

# Head First



# Rusz głową!

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

Andrew Stellman Jennifer Greene





Mydonie L

#### Tytuł oryginału: Head First C#: A Learner's Guide to Real-World Programming with C# and .NET Core, 5<sup>th</sup> Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-289-2113-9

 $\ensuremath{\mathbb{C}}$  2025 Helion S.A.

Authorized Polish translation of the English edition of *Head First, C# 5E* ISBN 9781098141783 © 2024 Andrew Stellman and Jennifer Greene.

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.

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.

Drogi Czytelniku! Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres *helion.pl/user/opinie/cshru5* Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem: https://ftp.helion.pl/przyklady/cshru5.zip

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

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#. Zbuduj coś wspaniałego szybko!	1
2.	Zmienne, instrukcje i metody. Zanurz się w kod C#	65
	Ćwiczenia z Unity nr 1. Poznawaj C# z Unity	111
3.	Przestrzenie nazw i klasy. Organizowanie kodu	127
4.	Dane, typy, obiekty i referencje. Zarządzanie danymi aplikacji	189
	Ćwiczenia z Unity nr 2. Pisanie kodu C# w Unity	257
5.	Hermetyzacja. O tym, jak obiekty strzegą swoich tajemnic	271
6.	Dziedziczenie. Drzewo genealogiczne obiektów	325
	Ćwiczenia z Unity nr 3. Instancje obiektów gry	403
7.	Interfejsy, rzutowanie i instrukcja "is". Spełnianie obietnic przez klasy	415
8.	Wyliczenia i kolekcje. Porządkowanie danych	473
	Ćwiczenia z Unity nr 4. Interfejsy użytkownika	539
9.	LINQ i lambdy. Kontroluj swoje dane	553
10.	Odczyt i zapis plików. Zachowaj dla mnie ostatni bajt	621
	Ćwiczenia z Unity nr 5. Ray casting	673
11.	Kapitan Wspaniały. Śmierć obiektu	687
12.	Obsługa wyjątków. Gaszenie pożarów robi się nudne	731
	Ćwiczenia z Unity nr 6. Nawigowanie po scenie	763
	Skorowidz	773

### 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 jazda na snowboardzie 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
Metapoznanie	xxxiii
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	xxxv
Przeczytaj to	xxxvi
Zespół redaktorów merytorycznych	xxxviii
Podziękowania	xl

Anie

## Zacznij pisać programy w jezyku C# Zbuduj coś wspaniałego... szybko

### Chcesz tworzyć fantastyczne aplikacje... od razu?

C# to nowoczesny język programowania i cenne narzędzie, które masz na wyciągniecie ręki. Ponadto w postaci Visual Studio otrzymujesz fantastyczne środowisko programistyczne z wysoce intuicyjnymi funkcjami, które sprawiają, że pisanie kodu jest niezwykle łatwe. Visual Studio jest nie tylko znakomitym środowiskiem do pisania kodu, ale też bardzo przydatnym narzędziem do nauki i eksplorowania języka C#. Brzmi nieźle? Pora zabrać się za programowanie!

							Urucha
							Utwórz
AnimalMatchingGame					- 0	×	W trolz
Gra w dop	asowywa	anie zwi	erząt				Willak
Czas: 0	0 s						visual
	No.	Ð	No.	L			Tworze w Visua
	The a	Sn	the a	7885			Konfig
		1	4.5	30			nostopr
			177	150			następi
							Napiszi
	d'	-	÷Q.	-			Tworze
	d. 6	-	and.				Urucha
	(T)	-	2	6 n			A 1:1
							Арнкас
							Roznoc

Poznaj C# i dowiedz się, jak zostać świetnym programistą	2
Pisz kod i poznawaj C# z Visual Studio	3
Instalowanie Visual Studio Community Edition	4
Uruchamianie Visual Studio	5
Utwórz i uruchom pierwszy projekt w C# w Visual Studio	6
W trakcie lektury C#. Rusz głową możesz używać Visual Studio Code	12
Tworzenie i uruchamianie pierwszego projektu w Visual Studio Code	14
Konfigurowanie Visual Studio Code na potrzeby następnego projektu	17
Napiszmy grę!	18
Tworzenie projektu .NET MAUI w Visual Studio	22
Uruchamianie nowej aplikacji .NET MAUI	24
Aplikacje MAUI działają na wszystkich urządzeniach	25
Rozpocznij edycję kodu XAML	27
Użyj układu FlexLayout, aby utworzyć siatkę przycisków ze zwierzętami	34
Napisz kod C#, aby dodać zwierzęta do przycisków	38
Uruchom aplikację!	46
Visual Studio ułatwia korzystanie z systemu Git	51
Dodaj kod C# do obsługi kliknięć myszą	52
Dodaj zegar do kodu gry	60
Dokończ kod gry	62



## Zmienne, instrukcje i metody Zanurz się w kod C#

### 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.

. . .

Wybierz gatun	ek ptaka	Wybrane gatunki
Sowa	~	Kaczka
		Struś
		Gołąb
		Kaczka
		Sowa
		Sowa



Przyjrzyj się plikom aplikacji konsolowej	66
Instrukcje są cegiełkami do tworzenia aplikacji	68
Instrukcje znajdują się w metodach	69
Metody używają zmiennych do pracy z danymi	70
Generowanie nowej metody używającej zmiennych	72
Dodaj do metody kod używający operatorów	73
Używanie debugera do obserwowania zmian zmiennych	74
Używanie fragmentów kodu do pisania pętli	76
Używanie operatorów do pracy ze zmiennymi	77
Instrukcje if służą do podejmowania decyzji	78
Pętle powtarzają wykonywanie operacji	79
Mechanika interfejsów użytkownika zależy od kontrolek	88
Inne kontrolki, których będziesz używać w tej książce	89
Utwórz nową aplikację do eksperymentów z kontrolkami	91
Zapoznaj się z nową aplikacją MAUI i zobacz, jak działa	92
Dodaj do aplikacji kontrolkę Entry	96
Dodaj właściwości do kontrolki Entry	97
Spraw, aby kontrolka Entry aktualizowała kontrolkę Label	<b>9</b> 8
Łączenie układów poziomych i pionowych	103
Dodaj kontrolkę Picker do wyświetlania listy pozycji do wyboru	104



# **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. W tych ćwiczeniach zaczniesz stosować w praktyce wiedzę, jaką zdobyłeś w rozdziałach 1. i 2.

Unity jest rozbudowanym narzędziem do projektowania gier	112
Pobieranie Unity Hub	113
Używanie Unity Hub do tworzenia nowego projektu	114
Przejmij kontrolę nad układem Unity	115
Sceną jest środowisko 3D	116
Gry Unity są tworzone za pomocą obiektów gry (GameObject)	117
Używanie narzędzia przenoszenia do przesuwania obiektów gry	118
Okno Inspector wyświetla komponenty obiektów gry	119
Określ materiał kuli	120
Obracanie kuli	123
Bądź kreatywny!	126

![](_page_5_Picture_4.jpeg)

![](_page_5_Picture_5.jpeg)

## Przestrzenie nazw i klasy Organizowanie kodu

### Świetni programiści dbają o organizację kodu i danych.

Jaką pierwszą rzecz robisz podczas tworzenia aplikacji? Myślisz o tym, **co ma robić**, niezależnie od tego, czy rozwiązujesz problem, tworzysz grę, czy po prostu się bawisz. Ale nie zawsze jest oczywiste, jak poszczególne instrukcje wpasowują się w ogólny obraz aplikacji. I właśnie w tym miejscu pojawiają się **klasy**. Pozwalają one **organizować kod** na podstawie tworzonych funkcji i rozwiązywanych przez aplikację problemów. Klasy mogą również pomóc w **organizowaniu danych**, ponieważ umożliwiają tworzenie **obiektów** reprezentujących dowolne "rzeczy" potrzebne w aplikacji. Projektowane klasy stanowią "wzorce" dla obiektów używanych w aplikacji.

Klasy pomagają organizować kod	128
Niektóre metody przyjmują parametry i zwracają wartość	130
Napisz program, który wybiera losowe karty	132
Tworzenie aplikacji z metodą Main	134
Użyj szybkich akcji, aby usunąć niepotrzebne wiersze ze słowem using	138
Zmiana stylu zapisu przestrzeni nazw	139
Używanie słowa kluczowego new do tworzenia tablicy łańcuchów znaków	140
Przygotuj papierowy prototyp klasycznej gry	148
Zbuduj wersję MAUI aplikacji do wybierania losowych kart	150
Ponownie wykorzystaj klasę CardPicker	154
Dodawanie dyrektywy using w celu użycia kodu z innej przestrzeni nazw	155
Klasa służy do tworzenia obiektów	159
Lepsze rozwiązanie dla Ani — wszystko dzięki obiektom	161
Instancja używa pól do śledzenia stanu	165
Używaj zrozumiałych nazw klas i metod	172
Zbuduj klasę reprezentującą ludzi	178
Użyj okna C# Interactive lub csi do uruchamiania kodu C#	188

![](_page_6_Picture_5.jpeg)

![](_page_6_Picture_8.jpeg)

![](_page_6_Picture_9.jpeg)

## Dane, typy, obiekty i referencje Zarządzanie danymi aplikacji

![](_page_7_Picture_2.jpeg)

### Dane i obiekty to cegiełki aplikacji.

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**, zobaczysz, jak pracować z danymi w programach, a także dowiesz się kilku nowych rzeczy na temat **obiektów** (i wiesz co? — obiekty także są danymi!).

Podręcznik dla mistrzów gry
Jacek Jacek

Typ zmiennej określa, jakiego rodzaju dane może ona przechowywać	192
C# udostępnia kilka typów do przechowywania liczb całkowitych	193
Porozmawiajmy o łańcuchach znaków	195
Literał to wartość bezpośrednio zapisana w kodzie	196
Rzutowanie umożliwia kopiowanie wartości, których C# nie potrafi automatycznie przekształcić na inny typ	202
C# niektóre konwersje przeprowadza automatycznie	205
Używaj zmiennych referencyjnych, aby uzyskać dostęp do obiektów	222
Referencje przypominają karteczki samoprzylepne na obiektach	223
Wiele referencji i ich efekty uboczne	226
Dwie referencje oznaczają DWIE zmienne, które mogą modyfikować dane tego samego obiektu	233
Obiekty komunikują się między sobą za pomocą referencji	234
Tablice przechowują wiele wartości	236
null oznacza referencję, która nic nie wskazuje	241
Gdy łańcuch znaków może mieć wartość null, użyj typu string?	243
Witaj w Budżetowych Kanapkach Jarka Niechluja!	246
Kontrolka Grid	248
Utwórz aplikację do generowania menu dla Jarka Niechluja i przygotuj siatkę	250
Użyj metody SetValue do zmiany właściwości semantycznych kontrolki	256

# 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	258
Dodaj skrypt C# do obiektu gry	259
Napisz kod C# do rotowania kuli	260
Dodaj punkt przerwania i zdebuguj grę	262
Użyj debugera, aby zrozumieć działanie wartości Time.deltaTime	263
Dodaj walec, aby zobaczyć, gdzie znajduje się oś Y	264
Dodaj do klasy pola określające kąt i szybkość rotacji	265
Użyj wywołania Debug.DrawRay, aby zbadać działanie	
wektorów trójwymiarowych	266
Uruchom grę, aby zobaczyć promień w widoku Scene	267
Rotowanie bili względem punktu sceny	268
Użyj Unity, aby przyjrzeć się rotacji i wektorom	269
Bądź kreatywny!	270

![](_page_8_Picture_4.jpeg)

Kup ksi k

## Hermełyzacja O tym, jak obiekty strzegą swoich tajemnic

### 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. Ponadto zabezpieczysz ważne dane obiektu przed **wyciekaniem do innych obiektów**, aby nie zostały przypadkowo niewłaściwie wykorzystane.

![](_page_9_Picture_4.jpeg)

![](_page_9_Picture_5.jpeg)

- PŁONĄCE MIECZE POWODUJĄ DODATKOWE 2PO.

Pomóż Oskarowi w definiowaniu rzutów obrażeń	272
Napisz aplikację konsolową do obliczania obrażeń	273
Zaprojektuj wersję MAUI aplikacji z kalkulatorem obrażeń	275
Użyj metody Debug.WriteLine do wyświetlania informacji diagnostycznych	281
Użyj hermetyzacji do kontrolowania dostępu do metod i pól klasy	286
Prywatne pola i metody są dostępne tylko w instancjach tej samej klasy	288
Po co jest hermetyzacja? Potraktuj obiekt jak czarną skrzynkę	293
Zastosuj hermetyzację, aby ulepszyć klasę SwordDamage	297
Napisz aplikację konsolową do testowania klasy PaintballGun	299
Automatycznie implementowane właściwości upraszczają kod	302
Użyj prywatnego settera, aby utworzyć właściwość tylko do odczytu	303
Do inicjowania właściwości użyj konstruktora z parametrami	305
Podaj argumenty, gdy używasz słowa kluczowego new	306
Inicjowanie pól i właściwości wewnątrzwierszowo lub w konstruktorze	313

![](_page_9_Figure_10.jpeg)

### Dziedziczenie

## 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**.

![](_page_10_Picture_5.jpeg)

Użyj instrukcji switch do dopasowywania wartości	327
Zastosowanie dziedziczenia pozwala napisać kod tylko raz	330
Jak zaprojektujesz symulator zoo?	332
W każdym miejscu, w którym możesz użyć klasy bazowej, możesz też użyć jednej z jej podklas	338
W podklasie można przesłaniać metody, aby modyfikować lub zastępować odziedziczone składowe	344
Zbuduj aplikację, aby lepiej poznać słowa kluczowe virtual i override	352
Podklasa może ukrywać metody z klasy bazowej	354
Używaj słów kluczowych override i virtual do dziedziczenia operacji	356
Klasa powinna robić jedną rzecz	366
Zbuduj system zarządzania rojem pszczół	370
Sprzężenie zwrotne wpływa na grę w zarządzanie rojem	388
System zarządzania rojem działa turowo; teraz przekształć go na system czasu rzeczywistego	390
Klasy abstrakcyjne celowo są niekompletne	394
Właściwości abstrakcyjne działają tak jak metody abstrakcyjne	398
Piekielny diament śmierci!	401

![](_page_10_Picture_7.jpeg)

![](_page_10_Picture_8.jpeg)

xviii

Kup ksi

# **Ćwiczenia** z Unity nr 3 Instancje obiektów gry

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

Zbudujmy grę w Unity!	404
Utwórz nowy materiał w katalogu Materials	405
Wygeneruj bilę w losowym punkcie sceny	406
Użyj debugera, aby zrozumieć wywołanie Random.value	407
Przekształć obiekt gry w obiekt prefab	408
Utwórz skrypt do sterowania grą	409
Dołącz skrypt do głównej kamery	410
Wciśnij Play, aby uruchomić kod	411
Użyj okna Inspector do pracy z instancjami obiektów gry	412
Wykorzystaj silnik fizyki, aby uniknąć pokrywania się bil	413
Bądź kreatywny!	414

¥ Hierarchy
★ All
Ø SampleScene \*
Ø Main Camera
Ø Directional Light
Ø OneBall(Clone)
Ø OneBall(Clone)</p

Pole ksi k

## Interfejsy, rzutowanie i instrukcja "is" 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!	416
Można zastosować rzutowanie, aby wywołać metodę DefendHive	417
Interfejs definiuje metody i właściwości, jakie klasa musi implementować	418
Interfejsy umożliwiają wykonywanie tego samego zadania niepowiązanym klasom	419
Nabierz wprawy w używaniu interfejsów	420
Nie możesz utworzyć obiektu typu interfejsu, ale możesz utworzyć referencję do interfejsu	426
Referencje typów interfejsowych są zwykłymi referencjami do obiektów	429
RoboPszczoła 4000 potrafi wykonywać pracę robotnic bez zużywania cennego miodu	430
Co zrobić, jeśli inne zwierzęta też mają pływać lub polować w stadzie?	438
Używaj interfejsów do pracy z klasami, które wykonują to samo zadanie	439
Bezpiecznie poruszaj się po hierarchii klas za pomocą instrukcji "is"	440
C# udostępnia też inne narzędzie do bezpiecznej konwersji typów: słowo kluczowe "as"	441
Stosuj rzutowanie w górę i w dół, aby poruszać się po hierarchii klas	442
Rzutowanie w górę i w dół działa także dla interfejsów	446
Domyślne implementacje zawierają ciało metod interfejsów	456
Wiązanie danych powoduje automatyczną aktualizację kontrolek MAUI	459
Polimorfizm oznacza, że jeden obiekt może przyjmować wiele	
różnych postaci	469

![](_page_12_Figure_5.jpeg)

![](_page_12_Picture_6.jpeg)

## Wyliczenia i kolekcje 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 formie rozmaitych ładunków, stert i wiązek. Będziesz potrzebować rozbudowanych narzędzi, aby je wszystkie uporządkować. Na szczęście C# udostępnia wszystko, czego potrzebujesz. **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.

Jeśli konstruktor tylko ustawia pola, użyj w zamian	
konstruktora głównego	474
Konstruktor główny może rozszerzać konstruktor bazowy	475
Wyliczenia umożliwiają pracę ze zbiorem poprawnych wartości	477
Wyliczenia umożliwiają reprezentowanie liczb za pomocą nazw	478
Listy umożliwiają łatwe przechowywanie kolekcji czegokolwiek	483
Utwórzmy aplikację do przechowywania butów	487
Generyczne kolekcje mogą przechowywać wartości dowolnego typu	490
Do tworzenia list możesz użyć wyrażenia kolekcji	496
Interfejs IComparable <duck> pomaga liście sortować kaczki</duck>	499
Utwórz obiekt komparatora	501
Komparatory potrafią wykonywać skomplikowane porównania	502
Możesz zrzutować w górę całą listę, używając interfejsu	
IEnumerable <t></t>	510
Przegląd możliwości słowników	513
CollectionView to kontrolka MAUI przeznaczona	
do wyświetlania kolekcji	524
ObservableCollection to kolekcja stworzona do wiązania danych	525
Użyj kodu XAML do utworzenia obiektów na potrzeby	
wiązania danych	529
Zmodyfikuj aplikację, aby używała słownika zasobów	530
Zmodyfikuj metody obsługi zdarzeń, aby korzystały	
ze słownika zasobów	532
Wykorzystaj zdobytą wiedzę do zbudowania aplikacji	
z dwiema taliami	533

Sortowanie według gatunków kaczek. \_\_\_\_\_

![](_page_13_Picture_7.jpeg)

Rzadko używana karta "książę bawołów".

![](_page_13_Picture_9.jpeg)

# 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	540
Dodaj dwa różne tryby gry	541
Dodaj tryb gry	542
Dodaj interfejs użytkownika do gry	544
Skonfiguruj obiekt Text wyświetlający wynik w interfejsie użytkownika	545
Dodaj przycisk, który wywołuje metodę uruchamiającą grę	546
Dodaj kod obsługi przycisku Zagraj ponownie i pola z wynikiem	547
Dokończ kod gry	548
Bądź kreatywny!	552

![](_page_14_Picture_4.jpeg)

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

![](_page_14_Picture_6.jpeg)

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

## LINQ i lambdy 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. Zobaczysz, jak używać obiektów anonimowych do zarządzania danymi w nowe i interesujące sposoby. 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	554
Użyj technologii LINQ do pisania kwerend dotyczących kolekcji	556
Użyj kwerendy LINQ do ukończenia aplikacji dla Jacka	564
Słowo kluczowe var powoduje, że C# wywnioskowuje	
typy zmiennych	566
Technologia LINQ jest wszechstronna	572
Użyj kwerendy grupującej do rozdzielenia sekwencji na grupy	574
Użyj kwerend złączających do scalania danych	
z dwóch sekwencji	577
Użyj słowa kluczowego new do utworzenia typu anonimowego	578
Testy jednostkowe pomagają zagwarantować, że kod działa	587
Zacznij pisać pierwszą metodę testową	588
Jeden projekt może uzyskać dostęp tylko do klas publicznych	
w innym projekcie	590
Zastosuj wzorzec Arrange-Act-Assert do napisania	501
skutecznego testu	591
Napisz test jednostkowy metody GetReviews	594
Użyj operatora => do tworzenia wyrażeń lambda	598
Użyj operatora ?: do podejmowania decyzji w lambdach	603
Kwerendy LINQ składają się z metod	604
Deklaratywne kwerendy LINQ można zrefaktoryzować	
do postaci łańcucha metod	606
Użyj operatora => do tworzenia wyrażeń switch	609
Omówienie klasy Enumerable	613
Użyj instrukcji vield return do tworzenia własnych sekwencji	615

![](_page_15_Picture_5.jpeg)

![](_page_15_Picture_6.jpeg)

![](_page_15_Picture_7.jpeg)

## Odczył i zapis plików Zachowaj dla mnie ostatni bajt

### Czasem trwałość się przydaje.

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 oraz bajtom w **danych szesnastkowych**, **Unicode i dwójkowych**.

W .NET do odczytu i zapisu danych używane są strumienie	622
Różne strumienie wczytują i zapisują różne dane	623
Użyj klasy StreamReader do odczytu pliku	629
Używanie statycznych klas File i Directory do pracy z plikami i katalogami	634
IDisposable gwarantuje, że obiekty zostaną poprawnie zamknięte	637
Unikaj błędów systemu plików dzięki instrukcjom using	638
Użyj klasy MemoryStream do strumieniowania danych do pamięci	639
Co się dzieje z obiektem w czasie serializacji?	645
Użyj klasy JsonSerializer do serializacji obiektów	648
Format JSON obejmuje tylko dane, a nie specyficzne typy języka C#	651
Łańcuchy znaków w C# są kodowane w formacie Unicode	655
Platforma .NET używa formatu Unicode do przechowywania znaków i tekstu	658
C# może wykorzystać tablice bajtów do przenoszenia danych	660
Użyj klasy BinaryWriter do zapisu danych binarnych	661
Użyj klasy BinaryReader do wczytania danych	662
Użyj klasy StreamReader do opracowania przeglądarki danych szesnastkowych	665
Użyj metody Stream.Read do wczytania bajtów ze strumienia	666
Zmodyfikuj przeglądarkę danych szesnastkowych, aby wczytywała dane bezpośrednio ze strumienia	667
Uruchamianie aplikacji w wierszu poleceń	668

![](_page_16_Figure_5.jpeg)

![](_page_16_Picture_7.jpeg)

0000:	54	6f	20	65	6c	65	6d	65	To eleme
0005:	6e	74	61	72	6e	65	2c	20	ntarne,
0010:	64	72	6f	67	69	20	57	61	drogi Wa
0015:	74	73	6f	6e	69	65			tsonie

## **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	674
Przygotuj kamerę	675
Utwórz obiekt gry reprezentujący gracza	676
Wprowadzenie do systemu nawigowania w Unity	677
Instalowanie pakietu AI Navigation	678
Czynności związane z nawigowaniem	679
Przygotowanie siatki nawigacyjnej	680
Spraw, aby postać automatycznie poruszała się po obszarze gry	683

![](_page_17_Picture_4.jpeg)

xxiv Kup ksi

## KAPITAN WSPANIAŁY ŚMIERĆOBIEKTU 4 zł Rozdział 11.

Życie i śmierć obiektu	690
Używaj (ostrożnie) klasy GC do wymuszania odśmiecania pamięci	691
Finalizator obiektu — Twoja ostatnia szansa, by coś ZROBIĆ	692
Kiedy DOKŁADNIE uruchamiany jest finalizator?	693
Finalizatory nie mogą zależeć od innych obiektów	695
Struktura wygląda jak obiekt	699
Wartości są kopiowane; referencje są przypisywane	700
Struktury są typami bezpośrednimi; klasy są typami referencyjnymi	701
Stos a sterta — więcej o pamięci	703
Używaj parametrów out, aby zwrócić z metody więcej niż jedną wartość	706
Przekazywanie przez referencję z użyciem modyfikatora ref	707
Używaj parametrów opcjonalnych do podawania wartości domyślnych	708
Referencja null nie wskazuje żadnego obiektu	709
Typy referencyjne niedopuszczające null pomagają unikać wyjątków NRE	710
Typy bezpośrednie dopuszczające null mogą się równać null i być bezpiecznie obsługiwane	713
Operator ?? automatycznie wykrywa wartości null	714
Kapitan Wspaniały — ale nie do końca	715
Rekordy automatycznie umożliwiają sprawdzanie równości wartości obiektów	717
Nie modyfikuj rekordów — kopiuj je	718
Metody rozszerzające dodają nowe operacje do ISTNIEJĄCYCH klas	723
Rozszerzanie typu podstawowego — string	724

Muszę... zrobić... – Wzdycha. – jedną... ostatnią... rzecz...

![](_page_18_Picture_4.jpeg)

![](_page_18_Picture_5.jpeg)

## Obsługa wyjątków Gaszenie pożarów robi się nudne

# 12

### Gdy musisz rozwiązywać błąd za błędem, nazywamy to "gaszeniem pożarów".

Wyobraź sobie, że minęło kilka lat. Spędziłeś cały ten czas na rozwoju umiejętności z zakresu C#. Kontynuujesz naukę i doskonalisz się, a teraz jesteś jednym ze starszych programistów w dużej firmie technologicznej. Ale spanikowani współpracownicy nadal dzwonią do Ciebie w środku nocy, ponieważ **Twój program się zawiesza** lub **nie działa tak, jak powinien**. Chcesz przeznaczać czas na pisanie kodu, a nie na gaszenie pożarów! Nic nie wybija programistów z rytmu tak jak konieczność wyeliminowania dziwnego błędu. Na szczęście C# udostępnia **obsługę wyjątków**, dzięki czemu 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.

	Р
	X
oje efr typu L	oonsho

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

![](_page_19_Picture_7.jpeg)

Przeglądarka danych szesnastkowych wczytuje nazwę pliku z wiersza poleceń	732
Gdy program zgłasza wyjatek. CLR generuje objekt wyjatku	736
Wszystkie obiekty wyjątków dziedziczą po klasie	
System.Exception	737
Istnieją pliki, które nie umożliwiają wykonania zrzutu szesnastkowego	740
Co się dzieje, gdy wywoływana metoda jest ryzykowna?	741
Obsługa wyjątków za pomocą bloków try-catch	742
Użyj debugera do prześledzenia przepływu sterowania	742
w bloku try-catch	/43
Ogólny blok catch obsługuje wyjątki typu System. Exception	745
Używaj wyjątku odpowiedniego do sytuacji	750
Filtry wyjątków pomagają tworzyć precyzyjne bloki do ich	
obsługi	754
Najgorszy blok catch W HISTORII — ogólny blok catch	
z komentarzami	756
Tymczasowe rozwiązania są akceptowalne (tymczasowo)	757
Użyj systemu NuGet, aby dodać do aplikacji bibliotekę	
do rejestrowania dzienników	759
Dodaj rejestrowanie dzienników do aplikacji	
ExceptionExperiment	760

![](_page_19_Figure_9.jpeg)

# **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	764
Dodaj platformę do sceny	765
Użyj opcji wstępnego obliczania, aby umożliwić chodzenie po platformie	766
Dodaj schody i rampę do siatki nawigacyjnej	767
Spraw, by gracz omijał przeszkody	769
Bądź kreatywny!	770

![](_page_20_Picture_4.jpeg)

Komponent NavMesh Obstacle <u>tworzy poruszającą się lukę w siatce</u> 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.

Skorowidz

Jesteś gotów?

## 1. Zacznij pisać programy w języku C#

## Zbuduj coś wspaniałego... szybko!

![](_page_22_Figure_2.jpeg)

#### 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 niezwykle łatwe. Visual Studio jest nie tylko znakomitym środowiskiem do pisania kodu, ale też **bardzo przydatnym narzędziem do nauki** i eksplorowania języka C#. Brzmi nieźle? *Pora zabrać się za programowanie!* 

## Poznaj C#... i dowiedz się, jak zostać świetnym programistą

Chcesz zostać świetnym deweloperem? Tak? W takim razie trafiłeś na właściwą książkę! *Możesz* się stać znakomitym programistą, a **C# jest idealnym językiem**, który Ci w tym pomoże. Oto dlaczego:

- ★ C# to nowoczesny język o wielkich możliwościach, który pozwala robić niesamowite rzeczy. Możesz go używać do tworzenia wszystkiego: od gier, przez strony internetowe, po zaawansowane aplikacje biznesowe. Czegokolwiek potrzebujesz, C# Ci to umożliwia.
- ★ Specjaliści ze znajomością C# są poszukiwani. Szukasz pracy związanej z programowaniem? Specjaliści ze znajomością języka C# należą do najbardziej rozchwytywanych, ponieważ firmy na całym świecie używają go do tworzenia aplikacji desktopowych i stron internetowych.
- ★ **C# jest** *wieloplatformowy*. Możesz pisać aplikacje działające w systemach Windows, macOS, Linux, a nawet na urządzeniach z Androidem i iPhone'ach.

### Skuteczny i ciekawy system nauki

Kiedy opanowujesz C# — i uczysz się *naprawdę* skutecznie nim posługiwać — poznajesz coś więcej niż tylko język. Uczysz się *zupełnie nowego sposobu myślenia*. I właśnie na tym etapie wkraczamy my. Poświęciliśmy ponad 15 lat na opracowywanie, badanie i testowanie różnych nowych sposobów, które pomogą Ci przyswoić sobie zagadnienia z obszaru C#. Będziesz używać **rozbudowanych** środowisk programistycznych do tworzenia prawdziwych projektów i pisania dużej ilości kodu. Poznasz i wypróbujesz w praktyce ważne **idee i wzorce** programistyczne, które pomogą Ci pisać świetny kod. Dowiesz się, jak korzystać z nowoczesnych narzędzi sztucznej inteligencji, aby szybciej rozwijać kod i efektywniej się uczyć. Dzięki tej książce zdobędziesz podstawy potrzebne do skutecznego i przyjemnego rozwoju oprogramowania.

Witamy w świecie C#. Zanurzmy się w nim!

Na przestrzeni lat wiele osób korzystających z wcześniejszych wydań tej książki skontaktowało się z nami, aby opowiedzieć, że pozycja ta pomogła im rozpocząć karierę programisty. Z niecierpliwością czekamy na wieści od Ciebie!

"Książka *C#. Rusz głową* pomogła mi rozpocząć karierę inżyniera oprogramowania i programisty backendów. Obecnie kieruję zespołem w firmie technologicznej i współtworzę oprogramowanie otwartoźródłowe".

— Zakaria Soleymani, kierownik zespołu programistów

"Bardzo Wam dziękuję! Wasze książki pomogły mi rozpocząć karierę".

- Ryan White, programista gier

![](_page_23_Picture_14.jpeg)

## Pisz kod i poznawaj C# z Visual Studio

Najlepszym sposobem na rozpoczęcie pracy z C# jest pisanie dużych ilości kodu.

W tej książce wykorzystujemy **obrazki, łamigłówki, quizy, historyjki i gry**, aby pomóc Ci w nauce języka C# w sposób odpowiedni dla Twojego mózgu. Każdy z tych elementów ma Ci ułatwiać pracę, a opracowaliśmy je wszystkie w jednym celu: aby przyswajanie koncepcji, zagadnień i umiejętności z zakresu C# było dla Ciebie interesujące.

Ta książka jest również **pełna projektów C#**, *specjalnie zaprojektowanych* w celu zapewnienia Ci wielu różnych sposobów na poznanie C# oraz ważnych idei i koncepcji, które pomogą Ci stać się świetnym programistą. Stworzyliśmy te projekty tak, aby były wciągające, zabawne i interaktywne. Dzięki temu będziesz mieć wiele okazji do zastosowania poszczególnych koncepcji, idei i umiejętności w praktyce.

### Visual Studio to <u>bezpłatna</u> brama do języka C#

Nauka języka C# polega na odkrywaniu i rozwijaniu swoich umiejętności *we własnym tempie* — i właśnie w tym obszarze przydatne jest **Visual Studio**. Jest to fantastyczne narzędzie opracowane przez Microsoft. Jego głównym elementem jest edytor kodu i projektów C#, ale Visual Studio udostępnia znacznie więcej możliwości. Środowisko to jest kreatywnym narzędziem, które pomaga w każdym aspekcie programowania w języku C#. W tej książce będziemy używać Visual Studio jako ważnego narzędzia pomagającego w nauce i poznawaniu języka C#.

Visual Studio to **IDE** (skrót od ang. *integrated development environment*, czyli *zintegrowane środowisko programistyczne*), obejmujące edytor tekstu, moduł projektowania wizualnego, menedżer plików i debuger... Jest więc jak narzędzie wielofunkcyjne do wykonywania wszelkich zadań potrzebnych w trakcie pisania kodu.

Oto tylko kilka rzeczy, w których pomaga Visual Studio:

- ★ Jest menedżerem plików i projektów. Projekty C# często składają się z wielu plików. Visual Studio ułatwia sprawdzenie, gdzie dokładnie się one znajdują, a ponadto zapewnia integrację z systemami kontroli wersji, takimi jak Git, aby zagwarantować, że nigdy nie zgubisz żadnego wiersza kodu.
- ★ Pomaga edytować kod i zarządzać nim. Visual Studio ma wiele intuicyjnych funkcji, które pomagają edytować kod i projekty C#, w tym rozbudowane narzędzia oparte na sztucznej inteligencji, takie jak wyskakujące okienka IntelliSense i mechanizm uzupełniania kodu IntelliCode. Rozwiązania te generują przydatne sugestie, które pomagają w płynnej pracy.
- ★ Jest debugerem, który pozwala podejrzeć działanie kodu. Podczas debugowania aplikacji w Visual Studio można się dokładnie przyjrzeć pracy kodu. Jest to świetny sposób na to, aby precyzyjnie zrozumieć działanie kodu C#.

## 🕼 🕄 Wskazówki dotyczące IDE

W tej książce Visual Studio często będzie nazywane po prostu **"środowiskiem IDE"**. Zwracaj uwagę na przydatne wskazówki dotyczące środowiska IDE, które pomogą Ci w wydajniejszym pisaniu kodu.

![](_page_24_Picture_14.jpeg)

Środowisko Visual Studio jest dostępne tylko dla systemu Windows, ale na szczęście można również użyć *Visual Studio Code* do uruchomienia wszystkich projektów opisanych w tej książce. Jeśli korzystasz z systemu macOS lub Linux albo jeżeli chcesz używać Visual Studio Code zamiast Visual Studio, przejdź do następnego fragmentu, gdzie pokażemy, jak skonfigurować to narzędzie i jak użyć go do zbudowania pierwszego projektu w C#.

![](_page_24_Picture_16.jpeg)

Visual Studio to rozbudowane środowisko programistyczne, ale także pomoc dydaktyczna przydatna w eksplorowaniu języka C#.

Jeśli zamiast Visual Studio zdecydujesz się używać Visual Studio Code, wtedy właśnie to narzędzie będzie Twoim środowiskiem IDE. Oba te narzędzia to środowiska IDE!

## Instalowanie Visual Studio Community Edition

Otwórz stronę *https://visualstudio.microsoft.com* i **pobierz Visual Studio Community Edition**. Narzędzie to jest dostępne zarówno dla systemu Windows, jak i macOS. Instalatory wyglądają nieco inaczej w zależności od używanej platformy. Pamiętaj, aby zainstalować narzędzia programistyczne Programowanie aplikacji klasycznych dla platformy .NET i Programowanie .NET Multi-Platform App UI. Będziemy tworzyć gry trójwymiarowe za pomocą Unity, więc zwróć uwagę na to, aby zaznaczyć również opcję związaną z tą platformą.

Po uruchomieniu instalatora Visual Studio zaznacz opcje *Programowanie aplikacji klasycznych dla platformy* .*NET*, *Programowanie .NET Multi-Platform App UI* i *Opracowywanie gier za pomocą aparatu Unity*, aby zainstalować narzędzia środowiska Visual Studio, których będziesz używać w tej książce. Ponadto jeśli planujesz pobrać *Head First C# Blazor Learner's Guide* i nauczyć się tworzenia stron internetowych w języku C#, zaznacz też opcję *Opracowywanie zawartości dla platformy ASP.NET*.

![](_page_25_Figure_4.jpeg)

## Uruchamianie Visual Studio

Przejdziemy od razu do kodu! Po zakończeniu instalacji uruchom Visual Studio.

![](_page_26_Picture_3.jpeg)

Zwróć uwagę na ramki "Zrelaksuj się". Omawiamy w nich wybrane typowe problemy, z którymi boryka się wielu Czytelników. Możesz więc się ich spodziewać i nie musisz się nimi martwić.

![](_page_26_Picture_5.jpeg)

## Zrelaksuj się

#### Zrób sobie kawę, ponieważ instalacja środowiska Visual Studio może zająć trochę czasu.

Nie martw się, jeśli instalacja Visual Studio zajmuje kilka minut (lub więcej!). A skoro już o tym mowa, to jest jeszcze coś, czym nie musisz się przejmować.

Wszystkie zrzuty ekranu zostały wykonane przy użyciu **Visual Studio 2022 Community Edition**, najnowszej wersji dostępnej podczas pisania tej książki. Jeśli po jej opublikowaniu firma Microsoft wydała nowszą wersję środowiska, możesz ją wypróbować! Kod i pomysły, które prezentujemy, nadal powinny działać poprawnie. Jeżeli jednak chcesz otrzymać takie same zrzuty ekranu, Microsoft udostępnia starsze wersje środowiska Visual Studio do pobrania (*https://visualstudio.microsoft.com/vs/older-downloads*). Zawsze możesz zainstalować różne wersje tego środowiska na tym samym komputerze.

Jeśli napotkasz problemy z instalacją Visual Studio lub uruchomieniem pierwszego projektu, odwiedź nasz kanał w serwisie YouTube (https://www.youtube.com/@headfirstcsharp), aby zobaczyć filmy z całego procesu instalacji.

## Utwórz i uruchom pierwszy projekt C# w Visual Studio

Najlepszym sposobem nauki języka C# jest rozpoczęcie pisania kodu. Dlatego w tej książce będziesz pisać dużo kodu (i tworzyć wiele aplikacji!). Dla każdej aplikacji dodasz osobny **projekt**, czyli generowany przez Visual Studio katalog ze specjalnymi plikami, który służy do porządkowania całego kodu.

![](_page_27_Picture_3.jpeg)

### Utwórz w środowisku Visual Studio nowy projekt.

Gdy uruchomisz środowisko Visual Studio, najpierw zobaczysz okno *Rozpocznij* z czterema opcjami. **Kliknij** *Utwórz nowy projekt*, aby utworzyć nowy projekt.

![](_page_27_Figure_6.jpeg)

Podczas tworzenia nowego projektu Visual Studio wyświetla pytanie o to, którego **szablonu projektu** chcesz użyć. Każdy projekt C# składa się z zestawu folderów i plików. Visual Studio udostępnia wiele wbudowanych szablonów, które służą do generowania różnych rodzajów projektów. W tej książce będziesz używać szablonów Visual Studio do tworzenia trzech rodzajów projektów: projektów aplikacji konsolowych, projektów .NET MAUI i projektów testów jednostkowych MSTest. (Będziesz także tworzyć projekty Unity, ale nie będziesz używać do tego Visual Studio).

![](_page_27_Picture_8.jpeg)

W tej książce będziesz pisać dużo kodu, co oznacza, że będziesz też **tworzyć wiele projektów**. Większość z nich będzie **projektami aplikacji konsolowych**, podobnych do tego, który tworzysz teraz. Dlatego za każdym razem, gdy tworzysz nowy projekt tego typu, możesz postępować zgodnie z przedstawionymi tu instrukcjami. Pamiętaj tylko, by za każdym razem podać inną nazwę projektu. Dzięki temu Visual Studio wygeneruje nowy projekt w nowym folderze (nie martw się, jeśli dana nazwa już istnieje, środowisko wyświetli ostrzeżenie).

#### zbuduj coś wspaniałego... szybko!

### Wybierz szablon projektu, jaki Visual Studio ma zastosować.

Visual Studio tworzy nowe projekty za pomocą *szablonu*, który określa, jakie pliki należy wygenerować. **Wybierz szablon** *Aplikacja konsoli* i kliknij przycisk *Dalej*.

Wpisz "Aplikacja konsoli" w polu wyszukiwania lub przewiń listę do szablonu Aplikacja konsoli.

Utwórz nowy projekt		Wyszukaj szablony (Alt+S)	- م	Wyczyść wszystko
Ostatnie szablony projektów		C# •	Wszystkie platformy -	Wszystkie typy projektów 🔹
Mplikacja konsoli	C#	Aplikacja konsoli Projekt służący do tworze	enia aplikacji wiersza polecenia, któ	óra może działać na
Pamiętaj, aby w a nie jakiś inny	ybrać C#, język. ———	C# Linux macO!	nach Windows, Linux i macOS S Windows Konsola	

### **3** Wprowadź nazwę projektu i kliknij przycisk Dalej.

Nazwa projektu jest ważna, ponieważ wpływa na nazwy plików i folderów, a także pojawia się w kodzie generowanym przez Visual Studio. Jeśli prosimy o podanie konkretnej nazwy, postępuj zgodnie z instrukcjami. W przeciwnym razie kod w projekcie może nie pasować do zrzutów z książki.

Konfiguruj nowy projekt	W tym miejscu wpisz
Aplikacja konsoli C# Linux macOS Windows Konsola	Visual Studio utworzy
Nazwa projektu	w tej lokalizacji nowy katalog zgodny z nazwą projektu. Możesz wybrać
MyFirstConsoleApp	inną lokalizację.
Lokalizacja	
C:\Users\walcz\source\repos	

#### Upewnij się, że korzystasz z aktualnej wersji .NET.

W czasie, gdy powstaje ta książka, aktualna jest wersja 8.0. Zadbaj o to, aby użyć właśnie jej (lub nowszej wersji). Następnie kliknij przycisk Utwórz, by wygenerować projekt.

Informacje dodatkowe
Aplikacja konsoli C# Linux macOS Windows Konsola
Platforma 🛈
.NET 8.0 (Wersja zapoznawcza)
🗌 Nie używaj instrukcji najwyższego poziomu 🛈
Włącz publikowanie native AOT 🛈

Gdy Visual Studio utworzy projekt, otworzy plik o nazwie Program.cs z następującym kodem:

Program.cs 🕂 🔅	×
C MyFirstConsole	App -
10	<pre>// See https://aka.ms/new-console-template for more information</pre>
2	Console.WriteLine("Hello, World!");

Kod aplikacji znajduje się w pliku *Program.cs.* Możesz go edytować w pokazanym tu oknie.

![](_page_29_Picture_1.jpeg)

### Uruchom nową aplikację.

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

MyFirstConsoleApp

### 6 Przyjrzyj się danym wyjściowym programu.

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

![](_page_29_Picture_7.jpeg)

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\
net8.0\MyFirstConsoleApp.exe (proces #####) zakończono z kodem 0 (0x0).
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 zamykającego je klawisza. Dzięki temu możesz zobaczyć dane wyjściowe, zanim okno zniknie. Wciśnij klawisz, aby zamknąć okno. Następnie ponownie uruchom program.

W ten sposób będziesz uruchamiać wszystkie projekty aplikacji konsolowych, jakie napiszesz w trakcie lektury tej książki.

Visual Studio to IDE, czyli zintegrowane środowisko programistyczne. Jeśli wcześniej nie korzystałeś z IDE, może ono wyglądać na bardzo skomplikowane. Teraz jest dobry czas na to, aby się do niego przyzwyczaić. Jednym z najważnie sposobów na przyswojenie sobie nowych pomysłów, informacji, umiejętności i narzędzi jest zapisanie informacji. Przyjrzyj się więć dokładnie różnym częściom Visual Studio i zanotuj, do czego Twoim zdaniem służą. Nie musisz m całkowitej pewności, po prostu zgaduj!		Visual Studio to świetne narzędzie pomagające w nau będziesz pisać dużo kodu, więc jest to świetny czas na	ce i poznawaniu języka C#. W tej książce zapoznanie się z tym środowiskiem.
Image: Section of the section of th	Visual Studio to wyglądać na ba sposobów na prz Przyjrzyj się więc całkowitej pewno	IDE, czyli zintegrowane środowisko programistyczne. Jeśli w rdzo skomplikowane. Teraz jest dobry czas na to, aby się do zyswojenie sobie nowych pomysłów, informacji, umiejętnośc c dokładnie różnym częściom Visual Studio i zanotuj, do czę ości, po prostu zgaduj!	cześniej nie korzystałeś z IDE, może ono niego przyzwyczaić. Jednym z najważniejsz i i narzędzi jest <b>zapisanie informacji</b> . ego Twoim zdaniem służą. Nie musisz mieć
Pik Edycja Wdok Git Projekt Kompilowanie Debuguj Teti Analiza Nuzędzia Rozzezenia Olino Romo P Wyczukaj MyfirstConoloApp X Zaloguj se      Pik Edycja Wdok Git Projekt Kompilowanie Debuguj Teti Analiza Nuzędzia Rozzezenia Olino Romo P Wyczukaj MyfirstConoloApp X Zaloguj se      Pik Edycator rozviązań     MyfirstConoloApp / D @ + W firstConoloApp / D @ + W firstCono	·····	······	
Image: Service Service       Provide Kompilowanie Debugij Test Ausiza Narodzija Rozzezenia Okno Pomor       Provide Service       Provide Service <t< th=""><th></th><th></th><th></th></t<>			
Programus       Image: State and Sta	00 Plik Edycja Wid ◎ - ③ 월 - 딸 멸	dok Git Projekt Kompilowanie Debuguj Test Analiza Narzędzia Rozszerzenia Okno Pomoc 1 129 1° - C² - Debug - Any CPU - ► MyfirstConsoleApp - ▷ 🕤 - 115 🔂 🕫 📩 🖬	요 Wyszukaj WyFirstConsoleApp 'R Zaloguj się - 미 고양 씨 쥐 줘 줘 구 문 GitHub Copilot
Image: State Stat	Program.cs + ×		Besplorator rozwiązań
2 Console.WriteLine("Hello, World!"); 3 Console.WriteLine("Hello, World!"); 3 Console.WriteLine("Hello, World!"); 4 MyTiteLine(Console.op) 4 Af Zielmodi 6 Program.cs 4 Program.cs 4 Program.cs 4 Console.WriteLine("Hello, World!"); 4 MyTiteLine(Console.op) 4 Af Zielmodi 6 Program.cs 4 Console.WriteLine("Hello, World!"); 4 MyTiteLine("Hello, World!"); 5 MyTiteLine("Hel	MyFirstConsoleApp	// See https://aka.ms/new-console-template for more information	
C. Programs     C. Progr	2 3	Console.WriteLine("Hello, World!");	Consider the second secon
Utsta bleddow Dane wyściowe			C- Programoa
100 % ● Nie ználeziono żadnych problemów 😤 ▼ W.: 1 Zn: 1 SPACJE CRLF Bospiorator rozwiązań Zmiany Git. Właściwości Lista błędów Dane wyjściowe			
100 % · @ ● Nie ználeziono żadnych problemów - W.: 1 Zn: 1 SPACIE CRLF Eksplorator rozwiązań Zmiany Git. Właściwości Lista błędów Dane wyjściowe			
100 % → @ Nie znałesiono żadnych problemów 😤 → W: 1 Zn: 1 SPACJE CRLF Eksplorator rozwiązań Zmiany Git Właściwości Lista błędów Dane wyjściowe			
100 % 🖓 🔍 Nie znałesiono żadnych problemów 🖉 🔹 W.: 1 Zn.: 1 SPACJE CRLF Eksplorator rozwiązań Zmiany Git Właściwości Lista błędów Dane wyjsciowe			
100 % – @ Nie znakeżono zadnych problemów 😚 – W.: 1 Zn.: 1 SPACJE CRLF Eksplorator rozwiązań Zmiany Git Właściwości Lista błędów Dane wyjściowe			
100 % 🔹 @ Nie znakeżono żadnych problemów 😚 🔹 W.: 1 Zn.: 1 SPACJE CRLF Eksplorator rozwiązań Zmiany Git Właściwości Lista błędów Dane wyjściowe			
	100 % • @ Ø t Lista blędów Dane wy	Nie znakejiono žadnych problemów   🖑 🕶 W.: 1 . . jściowe	Zn.: 1 SPACJE CRLF Ekoplorator rozwiązań Zmiany Git Właściwości
📮 Gotowe 🕆 Dodaj do kontroli zródia 🔹 🗟 Wybierz repozytorium 🔹	Gotowe		个 Dodaj do kontroli źródła 🔺 闭 Wybierz repozytorium 🔺 🚨
	Po utworzeniu a	plikacii zawierała ona plik z dwoma wierszami. Zapisz, co w	edług Ciebie robi każdy wiersz.
Po utworzeniu aplikacji zawierała ona plik z dwoma wierszami. Zapisz, co według Ciebie robi każdy wiersz.			
Po utworzeniu aplikacji zawierała ona plik z dwoma wierszami. Zapisz, co według Ciebie robi każdy wiersz.	// See https://aka	.ms/new-console-template for more information.	
Po utworzeniu aplikacji zawierała ona plik z dwoma wierszami. Zapisz, co według Ciebie robi każdy wiersz. // See https://aka.ms/new-console-template for more information.			
Po utworzeniu aplikacji zawierała ona plik z dwoma wierszami. Zapisz, co według Ciebie robi każdy wiersz. // See https://aka.ms/new-console-template for more information.	Console.Write	Line("Hello, World!");	

Spróbuj poprzesuwać panele w Visual Studio. Kliknij przycisk pinezki (‡), aby zwinąć okno Eksplorator rozwiązań przy panelu bocznym. Zresetuj układ. W tym celu wybierz opcję Resetuj układ okna z menu Okno, a następnie opcję Tak.

Rozwiązanie	
Możesz otworzyć wiele plików jednocześnie i przełączać się	Pole Wyszukaj pozwala otworzyć okno, któr
między nimi za pomocą kart.	umożliwia przeszukiwanie kodu w projekcie
	i znajdowanie funkcji Visual Studio.
Image: Construction       Image: Construction of the construction	Pomoc 《P Wyszukai * MyfirstConsoleApp 法 Zaloguj się – ロ × * La 作 当当日云云云石。 农 GitHub Copilot 家
Program.cs     >       Wh/FintConcoleApp     -       (a)     1.0°       2     Console.WriteLine("Hello, World!");	• • • • • • • • • • • • • • • • •
To okno edvtora. Umożliwia edytowanie zawartości	El MyfirstOnsolcApp     MyfirstOnsolcApp     M Zalezności     C= Program.cs
dowolnego pliku z rozwiązania. Używane są tu	
kolory, aby ułatwić czytanie kodu z pliku	Okno Eksplorator rozwiązań
Program.cs.	wyświetla wszystkie pliki
	I Katalogi projektu oraz
	W edytorze
	Wz.1 Zn.: 1 SPACJE CRLF Ekoplorator rozwiązań Zmiany Git Właściwości

Po utworzeniu aplikacji zawierała ona plik z dwoma wierszami. Zapisz, co według Ciebie robi każdy wiersz.

// See https://aka.ms/new-console-template for more information.

To jest komentarz. Nie wykonuje żadnych operacji, tylko zapewnia informacje dla osoby czytającej kod. Console.WriteLine("Hello, World!");

To wiersz kodu sprawiający, że aplikacja wyświetla tekst "Hello, World!" (bez cudzysłowów).

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

głupich pytań

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

 $\Theta$ : 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. Robi to przez przekazanie podanych przez Ciebie informacji do szablonu używanego do generowania plików. Nie potrafi jednak najważniejszego — ustalić, co program ma robić, i sprawić, by to robił. Żadne IDE tego nie potrafi. Choć Visual Studio jest jednym z najbardziej zaawansowanych środowisk programistycznych, ma ograniczone możliwości. To Ty (nie IDE) musisz napisać kod, który wykonuje najważniejsze zadania.

## $\mathcal{P}$ : Co zrobić, jeśli IDE wygeneruje kod, którego nie potrzebuję w projekcie?

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

**G**: 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. Płatne wersje oferują dodatkowe funkcje przydatne dla firm i zespołów zajmujących się rozwojem oprogramowania.

### 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 domyślne?

**G**: Jeśli wybierzesz opcję **Resetuj układ okna** z menu Okno, IDE przywróci domyślny układ okna. Następnie użyj menu **Widok**, by otworzyć brakujące okna. Niektóre z okien widocznych dalej w tym rozdziale znajdziesz w podmenu **Inne okna**. Przy jego użyciu możesz sprawić, że ekran będzie wyglądał tak jak w tym rozdziale i na dalszych stronach książki.

> Niektóre okna domyślnie są zwijane. Użyj przycisku pinezki w prawym górnym rogu okna, aby pozostało otwarte.

Visual Studio generuje kod, który możesz wykorzystać jako punkt wyjścia do tworzenia aplikacji. Sprawienie, by robiła ona to, co ma robić, należy całkowicie do Ciebie.

## W trakcie lektury C#. Rusz głową możesz używać Visual Studio Code

#### Jeśli używasz Visual Studio (a nie VSCode), możesz przeskoczyć do punktu "Napiszmy grę!".

Jeśli w ciągu ostatnich kilku lat interesował Cię świat programowania, prawdopodobnie wiesz, ile ekscytacji i poruszenia wywołało narzędzie **Visual Studio Code** (często nazywane **VSCode**). Jest to rozbudowany edytor kodu działający w systemach Windows, macOS i Linux, który zyskał dużą popularność wśród programistów, ponieważ jest łatwy w użyciu, wszechstronny, szybki i intuicyjny.

Jeśli korzystasz z systemu Windows, zalecamy użycie środowiska Visual Studio (*nie* VSCode), ponieważ zostało ono zbudowane specjalnie z myślą o języku C#, dlatego posiada wbudowane narzędzia, których obecnie brakuje w VSCode. Niemniej *wszystkie projekty przedstawione w tej książce można wykonać za pomocą VSCode*. Większość zrzutów ekranu w tej książce przedstawia środowisko Visual Studio, ale wyjaśniamy również, jak wykonać analogiczne operacje w VSCode, jeśli odbywa się to inaczej niż w Visual Studio.

Aby w trakcie pracy z tą książką używać Visual Studio Code, zacznij od pobrania tego narzędzia ze strony *https://code.visualstudio.com.* Uruchom instalator i wybierz domyślne opcje. Po zakończeniu instalacji otwórz *VSCode.* Zostanie wyświetlony monit o wybranie motywu kolorystycznego. Do wykonania zrzutów ekranu wybraliśmy Dark Modern, ponieważ dla Visual Studio używamy motywu jasnego, więc ciemny motyw dla VSCode pomoże łatwiej odróżnić zrzuty z obu narzędzi.

![](_page_33_Picture_6.jpeg)

Przy pierwszym uruchomieniu VSCode wyświetla kartę powitalną z wieloma ustawieniami (w tym z motywem kolorystycznym). Możesz się z nimi zapoznać lub zamknąć kartę.

*Korzystanie z Visual Studio Code jest opcjonalne.* W systemach Windows i macOS możesz użyć tego narzędzia do wykonania wszystkich projektów z tej książki. Czytelnicy korzystający z Linuksa mogą potrzebować emulatora Androida, aby wykonać projekty związane z .NET MAUI. Więcej na ten temat dowiesz się dalej.

Jeśli masz trudności z instalacją VSCode lub uruchomieniem pierwszego projektu, odwiedź nasz kanał w serwisie YouTube (*https://www.youtube.com/@headfirstcsharp*), aby zobaczyć filmy z całym procesem instalacji w systemach Windows i macOS.

#### zbuduj coś wspaniałego... szybko!

### Instalowanie rozszerzeń dla języka C#

Kliknij przycisk Extensions (E) po lewej stronie okna środowiska VSCode, aby otworzyć panel Extensions. W górnej części panelu znajduje się pole wyszukiwania z tekstem "Search Extensions in Marketplace". Znajdź wszystkie wymienione niżej rozszerzenia:

- ★ C# Dev Kit. To rozszerzenie zawiera narzędzia potrzebne do tworzenia, edytowania i debugowania projektów C# i .NET.
- ★ .NET MAUI. Większość rozdziałów w tej książce zawiera projekty wymagające .NET MAUI, platformy do tworzenia aplikacji desktopowych i mobilnych w języku C#.
- ★ Unity. Ćwiczenia z Unity umożliwiają sprawdzenie w praktyce umiejętności z zakresu C# w ramach tworzenia gier i symulacji trójwymiarowych.

Install V

.NET MAUI

Microsoft

Extend C# Dev Kit with tools ...

Upewnij się, że instalujesz oficjalne rozszerzenia od Microsoftu. **Kliknij przycisk Install** przy każdym rozszerzeniu, aby je zainstalować.

Official C# extension from ...

C#

NET MAUL

![](_page_34_Picture_7.jpeg)

To ustawienie określa, gdzie VSCode ma wyświetlać dane wyjściowe aplikacji. Tu użyj okna terminala. W przeciwnym razie niektóre aplikacje rozwijane w dalszej części książki nie będą działać.

ŝ

C# Dev Kit

Microsoft

![](_page_34_Picture_9.jpeg)

34K ± 5

Install N

Po zainstalowaniu rozszerzeń VSCode może wyświetlić prośbę o restart systemu. Może też wyświetlić zakładki Getting Started z przydatnymi informacjami.

### Zmiana ustawień konsoli debugowania C#

Po skonfigurowaniu rozszerzeń kliknij ikonę koła zębatego w lewym dolnym rogu okna środowiska VSCode i wybierz opcję *Settings* (lub naciśnij kombinację *Ctrl*+przecinek lub  $\Re$ +przecinek). Wyszukaj ustawienie **csharp.debug.console**. Powinna się pojawić lista rozwijana z kilkoma opcjami. Zmień ustawienie na **integratedTerminal**.

Teraz wszystko jest gotowe do rozpoczęcia pisania kodu C#!

![](_page_34_Picture_14.jpeg)

## Tworzenie i uruchamianie pierwszego projektu w Visual Studio Code

Visual Studio Code jest przede wszystkim **edytorem**, co oznacza, że jego funkcje są dostosowane do otwierania i edytowania wielu różnych rodzajów plików. Okno VSCode jest zwykle używane do edycji plików z **katalogu** i jego podkatalogów. Po otwarciu VSCode używa ostatnio otwartego katalogu, jednak przy pierwszym uruchomieniu należy wskazać potrzebny folder. Teraz przeprowadzimy Cię przez proces tworzenia katalogu z nowym projektem .NET.

![](_page_35_Picture_3.jpeg)

### Kliknij przycisk Create .NET Project.

W górnej części VSCode pojawi się okno z listą typów projektów i polem wyszukiwania z monitem *Select a template to create a new .NET Project.* Wpisz Console w polu wyszukiwania, a następnie wybierz *Console App* z listy szablonów, aby utworzyć nowy projekt .NET Console App.

![](_page_35_Picture_6.jpeg)
#### 2 Wybierz katalog dla nowego projektu.

VSCode wyświetli okno przeglądarki katalogów. **Wybierz lokalizację dla nowego projektu.** W oknie przeglądarki katalogów widoczny jest przycisk *New folder*. W tej książce utworzysz wiele projektów, więc sugerujemy dodanie przeznaczonego na nie katalogu o nazwie *Projects* w katalogu głównym lub w katalogu *Dokumenty*.

W folderze z projektami utwórz nowy katalog i **nazwij go MyFirstConsoleApp**. Następnie przejdź do właśnie utworzonego katalogu *MyFirstConsoleApp* i **kliknij** *Select Folder*.



#### Podaj nazwę projektu.

Każdy projekt C# ma nazwę. Zazwyczaj katalog projektu ma taką samą nazwę jak sam projekt. Po wybraniu katalogu VSCode wyświetli monit o podanie nazwy projektu:

	Name the new project
	MyFirstConsoleApp
F.	Press 'Enter' to confirm your input or 'Escape' to cancel

**Wpisz w polu MyFirstConsoleApp**, a następnie wciśnij klawisz *Enter*, aby utworzyć projekt. VSCode może wyświetlić pytanie, czy ufasz autorom folderu. Jest to bardzo przydatna funkcja bezpieczeństwa, ponieważ zapobiega przypadkowemu uruchomieniu złośliwego kodu. **Kliknij przycisk Yes, I trust the authors.** Możesz też zaznaczyć pole wyboru, aby poinformować, że zawsze ufasz wszystkim elementom w katalogu projektów.

#### Zainstaluj pakiet .NET Core SDK. Trzeba to zrobić tylko raz.

Zanim będzie można tworzyć i uruchamiać aplikacje C# i .NET, należy zainstalować pakiet **.NET Core SDK**. Najprostszym sposobem na zrobienie tego w systemie Windows jest zainstalowanie Visual Studio 2022. Jeśli nie masz zainstalowanego pakietu SDK, VSCode wyświetli okno z monitem o jego pobranie. **Kliknij przycisk** *Get the SDK*, a otworzy się okno przeglądarki ze stroną *https://dot.net/core-sdk-vscode*. Postępuj zgodnie z instrukcjami, aby pobrać najnowszą wersję pakietu SDK dla używanego systemu operacyjnego. Należy uważać, aby wybrać architekturę odpowiednią dla komputera. Użytkownicy komputerów Mac wyprodukowanych po 2019 roku powinni wybrać *Arm64*, a osoby korzystające ze starszych maszyn Mac z procesorem Intel powinny kliknąć *Intel*.

 The .NET Core SDK cannot be located: Error running dotnet - info: Error: Command failed: dotnet -- info 'dotnet' is not recognized as an internal or external command, operable program or batch file. 'dotnet' is not recognized as an internal or external command, operable program or batch file. . .NET Core debugging will not be enabled. Make sure the .NET Core SDK is installed and is on the path.
 Source: C# (Exten... Disable message in settings Get the SDK Help

Środowisko Visual Studio Code jest bardzo popularne wśród programistów, ponieważ wymaga mało zasobów, jest otwartoźródłowe i istnieje olbrzymi ekosystem powiązanych z nim rozszerzeń oraz narzędzi. Korzystanie z tego środowiska będzie jednak wymagało nieco więcej pracy ręcznej, dlatego zalecamy posługiwanie się Visual Studio.

#### uruchamianie aplikacji w visual studio code

#### 6 Rozwiń eksplorator rozwiązań i otwórz plik Program.cs.

Gdy VSCode utworzy projekt C#, panel eksploratora po lewej stronie będzie zawierał kilka zwijanych sekcji. VSCode używa plików i katalogów, a **eksplorator** służy do ich przeglądania i otwierania do edycji.

**Rozwiń sekcję** *Solution Explorer* w dolnej części eksploratora. *Solution Explorer* jest częścią pakietu C# Dev Kit, który umożliwia pracę z projektami C# w VSCode. To narzędzie wyświetla wszystkie pliki i podkatalogi, które VSCode utworzyło dla danego projektu. W tym przykładzie aplikacja obejmuje jeden plik z kodem C# o nazwie *Program.cs.* **Kliknij** *Program.cs* w sekcji *Solution Explorer*, aby otworzyć plik.

#### 6 Uruchom aplikację.

Gdy plik z kodem C# (o rozszerzeniu .cs) jest otwarty w sekcji Solution Explorer, w prawym górnym rogu okna pojawia się przycisk Run (). Kliknij go, aby uruchomić aplikację.

Aby uruchomić aplikację, możesz również **nacisnąć klawisz F5** i wybrać opcję *Start Debugging* z menu *Run.* VSCode może wtedy wyświetlić monit o wybranie debugera. Jeśli tak się stanie, wybierz C#. Gdy pojawi się pytanie o konfigurację uruchomieniową, wybierz tę, która pasuje do nazwy projektu. Zawsze, gdy chcesz uruchomić aplikację, możesz wcisnąć przycisk *F5*. Aplikacja rozpocznie pracę, a Visual Studio otworzy **panel konsoli debugowania**, gdzie wyświetla dane wyjściowe i umożliwia interakcję z nimi. *W ten sposób będziesz uruchamiać wszystkie projekty aplikacji konsolowych tworzone w całej książce*.



zresetować VSCode, aby

rozpocząć nowy projekt.

# Konfigurowanie Visual Studio Code na potrzeby następnego projektu

VSCode jest świetnym edytorem kodu, ale w odróżnieniu od Visual Studio nie został zaprojektowany specjalnie do rozwoju projektów C# i .NET. Świetnie radzi sobie z językiem C#, ale trzeba wykonać trochę dodatkowych czynności, aby przygotować go do pracy.

### Najpierw otwórz katalog, a następnie dodaj projekt

VSCode jest niezwykle elastyczny i można go używać na wiele sposobów. Jeśli dopiero poznajesz to narzędzie, zalecamy tworzenie nowego katalogu dla każdego projektu omawianego w książce. Gdy tworzysz nowy projekt, wybierz Close Folder z menu File, aby zamknąć bieżący katalog. Następnie utwórz nowy katalog i otwórz go. W ten sposób można

### Paleta poleceń

Wszystkie czynności potrzebne do tworzenia i uruchamiania projektów można wykonywać z poziomu **palety poleceń**. Jest to centrum wszystkich funkcji VSCode. Aby wyświetlić paletę poleceń, naciśnij klawisze Ctrl+Shift+P (lub  $\textcircled{}+ \Re + P$  na komputerze Mac). Użyj opcji *.NET: New Project*, aby utworzyć nowy projekt w bieżącym katalogu. Istnieją również polecenia do otwierania i zamykania rozwiązań .NET. Więcej informacji na temat rozwiązań znajdziesz w dalszej części rozdziału.

Po otwarciu katalogu projektu aplikacji .NET można ją uruchomić za pomocą opcji *Debug: Start Debugging* z palety poleceń. Wybierz opcję C#, aby uruchomić projekt aplikacji konsolowej.

Select debugger			
.NET MAUI C# Wybierz tę opcje	, aby uruchomić proj	jekt aplikacji k	o <b>nsolowej</b> . Suggested
Install an extension for C#			

Zanim przejdziesz do dalszej części rozdziału, musisz zainstalować .NET MAUI. Jeśli używasz Visual Studio 2022, wszystko jest już gotowe. Jeżeli jednak korzystasz z VSCode, musisz przeprowadzić instalację ręcznie.

### Przed lekturą reszty rozdziału zainstaluj .NET MAUI

W dalszej części tego rozdziału stworzysz grę przy użyciu **.NET MAUI** (Multi-platform App UI), rozbudowanego wieloplatformowego frameworka, który umożliwia tworzenie za pomocą .NET i C# graficznych aplikacji działających w systemach Windows, macOS, Android i iOS.

Zanim zaczniesz pisać i uruchamiać aplikacje .NET MAUI, należy zainstalować **obciążenie .NET MAUI dla .NET**. Najprostszym sposobem na zrobienie tego w systemie Windows jest zainstalowanie Visual Studio 2022 i wybranie opcji .NET MAUI.

Można również zainstalować .NET MAUI z poziomu wiersza poleceń. Zazwyczaj wygląda to następująco:

dotnet workload install maui  $\ lub$  sudo dotnet workload install maui

Jeśli korzystasz z systemu macOS lub Linux, konieczne może być użycie opcji sudo, aby uruchomić program z podwyższonymi uprawnieniami. Jeżeli masz komputer Mac, musisz również **zainstalować XCode**. Ponadto możesz zainstalować pakiet Android SDK (choć jest to opcjonalne). Więcej informacji znajdziesz na stronie *https://learn.microsoft.com/dotnet/maui/get-started/installation?tabs=visual-studio-code*.

### Jeśli korzystasz z systemu Linux, będziesz potrzebować urządzenia z systemem Android do projektów .NET MAUI

.NET MAUI nie działa natywnie w systemie Linux. Jeśli masz urządzenie z Androidem, możesz debugować kod bezpośrednio w nim. Na podanej niżej stronie pokazano, jak skonfigurować urządzenie z systemem Android, aby można było podłączyć je do komputera i uruchamiać na nim aplikacje MAUI: https://learn.microsoft.com/dotnet/maui/android/device/setup.

Ponadto każdy projekt MAUI z tej książki ma wersję opartą na technologii Blazor, która pozwala stworzyć aplikację internetową działającą w przeglądarce. Aby uzyskać więcej informacji, pobierz podręcznik *Head First C# Blazor Learner's Guide* z naszej strony w serwisie GitHub. Możesz go pobrać jako bezpłatny plik PDF na stronie *https://github.com/head-first-csharp/fifth-edition*.

# Napiszmy grę!

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



Pozostała część tego rozdziału poświęcona jest tworzeniu projektu AnimalMatchingGame. Jest on zbudowany przy użyciu .NET MAUI, wieloplatformowego frameworka do tworzenia aplikacji desktopowych i mobilnych. W tej książce znajdziesz kilka projektów MAUI. Można je wykorzystać także do *nauki tworzenia stron internetowych* przy użyciu Blazora, rozbudowanego frameworka Microsoftu do budowania stron internetowych. Aby się z nim zapoznać, pobierz podręcznik *Head First C# Blazor Learner's Guide*, bezpłatny plik PDF z wersjami internetowymi każdego projektu MAUI z tej książki (https://github.com/head-first-csharp/fifth-edition).

Gra wyświetla szesnaście przycisków z rozmieszczonymi

losowo ośmioma parami



- \* 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 występują **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, ile jego pisanie.

00

### Dzielenie dużych projektów na mniejsze porcje

Naszym celem w tej książce jest pomóc Ci w nauce języka C#, ale także **w staniu się świetnym programistą**, a jedną z najważniejszych umiejętności, które rozwijają takie osoby, jest radzenie sobie z dużymi projektami. W tej książce będziesz tworzyć wiele projektów. Początkowo będą one stosunkowo niewielkie, ale w miarę postępów zaczniesz rozwijać coraz większe projekty. Wtedy pokażemy Ci, jak dzielić je na mniejsze części, nad którymi można pracować jedna po drugiej. Ten projekt nie jest wyjątkiem. Jest on dość duży, podobnie jak te, które będziesz rozwijać w dalszej części książki, dlatego wykonasz go w pięciu częściach.



Celem tego projektu jest pomoc w opanowaniu pisania w języku C# i korzystania z IDE. Jeśli napotkasz jakiekolwiek problemy z tym projektem, możesz obejrzeć kompletną instrukcję wideo na naszym kanale w serwisie YouTube: https://www.youtube.com/@headfirstcsharp.

Cały kod i plik PDF z tym rozdziałem możesz pobrać z naszej strony w serwisie GitHub: https://github.com/head-first-csharp/fifth-edition.



# Zrelaksuj się

# Ten rozdział jest poświęcony nauce podstaw oraz opanowywaniu tworzenia projektów, edycji kodu i budowania gry.

Nie martw się, jeśli nie pojmujesz niektórych zagadnień. Do czasu zakończenia lektury zrozumiesz wszystko, co dzieje się w grze. Na razie wystarczy postępować zgodnie z instrukcjami krok po kroku, aby uruchomić grę. Dzięki temu zdobędziesz solidne podstawy, które później pozwolą Ci rozszerzyć wiedzę.

# Proces budowania gry

Zbudujesz swoją grę w dopasowywanie zwierząt przy użyciu .NET MAUI (skrót od .NET Multi-platform App UI; stosowana jest też nazwa MAUI). MAUI to technologia, której można używać do tworzenia aplikacji w języku C#, działających natywnie jako aplikacje desktopowe w systemach Windows i macOS oraz jako aplikacje mobilne na urządzeniach mobilnych z systemami Android i iOS.

W pozostałej części tego rozdziału przeprowadzimy Cię przez proces tworzenia gry. Zrobisz to w kilku oddzielnych etapach:



# Najpierw utworzysz nowy projekt .NET MAUI w Visual Studio.

Wcześniej utworzyłeś nową aplikację konsolową. Teraz zbudujesz nową aplikację MAUI.

#### 2 Następnie użyjesz języka XAML do zaprojektowania strony.

Poszczególne ekrany w aplikacjach MAUI nazywane są **stronami**. Do ich projektowania stosuje się język XAML, który służy do definiowania sposobu działania tych stron.

#### 3 Napiszesz kod C#, aby dodawać losowe emoji ze zwierzętami do strony.

Gdy aplikacja zostanie uruchomiona po raz pierwszy, wykona kod wyświetlający szesnaście przycisków z ośmioma parami emoji ze zwierzętami w losowej kolejności.

### 4 Sprawisz, że gra będzie działać.

Gra musi wykrywać, kiedy użytkownik klika pary emoji, śledzić pary i kończyć rozgrywkę, gdy gracz dopasuje wszystkie zwierzęta. Napiszesz potrzebny do tego kod.

# 5 Na koniec dzięki dodaniu licznika czasu sprawisz, że gra stanie się bardziej ekscytująca.

Licznik czasu zostanie uruchomiony w momencie rozpoczęcia gry przez gracza i będzie śledził, ile czasu zajmie znalezienie wszystkich ośmiu par zwierząt. Ten projekt może zająć od 20 minut do ponad godziny, w zależności od tego, jak szybko piszesz. Nauka jest skuteczniejsza, gdy nie czujesz presji czasu, więc nie spiesz się.



MainPage.xaml











# Tworzenie projektu .NET MAUI w Visual Studio

Aplikację .NET MAUI można utworzyć w Visual Studio w podobny sposób jak aplikację konsolową, którą wygenerowałeś na początku rozdziału. Należy użyć przycisku *Utwórz nowy projekt* wyświetlanego przy pierwszym otwarciu Visual Studio. Jeśli program jest już otwarty, wybierz opcję *Nowy/Projekt* (*Ctrl+Shift+N*) z menu *Plik*, aby wyświetlić okno *Utwórz nowy projekt*.



Wybierz szablon projektu **Aplikacja platformy .NET MAUI** i kliknij przycisk *Dalej.* Visual Studio wyświetli monit o nazwę projektu, podobnie jak podczas tworzenia projektu aplikacji konsolowej.

Podaj nazwę projektu, AnimalMatchingGame, a następnie kliknij przycisk Dalej.

Konfiguruj nowy projekt										
Aplikacja platformy .NET MAUI	C#	Android	iOS	Mac Catalyst	macOS	MAUI	Mobilny	Tizen	Windows	
Nazwa projektu AnimalMatchingGame					Kor	nieczni V przec	e wpisz iwnym r ze zr	nazwę azie k zutam	AnimalM od nie bę i z książki	latchingGame. dzie zgodny i.

Na koniec Visual Studio poprosi o wybranie wersji platformy .NET. Wybierz najnowszą wersję, tak jak podczas tworzenia projektu aplikacji konsolowej. Następnie kliknij przycisk *Utwórz*, aby utworzyć nowy projekt platformy .NET MAUI.

### Tworzenie projektu platformy .NET MAUI w Visual Studio Code

Jeśli korzystasz z Visual Studio Code, tworzenie projektu platformy .NET MAUI odbywa się bardzo podobnie jak generowanie projektu aplikacji konsolowej z początku rozdziału. Najpierw **zamknij bieżącą aplikację** za pomocą opcji *File/Close Folder* (*Ctrl+K F* lub #+K F). *Zamknięcie katalogu jest bardzo ważne*. Jeśli o tym zapomnisz, dodasz nowy projekt do tego samego rozwiązania.

Następnie utwórz projekt aplikacji platformy .NET MAUI. Użyj kombinacji Ctrl+Shift+P lub  $\Im+\Re+P$ , aby **otworzyć paletę poleceń**. Wybierz polecenie **.NET**: **NEW Project**, aby utworzyć nowy projekt. VSCode wyświetli pytanie o typ projektu.

Wybierz typ projektu .NET MAUI App. Możesz wpisać "MAUI", aby przefiltrować opcje.



Koniecznie wpisz nazwę AnimalMatchingGame. W przeciwnym razie kod nie będzie zgodny ze zrzutami z książki.

VSCode wyświetli prośbę o nadanie projektowi nazwy. Nazwij projekt AnimalMatchingGame.

Name the new project

AnimalMatchingGame

Press 'Enter' to confirm your input or 'Escape' to cancel

VSCode wyświetli monit o podanie katalogu. Wybierz katalog domyślny.

Projekt powinien być teraz widoczny w sekcji *Solution Explorer* w dolnej części panelu eksploratora.

Gdy zechcesz uruchomić projekt, wykonaj następujące czynności (różnią się one od uruchamiania aplikacji konsolowej):

- 1. Rozwiń sekcję Solution Explorer w panelu eksploratora.
- 2. Rozwiń sekcję pliku *MainPage.xaml*, aby wyświetlić plik *MainPage. xaml.cs* (możliwe, że już jest ona rozwinięta).
- 3. Kliknij MainPage.xaml.cs, by zaznaczyć plik.
- Otwórz paletę poleceń (*Ctrl+Shift+P* lub ☆+ૠ+P) i wybierz opcję Debug: Start Debugging. Możesz także otworzyć plik MainPage. xaml i nacisnąć F5 lub wybrać opcję Start Debugging z menu Run.
- VSCode może wyświetlić monit o wybranie debugera. Wybierz opcję .NET MAUI. Gdy to zrobisz, aplikacja powinna zacząć działać w nowym oknie.

Select debugger	
C#	Suggested
.NET MAUI	
Install an extension for C#	



Sekcja Solution Explorer może się znajdować na samym dole okna eksploratora.

jesteś tutaj > 23 Pole ksi k

# Uruchamianie nowej aplikacji .NET MAUI

W Visual Studio: kliknij przycisk 🕨 Windows Machine 🗸 na pasku narzędzi lub wybierz opcję Rozpocznij debugowanie (F5) w menu Debuguj.

W Visual Studio Code: otwórz plik MainPage.xaml i wybierz opcję Start Debugging (F5) w menu Run. Jeśli pojawi się monit o wybranie debugera, wybierz z listy .NET MAUI. W systemie macOS może się pojawić ostrzeżenie, że AnimalMatchingGame pochodzi od niezidentyfikowanego dewelopera, a także pytanie, czy na pewno chcesz go otworzyć. Kliknij Otwórz mimo to.

IDE *skompiluje* kod, co oznacza, że przekształci go w program wykonywalny, który może zostać uruchomiony przez system operacyjny. Następnie uruchomi aplikacie:

> AnimalMatchingGame Home

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

Trób tol

Te zrzuty zostały wykonane z użyciem

Jeśli w systemie macOS rozszerzenie .NET MAUI w VSCode wyświetli komunikat Debugging cancelled: Xcode not found, należy zainstalować lub zaktualizować Xcode, pakiet narzędzi programistycznych rozwijanych przez firmę Apple. Jeśli nie został on dodany w trakcie konfigurowania Visual Studio Code i rozszerzeń dla języka C#, możesz zainstalować go po pobraniu ze sklepu App Store. Pamiętaj, aby otworzyć pakiet Xcode i zaakceptować umowę licencyjną. W przeciwnym razie może się pojawić błąd przy próbie debugowania aplikacji MAUI w Visual Studio Code.

...



### Zatrzymywanie aplikacji MAUI

Aplikacje można zatrzymać przez zamknięcie jej okna. Można również wybrać opcje Zatrzymaj debugowanie (Shift+F5) z menu Debuguj w Visual Studio lub opcje Stop Debugging z menu Run w VSCode. Jeszcze inna możliwość to kliknięcie kwadratowego przycisku Stop na pasku narzędzi w IDE.

Aplikacje można uruchomić lub zatrzymać w dowolnym momencie. Jeśli w kodzie C# lub XAML występuja błedy składni (na przykład literówki lub nieprawidłowe słowa kluczowe), **IDE nie uruchomi aplikacji**.

Wskazówki dotyczące uruchamiania aplikacji znajdziesz na stronie https://github.com/head-first-csharp/fifth-edition.

24 Rozdział 1. Kup ksi k

# Aplikacje MAUI działają na wszystkich urządzeniach

MAUI to **wieloplatformowy framework** do tworzenia aplikacji graficznych. Oznacza to, że gotowe programy można uruchamiać na urządzeniach z systemami Android i iOS. Wiele rozdziałów w tej książce zawiera projekty .NET MAUI, co pomoże Ci nauczyć się tworzyć aplikacje graficzne.

Aplikacje MAUI można uruchamiać na urządzeniu z systemem Android bezpośrednio z poziomu Visual Studio. Na stronie *https://learn.microsoft.com/dotnet/maui/android/device/setup* wyjaśniono, jak skonfigurować urządzenie z systemem Android, aby można było podłączyć je do komputera i uruchamiać na nim aplikacje MAUI.

Aplikacje MAUI można również uruchamiać na urządzeniach z systemem iOS, ale wymaga to nieco więcej konfiguracji i poniesienia kosztów, ponieważ trzeba dołączyć do programu Apple Developer Program. Ze strony *https://learn.microsoft.com/dotnet/maui/ios/device-provisioning* dowiesz się, co należy zrobić.

# Aplikacje MAUI projektuje się za pomocą języka XAML

XAML (X wymawia się jak KS i rymuje się z "kamel") to język znaczników, którego będziesz używać do tworzenia interfejsów użytkownika dla aplikacji MAUI. XAML opiera się na języku XML (więc jeśli kiedykolwiek używałeś HTML-a, będzie Ci łatwiej). Oto przykład znacznika XAML dla przycisku:

<Button Text="Click" Clicked="Button\_Click" />

Ta książka jest poświęcona nauce języka C#, więc omawiamy XAML tylko w takim zakresie, aby umożliwić Ci tworzenie atrakcyjnie wyglądających aplikacji MAUI. Zadbamy też o zapewnienie Ci solidnych podstaw do dalszej nauki.

### Czy widzisz błędy lub natrafiasz na problemy w Visual Studio Code?

Jeśli podczas próby uruchomienia kodu pojawi się okno błędu (na przykład z komunikatem "Android SDK: Wymagana instalacja" lub ostrzeżeniem dotyczącym licencji), oznacza to, że nadal musisz wykonać kilka kroków instalacji. Otwórz wtedy stronę:

https://learn.microsoft.com/dotnet/maui/get-started/installation?tabs=visual-studio-code

Wykonaj wszystkie kroki opisane na tej stronie. Upewnij się, że zainstalowane są wszystkie rozszerzenia VSCode, obciążenia dla .NET i .NET MAUI, pakiet Android SDK, a także najnowszy pakiet XCode (jeśli używasz komputera Mac). Jeżeli nadal napotykasz problemy, utwórz nowy projekt .NET MAUI i sprawdź, czy w oknie terminala zaraz po otwarciu projektu nie pojawiają się komunikaty o błędach. Możesz poszukać instrukcji, jak zaakceptować licencję. Postępuj dokładnie według tych instrukcji. Konieczne może być zainstalowanie najnowszej wersji OpenJDK: https://learn.microsoft.com/java/openjdk/download.

Jeśli w systemie Windows nadal pojawiają się błędy dotyczące licencji, otwórz paletę poleceń i wybierz .NET MAUI: Configure Android, a następnie Review Android Licenses, aby zaakceptować licencje. Może to wymagać uruchomienia VSCode w trybie administratora. Możesz także wybrać "How to Configure Android", aby wyświetlić stronę internetową z kompletnymi instrukcjami.

Jeśli korzystasz z komputera Mac i podczas uruchamiania aplikacji pojawia się którykolwiek z następujących błędów: "No debug target available, skipping debugging", błąd dotyczący systemu Android lub iOS albo błąd dotyczący zakończenia zadań przed uruchomieniem, naciśnij kombinację  $\hat{T} + \Re + \Re$  aby otworzyć paletę poleceń, kliknij **.NET MAUI: Pick macOS Device** i wybierz swój komputer z listy, a następnie **otwórz plik MainPage.xaml** i **naciśnij F5**, by ponownie uruchomić aplikację.

Jeśli MAUI nadal sprawia trudności, Microsoft udostępnia stronę pomocną w rozwiązywaniu problemów: https://learn.microsoft.com/dotnet/maui/troubleshooting.





### Oto strona, którą zbudujesz

Kiedy rozpoczynasz projekt, pierwszą rzeczą, którą zawsze warto zrobić, jest poświęcenie kilku minut na zrozumienie ogólnego obrazu. Co zamierzasz stworzyć? Jak program ma działać? Przyjrzyjmy się stronie, którą zaraz zbudujesz.

Po otwarciu aplikacji zbudowanej za pomocą .NET MAUI pierwszą rzeczą, jaką widzisz, jest **strona**, z którą użytkownik wchodzi w interakcję. Znajdują się na niej **kontrolki** czy graficzne widżety, takie jak przyciski i etykiety. Tworzą one interfejs użytkownika, z którym można wchodzić w interakcje. Oto strona, którą zaprojektujesz: Do tworzenia układu strony użyjesz XAML-a. Wielu programistów C# uważa znajomość XAML-a za jedną z podstawowych umiejętności, a wiele ofert pracy w C# wymaga opanowania przynajmniej niektórych aspektów XAML-a, dlatego chcemy zadbać o zapewnienie Ci solidnych podstaw w tym zakresie.



Jeśli zamkniesz którykolwiek z plików, możesz otworzyć ie

ponownie za pomoca

dwukrotnego klikniecia

w Eksploratorze

# Rozpocznij edycję kodu XAML

Aby rozpocząć edycję plików z kodem, kliknij je dwukrotnie w *Eksploratorze rozwiązań* (w VSCode wystarczy pojedyncze kliknięcie). Będziemy pracować z dwoma plikami: *MainPage.xaml* (który zawiera kod XAML) i *MainPage.xaml.cs* (który zawiera kod C# gry). Tak to wygląda w Visual Studio:





#### Visual Studio Code ma ograniczone możliwości edycji kodu XAML.

Gdy używasz Visual Studio Code do edycji kodu XAML z pliku MainPage.xaml, możesz zauważyć, że brakuje niektórych funkcji omawianych w tym rozdziale. W trakcie edycji kodu XAML w Visual Studio masz do dyspozycji bardzo wygodne funkcje:

- ★ Visual Studio ma przybornik z kontrolkami, które można przeciągnąć do edytora XAML, a także okno Właściwości ułatwiające zmianę ich właściwości.
- ★ Po dodaniu zdarzenia Visual Studio automatycznie tworzy procedurę do jego obsługi:

FontSize= <b>"Large"</b> Clicked=" <b>"</b> />	Visual Studio pomaga powiązać zdarzenie z procedurą do jego
ed". Użyj opcji "Przejdź do definicji", aby 🗗 <nowa proced<="" td=""><th>ra obsługi zdarzeń&gt; obsługi lub utworzyć nową metodę tego rodzaju.</th></nowa>	ra obsługi zdarzeń> obsługi lub utworzyć nową metodę tego rodzaju.

W czasie, gdy piszemy tę książkę, rozszerzenie .NET MAUI dla VSCode nie udostępnia tych funkcji, więc trzeba pisać kod ręcznie, zamiast wygenerować go za pomocą IDE

### Dodaj kod XAML przycisku i etykiety

Pierwsza rzecza, która zrobimy, jest zaprojektowanie strony gry. Znajdzie się na niej szesnaście przycisków do wyświetlania emoji ze zwierzetami, a także przycisk Jeszcze raz?, który umożliwia ponowne uruchomienie gry po jej zakończeniu.



#### (1) Usuń wszystko pomiędzy otwierającym i zamykającym znacznikiem VerticalStackLayout.

XAML jest językiem znaczników opartym na tagach. Oznacza to, że w kodzie XAML używa się znaczników do definiowania wszystkiego, co pojawia się w aplikacji. Oto przykładowy znacznik, który znajdziesz w początkowej części pliku MainPage.xaml:

<ScrollView>

Jest to znacznik otwierający. Pasujący do niego znacznik zamykający znajduje się na końcu pliku:

</ScrollView>

Znaczniki te dodaja do strony **kontrolke ScrollView**. Sprawia ona, że jeśli aplikacja znajduje sie w oknie mniejszym niż jego zawartość, wszystko pomiedzy znacznikami otwierającym i zamykającym tej kontrolki można przewijać w góre i w dół.

Znajdź otwierający znacznik VerticalStackLayout. Znajduje sie on kilka wierszy dalej i wyglada następująco:

```
<VerticalStackLayout
    Padding="30,0"
    Spacing="25">
```

Jeśli używasz innej wersji .NET niż my, kod XAML strony może się zaczynać nieco inaczej. Nic nie szkodzi – wystarczy zmienić kod XAML w taki sposób, aby dokładnie odpowiadał wersji przedstawionej w tym miejscu.

Nastepnie znajdź zamykający znacznik VerticalStackLayout:

</VerticalStackLayout>

Teraz starannie usuń wszystkie wiersze między tymi dwoma znacznikami. Kod XAML w pliku MainPage.xaml powinien teraz wyglądać następująco:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="AnimalMatchingGame.MainPage">
```

<ScrollView>

<VerticalStackLayout Padding="30,0" Spacing="25">

W następnym kroku umieścisz nowy kod XAML w tym miejscu, z którego usunąłeś wcześniejszy fragment.

</VerticalStackLayout> </ScrollView>

</ContentPage>



W czasie, gdy powstaje ten tekst, rozszerzenie .NET MAUI dla Visual Studio Code nie udostępnia przybornika. Jeśli korzystasz z Visual Studio Code, nie da się wykonać w kodzie XAML metodą "przeciągnij i upuść" niektórych zaawansowanych zmian, które są możliwe w Visual Studio. Dlatego trzeba starannie wpisać cały kod XAML wiersz po wierszu. Ale nie martw się, tak napisana aplikacja będzie działać identycznie.

#### ② Usuń kod C# powiązany z właśnie usuniętym kodem XAML.

Jeśli spróbujesz teraz uruchomić aplikację, Visual Studio wyświetli komunikat o błędzie i odmówi jej włączenia, ponieważ kod C# wymaga właśnie usuniętych elementów. **Rozwiń sekcję pliku** *MainPage.xaml* w sekcji *Solution Explorer*, otwórz plik *MainPage.xaml.cs* i znajdź następujący fragment kodu:

```
private void OnCounterClicked(object sender, EventArgs e)
```

Usuń go i następnych dziesięć wierszy kodu, aż do zamykającego nawiasu klamrowego } włącznie. Uważaj jednak, aby nie usunąć ostatniego zamykającego nawiasu } na końcu pliku. Następnie usuń ten wiersz kodu: int count = 0;.

Kod C# powinien teraz wyglądać następująco:



#### (3) Wróć do pliku MainPage.xaml i użyj przybornika, aby dodać przycisk "Jeszcze raz?".

Będziesz ponownie edytować kod XAML, więc **wróć do zakładki** *MainPage.xaml*. Jeśli nie widzisz panelu przybornika, rozwiń go przez kliknięcie zakładki z boku okna. **Dodaj kilka dodatkowych pustych wierszy** w miejscu, w którym usunąłeś kod pomiędzy otwierającym i zamykającym znacznikiem VerticalStackLayout. Następnie **przeciągnij kontrolkę Button z panelu przybornika** i upuść ją w jednym z dodanych wierszy.



Riiknij Zakładkę Przybornik z boku okna, aby ją rozwinąć. Jeśli jej nie widzisz, wybierz Przybornik z menu Widok. Kliknij przycisk pinezki w prawym górnym rogu, aby zapobiec zwinięciu przybornika.

```
Pomiędzy znacznikami VerticalStackLayout powinien się pojawić nowy znacznik, Button. Nie przejmuj się, jeśli odstępy lub wcięcia wyglądają nieco inaczej, ponieważ dodatkowe spacje lub wiersze nie mają w kodzie XAML znaczenia:
```

```
<VerticalStackLayout
Padding="30,0"
Spacing="25">
```

```
<Button Text="" />
</VerticalStackLayout>
```

Po przeciągnięciu kontrolki Button z przybornika do kodu Visual Studio dodaje ten znacznik. Jeśli używasz VSCode, przybornik może być niedostępny. W takiej sytuacji dokładnie przepisz kod. Jeśli korzystasz z Visual Studio Code, przybornik i okno właściwości mogą być niedostępne. Należy wtedy wpisać kod XAML w pliku MainPage.xaml, aby wyglądał dokładnie tak jak w książce.

#### (4) Dodaj właściwości do znacznika przycisku "Jeszcze raz?" w kodzie XAML.

Znaczniki w kodzie XAML mają **właściwości**, które pozwalają konfigurować sposób ich wyświetlania na stronie. **Okno właściwości** w Visual Studio ułatwia ich modyfikowanie.

Kliknij kod znacznika Button w pliku *MainPage.xaml*, tak aby kursor znajdował się gdzieś pomiędzy otwierającym < a zamykającym nawiasem >. Następnie spójrz na **okno Właściwości**. Zwykle jest ono zadokowane w prawym dolnym rogu środowiska Visual Studio. Jeśli go nie widzisz, wybierz opcję *Okno właściwości* z menu *Widok*. Upewnij się, że u góry znajduje się napis *Typ Button*. Informuje on, że edytujesz przycisk.

Znajdź właściwość Text i ustaw jej wartość na "Jeszcze raz?".

Następnie znajdź właściwość FontSize i ustaw jej wartość na "Large".

Właściwości		• ‡ ×	Właściwości 🗸 🗸 🗸	×
Typ Button			Typ Button	
		P	<b>/</b>	ρ
Rozmieść według: Nazwa 🔻			Rozmieść według: Nazwa 🔻	
StyleId			FontFamily	•
Text $\longrightarrow$ Je	eszcze raz?	-	FontSize Large V	
TextColor	Brak pędzla		GestureRecognizers (Kolekcja)	

Po zakończeniu edycji przycisku jego kod XAML powinien wyglądać następująco:

<Button Text="Jeszcze raz?" FontSize="Large" />

Znacznik Button ma teraz właściwości Text i FontSize.

#### (5) Edytuj ręcznie kod XAML przycisku, aby nadać mu nazwę.

Możesz także edytować kod XAML ręcznie. Na przykład jeśli wystąpią problemy z oknem Właściwości, możesz wpisać kod XAML bezpośrednio w edytorze. Koniecznie sprawdź, czy wszystkie nawiasy, cudzysłowy i inne elementy zostały dokładnie skopiowane. W przeciwnym razie kod nie zostanie uruchomiony!

W następnej części projektu napiszesz kod C#, który sprawi, że przycisk *Jeszcze raz*? będzie widoczny po zakończeniu gry i ukryty podczas jej trwania. Nadasz przyciskowi **nazwę**, której kod C# będzie mógł użyć do jego wyświetlenia lub ukrycia.

Użyj edytora, aby **dodać właściwość x:Name** i nadać przyciskowi nazwę. Kod powinien wyglądać następująco:

<Button x:Name="PlayAgainButton" Text="Jeszcze raz?" FontSize="Large" />

### Znaczniki w kodzie XAML mają właściwości, które pozwalają konfigurować sposób ich wyświetlania na stronie.

Upewnij się, że modyfikujesz przycisk.

#### 6 Dodaj obsługę zdarzeń, aby przycisk uruchamiał jakieś operacje.

Kliknięcie przycisku powoduje wykonanie kodu C# zwanego **procedurą obsługi zdarzeń**. Visual Studio ułatwia dodanie takiej procedury. Umieść kursor myszy tuż przed znakami /> na końcu znacznika Button i zacznij wpisywać tekst **Clicked**. Visual Studio wyświetli okno IntelliSense:

FontSize="Large"	C1:	비 />
	5	Clicked
	۶	Clip

Wybierz opcję Clicked z listy i kliknij ją lub naciśnij klawisz *Enter*. Visual Studio wyświetli następujący komunikat:

```
FontSize="Large" Clicked=""/>
```

Znacznik w kodzie XAML może zajmować jeden wiersz lub kilka wierszy. Pamiętaj, aby podział wiersza wypadał w miejscu spacji (ale nie w tekście "Jeszcze raz?").

```
Naciśnij Enter, aby dodać nową procedurę obsługi zdarzeń. Znacznik XAML powinien teraz wyglądać następująco:
```

```
<Button x:Name="PlayAgainButton" Text="Jeszcze raz?" FontSize="Large"
```

```
Clicked="PlayAgainButton_Clicked" />
```

Wróć do zakładki MainPage.xaml.cs. Zobaczysz dodany przez Visual Studio kod:

```
private void PlayAgainButton_Clicked(object sender, EventArgs e)
```



}

Jeśli korzystasz z Visual Studio Code, wyskakujące okno <Nowa procedura obsługi zdarzeń> może się nie pojawić. Należy wtedy ręcznie dodać potrzebny kod do pliku MainPage.xaml.cs, a aplikacja będzie działać poprawnie.



Ćwiczenia dają okazję do samodzielnej praktyki. Koniecznie wykonuj je wszystkie. Stanowią one ważną część książki. Jeśli ćwiczenie jest częścią projektu, nie będzie on działać, dopóki nie wykonasz poprawnie zadania. Nie martw się jednak, zawsze podajemy rozwiązanie. Jeżeli utkniesz, zawsze dopuszczalne jest podejrzenie rozwiązania!

#### Dodaj kontrolkę Label do strony XAML.

Wróć do zrzutu ekranu gry przedstawiającego przycisk "Jeszcze raz?". Zauważ, że nad przyciskiem znajduje się tekst wyświetlający czas, który upłynął. To kontrolka Labe1, czyli etykieta. Do Ciebie należy dodanie reprezentującego ją znacznika. Oto co należy zrobić:

- 1. Przejdź do zakładki MainPage.xaml.
- 2. Otwórz przybornik i **przeciągnij kontrolkę Label** do kodu XAML. Upewnij się, że znajdzie się ona bezpośrednio pod przyciskiem, analogicznie jak w kroku 3. podczas dodawania przycisku.
- 3. Użyj okna właściwości, aby ustawić **właściwości Text na "Czas: 0,0 s" i FontSize na "Large"**, podobnie jak w kroku 4. podczas dodawania przycisku.
- 4. Zmodyfikuj kod XAML ręcznie i **ustaw właściwość x:Name na "TimeElapsed"**, tak jak w kroku 5. podczas dodawania przycisku.

Jeśli używasz VSCode i nie masz przybornika, przejdź bezpośrednio do rozwiązania ćwiczenia i starannie przepisz kod znacznika <Label ... />.



namespace AnimalMatchingGame;

```
public partial class MainPage : ContentPage
```

public MainPage() InitializeComponent();

Zanim przejdziesz dalej, upewnij się, że w Twoim środowisku kod XAML i C# dokładnie odpowiada naszemu, a aplikacja po uruchomieniu wyglada jak na przedstawionym zrzucie ekranu.

```
private void PlayAgainButton Clicked(object sender, EventArgs e)
                                                                 Gra w dopasowywanie zwierząt
}
Jeśli teraz uruchomisz aplikację, powinna wyglądać tak.
                                                                               leszcze raz?
           Naciśnij F5, aby ponownie uruchomić aplikację.
                                                                   Czas: 0.0 s
           W VSCode trzeba przejść z powrotem do kodu
           XAML w pliku MainPage.xaml, a następnie
           nacisnąć ten klawisz.
```

#### <sub>Nie ma</sub> głupich pytań

#### P: Czym dokładnie jest "strona" w aplikacji platformy MAUI?

**O**: Aplikacja platformy .NET MAUI jest zwykle zbudowana z jednej lub większej liczby **stron**, czyli pojedynczych ekranów, które mają różne układy i zawierają **kontrolki**, na przykład etykiety i przyciski. Niektóre aplikacje MAUI mają wiele stron i możliwa jest nawigacja między nimi. Aplikacja AnimalMatchingGame będzie miała tylko jedną stronę z 16 przyciskami zwierząt, przyciskiem "Jeszcze raz?" i etykietą pokazującą upływający czas.

# $\mathcal{P}$ : A więc przyciski i etykiety są kontrolkami?

G: Tak. Wszystko, co widzisz na stronie MAUI, jest kontrolką (w tym sama strona, która jest kontrolką ContentPage). Niektóre kontrolki służą do nadawania stronie określonego wyglądu. Dotyczy to na przykład kontrolki VerticalStackLayout, która powoduje rozmieszczanie innych elementów jeden nad drugim. Inne kontrolki, na przykład Button i Label, służą do wyświetlania widocznych na stronie widżetów, z którymi użytkownik może wchodzić w interakcje. Więcej o kontrolkach piszemy w następnym rozdziale.

#### P: Wygląda na to, że niektóre kontrolki zawierają inne, na przykład Vertical StackLayout w omawianej aplikacji obejmuje przycisk i etykietę. Jak to działa?

Scontrolka układu, na przykład VerticalStackLayout, po umieszczeniu na stronie jest niewidoczna. Służy do tego, aby inne kontrolki na stronie były wyświetlane w określony sposób. Wspomniana kontrolka sprawia, że inne są rozmieszczane jedna nad drugą. Potrzebny jest sposób na poinformowanie MAUI, które kontrolki na stronie mają być tak ułożone. Aby to zrobić, należy **zagnieździć** te elementy w kontrolce VerticalStackLayout. Wymaga to umieszczenia ich znaczników między otwierającym znacznikiem <VerticalStackLayout> a zamykającym znacznikiem </VerticalStackLayout>.

#### P: Dlaczego niektóre znaczniki, na przykład <ScrollView>, mają zamykający znacznik </ScrollView>, a inne, na przykład <Button>, go nie mają?

**O:** W kontrolce Button nie trzeba zagnieżdżać żadnych innych elementów, więc nie jest konieczne, aby miała znacznik zamykający. Zamiast tego wystarczy zakończyć znacznik za pomocą znaków />. Powstaje wtedy znacznik **samozamykający**.

Ramki "Wysil szare komórki" zawierają informacje, które mają zachęcić Cię do myślenia. Gdy natrafisz na taką ramkę, nie przechodź do następnego fragmentu. Poświęć kilka minut i zastanów się nad pytaniem. Pomoże Ci to szybciej przyswoić materiał!





Aplikacja wygląda dobrze, ale teraz trzeba dodać kilka przycisków. Jak myślisz, jak to zrobić? Jak sądzisz, co trzeba dodać do kodu XAML, aby wyświetlić 16 przycisków w układzie z czterema rzędami po cztery przyciski?

# Użyj układu FlexLayout, aby utworzyć siatkę przycisków ze zwierzętami

Kod XAML strony obejmuje obecnie trzy znaczniki, które określają jej układ. Nadrzędny jest znacznik ContentPage, który wyświetla cały widok. Zawiera on znacznik ScrollView. Wszystkie elementy zagnieżdżone między początkowym i końcowym znacznikiem ScrollView są przewijane, jeśli użytkownik wyjdzie poza widoczną część strony. Wewnątrz znajduje się kontrolka VerticalStackLayout, która powoduje, że wszystkie elementy pomiędzy jej znacznikami początkowym a końcowym są rozmieszczane jeden nad drugim w kolejności ich występowania. Wewnątrz wszystkich tych znaczników znajdują się samozamykające znaczniki Button i Label.

Teraz dodasz kontrolkę **FlexLayout**, która rozmieszcza wszystkie zagnieżdżone elementy w rzędach z przenoszeniem ich do następnego wiersza, tak aby wszystkie kontrolki zmieściły się na szerokość. Wewnątrz znacznika FlexLayout dodasz 16 znaczników Button. Wyświetlisz je w siatce 4 na 4. Szerokość każdego przycisku ustaw na 100, a szerokość kontrolki FlexLayout na 400, dzięki czemu w każdym rzędzie zmieszczą się dokładnie cztery przyciski.





To ćwiczenie wygląda na rozbudowane! Ale nie martw się, wykonuj je krok po kroku. Wiemy, że dasz radę! I pamiętaj, że podglądanie rozwiązania *nie jest oszukiwaniem...* Co więcej, zapoznanie się z nim jest świetnym sposobem na naukę.

Pora dokończyć projektowanie strony. W tym ćwiczeniu umieścisz układ FlexLayout pod etykietą dodaną w poprzednim ćwiczeniu. Następnie ustawisz jego właściwości. W dalszej kolejności dodasz przycisk. Na koniec skopiujesz kod XAML przycisku i wkleisz go jeszcze 15 razy, dzięki czemu otrzymasz w sumie 16 przycisków na stronie. Jeśli używasz VSCode i nie masz przybornika, przepisz kod XAML dokładnie tak, jak pojawia się w instrukcjach, zamiast przeciągać kontrolki z przybornika.

#### Dodaj dodatkowe miejsce dla kontrolki FlexLayout

Przyjrzyj się uważnie ostatniemu zrzutowi ekranu. Pokazuje on, jak działa cała strona. Teraz wróć do Visual Studio i spójrz na kod XAML strony, po czym określ, gdzie dokładnie powinna się znajdować kontrolka FlexLayout — tuż pod znacznikiem <Label ... />.

Teraz umieść kursor w tym miejscu i naciśnij *Enter* kilka razy, aby zapewnić sobie miejsce na przeciągnięcie kontrolki FlexLayout.

#### Dodaj kontrolkę FlexLayout tuż pod kontrolką Label

- Otwórz przybornik i przeciągnij kontrolkę FlexLayout do kodu XAML Upewnij się, że zostanie ona umieszczona bezpośrednio pod etykietą, w dodanym przed chwilą nowym miejscu. Znacznik będzie wyglądał następująco: <FlexLayout></FlexLayout>.
- 2. Umieść kursor między znacznikami > i < pośrodku właśnie wygenerowanego kodu XAML i **dodaj kilka pustych wierszy** między znacznikami otwierającym a zamykającym (w dalszej części ćwiczenia przeciągniesz w to miejsce przycisk).
- 3. Umieść kursor bezpośrednio na otwierającym znaczniku **<FlexLayout>**. Upewnij się, że okno Właściwości pokazuje, że typ kontrolki to FlexLayout.
- 4. Użyj okna właściwości, aby ustawić właściwość Wrap na "Wrap" i właściwość MaximumWidthRequest na "400".
- 5. Zmodyfikuj kod XAML ręcznie i **ustaw właściwość x:Name na "AnimalButtons"**, tak jak w poprzednim ćwiczeniu.

#### Dodaj pierwszy przycisk w kontrolce FlexLayout

- Otwórz przybornik i przeciągnij kontrolkę Button do kodu XAML Upewnij się, że zostanie ona umieszczona w nowym miejscu między otwierającym a zamykającym znacznikiem FlexLayout. Kod będzie wyglądał następująco: <Button Text="" />.
- 2. Umieść kursor wewnątrz znacznika Button. Upewnij się, że okno właściwości pokazuje, iż typ kontrolki to Button.
- Użyj okna właściwości, aby ustawić właściwość HeightRequest przycisku na "100", właściwość WidthRequest na "100", a właściwość FontSize na "60". Lista rozwijana w oknie właściwości nie zawiera liczb. Możesz wpisać "60" w oknie lub wybrać opcję Caption z listy rozwijanej, aby ustawić rozmiar czcionki.
- 4. Zmodyfikuj kod XAML przycisku i **usuń właściwość Text**. W tym celu zaznacz ją w edytorze kodu i wciśnij *Delete*. Kursor powinien się teraz znajdować wewnątrz kontrolki Button.
- 5. Nie przenoś kursora i ręcznie zmodyfikuj kod XAML, aby ustawić właściwość BackgroundColor na "LightBlue", właściwość BorderColor na "Black", a właściwość BorderWidth na "1". Wyskakujące okienko IntelliSense programu Visual Studio pomoże Ci dopasować kolory (jeśli używasz Visual Studio Code, to pomocne okienko się nie pojawi).
- 6. Dodaj **obsługę zdarzeń Clicked**, tak jak dla przycisku PlayAgainButton. **Wybierz opcję** <**Nowa procedura zdarzeń**> z listy rozwijanej, aby utworzyć nową procedurę obsługi zdarzenia w kodzie C#. Użyj nazwy domyślnej Button\_Clicked.

#### Dodaj pozostałe przyciski

Jeśli używasz VSCode, konieczne może być ręczne wprowadzenie kodu w pliku – MainPage.xaml.cs. Nazwa procedury będzie podobna do PlayAgainButton\_Clicked, tylko bez członu "PlayAgain".

Skopiuj właśnie dodany znacznik **<Button** ... />. Następnie **wklej pod nim 15 identycznych znaczników**. Teraz w kontrolce FlexLayout tuż pod kontrolką Label powinno się znajdować w sumie 16 identycznych znaczników Button. Uruchom aplikację. Jej wygląd powinien być zgodny ze zrzutem ekranu.

#### rozwiązanie ćwiczenia



# Ćwiczenie

#### Rozwiązanie

Pora dokończyć projektowanie strony. W tym ćwiczeniu umieścisz układ FlexLayout pod etykietą dodaną w poprzednim ćwiczeniu. Następnie ustawisz jego właściwości. W dalszej kolejności dodasz przycisk. Na koniec skopiujesz kod XAML przycisku, dzięki czemu otrzymasz w sumie 16 przycisków na stronie.

Jeśli kroki opisane w ćwiczeniu zostały wykonane poprawnie, kod XAML w pliku MainPage.xaml powinien wyglądać następująco:

```
<?xml version="1.0" encoding="utf-8" ?>
                                                                                       Upewnij sie
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                                                                                       że znacznik
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                                                                                       otwierający
                                                                                       FlexLavout
              x:Class="AnimalMatchingGame.MainPage">
                                                                                       znajduje sie tuż
                  Jeśli wybierzesz dla projektu inną nazwę, zobaczysz ją w tym miejscu.
                                                                                       pod etykieta.
  <ScrollView>
                                                                                       a właściwości są
    <VerticalStackLavout
                                                                                       zgodne z podanymi.
                                       Wygląda to na dużą ilość kodu XAML, ale jego
                                                                                       Pamietaj, aby
         Padding="30.0"
                                       wiekszość stanowi 16 identycznych znaczników
                                                                                       ustawić szerokość
         Spacing="25">
                                       Button, które zostały skopiowane i wklejone.
                                                                                       maksymalna
                                                                                       zamiast minimalnei
    <Button x:Name="PlayAgainButton" Text="Jeszcze raz?" FontSize="Large"
                                                                                       W przeciwnym
                     Clicked="PlayAgainButton Clicked" />
                                                                                       razie przyciski nie
                                                                                       utworza siatki 4×4.
    <Label x:Name="TimeElapsed" Text="Czas: 0,0 s" FontSize="Large" />
    <FlexLayout x:Name="AnimalButtons" Wrap="Wrap" MaximumWidthRequest="400">
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
         HeightRequest="100" WidthRequest="100" FontSize="60" Clicked="Button Clicked"/>
      <Button BackgroundColor="LightBlue" BorderColor="Black" BorderWidth="1"
```

Wygląda to na dużą ilość kodu XAML, ale jego większość zajmuje ten sam znacznik <Button ... />, który został skopiowany i wklejony 16 razy. identycznie. Właściwości mogą być podane w innej kolejności.



x:Name dokładnie odpowiadają tym z naszego rozwiązania. Kod C#, który napiszesz, będzie korzystał z tych właściwości.

#### Znajdujesz się tutaj



# Napisz kod C#, aby dodać zwierzęta do przycisków

Czytasz tę książkę, aby nauczyć się języka C#. Wszystkie przygotowania są już zakończone: projekt został utworzony, podobnie jak projekt strony aplikacji. Teraz nadszedł czas, **aby zacząć pisać kod C#**.

Udostępnimy cały kod projektu i pokażemy, gdzie się on znajduje. Celem jest jednak *rozpoczęcie nauki języka C#*, dlatego pomożemy Ci również zrozumieć, jak działa aplikacja. Dzięki temu zdobędziesz solidne podstawy pozwalające rozpocząć samodzielne pisanie kodu.

Dodasz kod, który będzie uruchamiany za każdym razem, gdy użytkownik kliknie przycisk "Jeszcze raz?". Oto co będzie robił ten kod:



Kod C#

Metodv

Instrukcje

Każda instrukcja

w C# kończy się

średnikiem (;).

# Rozpocznij edycję procedury obsługi zdarzeń PlayAgainButton

Podczas pisania kodu XAML dla przycisku "Jeszcze raz?" dodano obsługę zdarzeń:

Visual Studio dodało wtedy właściwość Clicked="PlayAgainButton\_Clicked" do znacznika w kodzie XAML przycisku. Dodało również poniższy kod C# do pliku *MainPage.xaml.cs*:

```
private void PlayAgainButton_Clicked(object sender, EventArgs e)
{
}
```

Jest to **metoda**. Kod C# składa się z instrukcji, czyli konkretnych zadań, które aplikacja ma wykonać. Instrukcje te są łączone w metody. Metody mają nazwę. Ta konkretna metoda nosi nazwę PlayAgainButton\_Clicked.

Visual Studio wygenerowało tę metodę automatycznie po dodaniu procedury obsługi zdarzeń Clicked do kodu XAML. W ten sposób zapewniane jest miejsce na dodanie instrukcji, które informują, co aplikacja ma zrobić po kliknięciu przycisku "Jeszcze raz?".

# Dodaj instrukcję C# do metody obsługi zdarzenia

Umieść kursor w wierszu pomiędzy otwierającym nawiasem klamrowym ({) a zamykającym nawiasem klamrowym (}) metody. Następnie zacznij wpisywać następujący wiersz kodu, aby przyciski zwierząt były widoczne:

```
AnimalButtons.IsVisible = true;
```



Podczas pisania zobaczysz niektóre z zaawansowanych narzędzi środowiska Visual Studio, które pomagają w pisaniu kodu:



Podczas wpisywania kodu Visual Studio może wyświetlać sugestie dotyczące uzupełnienia całej instrukcji. Jest to wysoce zaawansowana funkcja o nazwie *IntelliCode*. Wykorzystuje ona system sztucznej inteligencji wyuczony na milionach wierszy kodu, aby generować sugestie. Często może się wydawać, że funkcja ta potrafi czytać w myślach!

VSCode może nie wyświetlać takich sugestii. Jest to funkcja środowiska Visual Studio.

### Dodai koleine instrukcie do metodu obsługi zdarzenia

Gdy gracz kliknie przycisk "Jeszcze raz?", aplikacja wyświetli przyciski ze zwierzetami, ukryje ten przycisk, a następnie wypełni przyciski z ośmioma rozmieszczonymi w losowej kolejności parami emoji ze zwierzetami. Aby to zrobić, należy dodać instrukcje do metody obsługi zdarzeń PlayAgainButton Clicked.





#### Dodaj instrukcje ukrywającą przycisk "Jeszcze raz?".

Pamietasz, jak użyliśmy właściwości x:Name w kodzie XAML, aby nadać nazwy przyciskowi "Jeszcze raz?" i układowi FlexLayout, który zawiera szesnaście przycisków ze zwierzetami? Wróć do tego kodu XAML. Układowi FlexLayout nadaliśmy nazwe "AnimalButtons", a przed chwila właśnie dodałeś wiersz kodu, w którym ta nazwa jest używana.

Użyliśmy właściwości x:Name także do nadania przyciskowi "Jeszcze raz?" nazwy "PlayAgainButton". Teraz dodaj drugi wiersz kodu do metody obsługi zdarzenia:

```
private void PlayAgainButton Clicked(object sender, EventArgs e)
{
    AnimalButtons.IsVisible = true:
                                                    Wpisz ten wiersz kodu
                                                    bezpośrednio przed dodanym
    PlayAgainButton.IsVisible = false;
                                                    wcześniej poleceniem.
                                                                        Kod, który już znajduje
                                                                        się w plikach, wyróżniamy
                                                                        jaśniejszym kolorem.
Ta instrukcja ukrywa przycisk "Jeszcze raz?".
                                                                        aby łatwiej było zauważyć
                                                                        co należy dodać
```

#### 2 Ukryj przyciski ze zwierzętami w momencie uruchamiania aplikacji.

Przyjrzyj sie bliżej pierwszej instrukcji dodanej do metody obsługi zdarzenia. Wyświetla ona układ FlexLayout zawierający przyciski ze zwierzetami. Ale chwileczke – przecież ten element jest już widoczny! Pojawił się w momencie uruchamiania aplikacji. Trzeba coś z tym zrobić.

Wróć do kodu XAML w pliku MainPage.xaml i ustaw właściwość IsVisible na "false":

```
<FlexLayout x:Name="AnimalButtons" Wrap="Wrap"</pre>
             MaximumWidthRequest="400" IsVisible="false">
```

Czy widzisz, że ta sama właściwość IsVisible jest ustawiana zarówno w kodzie C#, jak i w kodzie XAML? Po uruchomieniu aplikacji fragment Isvisible="false" w kodzie XAML powoduje wyświetlenie strony bez układu FlexLayout i zawartych w nim przycisków. Po kliknieciu przycisku "Jeszcze raz?" pierwszy wiersz kodu w metodzie obsługi zdarzenia Clicked ustawia właściwość na true, co powoduje wyświetlenie na stronie układu FlexLayout z przyciskami.

Teraz aplikacja ukrywa przyciski ze zwierzętami po uruchomieniu. Gdy tylko gracz kliknie przycisk "Jeszcze raz?", aby rozpocząć grę, aplikacja ukryje wspomniany przycisk i wyświetli przyciski ze zwierzętami.

### Właściwości kontrolek można ustawiać zarówno w kodzie XAML, jak i w kodzie C#.

#### Uruchom aplikację i upewnij się, że na tym etapie działa.

Kiedy piszesz kod, nie tworzysz kompletnej aplikacji od początku do końca, aby dopiero potem uruchomić ją w celu sprawdzenia, czy działa. Wygląda to zupełnie inaczej! *Pisanie kodu to proces twórczy.* Poszczególne zadania można wykonać w kodzie na różne sposoby, a w wielu przypadkach jedyną metodą na upewnienie się, że program działa zadowalająco, jest wypróbowanie któregoś z tych sposobów. Jeśli efekt Ci się nie podoba, zastosuj inny sposób.

Ponadto w kodzie łatwo o **błędy składniowe**. Powstają one, gdy napiszesz coś, co nie jest prawidłowym kodem C#, na przykład niewłaściwie użyjesz słowa kluczowego lub symbolu albo podasz nieistniejącą nazwę. Przykładowo jeśli wpiszesz dodatkowy zamykający nawias klamrowy (}) na końcu metody, a następnie spróbujesz ją uruchomić, Visual Studio wyświetli błąd informujący, że nie może **skompilować** kodu (czyli przekształcić kodu C# na postać, którą komputer może wykonać).

#### Co to wszystko oznacza?

Mianowicie to, że będziesz *nieustannie uruchamiać rozwijane aplikacje*. I nie ma w tym nic złego! Zupełnie normalne jest uruchamianie aplikacji nawet po wprowadzeniu drobnych zmian, aby sprawdzić wynik modyfikacji. Im łatwiej będzie Ci przychodzić uruchamianie aplikacji, tym mocniejsze będzie poczucie, że możesz eksperymentować i wprowadzać zmiany, a także więcej będziesz mieć z tego przyjemności.

A więc **uruchom teraz swoją aplikację**. Upewnij się, że początkowo widoczny jest przycisk "Jeszcze raz?", a przyciski ze zwierzętami nie są wyświetlane. Kliknij przycisk "Jeszcze raz?" i sprawdź, czy aplikacja ukrywa go i wyświetla przyciski ze zwierzętami. Następnie zamknij aplikację (lub zatrzymaj ją z poziomu Visual Studio).

AnnublectoryCare		- D )
Gra w dopasowywanie z	wierząt	
	Jeszcze raz?	
Czas: 0,0 s		
	<b>^</b>	
-		
Gdy	uchamiasz aplikacie I	przycisk

Gdy uruchamiasz aplikację, przycisk "Jeszcze raz?" jest widoczny, ale przyciski ze zwierzętami są ukryte.



Kliknij przycisk "Jeszcze raz?", aby aplikacja go ukryła i wyświetliła przyciski ze zwierzętami

#### Podczas wprowadzania kodu C# nawet drobne błędy mogą mieć duże znaczenie.

**Uwaga!** Niektórzy twierdzą, że prawdziwym programistą stajesz się po tym, jak po raz pierwszy spędzisz wiele godzin na wyszukiwaniu błędnie umieszczonej kropki. Ponadto wielkość liter jest istotna. An imalButtons to coś innego niż an imalButtons. Dodatkowe przecinki, średniki, nawiasy i inne elementy mogą spowodować, że kod nie będzie działać lub, co jeszcze gorsze, sprawić, że aplikacja się skompiluje, ale będzie działać inaczej, niż powinna. Wspomagane przez sztuczną inteligencję funkcje IntelliSense i IntelliCode środowiska IDE mogą pomóc uniknąć tych problemów, ale nie zrobią wszystkiego za Ciebie. Musisz samodzielnie się upewnić, że kod jest poprawny i że robi to, czego oczekujesz.

### Dodaj zwierzęta do przycisków

Gra nie będzie ciekawa bez zwierząt, które można kliknąć. Zaktualizujmy metodę obsługi zdarzenia przycisku "Jeszcze raz?", aby skonfigurować przyciski z umieszczonymi na nich losowo ośmioma parami emoji.

#### 1 Zacznij tworzyć listę emoji ze zwierzętami.

Metoda obsługi zdarzeń musi się zaczynać od ośmiu par emoji, więc napiszesz instrukcję, która je tworzy i przechowuje w strukturze nazywanej listą (dowiesz się o niej znacznie więcej w rozdziale 8.).

Wróć do pliku *MainPage.xaml.cs* i zacznij wpisywać poniższy wiersz kodu zaraz *po* wcześniej dodanych instrukcjach. *Nie* kończ go jednak średnikiem, ponieważ to jeszcze nie koniec instrukcji:

```
List<string> animalEmoji = [
```

Podczas wpisywania pojawią się okna funkcji IntelliSense, które pomogą Ci we wprowadzaniu kodu. Wpisywany tekst będzie pogrubiony, a po nim pojawi się *sugestia* wygenerowana przez IntelliCode:



Po wpisaniu otwierającego nawiasu kwadratowego ] Visual Studio dodaje pasujący nawias zamykający i umieszcza kursor myszy między tymi dwoma nawiasami:

```
List<string> animalEmoji = [] 

Kursor myszy powinien się teraz znajdować

pomiędzy nawiasami [ i ].
```

Wciśnij *Enter*, a następnie dodaj średnik na końcu. Metoda PlayAgainButton\_Clicked powinna teraz wyglądać następująco:

#### 2 Dodaj do listy parę emoji ze zwierzętami.

Tworzona instrukcja w C# nie jest jeszcze gotowa. Upewnij się, że kursor znajduje się w pustym wierszu dodanym między nawiasami. Teraz podasz **osiem par emoji ze zwierzętami**. Emoji można znaleźć w witrynie z takimi symbolami (na przykład *https://emojipedia.org/nature*) i skopiować poszczególne symbole. Istnieje też inna możliwość...

Jeśli korzystasz z systemu Windows, użyj **panelu emoji tego systemu** (wciśnij kombinację klawisz z logo Windows+kropka). Jeśli korzystasz z komputera Mac, użyj **panelu Character Viewer** (naciśnij klawisz fn; na starszych komputerach Mac wybierz kombinację *Ctrl*+**#**+spacja).

Wróć do kodu i dodaj cudzysłów ("), a następnie wklej wybrany symbol (my użyliśmy ośmiornicy), po którym następują kolejny cudzysłów, przecinek, spacja, cudzysłów, ponownie ten sam symbol, jeszcze jeden cudzysłów i przecinek. Zauważ, że Visual Studio pomaga we wprowadzaniu elementów listy, na przykład po wpisaniu cudzysłowu otwierającego dodaje cudzysłów zamykający.

Lista powinna wyglądać teraz tak:

```
List<string> animalEmoji = [
",",",",","];
```

# Jak wprowadzać emoji?

Jeśli korzystasz z systemu Windows, użyj **panelu emoji**. W tym celu wciśnij kombinację klawisz z logo systemu Windows ( )+ kropka. Użyj pola wyszukiwania, aby znaleźć określone zwierzę. Gdy natrafisz na emoji, które chcesz zastosować, kliknij je, aby wprowadzić je tak jak za pomocą wpisywania.







#### Dodaj do listy pozostałe pary emoji ze zwierzętami.

Następnie zrób to samo z siedmioma kolejnymi emoji, aby uzyskać **osiem par emoji ze zwierzętami między nawiasami**. My dodaliśmy rozdymkę, słonia, wieloryba, wielbłąda, brontozaura, kangura i jeżozwierza, ale możesz wybrać dowolne zwierzęta (lub inne emoji!).

Dodaj znak ; po zamykającym nawiasie klamrowym. Instrukcja powinna teraz wyglądać tak:



```
Ukończ metodę.
```

Więcej o pętlach dowiesz się z następnego rozdziału.

# Wprowadź resztę kodu, aby dodać losowe emoji ze zwierzętami do przycisków. Ten kod znajduje się po zamykającym nawiasie ]; na końcu wyrażenia kolekcji, ale przed nawiasem } kończącym metodę:

```
foreach (var button in AnimalButtons.Children.OfType<Button>())
{
    int index = Random.Shared.Next(animalEmoji.Count);
    string nextEmoji = animalEmoji[index];
    button.Text = nextEmoji;
    animalEmoji.RemoveAt(index);
}
```

#### To *pętla foreach*. Pobiera ona elementy kolekcji (na przykład listy emoji) i dla każdego z nich wykonuje zestaw instrukcji.

Przed uruchomieniem aplikacji przeczytaj właśnie dodany kod. Nic nie szkodzi, jeśli nie rozumiesz jeszcze wszystkiego, co się w nim dzieje. Ważną częścią nauki języka C# jest to, aby próbować zrozumieć kod, a czytanie go jest świetnym ćwiczeniem.

### Czytanie kodu C#, nawet jeśli nie wszystko jeszcze rozumiesz, to świetny sposób, aby programy zaczęły nabierać sensu.

**44 Rozdział 1.** Kup ksi k

#### 5 Upewnij się, że Twój kod jest zgodny z naszym.

Oto cały kod C# dodany do tej pory. Fragmenty wygenerowane automatycznie przez Visual Studio wyróżniamy jaśniejszym kolorem, aby łatwiej było dostrzec samodzielnie wprowadzony kod.

```
    Jeśli Twój projekt ma inną nazwę,

namespace AnimalMatchingGame;
                                                          pojawi się ona w tym wierszu.
public partial class MainPage : ContentPage
                                                              Ten wiersz sprawia, że przyciski
    public MainPage()
                                                              ze zwierzętami są niewidoczne po
                                                              pierwszym uruchomieniu aplikacji.
         InitializeComponent();
    private void PlayAgainButton Clicked(object sender, EventArgs e)
         AnimalButtons.IsVisible = true;
         PlayAgainButton.IsVisible = false;
         List<string> animalEmoji = [
                                                                                 Visual Studio automatycznie
              "Q", "Q",
              "@", "@",
                                                                                doda wcięcia w Twoim kodzie,
                                                                                 aby wyglądał jak nasz. Jeśli
              "", "",
                                       Upewnij sie, że kod zawiera
              "@", "@",
                                                                                  używasz VSCode, naciśnij
                                       dokładnie osiem par pasujących
                                                                                 Alt+Shift+F lub \mathbb{T}+Shift+F,
              "3", "3",
                                       emoji. Jest to jeden z elementów
                                                                                 by automatycznie poprawić
                                       sprawiających, że gra działa.
              "L", "L",
                                                                                      wcięcia w pliku.
              "'', "'',"
              " 📣 "
         1:
        foreach (var button in AnimalButtons.Children.OfType<Button>())
         {
             int index = Random.Shared.Next(animalEmoji.Count);
             string nextEmoji = animalEmoji[index];
                                                                             Właśnie
             button.Text = nextEmoji;
                                                                             wprowadziliśmy
                                                                             ten kod, aby
             animalEmoji.RemoveAt(index);
                                                                             dodać emoji
                                                                             do przycisków
    private void Button Clicked(object sender, EventArgs e)
                 Visual Studio wygenerowało tę pustą metodę obsługi zdarzenia Button Clicked
                    po dodaniu właściwości C1 i cked do skopiowanego i wklejonego przycisku.
                 Upewnij się, że ta metoda istnieje! Jeśli używasz VSCode, być może trzeba będzie
                            wpisać ją ręcznie, jeśli nie została dodana automatycznie.
```

# Uruchom aplikację!

Ponownie uruchom aplikację. Pierwszą rzeczą, jaką zobaczysz, będzie przycisk "Jeszcze raz?". Kliknij go, a powinno się pojawić osiem par zwierząt w losowych miejscach:



Kilka razy zatrzymaj grę i uruchom ją ponownie. Za każdym razem, gdy klikniesz przycisk "Jeszcze raz?", zwierzęta powinny zostać rozmieszczone w inny sposób.



Nieźle, gra już zaczyna dobrze wyglądać!

#### Podstawy następnej dodawanej części są już przygotowane.

Kiedy tworzysz nową grę, nie tylko piszesz kod. Ważne jest także realizowanie projektu. Wysoce skutecznym sposobem realizowania projektu jest budowanie go w małych krokach i regularne ocenianie postępów, aby mieć pewność, że wszystko zmierza w dobrym kierunku. W ten sposób będziesz mieć wiele okazji do zmiany kursu.



Jeśli korzystasz z Visual Studio,

możesz zobaczyć pasek narzędzi

aplikacji w górnej części okna:

Następnie kliknij pierwszy przycisk w panelu *Dynamiczne drzewo wizualne*, aby włączyć lub wyłączyć pasek narzędzi aplikacji.









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.

Ćwiczenia 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 błędów (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.

Mówimy poważnie – poświęć czas na wykonanie ćwiczeń z ołówkiem i papierem. Są one starannie zaprojektowane, aby podkreślić ważne koncepcje. Ich wykonywanie to najszybszy sposób na przyswojenie sobie pomysłów z tej książki.



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

### **CELNE SPOSTRZEŻENIA**

- Visual Studio jest IDE Microsoftu (zintegrowanym środowiskiem programistycznym), które upraszcza i wspomaga edycję plików z kodem C# i zarządzanie nimi.
- Aplikacje konsolowe to działające w różnych systemach aplikacje z tekstowymi danymi wejściowymi i wyjściowymi.
- .NET MAUI (czyli .NET Multi-platform App UI) to wieloplatformowy framework do tworzenia aplikacji graficznych w języku C#.
- Interfejsy użytkownika w MAUI są projektowane w XAML-u (ang. eXtensible Application Markup Language), opartym na XML-u języku znaczników, w którym wykorzystuje się znaczniki i właściwości do definiowania kontrolek w interfejsie użytkownika.

- Aplikacje MAUI składają się ze stron, na których wyświetlane są kontrolki.
- Kontrolka FlexLayout zawiera inne kontrolki i opakowuje je w taki sposób, aby były wyświetlane na stronie.
- Okno Właściwości w IDE umożliwia łatwe modyfikowanie właściwości kontrolek, na przykład tekstu lub rozmiaru czcionki.
- Kod C# składa się z instrukcji pogrupowanych w metody.
- Metoda obsługi zdarzeń jest wykonywana, gdy wystąpią określone zdarzenia, takie jak kliknięcie przycisku.
- Wykorzystujące sztuczną inteligencję mechanizmy IntelliSense i IntelliCode z Visual Studio pomagają w szybszym wprowadzaniu kodu.



Mój projekt zawiera już mnóstwo kodu! Czy nie byłoby cudownie, gdyby istniał łatwy sposób na umieszczenie go w miejscu, w którym mogę zapisać kod, udostępniać go i znaleźć go zawsze, kiedy tego potrzebuję?

#### Możesz użyć systemu Git, aby zapisać cały kod, a Visual Studio to ułatwia.

W trakcie lektury tej książki będziesz pisać dużo kodu! Czy nie byłoby wspaniale, gdyby istniało wygodne miejsce do zapisywania go, aby zawsze można było do niego wrócić?

Jesteśmy przekonani, że napiszesz kilka aplikacji, które naprawdę Ci się spodobają, dlatego zechcesz podzielić się nimi ze znajomymi, aby mogli zobaczyć opracowane przez Ciebie wspaniałe programy.

Czy masz komputer stacjonarny i laptop? Komputer w domu i w biurze? Czy nie byłoby cudownie, gdyby można było rozpocząć projekt na jednym komputerze, a następnie dokończyć go na innym?

Wyobraź sobie, że pracujesz nad projektem. Po wielu godzinach poświęconych na uzyskanie poprawnego kodu jego poziom wreszcie Cię zadowala. Następnie wprowadzasz kilka zmian i... o nie! Coś poszło zupełnie nie tak, kod nie działa, a Ty nie pamiętasz, co dokładnie zostało zmodyfikowane. Byłoby wspaniale, gdyby można było zobaczyć historię wszystkich wprowadzonych zmian, prawda?

Git daje Ci wszystkie wymienione możliwości!

### Oto kilka rzeczy, które Git może dla Ciebie zrobić

- ★ Potrafi zapisywać pliki w miejscu, do którego można uzyskać dostęp z dowolnego miejsca i w dowolnym czasie.
- ★ Umożliwia zapisywanie snapshotów, dzięki czemu można wrócić i zobaczyć, co dokładnie się zmieniło.
- ★ Umożliwia udostępnianie kodu każdemu (lub zachowanie go w tajemnicy!).
- ★ Umożliwia wspólną pracę nad projektem grupie osób. Jeśli więc uczysz się języka C# ze znajomymi, możecie wszyscy razem pracować nad kodem.
## Visual Studio ułatwia korzystanie z systemu Git

Git to zaawansowane i wszechstronne narzędzie, które pomaga w zapisywaniu i udostępnianiu kodu oraz plików z wszystkich projektów oraz zarządzanie tymi elementami. Czasami praca z tym systemem bywa skomplikowana i nieintuicyjna! Na szczęście Visual Studio ma **wbudowaną obsługę systemu Git** i zajmuje się wszystkimi skomplikowanymi aspektami. Pomaga w obsłudze systemu Git, dzięki czemu można się skoncentrować na kodzie.

Visual Studio pomaga utworzyć nowe repozytorium Git w serwisie GitHub, popularnej platformie do hostowania kodu źródłowego i współpracy.

			Git Changes	
Wypchnij do nowego zdalnego	*	Zainicjuj lokalne repozytoriur	<ul> <li>☐ Git Changes</li> <li>↓ main</li> </ul>	
GitHub     Azure DevOps		Szablon .gitignore ① Do	Finished the third part of the animal matching game	
Inne		Szablon licencji ① Bra	Commit Staged V Amend	
Tylko lokalne	0	Utwórz nowe repozytorium G Konto	V Staged Changes There are no staged changes.	- Unstage All
		Właściciel	✓ Changes – 38 iii .gitignore	+ Stage All
		Opis W	AnimalMatchingGame	
		Repozytorium prywatne	AnimalMatchingGame.csproj AnimalMatchingGame     App.xaml AnimalMatchingGame	
Funkcje związane z syste Studio ułatwiają dodawan	emem nie ko	Git w Visual	AppShell.xaml AnimalMatchingGame     AppShell.xaml AnimalMatchingGame     AppShell.xaml.cs AnimalMatchingGame     MainPage yaml AnimalMatchingGame	

Zalecamy utworzenie konta w serwisie GitHub i używanie go do zapisywania kodu każdego projektu z tej książki. Ułatwi to powrót w dowolnym momencie do wcześniejszych projektów!

W naszym bezpłatnym pliku PDF z książką Head First C# Guide to Git znajdziesz prosty przewodnik krok po kroku dotyczący zapisywania kodu w systemie Git za pomocą Visual Studio. Plik ten możesz pobrać ze strony https://github.com/head-first-csharp/fifth-edition.

Omawiamy wszystko, czego potrzebujesz do używania Visual Studio do zapisywania i udostępniania projektów. Jednak system Git daje znacznie większe możliwości, zwłaszcza jeśli pracujesz w dużym zespole! Jeżeli system Git Cię zainteresował i chcesz się lepiej z nim zapoznać, przeczytaj książkę *Head First Git* Raju Gandhiego.



#### jak działają kliknięcia myszą?



## Dodaj kod C# do obsługi kliknięć myszą

Masz przyciski z losowymi emoji ze zwierzętami. Teraz musisz sprawić, aby reagowały one na kliknięcia gracza. Program ma działać tak:



Gracz klika pierwszy przycisk.

Gracz klika przyciski parami. Po kliknięciu pierwszego przycisku gra zapisuje, jakie zwierzę się na nim znajduje. Kliknięty przez gracza przycisk zmienia kolor, dzięki czemu widać, które zwierzę zostało wybrane.



Gracz klika drugi przycisk.

Gra sprawdza zwierzę z drugiego przycisku i porównuje je z emoji zapisanym po pierwszym kliknięciu.

> Gra powtarza cały proces do momentu dopasowania wszystkich ośmiu par zwierząt.



Gra sprawdza dopasowanie.

Jeśli zwierzęta *pasują* do siebie, gra sprawdza wszystkie pozycje na liście losowo uporządkowanych emoji. Gdy znajdzie na liście emoji odpowiadające znalezionej przez użytkownika parze zwierząt, zastępuje je pustymi polami.

Jeżeli zwierzęta *nie pasują* do siebie, gra nic nie robi.

W *obu scenariuszach* gra resetuje zapisane znalezione zwierzę, aby następne kliknięcie rozpoczynało cały \_\_\_\_\_ proces od nowa.

## Zaostrz ołówek



Po dodaniu obsługi zdarzeń **Clicked** do przycisku ze zwierzęciem Visual Studio **automatycznie dodało metodę o nazwie Button\_Clicked** do pliku *MainPage.xaml.cs.* Oto kod, który znajdzie się w tej metodzie. Zanim dodasz ten fragment do aplikacji, przeczytaj go i spróbuj się domyślić, co robi.

Zadaliśmy kilka pytań dotyczących działania kodu. Spróbuj zapisać odpowiedzi. *Nic nie szkodzi, jeśli nie będą one w pełni poprawne!* Chodzi o to, aby mózg zaczął traktować kod C# jako coś, co można przeczytać i zrozumieć.

```
Button lastClicked;
                                                                      1. Co robi matchesFound?
bool findingMatch = false:
int matchesFound; 🗲
private void Button Clicked(object sender, EventArgs e)
{
  if (sender is Button buttonClicked)
    if (!string.IsNullOrWhiteSpace(buttonClicked.Text) && (findingMatch == false))
                                                          2. Co robia te trzy wiersze kodu?
      buttonClicked.BackgroundColor = Colors.Red;
      lastClicked = buttonClicked;
      findingMatch = true;
    }
    else
      if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text))
       {
                                         To cudzysłów, po
                                         którym następuje
         matchesFound++:
                                                                  3. Co robi ten blok kodu?
                                         spacja i kolejny
         lastClicked.Text = " ";
                                         cudzysłów.
         buttonClicked.Text = " ";
      }
      lastClicked.BackgroundColor = Colors.LightBlue;
      buttonClicked.BackgroundColor = Colors.LightBlue;
      findingMatch = false;
    }
  }
  if (matchesFound == 8)
    matchesFound = 0;
    AnimalButtons.IsVisible = false;
    PlayAgainButton.IsVisible = true;
  }
}
4. Co robi sześć ostatnich wierszy metody, od if (matchesFound == 8) do końca?
```

#### ten kod jest uruchamiany w reakcji na kliknięcie

```
Zaostrz ołówek
      Rozwiązanie
                          Zadaliśmy kilka pytań dotyczących działania kodu. Spróbuj zapisać odpowiedzi.
                          Nic nie szkodzi, jeśli nie beda one w pełni poprawne! Chodzi o to, aby móża
                          zaczał traktować kod C# jako coś, co można przeczytać i zrozumieć.
Button lastClicked:
                                                                       1. Co robi matchesFound?
bool findingMatch = false;
                                                                       Rejestruje liczbę par zwierząt
int matchesFound; -
                                                                       znalezionych przez gracza,
private void Button Clicked(object sender, EventArgs e)
                                                                       dzięki czemu grę można zakończyć
                                                                      po wskazaniu wszystkich 8 par.
  if (sender is Button buttonClicked)
    if (!string.IsNullOrWhiteSpace(buttonClicked.Text) && (findingMatch == false))
       buttonClicked.BackgroundColor = Colors.Red;
                                                          2. Co robia te trzy wiersze kodu?
       lastClicked = buttonClicked;
                                                          Są one uruchamiane, gdy gracz kliknie pierwszy
       findingMatch = true;
                                                          przycisk potencjalnej pary. Ten kod zapisuje
    }
    else
                                                          przycisk i zmienia jego kolor na czerwony.
    {
       if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text))
       {
         matchesFound++;
                                                                  3. Co robi ten blok kodu?
         lastClicked.Text = " ";
         buttonClicked.Text = " ";
                                                                  Jest on uruchamiany, gdy gracz kliknie
       }
                                                                  drugi przycisk w parze. Jeśli zwierzęta
       lastClicked.BackgroundColor = Colors.LightBlue;
                                                                  pasuja do siebie, kod dodaje jeden do
       buttonClicked.BackgroundColor = Colors.LightBlue;
       findingMatch = false;
                                                                  matchesFound i ukrywa zwierzęta na
                                                                  obu przyciskach. Resetuje również kolor
                                                                  pierwszego przycisku i przygotowuje grę
  if (matchesFound == 8)
                                                                  tak, aby gracz mógł ponownie kliknąć
    matchesFound = 0;
                                                                  pierwszy przycisk w parze.
    AnimalButtons.IsVisible = false;
    PlavAgainButton.IsVisible = true:
  }
}
4. Co robi sześć ostatnich wierszy metody, od if (matchesFound == 8) do końca?
Jeśli matchesFound wynosi 8, gracz znalazł wszystkie 8 par zwierząt. Wtedy ten kod resetuje grę:
 ustawia matchesFound z powrotem na 0, ukrywa przyciski ze zwierzętami i wyświetla przycisk
 "Jeszcze raz?", aby gracz mógł go kliknąć w celu rozpoczęcia nowej gry.
```

#### Wprowadź kod obsługi zdarzeń

Czy udało Ci się wykonać ćwiczenie "Zaostrz ołówek"? Jeśli nie, poświęć kilka minut i zrób to. Być może nie rozumiesz jeszcze w pełni kodu w metodzie obsługi zdarzeń Button\_Clicked, ale prawdopodobnie masz przynajmniej podstawowe pojęcie o tym, co się w niej dzieje. Co ważniejsze, ćwiczenie to daje możliwość przyjrzenia się kodowi na tyle uważnie, że powinien wyglądać znajomo.

To poczucie znajomości ułatwi **korzystanie z IDE do wpisywania kodu metody**. Zatrzymaj aplikację, jeśli jest uruchomiona (w tym celu zamknij okno lub wybierz opcję *Zatrzymaj debugowanie* (*Shift+F5*) z menu *Debuguj* lub *Uruchom*), a następnie przejdź do pliku *MainPage.xaml.cs*, znajdź metodę obsługi zdarzeń Button\_Clicked, którą dodało Visual Studio, i kliknij wiersz między otwierającym ({) a zamykającym (}) nawiasem klamrowym.

Teraz zacznij **wpisywać wiersz po wierszu kod z rozwiązania ćwiczenia** "Zaostrz ołówek". Jeśli nie masz doświadczenia w pisaniu kodu za pomocą IDE, takiego jak Visual Studio lub VSCode, mogą dziwić Cię wyskakujące sugestie funkcji IntelliSense i IntelliCode. Użyj ich, jeśli potrafisz. Im bardziej się do nich przyzwyczaisz, tym szybsze i łatwiejsze będzie pisanie kodu w dalszej części książki.

Musisz zachować staranność podczas wprowadzania kodu, ponieważ jeśli otwierające nawiasy zwykłe lub klamrowe nie mają zamykających odpowiedników lub jeżeli pominiesz średnik na końcu instrukcji, kod się nie skompiluje. Na szczęście Visual Studio ma wiele funkcji pomagających pisać kod, który się kompiluje:

- ★ Po wpisaniu if automatycznie dodaje nawiasy otwierający i zamykający, (), aby ich przypadkowo nie pominąć.
- ★ Umieszczenie kursora przed otwierającym nawiasem zwykłym lub klamrowym spowoduje wyróżnienie powiązanego nawiasu zamykającego, dzięki czemu można łatwo go zobaczyć.
- ★ W wielu przypadkach, gdy wprowadzisz niepoprawny kod, na przykład wpiszesz matchesFnd zamiast matchesFound, Visual Studio wskaże błąd przez umieszczenie pod nim czerwonej falistej linii.



System operacyjny, taki jak Windows, macOS, Android lub iOS, nie potrafi uruchamiać kodu C#. Dlatego Visual Studio musi **skompilować** kod, czyli przekształcić go w plik **binarny** (który może być uruchomiony przez system operacyjny). Przeprowadźmy eksperyment i **wprowadźmy w kodzie błąd**.

Przejdź do pierwszego wiersza kodu w metodzie Button\_Clicked. Naciśnij Enter dwa razy, a następnie dodaj w osobnym wierszu litery xyz.

Przyjrzyj się dolnej części edytora kodu. Zobaczysz ikonę, która wygląda tak: 🙆 2 lub tak: 🙆 2 lub tak: która wygląda tak: 2 lub t

Kliknij ikonę (lub wybierz opcję Lista błędów z menu Widok), aby otworzyć okno Lista błędów. Zobaczysz w nim dwa błędy (jeśli używasz komputera Mac, okno nazywa się Errors zamiast Lista błędów i wygląda nieco inaczej, ale wyświetla te same informacje):

Lista blędów 👻 🕈 🖓								
Całe rozwiązanie	<ul> <li>2 Błędy</li> <li>0 Ostrzeżenia</li> <li>0 Komunikaty</li> </ul>	•		Przeszukaj listę błędów	<del>،</del> م			
"I Kod	Opis	Projekt	Plik	W Stan pominięcia				
S CS0103	Nazwa "xyz" nie istnieje w bieżącym kontekście	AnimalMatchingGame (net8.0-a.	. MainPage.xaml.cs	40				
S CS1002	Oczekiwano średnika (;)	AnimalMatchingGame (net8.0-a.	. MainPage.xaml.cs	40				

Visual Studio wyświetla te błędy, ponieważ xyz nie jest poprawnym kodem C#, a błędy uniemożliwiają kompilację aplikacji. Kod z takimi błędami nie będzie działał, usuń więc dodany wiersz xyz i ponownie skompiluj aplikację.

Jeśli w kodzie nie ma innych usterek, lista błędów powinna być pusta, a w dolnej części okna Visual Studio pojawi się ikona: 💿 Nie znaleziono żadnych problemów lub: 💽 Build successful. Informuje ona o udanej kompilacji aplikacji.

## Uruchom aplikację i znajdź wszystkie pary

Spróbuj uruchomić aplikację. Jeśli cały kod został wprowadzony poprawnie, aplikacja powinna się uruchomić i wyświetlić przycisk "Jeszcze raz?". Kliknij go, aby zobaczyć losową listę zwierząt. Następnie klikaj każdą parę zwierząt po kolei. Będą one znikać po ich kliknięciu. Po wybraniu ostatniej pary zwierząt przyciski znikną i ponownie pojawi się przycisk "Jeszcze raz?".







Jeśli gra nie działa tak, jak powinna, a nie widzisz błędów, cofnij się i porównaj wprowadzony kod z wersją z książki. Bardzo łatwo jest przeoczyć literówkę. Wyszukiwanie takich błędów nie jest marnowaniem czasu, ponieważ wykrywanie usterek w kodzie to bardzo cenna umiejętność programistyczna, nad którą warto popracować.

Spróbuj poeksperymentować z aplikacją. Klikaj niedopasowane pary. Kliknij w oknie, ale poza przyciskami. Kliknij etykietę "Czas". Kliknij pusty przycisk. Czy aplikacja działa?

## No pięknie, w kodzie znajduje się błąd

Jeśli cały kod został poprawnie przepisany, zapewne dostrzeżesz problem. Uruchom aplikację, kliknij przycisk "Jeszcze raz?", aby wyświetlić losowe zwierzęta, a następnie kliknij parę, by zwierzęta zniknęły z przycisków. Teraz **kliknij jeden z pustych przycisków, a następnie drugi. Powtórz to siedem razy.** Chwileczkę, co się stało? Czy przyciski ze zwierzętami zniknęły i pojawił się przycisk "Jeszcze raz?", tak jak po wygraniu gry? To nie powinno się zdarzyć! W grze kryje się błąd.

#### Nie martw się, ten błąd to nie Twoja wina!

Celowo umieściliśmy ten błąd w kodzie. W trakcie lektury tej książki będziesz pisać dużo kodu. Każdy rozdział zawiera kilka projektów i w każdym z nich mogą wystąpić błędy. Ich wykrywanie i naprawianie to zwyczajne aspekty pisania kodu, a także bardzo cenna umiejętność, którą warto ćwiczyć.

## Kiedy znajdziesz błąd, musisz wykryć jego źródło

Każdy błąd jest inny. Kod może stać się niepoprawny na wiele różnych sposobów. Jest jednak jedna wspólna cecha wszystkich błędów: *każdy z nich jest spowodowany problemem w kodzie*. Kiedy więc pojawia się błąd, Twoim zadaniem jest ustalenie, co go powoduje, ponieważ nie możesz naprawić problemu, dopóki nie wiesz, z czego wynika.

Jeśli kiedykolwiek zdarzyło Ci się czytać powieść kryminalną lub oglądać serial detektywistyczny, wiesz, że aby rozwiązać zagadkę, musisz **znaleźć winnego**. Zróbmy to więc teraz. Nadszedł czas, aby założyć czapkę Sherlocka Holmesa, chwycić szkło powiększające i **dowiedzieć się, co powoduje błąd**.

Każdy błąd jest spowodowany przez problem w kodzie, więc pierwszym krokiem do wyeliminowania problemu jest ustalenie jego przyczyny.



Wyszukiwanie i naprawianie błędów składają się w jednej części z pisania na klawiaturze, a w dziewięciu z myślenia. Dają też pewność, że staniesz się lepszym programistą. Do tego właśnie służą ćwiczenia "Wydedukuj to".

#### Przypadek nieoczekiwanego dopasowania

#### Prawdopodobnie znasz słowo "błąd".

Być może nawet zdarzyło Ci się powiedzieć w przeszłości znajomym: "Ta gra jest pełna błędów, ma tak wiele usterek". Każdy błąd ma swoje wytłumaczenie, a wszystko w programie dzieje się z jakiegoś powodu. Jednak nie każdy błąd jest łatwy do wyśledzenia. Dlatego w całej książce umieszczamy wskazówki dotyczące wykrywania błędów. Zaczynamy od tego ćwiczenia "Wydedukuj to".

#### Z każdym błędem wiąże się winny.

Błędy są czymś dziwnym. Pojawiają się, gdy kod robi coś, czego się nie spodziewasz.

Ale błędy są również czymś normalnym. Każdy programista przeznacza czas na znajdowanie i naprawianie usterek. To normalny aspekt pisania kodu. Możesz napisać kod, który nie działa zgodnie z Twoimi oczekiwaniami. Wtedy pierwszą rzeczą, którą należy zrobić, **jest ustalenie źródła błędu**.

## Pierwszym krokiem do znalezienia błędu jest zastanowienie się, co mogło go spowodować.

Sherlock Holmes powiedział kiedyś: "Zbrodnia jest pospolita. Logika jest rzadka. Dlatego też powinieneś kłaść nacisk raczej na przejawy logiki niż na samą zbrodnię". To świetna rada, gdy chcesz ustalić, co spowodowało błąd. Nie frustruj się tym, że aplikacja nie robi tego, co chcesz (to byłoby rozmyślanie o zbrodni!). Zamiast tego zastanów się nad logiką sytuacji. Spójrzmy więc na kod i ustalmy, co się dzieje.

#### Przeczytaj uważnie kod i poszukaj wskazówek.

Wiadomo, że cały kod do obsługi kliknięć myszą znajduje się we właśnie dodanej metodzie obsługi zdarzeń Button\_Clicked. Wróćmy więc do kodu i sprawdźmy, czy uda się znaleźć wskazówki dotyczące tego, co poszło nie tak.

Na szczęście **masz za sobą ćwiczenie** "Zaostrz ołówek". Przyjrzałeś się dokładnie kodowi w metodzie obsługi zdarzeń Button\_Clicked, aby go zrozumieć. (Jeśli jeszcze nie wykonałeś tego ćwiczenia, wróć i zrób to teraz!)

Na podstawie odkryć z ćwiczenia "Zaostrz ołówek" wiemy już o kodzie kilka rzeczy:

- Metoda obsługi zdarzeń używa zmiennej matchesFound do śledzenia liczby par zwierząt znalezionych przez gracza, dzięki czemu gra może się zakończyć po wskazaniu wszystkich ośmiu par.
- Część metody obsługi zdarzeń sprawdza, czy zwierzęta na dwóch przyciskach klikniętych przez gracza pasują do siebie. Jeśli tak jest, dodaje jeden do matchesFound i ukrywa oba przyciski.
- Jeśli matches Found jest równe 8, oznacza to, że gracz znalazł wszystkie 8 par zwierząt. Na końcu metody obsługi zdarzeń znajduje się kod sprawdzający, czy matches Found wynosi 8. Gdy tak jest, kod resetuje grę.

## Są to ważne wskazówki, które pomogą wykryć i naprawić błąd. Zanim przejdziesz dalej, czy potrafisz wydedukować, co jest powodem przedwczesnego zakończenia gry, gdy ciągle klikasz pusty przycisk?

zbudui coś wspaniałego… szybko!

Wydedukuj to





#### Dlaczego wystąpił błąd?

Zastanów się przez chwilę nad tymi trzema wskazówkami. Oto co wiadomo:

- Gra używa zmiennej matchesFound do śledzenia liczby par zwierząt znalezionych przez gracza.
- Jeśli gracz kliknie parę, gra zwiększy matchesFound o 1 i usunie zawartość wybranych przez gracza przycisków.
- Gdy matchesFound osiągnie 8, gra zostaje zresetowana.

Co więc mówią nam te wskazówki? Jest jeden wniosek, który możemy z nich wyciągnąć:

W jakiś sposób wartość matches Found jest zwiększana o 1, gdy gracz kliknie przycisk, który jest już pusty.

Oznacza to, że wiadomo, od czego zacząć – od kodu, który zwiększa wartość matchesFound o 1.

#### Wróć na miejsce zbrodni

Oto fragment kodu, który zwiększa wartość matchesFound. Konkretny wiersz, który to robi, jest pogrubiony:

```
if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text))
```

```
{
```

}

matchesFound++;
lastClicked.Text = " ";
buttonClicked.Text = " ";

W tej instrukcji używany jest operator ++, aby zwiększyć
wartość matchesFound o 1. Operator ++ oraz inne
operatory omawiamy w następnym rozdziale.

Pierwszy wiersz kodu we fragmencie, który właśnie pokazaliśmy, to **instrukcja if**. Sprawdza ona, czy podany warunek jest spełniony, a jeśli tak jest, to wykonuje podane polecenia. Tutaj jeśli gracz kliknął inny przycisk niż pierwszy w parze (to właśnie sprawdza wyrażenie "buttonClicked != lastClicked") i jeśli zwierzęta na obu przyciskach pasują do siebie ("buttonClicked.Text == lastClicked.Text"), kod zwiększa wartość matchesFound o 1 i ukrywa zawartość tych przycisków.

To właśnie tutaj pojawia się problem. To oznacza, że to w tym miejscu możemy naprawić błąd. Trzeba tylko znaleźć sposób, by zmienna matchesFound nie była zwiększana o 1, **jeśli gracz kliknął przycisk, który jest już pusty**.

#### Znaleźliśmy winowajcę, więc teraz możemy naprawić błąd.

Umieść kursor między dwoma ostatnimi nawiasami zamykającymi )) w instrukcji i f i naciśnij *Enter*, aby dodać wiersz. Następnie wprowadź następujący kod: && (!String.IsNullOrWhiteSpace(buttonClicked.Text)).

Kod powinien wyglądać teraz tak:

```
if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text)
    && (!String.IsNullOrWhiteSpace(buttonClicked.Text)))
{
    matchesFound++;
    lastClicked.Text = " ";
    buttonClicked.Text = " ";
}
Po modyfikacji instrukcji if ponownie uruchom aplikację. Błąd powinien zostać naprawiony.
```

ZNAJDUJESZ SIĘ TUTAJ



## Dodaj zegar, aby ukończyć grę

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

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



Zegar "cyka" 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. spraw, aby zegar zaczał działać

## Dodaj zegar do kodu gru

W tej ostatniej cześci projektu dodasz do gry licznik czasu, aby stała się ciekawsza. Bedzie on śledził upływający czas (mierzony w dziesietnych cześciach sekundy) od momentu klikniecia przycisku "Jeszcze raz?" do znalezienia ostatniego dopasowania.



#### (1)

#### Dodaj wiersz kodu na końcu metody obsługi zdarzenia PlayAgainButton Clicked, aby uruchamiać licznik czasu.

Przejdź na sam koniec metody obsługi zdarzeń PlayAgainButton Clicked. Znajduja sie tam dwa zamykające nawiasy klamrowe } w oddzielnych wierszach. Dodaj trzy wiersze miedzy tymi nawiasami, a następnie wpisz w tym miejscu poniższy wiersz kodu:

```
foreach (var button in AnimalButtons.Children.OfType<Button>())
    int index = Random.Shared.Next(animalEmoji.Count);
    string nextEmoji = animalEmoji[index];
    button.Text = nextEmoji:
    animalEmoji.RemoveAt(index):
}
Dispatcher.StartTimer(TimeSpan.FromSeconds(.1), TimerTick);
```

Dodany wiersz kodu powoduje, że aplikacja **uruchamia zegar**, który co 0,1 sekundy wykonuje metodę o nazwie TimerTick.

}

#### (2) Sprawdź błąd i kliknij TimerTick we właśnie dodanym kodzie.

Właśnie dodaliśmy wiersz kodu, aby uruchamiać zegar, który "tyka" co 0,1 sekundy. Za każdym razem wywołuje on metodę o nazwie TimerTick. Ale chwileczkę, w kodzie C# nie ma takiej metody. Jeśli spróbujesz skompilować kod, zobaczysz błąd w oknie listy błędów:

CS0103 Nazwa "TimerTick" nie istnieje w bieżącym kontekście

Ponadto pod nazwą TimerTick w dodanym wierszu kodu pojawi się czerwona falista linia. Kliknij TimerTick w kodzie C#. Visual Studio wyświetli wtedy na lewym marginesie ikonę w kształcie żarówki lub śrubokręta.



#### 3 Użyj Visual Studio, aby wygenerować nową metodę TimerTick.

Dodany kod zawiera błąd, ponieważ używana jest w nim metoda o nazwie TimerTick, która nie istnieje. Po kliknięciu tej nazwy na lewym marginesie pojawi się ikona żarówki lub śrubokręta. Jeśli najedziesz na nią kursorem, zobaczysz komunikat o błędzie i ikonę bezpośrednio pod spodem:



Kliknięcie ikony powoduje wyświetlenie **menu szybkich akcji**, które zawiera kilka sugerowanych możliwych poprawek błędu. Aby wyświetlić to menu, możesz także kliknąć nazwę TimerTick i wcisnąć kombinację *Alt+Enter* lub *Ctrl+*. w systemie Windows albo **#**+. na komputerze Mac (czyli *Control* lub **#**+kropka):

), TimerTick);		Platforms     Resources     App.xaml		
Generuj metodę "TimerTick"	*	SCS0103 Nazwa "TimerTick" nie istnieje w bieżącym kontekście		
Generuj właściwość "TimerTick" Generuj pole "TimerTick" Generuj pole tylko do odczytu "TimerTick" Generuj lokalny element "TimerTick" Generuj parametr "TimerTick"				
		<pre>+private bool TimerTick() +{ + throw new NotImplementedException(); +}</pre>		

Pierwszą opcją w menu szybkich akcji powinno być *Generuj metodę "TimerTick"*. Jeśli ją wybierzesz, zobaczysz podgląd kodu po prawej stronie. **Wybierz tę opcję.** 

Visual Studio **wygeneruje za Ciebie metodę TimerTick**. Zajrzyj do kodu C# w pliku *MainPage.xaml.cs* i znajdź metodę TimerTick dodaną przez Visual Studio:

```
private bool TimerTick()
{
    throw new NotImplementedException();
}
```

Gdy w kodzie C# występują błędy, Visual Studio czasami udostępnia sugestie dotyczące możliwych poprawek, które pozwalają wygenerować kod w celu wyeliminowania problemu.

## Dokończ kod gry

W tej ostatniej części projektu dodasz do gry licznik czasu, aby stała się ciekawsza. Będzie on śledził upływający czas (mierzony w dziesiętnych częściach sekundy) od momentu kliknięcia przycisku "Jeszcze raz?" do znalezienia ostatniego dopasowania.



## Dodaj pole do przechowywania czasu, który upłynął

Znajdź pierwszy wiersz metody TimerTick, którą właśnie wygenerowałeś. Umieść kursor myszy na początku wiersza, a następnie naciśnij dwukrotnie klawisz *Enter*, aby dodać nad metodą dwa wiersze.

Dodaj ten wiersz kodu tuż nad dodaną przed chwilą metodą TimerTick:

## Dokończ metodę TimerTick

Teraz masz wszystko, czego potrzebujesz, aby ukończyć metodę TimerTick. Oto jej kod:

```
private bool TimerTick()
                                               W tej instrukcji umieściliśmy dodatkowy
{
                                               znak podziału wiersza, aby kod zmieścił
    if (!this.IsLoaded) return false;
                                               się na stronie w książce, ale jeśli chcesz,
                                               możesz wpisać wszystko w jednym wierszu.
                                               Upewnij się, że nawiasy pasują do siebie.
    tenthsOfSecondsElapsed++:
    TimeElapsed.Text = "Czas: " +
         (tenthsOfSecondsElapsed / 10F).ToString("0,0 s");
    if (PlayAgainButton.IsVisible)
    {
         tenthsOfSecondsElapsed = 0;
                                                 AnimalMatchingGame
                                                Gra w dopasowywanie zwierząt
         return false:
    }
                                                  Czas: 5,2 s
    return true;
}
```

Uruchom grę. Teraz licznik czasu działa!





## Gra będzie jeszcze lepsza, gdy...

Twoja gra jest całkiem niezła. Dobra robota! 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.



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.

Gratulacje. Udało Ci się stworzyć grę, ale zrobiłeś coś więcej! Poświęciłeś czas, aby dokładnie zrozumieć, jak ona działa, a to bardzo ważny krok w oswojeniu się z językiem C#.



## **CELNE SPOSTRZEŻENIA**

- Metoda obsługi zdarzeń to metoda, którą aplikacja wywołuje po zajściu określonego zdarzenia (takiego jak kliknięcie myszą).
- Visual Studio ułatwia dodawanie metod obsługi zdarzeń i zarządzanie nimi.
- Okno Błędy w IDE wyświetla błędy uniemożliwiające kompilację kodu.
- Zegary w kółko wykonują metodę w określonych odstępach czasu.
- Pętla **foreach** iteracyjnie przetwarza elementy z kolekcji.

#### Czy dodałeś kod do repozytorium Git?

Jeśli tak, to jest to świetny moment na zatwierdzenie wszystkich zmian i przesłanie ich do repozytorium!

A jeśli jeszcze tego nie zrobiłeś, poświęć kilka minut i zapoznaj się z naszym bezpłatnym plikiem PDF z przewodnikiem *Head First C# Guide to Git*. Zawiera on instrukcje krok po kroku dotyczące bezpiecznego przechowywania kodu w Git.

Pobierz go już dziś z naszej strony z serwisu GitHub: https://github.com/head-first-csharp/fifth-edition.

- Gdy kod zawiera błąd, pierwszą rzeczą, jaką należy zrobić, jest ustalenie jego przyczyny.
- Błędy są czymś normalnym, a ich wykrywanie jest ważną umiejętnością programistyczną, którą będziesz rozwijać w tej książce.
- 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.







#### A

akcesory właściwości, 300 albedo, 405, 414 alokacja pamięci, 201 zasobów, 637 aplikacje .NET MAUI, 17, 22, 25, 49, 90, 150, 246, 250, 275, 528 biznesowe, 370 desktopowe, 18 dodawanie klasy, 136 graficzne, 25 internetowe, 17 konsolowe, 49, 66, 273, 352, 395, 420, 504, 510, 522, 563, 573, 665,670 mobilne. 18 zapewnienie dostępności, 95, 150, 255, 321 argumenty, 206 nazwane, 708 asercje, 591, 593, 619 automatyczna aktualizacja kontrolek, 459 konwersja, 202 automatyczne

implementowanie właściwości, 302, 323 sprawdzanie równości, 717 wykrywanie wartości null, 714

#### B

bajt, 199, 660, 663, 664 bezpieczeństwo danych, 298 biblioteka, 758 do rejestrowania dzienników, 759 biblioteki DLL, 759 Blazor, 17, 18 blok catch, 756 else if. 233 finally, 744, 761 try-catch, 742-746, 761 bład kompilacji, 55, 399 CS0029, 205 CS0103. 60. 72 CS0144, 396 CS0266, 202 CS0500, 397 CS0535, 421 CS1503, 206 CS7036, 305, 306 CS8618, 710 CS8852.718 CSO117, 477 błedv składniowe, 41, 196 wyszukiwanie i naprawianie, 212, 216, 280-282 zapobieganie, 283

#### ς

C#, 2 Dev Kit, 13 instalowanie rozszerzeń, 13 chatbot, 86 błędne odpowiedzi, 87 ChatGPT, 87, 186 Copilot, 87, 186 Gemini, 87, 186

generowanie dokumentacii XML. 219 kodu. 186 komentarzy, 219, 716 testów jednostkowych, 597 informacie na temat podziału zadań, 367 zasady DRY, 367 inżynieria podpowiedzi, 187, 471 prośba o przyjęcie osobowości, 471, 520 o refaktoryzację kodu, 310 pytanie o interfejsy, 471 sposoby korzystania, 86 sprawdzanie równości wartości, 716 wyszukiwanie informacji, 716, 762 ciało metody. 73 CIL. Common Intermediate Language, 229 CLR, Common Language Runtime, 225, 229, 256, 690, 693 czarna skrzvnka, 293 czytnik ekranu, 94, 95, 321

#### D

dane. 189 tekstowe i binarne, 666 weiściowe, 130 wyjściowe, 8, 13, 130 debuger lista użytecznych poleceń. 108 odtworzenie błedu, 107, 110 opcja Przekrocz nad, 74, 93 pasek narzędzi, 108 punkt przerwania, 74, 92, 109, 216 sprawdzanie działania klas, 365 konstruktora, 309 właściwości, 301 testu jednostkowego, 593 testowanie instrukcji yield return, 615 przesłaniania, 350 wartości Time.deltaTime, 263 zmiennych, 74

debuger ustawienia punktu przerwania, 262 wvrażenia kontrolne, 74 wvwołanie Random.value, 407 deklaracja listy generycznej, 490 metody, 604, 706 słownika, 512 wartości typu anonimowego, 578 zmiennei, 68, 70, 81 zmiennej tablicowej, 256 deserializacja, 644, 645, 652 destruktor, Patrz finalizator diagram klas, 136, 165, 173, 188 DLL, dynamic link library, 759 dokumentacja XML, 172, 188, 219 dostęp do metod, 286 dostępność aplikacji, 95, 150, 255, 321 DRY, don't repeat yourself, 366, 402, 472 dynamika gry, 368 dyrektywa using, 129, 138, 155, 638, 643 dziedziczenie, 325, 330, 341, 343, 352.355 interfeisów, 448 dzienniki. 758 rejestrowanie, 760, 761

### E

edytor kodu, 10, 14 XAML, 91 Unity, 112, 114 Eksplorator rozwiązań, 27 emergencja, 368, 391, 402 estetyka gry, 314, 387 etykieta, *Patrz* kontrolka Label

#### F

FIFO, first in, first out, 517 finalizator, 692, 696, 697, 712 uruchamianie, 693 używanie do serializacji, 695 format JSON, 650–652 Unicode, 655–658, 672 UTF-16, 659, 672 UTF-8, 656–659, 663, 672 formatowanie złożone, 643 funkcja anonimowa, 602

#### G

generator liczb losowych, 141, 164, 244, 254, 407 generowanie metody, 61, 72 generyczne kolekcje, 490, 516 getter, 300, 309, 316, 323 Git, 50, 51, 64 gra AnimalMatchingGame 18-62, 23 Go Fish, 244 **RPG**, 190 kalkulator obrażeń, 275 karty postaci, 191 punkty umiejetności, 208, 210 gry dostepność, 654 dynamika, 368, 387, 402 estetyka, 314, 387 mechanika, 84, 190, 244, 314, 368, 387 model MDA, 389 planszowe, 84 projektowanie w Unity, 112, 404 sprzeżenie zwrotne, 388 stołowe, 244 testy, 586

#### Η

hermetyzacja, 271, 283, 285, 291– 293, 296–298, 323, 710 hierarchia klas, 331, 336, 341

#### I

IDE, integrated development environment, 3, 49 implementowanie interfejsu, 453 indekser, 616 inicializator kolekcji, 515, 523 obiektu, 180, 188, 322 inicjowanie pól, 304, 313 właściwości, 305, 313, 322 instalowanie .NET MAUI, 17 rozszerzeń dla VSCode, 13 Unity, 113 Visual Studio, 4 VSCode, 12 XCode, 17

instancja klasy, 160, 165, 188 instrukcja, Patrz także słowo kluczowe Console.ReadLine, 242 Debug.WriteLine, 281, 282 if, 58, 78, 85, 233 if-else. 78 int.TrvParse, 243 is. 435 return. 85, 130, 131, 141 switch, 327, 328, 341, 346, 489 throw, 73 using, 129, 138, 155, 638, 643 vield return, 615-619 instrukcje, 66, 68, 81 najwyższego poziomu, 134, 135, 156.213 warunkowe, 68 IntelliCode, 3, 39 IntelliSense, 39 interfeis ICollection<T>, 492 IComparable<T>, 499, 508, 715 IComparer<T>, 501 IDisposable, 637, 643, 753 IEnumerable<T>, 509, 516, 523, 558. 571. 617 IEnumerator, 614 IEquatable, 716 IEquatable<string>,715 IGrouping, 575 IList<T>, 491 INotifyPropertyChanged, 462, 467, 468 IReadOnlvDictionary, 563, 571 interfejsy, 418-421, 424, 453 definiowanie zadań obiektu, 430 dodanie kodu. 454 domvślne implementacie, 454, 456 dostęp do składowych, 446 dziedziczenie, 448 implementowanie, 421, 453 metody, 433 określanie zadań, 434 referencje, 426 rzutowanie w dół, 446, 453 rzutowanie w górę, 446, 453 składowe, 421, 453 składowe statyczne, 455 stosowanie, 447, 452 tworzenie, 420 tworzenie zmiennej referencyjnej, 426

użytkownika, 26, 110, 552 mechanika, 88 tworzenie, 372, 544 w MAUI, 49 wiązanie danych, 459 właściwości, 433 wyświetlanie w oknie IntelliSense, 458 interpolacja łańcuchów znaków, 93, 296 inżynieria podpowiedzi, 187, 471 iteracja, 79

#### J

język C#, 1 XAML, 21, 25

#### Κ

kamera, 116, 410, 675 przesuwanie, 124 klasa, 128 Assert, 591 BinaryReader, 662 BinaryWriter, 661 Convert, 199, 207, 245 Debug, 283 Dictionary<TKey, TValue>, 512, 523 Directory, 634, 636, 643 metody, 634 Enumerable metody, 613, 619 File, 634, 636, 643 metody, 634 FileInfo, 634, 643 FileStream, 624, 643, 667 GC, 691, 712 metody, 691 JsonSerializer, 648, 649, 652, 659 List<T>, 483, 492, 495 MemoryStream, 639, 643 ObservableCollection<T>, 525 Program, 134, 135 Queue<T>, 517, 523 Random, 164, 245, 407, 414 Stack<T>, 518, 523 Stopwatch, 692 Stream, 623 StreamReader, 629, 643-667

StreamWriter, 625–628, 643 metody, 625, 626, 633 String, 724 StringReader, 633 System.Console, 296 System.Exception, 737, 761 System.String, 715 klasv. 128 abstrakcvine, 394, 396, 399, 402, 424 bazowe, 330-333, 336, 341, 376 dodawanie do aplikacii, 136 generyczne, 490 implementacja interfejsów, 418, 421.453 jako tvpv referencyjne, 701 metody, 128, 165, 300 opisowe nazwy metod, 177 pochodne, Patrz podklasy pola, 128, 165, 300 ponowne wykorzystanie, 154 rozszerzanie, 338, 723 składowe, 159, 165, 188 spełnianie obietnic, 437 statyczne, 289, 377, 383-386 tworzenie obiektów, 159 właściwości. 300 zamkniete, 723 klauzula from, 561, 565, 570, 576, 571 group...by, 575, 576, 585 join, 577, 582, 585 orderby, 561, 564, 565, 571, 607 select, 561, 565, 570, 571, 604, 607 where, 561, 565, 571, 607 kod C# aplikacji, 45, 54, 58, 63, 184, 185, 252, 276, 290, 320, 382, 467, 488, 532, 538, 728, Patrz także aplikacie konsolowe C# w Unity, 257, 260, 261, 265, 267, 268, 406, 409 klasv ArrowDamage, 365 CardPicker, 142 Clown, 167 ComicAnalyzer, 584, 612 Deck, 528 Elephant, 232 Guy, 179, 182 HiLoGame, 290 Menultem, 253 NectarCollector, 601 Queen, 384, 385

ShoeCloset, 488 SwordDamage, 272, 318, 365 WeaponDamage, 364 poprawianie czytelności, 170, 172 refaktoryzacja, 174, 188 szablonowy, 719, 729 XAML aplikacji, 36, 152, 249, 251, 278, 374, 536, 537 kontrolki, 28, 94, 102, 249, 531 stronv. 91 zaplecza, code-behind, 92, 110, 153.156 kodowanie znaków, 655-659, 666, 672 kolejka, Queue<T>, 516, 522 operacje, 517 kolekcja, 473, 495 do wiązania danych, 525 kolejka, 516, 517, 522 lista, 483, 488, 563 ObservableCollection, 525 słownik, 512, 523, 563 stos, 516, 703-705 kolekcie generyczne, 490, 492, 495, 516 kwerendy, 556 wvświetlanie, 524 komentarze, 10, 66, 70, 81, 171, 219 wielowierszowe, 81 z dokumentacja XML, 172, 256 komparator, 500-502 kompilator, 135, 153 komponenty materiału. 117 skrvptu, 117 transformacji, 117, 119 komunikaty diagnostyczne, 281 konsola debugowania, 8, 13 konstruktor, 283, 309, 322 bezparametrowy, 305 główny, 474, 475 klasy bazowej, 359, 360, 475 podklasy, 360 prywatny, 322, 323 z parametrami, 305, 323 konstruktory przeciążone, 475 kontrakt, 438, 447 kontrolka Border, 248, 372 układ VerticalStackLayout, 372 właściwość Padding, 372 Button, 26, 88 dodawanie emoji, 42

kontrolka przeciąganie do kodu XAML, 29.35 właściwość Clicked, 37, 39, 95 właściwość HorizontalOptions, 95 właściwość SemanticProperties. Hint. 150 właściwość Text, 30, 94 właściwość x:Name, 30, 94 Checkbox właściwość IsChecked, 276, 277 właściwość x:Name, 277 zdarzenie CheckedChanged. 276.277 CollectionView, 524 właściwość HeightRequest, 524 właściwość ItemsSource, 529, 531 właściwość SelectionMode, 526 wyświetlanie elementów, 524 zdarzenie SelectionChanged, 526 ContentPage, 33 Entry, 89, 96 właściwość Placeholder, 97, 152 właściwość SemanticProperties. Description, 150 właściwość SemanticProperties. Hint, 95, 97 właściwość TextChanged, 99 FlexLavout. 34. 49 dodawanie przycisków, 35 Grid, 247, 248 definiowanie wierszy i kolumn, 249 właściwość ColumnSpacing, 372 właściwość Margin, 372 właściwość MinimumHeightRequest, 372 właściwość SemanticProperties. Description, 255 HorizontalStackLayout, 103, 110 Label, 31, 88, 89 ustawienie Margin, 99 właściwość BackgroundColor, 98 właściwość HorizontalOptions, 152 właściwość SemanticProperties. Description, 98, 150 właściwość Text, 98 właściwość x:Name, 98, 278 Picker, 89, 103, 104 ScrollView, 28, 34, 91 właściwość BindingContext, 529

Slider. 89, 101 właściwość SemanticDescription. Hint. 101 zdarzenie ValueChanged, 101 Stepper, 89, 101 obsługa zdarzenia ValueChanged, 101 Switch, 89 VerticalStackLayout, 33, 91, 103, 110 kontrolki. 88 automatyczna aktualizacia, 459 zapewnianie dostepności aplikacji, 95 edytowanie właściwości, 156 ustawianie właściwości, 40 zagnieżdżanie, 103, 110 konwersja, 221 automatyczna, 205 typów, 204, 440

L

kowariancja, 510, 511

liczby całkowite, 193 losowe, 244 zmiennoprzecinkowe, 194 LIFO, last in, first out, 518 LINO, Language-Integrated Ouery, 553, 556 grupowanie danych, 574 klauzule, 561, 571 kwerendv. 604 kwerendy dotyczące kolekcji, 556 łańcuch metod, 557, 562, 606-608 metody, 571, 619, 723 metody wyliczające elementy, 558 modyfikowanie zwracanego elementu, 572 przetwarzanie odroczone, 573 scalanie danych, 577 składnia kwerend. 561 słowo kluczowe descending, 571 join, 577 lista, List<T>, 483, 488, 563 indekser, 486, 492 metoda Add, 492 Contains, 492 IndexOf, 492 Remove, 492 RemoveAt, 489, 492 Sort, 498

operacje, 484 przetwarzanie, 488 tworzenie, 42, 483, 484 tvp. 492 właściwość Count. 492 listv genervczne tworzenie, 490 rzutowanie w góre, 510 sortowanie, 498-502 tworzenie, 496 użycie wyrażeń kolekcji, 496, 497 literał, 196, 207 losowanie, Patrz generator liczb losowvch lukier składniowy, 753, 755, 761

#### Ł

łańcuch metod LINQ, 557, 562, 606–608 strumieni, 630 znaków, 195, 476 interpolacja, 276 kodowanie, 655

#### Μ

materiał, 120, 126 MDA, Mechanics-Dynamics-Aesthetics, 389 mechanika gry, 84, 190, 244, 314, 368, 387 interfejsów użytkownika, 88 mechanizm fragmentów kodu, 76, 85 refleksji, 292 menu szybkich akcji, 61 Widok, 11 metoda, 39, 68-70, 85, 128 Array.Resize, 381, 482 Compare, 500, 502 CompareTo, 499 Concat, 571, 557 Console.ReadLine, 131, 143 Console.Write, 143 Console.WriteLine, 283 Convert.ToByte, 205 Convert.ToInt32, 205 Convert.ToInt64, 205 Convert.ToString, 205 Debug.WriteLine, 281, 283

Dispose, 637, 638, 696, 753 Empty, 613 Encoding.Unicode, 672 Encoding.UTF8.GetString, 639, 658,672 Enum.GetValues, 614 Enumerable.Range, 559 Environment.GetFolderPath, 643 File.OpenRead, 667 FileStream.Read, 672 GC.Collect. 693 GetEnumerator, 509 GroupBy, 608 IComparable.CompareTo, 508 IComparer.Compare, 508 int.TryParse, 131, 143 Join, 611 Length, 256 List.Sort, 500, 508 Main, 132, 144 MemoryStream.ToArray, 639 OrderBy, 619 OrderByDescending, 608 Path.Combine, 629 Random.Shared.Next, 206, 479 Random.Shared.NextDouble, 256 RandomSuit, 141 RandomValue, 142 Range, 613 RemoveAt, 489, 492 Repeat, 613 Select, 604, 605 SemanticScreenReader.Default. Announce, 321 SetValue, 256 Sort, 508 Stream.Read, 666 StreamReader.ReadBlock, 672 String.IsNullOrEmpty, 109 String.Join. 572 String.Split, 641 Take, 557e, 571 TakeLast, 557, 571 TimerTick, 62, 63, 390 ToArray, 495 ToBoolean, 245 ToList, 573 ToString, 505-508, 729 TrvParse, 214, 707 Where, 619 metody, 39, 68-70, 85, 128 abstrakcyjne, 397, 398

deklaracja, 604 kontrolowanie dostępu, 286 najbardziej specyficzna wersja. 364.402 niestatyczne, 166 obiektu, 159 parametr out, 706 parametr ref. 707 parametry, 130, 206 parametry opcjonalne, 708, 712 przeciażanie, 479 przesłanianie, 334, 341, 344, 354, 505.508 rozszerzające, 723, 724, 729 statyczne, 166 sygnatura, 322 typ zwracanej wartości, 156 ukrywanie, 354, 357 wirtualne, 355 wyodrębnianie, 581 wyrażenia lambda, 604 wywoływane poprzez rzutowanie, 417 zwracana wartość, 130 model klas. 331, 340, 341 klas systemu zarzadzania. 375 MDA, 389 modyfikator out, 706, 712 ref. 707, 712 modyfikator dostępu internal, 140, 211, 596, 619 private, 285, 288 protected, 349, 402, 452, 455 public, 286, 590 sealed, 699

#### Ν

narzędzie developer command prompt, 188 do obsługi wyjątków, 740 do refaktoryzacji, 575, 581 Narrator, 95 Tablica znaków, 657 VoiceOver, 95 nawiasy klamrowe, 39, 47, 76, 207 kwadratowe, 104, 133, 236, 253 ostre, 483, 490 nawigowanie po scenie, 763 NuGet, 758, 761

#### 0

obiekt, 158, 189 komparatora, 500, 501 prefab, 408, 413, 414 typu Button, 546, 552 FileStream, 623, 624 GZipStream, 623 IComparer, 500 List<T>, 483, 484, 496 MemoryStream, 623 NetworkStream, 623 Random.Shared, 253 StreamReader, 629 StreamWriter, 632 obiekty deserializacja, 649 finalizator, 692 graf, 647, 649, 652 grv. 117 inicializator obiektu, 180, 188 komunikacja między obiektami, 234 niemodyfikowalne, 717 pola, 646 serializacja, 647-649 stan, 646 tworzenie, 158, 160, 392, 396, 690 usuwanie, 690, 692 właściwości, 646 wyjątków, 736, 737, 761 wypakowywanie, 704 obsługa wyjatków blok catch, 756 blok finally, 744, 761 blok try-catch, 742-746, 761 zdarzeń, Patrz procedura obsługi zdarzeń odbiornik śledzenia, trace listener, 283, 296 odśmiecanie pamięci, 224, 225, 690-693, 697 okno Błędy, 64 C# Interactive, 188, 195 Dane wyjściowe, 281 Eksplorator rozwiazań, 10 Hierarchy, 114, 119, 126 Inne okna, 11 Inspector, 117, 119 IntelliSense, 31

okno

Konsola debugowania, 8 Lokalne, 75, 108 Project, 114, 126 Przybornik, 96 Quick Info, 131 Rozpocznii, 6 Scene, 114, 116 Utwórz nowy projekt, 22 Właściwości, 27, 30, 49, 99, 156 Wvrażenia kontrolne, 108 opakowanie wartości, 713 opakowywanie, 720 opcia <Nowa procedura zdarzeń>, 35 Create/Material, 674 Dane wyjściowe, 281 Debug Console, 281 Dodaj wvrażenie kontrolne, 75, 216 Dodaj/Istniejacy element..., 154, 574 Dodaj/Klasa..., 136 Przeidź do definicii, 99, 491, 492 Refaktoryzuj/Wyodrębnij metodę, 581 Resetui układ okna. 10 Rozpocznii debugowanie, 24 Start Debugging, 23, 24 Stop Debugging, 24 Szybkie akcje i refaktoryzacje..., 605 Test/Debuguj wszystkie testy, 593 UI/Button - TextMeshPro, 546 Ustaw jako projekt startowy, 615 Widok/Inne okna/C# Interactive. 188 Widok/Terminal, 668 Zarządzaj pakietami NuGet, 759 Zatrzymaj debugowanie, 24 operator, 68, 73 \*=.167 ??. 714. 729 ??=, 714, 729 +,73++,73+=.216lambda, =>, 598, 619 przypisania, 217 równości, 217 trójargumentowy, ?:, 603, 619 operatory dwuargumentowe, 217 logiczne, 77

relacyjne, 77 równości, 77

#### Ρ

pakiet, 758 .NET Core SDK, 15 AI Navigation, 677 Android SDK, 17 TMP essentials, 544 Xcode, 24 paleta poleceń, 17, 23 panel Character Viewer, 43 emoii, 43 konsoli debugowania, 16 parametry, 130 opcjonalne, 708, 712 petla, 68, 79, 85 do-while, 76, 79 for. 79, 80 foreach, 64, 133, 161, 509, 523 while, 76, 79 petle działające w nieskończoność, 81 sprzeżenia zwrotnego, 388, 391 plik MainPage.xaml, 23, 27, 28, 91, 94.152 MainPage.xaml.cs, 23, 27, 29, 99.154 Program.cs, 7, 66, 67, 134 z teksturą, 120 pliki .sln, 67 aplikacii konsolowei, 66 DLL, 759 odczyt, 622, 629 zapis, 622 zapis tekstu, 625 podklasy, 330, 335, 337, 341 dostep do metod, 436 ukrywanie metody, 354 wywołanie konstruktora, 359 podział wiersza, 109 zadań, 402, 434 pola, 62, 128, 165 iniciowanie, 313 instancji, 188 prywatne, 285, 288, 291 bezpieczeństwo, 292 publiczne, 287

statyczne, 166, 188, 290 tylko do odczytu, 451 wewnetrzne, 300, 323 polimorfizm, 469 procedura obsługi zdarzeń, 31, 37, 39, 40, 42, 54, 99, 102, 104, 105, 154, 156, 321, 381, 382, 390, 461, 468, 525, 526 program Parallels Desktop, 4 VirtualBox, 4 programowanie obiektowe abstrakcja, 472 dziedziczenie, 472 hermetyzacja, 472 podział zadań, 472 polimorfizm, 472 zasada DRY, 366, 402, 472 sterowane testami, 620 projekt, 67 .NET MAUI, 22 tworzenie w Unity, 404, 674 w Visual Studio, 6 w VSCode, 14 projektowanie gier, 19, 84, 112, 147, 314, 368, 586,654 klas, 173, 176 projekty, 6, 67 dodawanie pakietów, 758 dzielenie. 20 typu MSTest, 615 prototyp papierowy, 148 prototypy, 145, 147, 157 przeciążanie konstruktorów, 475 metod. 479 operatorów, 716 przegladarka danych szesnastkowych, 664-667, 732 przekazywanie przez referencje, 707 przez wartość, 707 przesłanianie metod, 334, 341, 344, 354, 505, 508 przestrzeń nazw, 128 globalna, 135, 156 o zakresie blokowym, 139 o zakresie pliku, 139 przetwarzanie odroczone, 585 przybornik, 96 z kontrolkami, 27, 29

przycisk, *Patrz* kontrolka Button, obiekt Button przypisanie wewnątrzwierszowe, 313 złożone, 217 punkt przerwania, 74, 92, 109, 216 wybór opcji, 262 wejścia, 134

#### R

ray casting, 673, 684 refaktoryzacja, 49, 174, 188, 310, 402, 575, 581, 600 klasy, 300 referencje, 223, 238, 452 do interfejsu, 426 do jednego obiektu, 232, 233, 256 do wielu obiektów, 236, 429 do podklasy, 341, 346 do projektu, 588 do strumienia, 638 do wywoływania metod ukrytych, 355 efekty uboczne, 226 kopiowanie, 225 null, 709 przekształcanie, 440 przestawianie, 231 przypisywanie, 700 typów interfejsowych, 429, 452 usuwanie, 225 refleksja, 292 rekordy, 717, 719, 729 kopiowanie, 718 w postaci struktury, 729 rekurencja, 705 renderowanie, 117 repozytorium Git, 51, 64 rozszerzanie interfejsów, 449, 450 klasy, 338, 378, 453, 723 bazowej, 342, 347 String, 724 System.Exception, 750 rozszerzenie dla VSCode .NET MAUI, 13 C# Dev Kit, 13 Unity, 13 rzutowanie, 202, 204, 212, 216, 221, 245, 417 na typ wyliczeniowy, 614

nieprawidłowe, 440 w dół, 442, 445, 453 w górę, 442–444, 453

#### S

scena 3D, 116 sekwencja, 571 tworzenie, 613, 614 ucieczki, 195, 639, 659, 672 serializacja, 644, 645, 652, 695 obiektów, 648 setter, 300, 309, 316, 323 prywatny, 303 siatka, 34, 119, 247, 277, 372, 534, Patrz także kontrolka Grid kod XAML, 374 ustawianie układu, 249, 373 silnik gry, 112 słowa kluczowe zarezerwowane, 198 słownik, Dictionary<TKey, TValue>, 512, 523, 563 deklaracja, 512 inicjalizator kolekcji, 515 metoda Add. 513 Remove, 513 TryGetValue, 523 operacje, 513 użycie indeksera, 513 właściwość Count, 513 wyszukiwanie wartości, 513 słowo kluczowe abstract, 394, 396-399, 402 as, 441, 453, 704, 715 base, 358-360, 402, 450 catch, 742, 761 class, 719, 729 const, 272 default, 327 descending, 564, 606 else, 78 enum, 477 event, 463, 468 get, 300, 323 interface, 418, 439 internal, 140, 211, 596, 619 is, 435-441, 450, 453, 704 namespace, 139 new, 140, 158, 160, 161, 188, 256 null, 241 object, 195 out, 131, 685, 706

override, 344, 348, 351, 352, 356, 357 private, 285, 289, 295, 296, 323 protected, 349, 402, 452, 455 public, 140, 286 readonly, 451, 453, 729 record, 717 ref, 381, 707 return, 130 set, 300, 323 static, 140, 166, 188, 289, 290 string, 195, 715 struct, 699, 719 switch, 609 this, 234, 235, 238, 256, 305, 323, 475, 723, 729 throw, 739 try, 742, 761 value, 300, 323 var, 566-571, 578, 580, 582, 585 virtual, 344, 351, 352, 356, 357, 376 void. 173 when, 754, 761 with, 718, 729 sortowanie list, 498 sprzeżenie zwrotne, 388, 391 sterta, 168, 188, 703, 704 stos, Stack<T>, 516, 703-705 operacje, 518 struktura, 699, 712, 715, 720 Nullable<T>, 713 strukturv jako typy bezpośrednie, 701 strumienie, 622 łączenie, 630 metoda Read, 623 Seek, 623 Write, 623 opróżnianie, 640 standardowe błędów, 671 wejścia, 671 wyjścia, 671 zamvkanie, 633, 638 system kontroli wersji Git, 50, 51, 64 Unity Version Control, 122, 126 zarzadzania. 370 dodanie interfeisu, 420, 430, 432 drzewo obiektów, 392, 393

system działanie, 371 hierarchia klas, 378 interfejs użytkownika, 372 klasa bazowa, 376 klasa statyczna, 384 kod zaplecza, 382 model klas, 375 podklasy, 380 procedura obsługi zdarzeń, 381 stałe. 377 w czasie rzeczywistym, 390 szablon Aplikacja konsoli, 7 Aplikacja platformy .NET MAUI, 22 sztuczna inteligencja, Patrz chatbot szybka akcja, 138

## Ś

ślad stosu, 593 środowisko CLR, 225, 229, 256, 690, 693

#### T

tablica. 133, 236 baitów. 660 łańcuchów znaków, 133 referencji do obiektów, 237, 239 tablice dodawanie elementu, 481 operacje, 486 tworzenie, 133, 156, 236 właściwość Length, 237 tekst zastępczy, 97 tekstura, 120, 126 testy jednostkowe, 587, 596, 619 asercje, 593 dostep do klas, 589, 590 metody, 594 nietypowe dane, 595 platforma MSTest, 619 przypadki brzegowe, 595 ślad stosu, 593 tworzenie, 588, 592 uruchamianie, 588 wzorzec Arrange-Act-Assert, 591 transformacje, 126 tryb debugowania, 74, 108 tworzenie aplikacji konsolowej, 6, 14, 132, 134 gry, 18, 21

instancji, 165, 169 interfejsów, 420 interfejsu użytkownika, 88, 372 klasy bazowej, 333, 336 konta Microsoftu. 5 list, 42, 496 obiektów, 158, 160, 169, 392, 396 obiektu komparatora, 501 projektu .NET MAUI, 22, 25, 49, 90.150 projektu Unity, 114 prototypów, 147 repozytorium Git, 51 sekwencji, 613, 614 słownika, 515 tablic, 133, 156, 236 tablicy obiektów, 444 wyrażeń switch, 609 typ bool, 192 bvte. 193 char. 195 decimal, 194, 200, 207, 220, 253 double, 192, 194, 200, 220 float, 192, 194, 200 Func. 607 IEnumerable<T>, 613 int, 192 long, 193 Nullable<T>,713 sbyte, 193 short, 193 string, 192, 195, 711 string?, 243, 256, 711 System.ValueType, 699 uint, 193 ulong, 193 ushort, 193 wyliczeniowy, 473, 477, 478 typy anonimowe, 578, 582, 585 automatyczna konwersja, 202 automatyczne wnioskowanie, 566 bezpośrednie, 207, 700, 701 dopuszczajace null, 713, 729 genetyczne, 483 jawne, 581, 585 konwersja automatyczna, 205 nieliczbowe, 200 obiektowe, 700 parametrów, 206 referencyjne, 701

dopuszczające null, 711 niedopuszczające null, 710 rzutowanie, 202 ze znakiem, 193 zmiennej, 71, 81, 85 zmiennoprzecinkowe, 194, 200 znakowe, 200

#### ν

układ FlexLayout, 34 ScrollView. 34 siatki, 247 VerticalStackLayout, 372 Wide, 115, 119 Unity, 13, 111, 257 albedo, 405, 414 automatyczny ruch postaci, 683 chodzenie po platformie, 766 debugowanie gier, 262, 266 dodanie obiektów do siatki, 767 platformy, 765 reakcji obiektów gry, 540 działanie wektorów trójwymiarowych, 266 edvtor, 112, 114 edvtor skrvptów. 115 instalacja, 113 interfejs użytkownika, 539, 544 kamera, 675 klasa GameController, 409, 540 Input, 770 MoveCamera, 682 MoveToClick, 683, 768 OneBallBehaviour, 406, 550 komponent, 117, 125, 126 Button, 546 Game Controller, 540 Light, 125 NavMesh Agent, 679, 686, 764 NavMesh Obstacle, 770 NavMesh Surface, 686, 766 Rigidbody, 414 Script, 552 Transform, 119, 675 luki w siatce nawigacyjnej, 680, 769 materiały, 120, 126, 405 metoda Awake, 686 Debug.DrawRay, 266, 270

Destrov, 540 GameObject. FindGameObjectsWithTag, 552 Input.GetAxis, 686 Input.GetKey, 686 Input.GetMouseButtonDown, 686,764 Instantiate, 414, 540 InvokeRepeating, 414 NavMeshAgent.SetDestination, 686 OnMouseDown, 540 Physics.Raycast, 685, 686 ResetBall. 551 ScreenPointToRay, 685, 764 StartGame, 548 transform.LookAt, 770 transform.Rotate, 270 transform.RotateAround, 270 transform.Translate, 552 Update, 414 modyfikowanie właściwości obiektów. 269 narzędzie Hand, 124 Move, 118 Move Gizmo, 126 Move Tool, 126 Rotate, 123 Scale Gizmo, 264 Scene Gizmo, 124, 126, 270 Transform, 118 View Tool, 126 obiekt Canvas, 544, 552 Cylinder, 676 EventSystem, 544 Main Camera, 544, 675, 686 Moving Obstacle, 769 Plane, 674, 686 prefab, 408, 413, 414, 681, 682 Sphere, 265, 676 Text. 545 obiektv GameObject, 117 podstawowe, 674 okno Inspector, 412 omijanie przeszkód, 769, 770 pakiet AI Navigation, 677-679 panel AI Navigation, 678, 679 Transform, 264

przycisk uruchamiający grę, 546 ray casting, 673, 684, 764 rejestrowanie aktualnego trybu, 543 rotacia obiektu, 268 scena 3D, 116 siatka nawigacyjna, NavMesh, 677, 680, 686, 764, 767 silnik fizvki, 413 skrypty C#, 258, 270, 410 GameController, 409, 540 OneBallBehaviour, 406, 550 MoveCamera, 682 MoveToClick, 683, 768 stosowanie struktur, 720 tekstury, 126 tryby gry, 541, 542 tworzenie obiektu, 406 tworzenie projektu, 114, 404, 674 uruchomienie gry, 411 wektory trójwymiarowe, 270 widok dwuwymiarowy, 544 izometryczny, 765, 770 perspektywiczny, 765, 770 Scene, 126, 267 Unity Hub, 113 Unity Version Control, 122, 126 uruchamianie aplikacji, 8, 16 aplikacji .NET MAUI, 24 Visual Studio, 5

#### V

Visual Studio, 1, 3 automatyczna obsługa zdarzeń, 27, 31 debuger, 74 dodawanie klas. 136 edytor skryptów Unity, 115 funkcja zmiany nazw, 585 generowanie metody, 61 konfigurowanie nowego projektu. 7 menedżer pakietów NuGet, 759 narzędzie do refaktoryzacji, 581 personalizowanie środowiska, 5 przycisk Attach to Unity, 262 szablon Aplikacja konsoli C#, 7 tworzenie nowego projektu, 6 uruchamianie. 5 wybór opcji instalatora, 4 wybór składników, 4 znaki Unicode, 657

Visual Studio Code, Patrz VSCode Visual Studio Community Edition, 4.11 VSCode .NET MAUI, 13 C# Dev Kit. 13 dane wyiściowe aplikacji, 13 dodawanie klas, 136 instalowanie środowiska, 12 konfigurowanie, 17 okno błedu. 25 paleta poleceń, 17 panel Extensions, 13 tworzenie nowego projektu, 14-16 tworzenie projektu .NET MAUI. 23 Unity, 13 uruchamianie aplikacji, 16

#### W

wartość NaN. 194 null, 241, 710, 713, 714 Time.deltaTime, 263, 270 wiazanie danych, 459-468 w kodzie XAML, 529 wielodziedziczenie, 401 wiersz poleceń argumenty, 669, 672 uruchamianie aplikacji, 668 właściwości, 300, 323 abstrakcyjne, 398 automatycznie implementowane, 302, 323 inicjowanie, 304, 305, 313, 322 kontrolek, 40 semantyczne, 94, 110 tylko do odczytu, 303 znaczników, 30 wyjątek, 739 ArgumentNullException, 606 IndexOutOfRangeException, 736 **IOException**, 748 KeyNotFoundException, 523 NotImplementedException, 750 NullReferenceException, 710-712 System. IndexOutOfRangeException, 744 System. IO.FileNotFoundException, 738 System. UnauthorizedAccessException, 740,743

wyjątki filtry, 754, 761 nieobsługiwane, 732-734, 735 wviatki niestandardowe, 750, 761 przekazywanie w górę, 749 wyliczenia, 473, 477, 478, 487, 495 rzutowanie, 478 wyodrębnianie metod, 581 wvrażenia kolekcji, 156, 256, 496, 508, 514 kontrolne, 75, 93, 216 lambda, 598, 602 refaktoryzacja, 600 wzorzec, 162 Arrange-Act-Assert, 591

#### Х

XAML, eXtensible Application Markup Language, 49

#### Ζ

zakładka Przybornik, 29 zasada DRY, 366, 402, 472 zdarzenie, 31, 49, 64 CheckedChanged, 276, 277 Clicked, 35, 468 SelectionChanged, 526 ValueChanged, 101 zegar, 59, 60, 64, 390 zintegrowane środowisko programistyczne, IDE, 3 zmienne, 68, 199 nazwa, 81 przypisanie wartości, 71, 85 referencyjne, 222, 236, 237 tablicowe, 236, 256 typ, 70, 73, 81, 85 znacznik <Button>, 91 <ContentPage>, 34, 91, 97, 530

<ContentPage.Resources>, 529 <Grid>, 248, 534 <Image>, 91 <Label>, 91 <ScrollView>, 34, 97 <Shell>, 152 <ShellContent>, 152 <VerticalStackLavout>, 28, 33, 97 znaczniki otwierajace, 28 samozamvkajace, 33 właściwości, 30 zamykające, 28 znak !. 109 \$,93 ?, 711-713 @, 639 ~, 692, 712 dwukropka, :, 342, 418, 453 null, \0, 672

# 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



## UWAGA! Podręcznik przyjazny dla mózgu!

C# ma ugruntowaną pozycję jednego z najważniejszych języków programowania. Nowoczesny, wszechstronny i dojrzały, a do tego sukcesywnie rozwijany, zapewnia efektywne tworzenie kodu o wysokiej jakości. Nic nie stoi na przeszkodzie, aby C# stał się Twoim pierwszym językiem programowania i przy okazji pozwolił Ci się świetnie bawić!



Ta książka, podobnie jak inne pozycje z serii **Rusz głową!**, została przygotowana zgodnie z najnowszymi odkryciami nauk poznawczych, teorii uczenia się i neurofizjologii. W praktyce oznacza to, że dzięki niej zaangażujesz swój mózg, użyjesz wielu zmysłów i niepostrzeżenie przyswoisz język C# i umiejętność pracy w Visual Studio. Dowiesz się, jak pisać gry 3D w Unity i korzystać z LINQ. A to wszystko dzięki łamigłówkom, ćwiczeniom i tworzeniu rzeczywistych aplikacji. Intuicyjnie zrozumiesz ważne techniki i zagadnienia programistyczne. Nie musisz mieć doświadczenia – tylko chęć do nauki!



Wielkie dzięki! Wasze książki pomogły mi w rozpoczęciu kariery?

– Ryan White programista gier

Zwięzłe, rzetelne i – przede wszystkim – ciekawe wprowadzenie do programowania w C#!

Jon Galloway
 Microsoft

Latarnia geniuszu! Przekazuje wiedzę, pobudza ciekawość i podsyca pasję do programowania!

Gerald Versluis
 Microsoft

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.