O'REILLY®

Wydanie IV

Head First C#

Rusz głową!

Doskonały podręcznik
do nauki programowania
w C# i .NET Core

Andrew Stellman
Jennifer Greene

Helion

Tytuł oryginału: Head First C#: A Learner's Guide to Real-World Programming with C# and .NET Core, 4th Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-7900-8

© 2021 Helion S.A.

Authorized Polish translation of the English edition of Head First C#, 4th Edition ISBN 9781491976708
© 2021 Jennifer Greene, Andrew Stellman

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

Microsoft, Windows, Visual Studio, MSDN, the .NET logo, Visual Basic, and Visual C# are registered trademarks of Microsfot Corporation.

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 jakiejkolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem: *https://ftp.helion.pl/przyklady/cshru4.zip*

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/cshru4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

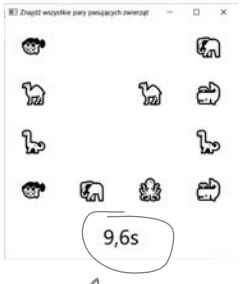
Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

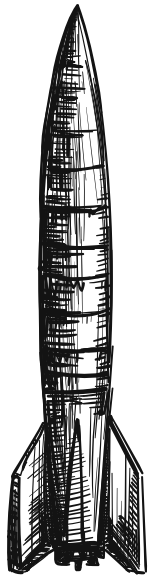
• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści (skrócony)

	Wprowadzenie	xxix
1.	Zacznij pisać programy w języku C#. *Utwórz coś wspaniałego… i to szybko*	1
2.	Zanurz się w C#. *Instrukcje, klasy i kod*	49
	Ćwiczenia z Unity nr 1. *Poznawaj C# z Unity*	87
3.	Obiekty — orientuj się! *Aby kod miał sens*	103
4.	Typy i referencje. *Pobieranie referencji*	155
	Ćwiczenia z Unity nr 2. *Pisanie kodu C# w Unity*	213
5.	Hermetyzacja. *Zapewnij prywatność swoich sekretów*	227
6.	Dziedziczenie. *Drzewo genealogiczne obiektów*	273
	Ćwiczenia z Unity nr 3. *Instancje obiektów gry*	343
7.	Interfejsy, rzutowanie i instrukcja „is”. *Spełnianie obietnic przez klasy*	355
8.	Wyliczenia i kolekcje. *Porządkowanie danych*	405
	Ćwiczenia z Unity nr 4. *Interfejsy użytkownika*	453
9.	LINQ i lambdy. *Kontroluj swoje dane*	467
10.	Odczyt i zapis plików. *Zachowaj dla mnie ostatni bajt*	529
	Ćwiczenia z Unity nr 5. *Ray casting*	577
11.	Kapitan Wspaniały. *Śmierć obiektu*	587
12.	Obsługa wyjątków. *Gaszenie pożarów robi się nudne*	623
	Ćwiczenia z Unity nr 6. *Nawigowanie po scenie*	651
	Ćwiczenie do pobrania — *walka z bossem w grze w dopasowywanie zwierząt*	661
A.	Projekty ASP.NET Core Blazor. *Przewodnik ucznia po Visual Studio for Mac*	663
B.	Kata z programowania. *Podręcznik dla zaawansowanych i/lub niecierpliwych*	725
	Skorowidz	729



Spraw, by gra stała się ciekawsza! W dolnej części okna ma się pojawiać czas od rozpoczęcia gry. Czas ma stale biec naprzód, a zatrzymywać się dopiero po dopasowaniu ostatniego zwierzęcia.



Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

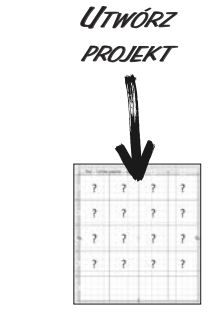
C# według Twojego mózgu. Siedzisz i próbujesz się czegoś *nauczyć*, ale *mózg* stale Ci powtarza, że cała ta nauka *nie jest ważna*. Twój umysł mówi: „Lepiej wyjdź z pokoju i zajmij się ważniejszymi sprawami — dowiedz się, których dzikich zwierząt unikać, a także sprawdź, czy strzelanie z łuku nago to dobry pomysł”. W jaki sposób *możesz* oszukać mózg, aby uznał, że Twoje życie naprawdę zależy od nauki C#?

Dla kogo przeznaczona jest ta książka?	xxx
Wiemy, co sobie myślisz	xxxi
Wiemy, co sobie myśli Twój mózg	xxxi
Metapoznanie — myślenie o myśleniu	xxxiii
Oto co zrobiliśmy	xxxiv
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	xxxv
Przeczytaj to	xxxvi
Zespół redaktorów merytorycznych	xxxiii
Podziękowania	xxxix

Kup książkę Poleć książkę

Zacznij pisać programy w języku C#

1 Utwórz coś wspaniałego... i to szybko



Chcesz tworzyć fantastyczne aplikacje... od razu?

C# to **nowoczesny język programowania** i **cenne narzędzie**, które masz na wyciągnięcie ręki. Ponadto w postaci **Visual Studio** otrzymujesz **fantastyczne środowisko programistyczne** z wysoce intuicyjnymi funkcjami, które sprawiają, że pisanie kodu jest tak łatwe, jak to możliwe. Visual Studio jest nie tylko znakomitym środowiskiem do pisania kodu, ale też **bardzo przydatnym narzędziem do nauki** i eksplorowania języka C#. Brzmi nieźle? Odwróć stronę i zabieraj się do programowania.

Dlaczego warto opanować C#?	2
Visual Studio jest narzędziem do pisania kodu oraz poznawania języka C#	3
Utwórz pierwszy projekt w Visual Studio	4
Napiszmy grę!	6
Proces tworzenia gry	7
Utwórz projekt WPF w Visual Studio	8
Użyj języka XAML do zaprojektowania okna	12
Zaprojektuj okno gry	13
Ustaw wielkość okna i podaj nagłówek za pomocą właściwości w XAML-u	14
Dodaj wiersze i kolumny do siatki w XAML-u	16
Wyrównaj wielkości wierszy i kolumn	17
Dodaj kontrolki TextBlock do siatki	18
Teraz możesz zacząć pisać kod gry	21
Wygeneruj metodę konfigurującą grę	22
Dokończ metodę SetUpGame	24
Uruchom program	26
Dodaj nowy projekt do systemu kontroli wersji	30
Następny krok tworzenia gry to dodanie obsługi kliknięć myszą	33
Spraw, aby kontrolki TextBlock reagowały na kliknięcia myszą	34
Dodaj kod metody TextBlock_MouseDown	37
Spraw, aby pozostałe kontrolki TextBlock wywoływały tę samą procedurę obsługi zdarzeń MouseDown	38
Ukończ grę, dodając zegar	39
Dodaj zegar do kodu gry	40
Użyj debugera do znalezienia przyczyny wyjątku	42
Dodaj resztę kodu i dokończ grę	46
Aktualizowanie kodu w systemie kontroli wersji	47

Kup książkę Poleć książkę

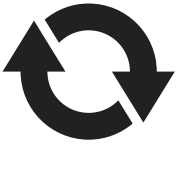
Zanurz się w C#

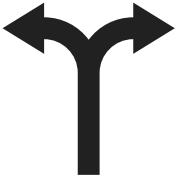
2 Instrukcje, klasy i kod

Nie jesteś jedynie użytkownikiem IDE. Jesteś programistą.

Za pomocą IDE możesz wykonać wiele zadań, ale nawet to środowisko nie zrobi wszystkiego za Ciebie. Visual Studio jest jednym z najbardziej zaawansowanych narzędzi programistycznych, jakie kiedykolwiek stworzono, ale **rozbudowane IDE** to tylko punkt wyjścia. Pora **zagłębić się w kod C#** i dowiedzieć się, jaką ma budowę, jak działa i jak można przejąć nad nim kontrolę. Nie ma ograniczeń w tym, co mogą robić Twoje aplikacje.

Przyjrzyj się plikom aplikacji konsolowej	50
W jednej przestrzeni nazw (i jednym pliku!) mogą się znajdować dwie klasy	52
Instrukcje są cegiełkami do tworzenia aplikacji	55
Programy używają zmiennych do pracy z danymi	56
Generowanie nowej metody używającej zmiennych	58
Dodaj do metody kod używający operatorów	59
Używanie debugera do obserwowania zmian zmiennych	60
Używanie operatorów do pracy ze zmiennymi	62
Instrukcje if służą do podejmowania decyzji	63
Pętle powtarzają wykonywanie operacji	64
Używanie fragmentów kodu do pisania pętli	67
Mechanika interfejsów użytkownika zależy od kontrolek	71
Tworzenie aplikacji WPF do eksperymentowania z kontrolkami	72
Dodawanie kontrolki TextBox do aplikacji	75
Dodawanie kodu C# do obsługi kontrolki TextBlock	77
Dodawanie procedury obsługi zdarzeń dopuszczającej tylko dane liczbowe	79
Dodawanie suwaków w dolnym wierszu siatki	83
Dodawanie kodu C# do obsługi pozostałych kontrolek	84





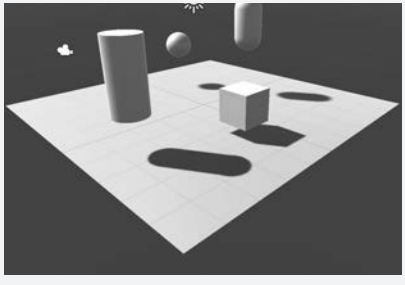
Kup książkę Poleć książkę

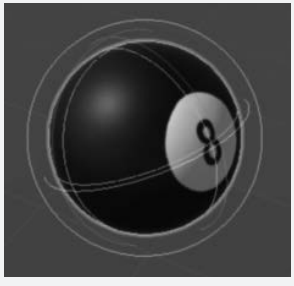
Ćwiczenia z Unity nr 1

Poznawaj C# z Unity

Witaj w pierwszych **ćwiczeniach z Unity w *C#. Rusz głową.*** Pisanie kodu jest umiejętnością i, podobnie jak inne umiejętności, wymaga **praktyki i eksperymentowania**, jeśli chcesz robić postępy. Unity jest wartościowym narzędziem, które Ci w tym pomoże.

Unity jest rozbudowanym narzędziem do projektowania gier	88
Pobieranie Unity Hub	89
Używanie Unity Hub do tworzenia nowego projektu	90
Przejmij kontrolę nad układem Unity	91
Sceną jest środowisko 3D	92
Gry Unity są tworzone za pomocą obiektów gry (GameObject)	93
Używanie narzędzia przenoszenia do przesuwania obiektów gry	94
Okno Inspector wyświetla komponenty obiektów gry	95
Określ materiał kuli	96
Obracanie kuli	99
Bądź kreatywny!	102





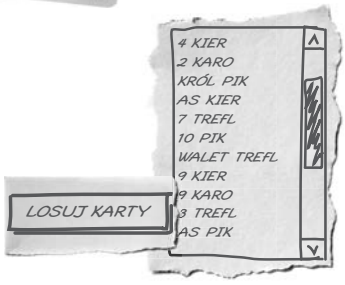
Kup książkę Poleć książkę

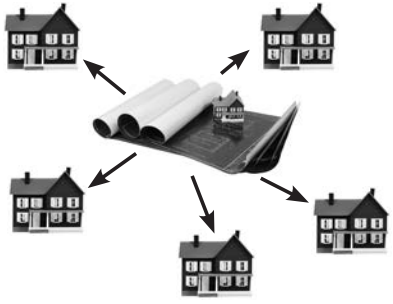
Obiekty — orientuj się!

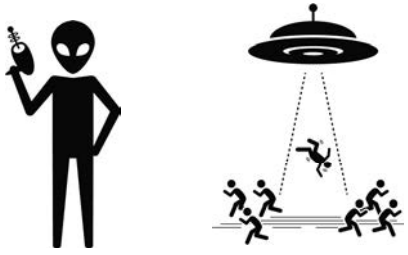
3 Aby kod miał sens

Każdy program, jaki piszesz, ma rozwiązywać problem.

Gdy tworzysz program, zawsze warto najpierw się zastanowić, jaki *problem* ma rozwiązywać. To dlatego **obiekty** są tak przydatne. Umożliwiają ustrukturyzowanie kodu zgodnie z rozwiązywanym problemem, dzięki czemu możesz *skupić się na nim*, a nie na zawiłościach pisania kodu. Gdy korzystasz z obiektów we właściwy sposób i poświęcasz odpowiednią ilość czasu na ich zaprojektowanie, otrzymujesz kod *prosty* do napisania, a także łatwy do zrozumienia i modyfikowania.







Jeśli kod jest przydatny, można ponownie go wykorzystać	104
Niektóre metody przyjmują parametry i zwracają wartość	105
Napisz program, który wybiera karty	106
Utwórz aplikację konsolową PickRandomCards	107
Dokończ metodę PickSomeCards	108
Gotowa klasa CardPicker	110
Ania pracuje nad następną grą	113
Przygotuj papierowy prototyp klasycznej gry	116
Następny krok: zbuduj wersję WPF aplikacji do wybierania kart	118
StackPanel to kontener rozmieszczający inne kontrolki jedna nad drugą	119
Ponownie wykorzystaj klasę CardPicker w nowej aplikacji WPF	120
Użyj siatki i kontrolki StackPanel do określenia układu głównego okna	121
Przygotuj układ okna aplikacji desktopowej do wybierania kart	122
Ania może użyć obiektów do rozwiązania problemu	126
Klasa służy do tworzenia obiektów	127
Gdy tworzysz nowy obiekt na podstawie klasy, jest on nazywany jej instancją	128
Lepsze rozwiązanie dla Ani — wszystko dzięki obiektom	129
Instancja używa pól do śledzenia stanu	133
Dziękujemy za pamięć	136
Co program ma na myśli	137
Czasem kod jest mało czytelny	138
Używaj zrozumiałych nazw klas i metod	140
Zbuduj klasę reprezentującą ludzi	146
Istnieje łatwiejszy sposób na inicjowanie obiektów w C#	148
Użyj okna C# Interactive do uruchamiania kodu C#	154

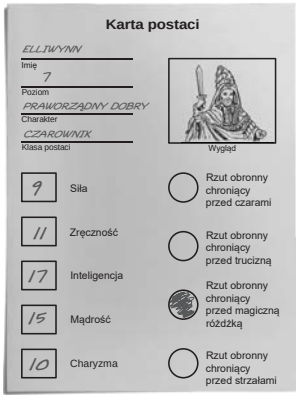
Kup książkę Poleć książkę

Typy i referencje

4 Pobieranie referencji

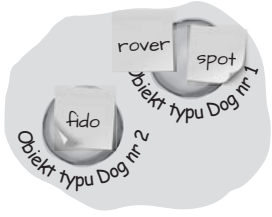
Czym byłyby aplikacje bez danych?

Zastanów się nad tym przez chwilę. Bez danych programy są... no cóż, trudno nawet wyobrazić sobie pisanie kodu bez danych. Potrzebujesz **informacji** od użytkownika i korzystasz z nich do generowania nowych informacji do zwrócenia. Prawie wszystko, co robisz w trakcie programowania, w jakiś sposób dotyczy **pracy z danymi**. W tym rozdziale poznasz tajniki **typów danych** i **referencji** w języku C#, zobaczysz, jak pracować z danymi w programach, a także dowiesz się kilku nowych rzeczy na temat **obiektów** (*i wiesz co — obiekty także są danymi!*).



Utworzenie referencji przypomina zapisanie nazwy na karteczce samoprzylepnej i przyklejenie jej do obiektu. Referencji używasz do nazywania obiektu, aby móc go później wskazać.

Oskar potrzebuje naszej pomocy! 156
Karty postaci służą do zapisywania różnych danych na kartce 157
Typ zmiennej określa, jakiego rodzaju dane może ona przechowywać 158
C# udostępnia kilka typów do przechowywania liczb całkowitych 159
Porozmawiajmy o łańcuchach znaków 161
Literał to wartość bezpośrednio zapisana w kodzie 162
Zmienne są jak kubek na dane 165
Także inne typy mają różne rozmiary 166
10 kilogramów danych w 5-kilogramowej torbie 167
Rzutowanie umożliwia kopiowanie wartości, których C# nie potrafi automatycznie przekształcić na inny typ 168
C# niektóre konwersje przeprowadza automatycznie 171
Gdy wywołujesz metodę, argumenty muszą być zgodne z typami parametrów 172
Pomóżmy Oskarowi w eksperymentach z punktami umiejętności 176
Użyj kompilatora C# do znalezienia błędnego wiersza kodu 178
Używaj zmiennych referencyjnych, aby uzyskać dostęp do obiektów 186
Wiele referencji i ich efekty uboczne 190
Obiekty komunikują się między sobą za pomocą referencji 198
Tablice przechowują wiele wartości 200
Tablice mogą zawierać zmienne referencyjne 201
null oznacza referencję, która nic nie wskazuje 203
Witaj w Budżetowych Kanapkach Jarka Niechluja 208



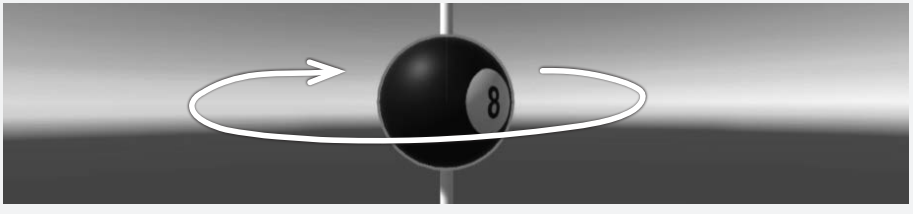
Kup książkę Poleć książkę

Ćwiczenia z Unity nr 2

Pisanie kodu C# w Unity

Unity nie jest *tylko* rozbudowanym, międzyplatformowym silnikiem i edytorem do budowania gier 2D i 3D oraz symulacji. Jest też **doskonałym narzędziem do nabierania praktyki w pisaniu kodu C#**. W tych ćwiczeniach nabierzesz wprawy w pisaniu kodu C# w projekcie w Unity.

Skrypty C# odpowiadają za działanie obiektów gry	214
Dodaj skrypt C# do obiektu gry	215
Napisz kod C# do rotowania kuli	216
Dodaj punkt przerwania i zdebuguj grę	218
Użyj debugera, aby zrozumieć działanie wartości Time.deltaTime	219
Dodaj walec, aby zobaczyć, gdzie znajduje się oś Y	220
Dodaj do klasy pola określające kąt i szybkość rotacji	221
Użyj wywołania Debug.DrawRay, aby zbadać działanie wektorów trójwymiarowych	222
Uruchom grę, aby zobaczyć promień w widoku Scene	223
Rotowanie bili względem punktu sceny	224
Użyj Unity, aby przyjrzeć się rotacji i wektorom	225
Bądź kreatywny!	226



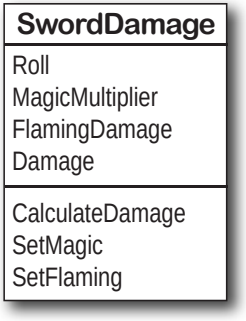
Kup książkę Poleć książkę

Hermetyzacja

5 Zapewnij prywatność swoich sekretów

Czy marzyłeś kiedyś o odrobinie prywatności?

Obiekty czasem czują się podobnie. Tak jak Ty nie chcesz, aby osoby, którym nie ufasz, czytały Twój dziennik lub przeglądały wyciągi bankowe, dobre obiekty nie pozwalają ***innym*** obiektom podglądać swoich pól. W tym rozdziale poznasz wartość **hermetyzacji**. Jest to mechanizm programistyczny pomagający pisać kod łatwy do modyfikowania i stosowania oraz odporny na próby niewłaściwego użycia. **Zapewnisz prywatność danym obiektów** oraz dodasz **właściwości** zabezpieczające dostęp do tych danych.



Pomóż Oskarowi w definiowaniu rzutów obrażeń 228
Napisz aplikację konsolową do obliczania obrażeń 229
Zaprojektuj kod XAML dla wersji WPF kalkulatora obrażeń 231
Kod zaplecza kalkulatora obrażeń 232
Rozmowy przy stole (a może — dyskusja nad kostkami?) 233
Spróbuj wyeliminować błąd 234
Użyj metody Debug.WriteLine do wyświetlania informacji diagnostycznych 235
Łatwo jest przypadkowo błędnie użyć obiektów 238
Hermetyzacja oznacza utworzenie niektórych danych klasy jako prywatnych 239
Użyj hermetyzacji do kontrolowania dostępu do metod i pól klasy 240
Ale czy pole RealName NAPRAWDĘ jest chronione? 241
Prywatne pola i metody są dostępne tylko w instancjach tej samej klasy 242
Po co jest hermetyzacja? Potraktuj obiekt jak czarną skrzynkę… 247
Zastosuj hermetyzację, aby ulepszyć klasę SwordDamage 251
Hermetyzacja zapewnia bezpieczeństwo danych 252
Napisz aplikację konsolową do testowania klasy PaintballGun 253
Właściwości ułatwiają hermetyzację 254
Zmodyfikuj metodę Main, aby używała właściwości Balls 255
Automatycznie implementowane właściwości upraszczają kod 256
Użyj prywatnego settera, aby utworzyć właściwość tylko do odczytu 257
Co zrobić, aby zmienić pojemność magazynka? 258
Do inicjowania właściwości użyj konstruktora z parametrami 259
Podaj argumenty, gdy używasz słowa kluczowego new 260

```
RealName: „Henryk Janik"
Alias: „Damian Marczuk"
Password: „kruk lata o północy"
```

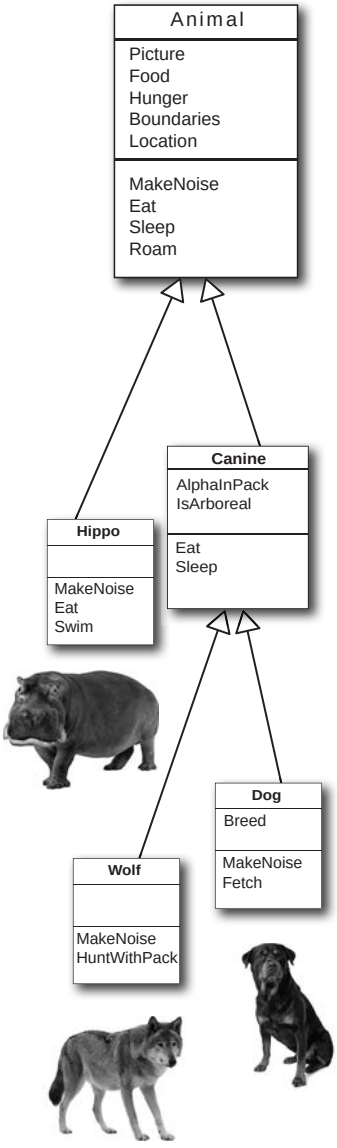

Kup książkę Poleć książkę

Dziedziczenie

6 Drzewo genealogiczne obiektów

Czasem *CHCESZ* być taki jak Twoi rodzice.

Czy natrafiłeś kiedyś na klasę, która robi ***prawie*** dokładnie to, co ma robić ***Twoja*** klasa? Czy naszła Cię myśl, że gdybyś tylko mógł ***zmienić kilka rzeczy***, klasa byłaby idealna? Dzięki **dziedziczeniu** możesz **rozszerzać** istniejące klasy. W ten sposób nowa klasa otrzyma wszystkie operacje starej, zachowując **swobodę** wprowadzania zmian, a Ty będziesz mógł dostosować nową klasę do swoich potrzeb. Dziedziczenie jest jedną z najważniejszych koncepcji i technik w języku C#. Dzięki niemu możesz **uniknąć powtórzeń w kodzie**, uzyskać dokładniejszy **model rzeczywistego świata** i budować aplikacje, które są **łatwiejsze w konserwacji** i **mniej narażone na błędy**.



Obliczanie obrażeń od DODATKOWYCH broni	274
Użyj instrukcji switch do dopasowywania wartości	275
Jeszcze jedna sprawa — czy możemy obliczyć obrażenia od sztyletu? I od buławy?	277
Zastosowanie dziedziczenia pozwala napisać kod tylko raz	278
Zbuduj model klas, zaczynając od klas ogólnych i przechodząc do bardziej wyspecjalizowanych	279
Jak zaprojektujesz symulator zoo?	280
Każda podklasa rozszerza klasę bazową	285
Użyj dwukropka, aby rozszerzyć klasę bazową	290
W podklasie można przesłaniać metody, aby modyfikować lub zastępować odziedziczone składowe	292
Niektóre składowe są implementowane tylko w podklasie	297
Użyj debugera, aby zrozumieć przesłanianie	298
Zbuduj aplikację, aby lepiej poznać słowa kluczowe virtual i override	300
Podklasa może ukrywać metody z klasy bazowej	302
Używaj słów kluczowych override i virtual do dziedziczenia operacji	304
Gdy klasa bazowa zawiera konstruktor, w podklasie trzeba go wywołać	307
Pora ukończyć zadanie dla Oskara	309
Zbuduj system zarządzania rojem pszczół	316
Klasa Queen — jak królowa zarządza robotnicami?	318
Interfejs użytkownika: dodaj kod XAML głównego okna	319
Sprzężenie zwrotne wpływa na grę w zarządzanie rojem	328
Na podstawie niektórych klas nigdy nie należy tworzyć obiektów	332
Klasy abstrakcyjne celowo są niekompletne	334
Jak wspominaliśmy, niektórych klas nie należy używać do tworzenia obiektów	336
Metoda abstrakcyjna nie ma ciała	337
Właściwości abstrakcyjne działają tak jak metody abstrakcyjne	338

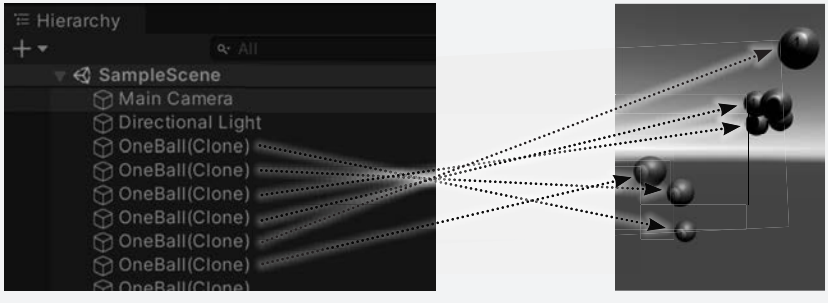
Kup książkę Poleć książkę

Ćwiczenia z Unity nr 3

Instancje obiektów gry

C# jest językiem obiektowym, a ponieważ celem ćwiczeń z Unity w książce *Rusz głową. C#* jest *praktykowanie pisania kodu w C#*, zrozumiałe jest, że w tych ćwiczeniach skupisz się na tworzeniu obiektów.

Zbudujmy grę w Unity!	344
Utwórz nowy materiał w katalogu Materials	345
Wygeneruj bilę w losowym punkcie sceny	346
Użyj debugera, aby zrozumieć wywołanie Random.value	347
Przekształć obiekt gry w obiekt prefab	348
Utwórz skrypt do sterowania grą	349
Dołącz skrypt do głównej kamery	350
Wciśnij Play, aby uruchomić kod	351
Użyj okna Inspector do pracy z instancjami obiektów gry	352
Wykorzystaj silnik fizyki, aby uniknąć pokrywania się bil	353
Bądź kreatywny!	354



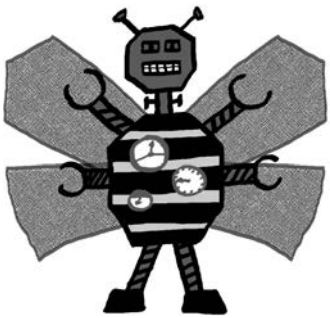
Kup książkę Poleć książkę

Interfejsy, rzutowanie i instrukcja „is”

7 Spełnianie obietnic przez klasy

Chcesz, aby obiekt wykonywał określone zadanie? Zastosuj interfejs.

Czasem programista chce pogrupować obiekty na podstawie **tego, co potrafią robić**, a nie na bazie klas, po jakich dziedziczą. **Interfejsy** to umożliwiają. Możesz użyć interfejsu do zdefiniowania **określonego zadania**. Każda instancja klasy **implementującej** dany interfejs *gwarantuje wykonanie tego zadania* niezależnie od tego, po jakich klasach dziedziczy. Aby ten mechanizm działał, każda klasa implementująca interfejs musi zobowiązać się do **spełnienia wszystkich obietnic** — w przeciwnym razie kompilator połamie jej kolana, rozumiesz?



Rój został zaatakowany!	356
Można zastosować rzutowanie, aby wywołać metodę DefendHive	357
Interfejs definiuje metody i właściwości, jakie klasa musi implementować	358
Nabierz wprawy w używaniu interfejsów	360
Nie możesz utworzyć obiektu typu interfejsu, ale możesz utworzyć referencję do interfejsu	366
Referencje typów interfejsowych są zwykłymi referencjami do obiektów	369
RoboPszczoła 4000 potrafi wykonywać pracę robotnic bez zużywania cennego miodu	370
Właściwość Job interfejsu IWorker to prowizorka	374
Do sprawdzania typu obiektu użyj instrukcji „is”	375
Użyj „is” do uzyskania dostępu do metod w podklasie	376
Co zrobić, jeśli inne zwierzęta też mają pływać lub polować w stadzie?	378
Używaj interfejsów do pracy z klasami, które wykonują to samo zadanie	379
Bezpiecznie poruszaj się po hierarchii klas za pomocą instrukcji „is”	380
C# udostępnia też inne narzędzie do bezpiecznej konwersji typów: słowo kluczowe „as”	381
Stosuj rzutowanie w górę i w dół, aby poruszać się po hierarchii klas	382
Rzutowanie w górę przekształca obiekt typu CoffeeMaker w obiekt typu Appliance	384
Rzutowanie w dół zmienia obiekt typu Appliance ponownie w obiekt typu CoffeeMaker	385
Rzutowanie w górę i w dół działa także dla interfejsów	386
Interfejsy mogą dziedziczyć po innych interfejsach	388
Interfejsy mogą zawierać składowe statyczne	395
Domyślne implementacje zawierają ciało metod interfejsów	396
Dodaj metodę ScareAdults z implementacją domyślną	397
Wiązanie danych powoduje automatyczną aktualizację kontrolek WPF	399
Zmodyfikuj system zarządzania rojem, aby używał wiązania danych	400
Polimorfizm oznacza, że jeden obiekt może przyjmować wiele różnych postaci	403

Kup książkę Poleć książkę

Wyliczenia i kolekcje

8 Porządkowanie danych

Dane nie zawsze są tak uporządkowane, jak byś sobie tego życzył.

W praktyce nie otrzymujesz danych w postaci uporządkowanych porcji i elementów. Nic z tego — dane będą trafiać do Ciebie w postaci rozmaitych **ładunków, stert i wiązek**. Będziesz potrzebować rozbudowanych narzędzi, aby je wszystkie uporządkować. Przydadzą Ci się do tego **wyliczenia** i **kolekcje**. Wyliczenia to typy, które pozwalają zdefiniować poprawne wartości do kategoryzowania danych. Kolekcje są specjalnymi obiektami przechowującymi wiele wartości. Pozwalają **przechowywać i sortować** wszelkie dane, jakie program musi przetwarzać, oraz **zarządzać** nimi. Dzięki temu możesz przeznaczyć czas na zastanowienie się nad programem pracującym z danymi, a obsługą ich samych zajmą się kolekcje.

Rzadko używana karta „książę bawołów".

Łańcuchy znaków nie zawsze nadają się do przechowywania kategorii danych 406
Wyliczenia umożliwiają pracę ze zbiorem poprawnych wartości 407
Wyliczenia umożliwiają reprezentowanie liczb za pomocą nazw 408
Moglibyśmy użyć tablicy do tworzenia talii kart… 411
Listy umożliwiają łatwe przechowywanie kolekcji czegokolwiek 413
Listy dają więcej możliwości niż tablice 414
Utwórzmy aplikację do przechowywania butów 417
Generyczne kolekcje mogą przechowywać wartości dowolnego typu 420
Inicjalizatory kolekcji są podobne do inicjalizatorów obiektów 426
Utwórzmy listę kaczek 427
Listy są proste, ale SORTOWANIE bywa skomplikowane 428
Interfejs IComparable<Duck> pomaga liście sortować kaczki 429
Użyj obiektu typu IComparer, aby poinformować listę, jak sortować elementy 430
Utwórz obiekt komparatora 431
Przesłonienie metody ToString umożliwia obiektowi opisywanie samego siebie 435
Zaktualizuj pętle foreach, aby obiekty typów Duck i Card wyświetlały informacje o sobie w konsoli 436
Możesz zrzutować w górę całą listę, używając interfejsu IEnumerable<T> 440
Użyj słownika do przechowywania kluczy i wartości 442
Przegląd możliwości słowników 443
Napisz program używający słownika 444
Jeszcze WIĘCEJ typów kolekcji… 445
Kolejka to kolekcja typu FIFO (first in, first out) 446
Stos to kolekcja typu LIFO (last in, last out) 447
Ćwiczenie do pobrania — dwie talie 452

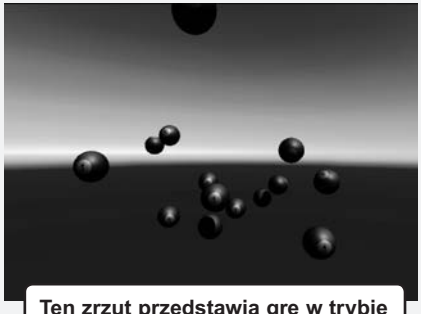
Kup książkę Poleć książkę

Ćwiczenia z Unity nr 4

Interfejsy użytkownika

W poprzednich ćwiczeniach z Unity zacząłeś pisać grę, używając obiektów prefab do tworzenia obiektów gry, które pojawiają się w losowych punktach przestrzeni gry trójwymiarowej i latają po ekranie. W tych ćwiczeniach z Unity będziesz kontynuować te prace, co pozwoli Ci wykorzystać wiedzę na temat interfejsów z C# i inne informacje.

Dodaj wynik zwiększany po kliknięciu bili przez gracza	454
Dodaj dwa różne tryby gry	455
Dodaj tryb gry	456
Dodaj interfejs użytkownika do gry	458
Skonfiguruj obiekt Text wyświetlający wynik w interfejsie użytkownika	459
Dodaj przycisk, który wywołuje metodę uruchamiającą grę	460
Dodaj kod obsługi przycisku Zagraj ponownie i pola z wynikiem	461
Dokończ kod gry	462
Wykaż się kreatywnością!	466



Ten zrzut przedstawia grę w trybie rozgrywki. Bile są dodawane, a gracz może je klikać, aby poprawić wynik.

Po dodaniu ostatniej bili gra przechodzi w tryb „Koniec gry". Wyświetlany jest przycisk „Zagraj ponownie", a gra nie dodaje nowych bil.

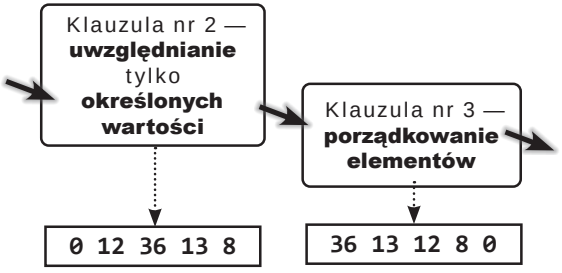
Kup książkę Poleć książkę

LINQ i lambdy

9 Kontroluj swoje dane

Świat jest sterowany przez dane. Wszyscy musimy nauczyć się w nim żyć.

Minęły już dni, kiedy można było dniami, a nawet tygodniami programować bez przetwarzania **dużych ilości danych**. Dziś ***wszystko kręci się wokół danych***. Tu właśnie przydaje się technologia **LINQ**. Jest to mechanizm języka C# i platformy .NET, który nie tylko pozwala w intuicyjny sposób **pisać kwerendy danych** z kolekcji platformy .NET, ale też **grupować dane** i **scalać dane z różnych źródeł**. W tym rozdziale dodasz **testy jednostkowe**, aby mieć pewność, że kod działa w oczekiwany sposób. Gdy już nauczysz się przekształcać dane na możliwe do przetworzenia porcje, zastosujesz **wyrażenia lambda**, aby zrefaktoryzować kod C#, by był jeszcze bardziej zwięzły.



Jacek jest wielkim fanem Kapitana Wspaniałego 468
Użyj technologii LINQ do pisania kwerend dotyczących kolekcji 470
LINQ działa dla każdego obiektu z implementacją interfejsu IEnumerable<T> 472
Składnia kwerend LINQ 475
LINQ działa dla obiektów 477
Użyj kwerendy LINQ do ukończenia aplikacji dla Jacka 478
Słowo kluczowe var powoduje, że C# wywnioskowuje typy zmiennych 480
Kwerendy LINQ są wykonywane dopiero, gdy chcesz użyć ich wyników 487
Użyj kwerendy grupującej do rozdzielenia sekwencji na grupy 488
Użyj kwerend złączających do scalania danych z dwóch sekwencji 491
Użyj słowa kluczowego new do utworzenia typu anonimowego 492
Dodaj projekt z testami jednostkowymi do aplikacji zarządzającej kolekcją komiksów Jacka 502
Napisz pierwszy test jednostkowy 503
Napisz test jednostkowy metody GetReviews 505
Napisz testy jednostkowe uwzględniające przypadki brzegowe i nietypowe dane 506
Użyj operatora => do tworzenia wyrażeń lambda 508
Jazda próbna z lambdami 509
Refaktoryzacja kodu dotyczącego klaunów za pomocą lambd 510
Użyj operatora ?: do podejmowania decyzji w lambdach 513
Wyrażenia lambda i LINQ 514
Kwerendy LINQ można zapisywać jako łańcuch metod LINQ 515
Użyj operatora => do tworzenia wyrażeń switch 517
Omówienie klasy Enumerable 521
Ręczne tworzenie sekwencji umożliwiającej wyliczanie 522
Użyj instrukcji yield return do tworzenia własnych sekwencji 523
Użyj instrukcji yield return, aby zrefaktoryzować klasę ManualSportSequence 524
Ćwiczenie do pobrania: Go Fish 527

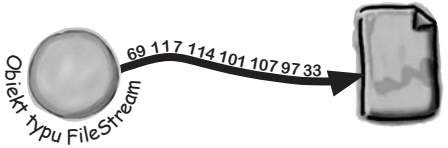
Kup książkę Poleć książkę

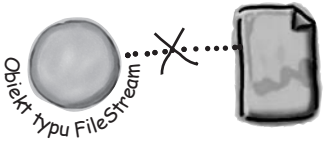
Odczyt i zapis plików

10 Zachowaj dla mnie ostatni bajt

Czasem opłaca się być (wy)trwałym.

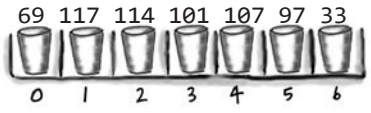
Do tej pory wszystkie omawiane programy miały krótki czas życia. Były uruchamiane, działały przez pewien czas, po czym kończyły pracę. Jednak ten model nie zawsze wystarcza — zwłaszcza gdy przetwarzasz ważne informacje. Potrzebna jest możliwość **zapisywania pracy**. W tym rozdziale zobaczysz, jak **zapisywać dane w pliku**, a także jak **wczytywać informacje** z pliku. Poznasz też **strumienie**, dowiesz się, jak zapisywać obiekty w plikach z użyciem **serializacji**, i przyjrzysz się bitom i bajtom w **danych szesnastkowych, Unicode i dwójkowych**.





W .NET do odczytu i zapisu danych używane są strumienie	530
Różne strumienie wczytują i zapisują różne dane	531
Klasa FileStream wczytuje i zapisuje bajty w plikach	532
Zapis tekstu w pliku w trzech prostych krokach	533
Kanciarz realizuje następny diaboliczny plan	534
Użyj klasy StreamReader do odczytu pliku	537
Dane mogą przechodzić przez więcej niż jeden strumień	538
Używanie statycznych klas File i Directory do pracy z plikami i katalogami	542
IDisposable gwarantuje, że obiekty zostaną poprawnie zamknięte	545
Użyj klasy MemoryStream do strumieniowania danych do pamięci	547
Co dzieje się z obiektem w czasie serializacji?	553
Czym dokładnie JEST stan obiektu? Co trzeba zapisać?	554
Użyj klasy JsonSerialization do serializacji obiektów	556
Format JSON obejmuje tylko dane, a nie specyficzne typy języka C#	559
Następny krok — zanurzenie się w dane	561
Łańcuchy znaków w C# są kodowane w formacie Unicode	563
Visual Studio bardzo dobrze współdziała z Unicode	565
Platforma .NET używa formatu Unicode do przechowywania znaków i tekstu	566
C# może wykorzystać tablice bajtów do przenoszenia danych	568
Użyj klasy BinaryWriter do zapisu danych binarnych	569
Użyj klasy BinaryReader do wczytania danych	570
Przeglądarka danych szesnastkowych pozwala zobaczyć bajty z pliku	572
Użyj metody Stream.Read do wczytania bajtów ze strumienia	574
Zmodyfikuj przeglądarkę danych szesnastkowych, aby używała argumentów z wiersza poleceń	575
Ćwiczenie do pobrania — zabawa w chowanego	576

Eureka! →



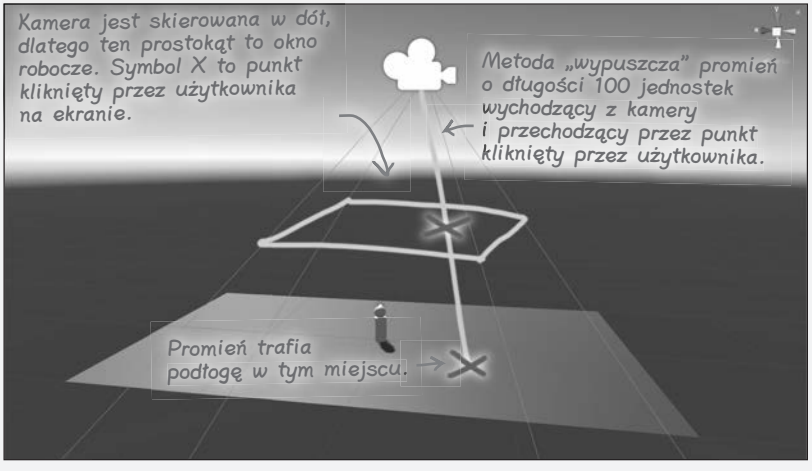
Kup książkę Poleć książkę

Ćwiczenia z Unity nr 5

Ray casting

Gdy konfigurujesz scenę w Unity, tworzysz wirtualny trójwymiarowy świat, w którym postacie z gry mogą się poruszać. Jednak w większości gier wiele wydarzeń nie jest bezpośrednio kontrolowanych przez gracza. W jaki więc sposób obiekty poruszają się po scenie? W tych ćwiczeniach zobaczysz, jak C# może w tym pomóc.

Utwórz nowy projekt w Unity i zacznij przygotowywać scenę	578
Przygotuj kamerę	579
Utwórz obiekt gry reprezentujący gracza	580
Wprowadzenie do systemu nawigowania w Unity	581
Przygotowanie siatki nawigacyjnej	582
Spraw, aby postać automatycznie poruszała się po obszarze gry	583



Kup książkę Poleć książkę

KAPITAN WSPANIAŁY

ŚMIERĆ OBIEKTU

C#. Rusz głową

4 zł | Rozdział 11.

Życie i śmierć obiektu 590
Używaj (ostrożnie) klasy GC do wymuszania odśmiecania pamięci 591
Finalizator obiektu — Twoja ostatnia szansa, by coś ZROBIĆ 592
Kiedy DOKŁADNIE uruchamiany jest finalizator? 593
Finalizatory nie mogą zależeć od innych obiektów 595
Struktura wygląda jak obiekt… 599
Wartości są kopiowane; referencje są przypisywane 600
Struktury są typami bezpośrednimi; klasy są typami referencyjnymi 601
Stos a sterta — więcej o pamięci 603
Używaj parametrów out, aby zwrócić z metody więcej niż jedną wartość 606
Przekazywanie przez referencję z użyciem modyfikatora ref 607
Używaj parametrów opcjonalnych do podawania wartości domyślnych 608
Referencja null nie wskazuje żadnego obiektu 609
Typy referencyjne niedopuszczające null pomagają unikać wyjątków NRE 610
Operator ?? pomaga radzić sobie z null 611
Typy bezpośrednie dopuszczające null mogą równać się null i być bezpiecznie obsługiwane 612
Kapitan Wspaniały — ale nie do końca 613
Metody rozszerzające dodają nowe operacje do ISTNIEJĄCYCH klas 617
Rozszerzanie typu podstawowego — string 619

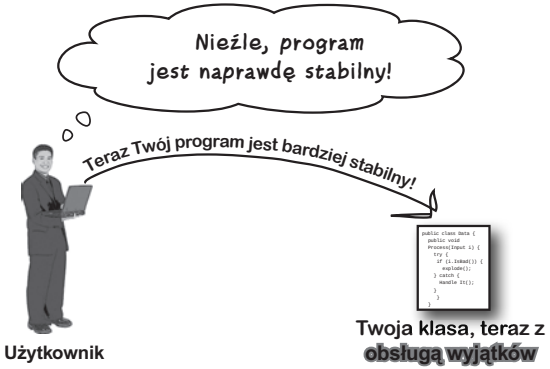
Kup książkę Poleć książkę

Obsługa wyjątków

12 Gaszenie pożarów staje się nudne

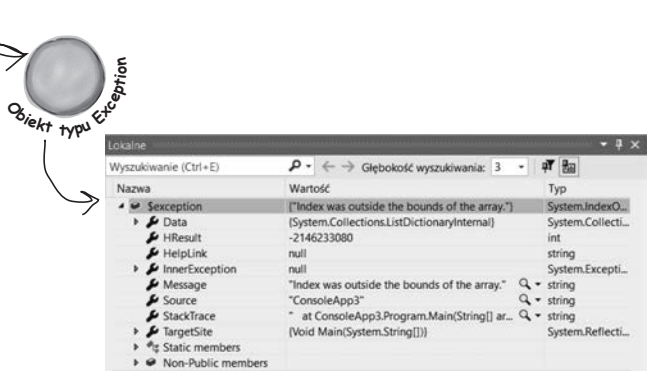
Programiści nie powinni być strażakami.

Ciężko pracowałeś, przedzierając się przez podręczniki techniczne i kilka wciągających książek z serii *Rusz głową*, i dotarłeś na sam szczyt swojej profesji. Ale spanikowani współpracownicy nadal dzwonią do Ciebie w środku nocy, ponieważ **Twój program się zawiesza** lub **nie działa tak, jak powinien**. Nic nie wybija programistów z rytmu tak, jak konieczność wyeliminowania dziwnego błędu. Jednak dzięki **obsłudze wyjątków** możesz pisać kod **radzący sobie z problemami**, które się pojawiają. Jeszcze lepsze jest to, że możesz nawet zaplanować działania na wypadek takich problemów, aby program **kontynuował działanie** po ich wystąpieniu.



Przeglądarka danych szesnastkowych wczytuje nazwę pliku z wiersza poleceń 624
Gdy program zgłasza wyjątek, CLR generuje obiekt wyjątku 628
Wszystkie obiekty wyjątków dziedziczą po klasie System.Exception 629
Istnieją pliki, które nie umożliwiają wykonania zrzutu szesnastkowego 632
Co się dzieje, gdy wywoływana metoda jest ryzykowna? 633
Obsługa wyjątków za pomocą bloków try-catch 634
Użyj debugera do prześledzenia przepływu sterowania w bloku try-catch 635
Jeśli masz kod, który trzeba ZAWSZE uruchamiać, użyj bloku finally 636
Ogólny blok catch obsługuje wyjątki typu System.Exception 637
Używaj wyjątku odpowiedniego do sytuacji 642
Filtry wyjątków pomagają tworzyć precyzyjne bloki do ich obsługi 646
Najgorszy blok catch W HISTORII — ogólny blok catch z komentarzami 648
Tymczasowe rozwiązania są akceptowalne (tymczasowo) 649


```
int[] anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```



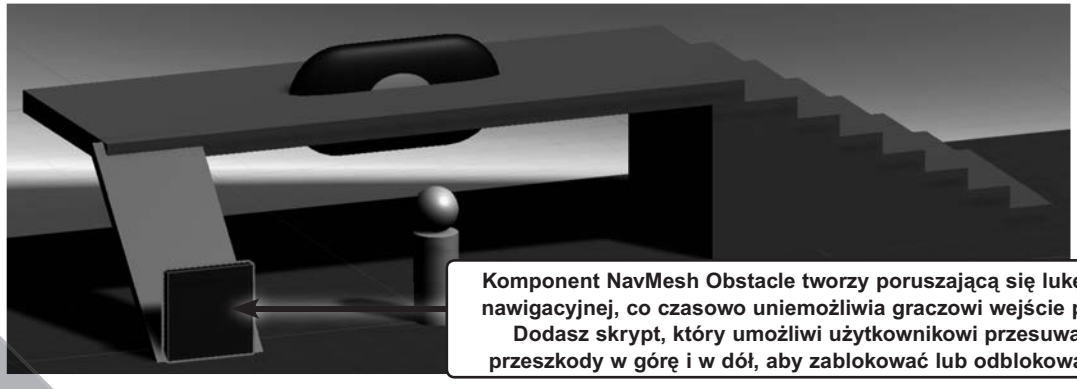
Kup książkę Poleć książkę

Ćwiczenia z Unity nr 6

Nawigowanie po scenie

W poprzednich ćwiczeniach z Unity utworzyłeś scenę z podłogą (płaszczyzną) i gracza (kulę powiązaną z walcem). Ponadto użyłeś siatki nawigacyjnej, obiektu NavMesh Agent i ray castingu, aby gracz podążał w miejsca kliknięcia myszą na scenie. W tych ćwiczeniach dodasz elementy do sceny, używając C#.

Zacznijmy od miejsca, w którym zakończyliśmy poprzednie ćwiczenia z Unity	652
Dodaj platformę do sceny	653
Użyj opcji wstępnego obliczania, aby umożliwić chodzenie po platformie	654
Dodaj schody i rampę do siatki nawigacyjnej	655
Rozwiąż problem z wysokością w siatce nawigacyjnej	657
Dodaj komponent NavMesh Obstacle	658
Dodaj skrypt do przesuwania przeszkody w górę i w dół	659
Wykaż się kreatywnością!	660



Komponent NavMesh Obstacle tworzy poruszającą się lukę w siatce nawigacyjnej, co czasowo uniemożliwia graczowi wejście po rampie. Dodasz skrypt, który umożliwi użytkownikowi przesuwanie tej przeszkody w górę i w dół, aby zablokować lub odblokować rampę.

Kup książkę Poleć książkę

Projekty ASP.NET Core Blazor

A Przewodnik ucznia po Visual Studio for Mac

Twój Mac jest pełnoprawnym obywatelem w świecie C# i .NET.

Dlaczego powinieneś opanować C#? 664
Utwórz pierwszy projekt w Visual Studio 666
Napiszmy grę! 670
Utwórz projekt WebAssembly platformy Blazor w Visual Studio 672
Uruchom aplikację sieciową Blazor w przeglądarce 674
Teraz jesteś gotowy, by zacząć pisać kod aplikacji 676
Dokończ tworzenie listy emoji i wyświetl ją w aplikacji 680
Rozmieść zwierzęta w losowej kolejności 682
Uruchamianie gry w debugerze 684
Dodaj nowy projekt do systemu kontroli wersji 688
Dodaj kod C# do obsługi kliknięć myszą 689
Dodaj procedury obsługi kliknięć do przycisków 690
Przetestuj kod obsługi zdarzeń 692
Użyj debugera do rozwiązania problemu 693
Namierz błąd, który powoduje problem 696
Dodaj kod do resetowania gry po jej ukończeniu 698
Dodaj zegar do kodu gry 702
Uporządkuj menu nawigacyjne 704
Kontrolki wpływają na mechanikę interfejsów użytkownika 706
Utwórz nowy projekt WebAssembly platformy Blazor 707
Utwórz stronę z suwakiem 708
Dodaj do aplikacji pole na dane tekstowe 710
Dodaj do aplikacji okna wyboru koloru i daty 713
Następne zadanie — zbuduj bazującą na platformie Blazor wersję aplikacji do wybierania kart 714
Układ strony jest określany za pomocą wierszy i kolumn 716
Wiązanie danych umożliwia modyfikowanie zmiennej za pomocą suwaka 717
Witaj w Budżetowych Kanapkach Jarka Niechluja! 720

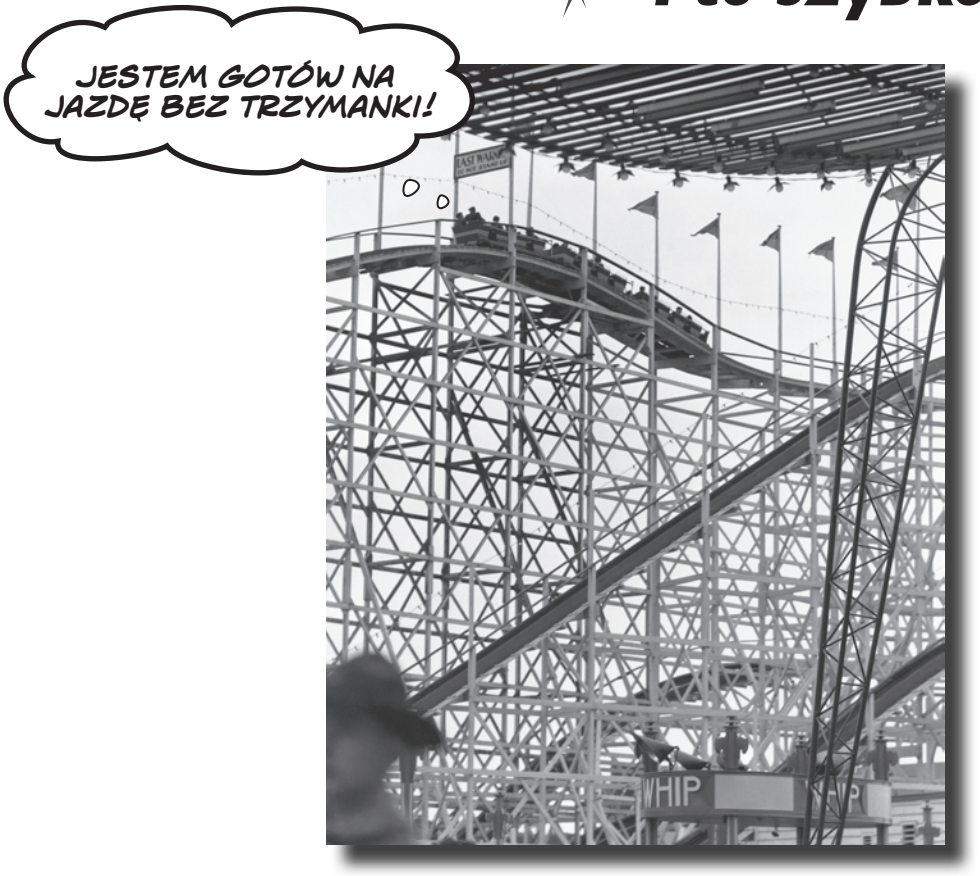
Kata z programowania

B Podręcznik dla zaawansowanych i/lub niecierpliwych

Skorowidz 729

Kup książkę Poleć książkę

Utwórz coś wspaniałego... i to szybko!



Chcesz tworzyć fantastyczne aplikacje... od razu?
C# to **nowoczesny język programowania** i **cenne narzędzie**, które masz na wyciągnięcie ręki. Ponadto w postaci **Visual Studio** otrzymujesz **fantastyczne środowisko programistyczne** z wysoce intuicyjnymi funkcjami, które sprawiają, że pisanie kodu jest tak łatwe, jak to możliwe. Visual Studio jest nie tylko znakomitym środowiskiem do pisania kodu, ale też **bardzo przydatnym narzędziem do nauki** i eksplorowania języka C#. Brzmi nieźle? Odwróć stronę i zabieraj się do programowania.

Kup książkę Poleć książkę

Dlaczego warto opanować C#?

C# jest prostym, nowoczesnym językiem, który umożliwia uzyskanie niezwykłych efektów. W trakcie nauki C# poznasz nie tylko sam język. C# otworzy przed Tobą cały świat platformy .NET i da Ci dostęp do otwartej platformy o wielkich możliwościach, która pozwala na pisanie aplikacji różnego rodzaju.

Upewnij się, że instalujesz środowisko Visual Studio, a nie Visual Studio Code.

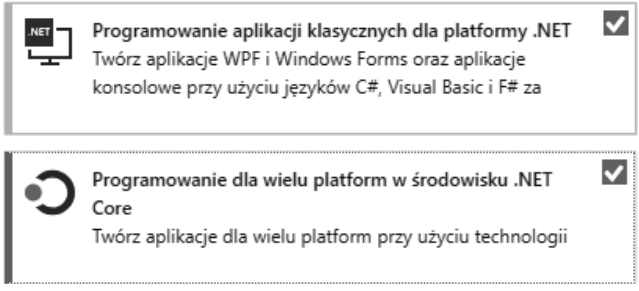
Visual Studio Code jest świetnym otwartym edytorem kodu działającym w różnych systemach, ale nie jest dostosowany do programowania na platformie .NET tak dobrze jak Visual Studio. To dlatego w tej książce to Visual Studio będzie używane jako narzędzie do nauki i poznawania C#.

Visual Studio jest Twoją przepustką do języka C#

Jeśli jeszcze nie zainstalowałeś środowiska Visual Studio 2019, pora to zrobić. Otwórz stronę *visualstudio.microsoft.com* i **pobierz wersję Visual Studio Community**. Jeżeli masz ją już zainstalowaną, uruchom instalator środowiska Visual Studio, aby zaktualizować zainstalowane komponenty.

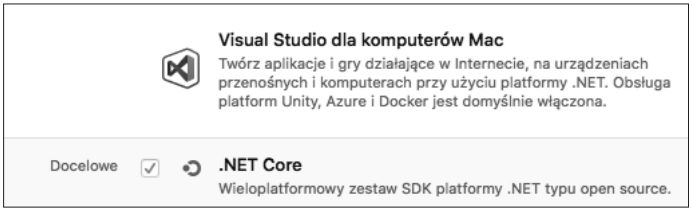
Jeśli używasz systemu Windows...

Koniecznie zaznacz opcje umożliwiające programowanie dla wielu platform w środowisku .NET Core i programowanie aplikacji desktopowych na platformę .NET. Nie zaznaczaj jednak opcji *Opracowywanie gier za pomocą aparatu Unity*. W dalszych rozdziałach książki zajmiesz się programowaniem gier 3D z użyciem silnika Unity, ale zainstalujesz go osobno.



Jeśli używasz systemu Mac...

Pobierz i uruchom instalator środowiska Visual Studio for Mac. Upewnij się, że zaznaczona jest docelowa platforma .NET Core.



W systemie Windows możesz też tworzyć projekty ASP.NET! Wystarczy zaznaczyć opcję „Opracowywanie zawartości dla platformy ASP.NET i sieci Web" w trakcie instalowania środowiska Visual Studio.

Większość aplikacji z tej książki to aplikacje konsolowe na platformę .NET Core, które działają w systemach Windows i Mac. W niektórych rozdziałach opisane są projekty dla systemu Windows, na przykład gra w dopasowywanie zwierząt, o której mowa w dalszej części tego rozdziału. W przypadku takich projektów zajrzyj do dodatku „Przewodnik ucznia po Visual Studio for Mac". Znajdziesz tam *kompletny zastępnik* kodu z rozdziału 1. i wersje innych projektów WPF bazujące na platformie ASP.NET Core Blazor.

Kup książkę Poleć książkę

Visual Studio jest narzędziem do pisania kodu oraz poznawania języka C#

Do pisania kodu w języku C# możesz używać notatnika lub innego edytora tekstu, jednak istnieje lepsze rozwiązanie. **Zintegrowane środowisko programistyczne** (ang. *integrated development environment* — **IDE**) to edytor tekstu, narzędzie do projektowania wizualnego, menedżer plików, debuger — jest jak scyzoryk szwajcarski do wykonywania wszystkich zadań związanych z pisaniem kodu.

Oto tylko kilka wybranych operacji, w jakich Visual Studio Ci pomoże:

1. **SZYBKIE tworzenie aplikacji.** Język C# ma duże możliwości i jest łatwy w nauce, a IDE Visual Studio upraszcza prace, automatycznie wykonując wiele żmudnych zadań. Oto kilka czynności, które Visual Studio wykonuje za programistę (lub w których bardzo mu pomaga):

   - ★ zarządzanie wszystkimi plikami projektu,
   - ★ ułatwianie edycji kodu projektu,
   - ★ zarządzanie grafiką, plikami dźwiękowymi, ikonami i innymi zasobami projektu,
   - ★ wsparcie w debugowaniu kodu przez wykonywanie go wiersz po wierszu.

2. **Projektowanie atrakcyjnego interfejsu użytkownika.** Narzędzie do projektowania wizualnego w IDE Visual Studio jest jednym z najłatwiejszych w użyciu narzędzi tego typu. Wykonuje za programistę tyle pracy, że przekonasz się, iż tworzenie interfejsów użytkownika będzie jednym z najbardziej satysfakcjonujących aspektów tworzenia aplikacji w C#. Możesz tworzyć kompletne profesjonalne programy bez konieczności poświęcania wielu godzin na dopracowywanie interfejsu użytkownika (chyba że będziesz miał na to ochotę).

Jeśli używasz środowiska Visual Studio for Mac, możesz tworzyć równie atrakcyjnie wyglądające aplikacje, ale zamiast używać języka XAML, połącz C# z HTML-em.

3. **Tworzenie znakomicie wyglądających programów.** Gdy **połączysz C# z XAML-em**, językiem znaczników służącym do projektowania interfejsów użytkownika dla aplikacji desktopowych WPF, uzyskasz jedno z najwydajniejszych narzędzi do tworzenia programów z interfejsem graficznym. Możesz wykorzystać je do tworzenia oprogramowania, które wygląda równie dobrze, jak działa.

Interfejs użytkownika aplikacji WPF jest tworzony w języku XAML (eXtensible Application Markup Language). W Visual Studio praca z językiem XAML jest naprawdę prosta.

Visual Studio to fantastyczne środowisko programistyczne, ale tu posłuży także do eksplorowania języka C#.

4. **Nauka i eksplorowanie języka C# oraz platformy .NET.** Visual Studio jest najwyższej klasy narzędziem programistycznym, ale — na szczęście dla nas — także znakomitym narzędziem wspomagającym naukę. ***Użyjemy tu tego środowiska IDE do eksplorowania języka C#***, co pozwoli *szybko* przenieść ważne zagadnienia programistyczne do Twojego mózgu.

W tej książce Visual Studio czasem będzie nazywane po prostu „środowiskiem IDE".

Kup książkę Poleć książkę

Utwórz pierwszy projekt w Visual Studio

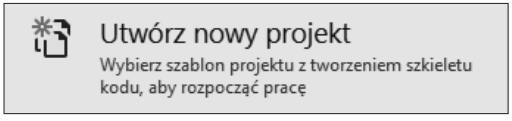
Najlepszy sposób na naukę języka C# polega na rozpoczęciu pisania kodu. Dlatego użyjesz Visual Studio do **utworzenia nowego projektu**… i od razu zaczniesz pisać kod!

Zrób to!

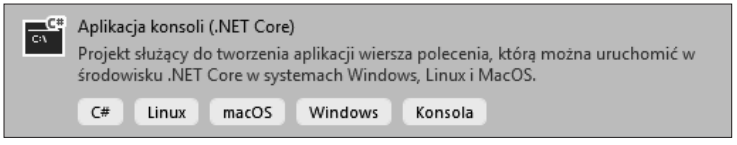
Gdy natrafisz na tekst Zrób to! (lub Zrób to teraz!, Debuguj to! itd.), otwórz Visual Studio i wykonaj opisane kroki. Dokładnie wyjaśniamy, co należy zrobić, i pokazujemy, na co zwrócić uwagę, aby jak najbardziej skorzystać z danego przykładu.

1 Tworzenie nowego projektu dla aplikacji konsolowej na platformę .NET Core.

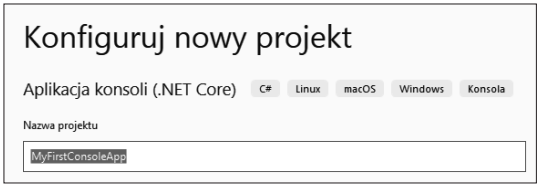
Uruchom środowisko Visual Studio 2019. Po uruchomieniu pojawi się okno *Utwórz nowy projekt* z kilkoma różnymi opcjami. Wybierz opcję ***Utwórz nowy projekt***. Nie martw się, jeśli zamknąłeś okno — zawsze możesz do niego wrócić, wybierając w menu opcję *Plik/Nowy/Projekt*.



Wybierz projekt typu ***Aplikacja konsoli (.NET Core)***, klikając go. Następnie kliknij przycisk ***Dalej***.



Nazwij projekt ***MyFirstConsoleApp*** i kliknij przycisk ***Utwórz***.



*Jeśli używasz **Visual Studio for Mac**, kod tego projektu (i wszystkich projektów aplikacji konsolowych dla .NET Core) będzie wyglądać identycznie, ale niektóre funkcje IDE będą inne. Wersję tego rozdziału dla komputerów Mac znajdziesz w dodatku **„Przewodnik ucznia po Visual Studio for Mac”**.*

2 Przyjrzyj się kodowi nowej aplikacji.

Gdy Visual Studio tworzy nowy projekt, generuje punkt wyjścia, od którego możesz zacząć. Zaraz po zakończeniu tworzenia nowych plików aplikacji środowisko powinno otworzyć plik *Program.cs* z następującym kodem:

```
Odwołania: 0
class Program
{
    Odwołania: 0
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Gdy Visual Studio tworzy nowy projekt aplikacji konsolowej, automatycznie dodaje klasę o nazwie Program.

Ta klasa rozpoczyna się od metody Main, która zawiera jedną instrukcję, wyświetlającą w konsoli wiersz tekstu. Klasy i metody są dokładnie opisane w rozdziale 2.

Kup książkę Poleć książkę

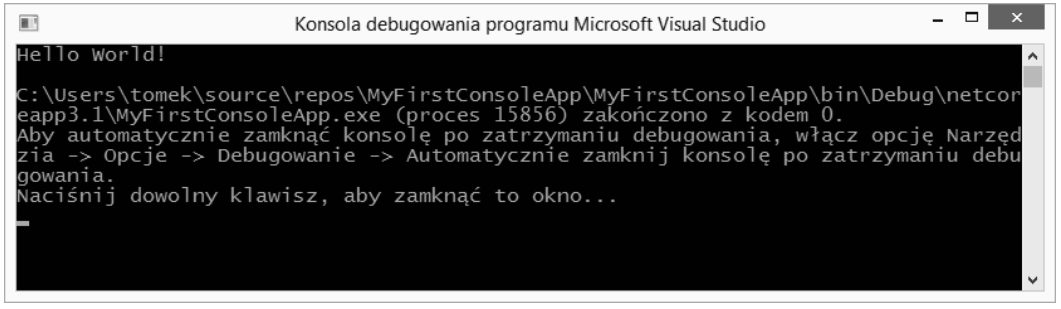
3 Uruchom nową aplikację.

Aplikacja, którą Visual Studio utworzyło za Ciebie, jest gotowa do uruchomienia. W górnej części IDE Visual Studio znajdź przycisk z zielonym trójkątem i nazwą aplikacji, po czym kliknij go:

4 Przyjrzyj się danym wyjściowym programu.

Gdy uruchomisz aplikację, pojawi się okno **Konsola debugowania programu Microsoft Visual Studio** z danymi wyjściowymi programu:

Gdy uruchomisz aplikację, ta wykona metodę Main, która wyświetla ten wiersz tekstu w konsoli.



Najlepszy sposób na naukę języka polega na pisaniu w nim dużej ilości kodu. Dlatego w tej książce będziesz pisać wiele programów. Liczne z nich będą projektami aplikacji konsolowych na platformę .NET Core, warto więc przeanalizować to, co właśnie zrobiłeś.

W górnej części okna znajdują się **dane wyjściowe programu**:

```
Hello World!
```

Dalej znajduje się pusty wiersz, a następnie dodatkowy tekst:

```
C:\<ścieżka-do-katalogu-projektu>\MyFirstConsoleApp\MyFirstConsoleApp\bin\Debug\
netcoreapp3.1\MyFirstConsoleApp.exe (proces #####) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia->
opcje->Debugowanie->Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno. . .
```

Ten sam komunikat zobaczysz w dolnej części każdego okna Konsoli debugowania. Omawiany program wyświetla wiersz tekstu (**`Hello World!`**), po czym kończy pracę. Visual Studio pozostawia okno otwarte do czasu wciśnięcia klawisza, co powoduje zamknięcie go. Dzięki temu możesz zobaczyć dane wyjściowe przed zniknięciem okna.

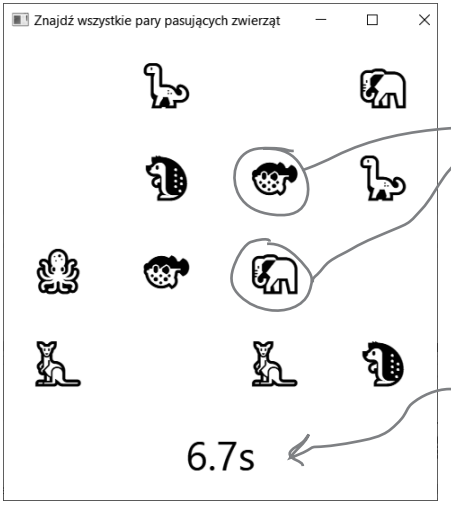
Wciśnij klawisz, aby zamknąć okno. Następnie ponownie uruchom program. W ten sposób będziesz uruchamiać wszystkie projekty aplikacji konsolowych na platformę .NET Core, jakie napiszesz w trakcie lektury tej książki.

Kup książkę Poleć książkę

Napiszmy grę!

Zbudowałeś swoją pierwszą aplikację w C# — świetnie! Teraz pora utworzyć coś trochę bardziej skomplikowanego. Teraz zbudujesz **grę w dopasowywanie zwierząt**, w której gracz widzi zestaw 16 zwierzaków i musi klikać pasujące pary, aby te zniknęły.

Gra w dopasowywanie zwierząt, jaką zbudujesz.



Gra wyświetla osiem par zwierząt rozmieszczonych losowo w oknie. Gracz klika dwa zwierzęta; jeśli te pasują do siebie, znikają z okna.

Zegar odmierza czas ukończenia gry. Celem jest dopasowanie wszystkich par w jak najkrótszym czasie.

Tworzenie projektów różnego rodzaju jest ważnym aspektem nauki języka C#. W niektórych projektach w tej książce używana jest technologia WPF (ang. *Windows Presentation Foundation*), ponieważ zapewnia narzędzia do projektowania zaawansowanych interfejsów użytkownika działających w różnych wersjach systemu Windows (nawet tak dawnych jak Windows XP).

Jednak C# jest przeznaczony nie tylko dla systemu Windows!

Używasz komputerów Mac? Masz szczęście! Specjalnie dla Ciebie dodaliśmy materiały dotyczące Visual Studio for Mac. W dodatku „Przewodnik ucznia po Visual Studio for Mac" na końcu książki znajdziesz kompletny zastępnik tego rozdziału i wersje wszystkich projektów WPF z tej książki dostosowane do komputerów Mac.

W wersjach projektów WPF dla komputerów Mac używana jest technologia ASP.NET Core. Projekty ASP.NET Core możesz budować także dla systemu Windows.

Gra w dopasowywanie zwierząt jest aplikacją WPF

Aplikacje konsolowe świetnie się sprawdzają, jeśli chcesz wprowadzać i wyświetlać sam tekst. Jeżeli potrzebujesz wizualnej aplikacji wyświetlanej w oknie, musisz zastosować inną technologię. To dlatego gra w dopasowywanie zwierząt będzie **aplikacją WPF**. WPF umożliwia tworzenie aplikacji desktopowych na różne wersje systemu Windows. W większości rozdziałów z tej książki opisywana będzie jedna aplikacja WPF. Celem tego projektu jest przedstawienie technologii WPF i zapewnienie Ci narzędzi do tworzenia atrakcyjnych wizualnie aplikacji desktopowych (obok aplikacji konsolowych).

Do czasu zakończenia pracy nad tym projektem dużo lepiej poznasz narzędzia, z których będziesz korzystać w tej książce do uczenia się i eksplorowania języka C#.

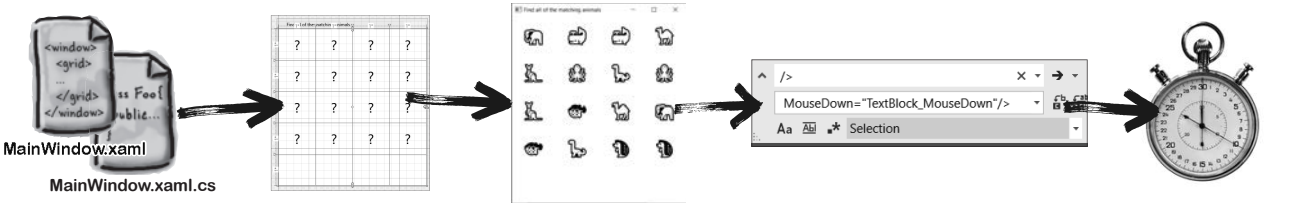
Kup książkę Poleć książkę

Proces tworzenia gry

W dalszej części rozdziału opisany jest proces budowania gry w dopasowywanie zwierząt. Proces ten jest podzielony na kilka odrębnych etapów:

1. Najpierw utworzysz nowy projekt aplikacji desktopowej w Visual Studio.
2. Następnie użyjesz języka XAML do zbudowania okna.
3. Potem napiszesz kod w C#, aby dodać do okna losowe emoji ze zwierzętami.
4. Gra wymaga od użytkownika klikania par pasujących emoji.
5. Na zakończenie trzeba dodać zegar, dzięki czemu gra będzie bardziej ekscytująca.

Realizacja tego projektu może zająć od 15 minut do godziny (w zależności od tego, jak szybko piszesz kod). Nauka daje lepsze efekty, gdy się nie spieszysz, dlatego zapewnij sobie odpowiednio dużo czasu.



Utwórz projekt → *Zaprojektuj okno* → *Napisz kod w C#* → *Dodaj obsługę kliknięć myszą* → *Dodaj zegar*

Zwróć uwagę na zamieszczone w książce ramki „Projektowanie gier... i nie tylko". Zasady projektowania gier posłużą do nauki i eksploracji ważnych koncepcji i zagadnień z obszaru programowania, które mają zastosowanie w każdym projekcie — nie tylko w grach komputerowych.

Projektowanie gier... i nie tylko

Czym jest gra?

Może się wydawać, że oczywiste jest, czym jest gra. Zastanów się jednak nad tym przez chwilę — sprawa wcale nie jest taka prosta.

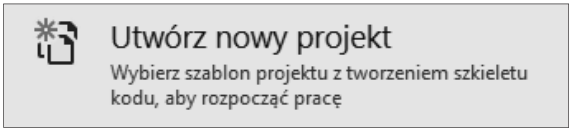
- Czy wszystkie gry mają **zwycięzcę**? Czy zawsze się kończą? Niekoniecznie. Jak to wygląda w symulatorze lotu, w grze w projektowanie parku rozrywki lub w grach takich jak The Sims?
- Czy gry zawsze są **ciekawe**? Nie dla każdego. Niektórzy gracze lubią „grind", czyli powtarzanie w kółko tych samych czynności. Inni nie cierpią takich gier.
- Czy w grach zawsze występuje **podejmowanie decyzji, konflikt lub rozwiązywanie problemów**? Nie we wszystkich. W grach typu *walking simulator* (dosłownie „symulator chodzenia") gracz jedynie eksploruje otoczenie. Często nie ma w nich żadnych zagadek ani konfliktów.
- Okazuje się, że dość trudno jest precyzyjnie określić, czym jest gra. Jeśli zajrzysz do podręczników projektowania gier, znajdziesz rozmaite definicje. Dlatego na potrzeby tej książki zdefiniujmy **znaczenie słowa „gra"** tak:

Gra jest programem, którego używanie daje (miejmy nadzieję) przynajmniej tyle przyjemności, co jego pisanie.

Kup książkę Poleć książkę

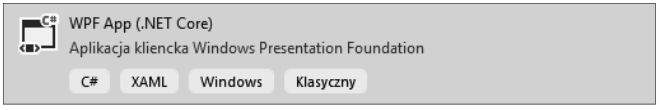
Utwórz projekt WPF w Visual Studio

Uruchom nową instancję środowiska Visual Studio 2019 i utwórz nowy projekt:

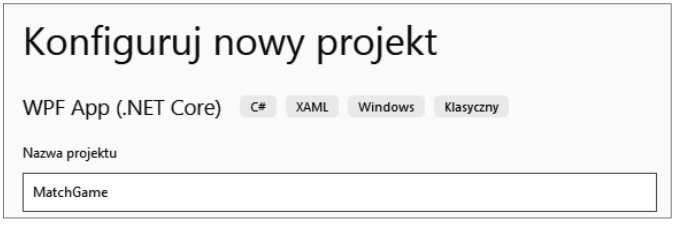


Skończyłeś już pracę nad projektem aplikacji konsolowej z pierwszej części rozdziału, dlatego możesz zamknąć instancję środowiska Visual Studio z tamtym programem.

Grę zbudujesz jako aplikację desktopową za pomocą technologii WPF. Wybierz więc **WPF App (.NET Core)** i kliknij *Dalej*:



Visual Studio wyświetli prośbę o skonfigurowanie projektu. **Jako nazwę wpisz `MatchGame`** (jeśli chcesz, możesz też zmienić lokalizację, w której projekt zostanie utworzony):



Kliknij przycisk *Utwórz*. Visual Studio utworzy nowy projekt o nazwie *MatchGame*.

Visual Studio utworzyło dla Ciebie katalog pełen plików

W momencie tworzenia nowego projektu Visual Studio dodało nowy katalog `MatchGame` i zapełniło go plikami oraz katalogami potrzebnymi w tym projekcie. Tu będziesz modyfikować dwa z tych plików: *MainWindow.xaml* i *MainWindow.xaml.cs*.

Ten plik zawiera kod w języku XAML definiujący interfejs użytkownika w głównym oknie.

MainWindow.xaml

Tu znajduje się kod w C# odpowiadający za działanie gry.

MainWindow.xaml.cs

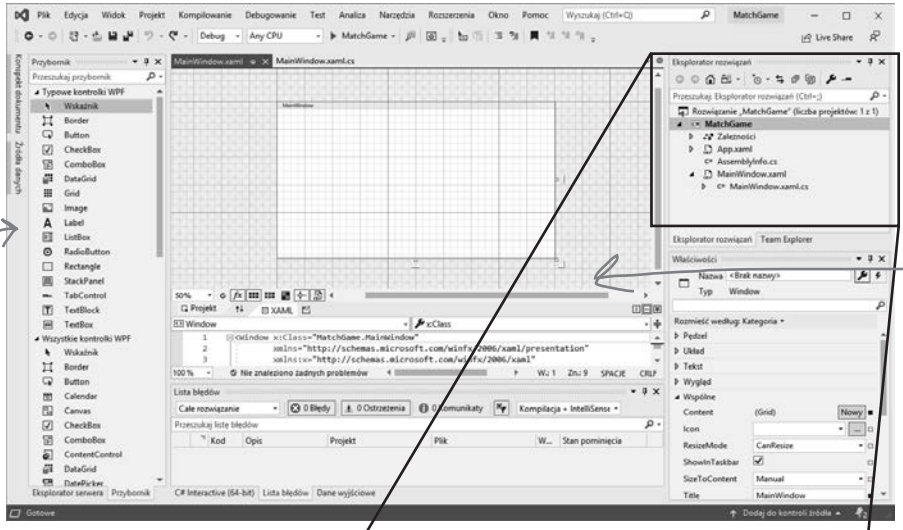
Jeśli natrafisz na problemy z tym projektem, otwórz naszą stronę w serwisie GitHub i poszukaj odsyłacza do filmu z instruktażem: https://github.com/head-first-csharp/fourth-edition.

Kup książkę Poleć książkę

Zaostrz ołówek

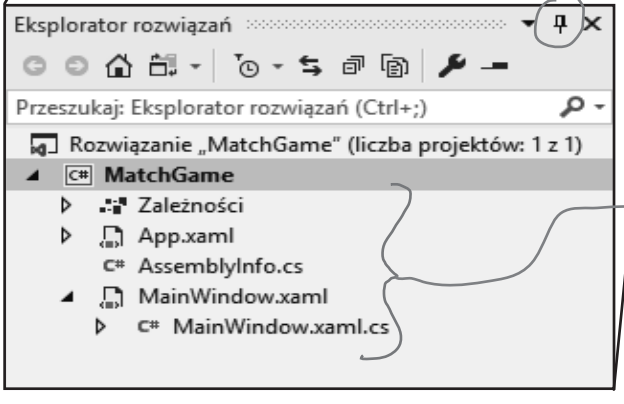
Prezentowane w książce ćwiczenia z użyciem kartki i ołówka nie są opcjonalne. Stanowią ważny aspekt nauki, praktyki i podnoszenia umiejętności z zakresu języka C#.

Dostosuj swoje środowisko IDE do zrzutu poniżej. Najpierw otwórz **plik *MainWindow.xaml***, klikając go dwukrotnie w oknie *Eksplorator Rozwiązań*. Następnie otwórz okna *Przybornik i Lista błędów*, **wybierając je w menu *Widok***. Przeznaczenie wielu okien i plików możesz określić na podstawie ich nazw i intuicji! Poświęć chwilę na **uzupełnienie pustych miejsc**. Postaraj się opisać, do czego służą poszczególne elementy IDE Visual Studio. Pierwsze pole wypełniliśmy już za Ciebie. Sprawdź, czy potrafisz odgadnąć funkcje innych komponentów.



Okno Projektant umożliwia modyfikowanie interfejsu użytkownika przez przeciąganie kontrolek.

Czy zauważyłeś, że okno Przybornik zniknęło? Kliknij ikonę pinezki, aby pozostało otwarte.



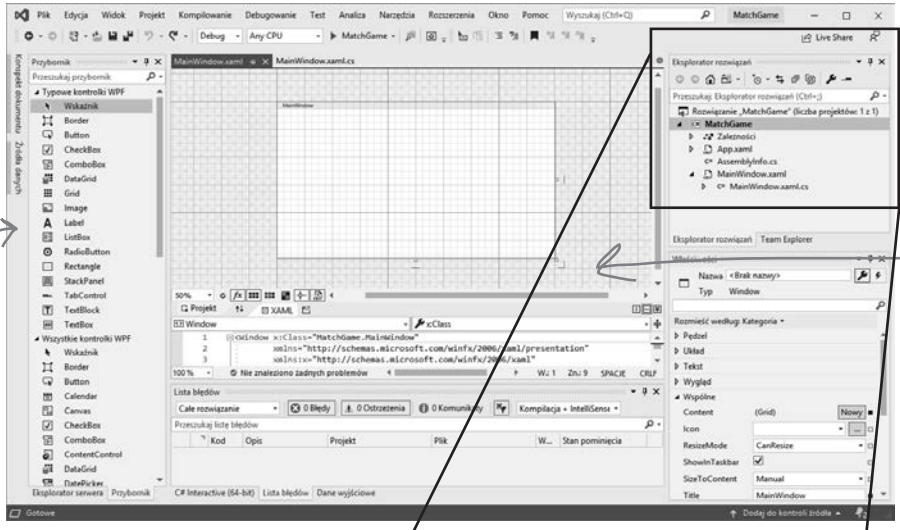
Używamy tu motywu kolorów „Jasny", aby zrzuty były bardziej czytelne. Aby zmienić motyw, wybierz opcję *Narzędzia/Opcje/Środowisko*.

Kup książkę Poleć książkę

Zaostrz ołówek
Rozwiązanie

Opisałeś działanie różnych komponentów IDE Visual Studio dla języka C#. Możliwe, że napisałeś coś innego niż my, ale mamy nadzieję, iż udało Ci się domyślić podstawowych funkcji każdego okna i komponentu w IDE. Nie martw się, jeśli Twoje odpowiedzi nieco różnią się od naszych! Będziesz mieć MNÓSTWO okazji do praktyki w używaniu IDE.

*Krótkie przypomnienie: w tej książce (także na tej stronie) określenia „Visual Studio" i „IDE" są stosowane **wymiennie**.*



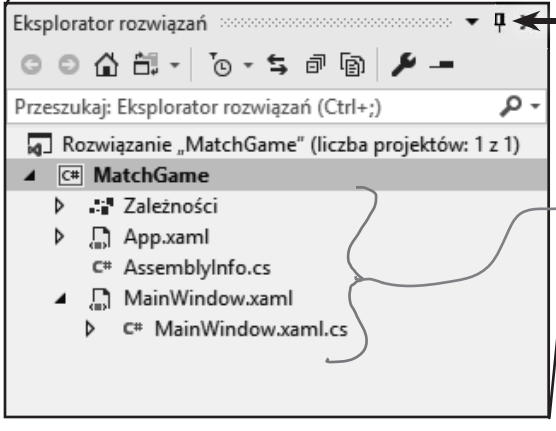
To jest Przybornik. Zawiera wiele wizualnych kontrolek, które możesz przeciągnąć do okna.

Okno Projektant umożliwia modyfikowanie interfejsu użytkownika przez przeciąganie kontrolek.

Okno Właściwości wyświetla właściwości elementu aktualnie zaznaczonego w oknie Projektant.

Okno Lista błędów informuje o błędach w kodzie. Znajdziesz tu informacje diagnostyczne na temat aplikacji.

Kliknięcie ikony pinezki powoduje włączenie lub wyłączenie automatycznego ukrywania okna. Okno Przybornik domyślnie jest automatycznie ukrywane.



Pliki C# i XAML wygenerowane przez IDE w trakcie tworzenia nowego projektu pojawiają się w Eksploratorze rozwiązań. Znajdziesz tu też inne pliki rozwiązania.

Za pomocą Eksploratora rozwiązań z IDE możesz przełączać się między plikami.

Kup książkę Poleć książkę

Nie istnieją głupie pytania

Zwróć uwagę na sekcje pytań i odpowiedzi. Często pomoże Ci ona rozwiać nurtujące Cię wątpliwości. Znajdziesz tu też kwestie interesujące inne osoby. Liczne z tych pytań zostały rzeczywiście zadane przez czytelników wcześniejszych wydań książki!

P: Skoro Visual Studio generuje za mnie cały kod, to czy nauka języka C# sprowadza się do nauki obsługi środowiska Visual Studio?

O: Nie. IDE świetnie sobie radzi z generowaniem fragmentów kodu, ale ma swoje ograniczenia. Niektóre zadania wykonuje bardzo dobrze — na przykład przygotowuje solidny punkt wyjścia dla programisty i automatycznie zmienia właściwości kontrolek w interfejsie użytkownika. Nie potrafi jednak najważniejszego — ustalić, co program ma robić, i sprawić, by to robił. Choć Visual Studio jest jednym z najbardziej zaawansowanych środowisk programistycznych, ma ograniczone możliwości. To ***Ty*** (nie IDE) musisz napisać kod, który wykonuje najważniejsze zadania.

P: Co zrobić, jeśli IDE wygeneruje kod, którego nie potrzebuję w projekcie?

O: Możesz zmodyfikować lub usunąć ten kod. IDE generuje kod zgodnie z tym, w jaki sposób przeciągnięte lub dodane elementy są zwykle używane. Jednak czasem potrzebujesz czegoś innego. Wszystko, co IDE zrobi dla Ciebie (każdy wiersz kodu, każdy dodany plik), można zmodyfikować — albo ręcznie, edytując pliki, albo za pomocą wygodnego interfejsu z IDE.

P: Dlaczego miałem zainstalować wersję Visual Studio Community? Na pewno nie będę potrzebować jednej z płatnych wersji, aby wykonać wszystkie zadania z tej książki?

O: W tej książce nie ma niczego, czego nie da się wykonać za pomocą bezpłatnej wersji Visual Studio (którą pobrałeś z witryny Microsoftu). Różnice między wersją Community i innymi edycjami nie przeszkadzają w pisaniu kodu w C# i tworzeniu w pełni funkcjonalnych, kompletnych aplikacji.

P: Pisaliście o łączeniu języków C# i XAML. Czym jest XAML i jak łączyć go z C#?

O: XAML jest **językiem znaczników** używanym do budowania interfejsów aplikacji WPF. XAML bazuje na XML-u, dlatego jeśli kiedykolwiek używałeś HTML-a, nauka będzie łatwiejsza. Oto przykładowy **znacznik** XAML generujący szarą elipsę:

```
<Ellipse Fill="Gray"
  Height="100" Width="75"/>
```

Jeśli w kodzie XAML projektu wpiszesz ten znacznik zaraz po znaczniku `<Grid>`, pośrodku okna pojawi się szara elipsa. Znaczniki rozpoznasz po znaku <, po którym następuje słowo (tu `Ellipse`). Jest to **początek znacznika**. Ten konkretny znacznik `Ellipse` ma trzy **właściwości**: jedna ustawia kolor elipsy na szary, a dwie pozostałe określają wysokość i szerokość figury. Kończy się on sekwencją />, ale niektóre znaczniki XAML mogą obejmować inne znaczniki. Możesz przekształcić go w **znacznik kontenerowy**, zastępując sekwencję /> znakiem >, dodając inne znaczniki (które mogą zawierać dodatkowe znaczniki) i zamykając go **znacznikiem końcowym**, który wygląda tak: `</Ellipse>`.

W trakcie lektury dowiesz się więcej o działaniu XAML-a i poznasz liczne znaczniki z tego języka.

P: Mój ekran wygląda inaczej niż na zrzutach! U mnie brakuje niektórych okien, a inne znajdują się w odmiennych miejscach. Czy zrobiłem coś nie tak? Jak mogę przywrócić ustawienia?

O: Jeśli wybierzesz opcję ***Resetuj układ okna*** z menu *Okno*, IDE przywróci domyślny układ okna. Następnie użyj opcji ***Widok/Inne okna***, aby otworzyć okna *Przybornik* i *Lista błędów*.

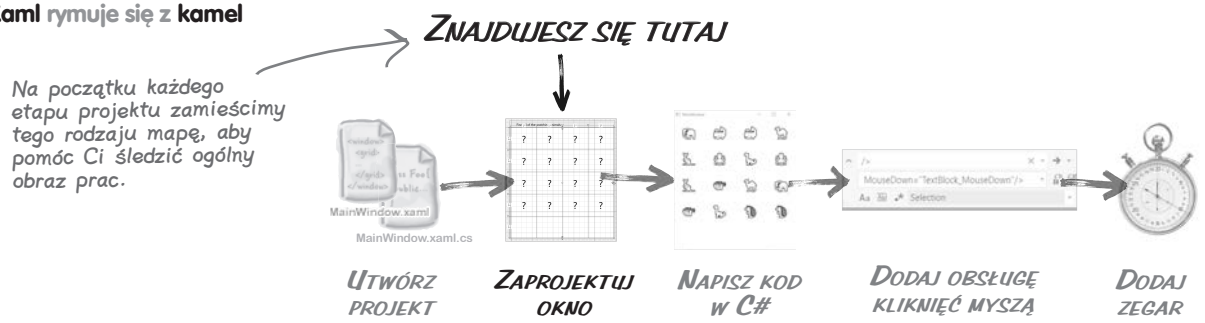
Okno Przybornik domyślnie jest zwijane. Za pomocą ikony pinezki w prawym górnym rogu Przybornika możesz spowodować, że okno to pozostanie otwarte.

Visual Studio generuje kod, który możesz wykorzystać jako punkt wyjścia do tworzenia aplikacji.

Sprawienie, by aplikacja robiła to, co ma robić, należy całkowicie do Ciebie.

Kup książkę Poleć książkę

Xaml rymuje się z kamel



Użyj języka XAML do zaprojektowania okna

Teraz, gdy Visual Studio utworzyło projekt WPF, pora zacząć pracę z językiem **XAML**.

XAML (od ang. **Extensible Application Markup Language**) to dający duże możliwości język znaczników, używany przez programistów języka C# do projektowania interfejsów użytkownika. Będziesz tworzyć aplikacje z użyciem dwóch rodzajów kodu. Przede wszystkim będziesz projektować interfejs użytkownika za pomocą języka XAML. Następnie dodasz kod w języku C# odpowiedzialny za działanie gry.

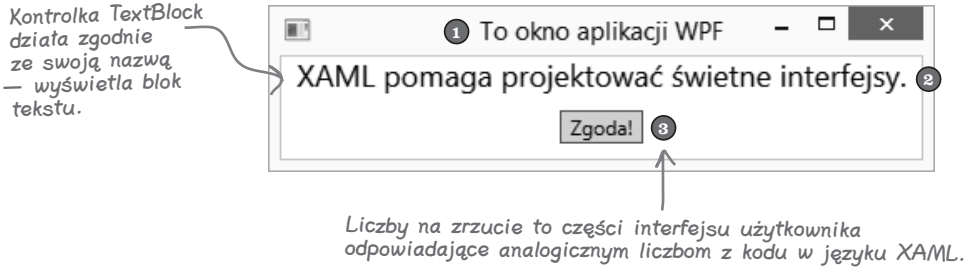
Jeśli używałeś kiedyś języka HTML do projektowania stron internetowych, dostrzeżesz dużo podobieństw w języku XAML. Oto krótki przykład ilustrujący układ okna zdefiniowany w języku XAML:

```
<Window x:Class="MyWPFApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="To okno aplikacji WPF" Height="100" Width="400"> (1)
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <TextBlock FontSize="18px" Text="XAML pomaga projektować świetne interfejsy."/> (2)
        <Button Width="50" Margin="5,10" Content="Zgoda!"/> (3)
    </StackPanel>
</Window>
```

Dodaliśmy liczby do fragmentów kodu definiujących tekst.

Poszukaj analogicznych liczb na zrzucie poniżej.

Poniżej pokazano, jak platforma WPF **renderuje** (wyświetla na ekranie) to okno. Platforma generuje okno z dwoma widocznymi **kontrolkami**: kontrolką `TextBlock`, która wyświetla tekst, i kontrolką `Button`, którą użytkownik może kliknąć. Te elementy są umieszczone w niewidocznej kontrolce `StackPanel`, która powoduje wyświetlanie kontrolek jedna pod drugą. Przyjrzyj się kontrolkom na zrzucie, a następnie wróć do kodu w języku XAML i znajdź w nim znaczniki `TextBlock` i `Button`.

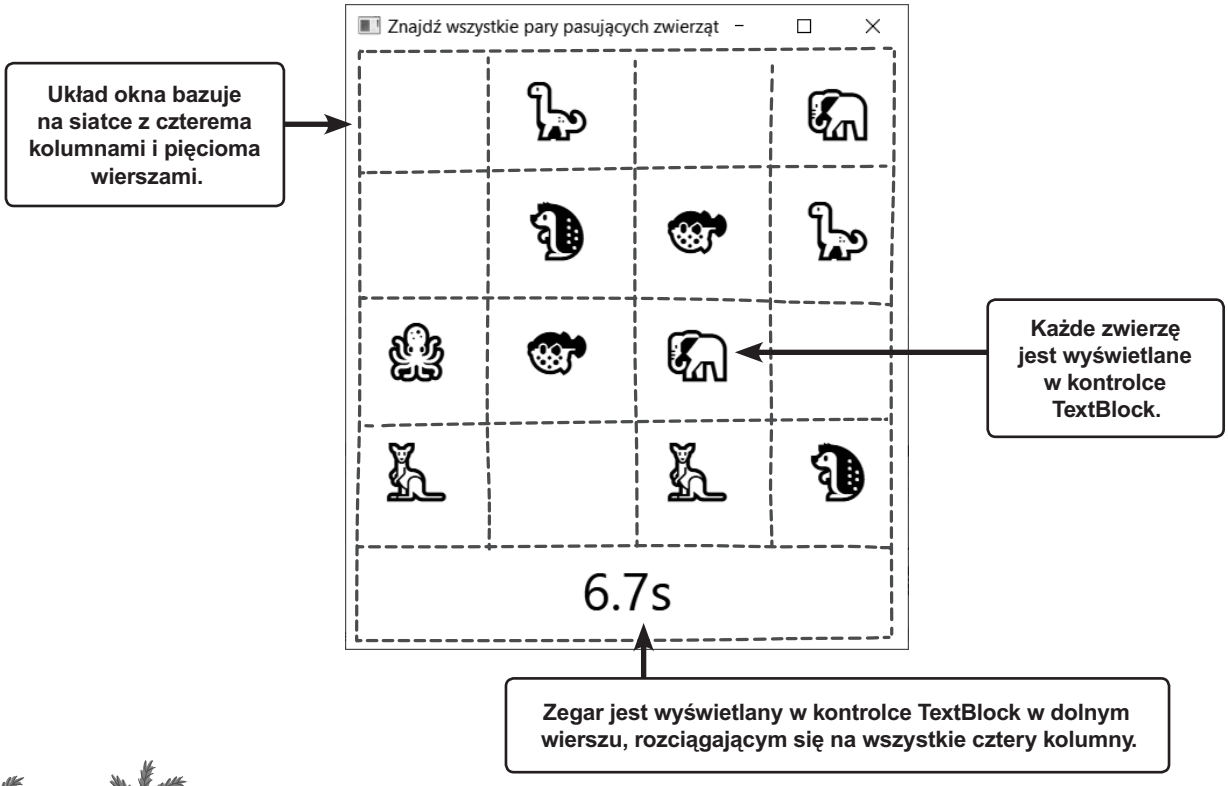


Kup książkę Poleć książkę

Zaprojektuj okno gry

Będziesz potrzebować aplikacji z graficznym interfejsem użytkownika, obiektów odpowiedzialnych za działanie gry i pliku wykonywalnego do uruchamiania. Wydaje się, że wymaga to dużo pracy, ale w pozostałej części rozdziału przygotujesz wszystkie elementy. W efekcie tego opanujesz na niezłym poziomie projektowanie atrakcyjnie wyglądających aplikacji WPF.

Oto układ okna aplikacji, jaką utworzysz:



Znajomość języka XAML jest ważna dla programistów języka C#.

Możesz sobie myśleć tak: „Chwileczkę! To książka *C#. Rusz głową*. Po co przeznaczać tyle czasu na XAML? Nie lepiej skupić się na C#?”.

W aplikacjach WPF (a także innych projektach w języku C#) do projektowania interfejsu użytkownika używany jest język XAML. XAML stosuje się nie tylko w aplikacjach desktopowych; w aplikacjach mobilnych dla systemów Android i iOS używana jest platforma Xamarin Forms, która używa odmiany języka XAML (z *odrobinę* innym zestawem kontrolek). To dlatego umiejętność tworzenia interfejsów użytkownika w języku XAML jest ważna dla każdego programisty języka C# i dlatego w tej książce dowiesz się o wiele więcej na temat XAML-a. Zobaczysz ***krok po kroku***, jak pisać kod XAML. Możesz też użyć XAML-owego *Projektanta* z Visual Studio 2019, aby przygotować interfejs użytkownika bez pisania dużej ilości kodu. ***A zatem, aby wszystko było jasne***:

XAML to język do definiowania interfejsu użytkownika. C# służy do definiowania działania programu.

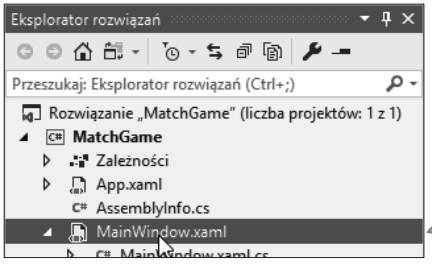
Kup książkę Poleć książkę

Ustaw wielkość okna i podaj nagłówek za pomocą właściwości w XAML-u

Zacznij tworzenie interfejsu użytkownika gry w dopasowywanie zwierząt. W pierwszej kolejności zmniejsz szerokość i nagłówek okna. Zaznajomisz się też z XAML-owym *Projektantem* — rozbudowanym narzędziem do projektowania atrakcyjnych interfejsów użytkownika swoich aplikacji.

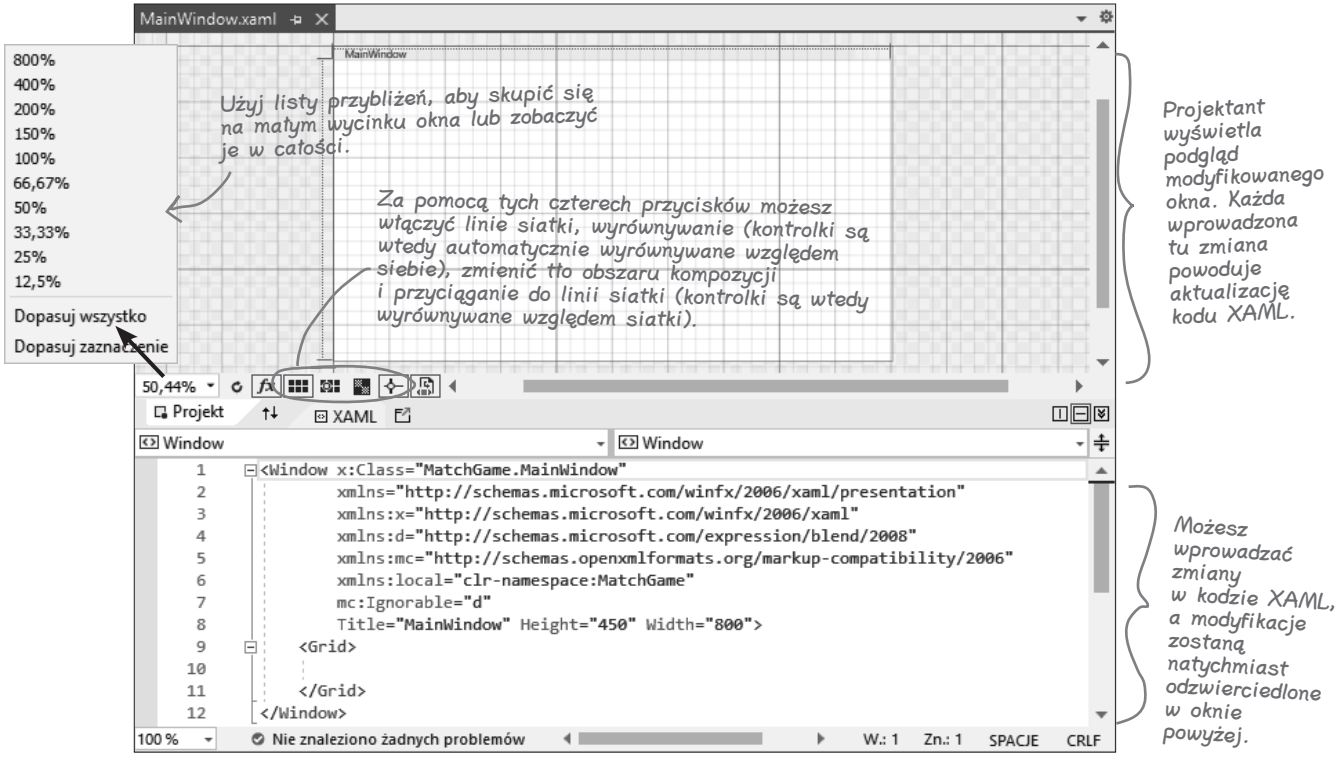
1 **Wybierz plik *MainWindow*.**

Kliknij dwukrotnie plik *MainWindow.xaml* w *Eksploratorze rozwiązań*.



Kliknij dwukrotnie plik w Eksploratorze rozwiązań, aby otworzyć go w odpowiednim edytorze. Pliki z kodem C# (rozszerzenie .cs) są otwierane w edytorze kodu. Pliki XAML (rozszerzenie .xaml) są otwierane w XAML-owym projektancie.

Gdy tylko to zrobisz, Visual Studio otworzy plik w XAML-owym *Projektancie*.



Projektant wyświetla podgląd modyfikowanego okna. Każda wprowadzona tu zmiana powoduje aktualizację kodu XAML.

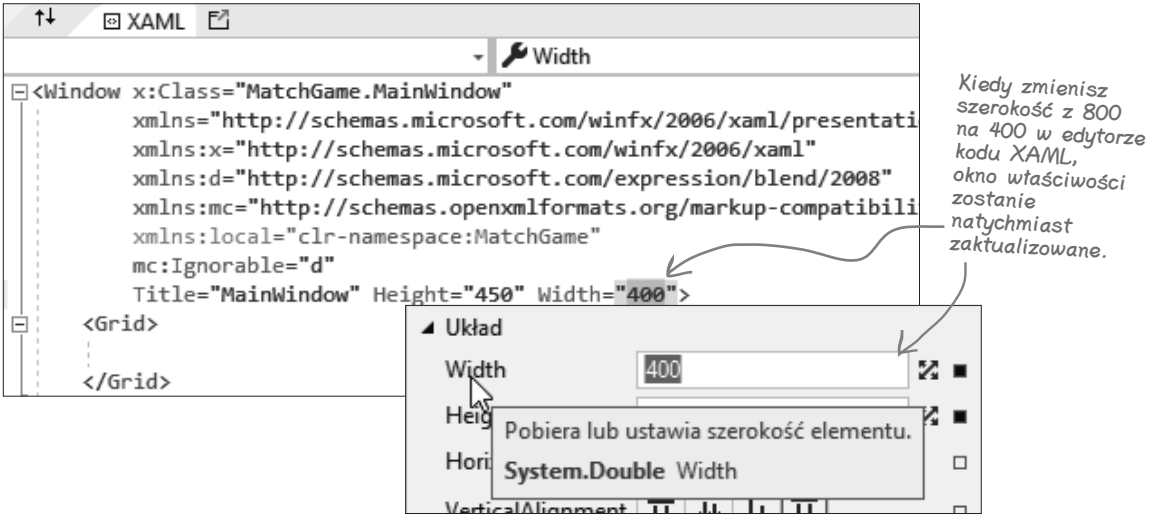
Możesz wprowadzać zmiany w kodzie XAML, a modyfikacje zostaną natychmiast odzwierciedlone w oknie powyżej.

Kup książkę Poleć książkę

2 **Zmień wielkość okna.**

Przenieś kursor do edytora kodu XAML i kliknij dowolny z pierwszych ośmiu wierszy kodu. W oknie z właściwościami powinieneś zobaczyć wtedy właściwości okna.

Rozwiń sekcję *Układ* i **zmień szerokość na 400**. Okno na panelu *Projektanta* natychmiast stanie się węższe. Spójrz na kod XAML — właściwość `Width` jest teraz równa `400`.



3 **Zmień nagłówek okna.**

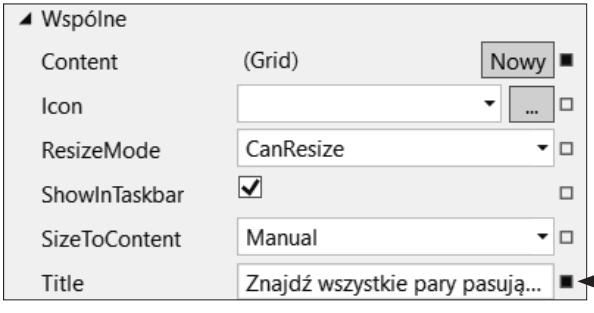
Znajdź w kodzie XAML ten wiersz (znajduje się on na końcu znacznika `Window`):

```
Title="MainWindow" Height="450" Width="400">
```

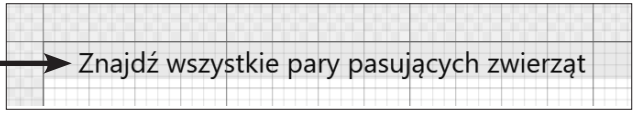
Zmień tytuł (właściwość `Title`) na **`Znajdź wszystkie pary pasujących zwierząt`**:

```
Title="Znajdź wszystkie pary pasujących zwierząt" Height="450" Width="400">
```

Zobaczysz zmianę w sekcji *Wspólne* okna *Właściwości*, jednak ważniejsze jest to, że na pasku tytułu w oknie widoczny jest nowy tekst.



Gdy modyfikujesz właściwości w znacznikach XAML, zmiany są natychmiast odzwierciedlane w oknie Właściwości. Gdy używasz okna Właściwości do zmiany interfejsu użytkownika, IDE aktualizuje kod XAML.



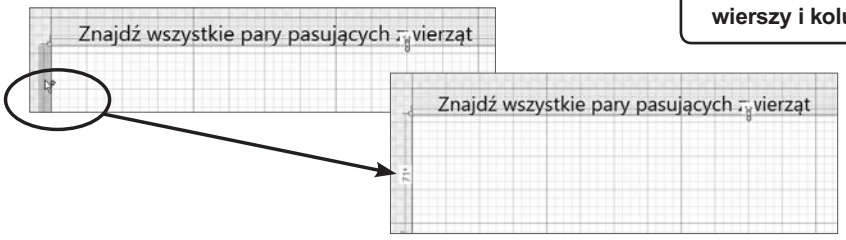
Kup książkę Poleć książkę

Dodaj wiersze i kolumny do siatki w XAML-u

Może się wydawać, że główne okno jest puste. Przyjrzyj się jednak dokładniej dolnej części kodu XAML. Czy zauważyłeś, że znajdują się tam wiersze **<Grid>** i **</Grid>**? W oknie znajduje się **siatka**, która jednak na razie jest niewidoczna, ponieważ nie obejmuje żadnych wierszy ani kolumn. Zaraz dodasz do niej wiersz.

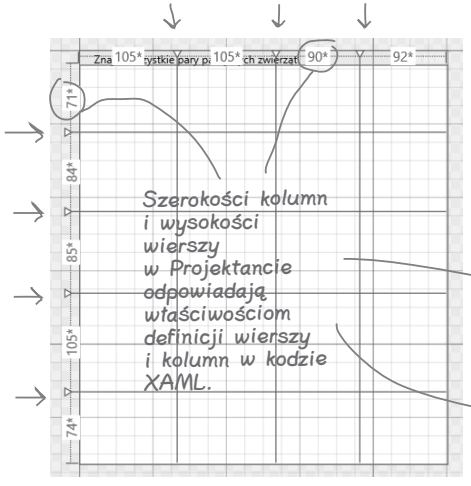
Przenieś kursor nad lewą stronę *Projektanta*. Gdy obok kursora pojawi się znak plus, kliknij myszą, aby dodać wiersz.

Interfejs użytkownika aplikacji WPF jest zbudowany z kontrolek (przycisków, etykiet, pól wyboru itd.). Siatka jest kontenerem, czyli kontrolką specjalnego rodzaju, która może zawierać inne kontrolki. Siatka pozwala użyć wierszy i kolumn do zdefiniowania układu okna.



Zobaczysz liczbę z gwiazdką i poziomą linię w oknie. Właśnie dodałeś wiersz do siatki! Teraz dodaj wiersze i kolumny:

- ★ Wykonaj wcześniejszą operację cztery razy, aby uzyskać w sumie pięć wierszy.
- ★ Umieść kursor w górnej części okna i kliknij, aby dodać cztery kolumny. Okno powinno wyglądać tak jak na zrzucie poniżej (widoczne liczby będą zapewne inne — nie stanowi to problemu).
- ★ Wróć do kodu XAML. Znajdują się w nim teraz znaczniki `ColumnDefinition` i `RowDefinition` odpowiadające dodanym wierszom i kolumnom.




```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="105*"/>
    <ColumnDefinition Width="105*"/>
    <ColumnDefinition Width="90*"/>
    <ColumnDefinition Width="92*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="71*"/>
    <RowDefinition Height="84*"/>
    <RowDefinition Height="85*"/>
    <RowDefinition Height="105*"/>
    <RowDefinition Height="74*"/>
</Grid.RowDefinitions>
```

Ramki „Obejrzyj to!" zwracają uwagę na ważne, ale często niejasne kwestie, które mogą utrudnić lub spowolnić pracę.

Obejrzyj to!

W Twoim IDE ekran może wyglądać nieco inaczej.

*Wszystkie zrzuty z tej książki zostały wykonane w środowisku **Visual Studio Community 2019 dla systemu Windows**. Jeśli używasz wersji Professional lub Enterprise, możesz dostrzec drobne różnice.*

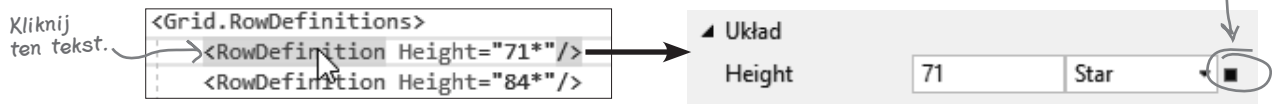
Nie przejmuj się — niezależnie od wersji, program będzie działał tak samo.

Kup książkę Poleć książkę

Wyrównaj wielkości wierszy i kolumn

Gdy gra wyświetla graczowi zwierzęta do dopasowania, rysunki powinny być równo rozmieszczone. Każde zwierzę będzie znajdować się w komórce siatki, a sama siatka będzie automatycznie dostosowywana do wielkości okna. Dlatego wiersze i kolumny powinny być równej wielkości. Na szczęście XAML umożliwia łatwą zmianę wielkości wierszy i kolumn. **Kliknij pierwszy znacznik `RowDefinition` w edytorze kodu XAML**, aby wyświetlić jego ustawienia w oknie *Właściwości*:

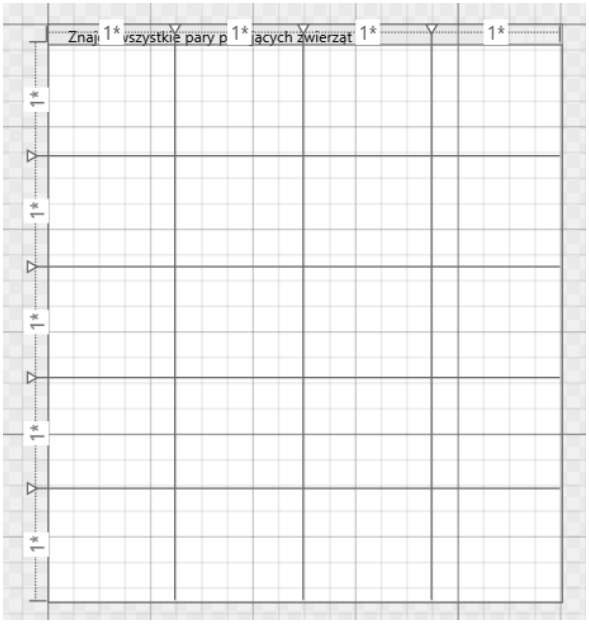
Gdy ten kwadracik jest wypełniony, wartość właściwości jest inna od domyślnej. Aby przywrócić ustawienia domyślne, kliknij kwadracik i wybierz w menu opcję Resetuj.



Przejdź do okna *Właściwości* i **kliknij kwadracik** na prawo od właściwości *Height*, po czym **wybierz opcję *Resetuj* z menu**, które się pojawi. Chwileczkę! Powoduje to zniknięcie wiersza w *Projektancie*! No cóż, tak naprawdę wiersz nie zniknął, stał się tylko bardzo niski. **Zresetuj właściwość *Height*** wszystkich wierszy. Następnie **zresetuj właściwość *Width*** wszystkich kolumn. Siatka powinna teraz mieć cztery kolumny i pięć wierszy o równej wielkości.

Naprawdę postaraj się przeczytać kod XAML. Jeśli nie pracowałeś wcześniej w językach HTML lub XML, początkowo kod może wydawać się serią niezrozumiałych <nawiasów> i /ukośników. Jednak im więcej będziesz mieć z nim do czynienia, tym bardziej będzie dla Ciebie zrozumiały.

Oto co powinieneś zobaczyć w Projektancie:



A to powinieneś zobaczyć w edytorze kodu XAML między otwierającym znacznikiem `<Window...>` i zamykającym znacznikiem `</Window>`:

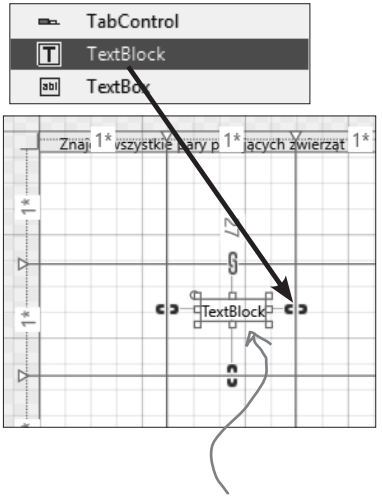
```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
</Grid>
```

To kod XAML tworzący siatkę z czterema równymi kolumnami i pięcioma równymi wierszami.

Kup książkę Poleć książkę

Dodaj kontrolki TextBlock do siatki

WPF używa **kontrolek `TextBlock`** do wyświetlania tekstu. My użyjemy ich do wyświetlania dopasowywanych zwierząt. Dodaj taką kontrolkę do okna.



Gdy przeciągniesz kontrolkę TextBlock z Przybornika do komórki, IDE doda tę kontrolkę do kodu XAML i ustawi dla niej wiersz, kolumnę oraz margines.

Rozwiń sekcję *Typowe kontrolki WPF* w *Przyborniku* i **przeciągnij kontrolkę `TextBlock` do komórki w drugiej kolumnie drugiego wiersza**. IDE doda znacznik `TextBlock` między początkowym i końcowym znacznikiem `Grid`:

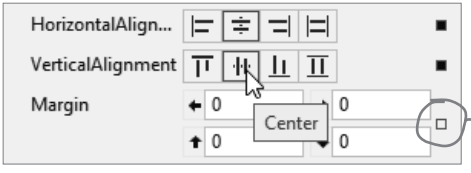
```
<TextBlock Text="TextBlock"
    HorizontalAlignment="Left" VerticalAlignment="Center"
    Margin="560,0,0,0" TextWrapping="Wrap" />
```

W kodzie XAML tej kontrolki `TextBlock` jest pięć właściwości:

- ★ `Text` określa tekst wyświetlany w oknie.
- ★ `HorizontalAlignment` wyrównuje tekst do lewej, do prawej lub względem środka.
- ★ `VerticalAlignment` wyrównuje tekst względem góry, środka lub dołu.
- ★ `Margin` ustawia margines względem krawędzi kontenera.
- ★ `TextWrapping` określa, czy tekst w kontrolce ma być zawijany.

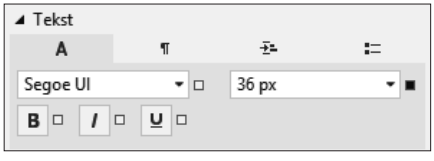
U Ciebie właściwości mogą być podane w innej kolejności, a właściwość *Margin* może mieć inną wartość, ponieważ zależy ona od miejsca w komórce, w które przeciągnąłeś kontrolkę. Wszystkie te właściwości można zmieniać i resetować w oknie *Właściwości* w IDE.

Każde zwierzę ma być wyśrodkowane. **Kliknij etykietę** w *Projektancie*, a następnie w oknie *Właściwości* kliknij Układ, aby rozwinąć **sekcję *Układ***. Kliknij ***Center*** dla obu właściwości, *HorizontalAlignment* i *VerticalAlignment*, a następnie użyj kwadracika po prawej stronie, aby **zresetować właściwość *Margin***.

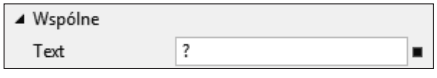


Kliknij kwadracik i wybierz opcję Resetuj, aby zresetować marginesy.

Ponadto zwierzęta mają być większe, dlatego **rozwiń sekcję *Tekst*** w oknie *Właściwości* i **zmień wielkość czcionki** na `36 px`. Następnie otwórz sekcję *Wspólne* i zmień właściwość *Text* na `?`, aby wyświetlany był znak zapytania.



Właściwość Text (w sekcji Wspólne) określa tekst kontrolki TextBlock.



Kliknij pole wyszukiwania w górnej części okna *Właściwości* i wpisz słowo `wrap`, aby znaleźć pasujące właściwości. Użyj kwadracika po prawej stronie, aby zresetować właściwość *TextWrapping*.

Kup książkę Poleć książkę

Nie istnieją głupie pytania

P: Po zresetowaniu wysokości pierwszych czterech wierszy wszystkie one znikają. Jednak kiedy zresetuję wysokość ostatniego, wszystkie wiersze znów są widoczne. Dlaczego tak się dzieje?

O: Wiersze wyglądały tak, jakby zniknęły, ponieważ w siatce WPF domyślnie używane jest **proporcjonalne ustawianie wielkości** wierszy i kolumn. Jeśli wysokość ostatniego wiersza wynosi 74*, to po ustawieniu domyślnej wysokości pierwszych czterech wierszy na 1* siatka zmienia ich wielkość na 1/78 (1,3%) wysokości siatki. Ostatni wiersz zajmuje wtedy 74/78 (94,8%) wysokości, dlatego pierwsze cztery wiersze wyglądają na bardzo niskie. Gdy ustawisz dla ostatniego wiersza wysokość domyślną 1*, siatka zmieni wielkość każdego wiersza na 20% wysokości siatki.

P: Gdy ustawiam szerokość okna na 400, co oznacza ta wartość? Jak szerokie jest to 400?

O: WPF używa **pikseli niezależnych od urządzenia**, zawsze równych 1/96 cala. To oznacza, że 96 pikseli na *nieprzeskalowanym* wyświetlaczu zawsze jest równe 1 cal. Jeśli jednak zmierzysz okno linijką, może się okazać, że nie zajmuje dokładnie 400 pikseli (4,16 cala). To dlatego, że Windows używa przydatnego mechanizmu zmieniającego skalę elementów na ekranie. Dzięki temu aplikacje nie są malutkie nawet wtedy, gdy wyświetlasz je, używając telewizora po drugiej stronie pokoju jako monitora komputera. Piksele niezależne od urządzenia pomagają platformie WPF zapewnić dobry wygląd aplikacji w każdej skali.

W książce natrafisz na wiele ćwiczeń tego rodzaju. Umożliwiają Ci one rozwój umiejętności programistycznych. Zawsze dopuszczalne jest podejrzenie rozwiązania!

Ćwiczenie

Masz już jedną kontrolkę `TextBlock` — to dobry punkt wyjścia! Jednak potrzebnych jest 16 takich kontrolek, aby wyświetlić wszystkie dopasowywane zwierzęta. Czy wiesz, jak dodać kod XAML, aby umieścić identyczne kontrolki XAML w komórkach z pierwszych czterech wierszy siatki?

Zacznij od przyjrzenia się znacznikowi XAML, jaki utworzyłeś. Powinien wyglądać podobnie do poniższego. Właściwości mogą być podane w innej kolejności, a tu rozdzieliliśmy je na dwa wiersze (też możesz to zrobić, jeśli chcesz, aby kod XAML był czytelniejszy):

```
<TextBlock Text="?" Grid.Column="1" Grid.Row="1" FontSize="36"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

Twoje zadanie polega na **skopiowaniu tej kontrolki `TextBlock`**, aby każda z pierwszych 16 komórek siatki zawierała identyczną kontrolkę. Aby ukończyć ćwiczenie, musisz więc *dodać do aplikacji jeszcze 15 kontrolek TextBlock*. Pamiętaj o kilku kwestiach:

- Wiersze i kolumny są numerowane od 0, co jest też wartością domyślną. Dlatego jeśli pominiesz właściwości `Grid.Row` lub `Grid.Column`, kontrolka `TextBlock` pojawi się w pierwszym od lewej wierszu lub w górnej kolumnie.
- Interfejs użytkownika możesz edytować w *Projektancie*. Możesz też skopiować i wkleić kod XAML. Wypróbuj obie te techniki i zobacz, która jest dla Ciebie najlepsza.

Kup książkę Poleć książkę

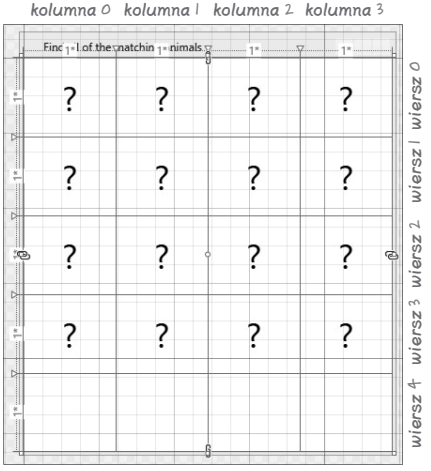
Ćwiczenie

Oto kod XAML 16 kontrolek TextBlock na dopasowywane zwierzęta. Wszystkie te kontrolki są prawie identyczne. Różnią się tylko właściwościami `Grid.Row` i `Grid.Column`, które pozwalają umieścić po jednej kontrolce `TextBlock` w każdej z 16 pierwszych komórek siatki o wymiarach 5 na 4.
Znacznik `Window` się nie zmienił, dlatego został pominięty w tym miejscu.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  <RowDefinition/>
</Grid.RowDefinitions>
```

Tak wygląda definicje kolumn i wierszy po zrównaniu wielkości tych elementów.

Tak wygląda okno w Projektancie w Visual Studio po dodaniu wszystkich kontrolek TextBlock.




```
<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="1"/>
<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="2"/>
<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="3"/>

<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="1"/>
<TextBlock Text="?" FontSize="36" Grid.Row="1" Grid.Column="1"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Text="?" FontSize="36" Grid.Row="1" Grid.Column="2"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Text="?" FontSize="36" Grid.Row="1" Grid.Column="3"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

W tych czterech kontrolkach TextBlock właściwość Grid.Row ma wartość 1, ponieważ znajdują się w drugim wierszu od góry (pierwszy wiersz ma numer 0).

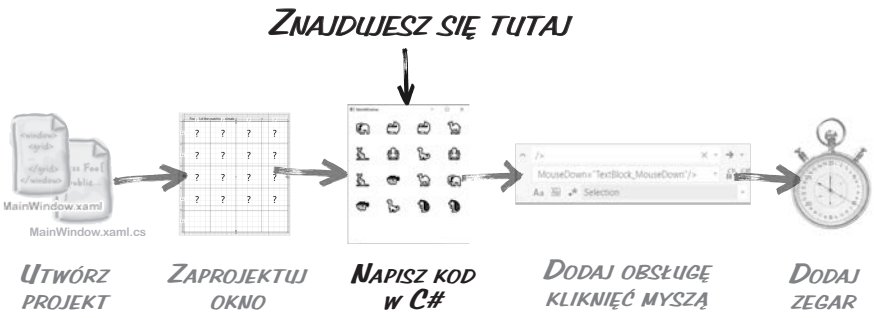
```
<TextBlock Text="?" FontSize="36" Grid.Row="2" HorizontalAlignment="Center" VerticalAlignment="Center"/>

<TextBlock Text="?" FontSize="36" Grid.Row="2" Grid.Column="1"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Text="?" FontSize="36" Grid.Row="2" Grid.Column="2"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
<TextBlock Text="?" FontSize="36" Grid.Row="2" Grid.Column="3"
    HorizontalAlignment="Center" VerticalAlignment="Center"/>
```

Dopuszczalne jest dodanie właściwości Grid.Row lub Grid.Column o wartości 0. Tu pominęliśmy takie właściwości, ponieważ 0 jest wartością domyślną.

```
  <TextBlock Text="?" FontSize="36" Grid.Row="3" HorizontalAlignment="Center" VerticalAlignment="Center"/>
  <TextBlock Text="?" FontSize="36" Grid.Row="3" Grid.Column="1"
      HorizontalAlignment="Center" VerticalAlignment="Center"/>
  <TextBlock Text="?" FontSize="36" Grid.Row="3" Grid.Column="2"
      HorizontalAlignment="Center" VerticalAlignment="Center"/>
  <TextBlock Text="?" FontSize="36" Grid.Row="3" Grid.Column="3"
      HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Grid>
```

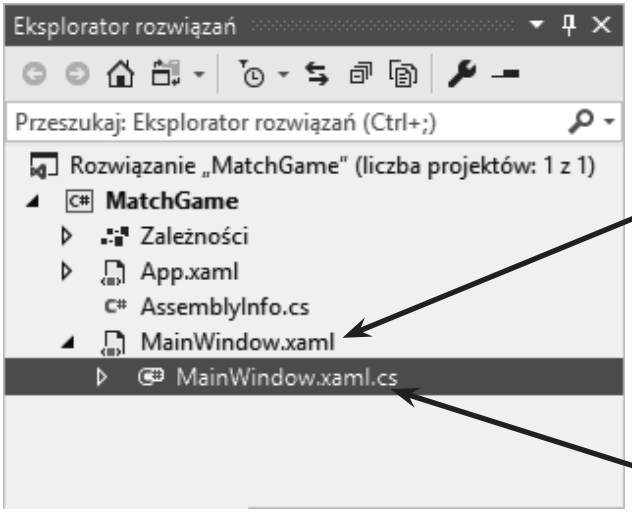
Kodu jest dużo, jednak jest to powtórzony 16 razy ten sam wiersz z drobnymi zmianami. Każdy wiersz zaczynający się od `<TextBlock` ma te same cztery właściwości (`Text`, `FontSize`, `HorizontalAlignment` i `VerticalAlignment`). Wiersze różnią się tylko właściwościami `Grid.Row` i `Grid.Column`. Właściwości mogą być podawane w dowolnej kolejności.

Kup książkę Poleć książkę



Teraz możesz zacząć pisać kod gry

Zakończyłeś projektowanie głównego okna, a przynajmniej wystarczającej jego części, aby przejść do następnego aspektu działania gry. Pora dodać kod C# odpowiedzialny za przebieg rozgrywki.



Edytowałeś kod XAML pliku *MainWindow.xaml*. To tam znajdują się wszystkie elementy projektowe okna. Kod XAML z tego pliku definiuje wygląd i układ okna.

Teraz zaczniesz pracę nad kodem C#. Znajdzie się on w pliku *MainWindow.xaml.cs*. Jest to kod zaplecza (ang. *code behind*) dla okna, ponieważ działa razem ze znacznikami z pliku XAML. To dlatego nazwy plików są prawie identyczne (różnią się tylko dodatkowym członem *.cs* na końcu). Do tego pliku dodasz kod C# definiujący działanie gry, w tym kod umieszczający emoji w siatce, obsługujący kliknięcia myszą i kontrolujący zegar.

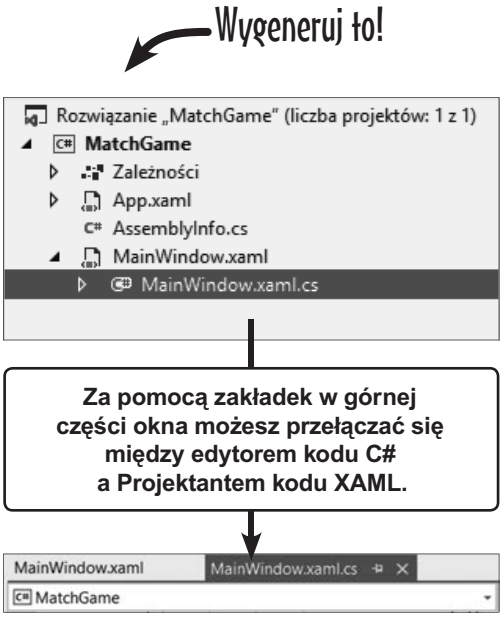
Obejrzyj to!

Gdy wpisujesz kod C#, musi on być w pełni poprawny.

*Część osób uważa, że programistą tak naprawdę stajesz się dopiero wtedy, gdy pierwszy raz godzinami szukasz błędnie umieszczonej kropki. Wielkość liter też jest ważna: `SetUpGame` to nie to samo co `setUpGame`. Nadmiarowe przecinki, średniki, nawiasy itd. mogą sprawić, że kod przestanie działać, lub — co jeszcze gorsze — wpłynąć na kod tak, że wprawdzie się skompiluje, ale będzie funkcjonować niezgodnie z oczekiwaniami. **Wykorzystujący sztuczną inteligencję mechanizm IntelliSense** z IDE pomaga unikać takich problemów, ale nie wykona za Ciebie całej pracy.*

Kup książkę Poleć książkę

Wygeneruj metodę konfigurującą grę



Teraz, po przygotowaniu interfejsu użytkownika, pora zacząć pisać kod gry. W tym celu **wygenerujesz metodę** (taką jak metoda `Main`, którą poznałeś wcześniej), a następnie dodasz do niej kod.

1 **Otwórz plik MainWindow.xaml.cs w edytorze.**

Kliknij trójkąt (▷) obok nazwy *MainWindow.xaml* w *Eksploratorze rozwiązań*, a następnie **kliknij dwukrotnie plik *MainWindow.xaml.cs***, aby otworzyć go w edytorze kodu w IDE. Zauważ, że plik zawiera już kod. Visual Studio pomoże Ci dodać do niego metodę.

To nie problem, jeśli na razie nie masz pewności, czym jest metoda.

2 **Wygeneruj metodę o nazwie SetUpGame.**

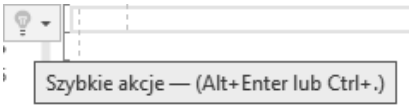
Znajdź następujący fragment w otwartym kodzie:

```
public MainWindow();
{
    InitializeComponent();
}
```

Kliknij na końcu wiersza `InitializeComponent();`, aby kursor znalazł się zaraz po średniku. Wciśnij dwukrotnie *Enter*, a następnie wpisz: `SetUpGame();`

Bezpośrednio po wpisaniu średnika nazwa `SetUpGame` zostanie podkreślona falowaną linią. Kliknij tekst `SetUpGame`. W oknie powinna pojawić się ikona żółtej żarówki. Kliknij ją, aby **otworzyć menu szybkich akcji**, i użyj go do wygenerowania metody.

Kup książkę Poleć książkę



Ikona żółtej żarówki informuje o tym, że zaznaczyłeś kod, dla którego dostępna jest szybka akcja. Oznacza to, że istnieje zadanie, które Visual Studio może zautomatyzować. Możesz kliknąć żółtą żarówkę lub wcisnąć kombinację *Alt+Enter* albo *Ctrl+.* (kropka), aby zobaczyć dostępne akcje.

3 Spróbuj uruchomić kod.

Kliknij przycisk w górnej części środowiska IDE, aby uruchomić program — tak jak wcześniej zrobiłeś to z aplikacją konsolową.

Przycisk „Rozpocznij debugowanie" na pasku narzędzi w górnej części IDE uruchamia aplikację. Możesz też wybrać opcję Rozpocznij debugowanie (F5) w menu Debugowanie.

Hmm — coś poszło nie tak. Zamiast wyświetlić okno, **program zgłosił wyjątek**:

```
21  public partial class MainWindow : Window
22  {
        Odwołania: 0
23      public MainWindow()
24      {
25          InitializeComponent();
26          SetUpGame();
27      }
28
        1 odwołanie
29      private void SetUpGame()
30      {
31          throw new NotImplementedException();
32      }
33  }
34  }
```

Wyjątek obsłużony przez użytkownika

System.NotImplementedException: „The method or operation is not implemented."

Wyświetl szczegóły | Kopiuj szczegóły | Rozpocznij sesję rozszerzenia Live Share

Ustawienia wyjątków

Może się wydawać, że program jest uszkodzony, ale w rzeczywistości można było oczekiwać takiego efektu! IDE wstrzymało program i wyróżniło ostatni uruchomiony wiersz kodu. Przyjrzyj się następującej instrukcji:

```
throw new NotImplementedException();
```

Metoda wygenerowana przez IDE dosłownie nakazuje językowi C# zgłosić wyjątek. Spójrz na komunikat powiązany z wyjątkiem:

```
System.NotImplementedException: "The method or operation is not implemented."
```

Ma to sens, ponieważ **to Ty musisz zaimplementować metodę** wygenerowaną przez IDE. Jeśli zapomniałeś to zrobić, wyjątek jest uprzejmym przypomnieniem, że masz jeszcze kod do napisania. Jeżeli generujesz dużo metod, dobrze jest mieć takie przypomnienie!

Kliknij kwadrat *Zatrzymaj debugowanie* ( ) na pasku narzędzi (lub wybierz opcję *Zatrzymaj debugowanie* w menu *Debugowanie*), aby zatrzymać program, co pozwoli Ci zakończyć implementowanie metody `SetUpGame`.

Gdy używasz IDE do uruchamiania aplikacji, przycisk Zatrzymaj debugowanie powoduje natychmiastowe jej zamknięcie.

Kup książkę Poleć książkę

Dokończ metodę SetUpGame

Jest to specjalna metoda nazywana konstruktorem. Więcej o działaniu takich metod dowiesz się z rozdziału 5.

Metoda `SetUpGame` znajduje się w metodzie `public MainWindow()`, ponieważ cały kod z metody `MainWindow` jest wykonywany bezpośrednio po uruchomieniu aplikacji.

Wkrótce dowiesz się dużo więcej o metodach.

Właśnie użyłeś IDE, aby pomogło Ci dodać metodę do aplikacji. Nie przejmuj się, jeśli nadal nie masz pewności, czym jest metoda. Znacznie więcej o metodach i strukturze kodu w języku C# dowiesz się z następnego rozdziału.

1 Zacznij dodawać kod do metody SetUpGame.

Metoda `SetUpGame` ma losowo przypisywać osiem par emoji przedstawiających zwierzęta do kontrolek `TextBlock`, aby gracz mógł dopasowywać rysunki. Dlatego metoda potrzebuje listy emoji, a w napisaniu kodu pomoże Ci IDE. Zaznacz instrukcję `throw` dodaną przez IDE i usuń ją. Następnie umieść kursor w miejscu, gdzie znajdowała się ta instrukcja, i wpisz `List`. IDE wyświetli **okno mechanizmu IntelliSense** z zestawem słów kluczowych rozpoczynających się członem `List`:

```
private void SetUpGame()
{
    List
}
```

List
List<>
ListBox
ListBoxItem

Wybierz pozycję `List` z listy IntelliSense. Następnie wpisz `<str`.
Pojawi się nowe okno IntelliSense z pasującymi słowami kluczowymi:

```
private void SetUpGame()
{
    List<str
}
```

List<T>
Represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.
T: The type of elements in the list.

string — Słowo kluczowe string
struct
StreamGeometry
StreamGeometryContext
Stretch

Wybierz `string`. Dokończ wpisywanie poniższego wiersza kodu, ale **jeszcze nie wciskaj *Enter***:

```
List<string> animalEmoji = new List<string>()
```

Lista jest kolekcją, która przechowuje zestaw wartości w określonym porządku. Więcej o kolekcjach dowiesz się z rozdziałów 8. i 9.

Słowo kluczowe „new" służy do tworzenia kolekcji List. Więcej o tym słowie dowiesz się w rozdziale 3.

Kup książkę Poleć książkę

② Dodawanie wartości do kolekcji List.

Instrukcja w C# nie jest jeszcze gotowa. Upewnij się, że kursor znajduje się zaraz po nawiasie) kończącym wiersz, a następnie wpisz otwierający nawias klamrowy {. IDE doda nawias zamykający za Ciebie i umieści kursor pomiędzy nawiasami. **Wciśnij Enter**. IDE automatycznie doda znaki podziału wiersza.

```
List<string> animalEmoji = new List<string>()
{

}
```

Gdy otwarty jest panel emoji, możesz wpisać słowo takie jak „ośmiornica", a zostanie ono zastąpione odpowiednim emoji.

Użyj **panelu emoji systemu Windows** (klawisz systemu *Windows+kropka*) lub przejdź do ulubionej witryny z emoji (na przykład *https://emojipedia.org/nature*) i skopiuj jedno emoji. Wróć do kodu, dodaj symbol ", następnie wklej emoji, wpisz kolejny symbol ", przecinek, spację i następny symbol ", jeszcze raz wklej to samo emoji, po czym wprowadź ostatni symbol " i przecinek. Następnie zrób to samo dla siedmiu kolejnych emoji. Celem jest uzyskanie **w nawiasie klamrowym ośmiu par emoji ze zwierzętami**. Dodaj **;** po zamykającym nawiasie klamrowym:

```
List<string> animalEmoji = new List<string>()
{
    "🐵","🐵",
    "🐶","🐶",
    "🐺","🐺",
    "🐱","🐱",
    "🦁","🦁",
    "🐯","🐯",
    "🦒","🦒",
    "🦊","🦊",
};
```

Umieść kursor nad kropkami pod nazwą animalEmoji. IDE poinformuje, że przypisana wartość nigdy nie jest używana. Ostrzeżenie zniknie zaraz po tym, jak użyjesz listy emoji w dalszym kodzie metody.

Panel emoji jest wbudowany w system Windows 10. Aby wyświetlić ten panel, wystarczy wcisnąć jednocześnie klawisz Windows i kropkę.

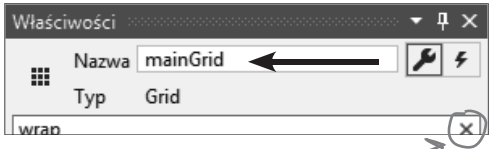
③ Ukończ metodę.

Teraz dodaj do metody **pozostały kod**. *Uważaj* na kropki, nawiasy zwykłe i klamrowe.

```
Random random = new Random();
```

Ten wiersz powinien znajdować się zaraz po zamykającym nawiasie klamrowym i średniku.

```
foreach (TextBlock textBlock in mainGrid.Children.OfType<TextBlock>())
{
    int index = random.Next(animalEmoji.Count);
    string nextEmoji = animalEmoji[index];
    textBlock.Text = nextEmoji;
    animalEmoji.RemoveAt(index);
}
```



Nie zapomnij wyzerować pola wyszukiwania.

Czerwona falowana linia pod nazwą `mainGrid` w IDE informuje o błędzie. Program nie skompiluje się, ponieważ w kodzie nie ma niczego o tej nazwie. **Wróć do edytora XAML** i kliknij znacznik `<Grid>`, następnie przejdź do okna *Właściwości* i wpisz `mainGrid` w polu *Name*.

Sprawdź kod XAML. Na początku kodu siatki zobaczysz znacznik `<Grid x:Name="mainGrid">`. Teraz w kodzie nie powinno być żadnych błędów. Jeśli są, ***starannie sprawdź każdy wiersz*** — łatwo jest coś przeoczyć.

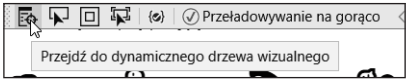
Jeśli po uruchomieniu gry wystąpi wyjątek, upewnij się, że na liście animalEmoji znajduje się dokładnie osiem par emoji, a w kodzie XAML jest 16 znaczników <TextBlock .../>.

Kup książkę Poleć książkę

Uruchom program

Kliknij przycisk ▶ Start na pasku narzędzi IDE, aby uruchomić program. Pojawi się okno z ośmioma parami zwierząt w losowych lokalizacjach:

Gdy pierwszy raz uruchomisz program, zobaczysz narzędzia środowiska uruchomieniowego w górnej części okna:

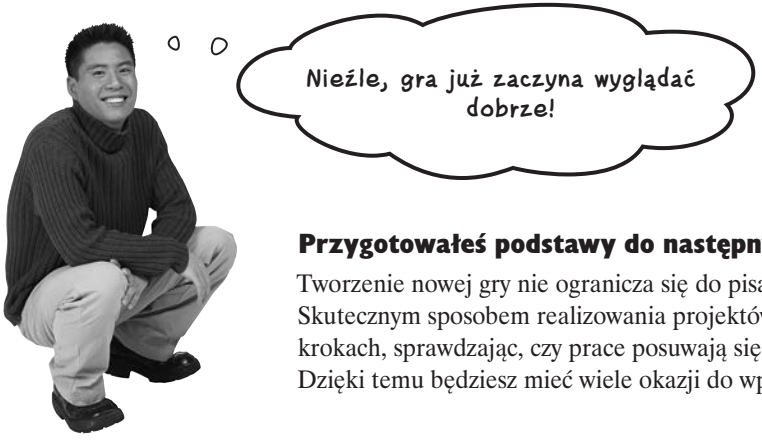


Kliknij pierwszy przycisk z narzędzi, aby wyświetlić panel *Dynamiczne drzewo wizualne* w IDE:

Następnie kliknij pierwszy przycisk panelu *Dynamiczne drzewo wizualne*, aby wyłączyć narzędzia środowiska uruchomieniowego.

Gdy program działa, IDE przechodzi w tryb debugowania. Przycisk *Start* jest zastępowany nieaktywnym (jasnoszarym) przyciskiem *Kontynuuj*, a na pasku narzędzi pojawiają się **kontrolki debugowania** (⏸ ■ ↻) z przyciskami do wstrzymywania, zatrzymywania i ponownego uruchamiania programu.

Zakończ program, klikając znak *X* w prawym górnym rogu okna lub kwadracik *Zatrzymaj* w kontrolkach debugowania. Uruchom program kilkakrotnie. Zwierzęta za każdym razem powinny pojawiać się gdzie indziej.



Przygotowałeś podstawy do następnych dodawanych elementów.

Tworzenie nowej gry nie ogranicza się do pisania kodu. Realizujesz też projekt. Skutecznym sposobem realizowania projektów jest tworzenie ich w małych krokach, sprawdzając, czy prace posuwają się w dobrym kierunku. Dzięki temu będziesz mieć wiele okazji do wprowadzenia poprawek.

Kup książkę Poleć książkę

To następne ćwiczenie z ołówkiem i kartką. Zdecydowanie warto poświęcić czas na wykonanie wszystkich takich ćwiczeń, ponieważ pomogą Ci one szybciej opanować ważne zagadnienia związane z językiem C#.

KTO CO ROBI?

Gratulacje — napisałeś działający program! To prawda, programowanie to coś dużo więcej niż kopiowanie kodu z książki. Jednak nawet jeśli nigdy wcześniej nie pisałeś kodu, może zaskoczyć Cię, jak duże jego fragmenty już rozumiesz. Narysuj linię łączącą każdą z instrukcji w C# (po lewej) z opisami ich działania (po prawej). Pierwszy punkt wykonaliśmy już za Ciebie.

Instrukcje C#

```
List<string> animalEmoji = new List<string>()
{
    "🐙","🐙",
    "🐷","🐷",
    "🐘","🐘",
    "🐮","🐮",
    "🐫","🐫",
    "🦒","🦒",
    "🐰","🐰",
    "🐌","🐌",
};
```

```
Random random = new Random();
```

```
foreach (TextBlock textBlock in
    mainGrid.Children.OfType<TextBlock>())
```

```
int index = random.Next(animalEmoji.Count);
```

```
string nextEmoji = animalEmoji[index];
```

```
textBlock.Text = nextEmoji;
```

```
animalEmoji.RemoveAt(index);
```

Działanie kodu

Przypisuje do kontrolki `TextBlock` losowe emoji z listy.

Znajduje wszystkie kontrolki `TextBlock` z głównej siatki i wykonuje dla każdej z nich podane instrukcje.

Usuwa losowe emoji z listy.

Tworzy listę ośmiu par emoji.

Wybiera liczbę losową z przedziału od 0 do liczby emoji pozostałych na liście i nazywa tę wartość „index”.

Tworzy nowy generator liczb losowych.

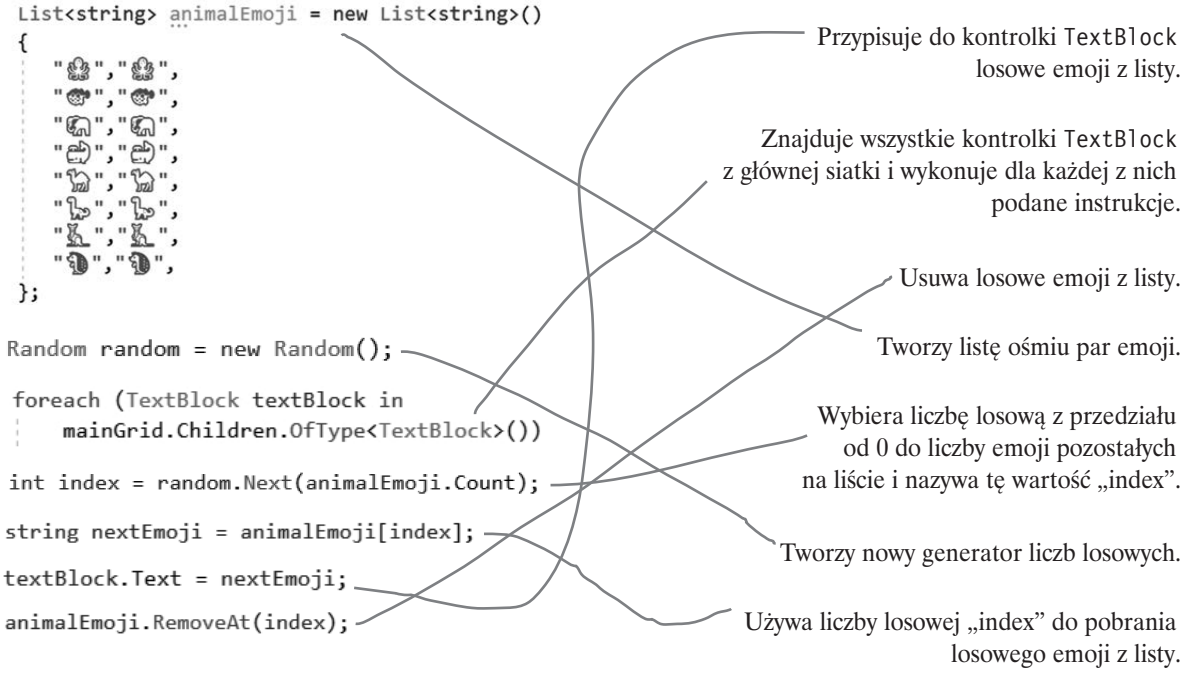
Używa liczby losowej „index” do pobrania losowego emoji z listy.

Kup książkę Poleć książkę

KTO CO ROBI?

ROZWIĄZANIE

Instrukcje C# **Działanie kodu**



MINI

Zaostrz ołówek

Oto ćwiczenie z ołówkiem i kartką, które pomoże Ci dobrze zrozumieć napisany kod w C#.

1. Weź kartkę i ułóż ją w orientacji poziomej. Narysuj pośrodku pionową linię.
2. Po lewej stronie zapisz całą metodę `SetUpGame`. Między poszczególnymi instrukcjami pozostaw puste miejsce. Nie musisz dbać o precyzję przy rysowaniu emoji.
3. Po prawej stronie kartki obok odpowiednich instrukcji zapisz każdą odpowiedź z ćwiczenia „kto co robi?”. Przeczytaj tekst po obu stronach linii — kod powinien nabrać sensu.

Kup książkę Poleć książkę

Nie jestem przekonana co do zadań „Zaostrz ołówek" i ćwiczenia z dopasowywaniem. Czy nie byłoby lepiej, gdybyście po prostu **pokazali mi kod** do wpisania w IDE?

Rozwijanie umiejętności zrozumienia kodu sprawi, że staniesz się lepszym programistą.

Ćwiczenie z użyciem ołówka i kartki **nie są opcjonalne**. Zapewniają Twojemu mózgowi inny sposób przyswajania informacji. Robią też coś jeszcze ważniejszego — dają Ci możliwość ***popełniania błędów***. Pomyłki są częścią procesu uczenia się i każdy z nas robił wiele bęłdów (możliwe nawet, że w tej książce znajdziesz jakąś literówkę lub dwie!). Nikt nie pisze od razu doskonałego kodu. Naprawdę dobrzy programiści zawsze zakładają, że kod, który piszą dziś, jutro może wymagać zmian. Dalej w książce zapoznasz się z procesem *refaktoryzacji*. Jest to technika programowania, która polega na ulepszaniu kodu już po jego napisaniu.

Dodajemy tego rodzaju listy wypunktowane, aby zapewnić krótkie streszczenie wielu opisanych do danego miejsca zagadnień i narzędzi.

CELNE SPOSTRZEŻENIA

- Visual Studio jest **IDE Microsoftu** (**zintegrowanym środowiskiem programistycznym**), które upraszcza i wspomaga edycję plików z kodem C# i zarządzanie nimi.
- **Aplikacje konsolowe .NET Core** to działające w różnych systemach aplikacje z tekstowymi danymi wejściowymi i wyjściowymi.
- **Wykorzystujący sztuczną inteligencję mechanizm IntelliSense** w IDE pomaga w szybszym wprowadzeniu kodu.
- **WPF** (ang. *Windows Presentation Foundation*) to technologia tworzenia w C# aplikacji z interfejsem graficznym.
- Interfejsy użytkownika w technologii WPF są projektowane w języku **XAML** (ang. *eXtensible Application Markup Language*). Jest to bazujący na XML-u język znaczników, w którym znaczniki i właściwości służą do definiowania kontrolek interfejsu użytkownika.
- **Kontrolka XAML `Grid`** służy do tworzenia układu siatki do przechowywania innych kontrolek.
- **Znacznik XAML `TextBlock`** reprezentuje kontrolkę do przechowywania tekstu.
- **Okno *Właściwości*** w IDE umożliwia łatwe modyfikowanie właściwości kontrolek — na przykład układu, tekstu albo wiersza lub kolumny siatki, gdzie kontrolka się znajduje.

Kup książkę Poleć książkę

Dodaj nowy projekt do systemu kontroli wersji

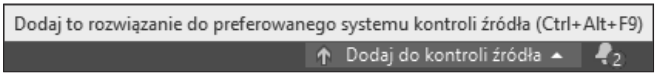
W tej książce będziesz tworzyć wiele różnych projektów. Czy nie byłoby świetnie, gdyby istniał łatwy sposób na archiwizowanie ich i łatwy dostęp do nich z dowolnego miejsca? Co zrobisz, jeśli popełnisz błąd — czy nie byłoby wygodnie, gdybyś mógł odzyskać starszą wersję kodu? No cóż, masz szczęście! Tak właśnie działają **systemy kontroli wersji** — umożliwiają łatwe zarchiwizowanie całego kodu i śledzenie wszystkich wprowadzonych zmian. Visual Studio pozwala na łatwe dodawanie projektów do takich systemów.

Git to popularny system kontroli wersji, a Visual Studio może zapisywać kod źródłowy w dowolnym **repozytorium** Git. Uważamy, że **GitHub** jest jednym z najłatwiejszych w użyciu dostawców systemu Git. Aby umieszczać kod w serwisie GitHub, musisz założyć w nim konto. Jeśli jeszcze nie masz takiego konta, otwórz stronę *https://github.com* i utwórz je.

Dodawanie projektu do systemu kontroli wersji jest opcjonalne.

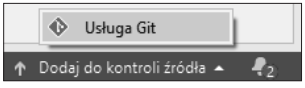
Możliwe, że pracujesz na komputerze w sieci firmowej, która nie zapewnia dostępu do serwisu GitHub (jest to polecany przez nas dostawca systemu Git). Możliwe też, że nie masz ochoty używać takiego systemu. Niezależnie od powodów możesz pominąć ten krok. Możesz też umieścić kod w prywatnym repozytorium, jeśli chcesz utworzyć kopię zapasową, ale nie chcesz, by inne osoby mogły znaleźć kod.

Znajdź opcję ***Dodaj do kontroli źródła*** na pasku stanu w dolnej części IDE:

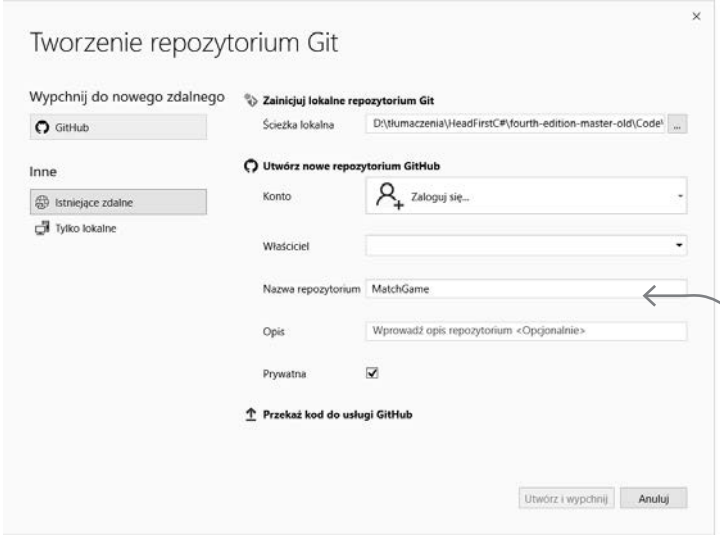


Zaraz po dodaniu kodu do systemu Git pasek stanu zmieni się, informując, że kod projektu znajduje się w systemie kontroli wersji. Git jest bardzo popularnym systemem kontroli wersji, a Visual Studio obejmuje kompletnego klienta tego systemu. Katalog projektu zawiera teraz ukryty katalog *.git* używany do śledzenia wszystkich poprawek w kodzie.

Kliknij tę opcję. Visual Studio pozwoli dodać kod do systemu Git:



Kliknij *Usługa Git*. Visual Studio wyświetli okno ***Tworzenie repozytorium Git***.

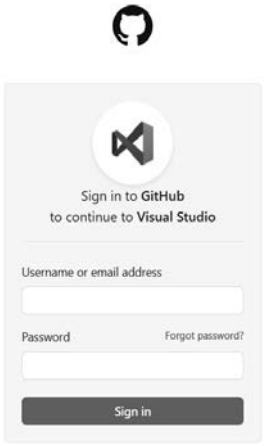


Visual Studio utworzy repozytorium w Twoim koncie w serwisie GitHub. Domyślnie nazwa repozytorium jest taka sama jak nazwa projektu.

Kup książkę Poleć książkę

Git jest otwartym systemem kontroli wersji. Istnieje wiele niezależnych serwisów takich jak GitHub, które udostępniają usługi związane z systemem Git (na przykład przestrzeń dyskową na kod i dostęp do repozytorium z poziomu internetu). Więcej o systemie Git dowiesz się na stronie *https://git-scm.com*.

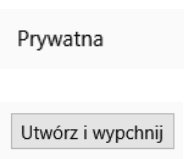
Kliknij Zaloguj się... . Visual Studio otworzy w przeglądarce **formularz logowania w serwisie GitHub**. Wprowadź nazwę użytkownika i hasło do serwisie GitHub. Jeśli skonfigurowałeś uwierzytelnianie dwuetapowe, zostaniesz poproszony o jego użycie. Po zalogowaniu się może pojawić się prośba o przyznanie serwisowi GitHub do środowiska Visual Studio. Jeśli zobaczysz taką prośbę, przyznaj uprawnienia, aby umożliwić środowisku Visual Studio tworzenie repozytoriów i przesyłanie kodu.



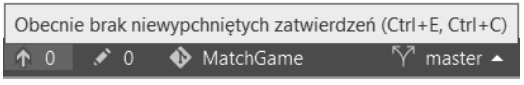
Upewnij się, że wpisana jest Twoja nazwa użytkownika z serwisu GitHub.

Po zalogowaniu się do serwisu GitHub status systemu Git na pasku stanu zmieni się. Zobaczysz, że nie ma już **zatwierdzeń** (zapisanych wersji kodu), które nie zostały ***wypchnięte*** do lokalizacji poza Twoim komputerem. To oznacza, że projekt jest zsynchronizowany z repozytorium z Twojego konta w serwisie GitHub.

Kliknij **przycisk *Utwórz i wypchnij***, aby utworzyć nowe repozytorium w serwisie GitHub i opublikować w nim kod. Po wypchnięciu kodu do serwisu GitHub

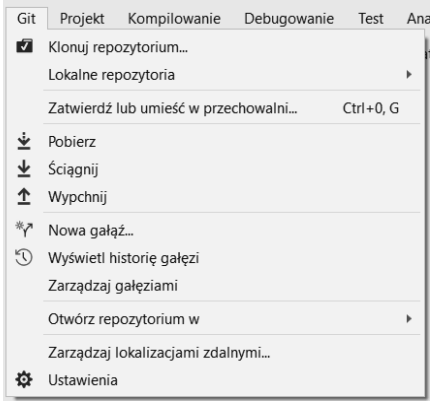


Po zalogowaniu się do serwisu GitHub użyj tego przycisku do opublikowania projektu na swoim koncie.



Zwróć uwagę na pasek stanu. Jeśli zobaczysz liczbę, na przykład ↑ 2, jest to informacja, że masz dwa niewypchnięte zatwierdzenia, które możesz przesłać do repozytorium w serwisie GitHub.

Po opublikowaniu kodu w serwisie GitHub możesz używać poleceń z menu Git do pracy z repozytorium systemu Git



Otwórz stronę *https://github.com/<twoja-nazwa-w-github>/MatchGame*, aby zobaczyć przesłany kod. Gdy zsynchronizujesz projekt ze zdalnym repozytorium, zobaczysz aktualizacje w sekcji z zatwierdzeniami.

Kup książkę Poleć książkę

Nie istnieją głupie pytania

P: Czy kod XAML to prawdziwy kod?

O: Tak, jak najbardziej. Pamiętasz czerwone falowane linie pod nazwą `mainGrid` w kodzie C#, które zniknęły dopiero po dodaniu nazwy do znacznika `Grid` w kodzie XAML? Stało się tak, ponieważ zmodyfikowałeś kod. Po dodaniu nazwy do kodu XAML kod C# mógł jej użyć.

P: Myślałem, że XAML działa jak HTML, który jest interpretowany przez przeglądarkę. Czy XAML działa tak samo?

O: Nie, kod XAML jest rozwijany razem z kodem C#. W następnym rozdziale zobaczysz, że można użyć słowa kluczowego `partial` do podziału klasy na kilka plików. Ten mechanizm służy też do złączania kodu XAML i C#. Kod XAML definiuje interfejs użytkownika, a C# definiuje operacje, a oba te komponenty są złączane za pomocą klas częściowych.

Dlatego ważne jest, aby traktować kod XAML jak zwykły kod. Z tego samego powodu opanowanie języka XAML jest ważne dla każdego programisty używającego C#.

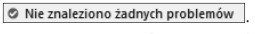
P: Zauważyłem MNÓSTWO wierszy `using` na początku pliku z kodem C#. Dlaczego jest ich aż tyle?

O: Aplikacje WPF korzystają z kodu z różnych *przestrzeni nazw* (w następnym rozdziale dowiesz się, czym one są). Gdy Visual Studio tworzy projekt WPF, automatycznie dodaje na początku pliku *MainWindow.xaml.cs* **dyrektywy `using`** dla najczęściej używanych przestrzeni nazw. Z niektórych przestrzeni nazw już korzystałeś. IDE używa jaśniejszej czcionki do wyróżniania przestrzeni nazw, które nie są wykorzystywane w kodzie.

P: Aplikacje desktopowe wydają się dużo bardziej skomplikowane od konsolowych. Czy naprawdę oba typy aplikacji działają tak samo?

O: Tak. Na podstawowym poziomie cały kod C# działa tak samo: wykonywana jest jedna instrukcja, potem druga itd. Aplikacje desktopowe wydają się bardziej skomplikowane, ponieważ niektóre metody są wywoływane tylko wtedy, gdy dzieją się określone rzeczy — na przykład wyświetlane jest okno lub użytkownik kliknął przycisk. Gdy metoda już zostanie wywołana, działa dokładnie tak samo jak w aplikacji konsolowej.

Wskazówka na temat IDE: Lista błędów

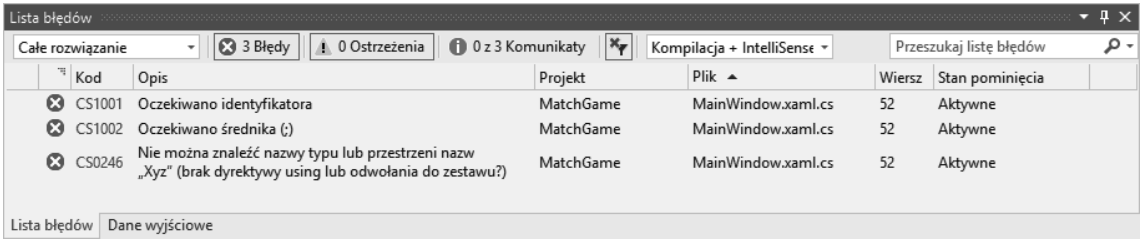
Spójrz na dolną część edytora kodu. Zwróć uwagę na komunikat 

. Oznacza to, że kod można **skompilować**. Kompilacja polega na przekształcaniu kodu przez IDE na postać **binarną**, którą system operacyjny może uruchomić. Wprowadź teraz błąd w kodzie.

Przejdź do pierwszego wiersza nowej metody `SetUpGame`. Wciśnij dwukrotnie klawisz *Enter*, a następnie w nowym wierszu wpisz `Xyz`.

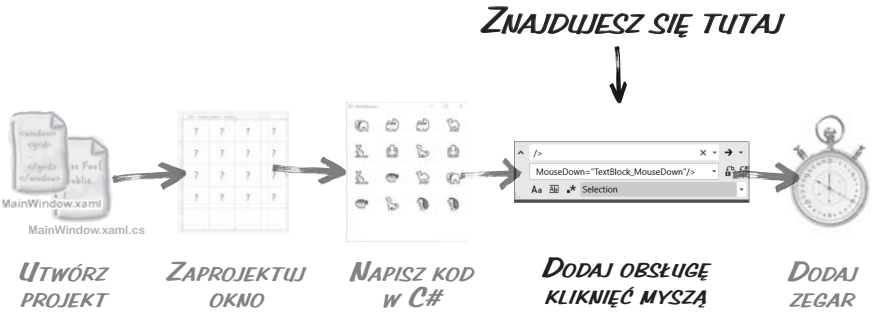
Ponownie spójrz na dolną część edytora kodu. Teraz widnieje tam komunikat

. Jeśli okno *Lista błędów* nie jest widoczne, otwórz je, wybierając je w menu *Widok*. Na liście błędów znajdują się teraz trzy błędy:



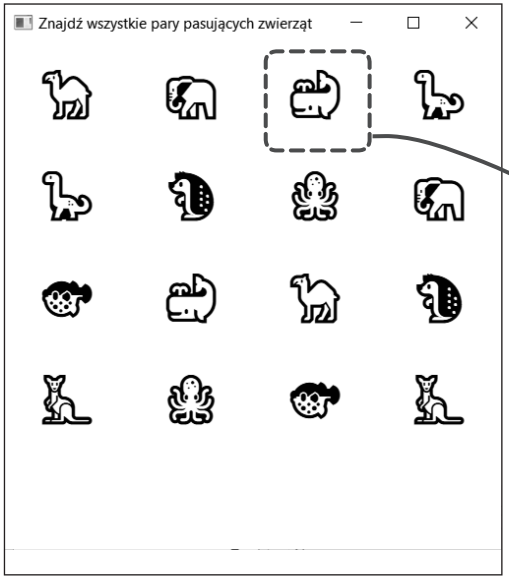
IDE wyświetliło informacje o błędach, ponieważ tekst Xyz nie jest poprawnym kodem w języku C# i uniemożliwia kompilację kodu. Dopóki w kodzie znajdują się błędy, nie będzie on działać. Dlatego usuń dodany wiersz Xyz.

Kup książkę Poleć książkę



Następny krok tworzenia gry to dodanie obsługi kliknięć myszą

Gdy gra wyświetla już zwierzęta, które gracz może kliknąć, trzeba dodać kod odpowiadający za przebieg gry. Gracz ma klikać pary zwierząt. Pierwsze kliknięte zwierzę znika. Jeśli drugie kliknięte zwierzę pasuje do pierwszego, także przestaje być widoczne. Jeśli jednak kliknięte rysunki nie pasują do siebie, program ma je ponownie wyświetlić. Aby umożliwić działanie tego procesu, dodasz **procedurę obsługi zdarzeń**. Taka procedura to metoda wywoływana, gdy w aplikacji zajdą określone zdarzenia (kliknięcie myszą, dwukrotne kliknięcie, zmiana wielkości okna itd.).



Gdy gracz kliknie jedno ze zwierząt, aplikacja wywoła metodę TextBlock_MouseDown, która obsługuje kliknięcia myszą. Oto, co ta metoda będzie robić.

```
TextBlock_MouseDown() {

  /* Jeśli kliknięte zostało pierwsze zwierzę,
   * należy zapisać, która kontrolka TextBlock
   * została kliknięta i usunąć grafikę. Jeżeli
   * kliknięto drugie zwierzę, należy albo też
   * usunąć grafikę (dopasowanie), albo ponownie
   * wyświetlić pierwszy rysunek (brak dopasowania).
   */
}
```

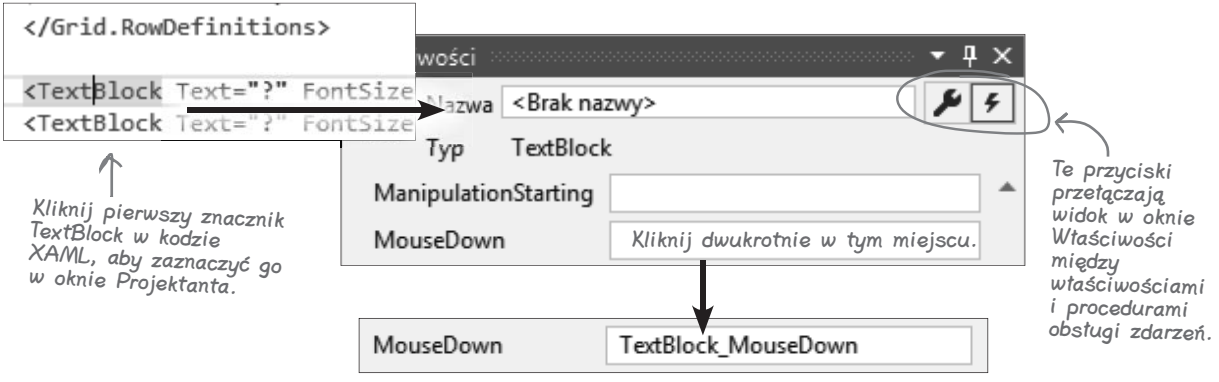
To jest komentarz. Wszystko między znakami / i */ jest ignorowane przez C#. Ten komentarz został dodany, aby opisać działanie metody TextBlock_MouseDown, a także by pokazać, jak wyglądają komentarze.*

Kup książkę Poleć książkę

Spraw, aby kontrolki TextBlock reagowały na kliknięcia myszą

Metoda `SetUpGame` modyfikuje kontrolki `TextBlock`, aby wyświetlały emoji ze zwierzętami. Wiesz już więc, w jaki sposób za pomocą kodu modyfikować kontrolki w aplikacji. Teraz napiszesz kod innego rodzaju, który będzie wywoływany przez kontrolki. Pomoże Ci w tym IDE.

Wróć do edytora kodu XAML i **kliknij pierwszy znacznik `TextBlock`**. IDE zaznaczy powiązaną kontrolkę w *Projektancie* i będziesz mógł zmienić jej właściwości. W oknie *Właściwości* kliknij przycisk *Procedury obsługi zdarzeń* ( ). **Procedura obsługi zdarzeń** jest metodą wywoływaną przez aplikację po zajściu określonych zdarzeń. Takie zdarzenia to wciśnięcie klawisza, przeciągnięcie elementu, zmiana wielkości okna lub, naturalnie, ruchy i kliknięcia myszą. Przewiń okno *Właściwości* w dół i przyjrzyj się nazwom różnych zdarzeń kontrolki `TextBlock`, dla jakich można dodać procedury obsługi. **Kliknij dwukrotnie pole na prawo od zdarzenia `MouseDown`**.



IDE zapełni pole `MouseDown` nazwą metody, `TextBlock_MouseDown`, a w kodzie XAML danej kontrolki `TextBlock` znajdzie się właściwość `MouseDown`:

```
<TextBlock Text="?" FontSize="36" HorizontalAlignment="Center"
    VerticalAlignment="Center" MouseDown="TextBlock_MouseDown"/>
```

Może tego nie zauważyłeś, ponieważ IDE ponadto **dodało nową metodę** do kodu zaplecza (powiązanego z kodem XAML) i natychmiast otworzyło edytor kodu C#, aby wyświetlić tę metodę. Zawsze możesz wrócić do tej metody z edytora kodu XAML, klikając prawym przyciskiem myszy właściwość `TextBlock_MouseDown` i wybierając opcję *Pokaż kod*. Oto ta metoda:

```
private void TextBlock_MouseDown(object sender, MouseButtonEventArgs e)
{

}
```

Gdy gracz kliknie tę kontrolkę `TextBlock`, aplikacja automatycznie wywoła metodę `TextBlock_MouseDown`. Dlatego teraz wystarczy dodać kod tej metody. Następnie będziesz musiał powiązać z tą metodą także wszystkie pozostałe kontrolki `TextBlock`, aby była wywoływana również dla nich.

Procedura obsługi zdarzenia jest metodą, którą aplikacja wywołuje w reakcji na zdarzenia takie jak kliknięcie myszą, wciśnięcie klawisza lub zmiana wielkości okna.

Kup książkę Poleć książkę

Zaostrz ołówek

Oto kod metody `TextBlock_MouseDown`. Zanim dodasz go do programu, przeczytaj go i spróbuj ustalić, co robi. Nie przejmuj się, jeśli nie udzielisz w pełni poprawnych odpowiedzi! Celem jest rozpoczęcie treningu mózgu w rozpoznawaniu kodu C# jako czegoś, co potrafisz odczytać i zrozumieć.

```
TextBlock lastTextBlockClicked;
bool findingMatch = false;

private void TextBlock_MouseDown(object sender, MouseButtonEventArgs e)
{
    TextBlock textBlock = sender as TextBlock;
    if (findingMatch == false)
    {
        textBlock.Visibility = Visibility.Hidden;
        lastTextBlockClicked = textBlock;
        findingMatch = true;
    }
    else if (textBlock.Text == lastTextBlockClicked.Text)
    {
        textBlock.Visibility = Visibility.Hidden;
        findingMatch = false;
    }
    else
    {
        lastTextBlockClicked.Visibility = Visibility.Visible;
        findingMatch = false;
    }
}
```

1. Co robi `findingMatch`?

_____
_____

2. Co robi blok kodu zaczynający się od `if (findingMatch == false)`?

_____
_____

3. Co robi blok kodu zaczynający się od `if (textBlock.Text == lastTextBlockClicked.Text)`?

_____
_____

4. Co robi blok kodu zaczynający się od `else`?

_____
_____

Kup książkę Poleć książkę

Zaostrz ołówek
Rozwiązanie

Oto kod metody `TextBlock_MouseDown`. Zanim dodasz go do programu, przeczytaj go i spróbuj ustalić, co robi. Nie przejmuj się, jeśli nie udzielisz w pełni poprawnych odpowiedzi! Celem jest rozpoczęcie treningu mózgu w rozpoznawaniu kodu C# jako czegoś, co potrafisz odczytać i zrozumieć.

```
TextBlock lastTextBlockClicked;
bool findingMatch = false;

private void TextBlock_MouseDown(object sender, MouseButtonEventArgs e)
{
    TextBlock textBlock = sender as TextBlock;
    if (findingMatch == false)
    {
        textBlock.Visibility = Visibility.Hidden;
        lastTextBlockClicked = textBlock;
        findingMatch = true;
    }
    else if (textBlock.Text == lastTextBlockClicked.Text)
    {
        textBlock.Visibility = Visibility.Hidden;
        findingMatch = false;
    }
    else
    {
        lastTextBlockClicked.Visibility = Visibility.Visible;
        findingMatch = false;
    }
}
```

Oto opis działania kodu metody `TextBlock_MouseDown`. Czytanie kodu w nowym języku programowania przypomina czytanie nut. Ta umiejętność wymaga praktyki, a im częściej będziesz ją wykorzystywać, tym lepiej będzie Ci szło.

1. Co robi `findingMatch`?

Rejestruje, czy gracz kliknął już wcześniej pierwsze zwierzę w parze i teraz próbuje znaleźć pasujący rysunek.

2. Co robi blok kodu zaczynający się od `if (findingMatch == false)`?

Gracz kliknął pierwsze zwierzę w parze, dlatego kod ukrywa rysunek tego zwierzęcia i zapisuje powiązaną kontrolkę TextBlock na wypadek, gdyby trzeba było ponownie wyświetlić grafikę.

3. Co robi blok kodu zaczynający się od `if (textBlock.Text == lastTextBlockClicked.Text)`?

Gracz znalazł pasujące zwierzę! Kod ukrywa drugie zwierzę w parze (i uniemożliwia jego kliknięcie) oraz resetuje wartość findingMatch, aby następne kliknięte zwierzę znów było traktowane jak pierwsze w parze.

4. Co robi blok kodu zaczynający się od `else`?

Gracz kliknął zwierzę, które nie pasuje do pierwszego. Dlatego kod ponownie wyświetla pierwsze kliknięte zwierzę i resetuje wartość findingMatch.

Kup książkę Poleć książkę

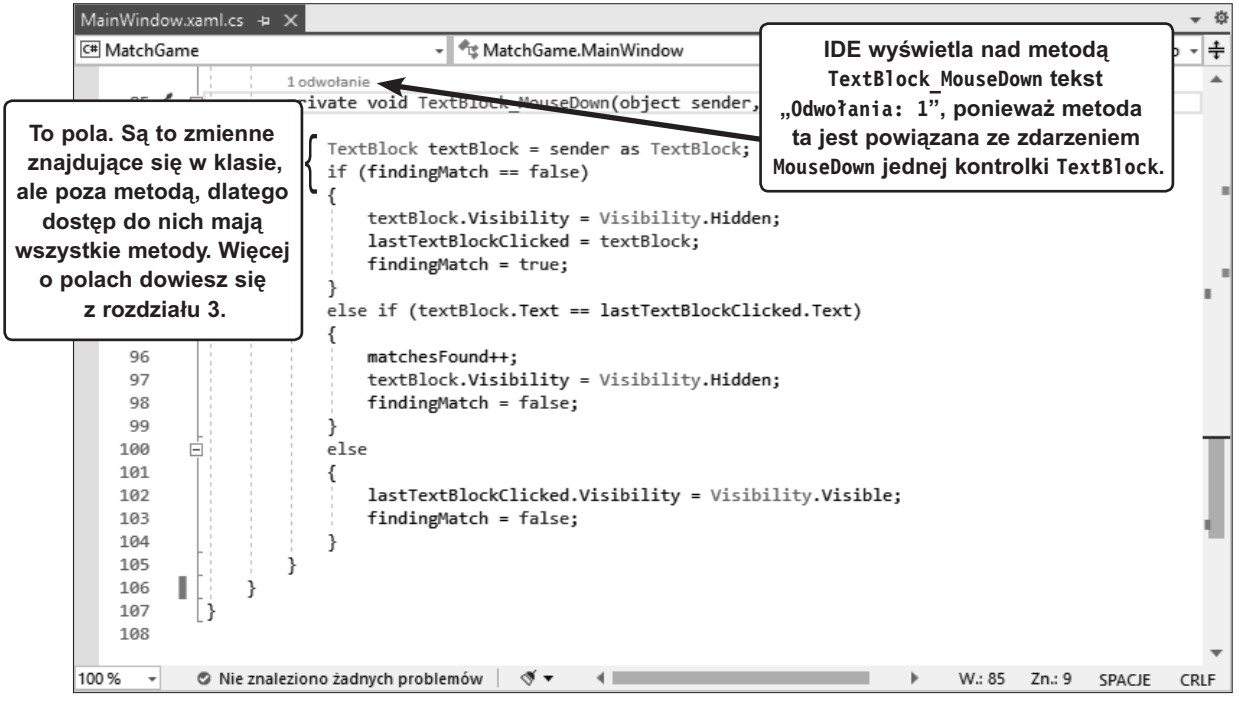
Dodaj kod metody TextBlock_MouseDown

Po zapoznaniu się z kodem metody `TextBlock_MouseDown` pora dodać go do programu. Oto, co powinieneś zrobić:

1. Dodaj dwa pierwsze wiersze z polami `lastTextBlockClicked` i `findingMatch` **nad pierwszym wierszem** metody `TextBlock_MouseDown` dodanej przez IDE. Upewnij się, że znajdują się one między zamykającym nawiasem klamrowym metody `SetUpGame` a nowym kodem dodanym przez IDE.

2. **Uzupełnij kod** metody `TextBlock_MouseDown`. Uważaj na znaki równości — jest duża różnica między symbolami = i == (dowiesz się o niej z następnego rozdziału).

W IDE kod powinien wyglądać tak:



Kup książkę Poleć książkę

Spraw, aby pozostałe kontrolki TextBlock wywoływały tę samą procedurę obsługi zdarzeń MouseDown

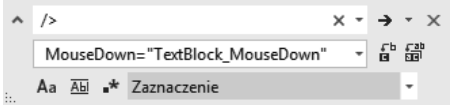
Obecnie tylko pierwsza kontrolka `TextBlock` ma procedurę obsługi zdarzenia `MouseDown`. Pora powiązać z tą procedurą pozostałych 15 kontrolek `TextBlock`. *Mógłbyś* to zrobić, wybierając każdą kontrolkę w *Projektancie* i wpisując `TextBlock_MouseDown` w polu obok zdarzenia `MouseDown`. Wiesz już jednak, że powoduje to tylko dodanie właściwości w kodzie XAML, dlatego można wybrać krótsze rozwiązanie.

1 **Zaznacz pozostałych 15 kontrolek TextBlock w edytorze kodu XAML.**

Przejdź do edytora kodu XAML, kliknij po lewej stronie drugiego znacznika `TextBlock` i przeciągnij kursor w dół do końca listy kontrolek `TextBlock` (do miejsca przed końcowym znacznikiem `</Grid>`). Teraz zaznaczonych powinno być ostatnich 15 kontrolek (bez pierwszej).

2 **Użyj funkcji Szybkie zamienianie, aby dodać procedury obsługi zdarzenia MouseDown.**

Wybierz opcję ***Znajdź i zamień*/*Szybkie zamienianie*** w menu *Edycja*. Jako szukaną sekwencję wpisz `/>` i zastąp ją kodem `MouseDown="TextBlock_MouseDown"/>`. Pamiętaj, aby przed kodem `MouseDown` znajdowała się spacja, a jako zakres wyszukiwania wpisz ***Zaznaczenie*** (właściwość zostanie więc dodana tylko do zaznaczonych kontrolek `TextBlock`).



Przed MouseDown znajduje się spacja, aby nowa właściwość nie została połączona z wcześniejszą.

3 **Zamień tekst we wszystkich 15 zaznaczonych kontrolkach TextBlock.**

Kliknij przycisk *Zamień wszystkie* ( ), aby dodać właściwość `MouseDown` do kontrolek `TextBlock`. Narzędzie powinno poinformować o 15 operacjach zamiany. Starannie sprawdź kod XAML, aby się upewnić, że każda kontrolka ma właściwość `MouseDown` identyczną z właściwością z pierwszej kontrolki `TextBlock`.

Upewnij się też, że edytor kodu C# informuje o **16 odwołaniach** do metody (wybierz opcję *Kompilowanie*/*Kompiluj rozwiązanie*, aby zaktualizować kod). Jeśli edytor informuje o 17 odwołaniach, przypadkowo dodałeś procedurę obsługi zdarzeń do kontrolki `Grid`. Z pewnością tego nie chcesz, ponieważ kliknięcie zwierzęcia spowoduje wtedy wyjątek.

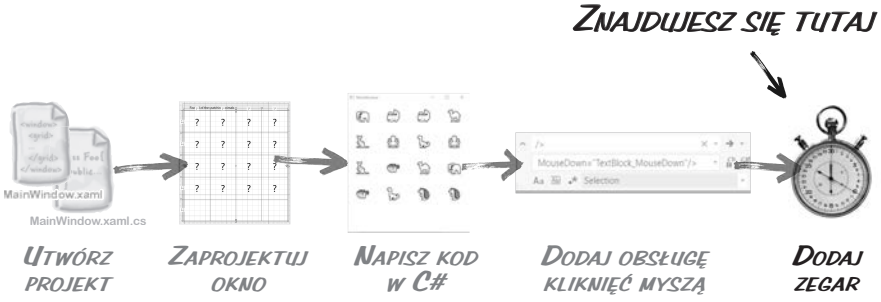
Uruchom program. Teraz możesz klikać pary zwierząt, aby usunąć je z siatki. Pierwsze kliknięte zwierzę znika. Jeśli potem klikniesz pasujące zwierzę, także ono zniknie. Jednak gdy klikniesz niepasujący rysunek, pierwsze zwierzę znów się pojawi. Po zniknięciu wszystkich zwierząt możesz ponownie uruchomić program lub go zamknąć.

Gdy napotkasz zadanie Wysil szare komórki, zatrzymaj się na chwilę i dobrze się zastanów nad zadanym pytaniem.

WYSIL SZARE KOMÓRKI

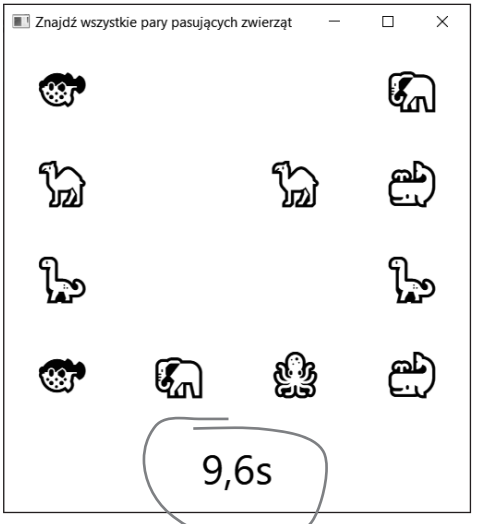
Dotarłeś do punktu kontrolnego w projekcie! Gra jeszcze nie jest skończona, ale działa i można w nią grać. Dlatego jest to dobry moment na zastanowienie się, jak można ją ulepszyć. Co możesz zrobić, aby stała się ciekawsza?

Kup książkę Poleć książkę



Ukończ grę, dodając zegar

Gra w dopasowywanie zwierząt będzie ciekawsza, jeśli gracze będą mogli próbować poprawić swój najlepszy czas. Dlatego dodasz **zegar**, który będzie „tykał” po określonych przedziałach czasu, wielokrotnie wywołując metodę.



Spraw, by gra stała się ciekawsza! W dolnej części okna ma się pojawiać czas od rozpoczęcia gry. Czas ma stale biec naprzód, a zatrzymywać się dopiero po dopasowaniu ostatniego zwierzęcia.

Zegar „tyka” po upływie określonych przedziałów czasu, w kółko wywołując metodę. Odliczanie ma się zaczynać w momencie rozpoczęcia gry przez gracza i kończyć po dopasowaniu ostatniego zwierzęcia.

Kup książkę Poleć książkę

Dodaj zegar do kodu gry

Dodaj to!

1. Najpierw znajdź słowo kluczowe `namespace` na początku pliku *MainWindow.xaml.cs* i bezpośrednio pod nim dodaj wiersz `using System.Windows.Threading;`.

```
namespace MatchGame
{
    using System.Windows.Threading;
```

2. Znajdź fragment `public partial class MainWindow` i zaraz po otwierającym nawiasie klamrowym, {, **dodaj ten kod**:

```
public partial class MainWindow : Window
{
    DispatcherTimer timer = new DispatcherTimer();
    int tenthsOfSecondsElapsed;
    int matchesFound;
```

Te trzy wiersze tworzą nowy zegar i dodają dwa pola do śledzenia upływu czasu i liczby dopasowanych par.

3. Trzeba poinstruować zegar, jak często ma „tykać” i jaką metodę ma wywoływać. Kliknij początek wiersza, w którym wywołujesz metodę `SetUpGame`, aby przenieść tam kursor edytora. Wciśnij *Enter*, a następnie wpisz dwa wiersze kodu z poniższego zrzutu zaczynające się od członu **`timer.`**. Zaraz po wpisaniu `+=` IDE wyświetli widoczny komunikat.

```
Odwołania: 0
public MainWindow()
{
    InitializeComponent();

    timer.Interval = TimeSpan.FromSeconds(.1);
    timer.Tick +=
    SetUpGame();
}
```

Timer_Tick1; (Naciśnij klawisz TAB, aby wstawić)

Następnie dodaj te dwa wiersze kodu. Zacznij wpisywać drugi wiersz: „timer.Tick +=”. Zaraz po wpisaniu znaku równości IDE wyświetli komunikat „Naciśnij klawisz TAB, aby wstawić”.

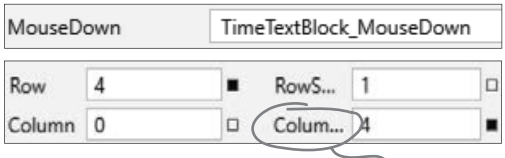
4. Wciśnij klawisz *Tab*. IDE dokończy wiersz kodu i doda metodę `Timer_Tick`:

```
Odwołania: 0
public MainWindow()
{
    InitializeComponent();

    timer.Interval = TimeSpan.FromSeconds(.1);
    timer.Tick += Timer_Tick;
    SetUpGame();
}

1 odwołanie
private void Timer_Tick(object sender, EventArgs e)
{
    throw new NotImplementedException();
}
```

Gdy wciśniesz klawisz Tab, IDE automatycznie wstawi metodę wywoływaną przez zegar.

Kup książkę Poleć książkę

5 Metoda `Timer_Tick` ma aktualizować kontrolkę `TextBlock` zajmującą cały dolny wiersz siatki. Wykonaj następujące operacje:



Właściwość ColumnSpan znajduje się w sekcji Układ okna Właściwości. Do przełączania się między właściwościami i zdarzeniami użyj przycisków w górnej części okna.

- ★ Przeciągnij kontrolkę **`TextBlock`** w lewe dolne pole.
- ★ Użyj **pola *Name*** w górnej części okna *Właściwości*, aby nazwać kontrolkę **`timeTextBlock`**.
- ★ Zresetuj **marginesy**, **wyśrodkuj** kontrolkę w komórce, ustaw właściwość ***FontSize*** na `36px`, a właściwość ***Text*** na `Czas: ` (podobnie jak robiłeś to w innych kontrolkach).
- ★ Znajdź właściwość ***ColumnSpan*** i ustaw jej wartość na `4`.
- ★ Dodaj **procedurę obsługi zdarzeń *MouseDown*** o nazwie `TimeTextBlock_MouseDown`.

Kod XAML kontrolki powinien wyglądać tak (starannie porównaj ten fragment z kodem w Twoim IDE):

```
<TextBlock x:Name="timeTextBlock" Text="Czas: " FontSize="36"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    Grid.Row="4" Grid.ColumnSpan="4" MouseDown="TimeTextBlock_MouseDown"/>
```

6 Gdy dodajesz procedurę obsługi zdarzeń `MouseDown`, Visual Studio tworzy w kodzie zaplecza metodę `TimeTextBlock_MouseDown` (podobnie jak dla innych kontrolek `TextBlock`). Dodaj do niej ten kod:

```
private void TimeTextBlock_MouseDown(object sender, MouseButtonEventArgs e)
{
    if (matchesFound == 8)
    {
      SetUpGame();
    }
}
```

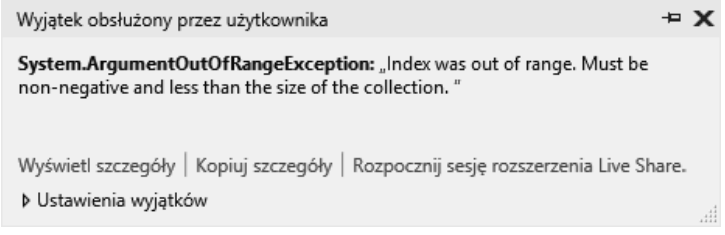
Ten kod resetuje grę, jeśli użytkownik znalazł wszystkich 8 pasujących par (w przeciwnym razie nic nie robi, ponieważ gra wciąż jest w toku).

7 Masz już wszystko, czego potrzebujesz do ukończenia metody `Timer_Tick`, która aktualizuje nową kontrolkę `TextBlock` na podstawie czasu, jaki upłynął, i zatrzymuje zegar po dopasowaniu wszystkich zwierząt:

```
private void Timer_Tick(object sender, EventArgs e)
{
    tenthsOfSecondsElapsed++;
    timeTextBlock.Text = (tenthsOfSecondsElapsed / 10F).ToString("0.0s");
    if (matchesFound == 8)
    {
        timer.Stop();
        timeTextBlock.Text = timeTextBlock.Text + " – Jeszcze raz?";
    }
}
```

Coś jednak jest nie tak. Uruchom kod — ups! Wystąpił **wyjątek**.

Zaraz naprawisz usterkę, jednak najpierw przyjrzyj się komunikatowi o błędzie i wyróżnionemu wierszowi w IDE.



Czy potrafisz odgadnąć, co spowodowało błąd?

O-o! Jak sądzisz, co się stało?

Kup książkę Poleć książkę

Użyj debugera do znalezienia przyczyny wyjątku

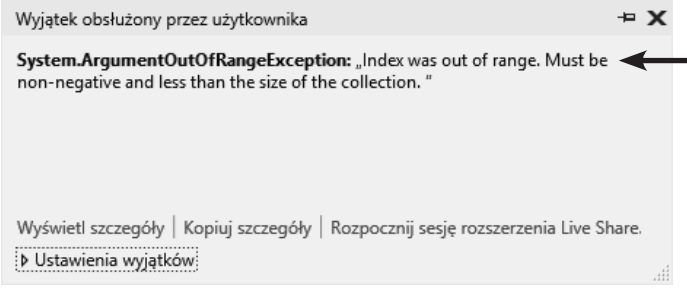
Możliwe, że zetknąłeś się już ze słowem „bug” (dosłownie „robak”, ale wyraz ten oznacza błąd). Może nawet słyszałeś, jak któryś z Twoim znajomych stwierdził: „Ta gra jest pełna bugów, ma mnóstwo usterek”. Każdy błąd ma wyjaśnienie i wszystko w programie dzieje się z jakiegoś powodu, jednak czasem trudno jest ustalić przyczynę usterki.

Zrozumienie błędu jest pierwszym krokiem do jego usunięcia. Na szczęście debuger w Visual Studio jest doskonałym narzędziem, które Ci w tym pomoże. Nazwa debuger pochodzi właśnie od tego, że narzędzie to pomaga eliminować błędy (bugi).

Zdebuguj to!

1 **Kilkakrotnie zrestartuj grę.**

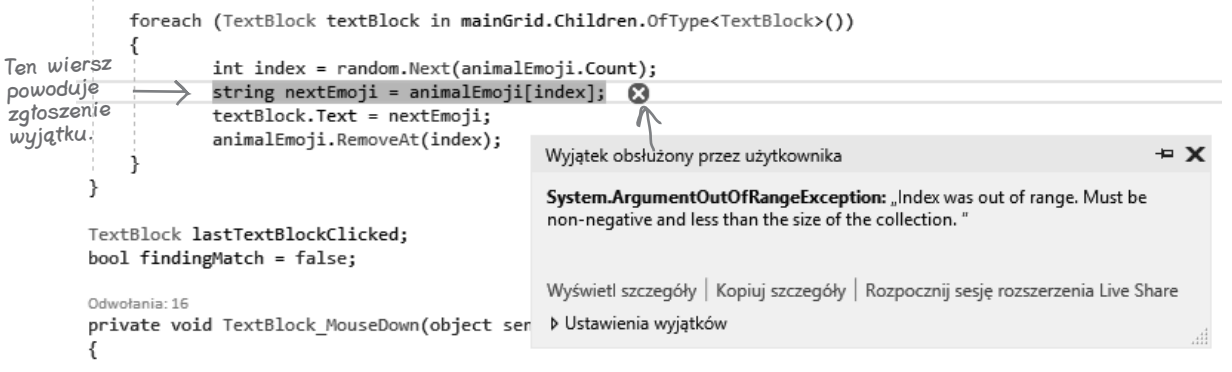
Pierwszą rzeczą, jaką warto zauważyć, jest to, że program zawsze zgłasza ten sam rodzaj wyjątku z tym samym komunikatem:



Wyjątki służą językowi C# do informowania o tym, że kod zadziałał nieprawidłowo. Każdy wyjątek jest określonego typu. Tu jest to typ `ArgumentOutOfRangeException`. Z wyjątkami powiązane są przydatne komunikaty pomagające ustalić przyczynę problemu. Tu treść komunikatu to: „Index was out of range”, czyli „parametr index jest poza zakresem”. Jest to cenna informacja pozwalająca stwierdzić, co poszło nie tak.

Gdy zobaczysz wyjątek, często możesz go traktować jak dobrą wiadomość — znalazłeś błąd i możesz go poprawić.

Jeśli przeniesiesz okno wyjątku, zobaczysz, że program zawsze zatrzymuje się w tym samym wierszu:



Ten wyjątek jest **powtarzalny**: możesz spowodować, by program zgłosił dokładnie ten sam wyjątek, i dobrze wiesz, gdzie znajduje się problem.

Kup książkę Poleć książkę

Anatomia debugera

Gdy aplikacja zostaje zatrzymana w debugerze (to tak zwane „przerwanie" aplikacji), na pasku narzędzi pojawiają się kontrolki debugera. W trakcie lektury tej książki nabierzesz wprawy w ich używaniu, dlatego nie musisz zapamiętywać, jak działają. Na razie przeczytaj opisy i umieść kursor nad poszczególnymi przyciskami, aby zobaczyć ich nazwy i skróty klawiaturowe.

Przycisk Wkrocz do wykonuje następną instrukcję. Jeśli jest to wywołanie metody, wykonywana jest tylko pierwsza instrukcja z danej metody.

Przycisk Uruchom ponownie restartuje aplikację. Jest to odpowiednik zatrzymania i ponownego uruchomienia aplikacji.

Możesz użyć przycisku Przerwij wszystkie, aby wstrzymać aplikację. Gdy aplikacja jest już wstrzymana, przycisk jest nieaktywny.

Przycisk Przekrocz nad też wykonuje następną instrukcję, jeśli jednak jest to wywołanie metody, wykonywana jest ona w całości.

Ten przycisk wznawia działanie aplikacji. Jeśli wciśniesz go teraz, ponownie zgłoszony zostanie ten sam wyjątek.

Już używałeś przycisku Zatrzymaj debugowanie do zatrzymywania aplikacji.

Przycisk Pokaż następną instrukcję przenosi kursor do następnej instrukcji, która ma zostać wykonana.

Przycisk Wyjdź kończy wykonywanie bieżącej metody i wstrzymuje program w pierwszym wierszu po wywołaniu tej metody.

2 Dodaj punkt przerwania w wierszu, który zgłasza wyjątek.

Ponownie uruchom program, aby zatrzymał się w miejscu wystąpienia wyjątku. Zanim zatrzymasz program, wybierz opcję ***Przełącz punkt przerwania (F9)*** z menu *Debugowanie*. Bieżący wiersz zostanie wtedy wyróżniony czerwonym kolorem, a na lewym marginesie pojawi się czerwona kropka. Teraz możesz **ponownie zatrzymać aplikację**. Wyróżnienie i kropka pozostaną w tym samym miejscu:

```
67
68    string nextEmoji = animalEmoji[index];
69    textBlock.Text = nextEmoji;
```

Właśnie dodałeś punkt przerwania do wiersza. Program będzie teraz wstrzymywany za każdym razem, gdy będzie wykonywał dany wiersz. Sprawdź to teraz. Ponownie uruchom aplikację. Program wstrzyma pracę w tym samym wierszu, ale tym razem ***nie zgłosi wyjątku***. Wciśnij przycisk *Kontynuuj*. Program ponownie wstrzyma działania w tym wierszu. Jeszcze raz wciśnij *Kontynuuj*. Program znów wstrzyma pracę. Powtarzaj tę operację do momentu wystąpienia wyjątku. Wtedy zatrzymaj aplikację.

Zaostrz ołówek

Ponownie uruchom aplikację, ale tym razem bacznie ją obserwuj i odpowiedz na następujące pytania:

1. Ile razy aplikacja wstrzymuje pracę w punkcie przerwania przed zgłoszeniem wyjątku? _____

2. W trakcie debugowania aplikacji pojawia się okno *Lokalne*. Jak myślisz, do czego służy to okno? Jeśli go nie widzisz, wybierz w menu opcję ***Debugowanie/Okna/Lokalne (Ctrl+D, L)***.

_____

_____

Kup książkę Poleć książkę

Zaostrz ołówek Rozwiązanie

Aplikacja wstrzymała pracę 17 razy. Po 17. razie zgłosiła wyjątek.

Okno Lokalne wyświetla aktualne wartości zmiennych i pól. Możesz go używać do obserwowania zmian tych wartości w trakcie wykonywania programu.

3 Zbierz dowody, aby ustalić, co powoduje problem.

Czy zauważyłeś coś ciekawego w oknie *Lokalne* po uruchomieniu aplikacji? Uruchom ją ponownie i bacznie śledź zmienną `animalEmoji`. Gdy aplikacja po raz pierwszy wstrzyma prace, w oknie *Lokalne* powinieneś zobaczyć takie informacje:

Wciśnij przycisk *Kontynuuj*. Wygląda na to, że wartość licznika zmniejszyła się o 1 (z 16 na 15):

Aplikacja dodaje losowe emocji z listy `animalEmoji` do kontrolek `TextBlock`, a następnie usuwa dodane emoji z tej listy, dlatego licznik za każdym razem powinien zmniejszać się o 1. Wszystko idzie dobrze do czasu, gdy lista `animalEmoji` staje się pusta (licznik wtedy przyjmuje wartość 0) i aplikacja zgłasza wyjątek. To pierwszy dowód! Drugim dowodem jest to, że wyjątek jest zgłaszany w **pętli `foreach`**. Ostatnim dowodem jest to, że ***problemy pojawiły się po dodaniu do okna nowej kontrolki `TextBlock`***.

Czas wcielić się w Sherlocka Holmesa. Czy potrafisz wydedukować, co powoduje wyjątek?

Za kulisami

Instrukcja foreach to rodzaj pętli i wykonuje operacje dla każdego elementu kolekcji.

Pętla umożliwia wielokrotne wykonywanie bloku kodu. W pokazanym kodzie używana jest **pętla `foreach`**. Jest to specjalna pętla wykonująca ten sam kod dla każdego elementu kolekcji (na przykład listy `animalEmoji`). Oto przykładowa pętla `foreach` przetwarzająca listę liczb:

```
List<int> numbers = new List<int>() { 2, 5, 9, 11 };
foreach (int aNumber in numbers)
{
    Console.WriteLine("Liczba to: " + aNumber);
}
```

Ta pętla foreach wykonuje instrukcję Console.WriteLine dla każdej wartości z listy liczb całkowitych.

Ta pętla `foreach` tworzy nową zmienną, `aNumber`. Następnie przetwarza po kolei elementy listy `numbers` i dla każdego z nich wywołuje instrukcję `Console.WriteLine`, przypisując do `aNumber` kolejne wartości z listy:

```
Liczba to: 2
Liczba to: 5
Liczba to: 9
Liczba to: 11
```

Pętla foreach wielokrotnie wykonuje ten sam kod dla każdego elementu kolekcji, za każdym razem przypisując do zmiennej następny element. Tu pętla przypisuje do aNumber kolejną liczbę z listy i używa tej wartości do wyświetlenia wiersza tekstu.

Wprowadzamy tu nowe zagadnienie (choć tylko pokrótce), aby działanie kodu było zrozumiałe. Więcej o pętlach dowiesz się z rozdziału 2. Dalej, w rozdziale 3., wrócimy do pętli `foreach`. Napiszesz wtedy pętlę bardzo podobną do tej z tego rozdziału. Dlatego jeśli wydaje Ci się, że omawiamy materiał trochę za szybko, to w trakcie lektury rozdziału 3. będziesz mógł wrócić do tego przykładu i zobaczyć, czy kod stanie się bardziej zrozumiały. Uważamy, że ponowna lektura kodu po lepszym opanowaniu języka bardzo pomaga przyswoić sobie różne techniki. Nie martw się więc, jeśli niektóre kwestie wydają Ci się niejasne.

Kup książkę Poleć książkę

Wydedukuj to

4 Ustal, co jest przyczyną błędu.

Program przestaje działać, ponieważ próbuje pobrać następne emoji z listy `animalEmoji`, ale ta jest już pusta. To skutkuje wyjątkiem `ArgumentOutOfRange`. Co spowodowało brak emoji do pobrania?

Przed wprowadzeniem najnowszej zmiany program działał. Następnie dodałeś kontrolkę `TextBlock` i wtedy aplikacja przestała funkcjonować — i to w pętli, która iteracyjnie przetwarza wszystkie kontrolki `TextBlock`. To niezwykle ciekawe…

A zatem, kiedy uruchamiasz aplikację, ***wstrzymuje ona pracę w tym wierszu przy przetwarzaniu każdej kontrolki `TextBlock` z okna***. Dla pierwszych 16 kontrolek `TextBlock` kod działa poprawnie, ponieważ w kolekcji dostępne są emoji:

```
foreach (TextBlock textBlock in mainGrid.Children.OfType<TextBlock>())
{
    int index = random.Next(animalEmoji.Count);
    string nextEmoji = animalEmoji[index];
    textBlock.Text = nextEmoji;
    animalEmoji.RemoveAt(index);
}
```

Debuger wyróżnia instrukcję, którą ma uruchomić. Tak wygląda ona tuż przed zgłoszeniem wyjątku.

Jednak teraz, kiedy w dole okna znajduje się nowa kontrolka `TextBlock`, program wstrzymuje pracę po raz 17., a ponieważ kolekcja `animalEmoji` zawiera tylko 16 emoji, jest już pusta:

Oznacza to, że przed wprowadzeniem zmiany było 16 kontrolek `TextBox` i lista 16 emoji. Liczba emoji była wystarczająca, aby dodać po jednym do każdej kontrolki `TextBlock`. Jednak teraz masz 17 kontrolek `TextBlock` i tylko 16 emoji, dlatego programowi kończą się emoji do dodania, po czym zgłasza wyjątek.

5 Napraw błąd.

Ponieważ wyjątek jest zgłaszany z powodu wyczerpania się listy emoji w pętli, która przetwarza kontrolki `TextBlock`, można rozwiązać problem, pomijając dodaną kontrolkę. W tym celu sprawdź nazwę kontrolki `TextBlock` dodanej na potrzeby wyświetlania czasu i pomiń ją. Usuń punkt przerwania (aby to zrobić, kliknij go ponownie lub wybierz opcję ***Usuń wszystkie punkty przerwania*** (***Ctrl+Shift+F9***) z menu *Debugowanie*).

```
foreach (TextBlock textBlock in mainGrid.Children.OfType<TextBlock>())
{
    if (textBlock.Name != "timeTextBlock")
    {
        textBlock.Visibility = Visibility.Visible;
        int index = random.Next(animalEmoji.Count);
        string nextEmoji = animalEmoji[index];
        textBlock.Text = nextEmoji;
        animalEmoji.RemoveAt(index);
    }
}
```

Dodaj tę instrukcję `if` w pętli `foreach`, aby pomijała kontrolkę `TextBlock` o nazwie `timeTextBlock`.

Dodaj ten kod, aby naprawić błąd.

Nie jest to jedyny sposób na usunięcie błędu. Gdy zaczniesz pisać więcej kodu, zobaczysz, że istnieje wiele, wiele, WIELE możliwości rozwiązania każdego problemu. I błędy nie są tu wyjątkiem (bez żadnych skojarzeń).

Kup książkę Poleć książkę

Dodaj resztę kodu i dokończ grę

Do zrobienia pozostała jedna rzecz. Metoda `TimeTextBlock_MouseDown` sprawdza wartość pola `matchesFound`, ale ta wartość nie jest nigdzie zmieniana. Dodaj więc trzy pokazane tu wiersze do metody `SetUpGame`; umieść je bezpośrednio po zamykającym nawiasie klamrowym pętli `foreach`:

```
                animalEmoji.RemoveAt(index);
            }
        }

        timer.Start();
        tenthsOfSecondsElapsed = 0;
        matchesFound = 0;
    }
```

Dodaj te trzy wiersze kodu na końcu metody SetUpGame, aby uruchamiać zegar i resetować pola.

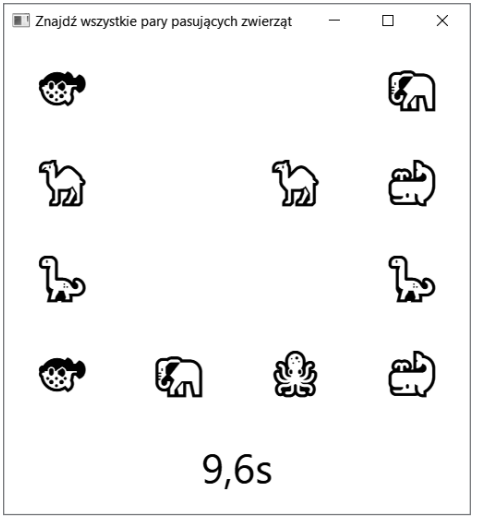
Następnie dodaj poniższą instrukcję wewnątrz bloku `if/else` w metodzie `TextBlock_MouseDown`:

```
else if (textBlock.Text == lastTextBlockClicked.Text)
{
    matchesFound++;
    textBlock.Visibility = Visibility.Hidden;
    findingMatch = false;
}
```

Dodaj ten wiersz kodu, aby zwiększać wartość pola matchesFound o jeden za każdym razem, gdy użytkownik dopasuje zwierzęta.

Teraz w grze działa zegar zatrzymywany, gdy gracz zakończy dopasowywanie zwierząt. Po zakończeniu gry można kliknąć napis „Jeszcze raz?", aby zagrać ponownie.
Napisałeś swoją pierwszą grę w C#. Gratulacje!

Teraz w grze działa zegar informujący, ile czasu graczowi zajęło znalezienie wszystkich par. Czy potrafisz poprawić swój najlepszy czas?



Na stronie https://github.com/head-first-csharp/fourth-edition możesz przejrzeć i pobrać kompletny kod tego projektu, a także wszystkich innych programów z tej książki. Spolszczoną wersję kodu znajdziesz w witrynie wydawnictwa Helion.

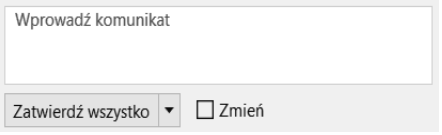
Kup książkę Poleć książkę

Aktualizowanie kodu w systemie kontroli wersji

Gra działa i jest gotowa. Jest to świetny moment, aby **wypchnąć zmiany do systemu Git**. Visual Studio umożliwia łatwe wykonanie tego zadania. Wystarczy *przygotować* zatwierdzenia, wprowadzić komunikat zatwierdzenia, a następnie zsynchronizować projekt ze zdalnym repozytorium.

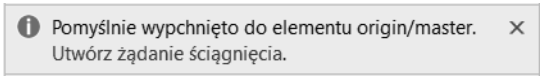
Możesz używać poleceń z menu Git do tworzenia nowych zatwierdzeń z najnowszymi zmianami w kodzie i wypychania ich do repozytorium systemu Git

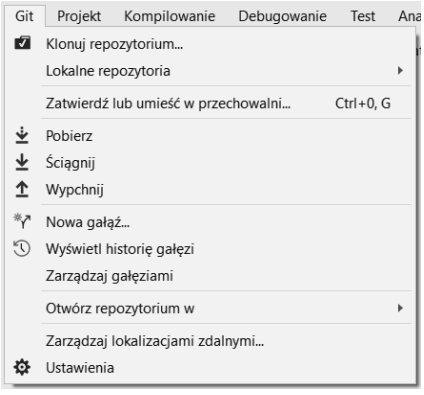
1. Wybierz opcję *Zatwierdź lub umieść w przechowalni (Ctrl+0, G)* z menu *Git*. Wprowadź **komunikat zatwierdzenia** z listą zmian.



2. Wciśnij **przycisk *Zatwierdź wszystko***. Visual Studio wyświetli informację o lokalnym utworzeniu zatwierdzenia.

3. Wybierz opcję ***Wypchnij*** z menu *Git*, aby wypchnąć zatwierdzenie do repozytorium. Po zakończeniu przesyłania kodu pojawi się informujący o tym komunikat.





Przesyłanie kodu do repozytorium w systemie Git jest opcjonalne, ale naprawdę warto to robić.

Przesyłanie kodu do repozytorium w systemie Git jest opcjonalne, ale naprawdę warto to robić.

Rozbicie gry na mniejsze fragmenty, którymi mogłem zająć się pojedynczo, było świetnym pomysłem.

Gdy pracujesz nad dużym projektem, zawsze warto podzielić go na mniejsze części.

Jedna z najprzydatniejszych umiejętności programistycznych dotyczy podziału dużych, skomplikowanych problemów na mniejsze i łatwiejsze fragmenty.

Na początku dużego projektu łatwo dać się przytłoczyć i pomyśleć: „No tak, to prawdziwy gigant”. Jeśli jednak uda Ci się znaleźć mniejszy fragment, nad którym możesz popracować, pozwoli Ci to zacząć. Po ukończeniu tego fragmentu możesz przejść do następnej niewielkiej części, i kolejnej, i jeszcze jednej. W trakcie rozwijania każdego fragmentu coraz lepiej będziesz poznawać cały projekt.

Kup książkę Poleć książkę

Gra będzie jeszcze lepsza, gdy...

Twoja gra jest całkiem niezła! Ale każdą grę (i prawie każdy program) można ulepszyć. Oto kilka rzeczy, które naszym zdaniem mogłyby pomóc poprawić grę:

- ★ Dodanie różnych gatunków zwierząt, aby nie powtarzać za każdym razem tych samych rysunków.
- ★ Zapisywanie najlepszego czasu gracza, aby mógł próbować go pobić.
- ★ Odliczanie czasu do zera (zamiast od zera), aby gracz miał ograniczoną ilość czasu na wykonanie zadania.

MINI

Zaostrz ołówek

Czy potrafisz wymyślić własne ulepszenia? To świetne ćwiczenie. Zastanów się przez kilka minut i zapisz przynajmniej trzy ulepszenia gry w dopasowywanie zwierząt.

Mówimy poważnie — poświęć kilka minut na wykonanie tego zadania. Zrobienie przerwy w lekturze i zastanowienie się nad ukończonym projektem to świetny sposób na utrwalenie zdobytych informacji.

CELNE SPOSTRZEŻENIA

- Visual Studio śledzi, ile razy metoda jest **wywoływana** w kodzie C# i XAML.
- **Procedura obsługi zdarzeń** to metoda, którą aplikacja wywołuje po zajściu określonego zdarzenia (takiego jak kliknięcie myszą, wciśnięcie klawisza lub zmiana wielkości okna).
- IDE ułatwia **dodawanie i kontrolowanie** procedur obsługi zdarzeń.
- **Okno *Lista błędów*** w IDE wyświetla błędy uniemożliwiające kompilację kodu.
- **Zegary** w kółko wykonują procedury obsługi zdarzeń `Tick` w określonych odstępach czasu.
- Pętla **`foreach`** iteracyjnie przetwarza elementy z kolekcji.
- Gdy program zgłasza **wyjątek**, zbierz dowody i spróbuj ustalić przyczynę problemu.
- Wyjątki są łatwiejsze do wyeliminowania, jeśli są **powtarzalne**.
- Visual Studio bardzo ułatwia używanie **systemu kontroli wersji** do archiwizowania kodu i śledzenia wszystkich wprowadzonych zmian.
- Możesz przesłać kod do **zdalnego repozytorium systemu Git**. Repozytorium z kodem źródłowym wszystkich projektów z tej książki znajduje się w serwisie GitHub.

Krótkie przypomnienie: w tej książce zamiast nazwy Visual Studio często używamy określenia IDE.

Kup książkę Poleć książkę

Skorowidz

.NET, 2
.NET Core, 2, 86

A

abstrakcja, 404
akcesor
  get właściwości, 254
  set właściwości, 254
akcja Użyj niejawnego typu, 495
aktualizowanie
  kodu, 47
  kontrolek, 85, 399
albedo, 354
alias, 161, 612
alokowanie zasobów, 545
analiza
  hierarchii klas, 322
  przepływu danych, 610
animacja, 217
API, application programming interface, 193
aplikacja
  gra Hi-Lo, 244
  gra w karty, 406
  kalkulator obrażeń, 231
  losowe pozycje menu, 209
  obliczanie obrażeń, 229
  PickACardUI, 118
  PickRandomCards, 106, 107
  porządkująca kolekcję, 497
  przeglądarka danych szesnastkowych, 573
  serializacja obiektów, 556
  symulacja działania zoo, 280
  symulacja rzutu kostką, 207
  system zarządzania rojem, 316–333, 400
  testowanie klasy PaintballGun, 253
  użycie kolejki, 449
  użycie kwerendy LINQ, 478
  użycie listy, 417
  użycie słownika, 444
  WebAssembly platformy Blazor, 670, 687, 707, 714, 720
  WPF PickACardUI, 120
  z testami jednostkowymi, 502
aplikacje
  biznesowe, 315
  desktopowe, 2, 270
  konsolowe, 2, 29, 50, 86
  sieciowe, 674
  sieciowe Blazor, 664, 697, 719
  WPF, 3, 6, 230
argument, 172, 173, 606
arytmetyka wektorów, 659
asercje, 504
atrybut
  DataContext, 401
  Text, 401
automatyczna
  aktualizacja kontrolek, 399
  konwersja, 168, 171
automatyczne
  implementowanie
    interfejsu, 388
    właściwości, 256, 271, 373
  tworzenie właściwości, 266
  zapisywanie, 74

B

bajt, 165, 572
bezpieczeństwo, 246, 252
bezpieczna konwersja typów, 381
bit, 165
Blazor, 664
blok
  catch, 638, 647, 650
    ogólny, 637, 638
    z komentarzami, 648
  finally, 636, 650
  try, 650
  try/catch, 634, 647
  try/finally, 645
błąd, 32
  DivideByZero, 627
  kompilacji, 178, 228, 244, 246, 260, 286, 298, 304, 307, 311, 334, 337, 361, 384, 385, 391, 408, 441, 498, 517, 634
  składniowy, 727
  systemu plików, 546
błędna
  nazwa pliku, 624
  sekwencja ucieczki, 547
błędne dane wejściowe, 406
błędy
  lista, 32
  usuwanie, 696
  trudne do wykrycia, 339
  w kodzie, bugi, 235, 693
Bootstrap, 719

C

C#, 1
Canvas, 458, 466
CIL, Common Intermediate Language, 193, 212
CLI, Common Language Infrastructure, 193
CLR, Common Language Runtime, 212, 271
czcionka, 122

D

debuger, 43, 61, 86, 693
  obserwowanie zmian zmiennych, 60
  uruchamianie gry, 684
debugowanie, 5, 23, 67, 182, 199, 218, 298, 313, 377
  blok finally, 636
  blok try-catch, 635
  gry, 684
  kodu obsługi zdarzeń, 694
deklaracje zmiennych, 55, 56
deserializacja, 552–555
destruktor, Patrz finalizator
diagram klas, 107, 131, 141, 154, 283
dokumentacja kodu, 139
  XML, 140
domyślne implementacje, 394, 396
dostęp
  do metod w podklasie, 376
  do pól, 187
  do składowych, 240
  tylko do odczytu, 244
dynamika gry, 314, 342
dyrektywa, 610
  using, 50, 398, 470, 473, 485
działanie właściwości, 255
dziedziczenie, 273, 277–281, 285, 289, 291, 341, 404
  interfejsów, 388, 390

E

edytor
  kolekcji, 81
  scen, 88
  skryptów Unity, 90
Eksplorator rozwiązań, 10
estetyka gry, 264

F

FIFO, first in, first out, 446
filtr wyjątków, 646, 650
finalizator, 592–597
  uruchamianie, 593
format
  JSON, 556–560
  szesnastkowy, 572
  Unicode, 561, 563, 566
formatowanie złożone, 541
formatujące łańcuchy znaków, 541
funkcja zmiany nazw, 499

G

GameObject, 93
generowanie
  kodu, 11
  liczb pseudolosowych, 27, 206, 212, 243, 685, 723
  metody, 22, 58
  raportów, 324
generyczne kolekcje, 420
getter, 254, 257
Git, 30, 47, 688
gra
  Go Fish, 206
  Hi-Lo, 244
  matematyczna, 262
  RPG, 156
  stołowa, 206

Kup książkę Poleć książkę

graf obiektów, 556, 560
graficzny interfejs użytkownika, 13
grawitacja obiektu, 354
grupowanie, 488
gry
  dodawanie nowego trybu, 455, 456
  dodawanie zegara, 702
  dynamika, 314, 342
  estetyka, 264
  interfejs użytkownika, 458, 463
  mechanika, 70, 314
  menu nawigacyjne, 704
  ponowne uruchamianie, 460–462
  poprawianie, 48
  proces tworzenia, 671
  prototyp, 115
  rejestracja trybu, 457
  resetowanie ustawień, 461, 698
  skrypt do sterowania, 349
  testowanie, 562
  wyświetlanie wyniku, 459

H

hermetyzacja, 239, 245–253, 271, 404
  właściwości Name, 610
hierarchia klas, 284, 382
HTML, 12

I

IDE, integrated development environment, 3, 10
IDE Visual Studio, 668
identyfikator, 54
implementacja domyślna, 397
implementowanie
  interfejsu, 372
  właściwości, 373
import pliku z teksturą, 97
indekser, 443, 524
informacje diagnostyczne, 235, 630
inicjalizator, 65, 210
  kolekcji, 212, 426, 438
  obiektu, 148, 154
inicjowanie
  kontrolki, 78
  właściwości, 258, 259, 271
instancja, 128, 129, 133, 154
instancje obiektów gry, 343
instrukcja, 51, 55
  Debug.WriteLine, 250
  if, 63, 173
  if/else, 63
  new, 128
  return, 105, 109, 124, 719
  switch, 275, 289, 294, 419, 517
  using, 546
  yield return, 523–526
instrukcje warunkowe, 55
IntelliSense, 21, 24, 398, 676
interfejs, 355, 358–360, 364, 387, 393, 439
  IClown, 361
  ICollection<T>, 421, 422
  IComparable<T>, 429, 430, 438
  IDefender, 358, 359
  IDisposable, 545, 645, 647
  IEnumerable<T>, 448, 472, 524, 525
  IEnumerator, 522
  INotifyPropertyChanged, 399–402
  IPuller, 387
  IScaryClown, 397
  ISwimmer, 379
  IWorker, 370–374
interfejsy
  dziedziczenie, 388, 390
  implementacje
    automatyczne, 388
    domyślne, 394, 396
  na platformie .NET Core, 398
  nazwy, 360
  publiczne, 373
  rozszerzanie, 389, 390
  rzutowanie, 386
  składowe statyczne, 395
  użytkownika, 3, 19, 29, 319, 453, 458
    mechanika, 71
  używanie, 392
interpolacja łańcuchów znaków, 235, 250

J

język
  C#, 1
  CIL, 212
  HTML, 12
  XAML, 3

K

kalkulator obrażeń, 231, 232
kamera, 579
  przesuwanie, 100
  sterowanie, 656
karty postaci, 157
kata z programowania, 725
katalog
  Materials, 653
  Shared, 704
katalogi
  pobieranie listy plików, 542
  tworzenie, 542
  usuwanie, 542
klasa, 50, 86, 124, 614
  Assert, 503
  BetterSportSequence, 524
  BinaryReader, 570, 576
  BinaryWriter, 569, 576
  Card, 406, 410
  CardPicker, 110, 120, 714
  CashRegister, 144
  CharacterSheet, 157
  Clown, 135
  CollectionAssert, 505
  ComicAnalyzer, 498
  Console, 237, 541
  Convert, 165, 171
  CryptoStream, 538
  Dictionary<TKey, TValue>, 442
  Directory, 542–544, 551
  DispatcherTimer, 330
  Duck, 429
  Encoding, 574
  Enumerable, 521, 526
  File, 542–544, 551, 571
  FileInfo, 542, 551
  FileStream, 532, 551, 571
  GC, 591
  Guy, 146, 147
  HiLoGame, 244
  JsonSerialization, 556
  List<T>, 414, 419–422, 438
  ManualSportSequenca, 524
  MemoryStream, 547, 548
  MoveToClick, 584
  OneBallBehaviour, 351, 464
  PaintballGun, 253
  Queen, 400
  Queue<T>, 446, 451
  Random, 131, 206, 211, 347, 354, 723
  Stack<T>, 447, 451
  Stopwatch, 592
  Stream, 531
  StreamReader, 537, 551, 571, 573, 576
  StreamWriter, 533, 535, 536, 541, 551, 571, 576
  SwordDamage, 228–230, 238, 251, 268
  System.Console, 250
  System.Diagnostics.Debug, 250
  System.Exception, 629
  Timer, 702
klasy, 50, 86, 124, 614
  abstrakcyjne, 303, 334, 339, 342, 365
  bazowe, 278–281, 289
  dziedziczenie, 278
  generyczne, 420
  hermetyzacja, 253
  modyfikowanie, 618
  rozszerzanie, 286, 289
  spełnianie obietnic, 355
  tworzenie, 107
  właściwości, 254
  wyjątków, 629
klauzula
  from, 475, 479, 484, 485
  group...by, 488–490, 499
  join, 491, 493, 496, 499
  orderby, 475, 479, 485
  select, 475, 479, 485
  where, 475, 479, 485, 515
kod
  dokumentacja, 139
  kontrolki
    C#, 77
    XAML, 76, 119
  mało czytelny, 138
  obsługi zdarzeń, 692, 694
  opakowanie, 546
  ponowne wykorzystanie, 104, 114
  refaktoryzacja, 142
  ryzykowny, 641
  zwięzły, 138
kodowanie znaków, 563
  ASCII, 571
  Unicode, 563–566, 571, 576
  UTF-16, 564, 567
  UTF-8, 564, 567
kolejka, Queue, 445, 446

Kup książkę Poleć książkę

kolejność wywoływania metod, 236
kolekcja, 405, 420, 425
  kolejka, Queue, 445, 446
  lista, List, 25, 413, 418, 419, 438
  słownik, Dictionary, 442
  stos, Stack, 447
  tablica, 411
kolekcje
  generyczne, 420, 422, 425
  inicjalizatory, 426
  kwerendy, 470
  obiektów typu Comic, 469
komentarze, 33, 69, 139
  dokumentacji XML, 140
  wielowierszowe, 69
komparator, 432
kompilator C#, 178
komponent, 101
  Audio Listener, 101, 579
  Ball Behaviour, 215
  Camera, 101, 579, 585
  Light, 101
  Move To Click, 584
  NavMesh Agent, 581, 583, 585, 586, 652
  NavMesh Obstacle, 658, 660
  Script, 221, 466
  Transform, 95, 101, 579, 580, 586
komponenty
  materiału, 93
  resetowanie, 95
  skryptu, 93
  transformacji, 93, 101, 102
komunikat o błędzie, 178, 260, 292
konfigurowanie projektu, 4, 8
konsola debugowania, 5, 112
konstruktor, 234, 237, 250, 258, 263
  bezparametrowy, 271
  obsługa wyjątków, 641
  podklasy i klasy bazowej, 308, 342
  prywatny, 263, 271
  z parametrami, 259, 271
kontekst danych, 399
kontener StackPanel, 119
kontrakt, 378, 387
kontrolka, 71, 86, 399, 706
  Button, 12
  CheckBox, 231
  ComboBox, 81, 319
  ListBox, 80
  ListBoxItem, 319
  RadioButton, 72, 80
  Slider, 83
  StackPanel, 12, 121
  TextBlock, 12, 18, 76, 77, 78, 209
  TextBox, 75, 319
  TextBoxstatusReport, 402
  XAML Grid, 29
kontrolki, 71, 86, 706
  aktualizacja automatyczna, 399
konwersja typów, 185, 381
  automatyczna, 171
  jawna, 168
  łańcucha znaków, 541
  niejawna, 440
kopiowanie
  kolejki do listy, 448
  przez referencję, 604
  przez wartość, 604
kowariancja, 440, 441
kula, 216
kwerenda, 485
  grupująca, 488, 490
  złączająca, 491, 493, 499
kwerendy
  do kolekcji, 470
  LINQ, 475, 487, 515

L

lambda, 508
liczba klatek na sekundę, FPS, 217
liczby
  całkowite, 159
  losowe, 27, 207
  rzeczywiste, 160
  zmiennoprzecinkowe, 160, 185
LIFO, last in, first out, 447
LINQ, Language-Integrated Query, 467–470, 486, 514, 618
  kwerendy, 487
  łańcuchy metod, 471
  metody, 473, 474
  obiekty, 477
  sekwencje danych, 470
  składnia kwerend, 475
lista, List, 413–416, 422
  deklaracja, 415
  dodawanie elementu, 414
  rzutowanie w górę, 440, 441
  sortowanie, 428
  sortowanie obiektów, 431
  sprawdzanie ilości elementów, 414
  sprawdzanie określonego elementu, 414
  tworzenie, 678, 680
  usuwanie elementu, 414
  zmienianie sposobu sortowania, 432
listy
  błędów, 32
  generyczne, 420
literał, 162
lukier składniowy, 645, 647

Ł

łańcuchy
  metod, 471, 476
    LINQ, 515
  znaków, 161, 166, 406
łączanie
  łańcuchów, 170, 171
  strumieni, 538

M

materiał, 96, 345
MDA, Mechanics-Dynamics-Aesthetics, 329
mechanika
  gry, 70, 314
  interfejsów użytkownika, 71
mechanizm
  Bake, 581
  IntelliSense, 21, 24, 398, 676
  odśmiecania pamięci, 590, 593, 597
  refleksji, 246
menu
  Debugowanie, 23
  Edycja, 38
  GameObject, 101, 220
  nawigacyjne, 704
  Okno, 11
  szybkich akcji, 22
  Widok, 9, 32
  Widok klas, 322
metoda, 51, 86, 124
  Add, 419, 422, 443
  AddABall, 455
  Array.Resize, 318
  Assert.AreEqual, 503, 505
  Average, 485
  Awake, 586
  BallBehaviour.Update, 225
  ButtonClick, 703
  Char.ToUpper, 275
  Clear, 446, 447
  ClickedOnBall, 456
  Close, 533, 541
  Compare, 430, 433
  CompareTo, 429, 430
  Concat, 471, 485
  Console.Error, 292, 634
  Console.ReadKey, 275
  Console.ReadLine, 111
  Console.Write, 111
  Console.WriteLine, 55, 111, 237, 436, 487
  Contains, 422
  Convert.ToByte, 171
  Convert.ToInt32, 171
  Convert.ToInt64, 171
  Convert.ToString, 171
  Debug.DrawRay, 222, 225, 226
  Debug.WriteLine, 235–237
  Dequeue, 446
  Destroy, 454
  Dictionary.ContainsKey, 451
  Dictionary.TryGetValue, 451
  Directory.CreateDirectory, 542
  Directory.Delete, 542
  Directory.GetFiles, 542
  Dispose, 545, 546, 595, 596, 647
  double.Parse, 625
  Encoding.UTF8.GetString, 547, 574, 576
  Enqueue, 446
  Enum.TryParse, 607
  Enumerable.Empty, 521
  Enumerable.Repeat, 521
  Environment.GetFolderPath, 541
  File.AppendAllText, 542
  File.Create, 542, 569
  File.OpenRead, 542, 574, 576
  File.OpenWrite, 542
  File.ReadAllBytes, 566
  File.WriteAllBytes, 566
  First, 485
  Flush, 548

Kup książkę Poleć książkę

GameControllerAddABall, 456, 457
GameObject.FindGame ObjectsWithTag, 466
GC.Collect, 591, 593, 595, 597
GC.GetTotalAllocatedBytes, 591
GC.GetTotalMemory, 591
GetEnumerator, 439, 524
GetReviews, 505, 520
GetType, 246
GroupBy, 516
GroupComicsByPrice, 520
IComparable.CompareTo, 438
IComparer.Compare, 438
IEnumerable<int>.Select, 514
Input.GetAxis, 660
Input.GetKey, 660
Input.GetMouseButtonDown, 583, 586
Input.mousePosition, 585
Instantiate, 354, 454
int.TryParse, 111, 585, 606
InvokeRepeating, 354
Last, 485
Length, 212
List.Sort, 429, 430, 438
Main, 50, 112, 179
Max, 485
MemoryStream.ToArray, 547, 548
Min, 485
MoveNext, 439
MoveToClick, 652
NavMeshAgent. SetDestination, 586, 652
numbers.Take, 471
numbers.TakeLast, 471
Object.ToString, 435
OneBallBehaviour.Update, 463
OnInitialized, 682
OnMouseDown, 454, 456
OnMouseDrag, 659
OrderBy, 515, 526
OrderByDescending, 516
Peek, 412, 446
Physics.Raycast, 585, 586
PickSomeCards, 108
Pop, 447
Push, 447
Random.value, 347, 463
Randomizer.Next, 211, 723
RandomSuit, 112
RandomValue, 112
Read, 531, 574
ReadAllBytes, 571
ReadAllLines, 571
ReadAllText, 571
ReadBlock, 573
ReadDouble, 180
ReadInt, 180
Remove, 443
RemoveAt, 419, 422
ResetBall, 463
ScreenPointToRay, 586, 652
Seek, 531
Serialize, 560
SetUpGame, 682
Shuffle, 489
Skip, 485
Sort, 428, 438
Start, 583
StartGame, 460
Stream.Read, 574
StreamReader.ReadBlock, 576
String.Join, 486
String.Split, 549
String.Substring, 573
Take, 485
TakeLast, 485
TextBlock_MouseDown, 36, 37
Timer_Tick, 41
ToArray, 425
ToList, 441
ToString, 212, 407, 435–438
transform.Rotate, 217, 224, 226
transform.RotateAround, 226
transform.Translate, 463, 466
TryParse, 607
Type.GetFields, 246
Update, 226
UpdateValue, 709
Where, 515, 526
Write, 531–533, 551, 569
WriteAllBytes, 571
WriteAllLines, 571
WriteAllText, 567, 571
WriteLine, 533, 534, 551
metody, 51, 86, 124
abstrakcyjne, 337, 338
klasy
Directory, 551
File, 551
GC, 591
Stream, 531
kolejność wywoływania, 236
LINQ, 473–475, 526, 618
łańcuchy LINQ, 471, 476, 515
przeciążanie, 409, 536
przesłanianie, 289, 292
rozszerzające, 617–619
sygnatura, 263
tworzenie, 107
ukrywanie, 302, 303
wirtualne, 303, 396
wyodrębnianie, 499
zwracanie wielu wartości, 606
model
klas, 279, 289, 317
klas na kartce, 342
MDA, 329
modulo, 185
modyfikator
internal, 507
out, 606
ref, 607
modyfikatory dostępu, 240

N

narzędzia do refaktoryzacji, 495
narzędzie
developer command prompt, 154
Move, 94
Move Gizmo, 102
przenoszenia, 102
Rotate, 99
Scene Gizmo, 100, 102
sceny, 100, 102
skalowania, 102
Transform, 94
nawias
klamrowy, 25, 54, 363
kwadratowy, 106, 200
ostry, 413, 420
nawigowanie po scenie, 651
nazwy klas i metod, 140

O

obiekt, 126, 133, 613
typu
FileStream, 531–533, 574
GZipStream, 531
List, 425
List<T>, 413
MemoryStream, 531, 548
NetworkStream, 531
Stream, 530
StreamWriter, 533, 548
string, 166
Text, 459
obiekty
błędne użycie, 238
finalizator, 592
gry, GameObject, 93, 101, 102
Button, 466
Canvas, 458
Cylinder, 580
Directional Light, 101
dodawanie tekstury, 97
Floor, 582
GameController, 585
Head, 580
komponenty, 93
Main Camera, 101, 579, 586
Moving Obstacle, 658, 659
obliczanie pozycji, 465
obliczanie szybkości, 463, 465
obracanie, 99, 216
Plane, 579, 586
Player, 580
prefab, 343, 348, 354
przesuwanie, 94
tworzenie instancji, 343
wyświetlanie komponentów, 95
inicjowanie, 148
jako typy referencyjne, 602
kolejność usuwania, 595
komparatora, 431, 434
komunikacja, 198
kopiowanie przez referencję, 604
metody, 127
podstawowe Unity, 92
przekształcanie, 384
stan, 554
strumieni, 548
tworzenie, 126–128, 335, 336, 590
usuwanie, 590–593
wyjątków, 629, 638
obsługa
kontrolek, 77, 84

Kup książkę Poleć książkę

obsługa
wyjątków, 623, 634
zdarzeń, 33, 38, 48, 79, 689–694, 705
odczyt, Patrz wczytywanie
odśmiecanie pamięci, 188, 189,193, 590–593, 605
okna
wielkość, 14
zmienianie wielkości, 15
zmienianie nagłówka, 15
okno
Anchor Presets, 459
Automatyczne, 78
Błędy, 697, 705
Color, 578
Dane wyjściowe, 235, 236
Eksplorator rozwiązań, 9
Eksplorator testów, 500, 503
External Script Editor, 90
External Tools, 216
Hierarchy, 102, 216, 353, 458, 580
Inspector, 95, 99, 352, 579, 466
IntelliSense, 678, 682
Interactive, 154, 161, 211
Klonuj repozytorium, 688
Konsola debugowania, 5
Lista błędów, 9, 10, 48
Lokalne, 44, 61, 628
Navigation, 581, 582, 654
NavMesh Display, 582
Podgląd zmian, 22
Podgląd znaków, 565
Preferences, 90
Project, 102, 215
Projektant, 9
Przybornik, 9, 11
Rozwiązanie, 668
Scene, 92
Terminal, 669
Tworzenie repozytorium Git, 31
Właściwości, 10, 15, 29
wyboru, 706, 713
Wyrażenia kontrolne, 61
opakowanie wartości, 612, 614, 616
opcja
3D Object/Cylinder, 220
3D Object/Sphere, 346
Browse..., 90
Create/Folder, 344
Create/Material, 345
Dane wyjściowe, 235
Dodaj/Klasa..., 107
Edytuj punkt przerwania..., 218
Import New Asset..., 97
Kompilowanie/Kompiluj rozwiązanie,, 38
Kontrola wersji/Opublikuj w kontroli wersji..., 688
Navigation Area, 654
Navigation Static, 582
Navigation/NavMesh Agent, 583
OnInitialized, 682
Physics, 353
Plik/Nowe rozwiązanie, 672
Preferences, 90
Przejdź do definicji, 421
Przejdź do deklaracji, 421
Przekrocz nad, 60
Przełącz punkt przerwania, 60
Reset, 95
Resetuj, 17
Resetuj układ, 11
Rozpocznij debugowanie, 23
Test/Uruchom wszystkie testy, 503
Testy, 502
Uruchamianie/Uruchom testy, 503
Ustaw następną instrukcję, 626
Usuń wszystkie punkty przerwania, 45
Utwórz nowy projekt, 4, 73
Walkable, 582
Wide, 91
Widok/Inne okna, 669
Widok/Testy, 500
wstępnego obliczania, 654
Wypchnij, 47
Zaimplementuj interfejs, 510, 526
Zatrzymaj debugowanie, 23, 67
Zatwierdź lub umieść w przechowalni, 47
Zmień nazwę, 489
Znajdź i zamień/Szybkie zamienianie, 38
operator, 59
?, warunkowy, trójargumentowy, 513, 526
?:, 513
??, 611
??=, 611
+=, 182
=, 183
=>, lambda, 477, 508, 512, 517, 526
porównania, 62
operatory
dwuargumentowe, 183
matematyczne, 55
logiczne, 62
relacyjne, 62
równości, 62
złożone przypisania, 183
ostrzeżenie, 610

P

panel
Projektanta, 15
Testy, 500
Wyniki testu, 503
Dynamiczne drzewo wizualne, 26
parametr, 105, 172, 173, 606
typu Func, 514
out, 606
parametry opcjonalne, 608
pętla, 55, 69
aktualizacji, 354
do/while, 64
for, 64, 65, 705, 712
foreach, 44, 422, 436, 475, 687, 696, 705
while, 64, 164
piksel, 19
platforma, 653, 654
.NET, 2
.NET Core, 2
Bootstrap, 719
testów jednostkowych, 526
plik
devenv.exe, 90
index.html, 704
Index.razor, 673, 682, 709, 717, 721, 722
MainWindow.xaml, 8
MainWindow.xaml.cs, 8, 21, 124
MyFirstConsoleApp.csproj, 51
NavMenu.razor, 704
Program.cs, 4, 50, 667
SampleScene.unity, 92
pliki
.dll, 54
.razor, 676
.sln, 51
dodawanie tekstu, 542
odczyt danych, 532, 537, 542
odczyt danych binarnych, 570
sprawdzanie, czy plik istnieje, 542
zapis danych, 532, 533, 536, 542
zapis danych binarnych, 569
zapis tekstu, 533
zwalnianie, 533
płótno, Canvas, 458, 466
Podgląd zmian, 22
podklasa, 283–285, 288, 289, 294
podręcznik, 139
do Unity, 658
podział obowiązków, 310, 342
pole, 133
kombi, 71, 72, 85
listy, 71, 80
na dane tekstowe, 710, 711
tekstowe, 71, 86, 706
XRotation, 221
polecenie HexDump, 575
polimorfizm, 403, 404
problem
rombu, 342
z liczbą 0.3, 184, 185
procedura obsługi zdarzeń, 34, 48, 79, 84, 705, 709, 711
programowanie
obiektowe, 404
abstrakcja, 404
dziedziczenie, 404
hermetyzacja, 404
polimorfizm, 404
sterowane testami, 527
projekt
gry, 6
konfigurowanie, 4, 8
system nawigowania, 578
tworzenie, 4, 8
zapisanie, 102
promień, 223
prototyp, 115

Kup książkę Poleć książkę

gry, 130
papierowy, 116
prowizorka, 393
prywatne pola i metody, 242
przeciążanie metod, 409, 438, 536
przeglądarka
internetowa, 674
danych szesnastkowych, 572–575, 624
przekazywanie
przez referencję, 607
przez wartość, 607
przekształcanie
obiektu, 384
referencji, 380
przesłanianie metod, 289, 292
ToString, 435, 438
przestawienie referencji, 195, 196
przestrzeń nazw, 51–54, 86, 164, 715
System.ComponentModel, 400
UnityEngine, 347
przesuwanie
kamery, 100
obiektów, 94
przeszkody, 659
przetwarzanie skrócone, 635
Przybornik, 10, 11
przycisk, 71, 712
opcji, 706
przyrostek
F, 173
M, 173
punkt
przerwania, 43, 45, 60, 182, 218, 705
wejścia, 54
pusty łańcuch znaków, 161

R

ray casting, 584, 586, 652
Razor, 687
refaktoryzacja, 142, 154, 340, 342, 495, 510, 511
refaktoryzowanie klasy ManualSportSequence, 524
referencja, 155, 186, 187, 200, 205, 212, 288, 590
null, 609
statyczna, 410
referencje
do interfejsu, 366
do metod ukrytych, 303
do obiektu, 188, 366, 369, 386, 392
do samego siebie, 199
do typów interfejsowych, 392
przekształcanie, 380
przypisywanie, 600
typów interfejsowych, 369
typu IEnumerable<T>, 440
typu interfejsowego, 392
refleksja, 246
rekurencja, 605
repozytorium GitHub, 48, 98
resetowanie
gry, 698
układu Wide, 223
reszta z dzielenia, 185
rotacja, 216, 224
kąt, 221
szybkość, 221
rozszerzanie
interfejsu, 389, 390
klas, 286, 289, 295
typu podstawowego, 619
GitHub for Unity, 98, 102
rzutowanie, 168, 169, 173, 178, 185, 357, 375, 408
w dół, 382–386, 393, 403
w górę, 382–387, 393

S

scena, 92, 652
dodawanie obiektu, 653
sekcja
@code, 721
Internet i konsola, 672
Tekst, 123
Typowe kontrolki WPF, 18
Układ, 15, 80, 123
Wspólne, 81, 123
sekwencja, 485, 523–525
ucieczki, 547, 567
serializacja, 552–560, 595
serwis GitHub, 31, 452, 527, 661, 688, 724
setter, 254, 257, 266
prywatny, 257
siatka, 16, 18, 121
nawigacyjna, 581, 582, 586, 652, 654
dodawanie obiektów, 655
powstawanie luk, 657
problem z wysokością, 657
wstępne obliczanie, 655, 660
silnik
fizyki, 353, 354
gry, 88
składnia select new, 499
składowe
abstrakcyjne, 334
klasy, 154
prywatne, 239
statyczne, 395
wirtualne, 337
skrót klawiaturowy
Alt+ Enter, 58
Alt+Enter, 178, 391
Alt+Tab, 61
Command+Tab, 697
Ctrl+., 58, 178
Ctrl+Alt+F9, 30
Ctrl+C, 61
Ctrl+D, 43, 154
Ctrl+E, 31, 61
Ctrl+G, 53
Ctrl+L, 43
Ctrl+O, 31, 47
Ctrl+S, 95
Ctrl+Shift+A, 120
Ctrl+Shift+F9, 45
skrót klawiaturowy
Ctrl+W, 235
Option+Return, 58
Option+Shift, 459
Shift+Alt+A, 120
Shift+F5, 67, 219
Shift+Tab, 680
Windows+kropka, 25
skrypt
GameController, 349–351
OneBallBehaviour, 351
skrypty C#, 214
słownik, Dictionary, 442
deklarowanie, 442
dodawanie elementu, 443
klucz i wartość, 443
nieistniejący klucz, 451
pobieranie listy kluczy, 443
usuwanie elementu, 443
wyszukiwanie wartości, 443
zliczanie par, 443
słowo kluczowe, 163
abstract, 336, 339, 342
as, 381, 403, 604
base, 306, 390
class, 164
const, 517
default, 275
descending, 478
else, 164
enum, 407
event, 400
for, 164
get, 271
if, 164
int, 719
interface, 358
internal, 507, 526
is, 375, 377–381, 385, 393, 403
join, 491, 493, 496
namespace, 40, 50, 164
new, 126, 128, 154, 164, 212, 302, 304, 342, 492
null, 203, 212
object, 161
out, 606
override, 292, 296, 299, 300, 304, 305, 342
partial, 52
private, 239, 241, 243, 271, 304
protected, 297, 304, 342, 392
public, 52, 241, 304
readonly, 391
ref, 607
return, 105
set, 271
static, 133, 134, 154, 211
struct, 601
switch, 275, 517
this, 198, 212, 271, 619
using, 164
value, 254
var, 480–485, 492, 494
virtual, 299, 300, 304, 305, 342
void, 719
while, 164
sortowanie, 428, 433, 434
obiektów, 431
sprzężenie zwrotne, 328
stała, 228, 322, 323, 326
stan obiektu, 554
standardowe wyjście błędów, stderr, 634
statyczna referencja, 410
sterowanie grą, 349

Kup książkę Poleć książkę

sterta, 136, 448, 593, 603, 604, 614
stos, 445, 447, 603–605, 614
struktura, 599, 601, 613, 614
  jako typ bezpośredni, 602
  kopiowanie przez wartość, 604
strumienie
  łączenie w łańcuch, 538
  odczyt danych, 531
  zapis danych, 531
  zmiana pozycji, 531
strumień, 530
  błędów, 237
  FileStream, 532
  MemoryStream, 547
  StreamReader, 550
  StreamWriter, 550
  wyjścia, 237
suwak, 72, 83, 706, 708
sygnatura
  metody, 263
  właściwości, 263
system
  czasu rzeczywistego, 330
  kontroli wersji Git, 30, 47, 688
  nawigowania i znajdowania drogi, 577, 581, 586
szybka akcja, 23, 495, 499, 524, 526

Ś

ścieżka do pliku, 534
ślad stosu, 504
śledzenie
  przepływu sterowania, 635
  stanu instancji, 133
środowisko
  3D, 92
  CLR, 189, 193, 205, 212, 590

T

tablica, 200, 212, 318, 411, 412, 416
  args, 575, 576
  bajtów, 541, 568
  łańcuchów znaków, 106
  obiektów, 204
  znaków, 565
tablice
  dodawanie elementu, 411
  usuwanie elementu, 412
technologia
  Blazor, 664
  LINQ, 467
tekstura, 97
test jednostkowy, 500, 503, 507, 526
  metody GetReviews, 505
  uwzględniający przypadki brzegowe, 506
testowanie, 253
  gier, 562
    indywidualne, 501
    ustrukturyzowane, 501
    wersja beta, 501
tryb
  debugowania, 26
  gry, 455, 456
  izometryczny, 660
  perspektywiczny, 660
  WYSIWYG, 53
tworzenie
  automatycznie implementowanej właściwości, 256
  gry, 7, 88, 671
  katalogu, 542
  klasy, 107
  listy, 413, 678, 680, 685
  materiału, 345, 653
  metody, 107
  obiektu, 126, 127, 335, 336
  obiektu typu List, 425
  okna wyboru, 713
  projektu, 4, 666
    Unity, 90, 578
    WPF, 73, 120
  prototypów, 115
  przycisków, 712
  referencji, 186, 366
  repozytorium Git, 30
  sekwencji, 522, 525
  strony z suwakiem, 708
  układu strony, 719
  wielu klas, 145
  własnych sekwencji, 523
  wyrażeń lambda, 508
  wyrażeń switch, 517
typ, 57
  bool, 158, 166
  byte, 159, 165, 168
  char, 166
  decimal, 160, 166, 173, 184
  double, 151, 158, 160, 166
  float, 158, 160, 166
  Guy, 151
  IEnumerable<T>, 521
  int, 105, 158, 165
  long, 159, 165
  Nullable<T>, 612
  Random, 151
  sbyte, 159
  short, 159, 165
  string, 59, 158, 166
  uint, 159
  ulong, 159
  ushort, 159
  void, 266
  wyliczeniowy, 408
typy
  anonimowe, 492, 494, 496
  bezpośrednie, 173, 601, 607
  bezpośrednie dopuszczające null, 612
  generyczne, 413
  obiektowe, 600
  parametrów, 172
  referencyjne, 601, 610
  referencyjne niedopuszczające null, 610
  wbudowane, 158
  zmiennej, 158
  zmiennoprzecinkowe, 173, 184
  zwracanych wartości, 719

U

układ
  okna, 209
  okna aplikacji, 121, 122
  strony, 677, 716, 719, 721
  Wide, 91
ukrywanie metod, 302, 303
Unicode, 563–566, 571, 576
unikanie wyjątków NRE, 610
Unity, 87, 213, 225, 343, 344, 453, 577, 651
  Collaborate, 102
  Hub, 89
uruchamianie
  aplikacji sieciowej, 674
  programu, 26
usuwanie
  błędu, 696
  obiektu, 592
utrata referencji, 197
używanie interfejsów, 360, 392

V

Visual Studio, 2, 10, 29
Visual Studio 2019 for Mac, 666
Visual Studio Code, 2
Visual Studio Community, 2
Visual Studio for Mac, 2, 663

W

wartości
  domyślne, 608
  kopiowanie, 600
wartość
  null, 203, 610, 611
  Time.deltaTime, 217, 219, 226
wcięcia wierszy, 680
wczytywanie
  bajtów ze strumienia, 574
  danych binarnych z pliku, 570
  danych z pliku, 531, 537
wektory trójwymiarowe, 222
wiązanie danych, 399–402, 717
widok
  dwuwymiarowy, 458
  sceny, 102
wiele referencji, 190, 197
wielkość czcionki, 122
wielodziedziczenie, 341
wiersz poleceń
  uruchamianie przeglądarki danych, 575
wiersze i kolumny, 16–19
właściwości, 271
  abstrakcyjne, 338
  automatycznie implementowane, 256, 271, 373
  tworzone, 266
  interfejsu, 374
  klas, 254
  sygnatury, 263
  tylko do odczytu, 257, 258, 266, 271, 323, 614
  tylko do zapisu, 266
  w interfejsie, 373
właściwość
  AutoToolTipPlacement, 83
  Balls, 255
  Checked, 84
  ColumnSpan, 41, 83
  Content, 80, 123
  Count, 422, 443
  Current, 439, 522
  gameObject, 462

Kup książkę Poleć książkę

Grid.Column, 20
Grid.Row, 20
Height, 17
IsEditable, 81
IsSnapToTickEnabled, 122
Length, 201, 609
List.Count, 419
Maximum, 83
Name, 409, 438
Row, 123
RowSpan, 123
StatusReport, 402
Text, 18, 319
TextBlock_MouseDown, 34
TextWrapping, 18
Title, 74, 123
Width, 15
WPF, Windows Presentation Foundation, 6, 29
wurażenia lambda, 508
wyjątek, 23, 42, 631
ArgumentOutOfRange, 45
FileNotFoundException, 638
IndexOutOfRangeException, 628
InvalidOperationException, 422
KeyNotFoundException, 451
NotImplementedException, 642
NullReferenceException, 609, 627
System.DivideByZeroException, 625, 627
System.FormatException, 625, 626
System.IO.FileNotFoundException, 624
System.lnvalidCastException, 625
System.NullReferenceException, 625
System.OverflowException, 625, 626
System.UnauthorizedAccessException, 632
UnauthorizedAccessException, 634, 635
wyjątki, 631
bloki try-catch, 634
filtry, 646, 650
generowanie, 626
nieobsłużone, 638, 641
niestandardowe, 642
odpowiednie do sytuacji, 642
ponowne zgłaszanie, 641
rozwiązywanie problemu, 630
ryzykowne metody, 633
tymczasowe rozwiązania, 649
typu System.Exception, 637
ukrywanie, 648
w finalizatorze, 597
węzeł $exception, 628
wyliczenie, 405–410, 425, 438, 472, 522, 526
wyodrębnianie metod, 499
wyrażenia lambda, 509–517
refaktoryzacja składowych, 511
WYSIWYG, 53
wyświetlanie informacji o obiekcie, 436
wywołanie konstruktora klasy bazowej, 307

X

XAML, eXtensible Application Markup Language, 3, 11–13, 32

Z

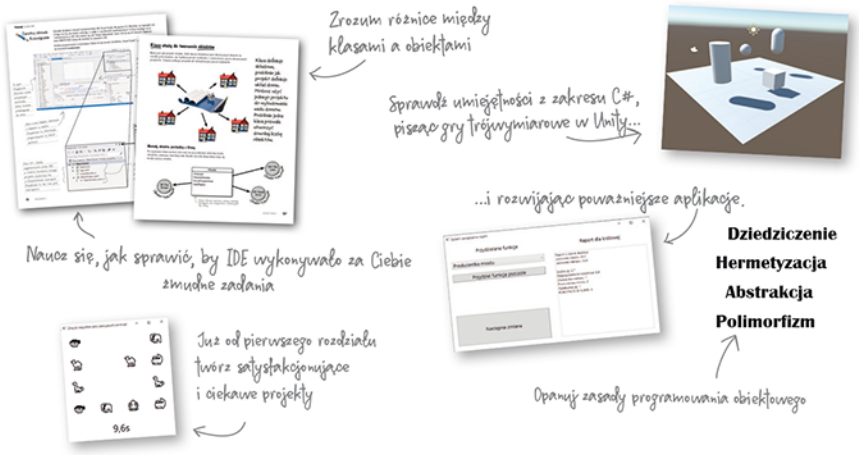
zachowania emergentne, 342
zakładka
Scene, 91
Zabezpieczenia, 632
zależność, 502
zaokrąglanie w dół, 178, 185
zapis
danych, 518, 531
binarnych w pliku, 569
w pamięci, 547
w pliku, 532, 535
obiektów w plikach, 552
szesnastkowy, 572
tekstu w pliku, 533
zarządzanie zasobami, 327
zasoby, assets, 215
zastępowanie metody właściwością, 254
zdarzenia myszy, 33
zdarzenie, 690
ButtonClick, 696
Change, 709
Click, 400, 401, 718
MouseDown, 38, 41
PreviewTextInput, 79, 86
PropertyChanged, 400, 402
SelectionChanged, 85
TextChanged, 79
Tick, 400, 703
zegar, 39, 46, 331, 701–705
zintegrowane środowisko programistyczne, IDE, 3
złączenia, 491
zmienne, 55, 56, 86, 165
logiczne, 59
referencyjne, 186, 201
tablicowe, 212
this, 205
typu obiektowego, 186
typu string, 59
znacznik, 11, 466
, 681
<Grid>, 82, 401
<ListBoxItem>, 82
<style>, 681
<TextBox>, 401
ColumnDefinition, 16
HTML, 680
input, 708
kolejności bajtów, 571
RowDefinition, 16
StackPanel, 121
TextBlock, 18
XAML TextBlock, 29
znak
#, 610
$, 250, 396, 397
?, 610, 612
@, 163, 396, 397, 547, 677, 687
dwukropka, 290, 358
lewego ukośnika, 541
myślnika, 83
równości, 600, 601
średnika, 69
ukośnika, 69
znaki
końca wiersza, 551
Unicode, 564, 571
zrzut szesnastkowy, 632

Kup książkę Poleć książkę

PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

C#: rusz głową, programuj i ciesz się kodem!

C# jest dziś uważany za jeden z najważniejszych języków programowania. Nowoczesny, wszechstronny, dojrzały i sukcesywnie rozwijany, zapewnia efektywne tworzenie kodu wysokiej jakości. Profesjonalni programiści, którzy pisali już kod w C++ lub Javie, bardzo często wybierają właśnie C#. Nic nie stoi na przeszkodzie, aby był pierwszym językiem programowania przyszłego programisty, który przy okazji nauki chce rozwijać pasję i świetnie się bawić!



Ta książka, podobnie jak inne pozycje z serii *Rusz głową!*, została przygotowana zgodnie z najnowszymi odkryciami nauk poznawczych, teorii uczenia się i neurofizjologii. Oznacza to tyle, że dzięki niej zaangażujesz swój mózg, wykorzystasz wiele zmysłów i niepostrzeżenie przyswoisz język C# 8.0 i umiejętność pracy w Visual Studio 2019. Naukę programowania rozpoczniesz od napisania działającej gry. Później dowiesz się, jak używać klas, czym jest programowanie obiektowe, jak tworzyć gry trójwymiarowe w Unity i jak korzystać z technologii LINQ. Będziesz się tego uczyć, rozwiązując łamigłówki, wykonując praktyczne ćwiczenia i pisząc aplikacje. Ani się obejrzysz, a staniesz się znakomitym, gotowym na dalsze wyzwania programistą C#!

Wielkie dzięki! Wasze książki pomogły mi w rozpoczęciu kariery

– Ryan White
programista gier

Andrew i Jennifer napisali zwięzłe, rzetelne, a przede wszystkim ciekawe wprowadzenie do programowania w C#

– Jon Galloway
starszy menedżer programu w zespole .NET Community, Microsoft

Jeśli chcesz dokładnie poznać C# i mieć z tego przyjemność, to trzymasz w ręku właściwą książkę

– Andy Parker
początkujący programista C#

Andrew Stellman pisał oprogramowanie dla finansistów, był wiceprezesem banku i zarządzał zespołami programistów.

Jennifer Greene zajmowała się jakością oprogramowania i prowadziła ciekawe projekty. Oboje są inżynierami, konsultantami i autorami książek. Współpracują od lat: piszą o programowaniu, doradzają firmom i występują na konferencjach dla inżynierów, architektów i menedżerów.

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

Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-7900-8
9 788328 379008
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 129,00 zł