

Microsoft® XNA® Game Studio 4.0

Projektuj i buduj gry dla konsoli Xbox 360®,
urządzeń z systemem Windows® Phone 7
i własnego PC



Tytuł oryginału: Microsoft® XNA® Game Studio 4.0: Learn Programming Now!: How to program for Windows Phone 7, Xbox 360, Zune devices, and more

Tłumaczenie: Mikołaj Szczepaniak (wstęp, rozdz. 1 – 3, 10 – 16);
Jacek Kowolik (rozdz. 4 – 9, 17 – 19)

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-246-3410-1

© 2012 Helion S.A.

Authorized translation of the English edition of Microsoft® XNA® Game Studio 4.0: Learn Programming Now! ISBN 9780735651579, First Edition © 2011, Microsoft Corporation.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls of all rights to publish and sell the same.

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

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

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

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/games4>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wstęp	15
Część I Wprowadzenie	
1. Komputery, C#, XNA i Ty	19
Wprowadzenie	19
Nauka programowania	19
Jak zostać świetnym programistą	20
Jak działa ta książka	21
Język C# i framework XNA	21
Do dzieła	22
Instalacja środowiska programowania i frameworku XNA	22
Konfiguracja komputera PC pod kątem uruchamiania gier XNA	23
Konfiguracja konsoli Xbox 360 pod kątem uruchamiania gier XNA	24
Konfiguracja systemu Windows Phone pod kątem uruchamiania gier XNA	26
Pisanie pierwszego programu	28
Tworzenie pierwszego projektu	28
Uruchamianie pierwszego programu	31
Zatrzymywanie programu	33
Przechowywanie gier w konsoli Xbox 360 lub urządzeniu z systemem Windows Phone	34
Uruchamianie tej samej gry XNA na różnych urządzeniach	34
Podsumowanie	37
Przegląd rozdziału w pytaniach	37
2. Programy, dane i ładne kolory	39
Wprowadzenie	39
Tworzenie programu gry	40
Wyrażenia w metodzie Draw	41
Praca z kolorami	43
Przechowywanie wartości kolorów	43
Ustawianie wartości koloru	44
Kontrolowanie koloru	45
Gry i klasy	46
Klasy jako biura	48
Dane świata gry	49
Przechowywanie danych w pamięci komputera	51
Rysowanie z wykorzystaniem zmiennych intensywności barw	52

Aktualizowanie kolorów	53
Przepełnienie pamięci i wartości danych	55
Pełnowartościowa wielokolorowa lampa	56
Podjęcie decyzji w programie	57
Gotowa gra w wielokolorową lampę	61
Znajdowanie błędów w programie	63
Podsumowanie	65
Przegląd rozdziału w pytaniach	65
3. Pobieranie danych wejściowych od gracza	67
Wprowadzenie	67
Odczytywanie stanu pada	68
Pady i klasy	68
Znajdowanie pada	70
Testowanie statusu pada	72
Stosowanie klawiatury	73
Zatrzymanie gry za pomocą klawisza Escape	75
Jednoczesne używanie pada i klawiatury	75
Dodawanie wibracji	77
Sterowanie wibracjami pada	77
Testowanie wartości intensywności	78
Błędy w programie	83
Podsumowanie	85
Przegląd rozdziału w pytaniach	85

Część II **Obrazy, dźwięk i tekst**

4. Wyświetlanie obrazów	89
Wprowadzenie	89
Zasoby i zawartość	90
Dodawanie obrazów	90
Zarządzanie zawartością z użyciem XNA	92
Praca z zawartością za pomocą XNA Game Studio	92
Rozwiązania i projekty w XNA Game Studio	92
Dodawanie zawartości do projektu	94
Korzystanie z zasobów w programie gry	97
Ładowanie tekstur w XNA	97
Pozycjonowanie sprajtu na ekranie	101
Wyświetlanie sprajtu za pomocą klasy SpriteBatch	103
Wypełnianie ekranu	105
Intellisense	106
Podsumowanie	108
Przegląd rozdziału w pytaniach	108

5. Wyświetlanie tekstu	111
Wprowadzenie	111
Komputery i tekst	111
Tekst jako zasób	112
Tworzenie projektu zegara w XNA	112
Dodawanie czcionki do zasobów	112
Format plików XML	115
Ładowanie czcionki	115
Wyświetlanie z użyciem czcionki	116
Zmiana właściwości czcionki	118
Uzyskiwanie daty i czasu	119
Tworzenie ładniejszego zegara z tekstem 3D	122
Wielokrotne wyświetlanie łańcuchów tekstu	122
Powtarzanie instrukcji z użyciem pętli for	124
Inne konstrukcje pętli	126
Zabawa z pętlą for	126
Tworzenie iluzji trójwymiaru	128
Tworzenie cieni z użyciem kolorów przezroczystych	128
Wyświetlanie obrazów z użyciem przezroczystości	130
Podsumowanie	131
Przegląd rozdziału w pytaniach	131
6. Tworzenie gry dla wielu graczy	133
Wprowadzenie	133
Tworzenie gry z wciskaniem przycisków	133
Dane w grze ButtonBash	134
Rozpoczynanie gry ButtonBash	134
Wyświetlanie wartości licznika wciśnięć przycisku	135
Zliczanie wciśnięć przycisku	135
Wykrywanie zmian położenia przycisku	136
Wykrywanie poziomu i zbocza	137
Konstruowanie kompletnej gry	138
Projektowanie kodu	140
Dodawanie kodu testowego	140
Podsumowanie	143
Przegląd rozdziału w pytaniach	143
7. Odtwarzanie dźwięków	145
Dodawanie dźwięku	145
Tworzenie projektu Drum Pad	145
Nagrywanie dźwięków w programie Audacity	146
Przechowywanie dźwięków w projekcie	147
Stosowanie dźwięków w programie w XNA	149

Odtwarzanie muzyki w tle	151
Ciągłe odtwarzanie dźwięku	151
Kontrola wartości null w referencjach	155
XACT audio tool	156
Odtwarzanie muzyki za pomocą klasy MediaPlayer	156
Podsumowanie	158
Przegląd rozdziału w pytaniach	158
8. Pomiar czasu	159
Tworzenie kolejnej gry	159
Błąd w pomiarze czasu reakcji	162
Ustalanie zwycięzcy za pomocą tablic	164
Tworzenie tablicy	165
Korzystanie z danych w tablicy	165
Przeglądanie tablicy	167
Zastosowanie tablicy jako słownika	168
Wyświetlanie zwycięzcy	170
Podsumowanie	172
Przegląd rozdziału w pytaniach	172
9. Wczytywanie tekstu wejściowego	173
Używanie klawiatury w XNA	173
Tworzenie projektu Tablica ogłoszeń	173
Rejestrowanie wciśnień klawiszy	174
Wykrywanie momentu wciśnięcia klawisza	175
Typ Keys	175
Typy wyliczeniowe	176
Praca z tablicami, obiektami i referencjami	177
Wartości i referencje	177
Tablice jako biura	178
Przywitaj się z procesem odzyskiwania pamięci	179
Używanie referencji i wartości	180
Do czego służą referencje i wartości?	181
Referencje i metoda GetPressedKeys	182
Wyświetlanie klawiszy	182
Wykrywanie wciśnień klawiszy	183
Dekodowanie znaków na klawiszach	188
Używanie klawiszy Shift	189
A co z polskimi znakami?	190
Edycja tekstu	192
Podsumowanie	193
Przegląd rozdziału w pytaniach	194

Część III **Pisanie poprawnych gier**

10. Używanie metod języka C# do rozwiązywania problemów	197
Wprowadzenie	197
Zabawa z obrazami	197
Przybliżanie obrazu	198
Tworzenie efektu oddalania	200
Aktualizowanie prostokąta rysowania	200
Tworzenie metody obliczającej wartości procentowe	203
Stosowanie słowa void	206
Diagnozowanie programów języka C#	210
Trafianie w punkt zatrzymania	212
Używanie liczb zmiennoprzecinkowych w języku C#	214
Kompilator i typy danych języka C#	215
Kompilatory i rzutowanie	217
Typy wyrażeń	218
Zatrzymywanie oddalania	220
Oddalanie ze środka zdjęcia	220
Podsumowanie	224
Przegląd rozdziału w pytaniach	224
11. Gra jako program języka C#	227
Wprowadzenie	227
Tworzenie grafiki gry	228
Projekty, zasoby i klasy	229
Rozwiązania i projekty środowiska XNA Game Studio	229
Plik Program.cs	232
Zmiana nazwy klasy Game1	238
Tworzenie obiektów gry	240
Sprajty w grze	240
Zarządzanie rozmiarem sprajtów gry	240
Wprawianie sprajtów w ruch	244
Odbijanie sera	246
Obsługa nadmiarowości ekranu	246
Podsumowanie	248
Przegląd rozdziału w pytaniach	249
12. Gry, obiekty i stan	251
Wprowadzenie	251
Dodanie chleba do gry	251
Stosowanie struktury do przechowywania informacji o sprajcie	252
Sterowanie ruchem za pomocą gałki pada	254

Doskonalenie programów za pomocą metod	256
Obsługa kolizji	259
Odbijanie sera od chleba	260
Dziwne zachowanie mechanizmu odbijania	260
Dziwne zachowanie krawędzi	261
Dodanie pomidorowych celów	264
Kolizje z pomidorami	267
Podsumowanie	269
Przegląd rozdziału w pytaniach	270
13. Tworzenie kompletnej gry	271
Wprowadzenie	271
Tworzenie skończonej gry	271
Dodanie punktacji do gry	271
Dodanie elementu ryzyka	273
Dodanie postępu w grze	275
Doskonalenie projektu kodu	278
Refaktoryzacja poprzez tworzenie metod dla istniejącego kodu	279
Refaktoryzacja poprzez zmianę identyfikatorów	280
Tworzenie obszarów kodu	282
Tworzenie przydatnych komentarzy	283
Dodanie tła	285
Dodanie ekranu tytułowego	286
Gry i stan	287
Stosowanie wartości stanu	287
Budowa maszyny stanów	288
Podsumowanie	291
Przegląd rozdziału w pytaniach	291
14. Klasy, obiekty i gry	293
Wprowadzenie	293
Projektowanie przy użyciu obiektów	293
Kurs odświeżający wiedzę o obiektach	294
Spójność i obiekty	295
Związki pomiędzy obiektami	298
Projektowanie interakcji pomiędzy obiektami	301
Obiekty kontenerów	303
Obiekty tła i ekranu tytułowego	305
Klasy i struktury	306
Tworzenie i stosowanie struktury	306
Tworzenie i stosowanie egzemplarza klasy	307

Referencje	309
Wiele referencji do jednego egzemplarza	309
Brak referencji do egzemplarza	310
Po co w ogóle stosować referencje?	311
Typy wartościowe i referencyjne	311
Czy obiekty w naszej grze powinny mieć postać klas, czy też struktur?	312
Tworzenie hierarchii klas sprajtów	314
Klasa BaseSprite	314
Rozszerzenie klasy BaseSprite w celu utworzenia klasy TitleSprite	316
Budowa hierarchii klas	317
Dodanie morderczej papryki	318
Tworzenie klasy DeadlySprite	318
Podsumowanie	323
Przegląd rozdziału w pytaniach	323
15. Tworzenie komponentów gry	325
Wprowadzenie	325
Obiekty i abstrakcja	325
Tworzenie klasy abstrakcyjnej w języku C#	326
Rozszerzanie klasy abstrakcyjnej	327
Projektowanie przy użyciu klas abstrakcyjnych	328
Referencje do abstrakcyjnych klas macierzystych	329
Konstruowanie egzemplarzy klas	330
Konstruktory w strukturach	332
Konstruktory w hierarchii klas	333
Dodanie stu zabójczych mandarynek	335
Tworzenie klasy KillerSprite	335
Ustawianie położenia sprajtów KillerSprite za pomocą liczb losowych	336
Stosowanie list referencji	339
Dodanie sztucznej inteligencji	342
Ściganie pałki chlebowej	343
Dodanie dźwięków w grze	347
Od obiektów do komponentów	350
Interfejsy języka C#	351
Tworzenie interfejsu	352
Implementowanie interfejsu	353
Referencje do interfejsów	353
Łączenie chleba, sera i pomidorów	354
Projektowanie przy użyciu interfejsów	354
Podsumowanie	355
Przegląd rozdziału w pytaniach	355

16. Tworzenie gier sieciowych dla wielu graczy	357
Wprowadzenie	357
Sieci i komputery	357
Wszystko zaczyna się od sygnału	358
Konstruowanie pakietów	358
Adresowanie komunikatów	358
Trasowanie	359
Połączenia i datagramy	360
Sieci i protokoły	361
Usługa Xbox Live	362
Karty graczy i usługa Xbox Live	362
Technologia System Link i framework XNA	363
Ping-pong chleba i sera	363
Zarządzanie profilami graczy we frameworku XNA	364
Sprawdzanie logowania gracza na potrzeby gry sieciowej	368
Tworzenie lobby gry	369
Gry sieciowe i stan	369
Rozgrywka	377
Kompletna gra	382
Podsumowanie	382
Przegląd rozdziału w pytaniach	383

Część IV **Tworzenie mobilnych gier dla systemu Windows Phone 7 za pomocą XNA** **385**

17. Gry sterowane ruchem telefonu	387
Wprowadzenie	387
Akcelerometr	387
Do czego tak naprawdę służy akcelerometr?	387
Przyspieszenie z punktu widzenia fizyki	388
Interpretowanie odczytów z akcelerometru	390
Tworzenie gry Cheese Lander sterowanej przechyleniem	391
Obiekty świata gry Cheese Lander	391
Dostęp do klasy Accelerometer z programu w XNA	392
Używanie klasy Accelerometer w grze dla środowiska XNA	395
Uruchamianie akcelerometru	397
Wykorzystanie odczytów z akcelerometru w programie gry	398
Poprawianie przebiegu gry z wykorzystaniem praw fizyki	398
Wyrażanie ruchu za pomocą wektorów	400
Dodawanie tarcia	401
Sterowanie dźwiękiem za pomocą wektorów	402

Wykrywanie potrząśnięcia	403
Krótka dygresja na temat wątków i synchronizacji	403
Podsumowanie	405
Przegląd rozdziału w pytaniach	406
18. Obsługa sterowania dotykiem	407
Wprowadzenie	407
Ekran dotykowy w telefonach z systemem Windows Phone	407
Pobieranie wejścia dotykowego	408
Tworzenie przycisku alarmowego	408
Odczytywanie zdarzeń z wejścia dotykowego	409
Typy obiektów TouchLocation	409
Obsługa miejsca dotknięcia	411
Tworzenie dotykowej perkusji	412
Tworzenie klasy SoundPad dla każdego z dźwięków perkusji	412
Przechowywanie w programie wartości typu soundPad	413
Wyświetlanie przycisków	414
Odświeżanie stanu przycisków	415
Podświetlanie przycisków	416
Tworzenie „ślizganej” gry planszowej	418
Klasa PuckSprite	419
Odświeżanie stanu nieruchomego krążka	420
Przemieszczanie krążka po ekranie	421
Krążek poruszający się po planszy	422
Emulatory i prawdziwe urządzenia	424
Podsumowanie	425
Przegląd rozdziału w pytaniach	425
19. Konstruowanie mobilnych gier	427
Wprowadzenie	427
Telefon z systemem Windows Phone	427
Windows Phone Marketplace	428
Wydłużanie czasu życia baterii telefonu podczas działania gier w XNA	428
Ustawianie częstotliwości odświeżania w grze	428
Obsługa zmian orientacji telefonu	429
Wybór orientacji z programu gry w XNA	430
Otrzymywanie komunikatu o zmianie orientacji	430
Używanie określonego rozmiaru ekranu w grach dla systemu Windows Phone	432
Ukrywanie paska stanu systemu Windows Phone	432
Wyłączanie blokowania ekranu podczas gry	433

Tworzenie maszyny stanów dla telefonu	434
Gry i stany	434
Tworzenie prostej maszyny stanów dla gry	435
Tworzenie bardziej skomplikowanych maszyn stanów	437
Obsługa przychodzących połączeń telefonicznych	439
Wykrywanie połączeń telefonicznych	441
Gra jako aplikacja w systemie Windows Phone	442
Przyciski Back i Start systemu Windows Phone	443
Uruchamianie nowych programów za pomocą przycisku Start	445
Korzystanie z izolowanych obszarów przechowywania do zapisywania stanu gry	446
Dostarczanie naszych gier do usługi Marketplace	453
Usługa Windows Phone Marketplace	453
Rejestracja w serwisie App Hub	454
Używane urządzenia z systemem Windows Phone	454
Tworzenie gier na sprzedaż	454
Podsumowanie	455
Przegląd rozdziału w pytaniach	455
 Dodatki	 457
 Odpowiedzi na pytania do rozdziałów	 459
Słownik pojęć	483
Skorowidz	511
O autorze	525

Rozdział 3.

Pobieranie danych wejściowych od gracza

W tym rozdziale:

- Dowiesz się, jak we frameworku Microsoft XNA są reprezentowane pady i klawiatury.
- Odkryjesz struktury języka programowania C#, które umożliwiają uzyskiwanie danych wejściowych od gracza.
- Napiszesz kilka naprawdę niezbyt mądrych gier, które skutecznie przestraszą graczy.

Wprowadzenie

Poznałeś już podstawy programowania gier komputerowych. Wiesz, że program jest w istocie sekwencją wyrażeń, z których każde wykonuje pojedynczą czynność. Przekonałeś się, że wyrażenia umieszcza się w metodach odpowiedzialnych za określone zadanie oraz że metody (wraz z niezbędnymi danymi) należą do klas. Sam program operuje na wartościach danych, które są przechowywane w zmiennych określonych typów. Program podejmuje też decyzje na podstawie wartości zawartych w tych zmiennych. (Jeśli uważasz, że żadne z powyższych zdań nie ma sensu, czytaj tak długo rozdział 2., zatytułowany „Programy, dane i ładne kolory”, aż te stwierdzenia będą dla Ciebie oczywiste).

W tym rozdziale będziesz miał okazję poszerzyć swoją wiedzę o techniki uzyskiwania danych wejściowych ze świata zewnętrznego, dzięki którym gry mogą reagować na zachowania gracza. Szybko odkryjesz, że po uzyskaniu tej wiedzy spektrum Twoich możliwości w roli twórcy gier komputerowych będzie nieporównanie szersze. Będziesz potrafił napisać kilka niezbyt jeszcze przemyślanych gier, w tym *Color Nerve*, *Mind Reader*, *The Thing That Goes Bump in the Night* oraz *Gamepad Racer*.

Zanim jednak przystąpimy do omawiania padów, musimy zdecydować, jak powinien działać Twój przyszły program. Przeanalizuj następujące wyrażenie języka C# zaczerpnięte z metody `Update` w ramach poprzedniej wersji programu nastrojowej lampy:

```
if (redCountingUp) redIntensity++;
```

To jeden z testów umożliwiających kontrolę intensywności czerwonego składnika wyświetlanego koloru. Wyrażenie w tej formie można opisać słowami: „Jeśli zmienna logiczna `redCountingUp` ma wartość `true`, zwiększ wartość zmiennej `redIntensity` o 1”.

Projekt programu: kontroler wielokolorowej lampy

W rozdziale 2. stworzyłeś prostą grę — swoiste światło, które zmieniało kolor w czasie. Wspomniałem też, że podobne rozwiązania najpewniej będą w przyszłości stosowane na statkach kosmicznych. Lampa zmieniająca kolor być może nie nadaje się do czytania książek, ale skutecznie poprawia nastrój. Przyjmijmy teraz, że kapitan naszego statku kosmicznego potrzebuje lampy, której kolor sam będzie mógł ustawić. Oznacza to, że tym razem przygotujesz lampę sterowaną za pośrednictwem pada konsoli Xbox. Użytkownik będzie naciskał czerwony, niebieski, zielony i żółty przycisk na padzie, aby zwiększać intensywność poszczególnych barw. Aby stworzyć opisane rozwiązanie, musisz dowiedzieć się, jak odczytywać stan pada.

Przytoczone wyrażenie jest przetwarzane dla każdego wywołania metody `Update` (obecnie ta częstotliwość wynosi 60 razy na sekundę), zatem jeśli zmienna `redCountingUp` ma wartość `true`, intensywność czerwonej barwy będzie stopniowo zwiększana.

Chcesz teraz napisać kod, który będzie wyrażał zdanie: „Jeśli na pierwszym padzie jest naciśnięty czerwony przycisk, zwiększ wartość zmiennej `redIntensity` o 1”. Jeśli gracz będzie przytrzymywał czerwony przycisk, ekran będzie stawał się coraz bardziej czerwony. Oznacza to, że jedyna niezbędna zmiana będzie polegała na takiej modyfikacji przytoczonego testu, aby odczytywał on stan przycisku na padzie. Stworzenie lampy sterowanej przez użytkownika jest więc zadziwiająco proste.

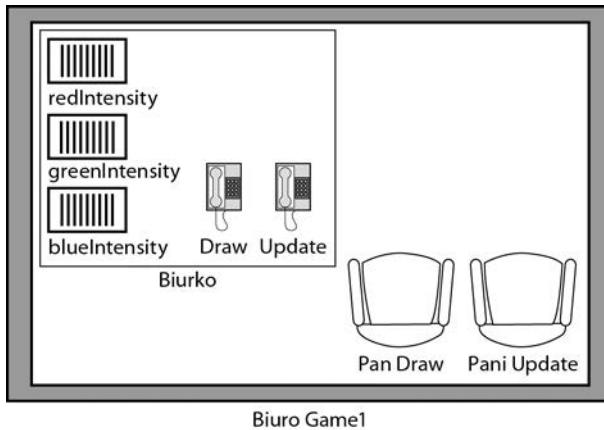
Odczytywanie stanu pada

Pady w rzeczywistości są bardzo skomplikowanymi urządzeniami. Są połączone z głównym urządzeniem albo za pośrednictwem magistrali USB (od ang. *universal serial bus*), albo przez połączenie bezprzewodowe. Sposób współpracy programów z padami nie zależy od sposobu ich połączenia z konsolą czy komputerem. Połączenie z padem można wykorzystać zarówno do odczytywania stanu przycisków i gałek analogowych, jak i do wysyłania poleceń do pada (na przykład w celu włączenia lub wyłączenia efektu wibracji). Konsola Xbox i framework XNA oferują obsługę maksymalnie czterech podłączonych jednocześnie padów.

Pady i klasy

Informacje o stanie pada są reprezentowane we frameworku XNA przez składowe klasy nazwanej `GamePadState`. Zadaniem tej klasy jest zapewnianie połączenia pomiędzy programem a fizycznym padem w dłoniach gracza. Aby zrozumieć, jak należy używać tej klasy, musisz pogłębić swoją wiedzę na temat działania wszystkich klas.

Z lektury punktu „Gry i klasy” zamieszczonego w rozdziale 2. wiesz już, czym jest klasa. Zawiera ona dane (zmienne, które mogą reprezentować rozmaite informacje) oraz metody (kod wykonujący operacje). Klasę możesz traktować jako odpowiednik biura z centralnie ustawionym biurkiem, na którym leżą zmienne, oraz pracownikami pełniącymi podobne funkcje jak metody. Na rysunku 3.1 pokazano plan biura dla klasy `Game1`, która — jak już wiesz — jest podstawowym elementem gry opartej na frameworku XNA.



RYSUNEK 3.1. Klasa `Game1` jako plan biura

Klasa zawiera pewne zmienne trzymane na biurku (w tym przypadku zmienne reprezentują intensywność poszczególnych barw) oraz dwie metody, które nazwaliśmy panem `Draw` i panią `Update`. Każda metoda dysponuje własnym telefonem. Programy mogą dzwonić na te telefony, aby żądać od poszczególnych metod wykonywania swoich zadań.

Wielka programistka mówi: w rzeczywistości klasy to nie biura Nasza wielka programistka przeczytała ten tekst i stwierdziła, że jest śmieszny. Mówi, że klasy nie przypominają biur, ale uznała, że skoro to porównanie ma ułatwić zrozumienie sposobu konstruowania programów, w takim postrzeganiu klas nie ma niczego złego.

Podczas uruchamiania gry system XNA tworzy **egzemplarz** (ang. *instance*) klasy `Game1`, który będzie bezpośrednim adresatem żądań wykonania metod `Draw` i `Update`. Podczas tworzenia egzemplarza klasy instrukcje zawarte w metodach tej klasy są ładowane do pamięci, a na potrzeby zmiennych należących do tego egzemplarza jest przydzielana przestrzeń w pamięci.

Pliki klas stanowią źródło swoistych planów klas, dzięki czemu podczas działania programu istnieje możliwość tworzenia egzemplarzy poszczególnych klas. W otaczającym nas świecie stworzenie biura gry wymaga budowy pomieszczenia, umieszczenia w nim biurka i kilku telefonów oraz zatrudnienia pana `Draw` i pani `Update`. Proces tworzenia egzemplarza klasy

jest dość podobny. Okazuje się jednak, że dla oszczędności pamięci działający program posługuje się tylko jedną kopią kodu metody — ta jedna kopia jest współdzielona przez wszystkie egzemplarze tej samej klasy.



Uwaga Pamiętaj, że opisane operacje są realizowane w czasie wykonywania programu. Proces tworzenia egzemplarzy klas nie jest wykonywany przez kompilator. Zadaniem kompilatora jest konwersja kodu źródłowego języka C# na instrukcje, które będzie można wykonać na docelowym urządzeniu. W momencie, w którym Twój program przejmuje kontrolę nad tym urządzeniem, rola kompilatora się kończy, a komputer wykonuje wygenerowane przez ten kompilator rozkazy języka maszynowego.

Znajdowanie pada

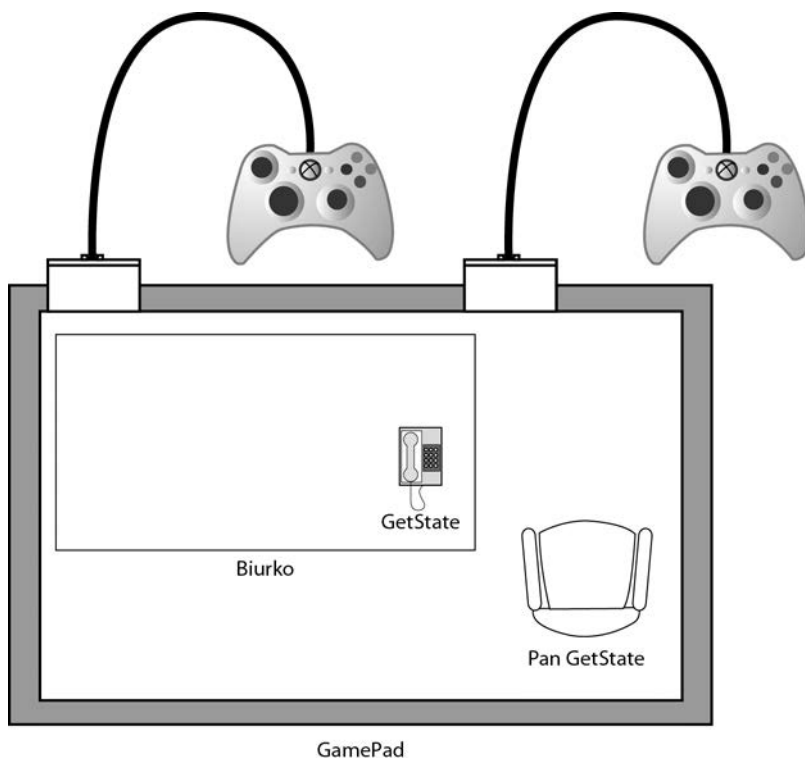
Framework XNA przejmuje odpowiedzialność także za wiele innych aspektów działania gry. Jednym z dostępnych rozwiązań jest klasa `GamePad` połączona ze wszystkimi padami. Nie musisz wiedzieć, jak pad jest fizycznie połączony z urządzeniem. Możesz postrzegać ten proces jako mikroskopijne chochliki wędrujące w przewodzie i przenoszące na jeszcze mniejszych karteczkach komunikaty, np: „Pierwszy gracz nacisnął czerwony przycisk”. Na rysunku 3.2 pokazano, jak wyglądałaby klasa `GamePad`, gdyby była biurem.

Klasa `GamePad` zawiera metodę, nazwaną `GetState`, która zwraca bieżący stan jednego z padów. Po wywołaniu metody `GetState` jej kod lokalizuje jeden z padów, odczytuje jego ustawienia, po czym wysyła uzyskane informacje z powrotem do wyrażenia, które wywołało tę metodę (i które może teraz użyć tych danych).

Metoda `GetState` otrzymuje na wejściu parametr identyfikujący pad, którego stan ma być odczytany. **Parametr** ten umożliwia przekazywanie informacji do metody z poziomu jej wywołania. Z parametrami mieliśmy do czynienia już wcześniej — w Twoim pierwszym programie, gdzie przekazywałeś parametry typu `Color` do metody `Clear`, aby wskazać kolor, który miał być użyty do wypełnienia ekranu.

W przypadku metody `GetState` parametr identyfikuje pad, którego stan chcesz odczytać. Jeśli porównujesz ten mechanizm z funkcjonowaniem biur, możesz traktować parametry jako fragmenty instrukcji przekazywanych współpracownikom przez telefon. Kiedy pan `GetState` odbiera telefon, słyszy w słuchawce: „Podaj mi stan pierwszego pada”. Informacje na temat stanu tego pada są następnie odsyłane do struktury `GamePadState` (patrz rysunek 3.3).

Jeśli chcesz, możesz traktować parametry jako zbiór elementów wypełnionego formularza, jednak w rzeczywistości jest to struktura języka C# zawierająca zarówno składowe widoczne na rysunku 3.3, jak i pewne dane dodatkowe.

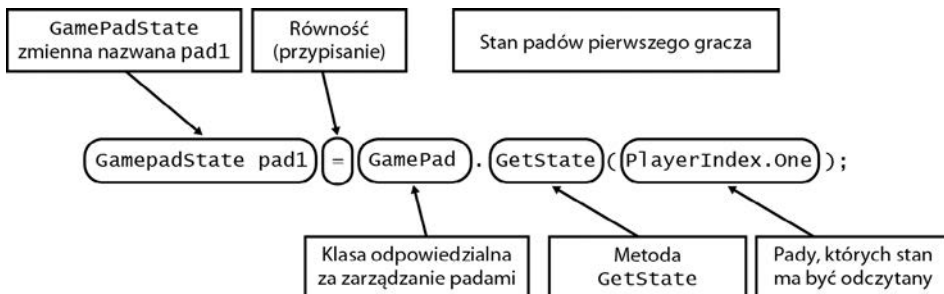


RYСУNEK 3.2. Klasa GamePad jako biuro

GamePadState	
Przyciski	
Zielony (A)	ButtonState.Pressed
Czerwony (B)	ButtonState.Released
Niebieski (X)	ButtonState.Released
Żółty (Y)	ButtonState.Released
Start	ButtonState.Released
Back	ButtonState.Released

RYСУNEK 3.3. Struktura GamePadState reprezentująca stan pada z naciśniętym zielonym przyciskiem (A)

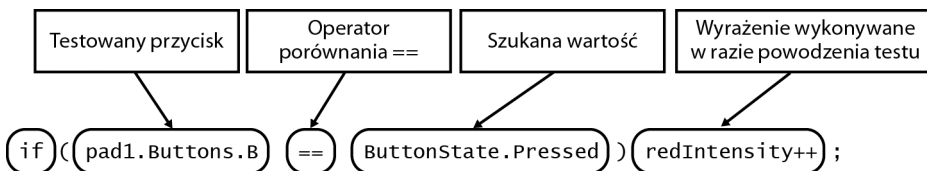
Jeśli więc pani Update chce uzyskać informacje na temat stanu jednego z padów konsoli Xbox, wywołuje metodę GetState klasy GamePad i prosi: „Czy możesz mi przekazać stan pada pierwszego gracza”? Pan GetState zrywa się z fotela, wypełnia formularz GamePadState i odsyła go do pani Update. Na rysunku 3.4 pokazano wyrażenie języka C# uzyskujące stan pada i przypisujące ten stan do zmiennej typu GamePadState.



RYSUNEK 3.4. Uzyskiwanie statusu pada

Testowanie statusu pada

Skoro dysponujesz już statusem pada, możesz wykorzystać ten status w swoim programie do sprawdzenia, czy gracz wcisnął jakiś przycisk. Na rysunku 3.5 pokazano kompletne wyrażenie języka C#, które wykonuje odpowiedni test.



RYSUNEK 3.5. Testowanie przycisku na padzie

Wyrażenie w tej formie porównuje stan czerwonego przycisku (`B`) z wartością `ButtonState.Pressed`. Jeśli obie wartości są równe, przycisk jest wciśnięty, zatem metoda `Update` powinna zwiększyć intensywność barwy czerwonej. Tych samych zasad możesz używać do zarządzania wartościami reprezentującymi intensywność barw niebieskiej i zielonej, co oznacza, że w swojej grze możesz posługiwać się metodą `Update` zawierającą następujący kod:

```
protected override void Update(GameTime gameTime)
{
    // Umożliwia wyjście z gry.
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    GamePadState pad1 = GamePad.GetState(PlayerIndex.One);

    if (pad1.Buttons.B == ButtonState.Pressed) redIntensity++;
    if (pad1.Buttons.X == ButtonState.Pressed) blueIntensity++;
    if (pad1.Buttons.A == ButtonState.Pressed) greenIntensity++;

    base.Update(gameTime);
}
```

W opisanym metodzie `Update` brakuje już tylko obsługi żółtego przycisku. Gdy gracz wciska żółty przycisk, program powinien zwiększyć intensywność barw zielonej i żółtej, zatem w przypadku spełnienia odpowiedniego warunku musi wykonać dwa wyrażenia. Okazuje się, że realizacja tego zadania jest bardzo łatwa — wystarczy umieścić te dwa wyrażenia w bloku kontrolowanym przez odpowiedni warunek, tak jak w poniższym kodzie:

```
if (pad1.Buttons.Y == ButtonState.Pressed)
{
    redIntensity++;
    greenIntensity++;
}
```

Z blokami miałeś do czynienia już wcześniej. Blokiem jest na przykład ciało metody (sekwencja wyrażen składających się na tę metodę). W języku programowania C# **blok** ma postać pewnej liczby wyrażen otoczonych nawiasami klamrowymi. Jeśli warunek w wyrażeniu `if` jest spełniony (ma wartość `if`), powyższy kod wykonuje oba wyrażenia, ponieważ znajdują się one w bloku kontrolowanym przez ten warunek.

Wielka programistka mówi: bloki to podstawa Nasza wielka programistka stara się stosować bloki za wyrażeniami `if`, nawet jeśli formalnie nie jest to konieczne. Mówi, że takie rozwiązanie poprawia czytelność tekstu programu i że znacznie ułatwia dodawanie ewentualnych wyrażen w przyszłości (jeśli zajdzie taka potrzeba).

Jeśli umieścisz powyższe wyrażenia w metodzie `Update` w ramach jednej z wcześniejszych wersji programu *Mood Light*, otrzymasz komunikaty ostrzeżeń kompilatora, ponieważ nowa wersja metody `Update` nie będzie używała wszystkich zmiennych utworzonych w starszych wersjach programu. Aby wyeliminować te ostrzeżenia, musisz usunąć wyrażenia tworzące nieużywane zmienne. Wielka programistka nie lubi, gdy programy zawierają zmienne, które nigdzie nie są używane. Mówi, że to wygląda nieprofesjonalnie, z czym trudno się nie zgodzić.

Przykładowy kod: ręczna, wielokolorowa lampa Wszystkie przykładowe projekty są dostępne pod adresem <ftp://ftp.helion.pl/przyklady/games4.zip>. Przykładowy projekt z omówioną w tym punkcie metodą `Update` znajduje się w katalogu *01 Manual MoodLight* w zasobach kodu źródłowego dla tego rozdziału. Możesz zwiększać jasność kolorów na ekranie, naciskając przyciski na swoim padzie.

Stosowanie klawiatury

Framework XNA obsługuje nie tylko pady, ale też klawiatury. Wielu użytkowników ze zdziwieniem odkrywa możliwość podłączenia klawiatury USB do konsoli Xbox 360 i używania jej tak jak w przypadku komputerów PC. Jeśli chcesz napisać program

Pomysł na grę: Color Nerve

Od czasu do czasu będziemy podejmowali próby tworzenia nowych koncepcji gier. Początkowo nasze gry będą bardzo proste, ale z czasem tworzone programy staną się coraz bardziej skomplikowane i interesujące. Do utworzenia swojej pierwszej gry możesz użyć kodu zawartego w projekcie *Manual MoodLight*. Gra wykorzystuje też pewien aspekt opisany w rozdziale 2. Jak zapewne pamiętasz, jeśli stale zwiększasz jakąś wartość, prędzej czy później osiąga ona punkt, w którym nie mieści się w przydzielonym dla niej obszarze pamięci, powodując tzw. przepełnienie. Właśnie dlatego ekran wypełniony jasnym, białym kolorem nagle stawał się czarny. Okazuje się, że ten sam efekt możesz wykorzystać do utworzenia swojej pierwszej „bardzo niemądrej gry”.

Color Nerve to gra dla co najmniej dwóch graczy. Każdy gracz w swojej kolejce naciska na padzie jeden lub wiele przycisków. (Pozostali gracze muszą uważnie obserwować ekran, aby mieć pewność, że gracz, do którego należy ruch, rzeczywiście naciśnie przynajmniej jeden przycisk). Każdy gracz może nacisnąć w swojej kolejce dowolną liczbę przycisków. Jeśli jednak ekran nagle zmieni kolor (z powodu zmiany intensywności przynajmniej jednej z barw z 255 na 0), gracz przegrywa, a dalszą rozgrywkę prowadzą pozostali gracze. Ostatni gracz, który pozostaje w grze, jest zwycięzcą.

Gra może mieć przebieg taktyczny. Przyciski można naciskać bardzo krótko, ale niektórzy gracze na początku gry mogą wykazać się mocnymi nerwami i przytrzymywać przyciski przez dłuższy czas, wpędzając w tarapaty pozostałych graczy. Mogą oni też podejmować próby odgadywania, która barwa „przekręciła licznik”, tak aby nieco dłużej przytrzymać odpowiedni przycisk w swoim ruchu. Gra sprawdza się na imprezach, ponieważ może w niej uczestniczyć wiele osób, a jej zasady są wyjątkowo proste. W rozdziale 4., zatytułowanym „Wyświetlanie obrazów”, udoskonalisz tę grę, dodając obrazy w miejsce pustego ekranu.

współpracujący z klawiaturą, wystarczy, że dodasz do swojego programu odpowiedni kod, na przykład taki jak poniżej:

```
KeyboardState keys = Keyboard.GetState();

if (keys.IsKeyDown(Keys.R)) redIntensity++;
if (keys.IsKeyDown(Keys.B)) blueIntensity++;
if (keys.IsKeyDown(Keys.G)) greenIntensity++;
if (keys.IsKeyDown(Keys.Y))
{
    redIntensity++;
    greenIntensity++;
}
```

Łatwo zauważyć, że prezentowany proces bardzo przypomina działanie padów, ale istnieją też nieznaczące różnice. Na potrzeby metody `GetState` nie musisz określać klasy `Keyboard`, dla której klawiatury chcemy uzyskać stan, ponieważ framework XNA obsługuje tylko jedną klawiaturę. Egzemplarz klasy `KeyboardState` zwracany przez to wywołanie

w rzeczywistości nie jest kartką papieru — jest obiektem udostępniającym metody, które możesz w swoim programie wykorzystać do określenia, czy poszczególne klawisze są wciśnięte. Zamiast sprawdzać, czy stan przycisku ma przypisaną wartość `ButtonState.Pressed`, program może wykonać metodę `IsKeyDown`. Na wejściu metody `IsKeyDown` przekazujesz parametr identyfikujący klawisz, którego stan chcesz sprawdzić, jak w poniższym przykładzie:

```
if (keys.IsKeyDown(Keys.R)) redIntensity++;
```

Kod jest wyrażeniem warunkowym zwiększającym wartość zmiennej `redIntensity`, w przypadku gdy klawisz `R` jest wciśnięty. Jeśli wskazany klawisz jest wciśnięty, metoda `IsKeyDown` zwraca wartość `true` — w przeciwnym razie zwraca wartość `false`. Oznacza to, że możemy wykorzystać to wyrażenie do sterowania aktualizowaniem wartości zmiennej `redIntensity`.

Zatrzymywanie gry za pomocą klawisza `Escape`

Metoda `Update` generowana w czasie tworzenia nowej gry na bazie frameworku XNA zawiera test, który sprawdza, czy na pierwszym padzie wciśnięto przycisk `Back` (wstecz), oraz wywołuje metodę `Exit` (zatrzymującą grę), jeśli ten warunek jest spełniony. Jeśli używasz klawiatury zamiast pada, nie będziesz mógł nacisnąć tego przycisku i — tym samym — zatrzymać gry. Możesz jednak dodać test sprawdzający, czy na klawiaturze nie wciśnięto klawisza `Escape`. Klawisz `Escape` należy do tzw. klawiszy sterujących (ang. *control key*). Nie reprezentują one znaków drukowalnych, a jedynie mają sygnalizować działania, których oczekujemy od programu. Innymi przykładami klawiszy sterujących są `Enter` i `Backspace`. Do sprawdzania, czy naciśnięto klawisz `Escape`, służy ta sama metoda `IsKeyDown`.

```
if (keys.IsKeyDown(Keys.Escape)) Exit();
```

Powyższy kod zatrzymuje grę w momencie naciśnięcia klawisza `Escape`.

Jednoczesne używanie pada i klawiatury

Jeśli chcesz jednocześnie używać pada i klawiatury, musisz testować stan obu tych urządzeń. Oznacza to, że metoda `Update` powinna mieć teraz następującą postać:

```
protected override void Update(GameTime gameTime)
{
    GamePadState pad1 = GamePad.GetState(PlayerIndex.One);

    if (pad1.Buttons.Back == ButtonState.Pressed) Exit();
    if (pad1.Buttons.B == ButtonState.Pressed) redIntensity++;
    if (pad1.Buttons.X == ButtonState.Pressed) blueIntensity++;
    if (pad1.Buttons.A == ButtonState.Pressed) greenIntensity++;
}
```

```

    if (pad1.Buttons.Y == ButtonState.Pressed)
    {
        redIntensity++;
        greenIntensity++;
    }

    KeyboardState keys = Keyboard.GetState();

    if (keys.IsKeyDown(Keys.Escape)) Exit();

    if (keys.IsKeyDown(Keys.R)) redIntensity++;
    if (keys.IsKeyDown(Keys.B)) blueIntensity++;
    if (keys.IsKeyDown(Keys.G)) greenIntensity++;
    if (keys.IsKeyDown(Keys.Y))
    {
        redIntensity++;
        greenIntensity++;
    }
    base.Update(gameTime);
}

```

Kod w tej formie jest niedoskonały, ponieważ wykonujemy tę samą czynność dwukrotnie, tyle że w wyniku dwóch różnych testów. Gdyby wielka programistka miała kiedykolwiek okazję zapoznać się z tym kodem, z pewnością nie byłaby zachwycona. Na szczęście język C# oferuje możliwość łączenia w programie dwóch warunków i wykonywania kodu, jeśli przynajmniej jeden z nich jest spełniony (ma wartość `true`). Warunki można łączyć za pomocą logicznego operatora alternatywy (*lub*), którego wynik jest prawdziwy, jeśli przynajmniej jeden z jego operandów ma wartość `true`. W kodzie programu operator zapisywany jest w formie dwóch pionowych kresek (`||`):

```

GamePadState pad1 = GamePad.GetState(PlayerIndex.One);
KeyboardState keys = Keyboard.GetState();

if (pad1.Buttons.B == ButtonState.Pressed ||
    keys.IsKeyDown(Keys.R)) redIntensity++;

```

Operator logiczny *lub* (ang. *or*) umieszcza się pomiędzy dwoma wyrażeniami logicznymi, które mogą mieć albo wartość `true`, albo wartość `false`. Jeśli choć jedno z tych wyrażań ma wartość `true`, cały warunek logiczny także ma wartość `true`.

W powyższym kodzie wartość zmiennej `redIntensity` jest zwiększana, jeśli wciśnięty jest czerwony przycisk na padzie lub przycisk *R* na klawiaturze (lub oba jednocześnie). Dokładnie o takie działanie właśnie nam chodziło. Co więcej, w grę *Color Nerve* można teraz grać zarówno przy użyciu pada, jak i przy użyciu klawiatury (lub korzystając z obu tych urządzeń jednocześnie). W tym i innych przypadkach mówi się o operatorach logicznych, ponieważ każdy z nich zwraca wartość logiczną, a nie wynik liczbowy. Istnieją jeszcze inne operatory logiczne, których będziesz używał podczas tworzenia bardziej skomplikowanych programów.



Uwaga Jeśli zrozumienie działania tego operatora logicznego sprawia Ci kłopot, wróć do istoty problemu, który próbujesz rozwiązać. Chcesz na przykład, aby Twój program wykonał pewne wyrażenie (w tym przypadku `redIntensity++`), wtedy gdy wciśnięto czerwony przycisk na padzie lub gdy wciśnięto klawisz `R` na klawiaturze. Jeśli przynajmniej jeden z tych warunków jest spełniony, wówczas posługujesz się operatorem `lub (|)` do połączenia obu testów i wykonania odpowiedniego wyrażenia.

Przykładowy kod: Color Nerve Przykładowy projekt implementujący tę grę można znaleźć w katalogu `02 Color Nerve` w zasobach dla tego rozdziału. Możesz zmieniać kolory na ekranie, naciskając przyciski na padzie lub klawisze na klawiaturze.

Dodawanie wibracji

Komunikacja pomiędzy padem a grą odbywa się w obu kierunkach. Twoje możliwości nie ograniczają się tylko do odczytywania stanu przycisków pada — możesz dodatkowo wysyłać do pada polecenia włączające lub wyłączające mechanizm wibracji. Także w tym przypadku nie musisz dysponować szczegółową wiedzą o sposobie dostarczania odpowiednich komunikatów. Wystarczy znajomość funkcji frameworku XNA używanych do sterowania efektami wibracji.

Oznacza to, że Twoja gra *Color Nerve* może być jeszcze bardziej wciągająca i atrakcyjna, jeśli pad będzie wibrował w czasie zbliżania się wartości intensywności barw do ich górnych granic. To ciekawe, jak podobne rozwiązania mogą poprawić oceny nawet najprostszych gier. W kolejnych kilku grach będziesz bardzo często korzystał z efektu wibracji.

Sterowanie wibracjami pada

Klasa `GamePad` udostępnia metodę, nazwaną `SetVibration`, która umożliwi programowi sterowanie silnikami odpowiedzialnymi za wibracje:

```
GamePad.SetVibration(PlayerIndex.One, 0, 1);
```

Metoda `SetVibration` używa trzech parametrów. Pierwszy parametr identyfikuje pad, który ma wibrować. Drugi parametr, który musi mieć wartość z przedziału od 0 do 1, steruje natężeniem wibracji generowanych przez lewy silnik. Im większa jest ta liczba, tym bardziej intensywne są wibracje pada. Trzeci parametr steruje prawym silnikiem dokładnie tak samo, jak drugi parametr steruje lewym silnikiem. Powyższe wyrażenie powoduje, że prawy silnik pierwszego pada wibruje z pełną prędkością., natomiast lewy silnik jest odpowiedzialny za wibracje z niską częstotliwością.

Jeśli sobie wyobrazisz, że biuro `GamePad` zatrudnia pana `SetVibration`, będziesz mógł przyjąć, że ten pracownik otrzymuje polecenia ustawiania wibracji konkretnego pada

wraz z parametrami obrotów lewego i prawego silnika. Po wywołaniu tej metody pad zaczyna wibrować i wibruje tak długo, aż kolejne wywołanie tej metody nie zmieni ustawień jego silników. Innymi słowy, metodę `SetVibration` można traktować jako przełącznik, który można ustawiać w wielu różnych pozycjach. Początkowo oba silniki pada mają ustawioną szybkość równą 0, co oznacza brak wibracji.

Testowanie wartości intensywności

Gra musi oczywiście zdecydować, w którym momencie należy włączyć wibracje. W tym celu musi sprawdzać wartości reprezentujące intensywność barw i włączać silnik wibracji, jeśli któraś z tych wartości będzie zbyt duża. Program może na przykład włączać silniki, jeśli intensywność którejkolwiek z barw — czerwonej, zielonej lub niebieskiej — będzie większa niż 220. Oznacza to, że program musi sprawdzać wartości intensywności w następujący sposób:

```
if (redIntensity > 220)
{
    GamePad.SetVibration(PlayerIndex.One, 0, 1);
}
```

Powyższy kod zawiera nową formę warunku. We wcześniejszych przykładach posługiwaliśmy się warunkami sprawdzającymi, czy dwie wartości były sobie równe. Tym razem kod sprawdza, czy jedna wartość jest większa od drugiej. Znak większości (>) jest kolejnym przykładem operatora logicznego. Operator umieszczony pomiędzy dwiema wartościami zwraca wartość `true`, jeśli wartość na lewo od tego operatora jest większa od wartości na prawo od operatora (w przeciwnym razie operator zwraca wartość `false`). Dokładnie takiego działania oczekiwałeś.

Powyższy kod powoduje, że w momencie przekroczenia wartości 220 przez zmienną reprezentującą intensywność barwy czerwonej pad zaczyna wibrować z powodu działania prawego silnika wibrującego. Jeśli dodasz ten kod do metody `Update` gry *Color Nerve*, przekonasz się, że pad zaczyna wibrować po zwiększeniu intensywności koloru czerwonego. Program zawiera jednak pewien błąd. Gdy intensywność barwy czerwonej spada do poziomu 0, pad nie przestaje wibrować. Musisz jeszcze dodać kod, który wyłączy silnik w momencie, w którym wartość intensywności koloru spadnie poniżej poziomu 220. Okazuje się, że jest to bardzo proste — wystarczy dodać część `else` do istniejącego warunku:

```
if (redIntensity > 220)
{
    GamePad.SetVibration(PlayerIndex.One, 0, 1);
}
else
{
    GamePad.SetVibration(PlayerIndex.One, 0, 0);
}
```


Wyrażenie następujące po słowie `else` jest wykonywane wówczas, jeśli warunek nie jest spełniony, czyli jeśli ma wartość `false`. (Część `else` możesz dodać do dowolnego utworzonego przez siebie warunku `if`). Oznacza to, że gdy intensywność barwy czerwonej wraca do poziomu 0, pad przestaje wibrować. Możesz dodatkowo rozszerzyć te testy za pomocą operatora **lub** (`||`), tak aby program testował wartości reprezentujące intensywność wszystkich barw:

```
if ( redIntensity > 220 ||
    greenIntensity > 220 ||
    blueIntensity > 220 )
{
    GamePad.SetVibration(PlayerIndex.One, 0, 1);
}
else
{
    GamePad.SetVibration(PlayerIndex.One, 0, 0);
}
```

Wibracje pada są teraz zależne od wszystkich wartości reprezentujących intensywność barw. Grę możesz dodatkowo udoskonalić, eksperymentując z różnymi rodzajami wibracji dla różnych kolorów, a nawet dodatkowo korzystając z silnika niskiej częstotliwości. Za sterowanie tym silnikiem odpowiada drugi parametr metody `SetVibration`:

```
GamePad.SetVibration(PlayerIndex.One, 1, 0);
```

Powyższy wiersz kodu włącza wibracje o niskiej częstotliwości. Możesz też eksperymentować z odmiennymi programami uruchamiania wibracji.

Program wciąż zawiera jednak pewne niedociągnięcie. Jeśli po jego uruchomieniu zostaną włączone wibracje pada, nie wyłączą się one same w momencie zakończenia programu. Musisz więc dodać kod wyłączający wibracje z chwilą opuszczania gry. Gra kończy się wtedy, gdy gracz naciska przycisk *Back* na swoim padzie. Odpowiedni test tego przycisku znajduje się w metodzie `Update`. Jeśli przycisk *Back* jest wciśnięty, następuje wywołanie metody `Exit`, która zatrzymuje i wyłącza grę:

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
    this.Exit();
```

Metoda `Exit` służy do eleganckiego usunięcia z ekranu elementów wizualnych gry i wyłączenia samego programu. Do Twojego programu należy odpowiedzialność za wyłączenie silników pada przed wywołaniem tej metody, dlatego w przypadku wciśnięcia przycisku *Back* Twój program musi wykonać dodatkowe wyrażenie. Potrzebujemy zatem kolejnego bloku:

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
{
    GamePad.SetVibration(PlayerIndex.One, 0, 0);
    this.Exit();
}
```

Teraz, gdy gracz naciśnie przycisk *Back*, aby zakończyć program, silniki wibrujące pada zostaną wyłączone.

Wielka programistka mówi: w razie wątpliwości sprawdź to sam Wielka programistka uważa, że jeśli znajdujesz się w sytuacji, w której nie jesteś pewien, czy jakiś warunek jest spełniony we wszystkich możliwych przypadkach, powinieneś dodać kod eliminujący potencjalne wątpliwości. Testując zachowanie opisanego w tym podrozdziale mechanizmu wibracji, odkryłem, że po opuszczeniu gry pad nadal wibruje tylko w starszych wersjach frameworku XNA, natomiast w nowszych wersjach wibracje są automatycznie wyłączone. Aby mieć absolutną pewność, że wibracje zostaną wyłączone niezależnie od wersji frameworku XNA, w której działa Twoja gra, powinieneś użyć powyższego kodu do samodzielnego wyłączenia wibracji.

Przykładowy kod: Color Nerve z wibracjami Katalog z kodem źródłowym dla tego projektu zawiera przykładowy projekt *03 Color Nerve with Vibes* z wersją gry *Color Nerve* korzystającą z efektu wibrowania padów.

Pomysł na grę: Tajne komunikaty w formie wibracji

Skoro już wiesz, że odczytywanie stanu przycisków pada i sterowanie jego silnikami wibrującymi jest dość proste, możesz przystąpić do tworzenia ciekawszych rozwiązań przy użyciu frameworku XNA, szczególnie jeśli dysponujesz padami bezprzewodowymi. Możesz tworzyć gry polegające na czytaniu w cudzych myślach, tak aby Twój asystent sprawiał wrażenie dysponowania umiejętnością precyzyjnego określenia, o czym aktualnie myślisz. Osoby postronne nie będą wiedziały, że obaj ukryliście w kieszeniach pady konsoli Xbox i że używacie ich do przesyłania sobie sygnałów w formie wibracji. Kod realizujący to działanie jest zadziwiająco prosty — powinieneś bez trudu poradzić sobie z jego zrozumieniem:

```
protected override void Update(GameTime gameTime)
{
    // Umożliwia wyjście z gry.
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
    {
        GamePad.SetVibration(PlayerIndex.One, 0, 0);
        GamePad.SetVibration(PlayerIndex.Two, 0, 0);
        this.Exit();
    }

    GamePadState pad1 = GamePad.GetState(PlayerIndex.One);
    GamePadState pad2 = GamePad.GetState(PlayerIndex.Two);

    if (pad1.Buttons.A == ButtonState.Pressed)
    {
```

```

GamePad.SetVibration(PlayerIndex.Two, 0, 1);
}
else
{
    GamePad.SetVibration(PlayerIndex.Two, 0, 0);
}

if (pad2.Buttons.A == ButtonState.Pressed)
{
    GamePad.SetVibration(PlayerIndex.One, 0, 1);
}
else
{
    GamePad.SetVibration(PlayerIndex.One, 0, 0);
}

base.Update(gameTime);
}

```

Metoda `Update` odczytuje stan przycisku `A` na padzie pierwszego gracza. Jeśli przycisk jest wciśnięty, włącza szybki silnik wibracji w padzie drugiego gracza. Program powtarza ten sam proces w przeciwnym kierunku, przysyłając sygnał z drugiego pada do pierwszego pada. W ten sposób możesz łatwo przysłać bezprzewodowe sygnały pomiędzy dwoma padami. Zwróć uwagę na części `else` w obu wyrażeniach warunkowych — jeśli przycisk nie jest wciśnięty, wibracje są wyłączone.

Ten sam mechanizm możesz wykorzystać także do praktycznych psikusów — możesz na przykład umieścić pad pod łóżkiem swojej ofiary, poczekać, aż zgasi światło i położyć się spać. Spróbuj następnie włączyć maksymalne wibracje w ukrytym padzie. Tylko nie miej do mnie pretensji, jeśli już nigdy nie odzyskasz swojego pada!

Przykładowy kod: Komunikaty w formie wibracji Przykładowy projekt w katalogu *04 Mind Reader* w ramach zasobów z kodem źródłowym dla tego rozdziału zawiera wersję programu do przekazywania komunikatów w formie wibracji. Pamiętaj tylko, żeby mądrze korzystać z tego programu. Program dodatkowo powoduje wyświetlenie czarnego ekranu, zatem jego prawidłowe działanie może nie być oczywiste.

Pomysł na grę: Gamepad Racer

Ostatni pomysł gry proponowany w tym rozdziale jest co prawda wyjątkowo niemądry, ale może dostarczać mnóstwo zabawy. Pierwszą rzeczą, którą musisz zrobić, jest znalezienie wielkiego, możliwie gładkiego stołu. Umieść kilka książek pod nogami na jednym końcu stołu, tak aby jego powierzchnia nie była pozioma. Jeśli umieścisz

pad konsoli Xbox na wyższym końcu tego stołu i włączysz jego wibracje, pad zsunie się po powierzchni stołu aż do drugiego, niższego końca. Możesz teraz poeksperymentować z kątem nachylenia stołu. Z moich testów wynika, że już po kilku próbach można tak dobrać ten kąt, aby podróż pada przez cały stół przy pełnej mocy silników wibrujących zajmowała około 30 sekund. Jeśli ułożysz cztery pady w linii blisko najwyższej krawędzi stołu, gracze będą mogli wybrać swoich „zawodników”, po czym przystąpić do wyścigu w dół stołu.

Kod tej gry jest wyjątkowo prosty; działanie metody Update sprowadza się do włączenia wszystkich silników wibrujących w padach:

```
protected override void Update(GameTime gameTime)
{
    // Umożliwia wyjście z gry.
    if(GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
    {
        GamePad.SetVibration(PlayerIndex.One, 0, 0);
        GamePad.SetVibration(PlayerIndex.Two, 0, 0);
        GamePad.SetVibration(PlayerIndex.Three, 0, 0);
        GamePad.SetVibration(PlayerIndex.Four, 0, 0);
        this.Exit();
    }

    GamePad.SetVibration(PlayerIndex.One, 1, 1);
    GamePad.SetVibration(PlayerIndex.Two, 1, 1);
    GamePad.SetVibration(PlayerIndex.Three, 1, 1);
    GamePad.SetVibration(PlayerIndex.Four, 1, 1);

    base.Update(gameTime);
}
```

Jedyną komplikacją jest konieczność wyłączenia wszystkich silników wibrujących w momencie zakończenia gry. Umieść teraz wszystkie pady na szczycie „toru wyścigowego” i uruchom ten program. Aby zatrzymać grę, naciśnij przycisk *Back* na pierwszym padzie.

Przykładowy kod: Gamepad Racer Przykładowy projekt w katalogu 05 *GamepadRacer* w ramach zasobów z kodem źródłowym dla tego rozdziału zawiera wersję programu wprawiającego w ruch ścigające się pady.



Uwaga Uważnie zmieniając wartości decydujące o szybkości działania silników wibrujących, możesz „sabotować” wybrane pady, tak aby zawsze wygrywali ci sami zawodnicy. Pamiętaj jednak, że nie pochwalam podobnych zachowań.

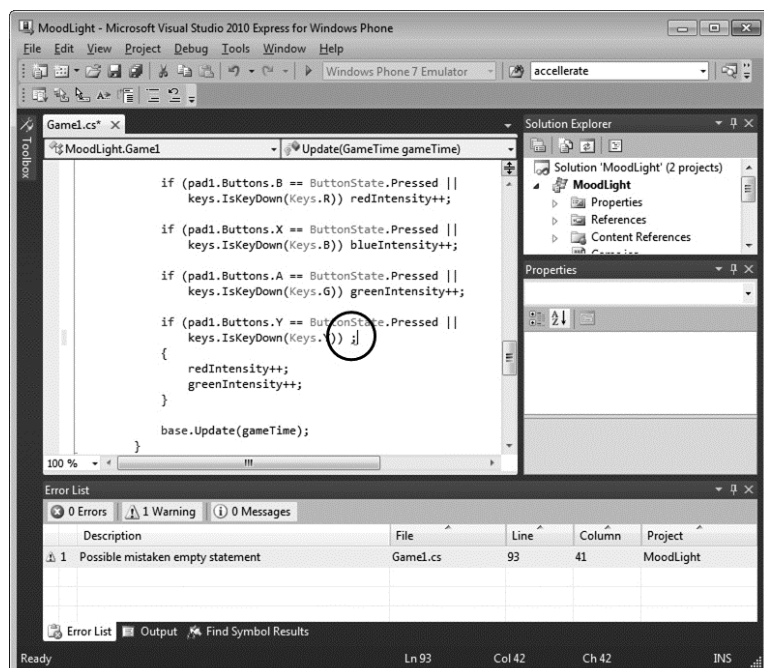
Błędy w programie

Twój młodszy brat wciąż próbuje nauczyć się programować, ale nadal napotyka problemy. Twierdzi, że ta książka zawiera błędy, ponieważ proponowane tutaj programy po zapisaniu na komputerze nie działają prawidłowo. Próbuje zmusić do działania grę *Color Nerve*, ale za każdym razem, gdy uruchamia ten program, intensywność żółtej barwy jest zwiększana, i to niezależnie od tego, czy naciska na padzie odpowiedni przycisk. Analizujesz jego program i znajdujesz następujący fragment kodu w metodzie Update:

```
if (pad1.Buttons.Y == ButtonState.Pressed ||
    keys.IsKeyDown(Keys.Y)) ;
{
    redIntensity++;
    greenIntensity++;
}
```

To jedyna część programu, w której jest zwiększana intensywność żółtej barwy, zatem wydaje się, że użyty warunek jest ignorowany.

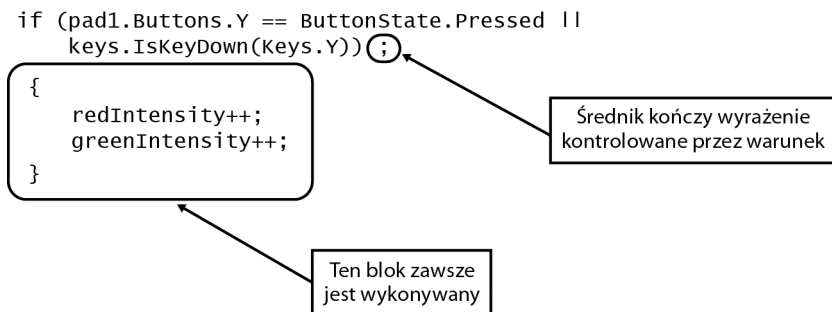
Kod sprawia wrażenie całkowicie prawidłowego, jest też poprawnie kompilowany i wykonywany, a mimo to intensywność żółtej barwy zdaje się nieustannie rosnąć. Warto w tym momencie przyrzeć się komunikatom wyświetlanym w środowisku Microsoft Visual Studio i sprawdzić, czy kompilator nie próbuje Ci czegoś zasygnalizować w związku z tym kodem. Na rysunku 3.6 pokazano kod Twojego brata po kompilacji.



RYСУNEK 3.6. Komunikat ostrzeżenia w środowisku Visual Studio

Twoją uwagę powinien zwrócić lewy dolny narożnik ekranu z komunikatem *Possible mistaken empty statement* (możliwy błąd pustego wyrażenia). Jeśli dwukrotnie klikniesz ten komunikat, kursor zostanie przeniesiony w miejsce bezpośrednio za wyrażeniem `if` (na rysunku 3.6 wyróżniłem to miejsce okręgiem).

Kompilator języka C# próbuje przekazać nam jakąś informację na temat tego wyrażenia. Jeśli wrócimy do oryginalnego listingu, odkrywamy, że Twój brat dopisał średnik bezpośrednio za warunkiem. Problem w tym, że średnik kończy wyrażenie kontrolowane przez ten warunek. Oznacza to, że jeśli zostanie wciśnięty czerwony przycisk na padzie, klawisz `R` na klawiaturze lub krzyżak, program nie wykona żadnej operacji (wykona wyrażenie puste), po czym wykona kolejne wyrażenia niezależnie od stanu pada czy klawiatury, doprowadzając do efektu, na który skarży się Twój młodszy brat. Działanie tego programu opisano na rysunku 3.7.



RYСУNEK 3.7. Skutek dopisania średnika

Usunięcie tego średnika spowoduje, że kompilator nie będzie już wyświetlał swojego ostrzeżenia, a program zacznie działać prawidłowo. Twój brat zaczyna powoli zmieniać swoją opinię na Twój temat; zaproponował nawet wyniesienie śmieci, mimo że dzisiaj przypada Twoja kolejka.

Wielka programistka mówi: pomaganie innym ludziom jest dobrym rozwiązaniem

Wielka programistka obserwowała Twoje postępowanie z rosnącą aprobatą. Powiedziała, że zawsze warto podejmować próby pomagania innym programistom, którzy napotykają na rozmaite problemy podczas pisania swoich programów. Zdarza się, że samo wyjaśnienie postronnemu obserwatorowi przez programistę negatywnych skutków ukrytego błędu w jakimś fragmencie kodu ułatwia temu pierwszemu szybką identyfikację źródła usterki. Oznacza to, że możesz dość łatwo, już przez samą obecność, zyskać sławę nieustraszonego pogromcy błędów. Co więcej, obserwowanie błędów popełnianych przez innych może być dla Ciebie cennym źródłem wskazówek, na co zwracać uwagę, gdy w Twoich programach coś nie działa tak, jak należy. A... i jeszcze może ominąć Cię kolejka wynoszenia śmieci.

Podsumowanie

W tym rozdziale nauczyłeś się całkiem sporo i wreszcie udało Ci się stworzyć gry, przy których gracze mogą miło spędzać czas. Dowiedziałeś się, jak framework XNA umożliwia programom interakcję z fizycznymi urządzeniami (za pośrednictwem metod wywoływanych dla klas), i opanowałeś sztukę podejmowania decyzji w programie zależnie od informacji uzyskiwanych z tych urządzeń. Swoją wiedzę wykorzystałeś do stworzenia kilku prostych (i niezbyt mądrych) gier

Przegląd rozdziału w pytaniach

Żaden rozdział nie byłby kompletny bez przeglądu. Oto przegląd dla tego rozdziału. Zapewne pamiętasz już zasady — zdecyduj tylko, czy poniższe zdania są prawdziwe, czy fałszywe. W dodatku A na końcu tej książki znajdziesz odpowiedzi, które pozwolą Ci rozstrzygnąć, czy po lekturze tego rozdziału jesteś zwycięzcą, czy przegranym.

1. Jeśli klasa jest biurem, metoda jest biurkiem w tym biurze.
2. Kompilator tworzy wszystkie egzemplarze klas w programie.
3. Wyrażenie `if` musi zawierać część `else`.
4. Parametr służy do przekazywania informacji do klasy.
5. Część `else` wyrażenia `if` zawsze jest wykonywana.
6. W programie na bazie frameworku XNA informacje o stanie pada są reprezentowane przez wartość typu `byte`.
7. Metoda `GamePad.GetState` może być używana do sprawdzania, czy naciśnięto jakiś przycisk na padzie (to trudne pytanie; przed udzieleniem odpowiedzi możesz zajrzeć do rozdziału).
8. Blok składa się z pewnej liczby wyrażeń języka C# otoczonych nawiasami klamrowymi.
9. W języku C# warunek `(true || false)` oznacza „prawda lub fałsz” i zawsze ma wartość `true`.
10. W języku C# warunek `(redIntensity > 220)` ma wartość `true`, jeśli wartość zmiennej `greenIntensity` jest większa niż 220.
11. Wibracje pada zawsze są automatycznie wyłączone w momencie zatrzymania wykonywania gry przez system XNA.

Część II

Obrazy, dźwięk i tekst

W tej części:

Rozdział 4. Wyświetlanie obrazów	89
Rozdział 5. Wyświetlanie tekstu	111
Rozdział 6. Tworzenie gry dla wielu graczy	133
Rozdział 7. Odtwarzanie dźwięków	145
Rozdział 8. Pomiar czasu	159
Rozdział 9. Wczytywanie tekstu wejściowego	173

Skorowidz

A

- abstrakcja, 325, 330
- adres rozgłaszania, 359
- adresowanie komunikatów, 358
- akcelerometr, 387
 - interpretowanie odczytów, 390
 - uruchamianie, 397
 - wykorzystanie odczytów, 398
 - wykrywanie potrząsania, 403
- aktualizacja metody Update, 321
- aktualizacja sprajtu papryki, 321
- aktywa, 92
- aktywność połączenia sieciowego, 375
- algebra Boola, 57
- algorytm, 57
- alokacja pamięci, 179
- Alt+Backspace, 281
- alternatywa arytmetyczna, 430
- analiza wartości zmiennej, 213
- aplikacja
 - XNA Game Studio Connect, 24
 - XNA Game Studio Device Center, 25
- automatyczne logowanie, 367

B

- backgroundColor, 43
- bajt, 51
- bezpieczeństwo typów, type-safe, 374
- biblioteka
 - DirectX 10, 23
 - dźwięków, 145
 - gier, 30
 - Microsoft.Devices.Sensors, 393
- bit, 55
- blok, 41, 73
- blokada, 404
- blokowanie ekranu, 433
- błąd, error, 59, 64
- błąd kompilacji, 32, 216, 242
- błąd w pomiarze czasu reakcji, 162
- błędy dźwiękowe, 154
- błędy kumulujące się, cumulative errors, 224
- błędy w programie, 63, 83
- brak pamięci, 127
- brak referencji do egzemplarza, 310
- Break, 187

C

- cechy list, 342
- chleb, 251
- ciało, body, 204
- ciało metody, 204
- ciągłe odtwarzanie dźwięku, 151
- cykl życiowy informacji, 410
- czas blokowania, 433
- czas życia baterii, 428
- czcionka, 112
- czcionka Kootenay, 113
- czcionka SpriteFont, 113
- czcionka True Type, 191
- czcionki bitmapowe, 191
- częstotliwość aktualizacji, 63
- częstotliwość odświeżania, 428
- częstotliwość próbkowania, 146

D

- dane, 49, 51
- dane prywatne, 296
- dane w grze ButtonBash, 134
- data i czas, 119
- datagram, 360, 361
- debuger, 214
- deklaracja zmiennej, 43
- deklarowanie prywatnych danych, 296
- deklarowanie zmiennej licznika, 166
- dekodowanie znaków na klawiszach, 188
- delegacja, 373, 395
- diagnozowanie programów, 210, 262
 - przeglądanie zmiennych, 212
 - punkt zatrzymania, 211
 - wznawianie działania, 213
- diagram przepływu danych, 160
- diagram stanów, 288, 289, 435
- długość wektora, 401
- długość wektora prędkości, 401
- długość wektora przyspieszenia, 403
- dodanie delegacji, 396
- dodanie
 - dźwięków, 347
 - ekranu tytułowego, 286
 - elementu ryzyka, 273
 - metody obsługi zdarzenia, 396

512 Skorowidz

dodanie
 nowego typu stanu, 437
 postępu, 275
 punktacji, 271
 punktu zatrzymania, 262
 referencji do biblioteki, 394
 sztucznej inteligencji, 342
 tekstury, 285
 tła, 285

dodawanie
 czcionki do zasobów, 112
 dźwięku, 145
 elementów do folderu Content, 148
 elementów do listy, 340
 łącz do zasobów, 96
 nowego elementu, 113
 obrazów, 90
 tarcia, 401
 wibracji, 77
 zasobu do projektu, 114
 zawartości do projektu, 94
 zdjęcia za pomocą łącza, 97

doskonalenie programów, 256
doskonalenie projektu, 278
dostęp do elementów listy, 340
dostęp do klasy Accelerometer, 392
double, 242
druga zasada dynamiki Newtona, 388
dwa znaki równości, 44
dyrektywa, 142
 #define, 142
 #endif, 142
 #if, 142
 using, 234
dziedziczenie, 316
dzielenie zwracające wynik
 całkowitoliczbowy, 218
 zmiennoprzecinkowy, 218
dźwięk, 402
dźwiękowa poziomica, 402

E

edycja tekstu, 192
efekt oddalania, 200
efekty dźwiękowe, 348
egzemplarz, instance, 69
 delegacji, 373
 klasy, 69
 Game1, 237
 gry, 238
 HugeObjectUsedForSums, 237
 KeyboardState, 75
sterujący, 155

ekran dotykowy, 407
ekran Loading, 52
ekran lobby, 375
ekran Xboksa, 90
element obrazu, 102
element tablicy, 165
element typu TouchLocation, 408
elementy zawartości, 92
else, 59
emulatory, 424
etykieta bagażowa, 308

F

falsz, false, 55
flaga noTomatoes, 276
float, 242
folder, 229
folder Images, 231
for, 124
format
 BMP, Windows Bitmap, 91
 daty i czasu, 121
 JPEG, Joint Photographic Experts Group, 91
 XML, 115
 PNG, Portable Network Graphics, 91
formaty bezstratne, 91
formaty stratne, 91
framework .NET, 233
framework XNA, 21, 23, 48, 234, 363
funkcja IntelliSense, 396

G

gałka analogowa, 255
generator liczb pseudolosowych, 336
generator statyczny liczb, 337
generator zdarzeń, 375
generowanie efektów dźwiękowych, 348
gesty, 425
globalna zmiana nazwy klasy, 239
GPU, 103
gra ButtonBash
 dane, 134
 dodawanie kodu testowego, 140
 konstruowanie gry, 138
 licznik wciśnień, 135
 projektowanie kodu, 140
 rozpoczynanie gry, 134
 zliczanie wciśnień, 135
 zmiany położenia przycisku, 136

gra Cheese Lander, 391
 dodawanie tarcia, 401
 klasa Accelerometer, 395
 obiekty, 391
 poprawianie przebiegu, 398
 sterowanie dźwiękiem, 402
 wektor ruchu, 400

gra Chleb i ser, 227

gra Color Nerve, 74

gra Gamepad Racer, 81

gra jako aplikacja, 442

gra planszowa, 418
 odświeżanie stanu krążka, 420
 przemieszczanie krążka, 421
 stany krążka, 419
 sterowanie krążkiem, 422

gra Pomiar czasu reakcji, 159

gra ShuffleBoard, 429

gra Superoddalenie, 200

gra Tajne komunikaty, 80

gra typu SystemLink, 376

gra w tenisa Bread vs. Cheese, 429

gra Wielokolorowa lampa, 61

gra Zawody we wciskaniu klawiszy, 133

gracz, 364

gry sieciowe, 369

H

hermetyzacja, encapsulation, 297

hierarchia klas, 317

hierarchia klas GameSprite, 313

host, 378

I

IDE, Integrated Development Environmen, 22

identyfikacja stanów, 291

identyfikator, 43, 177

identyfikator Windows Live ID, 25

if, 58

ilość danych, 312

implementowanie
 poprawki pomiaru czasu, 164
 interfejsu, 353
 klasy abstrakcyjnej, 327

instrukcja
 if, 189
 ładująca zawartość, 99
 przypisania, 181
 switch, 188

Intellisense, 106, 283

interakcja pomiędzy obiektami, 301

interakcja pomiędzy sprajdami, 259

interfejs, 351, 353, 354
 ISprite, 354
 ISpriteBasedGame, 354
 programowy, 351
 programu Audacity, 146

Internet, 362

IP, Internet Protocol, 362

ISP, Internet service provider, 359

izolowany obszar zapisu, 449

J

jednostka przetwarzania grafiki, GPU, 103

język C#, 21

K

kanal alfa, 129

karta gracza, gamertag, 362

catalog
 Content, 157
 Sounds, 149
 z rozwiązaniem JakeDisplay, 92

klasa, 46, 69, 312
 AbstractSprite, 326
 abstrakcyjna, 326
 implementowanie klasy, 327
 projektowanie systemów, 328
 rozszerzanie klasy, 327
 tworzenie klasy, 326
 tworzenie referencji, 329
 wypełnienie klasy, 327

Accelerometer, 395

BaseSprite, 314
 obsługa tła, 315
 rozszerzenie klasy, 316

CheeseBall, 349

DeadlySprite, 318

Game1, 46, 48

Game1 jako plan biura, 69

Game1 z metodą LoadContent, 98

GamePad, 70, 395

GamePad jako biuro, 71

GamePadState, 68

GraphicsDevice, 49

gry, 104

Guide, 368

Keyboard, 75

KillerSprite, 335

kolekcji, 339

514 Skorowidz

klasa

- List, 339
- macierzysta (bazowa), 313
- MediaPlayer, 156
- MovingSprite, 335
- potomna, 313
- potomna MovingSprite, 317
- PuckSprite, 419
- Random, 337
- Rectangle, 200
- SoundEffect, 158
- SoundEffectInstance, 151
- SoundPad, 412
- sprajtu, 316
- SpriteBatch, 103
- Texture2D, 235
- TitleSprite, 316
- TouchPanel, 409

klasy i struktury, 306

klasy zarządzane przez referencje, 306

klawiatura, 74

klawisz Backspace, 75, 192

klawisz Escape, 75

klawisz Shift, 189

klucz programowy, 454

kod

- 3-D Big Clock, 125
- 3-D Shadow Clock, 130
- Big Clock, 121
- Broken Button Bash, 135
- Broken Reaction Timer Game, 162
- Button Bash, 140
- Button-Bash Test, 142
- Cheese Lander Game, 398
- Color Nerve, 77
- Drum Pad, 150
- First Message Display, 186
- Fixed ReactionTimer, 164
- Gamepad Racer, 82
- gra w chleb i ser, 277
- Image MoodLight, 108
- Jake Color Nerve, 108
- Jake Display, 104
- Jake Full Screen, 107
- klasy Game1, 238
- Komunikaty w formie wibracji, 81
- konstruktora, 331
- kopiujący odczyty, 397
- maszynowy, 142
- MediaPlayer, 157
- nieprawidłowe oddalenie środka zdjęcia, 223
- nieprawidłowe przybliżenie, 201
- obsługi klawiatury wyzwany zboczem, 174

Odbijanie sera, 246

Odbijanie sera z uwzględnieniem nadmiarowości, 248

programu, 51

Przybliżanie Jake'a, 220

RayGun, 154

ReactionTimer with Winner Display, 171

Shake Tester, 403

sieciowa gra ping-pong, 382

struktura sprajtu pałki, 298

testowy, 140

Working Button Bash, 138

Zabójca pomidorów, 269

zewnątrzny, 298

źródłowy programu, 32

Żółte światło ostrzegawcze, 45

kodowanie znaków, 191

kolekcja, 339

kolekcja dotykanych punktów, 409

kolekcja SignedInGamers, 368

kolekcja typu List, 342

kolor, 43

kolory przezroczyste, 128

komentarz, 42

kompilacja plików z kodem źródłowym, 32

kompilacja warunkowa, 141

kompilator, 52, 70, 141, 216, 234, 316, 329

kompilator języka C#, 52, 125

komponent, 351

komunikat, 360

komunikat o błędzie, 217, 234

komunikat o zmianie orientacji, 430

komunikat ostrzeżenia, 83

komunikat Possible mistaken empty statement, 84

komunikaty i organizacja, 302

konfiguracja

- komputera PC, 23

- konsoli Xbox 360, 24

- systemu Windows Phone, 26

konsola Xbox 360, 22, 363

konstrukcja pętli for, 125

konstruktor, 331

- dla typu wartościowego, 332

klasy

- BaseSprite, 332

- bazowej, 334

- KillerSprite, 336

- macierzystej, 334

- MovingSprite, 336

- potomnej, 334

- TitleSprite, 334

piłki, 354

w hierarchii klas, 333

- w klasach potomnych, 334
- w strukturach, 332
- konstruowanie gry, 138
- kopiowanie efektów dźwiękowych, 349
- kopiowanie projektu, 34
- korzystanie z zasobów, 97
- krążek, 419
- kropka dziesiętna, 242
- krój znaków, 112

L

- liczba kanałów audio, 151
- liczba rzeczywista, 214
- liczba zmiennoprzecinkowa, 214
- liczby losowe, 336
- liczby pseudolosowe, 336
- licznik czasu, 159
- licznik wciśnięć przycisku, 135
- lista dynamiczna, 413
- lista nazw plików, 93
- lista referencji dla gry, 393

Ł

- ładowanie czcionki, 115
- ładowanie tekstur, 97
- ładowanie zapisanej gry, 450
- łańcuch wiadomości, 182
- łańcuchy tekstowe, 122
- łączenie konsoli z komputerem, 25
- łączenie strumieni danych, 448
- łączenie urządzenia Windows Phone z komputerem, 27

M

- magistrala USB, 68
- mandarynka, 335
- Marketplace, 453
- maszyna stanów, 289, 409, 434–435
- maszyna stanów dla telefonu, 434
- mechanizm odbijania, 260
- mechanizm refraktoryzacji, 279
- mechanizm właściwości, 343
- mechanizm zarządzania treścią, 29
- menedżer treści, Content Manager, 29
- metoda
 - accel_ReadingChanged, 397
 - CheckCollision, 300
 - Clear, 42, 46
 - Close, 450

- Contains, 411
- CreateInstance, 152
- DoAdd, 373
- Draw, 46, 52, 98, 104, 273, 304
- drawText, 273
- DrawText, 116
- Exit, 75, 79
- Find, 376
- gameOver, 290
- getPercentage, 204, 247
- GetPressedKeys, 175, 177, 182
- GetState, 70, 409
- hostSession_GamerJoined, 374
- Initialize, 102, 238, 241, 308
- Intersects, 259
- IsKeyDown, 75, 174
- Length, 401
- Load, 99
- LoadContent, 98, 155, 222, 238, 409, 414
- LoadTexture, 316
- Loselife, 351
- ładująca, 450
- Main, 236
- Math.Abs, 400
- Parse, 451
- Play, 156
- playerSession_GamerLeft, 376
- positionButton, 431
- Remove, 342
- resetTomatoDisplay, 276
- Run, 237
- saveGame, 449
- scaleSprites, 243, 256
- SendData, 379
 - Chat, 379
 - InOrder, 379
 - None, 379
 - Reliable, 379
 - ReliableInOrder, 379
- setupScreen, 248
- SetVibration, 77
- StartGame, 319
- titleUpdate, 290
- ToString, 182
- UnloadContent, 103
- Update, 46, 53, 75, 271
- Update główna, 416
- Update klasy KillerTangerine, 346
- Update w klasie soundPad, 416
- updateBackground, 285
- updatePlayingGame, 438
- ustawiająca parametry papryki, 319
- wirtualna, 316

516 Skorowidz

metoda

- Write, 378
 - WriteLine, 450
 - zajmująca się wyświetlaniem, 129
 - zwracająca sterowanie, return, 206
- metody, 41, 46, 177
- ciało, 205
 - nagłówek, 204
 - typ i liczba parametrów, 205
 - wywoływanie, 205
- metody klasy BreadAndCheeseGame, 302
- metody kolekcji celów, 303
- metody nie zwracające wyników, void, 206
- metody publiczne, 296
- metody statyczne, 236
- metody struktury BatSpriteStruct, 301
- metody struktury TargetRowStruct, 304
- metody typu DateTime, 120
- metody uogólnione, 99
- miękkie lądowanie, 399
- modyfikator ref, 258
- modyfikatory, 253
- moment wciśnięcia klawisza, 175

N

- nadmiarowość ekranu, overscan, 246
- nagłówek, header, 204
- nagłówek i ciało metody getPercentage, 205
- nagłówek metody, 204
- narzędzie Microsoft Cross-Platform Audio Creation Tool (XACT), 145
- narzędzie XACT audio tool, 156
- nawias klamrowy, 41, 73
- nawias kwadratowy, 166
- nazwa klasy SpriteBatch, 104
- nazwa zasobu, 100
- nazwa zmiennej spriteBatch, 104
- niebieski ekran, 36
- niebieskie linie, 216
- Now, 120

O

obiekt, 294

- BatSpriteStruct, 296
- ekranu tytułowego, 305
- gry Cheese Lander, 391
- jakeRect, 198
- klasy Rectangle, 102, 307
- klasy SpriteBatch, 99
- localhost, 380

- pałki, 302
- piłki, 300
- SpriteBatch, 103
- strumienia danych, 447
- tła, 305
- typu Rectangle, 102
- typu SoundEffect, 150
- typu StreamReader, 451
- typu StreamWriter, 450
- uchwyty, 152

obiekty

- kontenerów, 303
 - występujące w grze, 228
 - zarządzane przez referencje, 311
 - zarządzane przez wartości, 311
- obraz wysokiej rozdzielczości, 90
- obsługa
- klawiatury, 174
 - kolizji, 259
 - miejsca dotknięcia, 411
 - nadmiarowości ekranu, 246
 - przychodzących połączeń, 439–440
 - przycisków na telefonie, 442
- obszar roboczy, workspace, 30
- ochrona danych, 296
- ochrona danych w hierarchii klas, 317
- oddalanie ze środka zdjęcia, 220
- odświeżanie stanu krążka, 420
- odświeżanie stanu przycisków, 415
- odtwarzanie dźwięków, 149
- odtwarzanie efektu dźwiękowego, 152
- odtwarzanie muzyki w tle, 151
- odzyskiwanie pamięci, garbage collector, 179
- okno
- Add Existing Item — Content, 95, 147
 - Add Reference, 393
 - błędów, 58
 - dialogowe błędów kompilacji, 215
 - Rename, 280
- opcja
- Create Copy Of Project For Xbox 360, 34
 - Create New Profile, 366
 - Error List, 58
 - Extract Method, 279
 - New Project, 29
 - Refactor, 279
 - Set as StartUp Project, 35
- operand, 54, 210
- operator, 54
- operator ||, Patrz operator logiczny lub
- operator /, 218
- operator ++, 54

operator +=, 374
 operator alternatywy || (lub), *Patrz* operator logiczny lub
 operator internetowy, 359
 operator koniunkcji && (i), 137, 209
 operator logiczny lub, or, ||, 76, 137, 209
 operator new, 102
 opornościowy ekran dotykowy, 407
 organizacja obiektów, 299
 ostrzeżenie, warning, 59
 oświetlenie obiektów, 228

P

P2P, peer to peer, 377
 pady, 68
 pakiet XNA SDK, 22
 pakiety, 358
 pałka, 343
 pamięć, 51
 panel Solution Explorer, 34, 40, 93
 papryka, 318
 parametr, 204

- identyfikujący klawisz, 75
- identyfikujący pad, 70
- metody Main, 236
- przekazywany przez referencję, 258
- przekazywy przez wartość, 257
- sender, 380
- sprajtu, 256, 341

 pasek stanu, 432
 pasek sterowania programem, 213
 pełna nazwa, 234
 perkusja, 412

- przyciski, 414-416
- wartości soundPad, 413

 pętla

- do-while, 126
- for, 124
 - przyrost, 125
 - wartość początkowa, 124
 - warunek kontynuacji, 124
- foreach, 341
- while, 126

 piksel, 90, 101
 ping-pong, 364
 plik

- BreadAndCheeseGame.cs, 239
- cymbal.wav, 412
- Game.ico, 93
- Game1.cs, 40, 142, 238
- GameStatus.txt, 449
- GameThumbnail.png, 93

JakeDisplay, 92
 Program.cs, 232, 239

- przestrzeń nazw, 233
- słowo using, 237

 pliki

- .mp3, 147
- .sln, 36
- .wav, 146
- .wma, 147
- czcionki, 112
- klas, 69
- programu, 232
- projektu JakeDisplay, 93
- rozwiązania, 93
- XML, Extensible Markup Language, 114
- źródłowe, 239

 pobieranie obrazu z zasobu, 97
 pobieranie wejścia dotykowego, 408
 podpowiedzi Intellisense, 106, 283
 podświetlanie przycisków, 416
 pojemnościowy ekran dotykowy, 407
 pole, 102, 177

- Create Directory For Solution, 31
- danych, 311
- Preview Reference Changes, 280
- Visible, 268

 polecenie

- Edit (edycja), 281
- New Project, 92
- Rename, 238
- Undo (cofnij), 281

 polskie znaki diakrytyczne, 190
 połączenie, 360, 361
 połączenie bezprzewodowe, 68
 położenie pałki, 343
 położenie prostokąta, 101
 położenie sprajtów, 336
 pomiar czasu, 159
 pomiar siły, 389
 pomidor, 264
 pomoc Intellisense, 320
 porównanie (==), 61
 potok, 97
 potok zarządzania zawartością, 97
 potok zawartości w XNA, 97
 powtarzanie instrukcji, 124
 pozycjonowanie sprajtu na ekranie, 101
 pozycjonowanie wyświetlania liczników, 141
 prawda, true, 55
 prawe ukośniki (//), 42
 preprocesor, 142
 preprocesor WINDOWS_PHONE, 245
 prędkość, 398

518 Skorowidz

- procedura obsługi zdarzenia połączenia przychodzącego, 440
- proces, 405
- proces rejestracji programów, 454
- profil gracza, 364
 - automatyczne logowanie, 367
 - nazwa, 366
 - przełączanie, 367
 - sprawdzanie logowania, 368
 - tworzenie profilu, 365
 - wylogowanie, 367
 - zapisywanie, 367
- program
 - Audacity, 146
 - emulatora, 23
 - Microsoft Paint, 91
 - Mood Light, 73
 - MSDN Academic Alliance, 24
 - Paint.NET, 91, 228
 - wielokolorowej lampy, 56
 - XNA Game Studio Connect, 25
 - Zune, 37
- programowanie obiektowe, 293
- projekt
 - BigClockContent, 113
 - BreadAndCheese, 229
 - ButtonBash, 133
 - Color Nerve with Vibes, 80
 - dla kodu programu, 94
 - dla zawartości, 94
 - Drum Pad, 145
 - Giant Clock, 111
 - gry, 94
 - JakeDisplay, 93
 - JakeDisplayContent, 93
 - kontroler wielokolorowej lampy, 68
 - MoodLight, 31, 34
 - Picture Display, 89
 - Tablica ogłoszeń, 173
 - wielokolorowa lampa, 39
 - zawartości, 94
- projektowanie maszyn stanów, 439
- projektowanie testów, 208
- proporcja, aspect ratio, 242
- protokół, 361
- protokół IP, 362
- protokół TCP, 362
- protokół TCP/IP, 362
- przechowywanie dźwięków, 147
- przechowywanie gier w konsoli Xbox 360, 34
- przechowywanie rozkazów programu, 51
- przechowywanie tła, 315
- przechowywanie wartości w pamięci, 51
- przechylenie telefonu, 390
- przeglądanie tablicy, 167
- przeglądanie zmiennych, 212
- przeglądy kodu, 331
- przejścia pomiędzy stanami, 288
- przekształcanie sprajtów w komponenty, 351
- przemieszczanie krążka, 421
- przepełnienie, overflow, 55
- przepełnienie pamięci, 55
- przerwanie wykonywania pętli, 187
- przestrzeń nazw, 233, 234
- przestrzeń nazw Microsoft.XNA.Graphics, 235
- przestrzeń robocza, 229
- przesuwanie obiektów w zwarciu, 261
- przesyłanie strumieniowe, 361
- przezroczystość, 129, 417
- przybliżanie obrazu, 198
- przycisk
 - Back, 33, 75, 443, 445
 - Start, 443, 445
 - Start Debugging, 31, 35
 - Stop, 33
- przykrywanie metod klasy macierzystej, 316
- przypisanie (=), 61
- przypisanie wartości typu Color do zmiennej, 44
- przyspieszenie, 388, 399
- przyspieszeniomierz, 387
- punkt dostępowy WiFi, 363
- punkt zatrzymania, 211

R

- refaktoryzacja, 240
- refaktoryzacja kodu, 279
- refaktoryzacja poprzez tworzenie metod, 279
- refaktoryzacja poprzez zmianę identyfikatorów, 280
- referencja, 93, 154, 177, 306, 309
 - do czcionki, 113, 114
 - do interfejsów, 353
 - do klasy macierzystej, 329
 - do metody, 373, 395
 - do obiektu, 179, 395
 - do obiektu gry, 305
 - do samego siebie, 301
 - do tablicy, 178, 187
 - do zasobów, 95
 - i instancja tablicy, 165
 - shootSoundEffectInstance, 154
 - typu AbstractSprite, 329
- referencje do jednego egzemplarza, 309
- reguły obsługiwanie przycisków, 443
- rejestracja urządzenia, 28, 37, 454

- rejestracja w serwisie App Hub, 454
- rejestrowanie wciśnięć klawiszy, 174
- rodzaje składowych, 50
- rollover, 174
- rozdzielczość ekranu, 432
- rozdzielczość obrazu, 90
- rozdzielczość próbki, 146
- rozmiar ekranu, 432
- rozmiar obrazu, 267
- rozmiar sprajtów, 240
- rozmieszczanie obrazów na ekranie, 199
- rozszerzanie klasy abstrakcyjnej, 327
- rozszerzenie klasy BaseSprite, 316
- rozwiązanie, solution, 92, 229
 - projekt obejmujący kod gry, 229
 - treści wykorzystywane przez grę, 229
- rozwiązanie BreadAndCheese, 229
- rozwiązanie JakeDisplay, 93
- ruch, 400
- rysowanie obrazu, 242
- rzutowanie, 217

S

- samodzielne wyłączenie wibracji, 80
- SDK, Software Development Kit, 22
- sekwencje liczb, 338
- serwer, 377
- serwis App Hub, 24
 - Registered, 25
 - Trial, 24
 - Visitor, 24
- serwis Windows Live, 454
- sieci komputerowe, 357
- sieć lokalna, 361, 363
- sieć Xbox Live, 362
- siła grawitacji, 388
- skalowanie obrazu, 90
- składowe klasy, 48
- składowe klasy Game1, 50
- składowe koloru, 108, 129
- składowe prywatne, 317
- słowo kluczowe
 - base, 320
 - break, 187
 - lock, 404
 - new, 308
 - override, 206
 - private, 318
 - protected, 206, 318
 - public, 318
 - return, 206
 - this, 301

- using, 237
- value, 344
- SoundPad
 - efekt dźwiękowy, 412
 - prostokąt, 412
 - tekstura, 412
- społeczność App Hub, 453
- sposób wiązania obiektów, 299
- spójność, cohesion, 295
- sprajt, sprite — duszek, 101, 240, 317
 - rozmiar, 240
 - scaleSprites, 245
 - wielkość, 243
 - wprawianie w ruch, 244
- sprajt chleba, 252
- sprajt chleba, pomidorów i sera, 267
- sprajt pałki, 296
- sprajt papryki, 319
 - aktualizacja, 321
 - rysowanie sprajtu, 319
 - ustawianie parametrów, 319
- sprajt sera, 243, 247
- sprzedaż gry, 453
- sprzężenie, 298
- stan
 - ekranu tytułowego, 287
 - gry, 47, 287, 369, 434
 - gry sieciowej, 369
 - ekran tytułowy, 371
 - gra w roli hosta, 375
 - logowanie graczy, 371
 - oczekiwanie na hosta, 376
 - rozgrywka, 377
 - wybór roli gospodarza, 371
 - wybór roli gracza, 376
 - Koniec gry, 439
 - krążka, 419
 - pada, 68
 - PlayingAsHost, 377
 - PlayingAsPlayer, 377
 - playingGame, 287
 - początkowy, 288
 - przycisku na osi czasu, 136
 - WaitingAsPlayer, 376
- standard języka XML, 115
- standard UTF-8, 191
- standardowe biblioteki XNA, 392
- status pada, 72
- sterowanie
 - przechylaniem, 391
 - ruchem, 254
 - silnikiem, 79
 - wibracjami pada, 77

520 Skorowidz

sterowanie
 widocznością sprajtów, 267
 zachowaniem metody Update, 420

stos, 179

stosowanie dźwięków, 149

stosowanie hierarchii klas, 314

stosowanie interfejsów, 355

struktura, 43, 252, 312
 BatSpriteStruct, 295
 DateTime, 119
 typu GameSpriteStruct, 257
 zarządzana przez wartości, 306

strumień danych, 447

strumień danych rawStream, 450

strumień danych surowych, 448

strumień tekstowy, 448

switch, 188

synchronizacja procesów, 405

system Content Management, 448

system hosta, 377

System Link, 363

system Windows Phone, 23, 427

szkielet metody Update, 53

sztuczna inteligencja
 położenie pałki, 343
 pościg sprajtów, 344
 ściganie pałki, 343
 trafianie mandarynek, 346

szybkość reakcji ekranu dotykowego, 424

szybkość ruchu, 245

Ś

średnik, 84

środek zdjęcia, 220

środowisko
 Visual Studio 2010, 28
 IDE, 22
 Microsoft .NET Framework, 120
 Microsoft Visual Studio 2010 Express Edition
 for Windows Phone, 22
 Visual Studio, 23
 XNA Game Studio, 29, 31

T

tabela najlepszych wyników, 93

tablica
 AbstractSprite, 329
 BaseSprite, 330
 Keys jako biuro, 177, 178
 ogłoszeń, 193

 słownikowa, 169
 targetVisibility, 304
 tomatoes, 265

tablice, 164
 dane, 165
 element, 165
 granice, 167
 indeks, 165
 jako słownik, 168
 jednowymiarowe, 165
 przeglądanie, 167
 rozmiar, 170
 wpisywanie łańcuchów, 169
 zmienna tablicowa, 165

tag Xbox Live Gamer Tag, 25

takt, tick, 245, 428

takt zegara, 428

tarcie, 401

TCP, Transport Control Protocol, 362

Teal, 43

technika refaktoryzacji, 256

technologia System Link, 363

tekst 3D, 122

tekst jako zasób, 112

tekstura, 97, 265

tekstura obrusa, 306

tekstura tła, 285

testowanie
 gry, 140
 metody, 207
 programów, 208, 425
 przycisku na padzie, 72
 wartości, 60
 wartości intensywności, 78

topologia gry, 377

trasowanie, 359

tryb Guide, 382

tryb nadmiarowości, overscan, 107

tryb przyciągania uwagi, 290

tryb wrażliwy na poziom, 174

tryb wyzwalany zbczem, 150

tworzenie
 ciała metody getPercentage, 209
 cieni, 128
 dotykowej perkusji, 412
 efektu oddalania, 200
 egzemplarza delegacji, 373
 egzemplarza klasy, 307
 egzemplarza klasy abstrakcyjnej, 326
 gier na sprzedaż, 454
 gier sieciowych, 357, 363
 grafiki gry, 228
 gry, 452

gry Cheese Lander, 391
 gry dla wielu graczy, 133
 gry z wciskaniem przycisków, 133
 hierarchii klas sprajtów, 314
 iluzji trójwymiaru, 128
 interfejsu, 352
 klasy abstrakcyjnej, 326
 klasy DeadlySprite, 318
 klasy KillerSprite, 335
 klasy SoundPad, 412
 kolekcji typu List, 339
 komentarzy, 283
 kompletnej gry, 271
 komponentów gry, 325
 kopii projektu XNA, 34
 lobby gry, 369
 maszyny stanów, 434, 435
 metod dla istniejącego kodu, 279
 metod statycznych, 236
 metody, 280
 metody getPercentage, 207
 metody obliczającej wartości procentowe, 203
 mobilnych gier, 385
 nowego folderu treści, 231
 nowego projektu, 30, 92
 nowych koncepcji gier, 74
 obiektów gry, 240
 obiektu, 102
 obiektu klasy gry, 233
 obszarów kodu, 282
 pierwszego projektu, 28
 profilu, 365
 programu gry, 40
 projektu Drum Pad, 145
 projektu Tablica ogłoszeń, 173
 projektu zegara, 112
 przycisków dźwiękowych, 413
 przycisku alarmowego, 408
 sesji gry sieciowej, 372
 spójnej struktury, 295
 statycznego generatora liczb, 337
 struktury, 306
 struktury katalogów, 92
 ślizganej gry planszowej, 418
 tablicy, 165
 tablicy ogłoszeń, 182
 uchwytu do efektu dźwiękowego, 155
 wersji demonstracyjnych, 143
 wyrażeń warunkowych, 58
 zachowań klienta, 381
 zachowań serwera, 377
 zegara, 122
 zmiennej typu SpriteBatch, 103
 związku pomiędzy pałką a piłką, 300

typ

bool, 57
 bute, 51
 byte, 55
 Color, 43, 51
 DateTime, 119
 double, 177
 GameSpriteStruct, 295
 int, 180, 214
 Keys, 175
 Rectangle, 101, 102
 string, 177
 Texture2D, 98
 TimeSpan, 428

typy

danych, 215
 obiektów TouchLocation, 409
 projektów, 30
 przechowujące wartość, 177
 referencyjne, 177, 311
 tablicowe, 178
 uogólnione, 339
 wartościowe, 311, 312
 wyliczeniowe, 176, 287, 370
 wyrażeń, 218
 zmiennych zarządzane przez referencje, 154
 zmiennych, 43

U

ukrywanie paska stanu, 432
 umieszczanie zawartości w grze, 92
 uruchamianie gry, 99
 uruchamianie programu, 31
 urządzenia sieciowe, 359
 urządzenie Zune, 27
 usługa

- App Hub, 362
- Marketplace, 27
- Silver Xbox Live, 24
- Windows Phone Marketplace, 428, 453
- Xbox Live, 25, 40, 362
- Xbox Live Games, 34
- Xbox Live Indie Games, 25

 ustalanie zwycięzcy, 164
 ustawianie punktu zatrzymania, 211
 ustawienie właściwego położenia, 223
 usuwanie zapisanych plików, 452
 utrata danych, 242
 używanie klawiszy Shift, 189
 używanie pada i klawiatury, 75
 używanie referencji i wartości, 180

V

Visual Studio 2010, 393
void, 206

W

wartości, 306
całkowitoliczbowe, 241
typu soundPad, 413
współdzielone, 48
zmiennoprzecinkowe, 241
zmiennoprzecinkowe dwukrotnej precyzji, 242

wartość
ButtonState.Pressed, 75
graniczna, 423
graphics, 430
indeksu, 166
koloru, 47
null, 155
null w referencjach, 155, 308
prędkości, 424
TouchLocation, 422
typu double, 242
wyrażenia sterującego, 189

warunek if, 58
warunki dozoru, 288
wbudowany zbiór kolorów, 43
wczytywanie tekstu, 173, 193
wejście dotykowe
odczytywanie zdarzeń, 409
pobieranie wejścia, 408

wektor, 116, 400, 402
widoczność obiektów, 268
widok Solution Explorer, 393
wielkość sprajtu, 243
wielokrotnie wyświetlony tekst, 123
Windows Phone Marketplace, 428
właściwości, 120, 177
czcionki, 118
dźwięku, 153
IsLooped, 153
Pan, 153
Pitch, 153
zasobu, 99

właściwość
IsDataAvailable, 380
IsLooped, 153
IsVisible, 368
Length, 178
Now, 120
Pan, 153
Pitch, 153

State, 157
TargetElapsedTime, 428
wolny kanał do odtwarzania, 155
współczynnik proporcji obrazu, 105, 242
współrzędne ekranu, 101
wybór częstotliwości próbkowania, 147
wybór metody do przykrycia, 320
wybór uruchamianego projektu, 36
wyjątek, 100, 307, 328, 451
wyjątek NullReferenceException, 154
wykrywacz poziomu, 137
wykrywacz zbrocza, 137
wykrywanie
kolizji, 269, 382
połączeń telefonicznych, 441
potrząsania, 403
wciśnięć klawiszy, 183
wypełnianie ekranu, 105
wyrażenia w metodzie Draw, 41
wyrażenie, 41
wyrażenie public override, 320
wyrażenie using, 237
wyrażenie warunkowe if z częścią else, 60
wyświetlanie
bieżącego czasu, 121
czasu, 123
klawiszy, 182
obrazów z użyciem przezroczyistości, 130
przycisków, 414
sprajtu, 103, 107
tekstu, 111, 116, 272
zwycięzcy, 170

wytwarzanie sterowane testami, 207
wywoływanie metody, 49, 205
wywoływanie metody Main, 233
wywoływanie przykrywanych metod, 320
wyzwalanie zbroczem, 174

X

XNA Game Studio, 92

Z

zachowania obiektu BatSpriteStruct, 296
zachowanie, 46
zachowanie krawędzi, 261
zachowanie mechanizmu odbijania, 260
zadania kompilatora, 141
zagnieżdżanie pętli, 185
zapisywanie plików, 447
zapisywanie stanu gry, 446, 448

- zarządzanie
 - modyfikacjami, 299
 - profilami graczy, 364
 - stanem gry, 287
 - strukturami, 252
- zasoby i zawartość, 90
- zatrzymywanie oddalania, 220
- zatrzymywanie programu, 33
- zawartość, 92
- zawartość katalogu projektu
 - JakeDisplayContent, 94
- zawężanie, narrowing, 217
- zbiór rozkazów maszynowych, 32
- zdarzenie, 372
- zdarzenie Activated, 441
- zdarzenie Deactivated, 441
- zdarzenie Pressed, 421
- zegar sprzętowy, 119
- zintegrowane środowiska programowania, IDE, 22
- zliczanie wciśnień, 135
- zmiana
 - identyfikatorów, 280
 - nazwy metody, 281
 - nazwy pliku klasy, 239
 - organizacji kontrolek, 32
 - orientacji telefonu, 429
 - położenia przycisku, 136
 - rozmiaru prostokąta, 199
 - stanów, 291
 - stanu klawisza, 174
- zmienna, variable, 43
 - Background, 308
 - fraction, 214
 - KillerTangerine, 339
 - oldKeyState, 175
 - rand, 337
 - redCountingUp, 57
 - redIntensity, 75
 - SoundEffectInstance, 155
 - spriteBatch, 104
 - state, 370
- zmienne
 - intensywności barw, 48
 - lokalne, 47
 - referencyjne, 180
 - stanu, 287
 - tablicowe, 165
- zmniejszanie
 - nieproporcjonalne obrazu, 200
 - proporcjonalne obrazu, 203
- znak #, 142
- znak <, 126
- znak >, 78, 126
- znak =, 44
- znak nowego wiersza, 192
- związki pomiędzy obiektami, 298

Ż

- żądanie utworzenia strumienia danych, 449
- żądanie zapłaty za testowanie, 143

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>

Każdy pasjonat gier komputerowych zadaje sobie czasem pytania typu:

„Niesamowite, jak oni to zrobili?” albo „Jak można uzyskać taki efekt?”. A co się dzieje, jeśli to pytanie przychodzi do głowy programiście? Jego następną myślą zwykle jest: „Muszę spróbować sam to zrobić!”. Na co czekasz? Dziś już nic nie stoi na przeszkodzie! Odkąd Microsoft udostępnił framework XNA, każdy może wymyślić i stworzyć swoją własną profesjonalną grę komputerową, uruchomić ją na prawdziwej konsoli, a nawet wprowadzić na rynek dzięki usłudze Xbox Live® lub Windows® Phone Marketplace.

Jeśli zawsze marzyłeś o profesjonalnym programowaniu gier, ale nie do końca wiedziałeś, jak się za to zabrać, oto książka, która wszystko zmieni! Ten podręcznik będzie dla Ciebie doskonałym przewodnikiem, prowadzącym od opanowania podstaw frameworku XNA, przez zasady programowania w użytecznym języku C#, aż do tworzenia własnych gier. Przejdziesz niezwykłą drogę: na jej początku czeka Cię instalacja i przygotowanie środowiska programistycznego, potem projektowanie realistycznej grafiki gry i jej ciekawe udźwiękowienie, a uwieńczeniem Twoich działań będzie stworzenie profesjonalnych projektów dla konsoli Xbox 360®, telefonów z systemem Windows® Phone 7 czy komputerów PC. Co ważne, nie musisz być zaawansowanym i doświadczonym programistą. Wszystko, czego będziesz potrzebował do realizacji tej misji, znajdziesz w tej książce!

Nauczysz się m.in.:

- pisać kod tworzący zachowania w grze i sterujący nimi
- projektować wygląd gry — grafikę, czcionki, kolory, oświetlenie i efekty trójwymiarowe
- budować obiekty gry, tzw. sprajty
- nagrywać i odtwarzać dźwięki
- przetwarzać dane wejściowe uzyskiwane z padów i klawiatur
- korzystać z ekranu dotykowego i akcelerometru w systemie Windows® Phone
- dodawać stopery, systemy punktacji, elementy zagrażające graczowi i postęp w rozgrywce
- tworzyć gry wieloosobowe i sieciowe

Opanuj sztukę programowania gier i zmieniaj własne pomysły w ekscytującą rzeczywistość!

Rob Miles uczy programowania od ponad dwudziestu pięciu lat. Jest ekspertem w dziedzinie programowania w języku Visual C#, przy użyciu frameworku XNA oraz Microsoft MVP for Windows Phone Development. Oprócz pisania własnych gier i aplikacji oraz pracy dydaktycznej na uniwersytecie Rob brał udział w wielu różnych komercyjnych projektach informatycznych.

helion.pl
księgarnia
internetowa

Nr katalogowy: 7742

Księgarnia internetowa:
<http://helion.pl>

Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

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

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN: 978-83-246-3410-1



Cena: 89,00 zł

Informatyka w najlepszym wydaniu