // odwróć początkową wartość prędkości kolejnego celu
192
193     // utwórz nowy cel i dodaj go do listy
194     targets.add(new Target(this, color, HIT_REWARD, targetX, targetY,
195         (int) (TARGET_WIDTH_PERCENT * screenWidth),
196         (int) (TARGET_LENGTH_PERCENT * screenHeight),
197         (int) velocity));
198
199     // zmień wartość współrzędnej x, przechodząc
200     // do kolejnego celu znajdującego się po prawej stronie
201     targetX += (TARGET_WIDTH_PERCENT + TARGET_SPACING_PERCENT) *
202         screenWidth;
203 }
204
205 // utwórz nową przeszkodę
206 blocker = new Blocker(this, Color.BLACK, MISS_PENALTY,
207     (int) (BLOCKER_X_PERCENT * screenWidth),
208     (int) ((0.5 - BLOCKER_LENGTH_PERCENT / 2) * screenHeight),
209     (int) (BLOCKER_WIDTH_PERCENT * screenWidth),
210     (int) (BLOCKER_LENGTH_PERCENT * screenHeight),
211     (float) (BLOCKER_SPEED_PERCENT * screenHeight));
212
213 timeLeft = 10; // rozpocznij odliczanie od 10 sekund
214
215 shotsFired = 0; // ustaw początkową liczbę oddanych strzałów
216 totalElapsedTime = 0.0; // wyzeruj wartość określającą czas, jaki upłynął od rozpoczęcia gry
217
218 if (gameOver) { // uruchom nową grę po zakończeniu poprzedniej
219     gameOver = false; // gra jeszcze się nie skończyła
220     cannonThread = new CannonThread(getHolder()); // utwórz wątek
221     cannonThread.start(); // uruchom wątek pętli gry
222 }
223
224 hideSystemBars();
225 }
226

```


Kod znajdujący się w linii 165. tworzy nowy obiekt `Random`, który jest używany podczas losowania prędkości obiektów `Target`. W linii 166. tworzona jest tablica `ArrayList` zawierająca obiekty `Target`. Linia 169 inicjalizuje zmienną `targetX` liczbą pikseli znajdujących się po lewej stronie, licząc od miejsca, w którym zostanie umieszczony pierwszy obiekt `Target`. Pierwszy obiekt `Target` jest umieszczony w miejscu określanym przez `TARGET_FIRST_X_PERCENT`. W liniach 172 – 173 zmienna `targetY` jest inicjowana wartością używaną do wyśrodkowania wszystkich obiektów `Target` w płaszczyźnie pionowej ekranu. Kod znajdujący się w liniach 176 – 203 konstruuje dziewięć nowych obiektów celów `TARGET_PIECES` i dodaje je do obiektu `targets`. Linie 180 – 182 wybierają losowo prędkość nowego obiektu typu `Target` — prędkość przybiera wartość pomiędzy ułamekami procentowymi wysokości ekranu: `TARGET_MIN_SPEED_PERCENT` i `TARGET_MAX_SPEED_PERCENT`. Kod znajdujący się w liniach 185 – 189 sprawia, że kolor nowego obiektu `Target` będzie wybierany spośród listy dwóch kolorów: `R.color.dark` i `R.color.light`, a jego prędkość będzie przybierała naprzemiennie wartość dodatnią i ujemną. Linia 191. odwraca prędkość każdego nowego celu, aby niektóre cele poruszały się do góry, a inne do dołu. Nowy obiekt `Target` po skonstruowaniu jest dodawany do obiektu `targets` (linie 194 – 197). Szerokość obiektu `Target` jest równa ułaskowi `TARGET_WIDTH_PERCENT` szerokości ekranu, a jego wysokość jest równa ułaskowi `TARGET_HEIGHT_PERCENT` wysokości ekranu. Na koniec zmienna `targetX` jest inkrementowana do pozycji kolejnego obiektu `Target`.

Nowy obiekt `Blocker` po skonstruowaniu jest przechowywany w obiekcie `blocker` (linie 206 – 211). Obiekt `Blocker` jest umieszczany w odległości będącej ułaskiem `BLOCKER_X_PERCENT` szerokości ekranu od jego lewej krawędzi i na początku gry jest wyśrodkowywany w płaszczyźnie pionowej. Szerokość obiektu `Blocker` jest równa ułaskowi `BLOCKER_WIDTH_PERCENT` szerokości ekranu, a jego wysokość jest równa ułaskowi `BLOCKER_HEIGHT_PERCENT` wysokości ekranu. Prędkość tego obiektu jest równa ułaskowi `BLOCKER_SPEED_PERCENT` wysokości ekranu.

Jeżeli zmiennej `gameOver` przypisano wartość logiczną `true` (może do tego dojść tylko *po* zakończeniu gry), to kod znajdujący się w linii 219. przywraca wartość początkową tej zmiennej, a linie 220 – 221 tworzą nowy wątek `CannonThread` i wywołują jego metodę `start`, co powoduje uruchomienie *pętli* gry sterującej jej pracą. Linia 224. wywołuje metodę `hideSystemBars` (patrz sekcja 6.13.16), która aktywuje tryb pełnoekranowy — ukrywa paski systemowe i pozwala użytkownikowi na ich wyświetlenie poprzez przesunięcie palcem po ekranie od góry.

6.13.7. Metoda `updatePositions`

Metoda `updatePositions` (patrz listing 6.21) jest wywoływana przez metodę `run` wątku `CannonThread` (patrz sekcja 6.13.15) w celu zaktualizowania pozycji elementów wyświetlonych na ekranie i przeprowadzenia operacji prostego *wykrywania kolizji*. Nowe miejsca położenia elementów są obliczane na podstawie czasu wyrażonego w milisekundach, który upłynął pomiędzy wyświetleniem poprzedniej klatki a rozpoczęciem tworzenia obecnej klatki. Pozwala to grze na aktualizowanie odległości przesunięcia jej elementów na podstawie *częstotliwości odświeżania ekranu*. Zagadnienie to zostanie opisane w sposób bardziej szczegółowy w sekcji 6.13.15, gdy będziemy omawiać *pętlę* gry.

LISTING 6.21. Metoda `updatePositions` klasy `CannonView`

```
227 // metoda wywoływana wielokrotnie przez wątek CannonThread w celu aktualizacji położenia elementów gry
228 private void updatePositions(double elapsedTimeMS) {
229     double interval = elapsedTimeMS / 1000.0; // zamień na sekundy
230
231     // jeżeli kula jest wyświetlana na ekranie, to aktualizuj jej położenie
232     if (cannon.getCannonball() != null)
233         cannon.getCannonball().update(interval);
234
235     blocker.update(interval); // aktualizuj położenie przeszkody
236 }
```

```

237     for (GameElement target : targets)
238         target.update(interval); // aktualizuj położenie celów
239
240     timeLeft -= interval; // odejmuj od czasu, który pozostał
241
242     // jeżeli stoper osiągnął zero
243     if (timeLeft <= 0) {
244         timeLeft = 0.0;
245         gameOver = true; // gra jest zakończona
246         cannonThread.setRunning(false); // zakończ wątek
247         showGameOverDialog(R.string.lose); // pokaż komunikat informujący o przegranej
248     }
249
250     // jeżeli wszystkie cele zostały trafione
251     if (targets.isEmpty()) {
252         cannonThread.setRunning(false); // zakończ wątek
253         showGameOverDialog(R.string.win); // pokaż komunikat informujący o wygranej
254         gameOver = true;
255     }
256 }
257

```

Czas, jaki upłynął od utworzenia poprzedniej klatki animacji

Kod znajdujący się w linii 229. dokonuje konwersji wartości określającej ilość czasu, jaki minął od wyświetlenia ostatniej klatki animacji, z milisekund na sekundy. Wartość ta jest używana do modyfikacji położenia różnych elementów gry.

Aktualizacja położenia kuli, przeszkody i celów

W celu zaktualizowania położenia obiektów `GameElement` kod znajdujący się w liniach 232 – 238 wywołuje metody `update` obiektu `Cannonball` (jeżeli kula jest wyświetlana na ekranie), `Blocker` i wyświetlanych na ekranie obiektów `Target`. Do metody `update` przekazywany jest czas, jaki upłynął od wyświetlenia poprzedniej klatki. Dzięki temu elementy widoczne na ekranie mogą zostać przesunięte na prawidłową odległość.

Aktualizacja czasu pozostałego do końca gry i sprawdzanie, czy wyznaczony czas gry nie upłynął

Wartość `timeLeft` jest zmniejszana o ilość czasu, jaki upłynął od wyświetlenia poprzedniej klatki animacji (linia 240.). Jeżeli wartość zmiennej `timeLeft` osiągnie zero, to gra ulega zakończeniu, a zmiennej tej zostanie przypisana wartość 0,0 (zapobiega to wyświetlaniu ujemnej wartości czasu). Następnie zmiennej `gameOver` przypisywana jest wartość logiczna `true`, a wątek `CannonThread` jest zamykany w wyniku wywołania metody `setRunning` z argumentem `false`. Na koniec wywoływana jest metoda `showGameOverDialog`, do której przekazywany jest łańcuch będący identyfikatorem zasobu komunikatu o przegranej.

6.13.8. Metoda `alignAndFireCannonball`

W momencie dotknięcia ekranu przez użytkownika metoda `onTouchEvent` (patrz sekcja 6.13.14) wywołuje metodę `alignAndFireCannonball` (patrz listing 6.22). Kod znajdujący się w liniach 267 – 272 oblicza kąt (`angle`) niezbędny do wycelowania lufy działa w kierunku miejsca dotknięcia ekranu. W linii 275. wywoływana jest metoda `align` obiektu `Cannon`, która ustawia lufę działa pod właściwym kątem. Na koniec (gdy na ekranie znajduje się obiekt `Cannonball`) kod znajdujący się w liniach 280 – 281 wystrzeliwuje kulę i dokonuje inkrementacji zmiennej `shotsFired` określającej liczbę oddanych strzałów.

LISTING 6.22. Metoda alignAndFireCannonball klasy CannonView

```
258 // metoda ustawiająca lufę działa pod właściwym kątem i dokonująca wystrzału
259 // jeżeli kula wystrzelona wcześniej nie znajduje się już na ekranie
260 public void alignAndFireCannonball(MotionEvent event) {
261     // ustal lokalizację dotknięcia widoku
262     Point touchPoint = new Point((int) event.getX(),
263         (int) event.getY());
264
265     // oblicz odległość dotknięcia od środka ekranu
266     // w płaszczyźnie osi y
267     double centerMinusY = (screenHeight / 2 - touchPoint.y);
268
269     double angle = 0; // przypisz kątowi wartość 0
270
271     // oblicz kąt pomiędzy lufą a osią x
272     angle = Math.atan2(touchPoint.x, centerMinusY);
273
274     // skieruj działo w kierunku punktu dotknięcia ekranu
275     cannon.align(angle);
276
277     // wystrzel kulę, jeżeli wystrzelona wcześniej kula nie jest już wyświetlana na ekranie
278     if (cannon.getCannonball() == null ||
279         !cannon.getCannonball().isOnScreen()) {
280         cannon.fireCannonball();
281         ++shotsFired;
282     }
283 }
284
```

6.13.9. Metoda showGameOverDialog

Po zakończeniu gry metoda showGameOverDialog (patrz listing 6.23) wyświetla obiekt DialogFragment (taką technikę opisaliśmy w sekcji 4.7.10) zawierający okno AlertDialog informujące gracza o wygranej lub przegranej, liczbie oddanych strzałów i czasie trwania gry. Wywołanie metody setPositiveButton (linie 301 – 311) powoduje utworzenie przycisku uruchamiającego nową grę.

LISTING 6.23. Metoda showGameOverDialog klasy CannonView

```
285 // wyświetl okno AlertDialog, gdy gra zostanie zakończona
286 private void showGameOverDialog(final int messageId) {
287     // obiekt DialogFragment wyświetlający wynik gry i mogący uruchomić kolejną grę
288     final DialogFragment gameResult =
289         new DialogFragment() {
290             // utwórz obiekt AlertDialog i go zwróć
291             @Override
292             public Dialog onCreateDialog(Bundle bundle) {
293                 // utwórz okno wyświetlające łańcuch messageId
294                 AlertDialog.Builder builder =
295                     new AlertDialog.Builder(getActivity());
296                 builder.setTitle(getResources().getString(messageId));
297
298                 // wyświetl liczbę oddanych strzałów i całkowity czas gry
299                 builder.setMessage(getResources().getString(
```

```

300         R.string.results_format, shotsFired, totalElapsedTime));
301     builder.setPositiveButton(R.string.reset_game,
302         new DialogInterface.OnClickListener() {
303         // metoda wywoływana po wciśnięciu przez użytkownika przycisku Uruchom ponownie
304         @Override
305         public void onClick(DialogInterface dialog,
306             int which) {
307             dialogIsDisplayed = false;
308             newGame(); // przygotuj i uruchom nową grę
309         }
310     });
311 );
312
313     return builder.create(); // zwróć obiekt AlertDialog
314 }
315 };
316
317 // wyświetl obiekt DialogFragment za pomocą menedżera FragmentManager w wątku interfejsu użytkownika
318 activity.runOnUiThread(
319     new Runnable() {
320     public void run() {
321         showSystemBars(); // wyjdź z trybu pełnoekranowego
322         dialogIsDisplayed = true;
323         gameResult.setCancelable(false); // okno modalne
324         gameResult.show(activity.getFragmentManager(), "results");
325     }
326 }
327 );
328 }
329

```

Metoda `onClick` obiektu nasłuchującego zdarzeń przycisku zamyka okno i wywołuje metodę `newGame` uruchamiającą nową grę. Okno musi zostać wyświetlone za pomocą wątku graficznego interfejsu użytkownika, a więc kod znajdujący się w liniach 318 – 327 wywołuje metodę `runOnUiThread` klasy `Activity` w celu określenia obiektu `Runnable`, który powinien zostać wykonany jak najszybciej w wątku graficznego interfejsu użytkownika. Argument jest obiektem anonimowej klasy wewnętrznej implementującej obiekt `Runnable`. Metoda `run` obiektu `Runnable` wywołuje metodę `showSystemBars` (patrz sekcja 6.13.16), która wyłącza tryb pełnoekranowy, a następnie wyświetla okno dialogowe.

6.13.10. Metoda `drawGameElements`

Metoda `drawGameElements` (patrz listing 6.24) wyświetla obiekty `Cannon`, `Cannonball`, `Blocker` i `Targets` w widoku `SurfaceView` za pomocą obiektów `Canvas` uzyskanych przez wątek `CannonThread` (patrz sekcja 6.13.15) z obiektu `SurfaceHolder` widoku `SurfaceView`.

LISTING 6.24. Metoda `drawGameElements` klasy `CannonView`

```

330 // rysuje grę na danym obiekcie Canvas
331 public void drawGameElements(Canvas canvas) {
332     // wyczyść tło
333     canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(),
334         backgroundPaint);
335

```

```

336 // wyświetl pozostałe elementy
337 canvas.drawText(getResources().getString(
338     R.string.time_remaining_format, timeLeft), 50, 100, textPaint);
339
340 cannon.draw(canvas); // rysuj działo
341
342 // narysuj obiekty typu GameElement
343 if (cannon.getCannonball() != null &&
344     cannon.getCannonball().isOnScreen())
345     cannon.getCannonball().draw(canvas);
346
347 blocker.draw(canvas); // narysuj przeszkodę
348
349 // narysuj wszystkie obiekty Target
350 for (GameElement target : targets)
351     target.draw(canvas);
352 }
353

```

Czyszczenie obiektu Canvas za pomocą metody drawRect

Ma początek wywoływana jest metoda `drawRect` obiektu `Canvas` (linie 333 – 334), która czyści obiekt `Canvas`, umożliwiając wyświetlenie elementów gry w nowych pozycjach. Metoda ta otrzymuje współrzędne x i y górnego lewego rogu prostokąta, jego szerokość i wysokość, a także obiekt `Paint` określający charakterystyki wyświetlanego obiektu — przypominamy, że parametr `backgroundPaint` określa, że tło jest koloru białego.

Wyświetlanie czasu pozostałego do końca gry za pomocą metody drawText obiektu Canvas

Następnie wywoływana jest metoda `drawText` obiektu `Canvas` (linie 337 – 338) wyświetlająca ilość czasu pozostałego do końca gry. Metodzie tej jako argument przesyłamy łańcuch, który ma zostać wyświetlony, współrzędne x i y , pod którymi komunikat ma zostać wyświetlony, i obiekt `textPaint` (patrz linie 138 – 139 listingu 6.18) określający wygląd wyświetlanego tekstu (rozmiar czcionki, jej kolor i inne atrybuty).

Wyświetlanie obiektów Cannon, Cannonball, Blocker i Target za pomocą metody draw

Kod znajdujący się w liniach 339 – 350 rysuje obiekty `Cannon`, `Cannonball` (jeżeli kula znajduje się na ekranie), `Blocker` i każdy z obiektów `Target`. Każdy z tych obiektów jest rysowany w wyniku wywołania metody `draw` obiektu i przekazania do obiektu `canvas`.

6.13.11. Metoda testForCollisions

Metoda `testForCollisions` (patrz listing 6.25) sprawdza, czy obiekt `Cannonball` zderza się z którymsz z obiektów `Target` lub z obiektem `Blocker`, a w razie wystąpienia takiej kolizji stosuje odpowiednie efekty. Kod znajdujący się w liniach 359 – 360 sprawdza, czy na ekranie znajduje się obiekt `Cannonball`. Jeżeli obiekt ten znajduje się na ekranie, to linia 362. wywołuje metodę `collidesWith` obiektu `Cannonball` sprawdzającą, czy obiekt ten nie zderza się z obiektem `Target`. Jeżeli dochodzi do takiej kolizji, to kod znajdujący się w linii 363. wywołuje metodę `playSound` obiektu `Target`, która odtwarza dźwięk trafienia w cel, a 366. linia kodu inkrementuje zmienną `timeLeft`, zwiększając limit czasu — nagradza użytkownika

za trafienie w cel. Kod znajdujący się w liniach 368 – 369 usuwa z ekranu obiekty Cannonball i Target. Linia numer 370 zawiera kod zmniejszający wartość n — dzięki temu możemy mieć pewność, że cel znajdujący się obecnie w pozycji n zostanie sprawdzony pod kątem wystąpienia zderzenia. Kod w 376. linii usuwa obiekt Cannonball, jeżeli znajdzie się on poza ekranem. Jeżeli obiekt Cannonball wciąż znajduje się na ekranie, linie 380 – 381 wywołują ponownie metodę collidesWith, aby sprawdzić, czy nie zderzył się on z przeszkodą. Jeżeli dojdzie do takiego zderzenia, to kod znajdujący się w linii 382. wywoła metodę playSound obiektu Blocker odtwarzającą dźwięk uderzenia w przeszkodę, linia 385. zmieni znak wartości definiującej szybkość ruchu kuli w płaszczyźnie poziomej, wywołując metodę reverseVelocityX klasy Cannonball, a linia 388. wykona operację zmniejszenia wartości przypisanej zmiennej timeLeft — gracz zostanie ukarany za trafienie w przeszkodę (obiekt Blocker).

LISTING 6.25. Metoda testForCollisions klasy CannonView

```
354 // sprawdza, czy nie doszło do zderzenia kuli z przeszkodą lub celami,
355 // i obsługuje zdarzenie wywołane kolizją
356 public void testForCollisions() {
357     // usuń wszystkie cele,
358     // w które trafiła kula
359     if (cannon.getCannonball() != null &&
360         cannon.getCannonball().isOnScreen()) {
361         for (int n = 0; n < targets.size(); n++) {
362             if (cannon.getCannonball().collidesWith(targets.get(n))) {
363                 targets.get(n).playSound(); // odtwórz dźwięk trafienia w cel
364
365                 // do pozostałego czasu gry dodaj czas będący nagrodą za trafienie w cel
366                 timeLeft += targets.get(n).getHitReward();
367
368                 cannon.removeCannonball(); // usuń kulę
369                 targets.remove(n); // usuń trafiony cel
370                 --n; // zapewnia sprawdzenia kolizji kuli z nowym celem numer n
371                 break;
372             }
373         }
374     }
375     else { // usuń kulę, jeżeli nie powinno jej być na ekranie
376         cannon.removeCannonball();
377     }
378
379     // sprawdź, czy kula zderza się z przeszkodą
380     if (cannon.getCannonball() != null &&
381         cannon.getCannonball().collidesWith(blocker)) {
382         blocker.playSound(); // odtwórz dźwięk trafienia w przeszkodę
383
384         // odwróć kierunek ruchu kuli
385         cannon.getCannonball().reverseVelocityX();
386
387         // od ilości pozostałego czasu odejmij czas będący karą za trafienie w przeszkodę
388         timeLeft -= blocker.getMissPenalty();
389     }
390 }
391
```

6.13.12. Metody stopGame i releaseResources

Metody onPause i onDestroy klasy MainActivityFragment (patrz sekcja 6.13) wywołują (odpowiednio) metody stopGame i releaseResources klasy CannonView (patrz listing 6.26). Metoda stopGame (patrz linie 393 – 396) jest wywoływana z głównego obiektu Activity w celu zatrzymania gry w wyniku wywołania metody onPause tego obiektu — dla uproszczenia w tym przykładzie nie zapisujemy stanu gry. Metoda releaseResources (patrz linie 399 – 402) wywołuje metodę release obiektu SoundPool w celu zwolnienia zasobów skojarzonych z tym obiektem.

LISTING 6.26. Metody stopGame i releaseResources klasy CannonView

```
392 // metoda zatrzymująca grę wywoływana przez metodę onPause obiektu CannonGameFragment
393 public void stopGame() {
394     if (cannonThread != null)
395         cannonThread.setRunning(false); // wyślij polecenie zamknięcia wątku
396 }
397
398 // metoda zwalnająca zasoby wywoływana przez metodę onDestroy obiektu CannonGame
399 public void releaseResources() {
400     soundPool.release(); // zwolnij wszystkie zasoby używane przez obiekt SoundPool
401     soundPool = null;
402 }
403
```

6.13.13. Implementacja metod interfejsu SurfaceHolder.Callback

Listing 6.27 przedstawia kod implementujący metody surfaceChanged, surfaceCreated i surfaceDestroyed interfejsu SurfaceHolder.Callback. W przypadku tej aplikacji metoda surfaceChanged jest pusta, ponieważ aplikacja jest zawsze wyświetlana w orientacji poziomej. Metoda ta jest wywoływana przy zmianie rozmiaru lub orientacji obiektu SurfaceView i jest zwykle używana do ponownego wyświetlenia grafiki po zmianie tych parametrów.

LISTING 6.27. Implementacja metod interfejsu SurfaceHolder.Callback

```
404 // metoda wywoływana w wyniku zmiany rozmiaru powierzchni
405 @Override
406 public void surfaceChanged(SurfaceHolder holder, int format,
407     int width, int height) { }
408
409 // metoda wywoływana, gdy powierzchnia jest tworzona po raz pierwszy
410 @Override
411 public void surfaceCreated(SurfaceHolder holder) {
412     if (!dialogIsDisplayed) {
413         newGame(); // przygotuj nową grę i uruchom ją
414         cannonThread = new CannonThread(holder); // utwórz wątek
415         cannonThread.setRunning(true); // uruchom grę
416         cannonThread.start(); // uruchom wątek pętli gry
417     }
418 }
419
420 // metoda wywoływana, gdy powierzchnia jest usuwana
```

```

421 @Override
422 public void surfaceDestroyed(SurfaceHolder holder) {
423     // upewnij się, że wątek zostanie poprawnie zakończony
424     boolean retry = true;
425     cannonThread.setRunning(false); // zakończ wątek cannonThread
426
427     while (retry) {
428         try {
429             cannonThread.join(); // poczekaj na zakończenie wątku cannonThread
430             retry = false;
431         }
432         catch (InterruptedException e) {
433             Log.e(TAG, "Przerwanie wątku", e);
434         }
435     }
436 }
437

```

Metoda `surfaceCreated` (linie 410 – 418) jest wywoływana w momencie utworzenia obiektu `SurfaceView` — np. podczas pierwszego załadowania aplikacji lub wtedy, gdy aplikacja jest przywracana z tła. Z metody tej korzystamy w celu utworzenia i uruchomienia wątku `CannonThread` — rozpoczęcia wykonywania pętli gry. Metoda `surfaceDestroyed` (linie 421 – 436) jest wywoływana, gdy obiekt `SurfaceView` jest usuwany — np. wtedy, gdy aplikacja jest zamykana. Z metody tej korzystamy w celu upewnienia się, że wątek `CannonThread` zostanie zamknięty poprawnie. Na początek linia 425. wywołuje metodę `setRunning` wątku `CannonThread`, przekazując do niej argument `false` — zabieg ten ma na celu przekazanie wątkowi informacji o tym, że powinien zostać *zatrzymany*. Następnie kod znajdujący się w liniach 427 – 435 czeka na *zakończenie* wątku. Dzięki temu mamy pewność, że nie zostanie podjęta próba rysowania elementów gry na powierzchni `SurfaceView` po zakończeniu wykonywania metody `surfaceDestroyed`.

6.13.14. Przedefiniowywanie metody `onTouchEvent` klasy `View`

W tym przykładzie musimy przeddefiniować metodę `onTouchEvent` (patrz listing 6.28) klasy `View` w celu określenia, kiedy użytkownik dotyka ekranu. Parametr `MotionEvent` zawiera informacje dotyczące zdarzenia, które miało miejsce. Linia 442. korzysta z metody `getAction` obiektu `MotionEvent` w celu określenia rodzaju zdarzenia dotyku, do którego doszło. Następnie kod znajdujący się w liniach 445 – 446 określa, czy użytkownik dotknął ekranu (`MotionEvent.ACTION_DOWN`), czy przeciągnął po nim palcem (`MotionEvent.ACTION_MOVE`). W obu przypadkach linia numer 448 wywołuje metodę `alignAndFireCannonball` klasy `cannonView` w celu wycelowania lufą w punkt dotknięcia i wystrzelenia kuli w tym kierunku. Kod znajdujący się w linii numer 451 zwraca wartość logiczną `true` w celu zaznaczenia tego, że zdarzenie zostało obsłużone.

LISTING 6.28. Przedefiniowywanie metody `onTouchEvent` klasy `View`

```

438 // metoda wywoływana po dotknięciu ekranu przez użytkownika w tej aktywności
439 @Override
440 public boolean onTouchEvent(MotionEvent e) {
441     // uzyskaj wartość int opisującą rodzaj czynności, która doprowadziła do tego zdarzenia
442     int action = e.getAction();
443
444     // użytkownik dotknął ekranu lub przeciągnął po nim palcem
445     if (action == MotionEvent.ACTION_DOWN ||
446         action == MotionEvent.ACTION_MOVE) {

```



```

447     // wystrzel kulę w kierunku punktu dotyku
448     alignAndFireCannonball(e);
449 }
450
451 return true;
452 }
453

```

6.13.15. Korzystanie z wątku CannonThread w celu utworzenia pętli gry

Kod znajdujący się w listingu 6.29 definiuje podklasę Thread, która aktualizuje grę. Wątek podtrzymuje odwołanie do interfejsu SurfaceHolder obiektu SurfaceView i zmienną logiczną wskazującą działanie wątku.

LISTING 6.29. Zagnieżdżona klasa wątku CannonThread zarządza pętlą gry, aktualizując jej elementy co ilość czasu określoną przez zmienną TIME_INTERVAL (czas jest wyrażony w milisekundach)

```

454 // Klasa zagnieżdżona wątku sterująca pętlą gry.
455 private class CannonThread extends Thread {
456     private SurfaceHolder surfaceHolder; // do manipulowania obiektem canvas
457     private boolean threadIsRunning = true; // domyślnie uruchamiany
458
459     // inicjalizuje obiekt SurfaceHolder
460     public CannonThread(SurfaceHolder holder) {
461         surfaceHolder = holder;
462         setName("CannonThread");
463     }
464
465     // zmienia stan wykonywania
466     public void setRunning(boolean running) {
467         threadIsRunning = running;
468     }
469
470     // steruje pętlą gry
471     @Override
472     public void run() {
473         Canvas canvas = null; // używany do rysowania
474         long previousFrameTime = System.currentTimeMillis();
475
476         while (threadIsRunning) {
477             try {
478                 // uzyskaj obiekt Canvas przeznaczony do wyświetlania elementów tylko przez ten wątek
479                 canvas = surfaceHolder.lockCanvas(null);
480
481                 // zablokuj obiekt surfaceHolder w celu rysowania elementów
482                 synchronized(surfaceHolder) {
483                     long currentTime = System.currentTimeMillis();
484                     double elapsedTimeMS = currentTime - previousFrameTime;
485                     totalElapsedTime += elapsedTimeMS / 1000.0;
486                     updatePositions(elapsedTimeMS); // aktualizuj status gry
487                     testForCollisions(); // sprawdź, czy nie dochodzi do kolizji z elementem GameElement
488                     drawGameElements(canvas); // rysuj, korzystając z obiektu canvas
489                     previousFrameTime = currentTime; // aktualizuj czas
490                 }

```

```

491     }
492     finally {
493         // wyświetl zawartość obiektu canvas w widoku CannonView
494         // i pozwól innym wątkom na korzystanie z tego obiektu
495         if (canvas != null)
496             surfaceHolder.unlockCanvasAndPost(canvas);
497     }
498 }
499 }
500 }

```

Metoda `run` tej klasy (patrz linie 471 – 499) steruje powstawaniem *kolejnych klatek animacji* — jest to tzw. *pętla gry*. Przesunięcia elementów gry wyświetlanych na kolejnej klatce animacji są określane na podstawie liczby milisekund, które upłynęły od wyświetlenia poprzedniej klatki. Linia numer 474 odczytuje stan systemowy (wyrażony w milisekundach) w momencie uruchomienia wątku. Kod umieszczony w liniach 476 – 498 jest pętlą wykonywaną do momentu, w którym zmiennej `threadIsRunning` zostanie przypisana wartość logiczna `false`.

Na początku uzyskujemy obiekt `Canvas` służący do rysowania w widoku `SurfaceView` poprzez wywołanie metody `lockCanvas` obiektu `SurfaceHolder` (linia numer 479). Grafika może być tworzona na obiekcie `SurfaceView` tylko przez jeden wątek. W tym celu musisz najpierw zablokować obiekt `SurfaceHolder`, podając go jako wyrażenie umieszczone w nawiasach w bloku `synchronized` (linia 482.). Następnie odczytujemy bieżący czas (wyrażony w milisekundach), obliczamy czas, jaki upłynął od wyświetlenia poprzedniej klatki, i dodajemy go do całkowitego czasu gry — wartość ta przyda się do wyświetlenia całkowitego czasu pozostałego do zakończenia gry. Linia 486. wywołuje metodę `updatePositions` w celu przesunięcia wszystkich elementów gry, przekazując do niej w roli argumentu wyrażony w milisekundach czas, jaki minął od wyświetlenia poprzedniej klatki. Dzięki temu gra działa z taką samą szybkością *niezależnie od wydajności urządzenia*. Jeżeli pomiędzy wyświetleniem kolejnych klatek upływa więcej czasu (tj. urządzenie jest wolniejsze), to na kolejnych klatkach animacji elementy gry będą przesuwane o większe odległości. Jeżeli czas pomiędzy wyświetleniem kolejnych klatek jest krótszy (tj. urządzenie jest szybsze), to przesunięcie obiektów gry pomiędzy kolejnymi klatkami animacji będzie mniejsze. Kod znajdujący się w linii 487. wywołuje metodę `testForCollisions` w celu sprawdzenia, czy kula nie uderzyła w przeszkodę lub cel:

- Jeżeli dojdzie do kolizji z przeszkodą, to metoda `testForCollisions` odwróci kierunek ruchu kuli.
- Jeżeli dojdzie do kolizji z celem, to metoda `testForCollisions` usuwa kulę.

Na koniec linia 488. wywołuje metodę `drawGameElements` w celu wyświetlenia elementów gry za pomocą obiektów `Canvas` widoku `SurfaceView`, a linia 489. przypisuje wartość `currentTime` do zmiennej `previousFrameTime` — w przyszłości wartość ta zostanie użyta w celu obliczenia czasu, jaki upłynął pomiędzy wyświetleniem bieżącej i *kolejnej* klatki.

6.13.16. Metody `hideSystemBars` i `showSystemBars`

Aplikacja korzysta z *trybu pełnoekranowego*, w którym użytkownik może w dowolnym momencie wyświetlić paski systemowe, przeciągając palcem od góry ekranu. Tryb ten jest dostępny tylko w systemach Android 4.4 i nowszych. W związku z tym metody `hideSystemBars` i `showSystemBars` (patrz listing 6.30) sprawdzają najpierw wersję systemu Android uruchomioną na danym urządzeniu — sprawdzają, czy stała `Build.VERSION.SDK_INT` ma wartość równą lub wyższą od wartości `Build.VERSION_CODES.KITKAT` (stałej systemu Android 4.4 o poziomie API 19). Jeżeli wartość ta jest równa lub wyższa, to metoda `setSystemUiVisibility` klasy `View` jest używana do skonfigurowania pasków systemowych i paska aplikacji

(w praktyce pasek aplikacji został ukryty wcześniej w wyniku modyfikacji motywu aplikacji). W celu ukrycia pasków systemowych i paska aplikacji, a także przełączenia interfejsu użytkownika w tryb pełnoekranowy przekazujesz w liniach 517 – 519 metodzie `setSystemUiVisibility` stałe połączone za pomocą operatora bitowego `OR (|)`. Te kombinacje stałych `View` zapewniamy, że rozmiary obiektu `CanvasView` nie będą zmieniane za każdym razem, gdy paski systemowe i aplikacji będą ukrywane i wyświetlane. Paski systemowe i pasek aplikacji będą wyświetlane, nakładając się na obiekt `CanvasView` — część tego obiektu będzie niewidoczna, gdy na ekranie będą wyświetlane paski systemowe. Więcej informacji na temat trybu pełnoekranowego znajdziesz na stronie:

<http://developer.android.com/training/system-ui/immersive.html>.

LISTING 6.30. Metody `hideSystemBars` i `showSystemBars` klasy `DoodleView`

```
501 // ukryj paski systemowe i pasek aplikacji
502 private void hideSystemBars() {
503     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
504         setSystemUiVisibility(
505             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
506             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
507             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN |
508             View.SYSTEM_UI_FLAG_HIDE_NAVIGATION |
509             View.SYSTEM_UI_FLAG_FULLSCREEN |
510             View.SYSTEM_UI_FLAG_IMMERSIVE);
511 }
512
513 // pokaż paski systemowe i pasek aplikacji
514 private void showSystemBars() {
515     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
516         setSystemUiVisibility(
517             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
518             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
519             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
520 }
521 }
```

6.14. Podsumowanie

W tym rozdziale pracowałeś nad aplikacją *Cannon Game* — grą, w której użytkownik ma za zadanie trafić w dziewięć celów przed upływem 10 sekund. Użytkownik celuje lufą działa i oddaje strzał poprzez dotknięcie ekranu. Stworzyłeś widok rozszerzający klasę `SurfaceView`, dzięki czemu obiekty były wyświetlane na ekranie za pomocą oddzielnego wątku. Dowiedziałeś się, że nazwy komponentów spersonalizowanej klasy muszą zostać w pełni sklasyfikowane w kodzie XML rozkładu reprezentującego dany komponent. Opisałeś dodatkowe metody cyklu życiowego obiektów `Fragment`. Dowiedziałeś się, że metoda `onPause` jest wywoływana, gdy `Fragment` jest wstrzymywany, a metoda `onDestroy` jest wywoływana, gdy `Fragment` jest usuwany. Obsługiwałeś zdarzenia dotyku za pomocą zdefiniowanej metody `onTouchEvent` klasy `View`. W folderze `res/raw` umieściłeś efekty dźwiękowe, a następnie zarządzałeś nimi za pomocą obiektu `SoundPool`. Ponadto korzystałeś z usługi systemowej `AudioManager` w celu ustalenia aktualnej głośności muzyki odtwarzanej przez urządzenie i stosowania tej wartości do regulacji głośności dźwięków odtwarzanych przez grę.

Animacja tej aplikacji jest wykonywana w sposób manualny poprzez aktualizację położenia elementów gry wyświetlanych na obiekcie `SurfaceView` za pomocą kodu wykonywanego w oddzielnym wątku. W celu zaimplementowania takiego rozwiązania rozszerzyłeś klasę `Thread` i stworzyłeś metodę `run`, która wyświetlała grafikę, wywołując metody klasy `Canvas`. Dostęp do właściwej klasy `Canvas` uzyskiwałeś za pomocą obiektu `SurfaceHolder` klasy `SurfaceView`. Ponadto dowiedziałeś się również, jak utworzyć pętlę gry sterującą jej pracą. Kolejne klatki animacji są budowane przez tę pętlę na podstawie ilości czasu, jaki upłynął pomiędzy wyświetleniem kolejnych klatek, a więc gra będzie działała z taką samą szybkością niezależnie od wydajności procesora urządzenia, na którym jest uruchomiona. Na koniec korzystałeś z trybu pełnoekranowego — aplikacja była wyświetlana na całym ekranie.

W rozdziale 7. będziesz pracować nad aplikacją **WeatherViewer**. Skorzystasz z usług sieciowych w celu pobrania szesnastodniowej prognozy pogody z serwisu *OpenWeatherMap.org*. Serwis ten, podobnie jak wiele współczesnych serwisów internetowych, zwraca dane w formacie JavaScript Object Notation (JSON). Dane te będziesz przetwarzać za pomocą klas `JSONObject` i `JSONArray` wchodzących w skład pakietu *org.json*. Prognozę pogody na dany dzień wyświetlisz na ekranie za pomocą obiektu `ListView`.

Skorowidz

A

adapter
 ArrayAdapter, 309
 RecyclerView.Adapter, 345, 403
adres URL, 317
agencje public relations, 438
aktywność, 75
analiza, 55
Android
 2.2, 39
 2.3, 39
 3.0 – 3.2, 40
 4.0 – 4.0.4, 40
 4.1 – 4.3, 42
 4.4, 43
 5.0 – 5.1, 39, 43
 6, 45
 6 SDK, 19, 29
 Studio, 50, 71
 TV, 21
 Wear, 21
animacja, 153, 163, 256
 klatka po klatce, 253
anonimowa klasa, 193
 wewnętrzna, 133, 228, 246, 250, 343
aplikacja
 Address Book, 357
 Cannon Game, 253
 Doodlz, 201
 Flag Quiz, 141
 obliczająca napiwek, 107

 powitalna, 69
 Tip Calculator, 55, 61, 63, 109
 Twitter Searches, 321
 WeatherViewer, 291
aplikacje
 darmowe, 426
 płatne, 426
 popularne, 48
 wbudowane, 36
atrybuty, 54
AVD, Android Virtual Device, 30, 55

B

baza danych SQLite, 363
biblioteka
 Android Design Support Library, 332
 Android Support Library, 211
 AppCompat, 111
bitmapy, 210

C

cechy systemu, 35
certyfikat cyfrowy, 423
cienie, 114
cykl życiowy obiektu Fragment, 208

D

darmowe aplikacje, 426
debugowanie, 423

definiowanie
 parametru gravity, 92
 parametru weight, 94
 parametru textColor, 91
 własności przycisków, 172
docelowi odbiorcy, 18
dodawanie
 aktywności, 76
 dźwięku, 256, 259
 ikony, 85
 ikony aplikacji, 367
 ikony przycisku, 334
 kontaktu, 360
 obiektu CannonView, 260
 obrazów do projektu, 158
 pola ImageView, 95
 widoków, 118, 119
 zależności, 211
 zezwoleń, 216
dokumentacja, 66
domyślny interfejs użytkownika, 81
dostawcy usług mobilnych, 429
dostęp
 do aplikacji, 100
 do bazy danych, 364
 do płatnych aplikacji, 422
 do zasobów, 345
dostępność, 72
drukowanie, 21, 211
 obrazu, 208
działanie aplikacji Doodlz, 203
dziedziczenie, 54

E

edycja
 kontaktu, 361
 zapytania, 326
edytor
 rozkładu, 80, 83
 tłumaczeń, 102
ekran
 Ustawienia, 145
 wielodotkowy, 36
element
 activity, 137
 application, 136
 intent-filter, 137
 manifest, 136
emulacja przycisków, 52
emulator
 smartfona, 60
 systemu Android, 30, 51
etykiety, 421

F

folder
 assets, 150
 drawable, 84
foldery zasobów, 150, 255
fora dyskusyjne, 67
formatowanie
 liczb, 113
 łańcuchów, 159
fragment, 20, 148
 AddEditFragment, 373
 ContactsFragment, 371
 DetailFragment, 372, 375
 DialogFragment
 confirmDelete, 415
 LineWidthDialogFragment,
 219
funkcja TalkBack, 100, 101, 105
funkcje
 systemu, 41, 42, 46
 środowiska Java SE 7, 155
 urządzeń, 52

G

gesty, 36, 51
gra, 254
graficzny
 edytor rozkładu, 172
 interfejs użytkownika, 81, 112,
 115, 120, 217, 257, 297, 300,
 332, 365

H

hermetyzacja, 54

I

ikona, 85, 421
 aplikacji, 367
 przycisku, 334
implementacja
 interfejsu
 OnSeekBarChangeListener,
 113
 interfejsu TextWatcher, 113
import klasy MainActivity, 339
instrukcja
 import, 128, 175, 182, 234,
 307, 312
 package, 128, 175, 182, 234,
 313
interfejs
 ddEditFragment.AddEdit
 ↳ FragmentListener, 394
 AddEditFragmentListener,
 404
 ContactClickListener, 403
 ContactsFragment.Contacts
 ↳ FragmentListener, 391
 ContactsFragmentListener,
 396
 DetailFragment.Detail
 ↳ FragmentListener, 394
 DetailFragmentListener, 412
 Editable, 128
 LoaderManager.Loader
 ↳ Callbacks, 365, 399, 410
 OnClickListener, 343

OnSeekBarChangeListener,
 113, 133, 134
SeekBar.OnSeekBarChange
 ↳ Listener, 129
SurfaceHolder.Callback, 285
TextWatcher, 113, 128
View.OnLongClickListener,
 347
internacjonalizacja, 72, 101

J

Java Development Kit, 27
JDK, 27
język XML, 71
JSON, 295

K

kasowanie
 kontaktu, 361
 zapytania, 328
klasa, 53
 Activity, 110, 130
 Adapter, 331
 AddEditFragment, 366, 376,
 404
 AddressBookContentProvider,
 367, 376, 380
 AddressBookDatabaseHelper,
 366, 376, 379
 AppCompatActivity, 111, 128
 ArrayAdapter, 298
 AsyncTask, 297, 298, 311
 Blocker, 265
 Bundle, 128
 Cannon, 266
 Cannonball, 270
 CannonView, 257, 272, 273
 Canvas, 210
 ColorDialogFragment, 219,
 243
 Contact, 378
 ContactsAdapter, 366, 376,
 401
 ContactsFragment, 376, 395
 ContentProvider, 363

- DatabaseDescription, 366, 377
- DetailFragment, 366, 377, 411
- DoodleView, 234
- EraseImageDialogFragment, 223, 250
- FragmentTransaction, 362
- GameElement, 263
- Handler, 152
- ItemDivider, 338, 353, 366, 377
- ItemDividerMainActivity, 338, 376
- LineWidthDialogFragment, 247
- ListView, 298
- LoaderManager, 365
- MainActivity, 129, 133, 175, 176, 178, 223, 261
- MainActivityFragment, 182, 187, 189, 224, 259
- NumberFormat, 129
- Paint, 210
- PrintHelper, 211, 212
- RecyclerView.Adapter, 350
- RecyclerView.ItemDecoration, 331, 353
- SearchesAdapter, 338, 350
- SettingsActivity, 166, 196
- SettingsActivityFragment, 197
- Target, 266
- View, 256
- ViewHolder, 308, 351, 403
- Weather, 304
- WeatherArrayAdapter, 307
- klawiatura, 109
- klucz prywatny, 423
- klucze, 341
- kod wersji, 422
- kolory, 259
- komponent Snackbar, 300
- komunikat Intent, 154
- komunikaty, 54
- konfiguracja
 - konta sprzedawcy, 430
 - obiektu amountEditText, 121
 - obiektu percentSeekBar, 122

- obiektu percentTextView, 122
- obiektu RecyclerView, 342
- obiektu tipTextView, 123
- parametru textSize, 90
- przycisku, 348
- quizu, 143
- urządzenia wirtualnego, 59
- konkretyzacja, 53
- konstruktor klasy
 - CannonView, 274
 - DoodleView, 236
 - Weather, 305
 - WeatherArrayAdapter, 308
- kontakt, 361
- konto sprzedawcy, 430
- kończenie quizu, 146

L

- licencja EULA, 421
- lista
 - kolorów stanów, 153
 - mailingowa, 67
 - zapisanych zapytań, 329
- logika aplikacji, 127
- lokalizacja, 101, 105
- łańcuchów, 102

Ł

- ładowanie
 - aplikacji do serwisu, 431
 - dźwięków, 275
- łańcuchy
 - aplikacji, 258
 - procentowe, 113
 - sformatowane, 158

M

- marketing wirusowy, 435
- mashup, 37
- materiały wideo, 23
- mechanizm płatności, 428
- menedżer
 - LoaderManager, 364
 - urządzeń wirtualnych, 60

- menu, 148, 214
 - MainActivityFragment, 214
- metoda, 53
 - addTaggedSearch, 344
 - align, 267
 - alignAndFireCannonball, 280
 - animate, 191
 - calculate, 133
 - clear, 237
 - collidesWith, 270
 - confirmErase, 229
 - convertJSONtoArrayList, 318
 - convertTimeStampToDay, 306, 307
 - createUrl, 315
 - delete, 388
 - deleteContact, 415
 - deleteSearch, 349
 - disableAccelerometerListening, 227
 - disableButtons, 195
 - dismissKeyboard, 315
 - displayAddEditFragment, 393
 - displayContact, 392
 - draw, 265, 268, 272
 - drawGameElements, 282
 - drawText, 283
 - enableAccelerometerListening, 226
 - fireCannonball, 268
 - getCannonball, 269
 - getCountryName, 191
 - getDoodleFragment, 245
 - getDoodleView, 233
 - getDrawingColor, 237
 - getHolder, 275
 - getItemCount, 353, 403
 - getLineWidth, 237
 - getRadius, 270
 - getScreenHeight, 277
 - getScreenWidth, 277
 - getType, 382
 - getView, 309
 - hideSystemBars, 288
 - insert, 385
 - isOnScreen, 270
 - newGame, 277
 - onActivityCreated, 398

metoda

- onAddContact, 392
- onAttach, 398, 406, 413
- onBindViewHolder, 353, 403
- onCreate, 131, 176, 314, 390
- onCreateDialog, 250
- onCreateLoader, 400
- onCreateOptionsMenu, 179, 414
- onCreateView, 184, 226, 397, 406
- onCreateViewHolder, 353, 403
- onDetach, 398, 406, 413
- onDraw, 238
- onDrawOver, 354
- onEditContact, 394
- onLoaderReset, 401
- onLoadFinished, 400
- onOptionsItemSelected, 180, 414
- onPause, 227
- onPostExecute, 317
- onProgressChanged, 134
- onProgressUpdate, 317
- onResume, 226
- onSizeChanged, 236, 276
- onStart, 178
- onTouchEvent, 238, 256, 286
- playSound, 265, 277
- printImage, 243
- query, 383
- releaseResources, 285
- removeCannonball, 269
- resetQuiz, 187, 188, 189
- reverseVelocityX, 270
- run, 288
- saveContact, 408, 409
- saveImage, 231, 242
- setDialogOnScreen, 233
- setDrawingColor, 237
- setLineWidth, 237
- shareSearch, 348
- showGameOverDialog, 281
- showSystemBars, 288
- stopGame, 285
- swapCursor, 403
- testForCollisions, 283

- touchEnded, 241
- touchMoved, 240
- touchStarted, 239
- update, 265, 271, 387
- updateContactList, 399
- updateGuessRows, 186, 187
- updatePositions, 279
- updateRegions, 187
- updateSaveFAB, 342

metody

- cyklu roboczego, 110
- cyklu życia, 255
- cyklu życiowego, 149
- klasy RecyclerView.Adapter, 352
- obiektu Fragment, 230
- onCreate, 130

mobilne sieci reklamowe, 439

modyfikacja motywu, 258

motyw

- domyślny, 123
- parent, 123

motywy Material Themes, 113

multimedia, 22

N

narzędzie

- AudioManager, 256
- Cloud Test Lab, 21
- SoundPool, 256

nasłuchiwanie zdarzeń, 132, 193, 343, 407

nazwy wersji, 422

nazywanie folderów, 101

O

obiekt

- Activity, 176, 208
- amountEditText, 121
- CannonView, 260, 262
- Canvas, 283
- colorView, 219
- ContentProvider, 363, 364, 383
- ContentResolver, 364
- Cursor, 385

- DialogFragment, 209
- EditText, 113, 117
- Fragment, 184, 208
- Intent, 346, 349
- Loader, 364
- OnClickListener, 193, 343
- OnSharedPreferenceChange
 - ↳ Listener, 180
- Paint, 276
- Path, 210
- percentSeekBar, 122
- percentTextView, 122
- Runnable, 152
- saveContactButtonClicked, 408
- SensorManager, 209
- SeekBar, 117
- SharedPreferences, 341, 344, 346
- SoundPool, 275
- SQLiteDatabase, 385
- SurfaceHolder, 256
- SurfaceView, 256
- TextViews, 117
- tipTextView, 123
- totalTextView, 123
- Uri, 364
- UriMatcher, 384
- View.OnClickListener, 345

obiekty

- nasłuchujące zdarzeń, 407, 408
- View, 153

obraz promocyjny, 423

obsługa

- bazy danych, 363
- ekranów, 151
- intencji, 330
- środowiska Java SE 7, 158
- zdarzeń, 342, 348

odczytywanie łańcuchów, 346

okna edytora, 79

okno

- AlertDialog, 153, 195, 348
- Android Studio, 77
- Component Tree, 80
- Create New Project, 73, 74
- Device Chooser, 61

- Devices, 424
- Project, 78
- Properties, 89
 - wyboru aktywności, 349
- operatory wyszukiwania, 324
- organizowanie widoków, 112
- orientacja
 - obiekту LinearLayout, 303
 - pionowej, 178
 - tabletu, 173
 - urządzenia, 152
- otwieranie sklepu, 433

P

- pakiet, 47–50
 - Android Studio, 28
 - android.graphics, 210
 - android.text, 128
 - com.deitel.addressbook.data, 366, 376
 - org.json, 295, 297
- paleta, 71
- parametr
 - Bundle, 131
 - elevation, 114
 - gravity, 92
 - id, 88, 116
 - textColor, 91
 - textSize, 90
- parametr layout
 - gravity, 93
 - weight, 94
- pasek
 - aplikacji Doodlz, 203
 - widthSeekBar, 250
- personalizacja
 - aktywności, 77
 - kolorów motywu, 123, 124
 - obiektów View, 209
 - widoków, 120
- piksele, 90
- platforma programistyczna
 - Fabric, 355
- platformy mobilne, 434
- plik
 - activity_main.xml, 81, 259, 302, 334

- AndroidManifest.xml, 115, 135, 197, 300, 332
- arrays.xml, 159, 160, 333
- bug.png, 85
- button_text_color.xml, 162
- colors.xml, 161, 301, 333
- content_main.xml, 335
- dimens.xml, 334
- doodle_fragment_menu.xml, 215
- dziennika, 154
- fragment_color.xml, 220
- fragment_details_menu.xml, 375
- incorrect_shake.xml, 163
- list_item.xml, 337
- menu_main.xml, 162
- preferences.xml, 164
- SharedPreferences, 329
- strings.xml, 158, 258, 301, 333, 367
- styles.xml, 367
- textView_border.xml, 369
- pliki
 - .apk, 423
 - z zasobami aplikacji, 80
- płatne aplikacje, 426
- pobieranie obrazów, 311, 312
- podgląd projektu, 97
- podpis cyfrowy, 423
- pola, 183
 - klasy AddEditFragment, 405
 - klasy ContactsFragment, 396
 - klasy DetailFragment, 413
 - klasy MainActivity, 339
 - static, 377
- pole
 - answerTextView, 171
 - EditText, 299, 374, 375
 - ImageView, 95, 303
 - TextView, 304, 337, 365
- połączenie HttpURLConnection, 297
- ponowne wykorzystanie, 54
- preferencje aplikacji, 164
- prognoza pogody, 293
- programowanie obiektowe, 53

- projektowanie
 - ikon, 422
 - zorientowane obiektowo, 55
- promocja aplikacji, 425
- przechowywanie danych, 329
- przesyłanie danych, 363
- przycisk
 - FloatingActionButton, 299, 342
 - saveButton, 343
- przyciski pływające, 299
- publikacja, 420

R

- recenzje aplikacji, 437
- rejestracja, 430
- rejestrowanie obiektu
 - nasłuchującego, 275
- reklamowanie aplikacji, 435
- REST, 20
- rozkład, 71
 - activity_main.xml, 167
 - aktywności, 369
 - content_main.xml, 157, 173, 217, 369–371
 - fragment_color.xml, 218, 219
 - fragment_contacts.xml, 371
 - fragment_line_width.xml, 219, 222
 - fragment_main.xml, 157, 167, 217
 - GridLayout, 112, 115, 117, 303
 - LinearLayout, 87, 168, 169, 336, 373
 - list_item.xml, 302
 - queryTextInputLayout, 336
 - RelativeLayout, 86, 168
 - tagTextInputLayout, 336
- rozkłady szablonu, 157
- rozwiązywanie quizu, 145
- rysowanie, 210, 269
 - kropel, 207
 - płatków kwiatka, 205

S

SDK, 50
serwis
 Google Payments, 430
 Google Play, 22, 47, 419, 430
 Twitter, 324
serwisy
 pośredniczące w sprzedaży, 434
 społecznościowe, 436
składnia obiektowa JSON, 20
Software Development Kit, 50
specyfikacja material design, 19
społeczności, 24
struktury danych, 155
style, 126
 pól, 367
suwak SeekBar, 113, 246
system operacyjny, 38
szablon Blank Activity, 157, 301

Ś

środowisko
 Android Studio, 29, 55, 72
 Java SE 7, 155

T

tablica, 160
 ArrayList, 341
technologie, 71
testowanie, 21, 104
 aplikacji, 31, 101, 421
 Address Book, 360
 Cannon Game, 255
 Flag Quiz, 143
 Tip Calculator, 109
 Twitter Searches, 323
 WeatherViewer, 293
 wersji językowej, 103
tło pola, 365
tłumaczenie, 102
tryb
 ListView, 20
 pełnoekranowy, 21, 257

RecyclerView, 20
uruchamiania aktywności, 198
tworzenie
 animacji, 163
 aplikacji, 72
 aplikacji doskonałych, 64
 gier, 257
 graficznego interfejsu użytkownika, 83, 112, 167
 klas, 366
 licencji, 422
 logiki aplikacji, 127
 menu, 214
 projektu, 72, 156, 212
 projektu TipCalculator, 117
 rozkładu interfejsu użytkownika, 174
 rysunków, 202
 wirtualnych urządzeń, 57
 zapytań SQL, 384

U

udostępnianie zapytania, 327
uruchamianie
 aplikacji, 98
 aplikacji Tip Calculator, 63
 środowiska, 72
 urządzeń wirtualnych, 63
urządzenie wirtualne Android, 30
usługa licencyjna, 422
usługi sieciowe, 37, 38, 294
 REST, 20, 294
ustawienia pól TextView, 373

W

wątek CannonThread, 287
wersje, 39, 422
 Android SDK, 75
wiązanie danych, 132, 298
widok, 71
 ListView, 331
 RecyclerView, 331, 337
widzet, 131

wirtualne
 dobra, 428
 urządzenia, 57
własności przycisków, 172
własność
 divider, 370
 showDividers, 370
 weightSum, 370
właściwość
 id, 88
 text, 88
współczynnik klikalności, 439
wybór
 aktywności, 138
 docelowych urządzeń, 74
 koloru, 125–127
 maszyny wirtualnej, 61
wycena aplikacji, 425
wyjątki, 154
wykonywanie aplikacji, 138
wykrywanie zderzeń, 257
wyniki wyszukiwania, 324
wyswietlanie
 komunikatów, 152
 listy elementów, 348
 listy opcji, 332
 obiektów Fragment, 362
 wytyczne material design, 114
 wywoływanie metod, 54
 usługi sieciowej, 316
wzorzec ViewHolder, 298, 331

X

XML, Extensible Markup Language, 71

Z

zaciemnianie kodu, 423
zadanie AsyncTask, 312
zapisywanie
 komunikatów, 154
 obrazów, 208, 211
zapytanie, 326–328
zapytania SQL, 384

- zarabianie, 426
 - na reklamach, 428
 - na sprzedaży wirtualnych dóbr, 428
- zarządzanie
 - aplikacjami, 434
 - fragmentami, 149
- zasoby
 - aplikacji, 71
 - typu String, 213
- zdarzenia
 - dotknięcia, 210
 - paska widthSeekBar, 250
 - przyspieszeniomierza, 228
- zintegrowane środowisko programistyczne, 19

- zmiana
 - grubości linii, 205
 - koloru pędzla, 204, 205
 - lokalizacji, 104
 - rozkładu, 86, 87
- zmiennne
 - egzemplarzowe, 54, 264
 - klasy, 129
 - lokalne final, 348
 - obiektowe, 129
- zrzuty ekranu, 423

Ź

- źródła bibliotek, 35
- źródło usług sieciowych, 38

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>

Sprawdź, jak niesamowite możliwości daje Ci Android!

W 2015 roku pod kontrolą Androida pracowało około 80% smartfonów. Serwis Google Play odnotowuje miliardy pobrań najróżniejszych aplikacji dla tego systemu. Coraz więcej różnych urządzeń wyposaża się w Androida i są to również roboty, silniki odrzutowe, satelity NASA, lodówki, telewizory, kamery, urządzenia medyczne, systemy samochodowe i wiele innych. Wygląda na to, że Android opanuje internet rzeczy, a przed programistami androidowych aplikacji otworzą się niespotykane dotąd możliwości.

Niniejsza książka stanowi solidne kompendium wiedzy dla osób profesjonalnie tworzących aplikacje dla Androida w jego najnowszej, 6. wersji. Autorzy przyjęli analizę aplikacji jako metodę nauki — wszystkie rozwiązania i technologie są opisywane w kontekście kompletnych, działających programów, których dobór umożliwił przedstawienie najważniejszych funkcji i interfejsów programistycznych systemu Android. Dzięki tej książce błyskawicznie nauczysz się wszystkiego, co jest niezbędne do rozpoczęcia w pełni samodzielnej pracy w Android Studio i Android 6 SDK.

Najważniejsze zagadnienia:

- analiza aplikacji dla Androida, jej działanie i budowa
- zasady projektowania aplikacji w specyfikacji material design wprowadzonej przez Google
- obsługa bibliotek i zapewnienie zgodności nowych aplikacji ze starszymi wersjami Androida
- testowanie aplikacji w urządzeniach, emulatorach i za pomocą Cloud Test Lab
- technologie przydatne do obsługi systemów Android Wear i Android TV
- umieszczanie aplikacji w serwisie Google Play, jej promocja i rozpowszechnianie

Paul Deitel i Harvey Deitel są współzałożycielami wydawnictwa Deitel & Associates, które specjalizuje się w publikowaniu książek o programowaniu. Jest też organizatorem popularnych szkoleń dla programistów. Poza tym publikuje bardzo przydatne lekcje wideo z serii LiveLessons i udostępnia zasoby internetowe ułatwiające naukę tworzenia aplikacji dla Androida (Java) i iOS (Swift) oraz naukę posługiwania się wieloma technologiami, w tym C#, .NET czy Visual Basic.

Alexander Wald jest uzdolnionym programistą, specjalizującym się w rozwijaniu aplikacji, również dla Androida.

księgarnia Internetowa		ul. Kościuszki 1c, 44-100 Gliwice	
http://helion.pl		tel.: 32 230 98 63	
zamówienia telefoniczne		e-mail: helion@helion.pl	
	0 801 339900	http://helion.pl	
	0 601 339900	Sprawdź najnowsze promocje:	
Informatyka w najlepszym wydaniu		• http://helion.pl/promocje	
cena: 77,00 zł		Książki najchętniej czytane:	
		• http://helion.pl/bestsellery	
		Zamów informacje o nowościach:	
		• http://helion.pl/nowosci	

ISBN 978-83-283-2578-4
9 788328 325784