O'REILLY®

Wydanie V

Head First C#

Rusz głową!

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

Andrew Stellman
Jennifer Greene

Helion

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

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-289-2113-9

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.

Spis treści (skrócony)

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

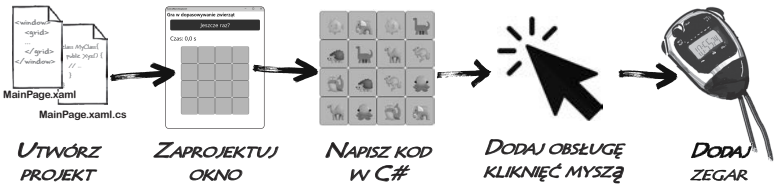
C# według Twojego mózgu. Siedzisz i próbujesz się czegoś *nauczyć*, ale *mózg* stale Ci powtarza, że cała ta nauka *nie jest ważna*. Twój umysł mówi: „Lepiej wyjdź z pokoju i zajmij się ważniejszymi sprawami — dowiedz się, których dzikich zwierząt unikać, a także sprawdź, czy jazda na snowboardzie nago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

Zacznij pisać programy w języku C#

1 Zbuduj coś wspaniałego... szybko

Chcesz tworzyć fantastyczne aplikacje... od razu?

C# to nowoczesny język programowania i **cenne narzędzie**, które masz na wyciągnięcie ręki. Ponadto w postaci **Visual Studio** otrzymujesz fantastyczne środowisko programistyczne z wysoce intuicyjnymi funkcjami, które sprawiają, że pisanie kodu jest 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!***

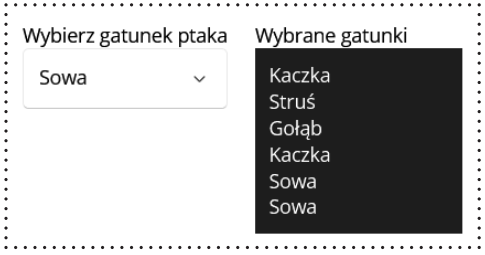


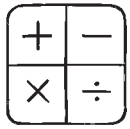
Zmienne, instrukcje i metody

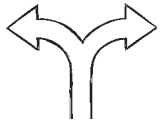
2 Zanurz się w kod C#

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.



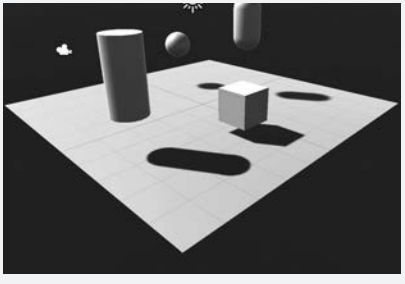


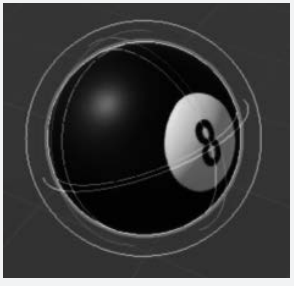


Ćwiczenia z Unity nr 1

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.



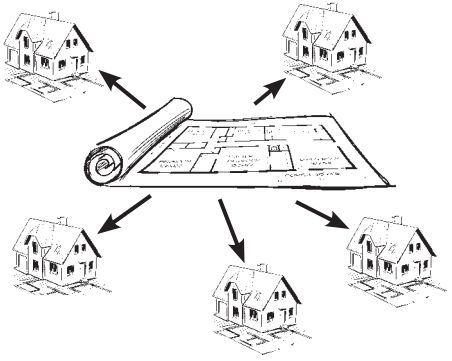


Przestrzenie nazw i klasy

3 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.

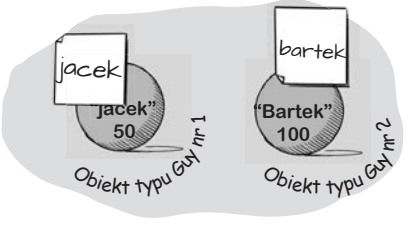


Dane, typy, obiekty i referencje

4 Zarządzanie danymi aplikacji

Dane i obiekty to cegiełki aplikacji.

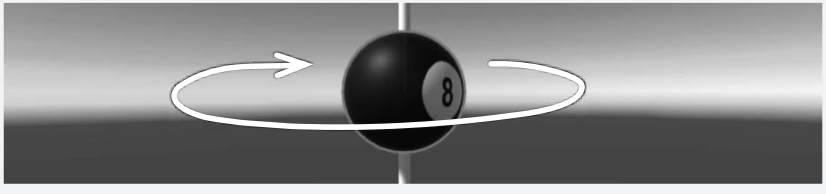
Czym byłyby aplikacje bez danych? Zastanów się nad tym przez chwilę. Bez danych programy są… no cóż, trudno nawet wyobrazić sobie pisanie kodu bez danych. Potrzebujesz **informacji** od użytkownika i korzystasz z nich do generowania nowych informacji do zwrócenia. Prawie wszystko, co robisz w trakcie 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!).



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.

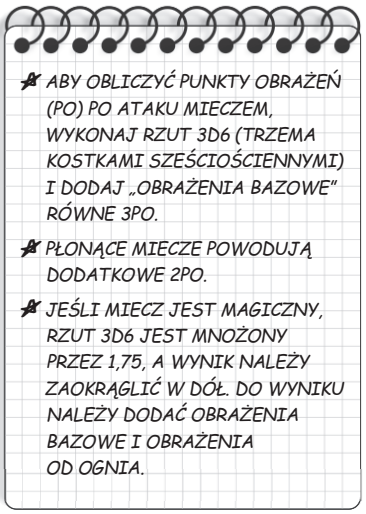


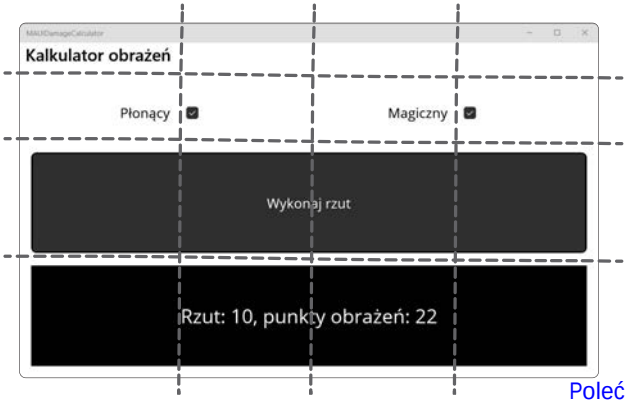
Hermetyzacja

5 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.



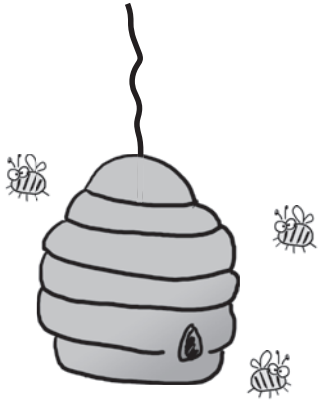


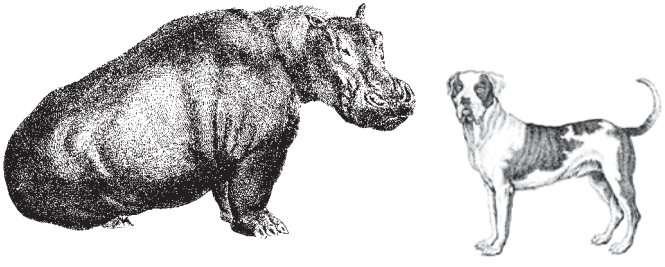
Dziedziczenie

6 Drzewo genealogiczne obiektów

Czasem CHCESZ być taki jak Twoi rodzice.

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

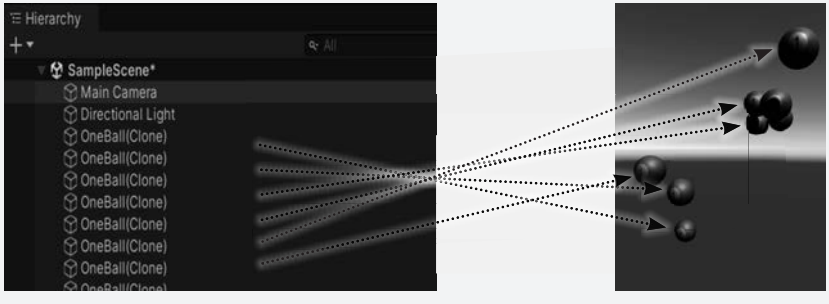




Ćwiczenia z Unity nr 3

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.



Interfejsy, rzutowanie i instrukcja „is”

7 Spełnianie obietnic przez klasy

Chcesz, aby obiekt wykonywał określone zadanie? Zastosuj interfejs.

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

Wyliczenia i kolekcje

8 Porządkowanie danych

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

W praktyce nie otrzymujesz danych w postaci uporządkowanych porcji i elementów. Nic z tego — dane będą trafiać do Ciebie w 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.

Rzadko używana karta „książę bawołów".

Sortowanie według gatunków kaczek.

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.

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

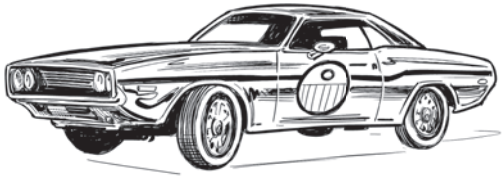
Po dodaniu ostatniej bili gra przechodzi w tryb *Koniec gry*. Wyświetlany jest przycisk *Zagraj ponownie*, a gra nie dodaje nowych bil.

LINQ i lambdy

9 Kontroluj swoje dane

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

Minęły już dni, kiedy można było dniami, a nawet tygodniami programować bez przetwarzania **dużych ilości danych**. Dziś ***wszystko kręci się wokół danych***. Tu właśnie przydaje się technologia **LINQ**. Jest to mechanizm języka C# i platformy .NET, który nie tylko pozwala w intuicyjny sposób **pisać kwerendy danych** z kolekcji platformy .NET, ale też **grupować dane** i **scalać dane z różnych źródeł**. 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.

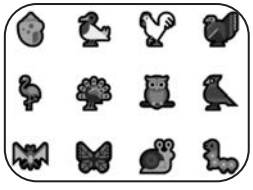


Odczyt i zapis plików

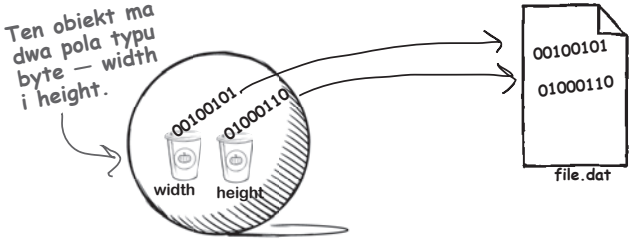
10 Zachowaj dla mnie ostatni bajt

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**.



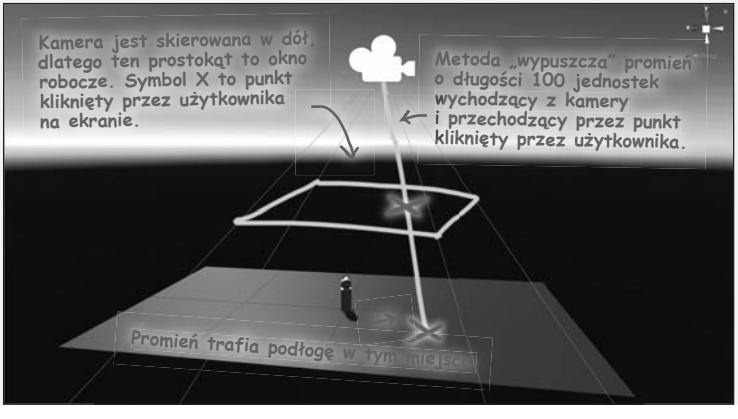
```
0000: 54 6f 20 65 6c 65 6d 65 To eleme
0005: 6e 74 61 72 6e 65 2c 20 ntarne,
0010: 64 72 6f 67 69 20 57 61 drogi Wa
0015: 74 73 6f 6e 69 65       tsonie
```

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.

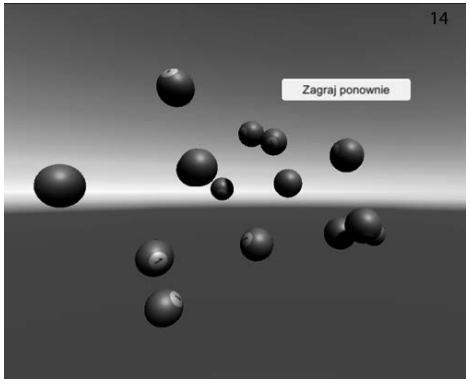


KAPITAN WSPANIAŁY

ŚMIERĆ OBIEKTU

C#. Rusz głową

4 zł | Rozdział 11.



12

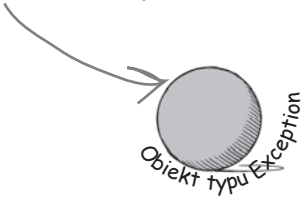
Obsługa wyjątków

Gaszenie pożarów robi się nudne

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.

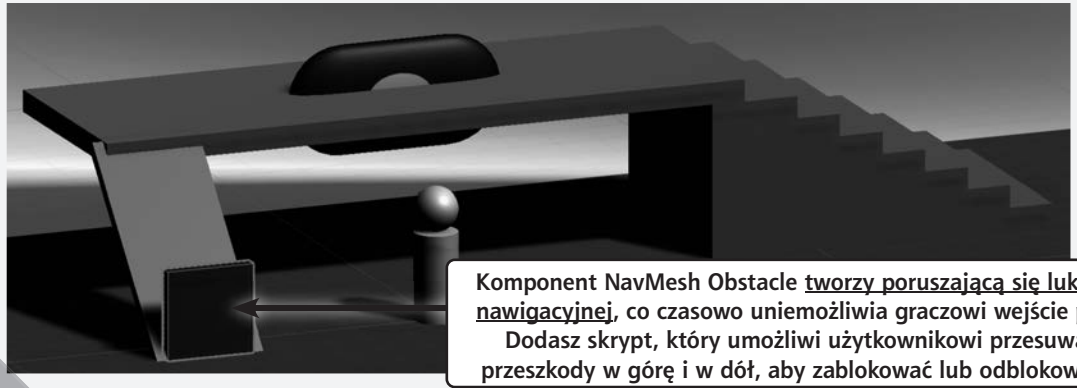

```
int[] anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```



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#.



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

Jesteś gotów?

1. Zacznij pisać programy w języku C#

Zbuduj coś wspaniałego... szybko!

Chcesz tworzyć fantastyczne aplikacje... od razu?

C# to nowoczesny język programowania i **cenne narzędzie**, które masz na wyciągnięcie ręki. Ponadto w postaci **Visual Studio** otrzymujesz fantastyczne środowisko programistyczne z wysoce intuicyjnymi funkcjami, które sprawiają, że pisanie kodu jest 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

Pisz kod i poznawaj C# z Visual Studio

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#.

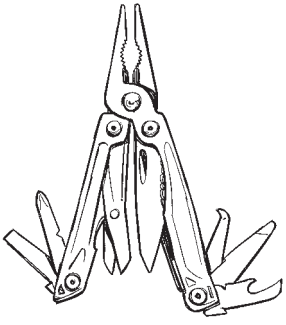
Najlepszym sposobem na rozpoczęcie pracy z C# jest **pisanie dużych ilości kodu**.

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 bezpłatna 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#.

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

Wskazówki dotyczące IDE

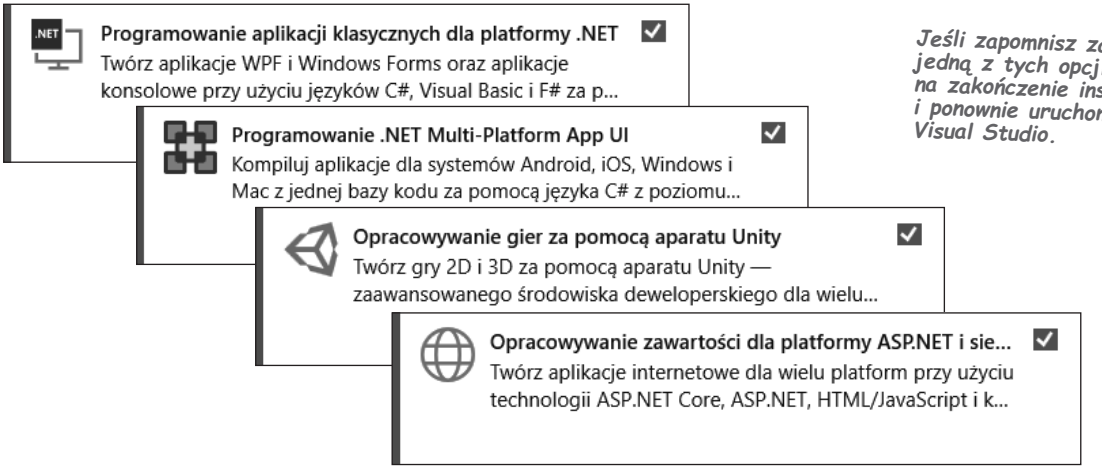
W tej książce Visual Studio często będzie nazywane po prostu **„środowiskiem IDE"**. Zwracaj uwagę na przydatne wskazówki dotyczące środowiska IDE, które pomogą Ci w wydajniejszym pisaniu kodu.

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*.



Jeśli zapomnisz zaznaczyć jedną z tych opcji, poczekaj na zakończenie instalacji i ponownie uruchom instalator Visual Studio.

Ta książka została napisana na komputerze Mac. Wszystkie zrzuty ekranu Visual Studio i Unity w oryginalnym wydaniu książki zostały wykonane w systemie Windows 11 na maszynie wirtualnej Parallels Desktop.

Możesz używać środowiska Visual Studio for Mac, ale Microsoft nie zapewnia już dla niego wsparcia technicznego.

Środowisko Visual Studio for Mac było świetnym narzędziem do nauki języka C# na komputerach Mac! Jednak kiedy kończyliśmy to wydanie C#. Rusz głową, Microsoft ogłosił, że zaprzestanie jego rozwoju i zakończy wsparcie w 2024 roku. Jeśli korzystasz z systemu macOS lub Linux, Microsoft zaleca uruchomienie maszyny wirtualnej z systemem Windows za pomocą oprogramowania takiego jak VirtualBox lub Parallels Desktop.

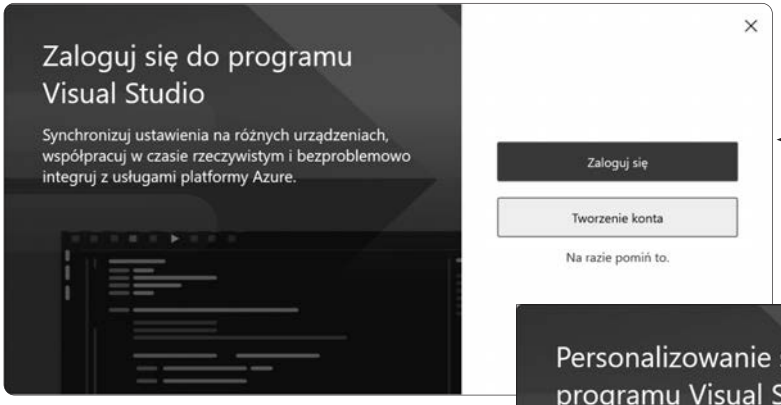
Łatwym sposobem na rozpoczęcie jest pobranie bezpłatnie VirtualBox ze strony https://www.virtualbox.org, a następnie maszyny wirtualnej z systemem Windows Microsoftu, która zawiera wersję ewaluacyjną tego systemu (https://developer.microsoft.com/en-us/windows/downloads/virtual-machines).

Dalej w rozdziale pokazujemy też, jak korzystać z systemu **Visual Studio Code**.

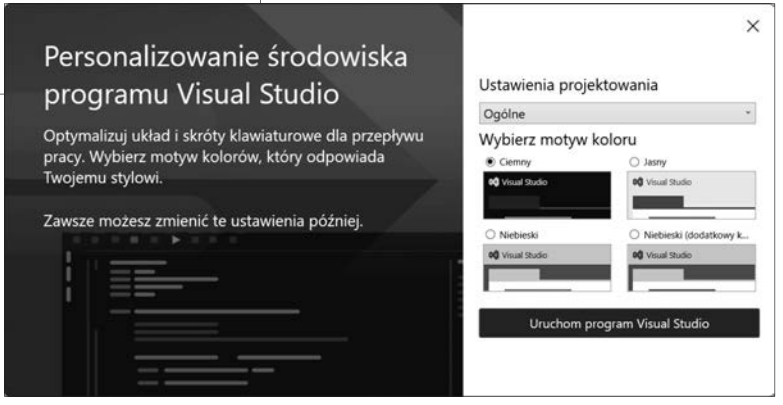
Wszystkie projekty z tej książki można uruchomić za pomocą środowiska Visual Studio Code w systemach Windows i macOS.

Uruchamianie Visual Studio

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



Przy pierwszym uruchomieniu Visual Studio może poprosić o zalogowanie się na konto Microsoftu. Jeśli go nie posiadasz, możesz je utworzyć. W tym celu kliknij przycisk *Tworzenie konta*. Microsoft aktywuje następnie bezpłatną licencję na wersję Community środowiska Visual Studio.



Pojawi się również prośba o wybranie motywu kolorystycznego. Na zrzutach ekranu w tej książce używamy motywu jasnego, ponieważ lepiej wygląda w druku, ale wielu użytkowników preferuje motyw ciemny.

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.

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.

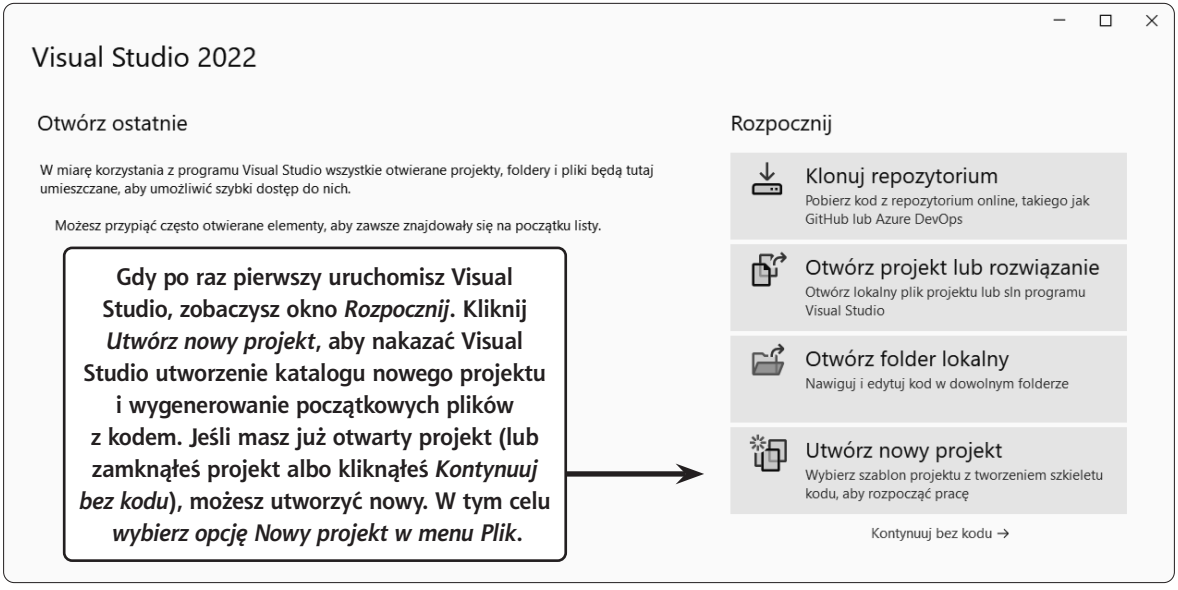
Jeśli napotkasz problemy z instalacją Visual Studio lub uruchomieniem pierwszego projektu, odwiedź nasz kanał w serwisie YouTube *(https://www.youtube.com/@headfirstcsharp)*, aby zobaczyć filmy z całego procesu instalacji.

Utwórz i uruchom pierwszy projekt C# w Visual Studio

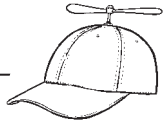
Najlepszym sposobem nauki języka C# jest rozpoczęcie pisania kodu. Dlatego w tej książce będziesz pisać dużo kodu (i tworzyć wiele aplikacji!). Dla każdej aplikacji dodasz osobny **projekt**, czyli generowany przez Visual Studio katalog ze specjalnymi plikami, który służy do porządkowania całego kodu.

1 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.



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).



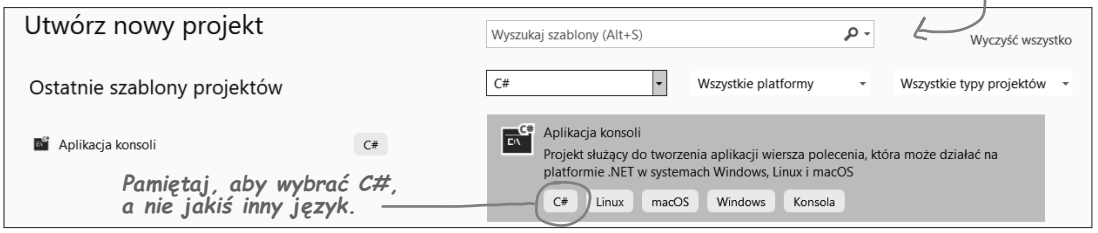
Pora napisać trochę kodu!

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).

2 Wybierz szablon projektu, jaki Visual Studio ma zastosować.

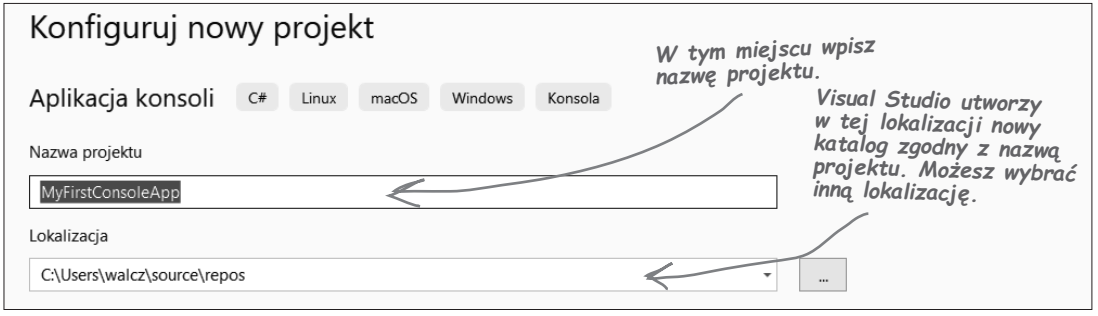
Visual Studio tworzy nowe projekty za pomocą *szablonu*, który określa, jakie pliki należy wygenerować. **Wybierz szablon *Aplikacja konsoli*** i kliknij przycisk *Dalej*.

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



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.



4 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.

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

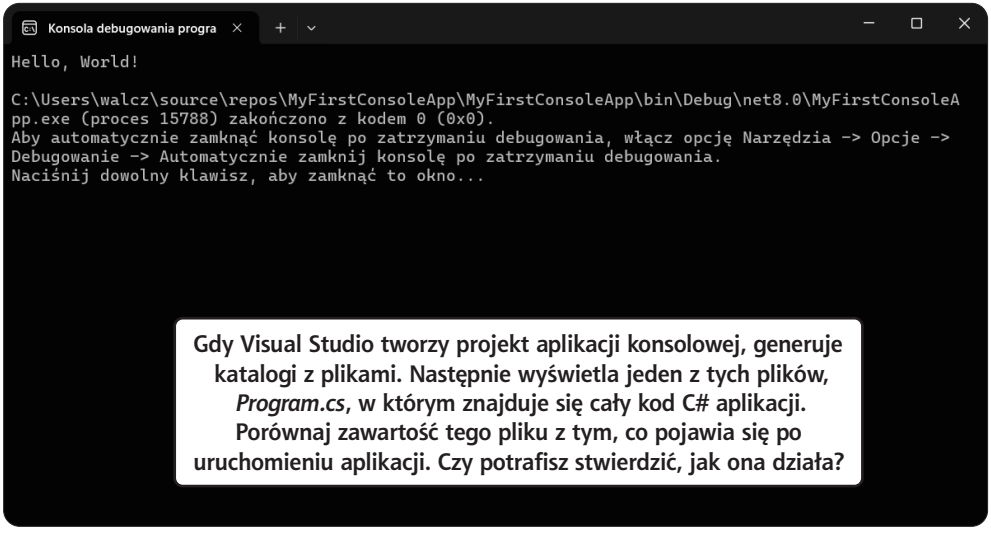
Kod aplikacji znajduje się w pliku *Program.cs*. Możesz go edytować w pokazanym tu oknie.

5 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:

6 Przyjrzyj się danym wyjściowym programu.

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



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.

Zaostrz ołówek

Visual Studio to świetne narzędzie pomagające w nauce i poznawaniu języka C#. W tej książce będziesz pisać dużo kodu, więc jest to świetny czas na zapoznanie się z tym środowiskiem.

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 najważniejszych sposobów na przyswojenie sobie nowych pomysłów, informacji, umiejętności i narzędzi jest **zapisanie informacji**. Przyjrzyj się więc dokładnie różnym częściom Visual Studio i zanotuj, do czego Twoim zdaniem służą. Nie musisz mieć całkowitej pewności, po prostu zgaduj!

.....

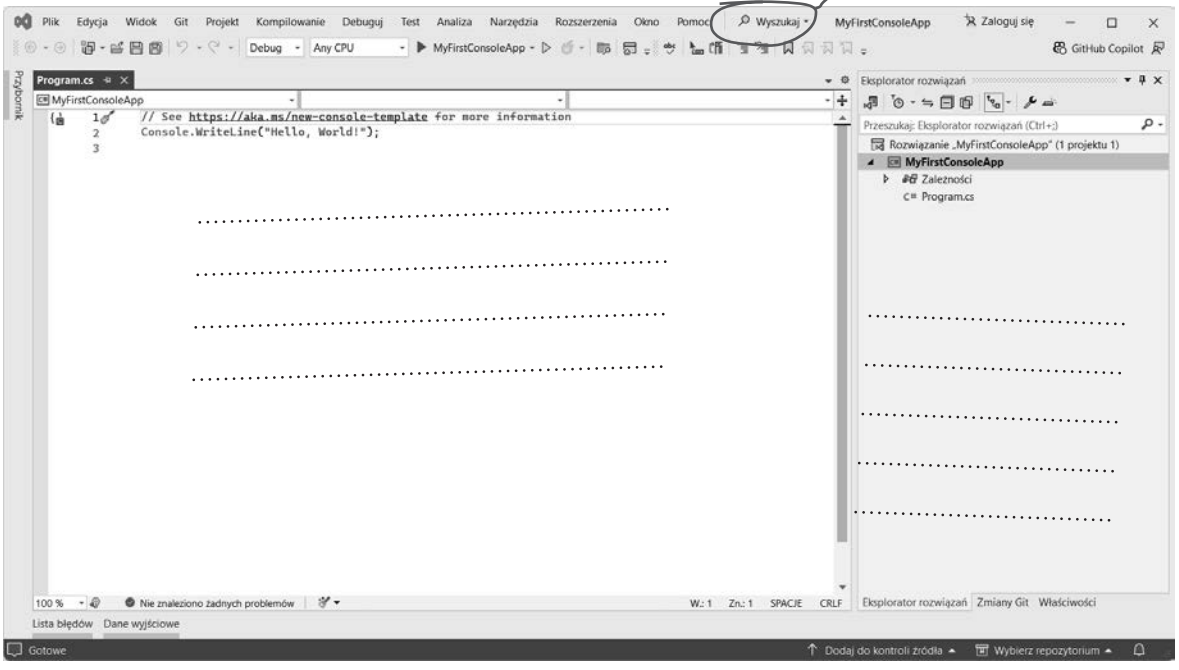
.....

.....

.....

.....

.....



.....

.....

.....

.....

.....

.....

.....

.....

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.WriteLine("Hello, World!");
```

.....

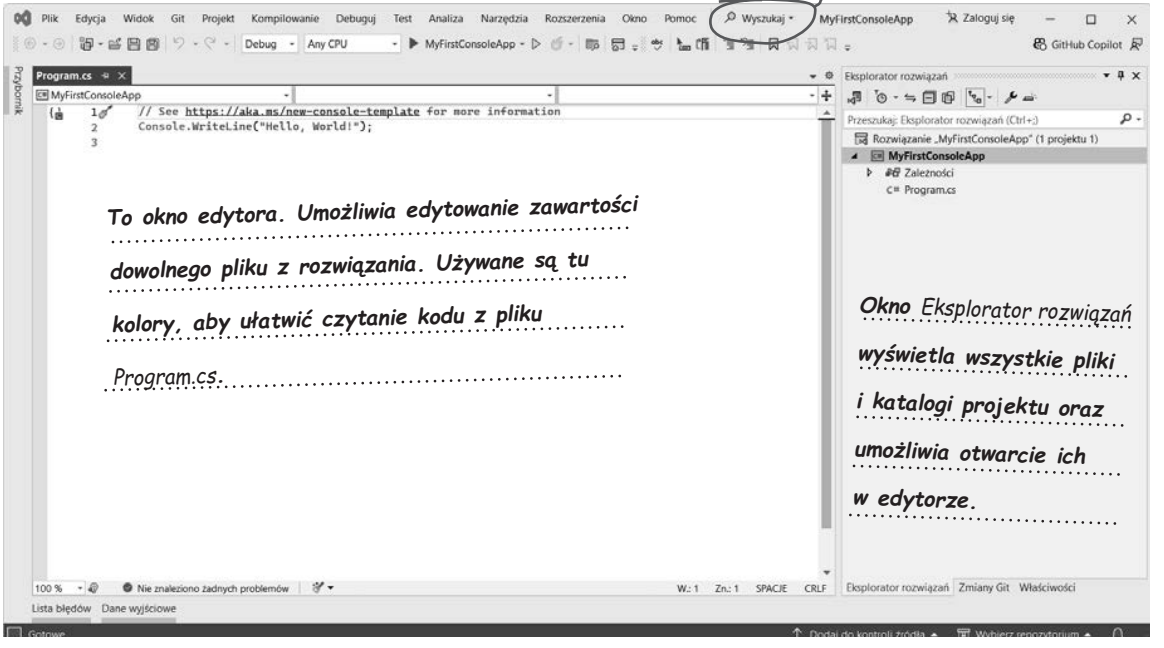
Spróbuj poprzesuwać panele w Visual Studio. Kliknij przycisk pinezki (📌), aby zwinąć okno Eksplorator rozwiązań przy panelu bocznym. Zresetuj układ. W tym celu wybierz opcję Resetuj układ okna z menu Okno, a następnie opcję Tak.

Zaostrz ołówek

Rozwiązanie

Możesz otworzyć wiele plików jednocześnie i przełączać się między nimi za pomocą kart.

Pole Wyszukaj pozwala otworzyć okno, które umożliwia przeszukiwanie kodu w projekcie i znajdowanie funkcji Visual Studio.



To okno edytora. Umożliwia edytowanie zawartości dowolnego pliku z rozwiązania. Używane są tu kolory, aby ułatwić czytanie kodu z pliku Program.cs.

Okno Eksplorator rozwiązań wyświetla wszystkie pliki i katalogi projektu oraz umożliwia otwarcie ich w edytorze.

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!

Nie ma głupich pytań

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

O: Nie. IDE świetnie sobie radzi z generowaniem fragmentów kodu, ale ma swoje ograniczenia. Niektóre zadania wykonuje bardzo dobrze — na przykład przygotowuje solidny punkt wyjścia dla programisty i automatycznie zmienia właściwości kontrolek w interfejsie użytkownika. 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.

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

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

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

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

O: 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.

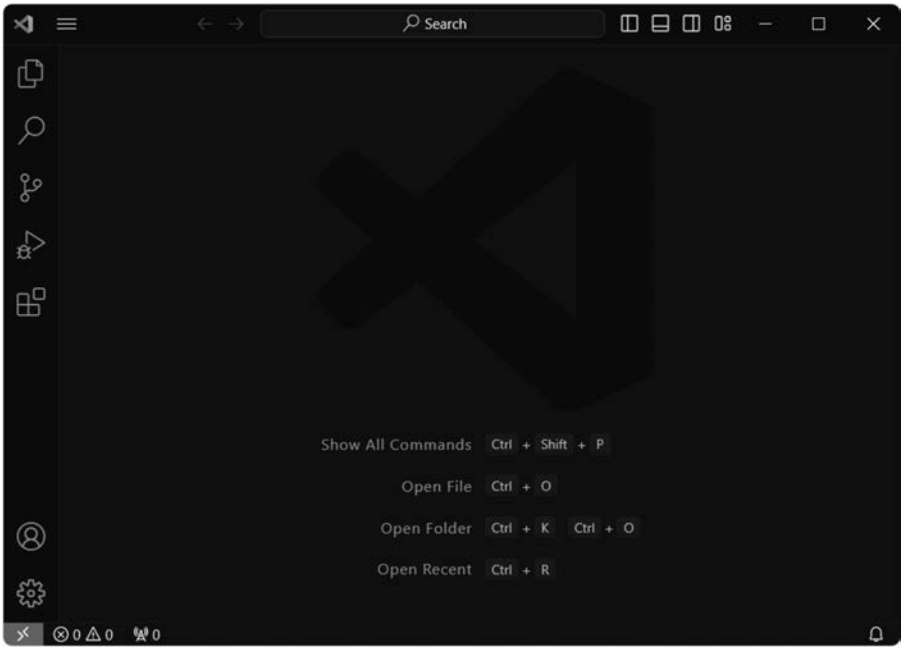
W trakcie lektury C#. Rusz głową możesz używać Visual Studio Code

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.



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

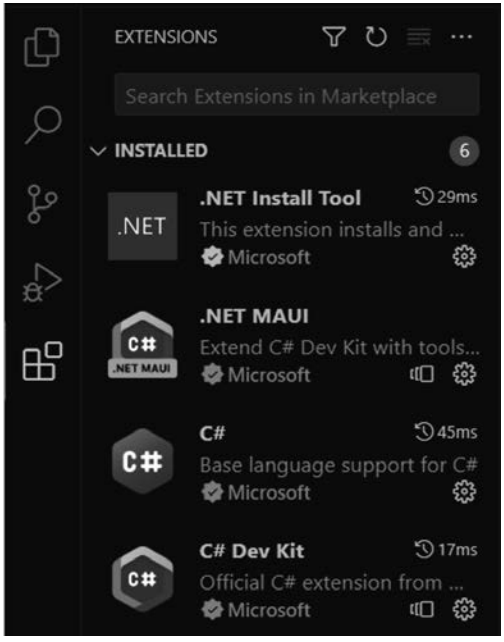
Korzystanie z Visual Studio Code jest opcjonalne. W systemach Windows i macOS możesz użyć tego narzędzia do wykonania wszystkich projektów z tej książki. Czytelnicy korzystający z Linuksa mogą potrzebować emulatora Androida, aby wykonać projekty związane z .NET MAUI. Więcej na ten temat dowiesz się dalej.

> **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.**

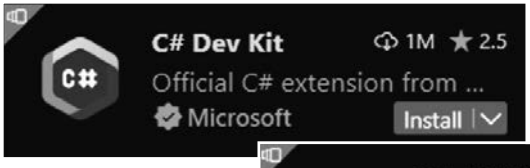
Instalowanie rozszerzeń dla języka C#

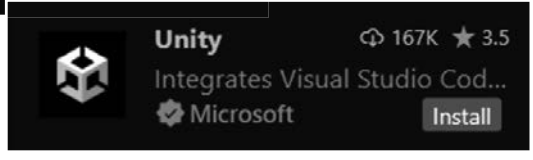
Kliknij przycisk Extensions () po lewej stronie okna środowiska VSCode, aby otworzyć panel Extensions. W górnej części panelu znajduje się pole wyszukiwania z tekstem „Search Extensions in Marketplace”. **Znajdź wszystkie wymienione niżej rozszerzenia**:

- ★ **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.



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



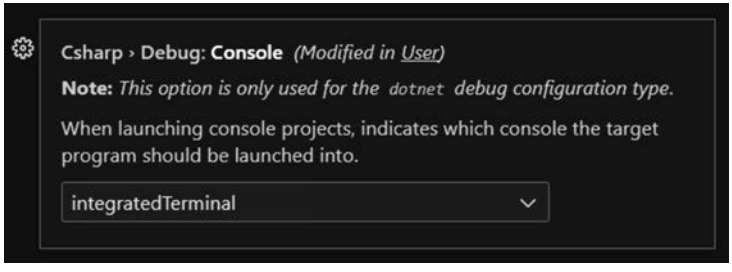


Po zainstalowaniu rozszerzeń VSCode może wyświetlić prośbę o restart systemu. Może też wyświetlić zakładki Getting Started z przydatnymi informacjami.

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.

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 ⌘+przecinek). Wyszukaj ustawienie **csharp.debug.console**. Powinna się pojawić lista rozwijana z kilkoma opcjami. Zmień ustawienie na **integratedTerminal**.



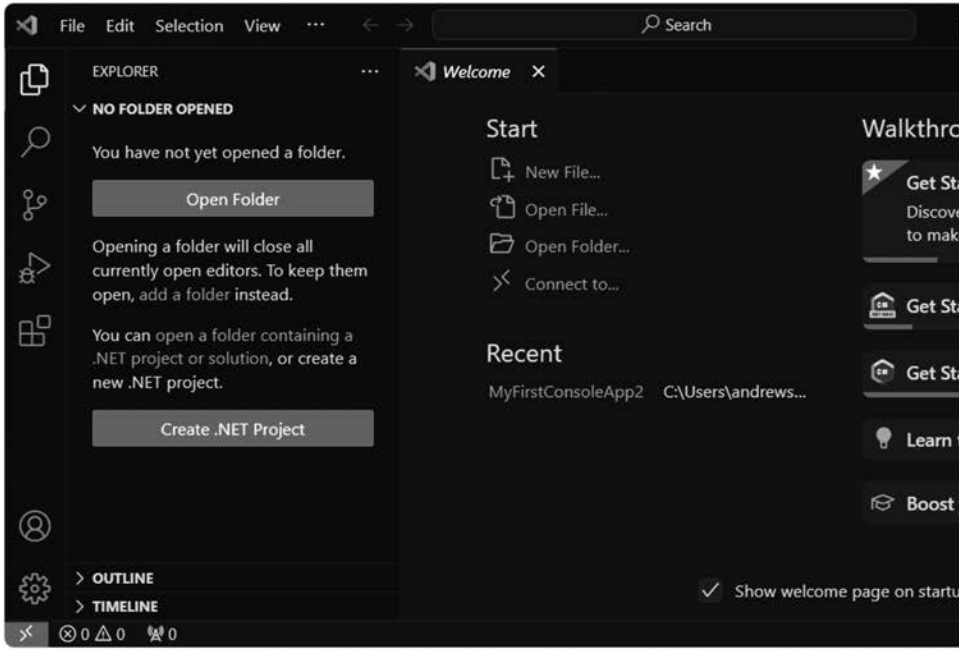
Teraz wszystko jest gotowe do rozpoczęcia pisania kodu C#!

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.

W systemie Windows menu znajduje się w górnej części okna (jeśli jest ono bardzo wąskie, kliknij przycisk menu , aby je wyświetlić). W systemie macOS menu znajduje się na pasku w górnej części ekranu.

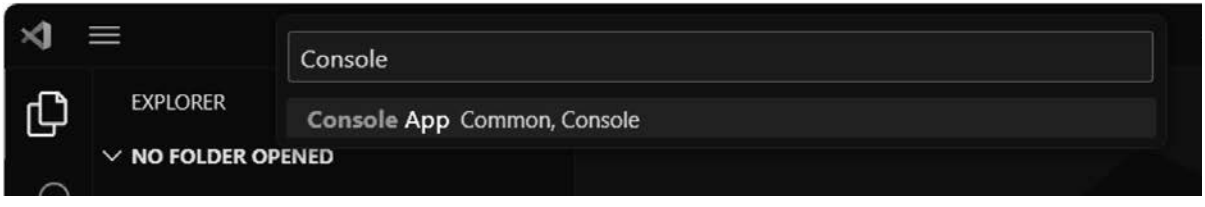
Kliknij ten przycisk, aby otworzyć lub zamknąć lewy panel. Jeśli w danym momencie w VSCode nie ma otwartego katalogu, panel będzie zawierał przyciski umożliwiające otwarcie folderu i utworzenie nowego projektu .NET.



Jeśli nie widzisz przycisku *Create .NET Project*, możesz otworzyć paletę poleceń (za pomocą kombinacji *Ctrl+Shift+P* lub ⇧+⌘+*P*), a następnie w menu, które się pojawi, wybrać opcję *.NET: New Project*.

1 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.



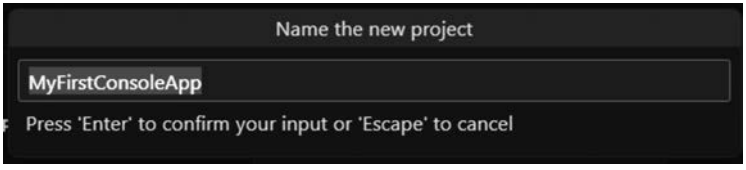
2 Wybierz katalog dla nowego projektu.

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***.

3 Podaj nazwę projektu.

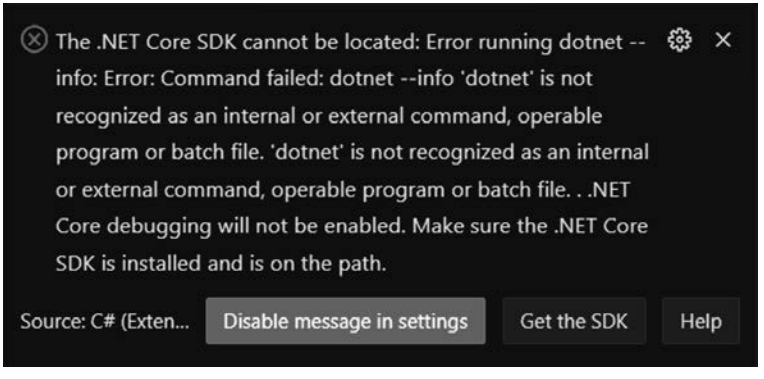
Każdy projekt C# ma nazwę. Zazwyczaj katalog projektu ma taką samą nazwę jak sam projekt. Po wybraniu katalogu VSCode wyświetli monit o podanie nazwy projektu:



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.

4 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*.



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.

5 Rozwiń eksplorator rozwiązań i otwórz plik *Program.cs*.

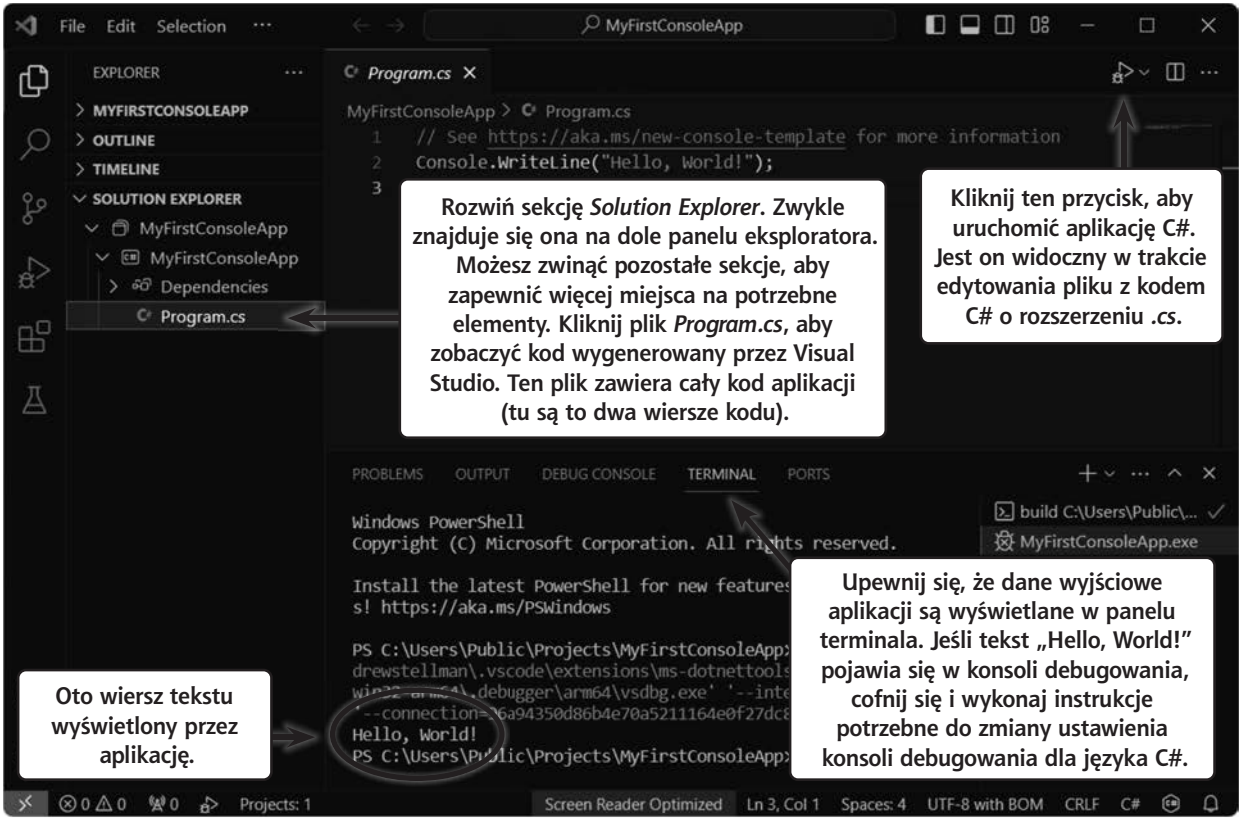
Gdy VSCode utworzy projekt C#, panel eksploratora po lewej stronie będzie zawierał kilka zwijanych sekcji. VSCode używa plików i katalogów, a **eksplorator** służy do ich przeglądania i otwierania do edycji.

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.***



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

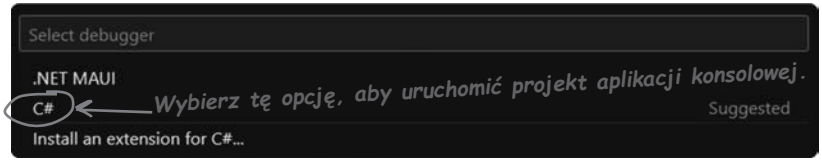
VSCode jest niezwykle elastyczny i można go używać na wiele sposobów. Jeśli dopiero poznajesz to narzędzie, zalecamy **tworzenie nowego katalogu dla każdego projektu omawianego w książce**. Gdy tworzysz nowy projekt, **wybierz *Close Folder*** z menu *File*, aby zamknąć bieżący katalog. Następnie utwórz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

W ten sposób można zresetować VSCode, aby rozpocząć nowy projekt.

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 ⇧+⌘+*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.



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

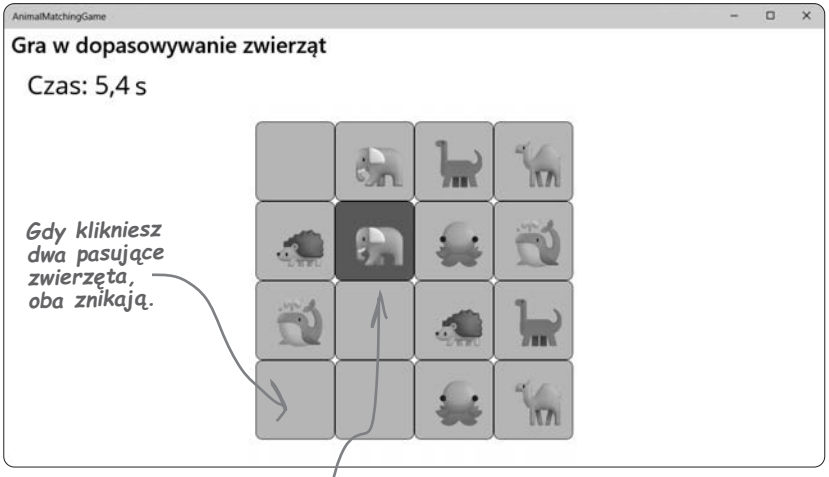
.NET MAUI nie działa natywnie w systemie Linux. Jeśli masz urządzenie z Androidem, możesz debugować kod bezpośrednio w nim. Na podanej niżej stronie pokazano, jak skonfigurować urządzenie z systemem Android, aby można było podłączyć je do komputera i uruchamiać na nim aplikacje MAUI: *https://learn.microsoft.com/dotnet/maui/android/device/setup*.

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.

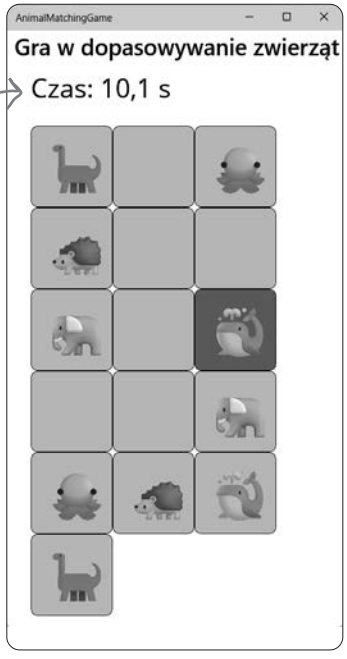
Gra wyświetla szesnaście przycisków z rozmieszczonymi losowo ośmioma parami zwierząt. Gracz klika dwa zwierzęta: najpierw pierwsze, a potem pasujące do niego. Gdy dopasujesz osiem par, wygrywasz!



Gdy klikniesz dwa pasujące zwierzęta, oba znikają.

Po kliknięciu pierwszego przycisku zmienia on kolor. Jeśli klikniesz jego odpowiednik, oba zwierzęta znikną. Jeżeli wybierzesz dowolne inne zwierzę, kolor pierwszego przycisku ponownie się zmieni i trzeba zacząć od nowa z nową parą.

Aby gra była bardziej ekscytująca, zaraz po jej rozpoczęciu uruchamiany jest licznik czasu. Czy uda Ci się pobić rekord?



Gdy zmienisz wielkość okna, układ przycisków ze zwierzętami dopasuje się do nowej szerokości.

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).

Gdy odnajdziesz wszystkie osiem par zwierząt, gra wyświetli duży przycisk „Jeszcze raz?", a pod nim uzyskany czas. Kliknij przycisk, aby zresetować grę i zacząć od nowa!

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

Projektowanie gier... i nie tylko

Czym jest gra?

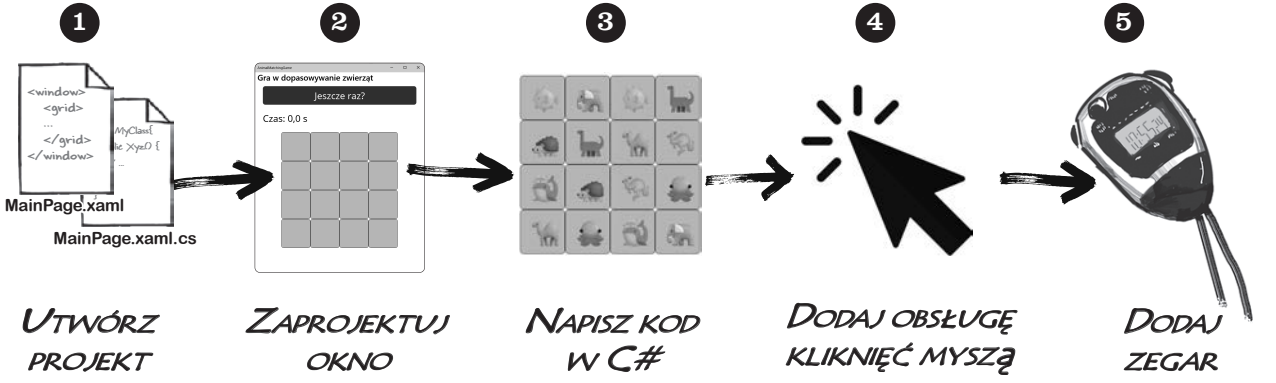
Może się wydawać, że to oczywiste, czym jest gra. Zastanów się jednak nad tym przez chwilę — sprawa wcale nie jest taka prosta.

* Czy wszystkie gry mają **zwycięzcę**? Czy zawsze się kończą? Niekoniecznie. Jak to wygląda w symulatorze lotu, w grze w projektowanie parku rozrywki lub w grach takich jak The Sims?
* Czy gry zawsze są **ciekawe**? Nie dla każdego. Niektórzy gracze lubią „grind", czyli powtarzanie w kółko tych samych czynności. Inni nie cierpią takich gier.
* Czy w grach zawsze 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.

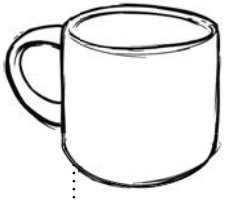
Dzielenie dużych projektów na mniejsze porcje

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.

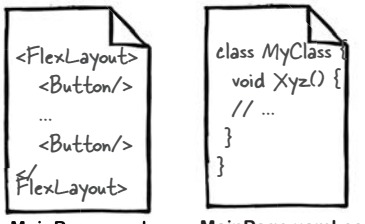
Nie martw się, jeśli nie pojmujesz niektórych zagadnień. Do czasu zakończenia lektury zrozumiesz wszystko, co dzieje się w grze. Na razie wystarczy postępować zgodnie z instrukcjami krok po kroku, aby uruchomić grę. Dzięki temu zdobędziesz solidne podstawy, które później pozwolą Ci rozszerzyć wiedzę.

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:

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 MainPage.xaml.cs

1. **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.**

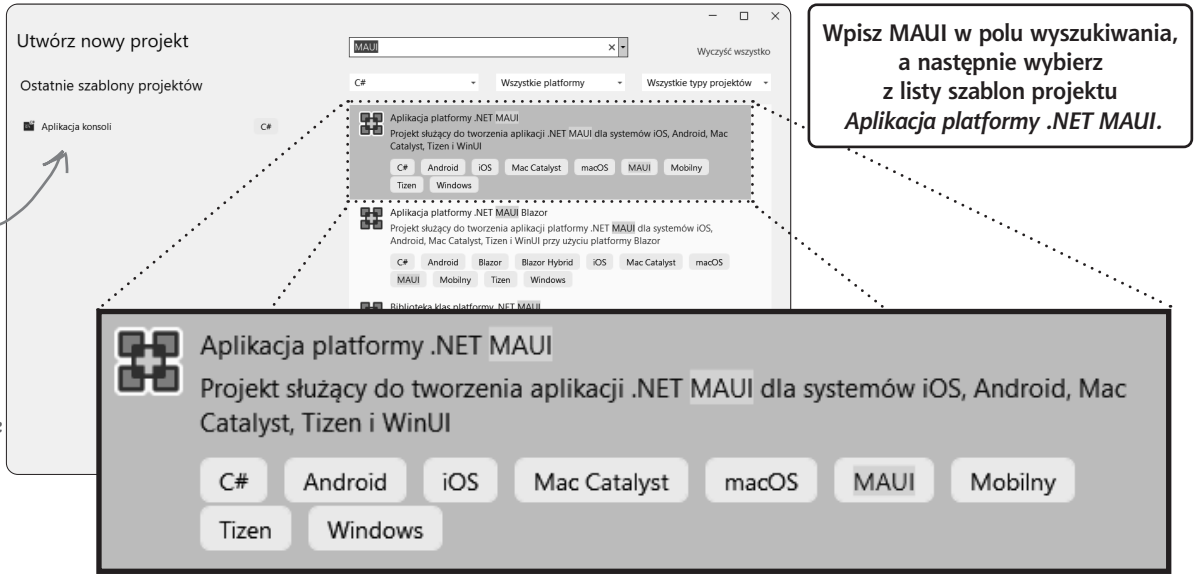
   Gra musi wykrywać, kiedy użytkownik klika pary emoji, śledzić pary i kończyć rozgrywkę, gdy gracz dopasuje wszystkie zwierzęta. Napiszesz potrzebny do tego kod.

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.

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*.

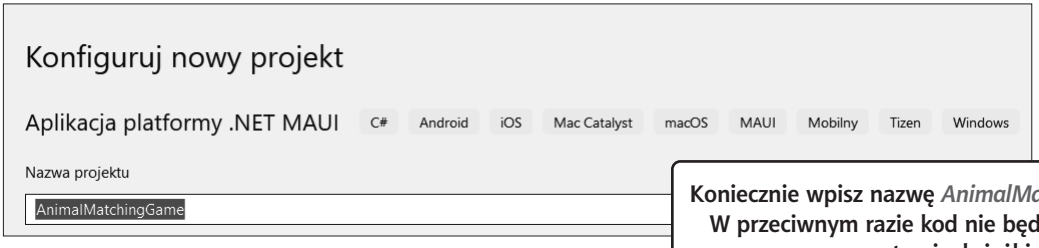


Na liście niedawno używanych szablonów zobaczysz pozycję Aplikacja konsoli, ponieważ wcześniej utworzyłeś taką właśnie aplikację.

Wpisz MAUI w polu wyszukiwania, a następnie wybierz z listy szablon projektu *Aplikacja platformy .NET MAUI*.

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*.



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

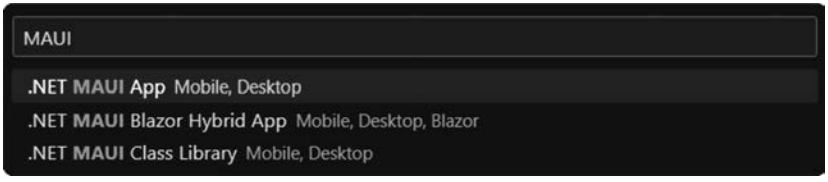
Na koniec Visual Studio poprosi o wybranie wersji platformy .NET. Wybierz najnowszą wersję, tak jak podczas tworzenia projektu aplikacji konsolowej. Następnie kliknij przycisk *Utwórz*, aby utworzyć nowy projekt platformy .NET MAUI.

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.

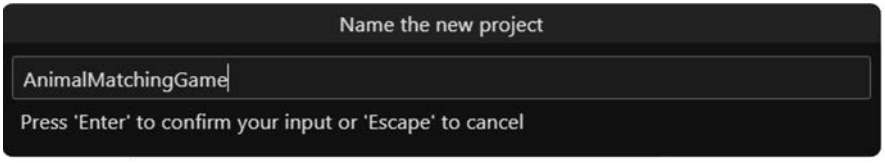
Następnie utwórz projekt aplikacji platformy .NET MAUI. Użyj kombinacji *Ctrl+Shift+P* lub ⇧+⌘+*P*, aby **otworzyć paletę poleceń**. Wybierz polecenie ***.NET: NEW Project***, aby utworzyć nowy projekt. VSCode wyświetli pytanie o typ projektu.

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.**



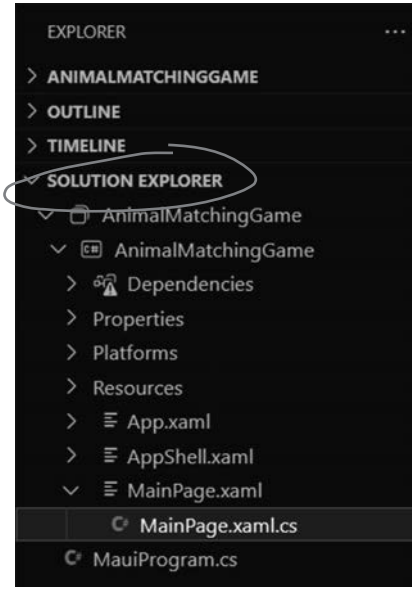
VSCode wyświetli monit o podanie katalogu. Wybierz **katalog domyślny**.

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.
4. Otwórz paletę poleceń (*Ctrl+Shift+P* lub ⇧+⌘+*P*) i wybierz opcję ***Debug: Start Debugging***. Możesz także **otworzyć plik *MainPage.xaml*** i **nacisnąć *F5*** lub wybrać opcję *Start Debugging* z menu *Run*.
5. VSCode może wyświetlić monit o wybranie debugera. Wybierz opcję ***.NET MAUI***. Gdy to zrobisz, aplikacja powinna zacząć działać w nowym oknie.

***Rozwiń* sekcję *Solution Explorer* i *zwiń* wszystkie pozostałe sekcje w oknie eksploratora.**



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

Uruchamianie nowej aplikacji .NET MAUI

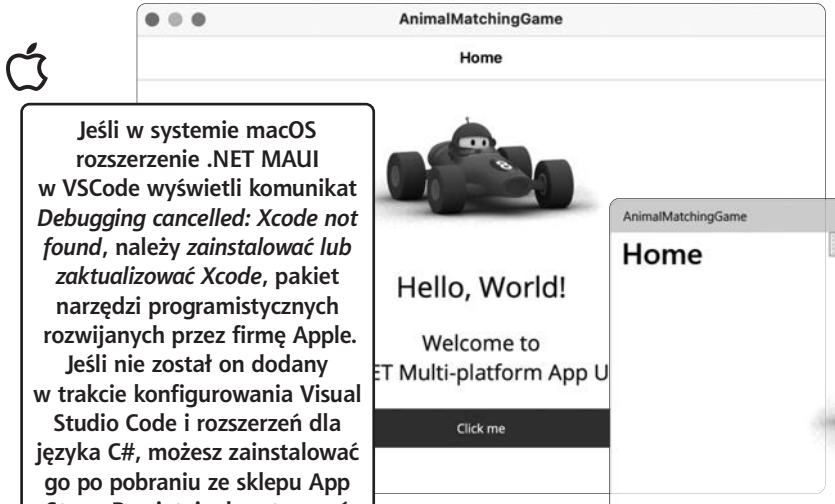
Zrób to!

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*.**

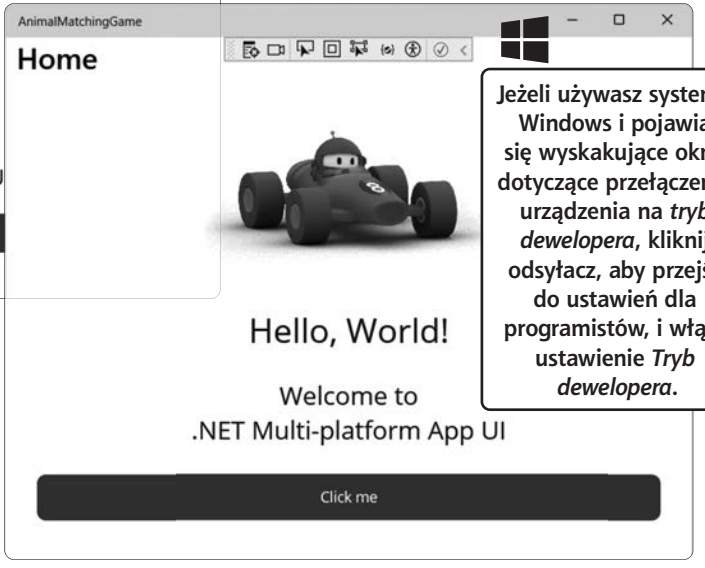
IDE ***skompiluje*** kod, co oznacza, że przekształci go w program wykonywalny, który może zostać uruchomiony przez system operacyjny. Następnie uruchomi aplikację:

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



Te zrzuty zostały wykonane z użyciem platformy .NET 8. Jeśli korzystasz z innej wersji .NET, domyślna aplikacja MAUI może wyglądać nieco inaczej.

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.



Jeżeli używasz systemu Windows i pojawia się wyskakujące okno dotyczące przełączenia urządzenia na *tryb dewelopera*, kliknij odsyłacz, aby przejść do ustawień dla programistów, i włącz ustawienie *Tryb dewelopera*.

Zatrzymywanie aplikacji MAUI

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

Aplikację można uruchomić lub zatrzymać w dowolnym momencie. Jeśli w kodzie C# lub XAML występują błędy 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*.

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:

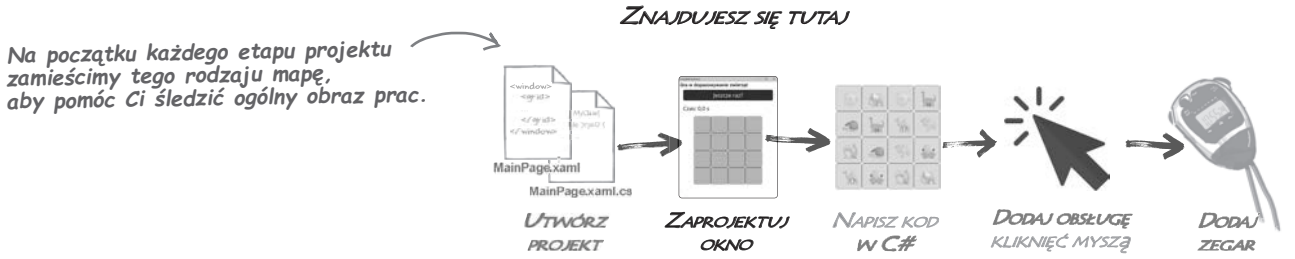
https://learn.microsoft.com/dotnet/maui/get-started/installation?tabs=visual-studio-code

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 ⇧+⌘+P, 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*.

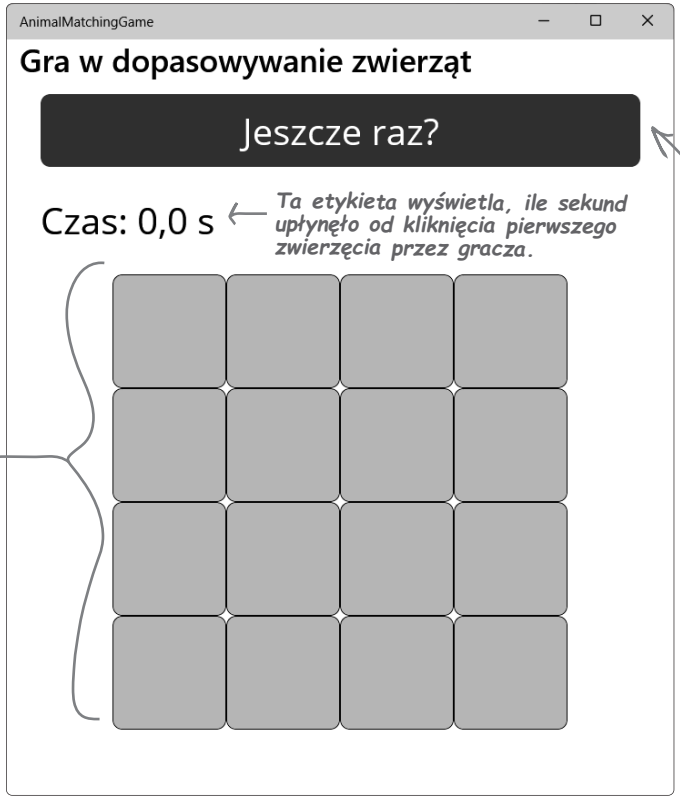


Oto strona, którą zbudujesz

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.



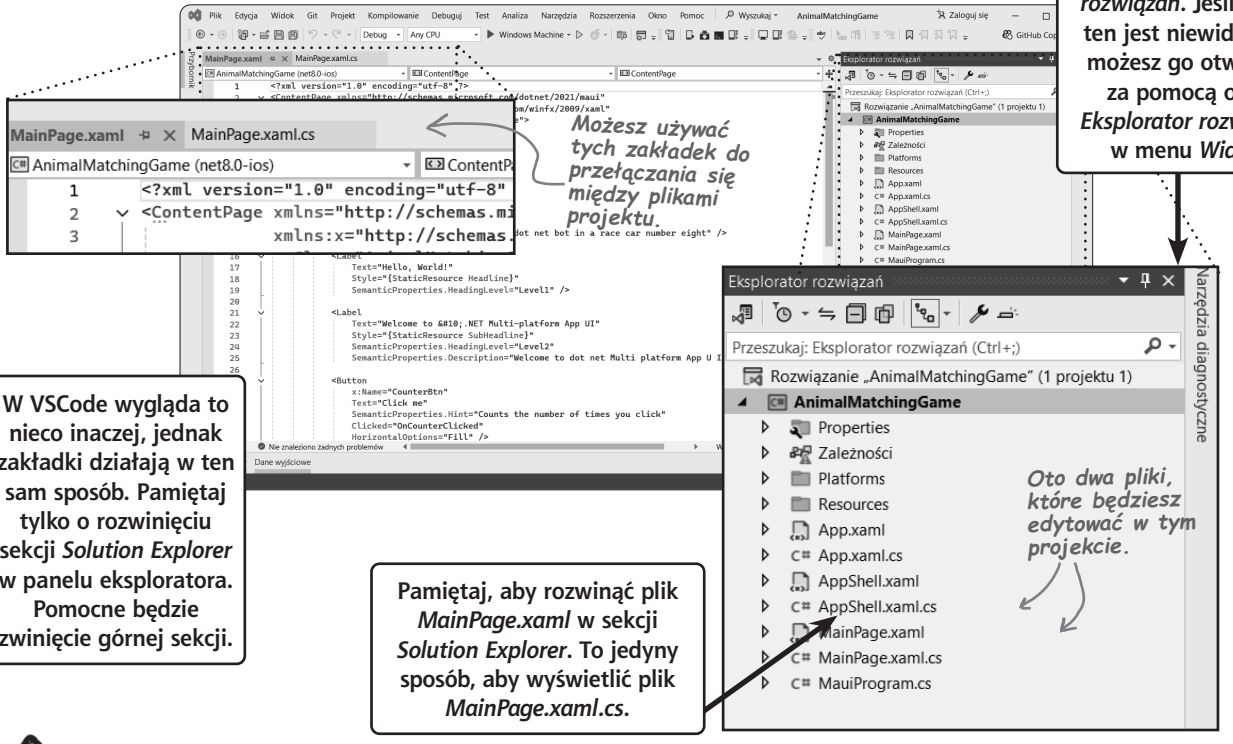
Na stronie znajduje się grupa kontrolek Button, które użytkownik może kliknąć. Później napiszesz kod C# do wyświetlania na nich w losowej kolejności emoji ze zwierzętami.

Ten przycisk wyświetla tekst „Jeszcze raz?" i umożliwia rozpoczęcie gry. Później napiszesz kod C#, aby ukrywać ten przycisk w trakcie gry.

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:

Jeśli zamkniesz którykolwiek z plików, możesz otworzyć je ponownie za pomocą dwukrotnego kliknięcia w *Eksploratorze rozwiązań*. Jeśli panel ten jest niewidoczny, możesz go otworzyć za pomocą opcji *Eksplorator rozwiązań* w menu *Widok*.



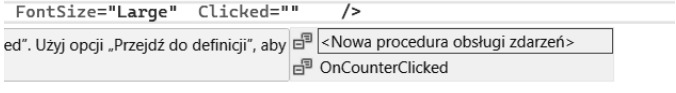
W VSCode wygląda to nieco inaczej, jednak zakładki działają w ten sam sposób. Pamiętaj tylko o rozwinięciu sekcji *Solution Explorer* w panelu eksploratora. Pomocne będzie zwinięcie górnej sekcji.

Pamiętaj, aby rozwinąć plik *MainPage.xaml* w sekcji *Solution Explorer*. To jedyny sposób, aby wyświetlić plik *MainPage.xaml.cs*.

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:*



Visual Studio pomaga powiązać zdarzenie z procedurą do jego 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

Pierwszą rzeczą, którą zrobimy, jest zaprojektowanie strony gry. Znajdzie się na niej szesnaście przycisków do wyświetlania emoji ze zwierzętami, 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 dodają do strony **kontrolkę `ScrollView`**. Sprawia ona, że jeśli aplikacja znajduje się w oknie mniejszym niż jego zawartość, wszystko pomiędzy znacznikami otwierającym i zamykającym tej kontrolki można przewijać w górę i w dół.

Znajdź otwierający znacznik `VerticalStackLayout`. Znajduje się on kilka wierszy dalej i wygląda następująco:

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

Następnie znajdź zamykający znacznik `VerticalStackLayout`:

```
</VerticalStackLayout>
```

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.

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">

        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

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

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:

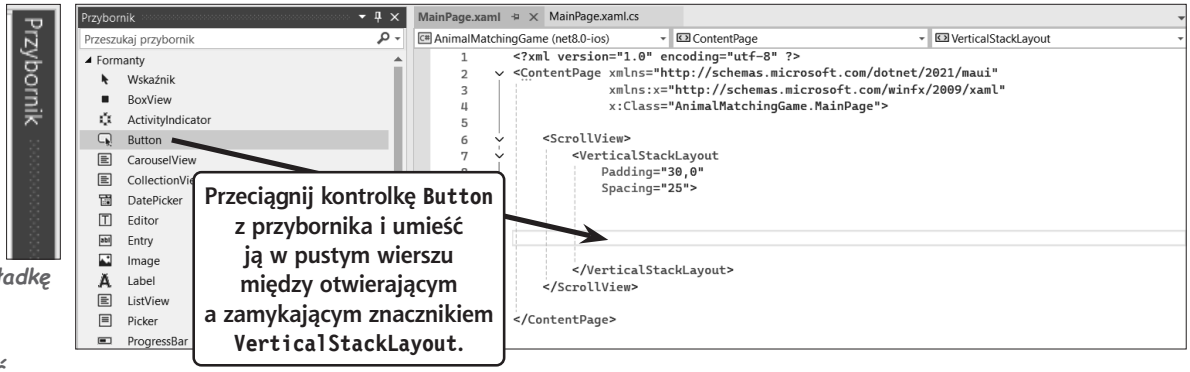
```
namespace AnimalMatchingGame
{
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}
}
```

Klasy i przestrzenie nazw omawiamy w rozdziale 3.

Koniecznie wróć do sekcji Solution Explorer i otwórz plik MainPage.xaml.cs, aby móc edytować kod C#. By ten plik stał się widoczny, być może trzeba będzie rozwinąć sekcję pliku MainPage.xaml.

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.



Kliknij 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.

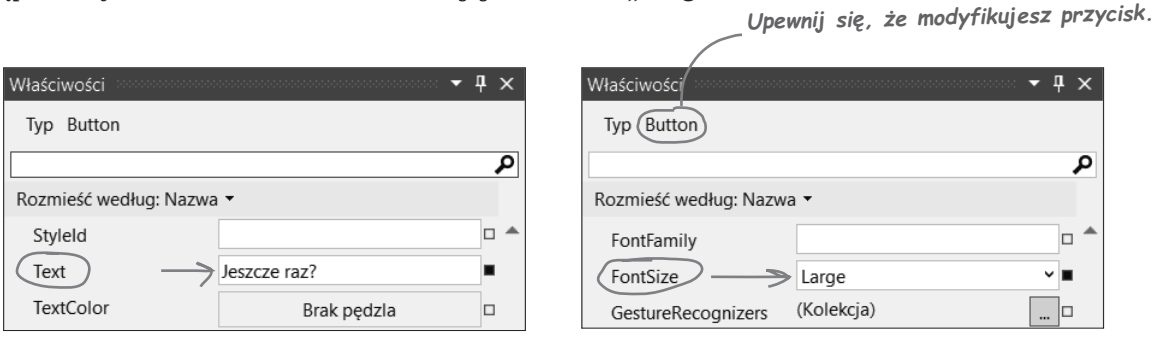
④ Dodaj właściwości do znacznika przycisku „Jeszcze raz?" w kodzie XAML.

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"**.



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.

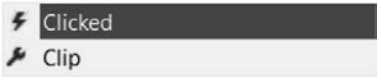
Użyj edytora, aby **dodać właściwość `x:Name`** i nadać przyciskowi nazwę. Kod powinien wyglądać następująco:

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

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

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" Cli />
```



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

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

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

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?").

```
<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!

Ćwiczenie

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 `Label`, 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 ... />`.

Ćwiczenie

Rozwiązanie

Dodaj kontrolkę Label do strony XAML.

Jeśli kroki opisane w ćwiczeniu zostały wykonane poprawnie, kod XAML w pliku *MainPage.xaml* powinien 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">

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

            <Label x:Name="TimeElapsed" Text="Czas: 0,0 s"
                   FontSize="Large" />

        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

Podział wiersza może się znajdować *między* właściwościami w znaczniku. Ponadto możesz zapisać właściwości w innej kolejności.

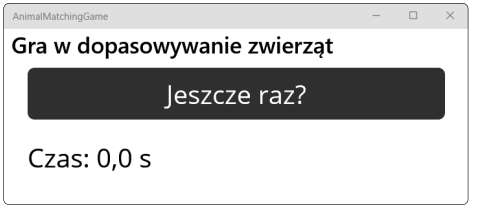
Oto etykieta dodana w ćwiczeniu. Umieściliśmy podział wiersza między właściwościami Text i FontSize, ale nie jest to konieczne.

Kod C# w pliku *MainPage.xaml.cs* nie został zmodyfikowany w ramach ćwiczenia, więc nadal powinien wyglądać tak jak poniżej. Jeśli używasz Visual Studio Code, należy starannie ręcznie przepisać metodę `PlayAgainButton_Clicked`:

```
namespace AnimalMatchingGame;

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void PlayAgainButton_Clicked(object sender, EventArgs e)
    {

    }
}
```

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

Jeśli teraz uruchomisz aplikację, powinna wyglądać tak.



Naciśnij F5, aby ponownie uruchomić aplikację. W VSCode trzeba przejść z powrotem do kodu XAML w pliku MainPage.xaml, a następnie nacisnąć ten klawisz.

Nie ma 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.

P: A więc przyciski i etykiety są kontrolkami?

O: 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 `VerticalStackLayout` w omawianej aplikacji obejmuje przycisk i etykietę. Jak to działa?

O: Kontrolka 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!

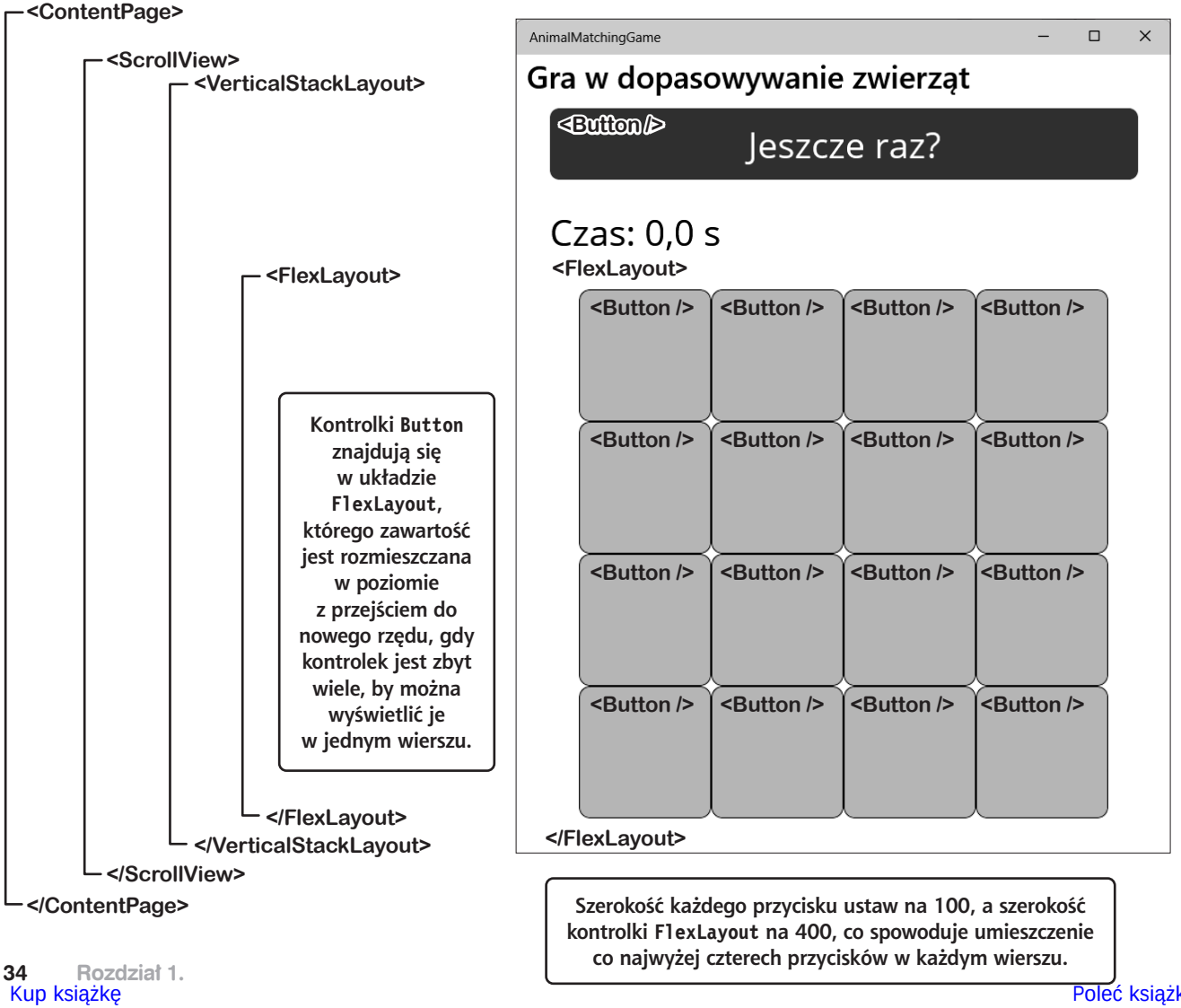
WYSIL SZARE KOMÓRKI

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.



Ćwiczenie

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

1. 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

1. 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`.
3. 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`.

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”.

Dodaj pozostałe przyciski

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.

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" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AnimalMatchingGame.MainPage">
```

Jeśli wybierzesz dla projektu inną nazwę, zobaczysz ją w tym miejscu.

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

Wygląda to na dużą ilość kodu XAML, ale jego większość stanowi 16 identycznych znaczników Button, które zostały skopiowane i wklejone.

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

    <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"
```

Upewnij się, że znacznik otwierający FlexLayout znajduje się tuż pod etykietą, a właściwości są zgodne z podanymi. Pamiętaj, aby ustawić szerokość maksymalną zamiast minimalnej. W przeciwnym razie przyciski nie utworzą siatki 4×4.

Wygląda to na dużą ilość kodu XAML, ale jego większość zajmuje ten sam znacznik `<Button … />`, który został *skopiowany i wklejony 16 razy.*

Szesnaście znaczników Button powinno wyglądać identycznie. Właściwości mogą być podane w innej kolejności.

Ćwiczenie
Rozwiązanie

```
  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"/>
</FlexLayout>

    </VerticalStackLayout>
  </ScrollView>

</ContentPage>
```

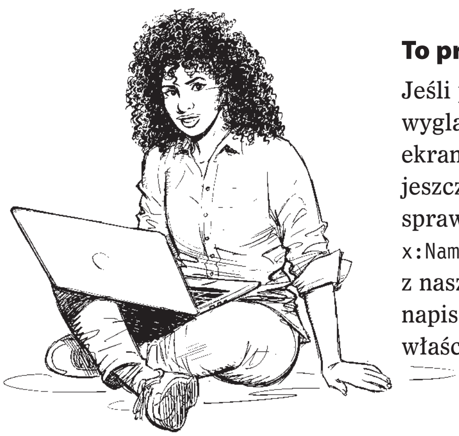
Upewnij się, że każdy przycisk ma właściwość `Clicked="Button_Clicked"`. Jeśli procedura obsługi zdarzeń `Clicked` ma inną nazwę, Twój kod C# nie będzie zgodny z naszym. Możesz usunąć właściwość `Clicked` ze wszystkich przycisków, a następnie *dodać ją ponownie z poprawną nazwą*. Po dodaniu procedury obsługi zdarzeń pojawi się ona na liście rozwijanej w momencie modyfikowania innych przycisków.

Oto procedura obsługi zdarzeń `Button_Clicked` w pliku *MainPage.xaml.cs*:

```
private void Button_Clicked(object sender, EventArgs e)
{

}
```

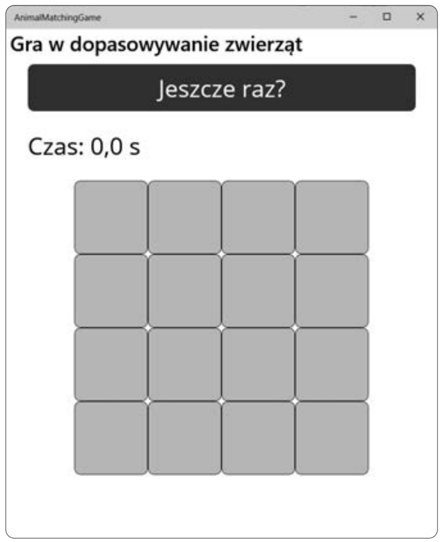
Jeśli korzystasz z VSCode, musisz otworzyć plik MainPage.xaml.cs i ręcznie dodać tę procedurę obsługi zdarzeń. Upewnij się, że kod jest identyczny!

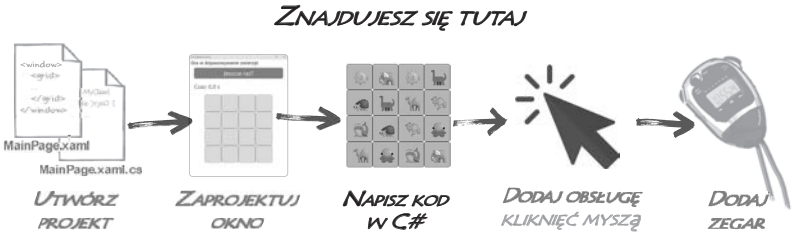
Aby sprawdzić, czy rozwiązanie jest prawidłowe, mogę **porównać je ze zrzutem ekranu**. To ułatwia sprawę!



To prawda!

Jeśli po uruchomieniu aplikacja wygląda tak jak na poniższym zrzucie ekranu, wszystko działa poprawnie. Jest jeszcze tylko jedna rzecz, którą musisz sprawdzić: upewnij się, że właściwości `x:Name` ***dokładnie*** odpowiadają tym z naszego rozwiązania. Kod C#, który napiszesz, będzie korzystał z tych właściwości.





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:

```
Sprawiał, że przyciski ze zwierzętami staną się widoczne
                    ↓
Sprawiał, że przycisk „Jeszcze raz?” zostanie ukryty
                    ↓
Tworzył listę 16 par emoji ze zwierzętami
                    ↓
Dla każdego z 16 przycisków:
      Wybierał losowe zwierzę z listy
                    ↓
      Dodawał to zwierzę do przycisku
                    ↓
      Usuwał zwierzę z listy
                    ↓
      Powtarzał te operacje do czasu
      przetworzenia wszystkich przycisków
```

Rozpocznij edycję procedury obsługi zdarzeń PlayAgainButton

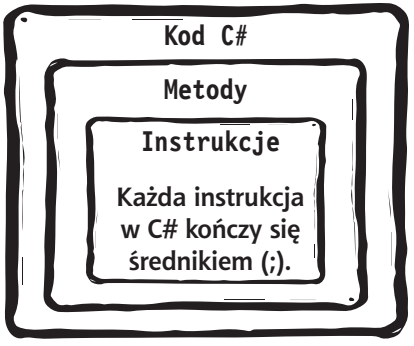
Podczas pisania kodu XAML dla przycisku „Jeszcze raz?” dodano obsługę zdarzeń:

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

<Nowa procedura obsługi 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?”.

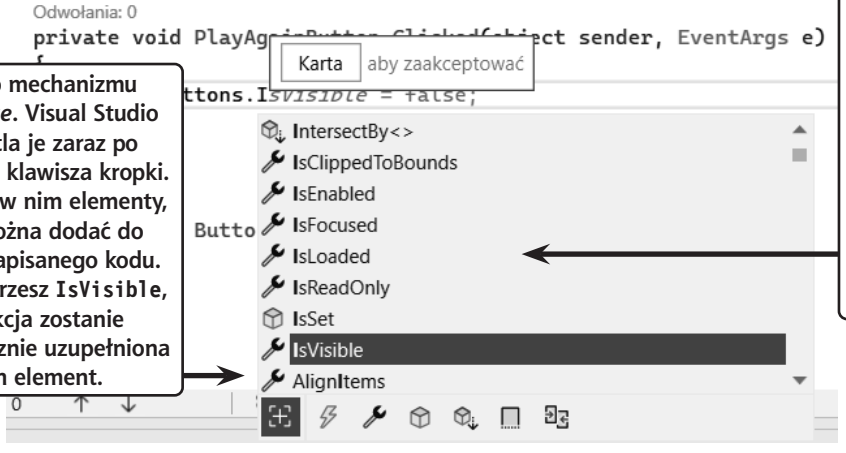
Dodaj instrukcję C# do metody obsługi zdarzenia

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

Zrób to!

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

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



To okno mechanizmu *IntelliSense*. Visual Studio wyświetla je zaraz po naciśnięciu klawisza kropki. Zobaczysz w nim elementy, które można dodać do właśnie napisanego kodu. Jeśli wybierzesz `IsVisible`, instrukcja zostanie automatycznie uzupełniona o ten element.

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.

Dodaj kolejne instrukcje do metody obsługi zdarzenia

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

Zrób to!

1 Dodaj instrukcję ukrywającą przycisk „Jeszcze raz?”.

Pamiętasz, jak użyliśmy właściwości `x:Name` w kodzie XAML, aby nadać nazwy przyciskowi „Jeszcze raz?” i układowi `FlexLayout`, który zawiera szesnaście przycisków ze zwierzętami? Wróć do tego kodu XAML. Układowi `FlexLayout` nadaliśmy nazwę „AnimalButtons”, a przed chwilą 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;
    PlayAgainButton.IsVisible = false;
}
```

Wpisz ten wiersz kodu bezpośrednio przed dodanym wcześniej poleceniem.

Kod, który już znajduje się w plikach, wyróżniamy jaśniejszym kolorem, aby łatwiej było zauważyć, co należy dodać.

Ta instrukcja ukrywa przycisk „Jeszcze raz?”.

2 Ukryj przyciski ze zwierzętami w momencie uruchamiania aplikacji.

Przyjrzyj się bliżej pierwszej instrukcji dodanej do metody obsługi zdarzenia. Wyświetla ona układ `FlexLayout` zawierający przyciski ze zwierzętami. Ale chwileczkę — 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"
            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 kliknięciu 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#.

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

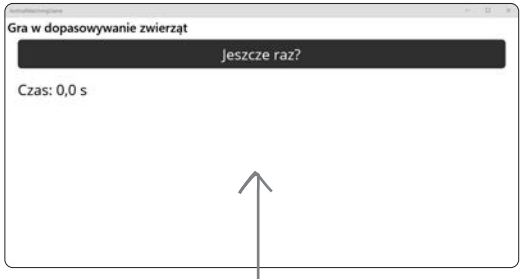
Kiedy piszesz kod, nie tworzysz kompletnej aplikacji od początku do końca, aby dopiero potem uruchomić ją w celu sprawdzenia, czy działa. Wygląda to zupełnie inaczej! ***Pisanie kodu to proces twórczy.*** Poszczególne zadania można wykonać w kodzie na różne sposoby, a w wielu przypadkach jedyną metodą 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).

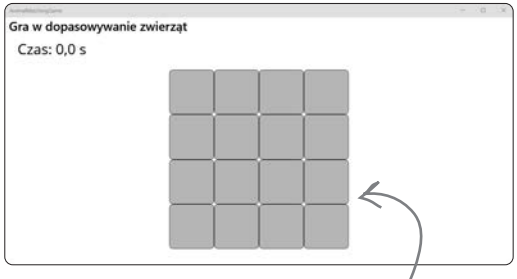
Co to wszystko oznacza?

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).



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.

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. `AnimalButtons` *to coś innego niż* `animalButtons`*. 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.

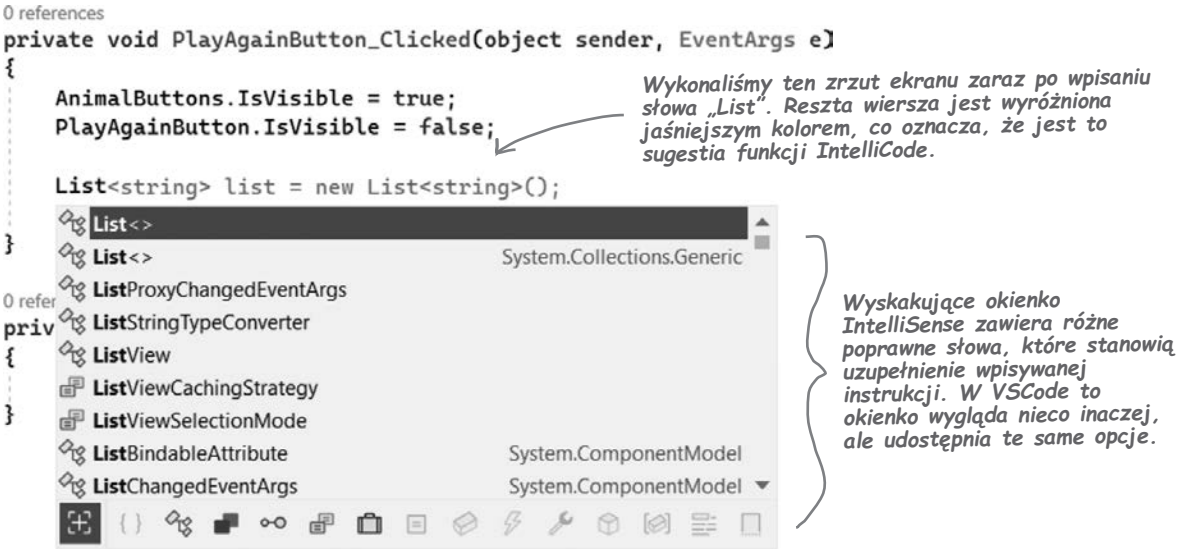
1 Zacznij tworzyć listę emoji ze zwierzętami.

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:

```
private void PlayAgainButton_Clicked(object sender, EventArgs e)
{
    AnimalButtons.IsVisible = true;
    PlayAgainButton.IsVisible = false;

    List<string> animalEmoji = [

    ];
```

Pamiętaj, aby dodać średnik po zamykającym nawiasie].

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.

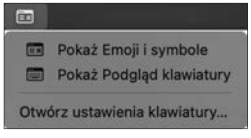
Lista powinna wyglądać teraz tak:

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

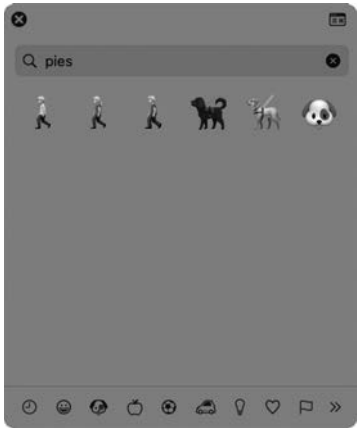
Jak wprowadzać emoji?

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.

Jeśli korzystasz z komputera Mac, użyj panelu **Character Viewer**. W tym celu wciśnij kombinację klawiszy *Ctrl*+⌘+spacja. Użyj pola wyszukiwania, aby znaleźć określone zwierzę. Gdy natrafisz na emoji, które chcesz dodać, kliknij je, aby wprowadzić je tak jak za pomocą wpisywania.



Przeglądarkę znaków systemu macOS można również otworzyć za pomocą menu Input z paska menu. Jeśli nie widzisz menu Input, otwórz ustawienia systemowe i wyszukaj „input menu". Dostępna jest tam opcja, którą możesz włączyć, aby wyświetlić menu Input na pasku menu.



3 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:

Lista to kolekcja przechowująca zestaw wartości w określonej kolejności. Więcej o kolekcjach dowiesz się z rozdziałów 8. i 9.

Gdy używasz nawiasów [i] do utworzenia listy, stosujesz wyrażenia kolekcji. Każde zwierzę w cudzysłowie jest oddzielną wartością tekstową, a poszczególne wartości są oddzielone przecinkami. Więcej o wyrażeniach dowiesz się w rozdziale 8.

```
List<string> animalEmoji = [
    "🐡", "🐡",
    "🐘", "🐘",
    "🐳", "🐳",
    "🐫", "🐫",
    "🦕", "🦕",
    "🦘", "🦘",
    "🦃", "🦃",
    "🦔", "🦔",
];
```

Zwracaj baczną uwagę na cudzysłowy i przecinki. Jeśli pominiesz jeden z nich, kod się nie skompiluje.

Upewnij się, że dodajesz ten kod wewnątrz nawiasów { i } po metodzie PlayAgainButton_Clicked, a nie po metodzie Button_Clicked.

Więcej o pętlach dowiesz się z następnego rozdziału.

4 Ukończ metodę.

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.

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.

```
namespace AnimalMatchingGame;

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void PlayAgainButton_Clicked(object sender, EventArgs e)
    {
        AnimalButtons.IsVisible = true;
        PlayAgainButton.IsVisible = false;

        List<string> animalEmoji = [
            "🐙", "🐙",
            "🐵", "🐵",
            "🐘", "🐘",
            "🐰", "🐰",
            "🐈", "🐈",
            "🐓", "🐓",
            "🐃", "🐃",
            "🦔", "🦔",
        ];

        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);
        }
    }

    private void Button_Clicked(object sender, EventArgs e)
    {

    }
}
```

Jeśli Twój projekt ma inną nazwę, pojawi się ona w tym wierszu.

Ten wiersz sprawia, że przyciski ze zwierzętami są niewidoczne po pierwszym uruchomieniu aplikacji.

Upewnij się, że kod zawiera dokładnie osiem par pasujących emoji. Jest to jeden z elementów sprawiających, że gra działa.

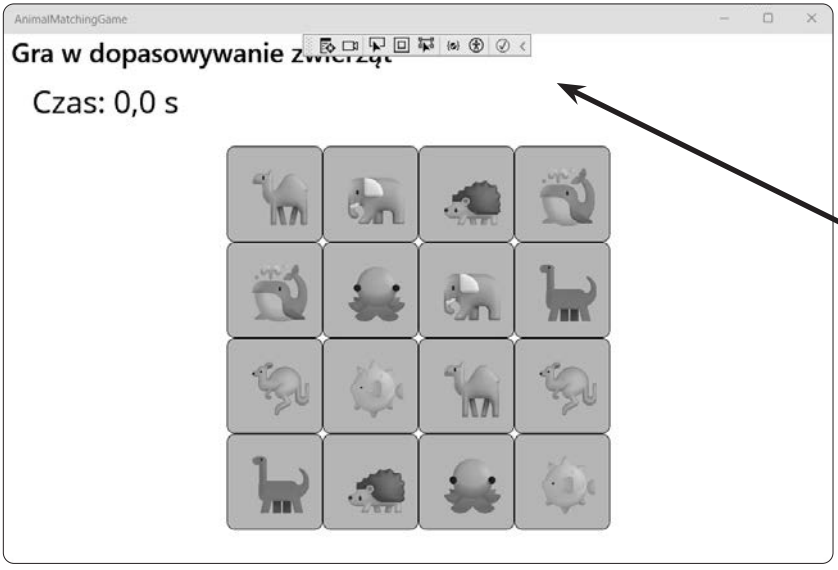
Visual Studio *automatycznie doda wcięcia* w Twoim kodzie, aby wyglądał jak nasz. Jeśli używasz VSCode, naciśnij *Alt+Shift+F* lub ⌥*+Shift+F*, by automatycznie poprawić wcięcia w pliku.

Właśnie wprowadziliśmy ten kod, aby dodać emoji do przycisków.

Visual Studio wygenerowało tę pustą metodę obsługi zdarzenia `Button_Clicked` po dodaniu właściwości `Clicked` 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.

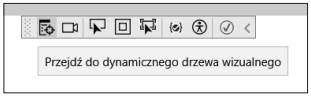
Uruchom aplikację!

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



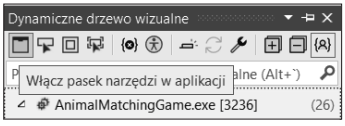
Kilka razy zatrzymaj grę i uruchom ją ponownie. Za każdym razem, gdy klikniesz przycisk „Jeszcze raz?”, zwierzęta powinny zostać rozmieszczone w inny sposób.

Jeśli korzystasz z Visual Studio, możesz zobaczyć *pasek narzędzi aplikacji* w górnej części okna:



Na zrzutach ekranu ukryliśmy ten pasek narzędzi. Możesz pozostawić go na widoku lub użyć widocznej po prawej stronie strzałki, aby go zwinąć.

Jeśli chcesz, możesz go również wyłączyć (ale nie musisz tego robić!). Kliknij pierwszy przycisk w narzędziach uruchomieniowych, aby wyświetlić panel dynamicznego drzewa wizualnego w IDE:



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

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.

To ćwiczenie z użyciem ołówka i kartki. W książce zamieściliśmy wiele gier i łamigłówek tego typu. Należy wykonać je wszystkie, ponieważ istnieją dowody z neuronauki na to, że zapisywanie jest skutecznym sposobem na szybsze przyswajanie ważnych pojęć.

KTO CO ROBI?

Instrukcja w języku C#

```
List<string> animalEmoji = [
    "🐵", "🐵",
    "🐯", "🐯",
    "🐘", "🐘",
    "🐒", "🐒",
    "🐄", "🐄",
    "🦙", "🦙",
    "🦃", "🦃",
    "🦔", "🦔",
];
```

```
foreach (var button in AnimalButtons.Children.OfType<Button>())
{
```

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

```
button.Text = nextEmoji;
```

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

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

```
int index = Random.Shared.Next(animalEmoji.Count);
```

```
PlayAgainButton.IsVisible = false;
```

Co robi?

Sprawia, że przycisk wyświetla wybrane emoji.

Znajduje wszystkie przyciski w układzie `FlexLayout` i powtarza dla nich instrukcje podane między nawiasami klamrowymi `{ }`.

Ponownie ukrywa przycisk „Jeszcze raz?”.

Tworzy listę ośmiu par emoji.

Wyświetla układ `FlexLayout` z przyciskami z emoji.

Wybiera losową wartość między 0 a liczbą emoji pozostałych na liście i nazywa ją `index`.

Usuwa wybrane emoji z listy.

Używa liczby losowej o nazwie `index` do pobrania losowego emoji z listy.

KTO CO ROBI?

Instrukcja w języku C#

```
List<string> animalEmoji = [
    "🐶", "🐶",
    "🐵", "🐵",
    "🐘", "🐘",
    "🐏", "🐏",
    "🐫", "🐫",
    "🦃", "🦃",
    "🦔", "🦔",
];

foreach (var button in AnimalButtons.Children.OfType<Button>())
{

animalEmoji.RemoveAt(index);

button.Text = nextEmoji;

string nextEmoji = animalEmoji[index];

AnimalButtons.IsVisible = true;

int index = Random.Shared.Next(animalEmoji.Count);

PlayAgainButton.IsVisible = false;
```

Co robi?

Sprawia, że przycisk wyświetla wybrane emoji.

Znajduje wszystkie przyciski w układzie `FlexLayout` i powtarza dla nich instrukcje podane między nawiasami klamrowymi { }.

Ponownie ukrywa przycisk „Jeszcze raz?”.

Tworzy listę ośmiu par emoji.

Wyświetla układ `FlexLayout` z przyciskami z emoji.

Wybiera losową wartość między 0 a liczbą emoji pozostałych na liście i nazywa ją `index`.

Usuwa wybrane emoji z listy.

Używa liczby losowej o nazwie `index` do pobrania losowego emoji z listy.

Oto następne ćwiczenie z ołówkiem i kartką. Poświęć kilka minut na jego wykonanie!

MINI Zaostrz ołówek

Oto ćwiczenie z ołówkiem i kartką, które pomoże Ci **lepiej zrozumieć** kod C#.

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

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 narzędzi.

CELNE SPOSTRZEŻENIA

- Visual Studio jest **IDE** Microsoftu (*zintegrowanym środowiskiem programistycznym*), które upraszcza i wspomaga edycję plików z kodem C# i zarządzanie nimi.
- **Aplikacje konsolowe** 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.

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!

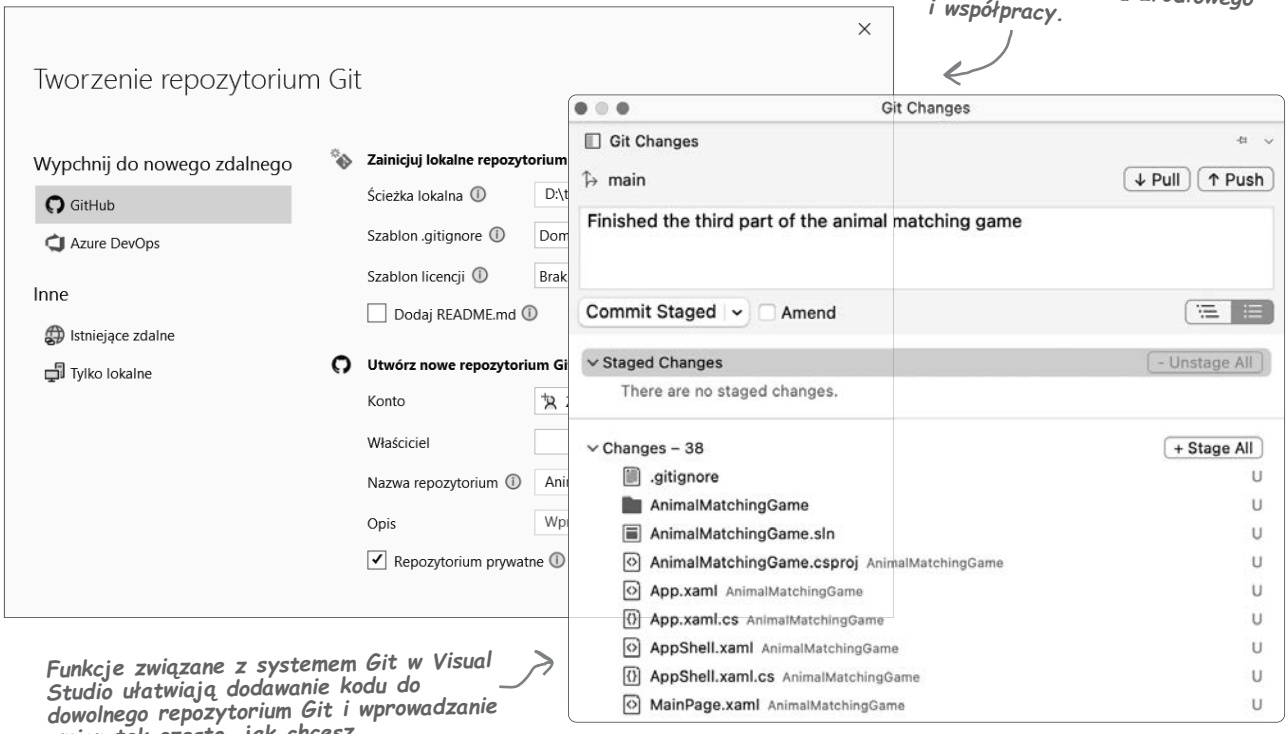
Oto kilka rzeczy, które Git może dla Ciebie zrobić

- ★ 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.

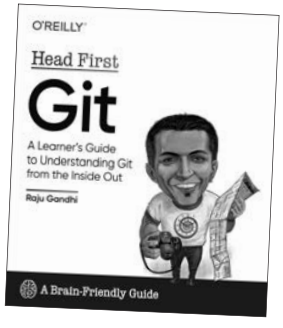


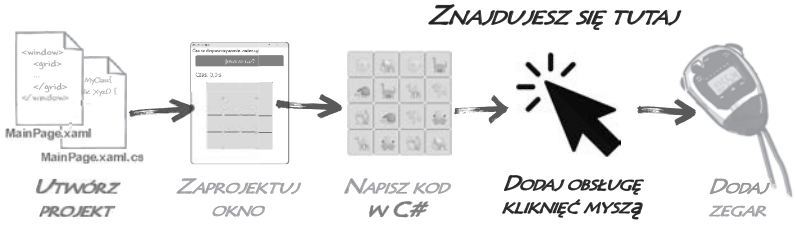
Funkcje związane z systemem Git w Visual Studio ułatwiają dodawanie kodu do dowolnego repozytorium Git i wprowadzanie zmian tak często, jak chcesz.

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*.

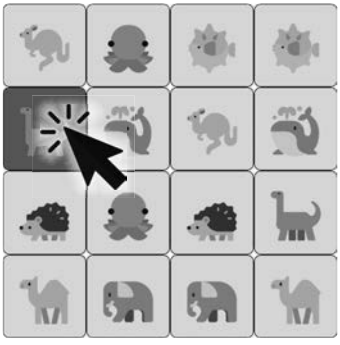
Omawiamy wszystko, czego potrzebujesz do używania Visual Studio do zapisywania i udostępniania projektów. Jednak system Git daje znacznie większe możliwości, zwłaszcza jeśli pracujesz w dużym zespole! Jeżeli system Git Cię zainteresował i chcesz się lepiej z nim zapoznać, przeczytaj książkę *Head First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





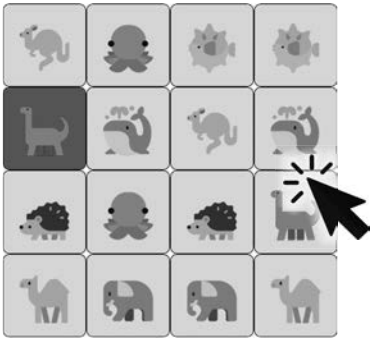
Dodaj kod C# do obsługi kliknięć myszą

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



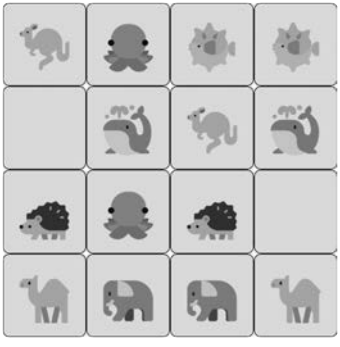
Gracz klika pierwszy przycisk.

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 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.

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

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;
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))
    {
      buttonClicked.BackgroundColor = Colors.Red;
      lastClicked = buttonClicked;
      findingMatch = true;
    }
    else
    {
      if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text))
      {
        matchesFound++;
        lastClicked.Text = " ";
        buttonClicked.Text = " ";
      }
      lastClicked.BackgroundColor = Colors.LightBlue;
      buttonClicked.BackgroundColor = Colors.LightBlue;
      findingMatch = false;
    }
  }

  if (matchesFound == 8)
  {
    matchesFound = 0;
    AnimalButtons.IsVisible = false;
    PlayAgainButton.IsVisible = true;
  }
}
```

To cudzysłów, po którym następuje spacja i kolejny cudzysłów.

1. Co robi matchesFound?

2. Co robią te trzy wiersze kodu?

3. Co robi ten blok kodu?

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 będą one w pełni poprawne!*** Chodzi o to, aby mózg zaczął traktować kod C# jako coś, co można przeczytać i zrozumieć.

```
Button lastClicked;
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))
    {
      buttonClicked.BackgroundColor = Colors.Red;
      lastClicked = buttonClicked;
      findingMatch = true;
    }
    else
    {
      if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text))
      {
        matchesFound++;
        lastClicked.Text = " ";
        buttonClicked.Text = " ";
      }
      lastClicked.BackgroundColor = Colors.LightBlue;
      buttonClicked.BackgroundColor = Colors.LightBlue;
      findingMatch = false;
    }
  }

  if (matchesFound == 8)
  {
    matchesFound = 0;
    AnimalButtons.IsVisible = false;
    PlayAgainButton.IsVisible = true;
  }
}
```

1. Co robi matchesFound?

Rejestruje liczbę par zwierząt znalezionych przez gracza, dzięki czemu grę można zakończyć po wskazaniu wszystkich 8 par.

2. Co robią te trzy wiersze kodu?

Są one uruchamiane, gdy gracz kliknie pierwszy przycisk potencjalnej pary. Ten kod zapisuje przycisk i zmienia jego kolor na czerwony.

3. Co robi ten blok kodu?

Jest on uruchamiany, gdy gracz kliknie drugi przycisk w parze. Jeśli zwierzęta pasują do siebie, kod dodaje jeden do matchesFound i ukrywa zwierzęta na obu przyciskach. Resetuje również kolor pierwszego przycisku i przygotowuje grę tak, aby gracz mógł ponownie kliknąć pierwszy przycisk w parze.

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.

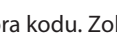
Musisz zachować staranność podczas wprowadzania kodu, ponieważ jeśli otwierające nawiasy zwykłe lub klamrowe nie mają zamykających odpowiedników lub jeżeli pominiesz średnik na końcu instrukcji, kod się nie skompiluje. Na szczęście Visual Studio ma wiele funkcji pomagających pisać kod, który się kompiluje:

- ★ 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.

Wskazówka dotycząca IDE: lista błędów

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***.

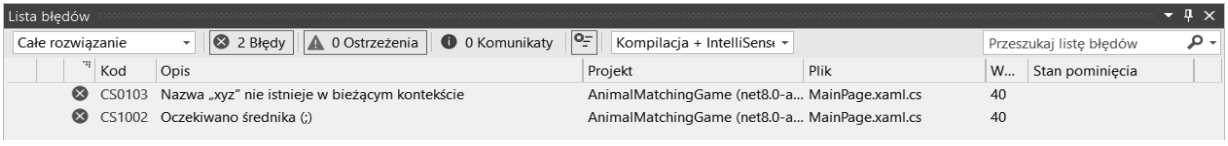
Przejdź do pierwszego wiersza kodu w metodzie `Button_Clicked`. Naciśnij Enter dwa razy, a następnie dodaj w osobnym wierszu litery `xyz`.

Przyjrzyj się dolnej części edytora kodu. Zobaczysz ikonę, która wygląda tak: 

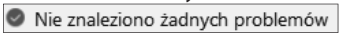
lub tak: 

. Jeśli nie widzisz ikony, wybierz opcję Kompiluj rozwiązanie z menu Kompilowanie, aby nakazać Visual Studio próbę kompilacji kodu.

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):



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.

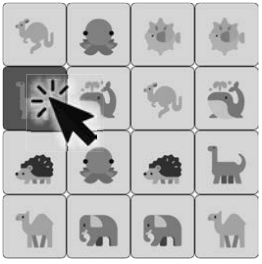
Jeśli w kodzie nie ma innych usterek, lista błędów powinna być pusta, a w dolnej części okna Visual Studio pojawi się ikona: 

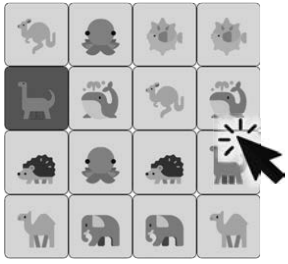
lub: 

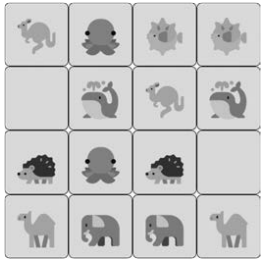
. Informuje ona o udanej kompilacji aplikacji.

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.

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

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**.

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".

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.

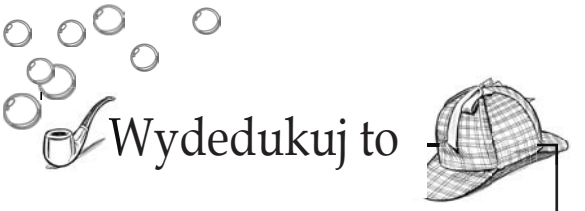
Wiadomo, że cały kod do obsługi kliknięć myszą znajduje się we właśnie dodanej metodzie obsługi zdarzeń `Button_Clicked`. Wróćmy więc do kodu i sprawdźmy, czy uda się znaleźć wskazówki dotyczące tego, co poszło nie tak.

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 kliknięty przez gracza pasują do siebie. Jeśli tak jest, dodaje jeden do `matchesFound` i ukrywa oba przyciski.
- Jeśli `matchesFound` jest równe 8, oznacza to, że gracz znalazł wszystkie 8 par zwierząt. Na końcu metody obsługi zdarzeń znajduje się kod sprawdzający, czy `matchesFound` 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?



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 `matchesFound` 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.

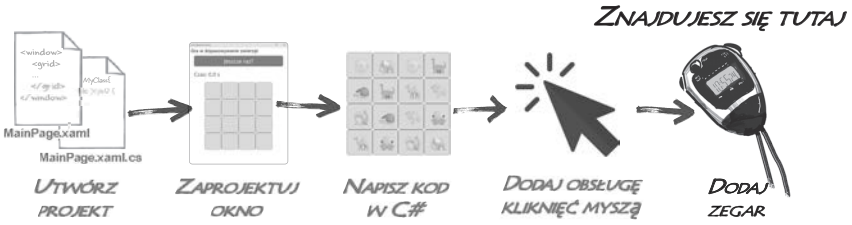
Umieść kursor między dwoma ostatnimi nawiasami zamykającymi `))` w instrukcji `if` i naciśnij *Enter*, aby dodać wiersz. Następnie wprowadź następujący kod: `&& (!String.IsNullOrWhiteSpace(buttonClicked.Text))`.

Kod powinien wyglądać teraz tak:

```
if ((buttonClicked != lastClicked) && (buttonClicked.Text == lastClicked.Text)
    && (!String.IsNullOrWhiteSpace(buttonClicked.Text)))
{
    matchesFound++;
    lastClicked.Text = " ";
    buttonClicked.Text = " ";
}
```

Dodanie tego fragmentu do instrukcji if spowoduje, że kod przed dodaniem 1 do matchesFound sprawdzi, czy kliknięty przez gracza przycisk nie jest pusty.

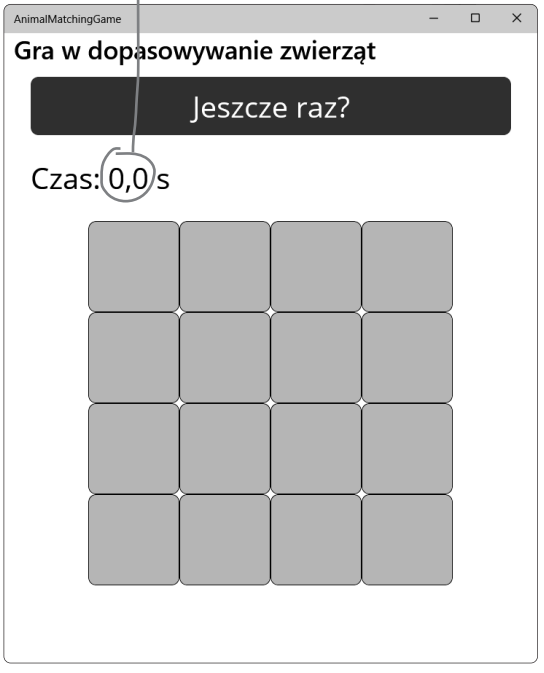
Po modyfikacji instrukcji `if` ponownie uruchom aplikację. Błąd powinien zostać naprawiony.



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



Zegar „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.

Dodaj zegar do kodu 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.

① Dodaj wiersz kodu na końcu metody obsługi zdarzenia `PlayAgainButton_Clicked`, aby uruchamiać licznik czasu.

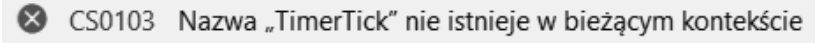
Przejdź na sam koniec metody obsługi zdarzeń `PlayAgainButton_Clicked`. Znajdują się tam dwa zamykające nawiasy klamrowe } w oddzielnych wierszach. Dodaj trzy wiersze między tymi nawiasami, a następnie wpisz w tym miejscu poniższy wiersz kodu:

```
        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);
    }
```

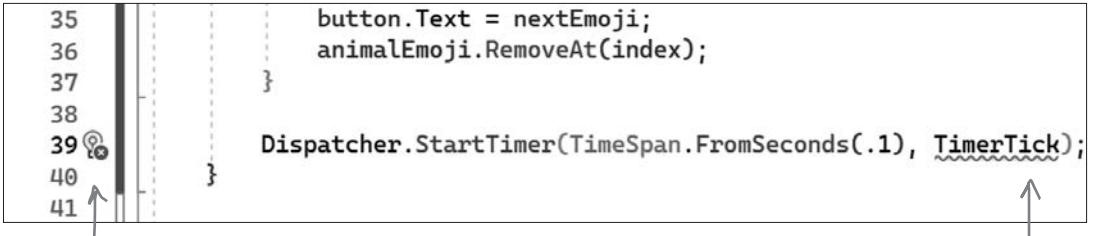
Dodany wiersz kodu powoduje, że aplikacja **uruchamia zegar**, który co 0,1 sekundy wykonuje metodę o nazwie `TimerTick`.

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:



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:

W VSCode ikona szybkich akcji wygląda jak żarówka, a nie jak śrubokręt.

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):

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.

Zrób to!

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`:

```
int tenthsOfSecondsElapsed = 0;

private bool TimerTick()
```

To jest pole. Więcej na temat działania pól dowiesz się w rozdziale 3.

Dokończ metodę TimerTick

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

```
private bool TimerTick()
{
    if (!this.IsLoaded) return false;

    tenthsOfSecondsElapsed++;
    TimeElapsed.Text = "Czas: " +
        (tenthsOfSecondsElapsed / 10F).ToString("0,0 s");

    if (PlayAgainButton.IsVisible)
    {
        tenthsOfSecondsElapsed = 0;
        return false;
    }

    return true;
}
```

W tej instrukcji umieściliśmy dodatkowy znak podziału wiersza, aby kod zmieścił się na stronie w książce, ale jeśli chcesz, możesz wpisać wszystko w jednym wierszu. Upewnij się, że nawiasy pasują do siebie.

Uruchom grę. Teraz licznik czasu działa!

Metoda TimerTick pod lupą

Przyjrzyjmy się bliżej metodzie `TimerTick`, aby zobaczyć, jak, no cóż, tyka. Obejmuje ona w sumie siedem instrukcji, a każda z nich jest ważna.

```
private bool TimerTick()
{
    if (!this.IsLoaded) return false;

    tenthsOfSecondsElapsed++;

    TimeElapsed.Text = "Czas: " +
       (tenthsOfSecondsElapsed / 10F).ToString("0,0 s");

    if (PlayAgainButton.IsVisible)
    {
       tenthsOfSecondsElapsed = 0;
       return false;
    }

    return true;
}
```

Jeśli zamkniesz aplikację, licznik czasu może nadal tykać po zniknięciu etykiety `TimeElapsed`, co może spowodować błąd. Ta instrukcja zapobiega takiej sytuacji.

Zegar tyka co 0,1 sekundy. Dodanie 1 do tego pola pozwala śledzić, ile okresów o długości 0,1 sekundy upłynęło.

Ta instrukcja aktualizuje etykietę `TimeElapsed` na podstawie aktualnego czasu. W tym celu dzieli dziesiętne części sekundy przez 10, aby uzyskać czas w sekundach.

Jeśli przycisk „Jeszcze raz?" jest ponownie widoczny, oznacza to, że gra dobiegła końca i można zatrzymać licznik czasu. Ta instrukcja `if` uruchamia kolejne dwie instrukcje tylko wtedy, gdy gra jest w toku.

Należy zresetować licznik okresów o długości 0,1 sekundy, aby przy następnym uruchomieniu gry zacząć od 0.

Ta instrukcja powoduje zatrzymanie zegara. Żadne inne instrukcje w metodzie nie są wtedy wykonywane.

Ta instrukcja jest wykonywana tylko wtedy, gdy instrukcja `if` nie znalazła widocznego przycisku „Jeszcze raz?". Jest to dla zegara informacja, aby kontynuował działanie.

Ostatnia rzecz dotycząca zegara. Użyty kod sprawia, że zegar będzie się uruchamiał nie częściej niż co 0,1 sekundy, ale może dziać się to nieco rzadziej. Oznacza to, że zegar może *działać za wolno*. W tej grze jest to całkowicie akceptowalne!

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.

MINI Zaostrz ołówek

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

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

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#.

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*.

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.
- 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.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

P

Ś

T

U

V

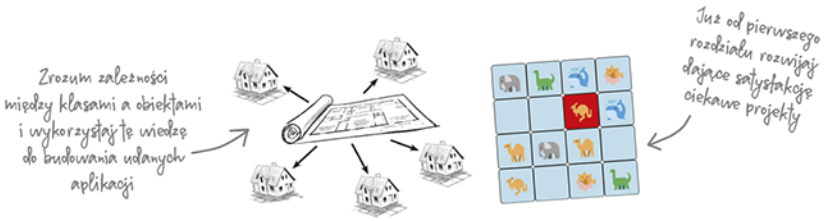
W

X

Z

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.

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-2113-9
9 788328 921139

Cena: 169,00 zł