

Spis treści (skrócony)

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

Spis treści (z prawdziwego zdarzenia)

Wprowadzenie

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

Dla kogo przeznaczona jest ta książka?	xxx
Wiemy, co sobie myślisz	xxxi
Metapoznanie	xxxiii
Oto co możesz zrobić, aby zmusić swój mózg do posłuszeństwa	xxxv
Przeczytaj to	xxxvi
Zespół redaktorów merytorycznych	xxxviii
Podziękowania	xl

Zaczynij pisać programy w języku C#

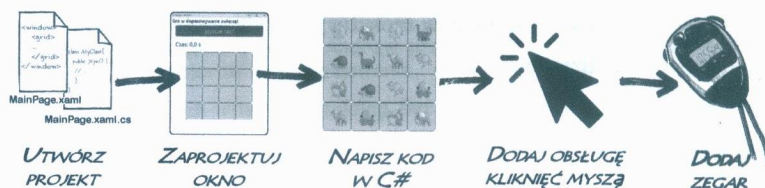
1

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!**



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



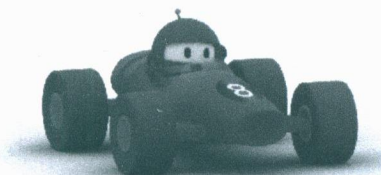
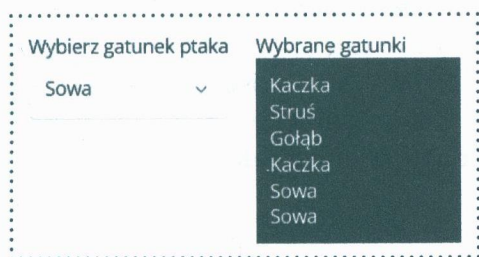
2

Zmienne, instrukcje i metody

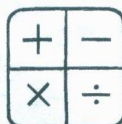
Zanurz się w kod C#

Nie jesteś jedynie użytkownikiem IDE. Jesteś programistą.

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



Przyjrzyj się plikom aplikacji konsolowej	66
Instrukcje są cegiełkami do tworzenia aplikacji	68
Instrukcje znajdują się w metodach	69
Metody używają zmiennych do pracy z danymi	70
Generowanie nowej metody używającej zmiennych	72
Dodaj do metody kod używający operatorów	73
Używanie debugera do obserwowania zmian zmiennych	74
Używanie fragmentów kodu do pisania pętli	76
Używanie operatorów do pracy ze zmiennymi	77
Instrukcje if służą do podejmowania decyzji	78
Pętle powtarzają wykonywanie operacji	79
Mechanika interfejsów użytkownika zależy od kontroltek	88
Inne kontrolki, których będziesz używać w tej książce	89
Utwórz nową aplikację do eksperymentów z kontrolkami	91
Zapoznaj się z nową aplikacją MAUI i zobacz, jak działa	92
Dodaj do aplikacji kontrolkę Entry	96
Dodaj właściwości do kontrolki Entry	97
Spraw, aby kontrolka Entry aktualizowała kontrolkę Label	98
Łączenie układów poziomych i pionowych	103
Dodaj kontrolkę Picker do wyświetlania listy pozycji do wyboru	104

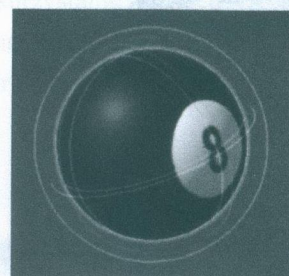
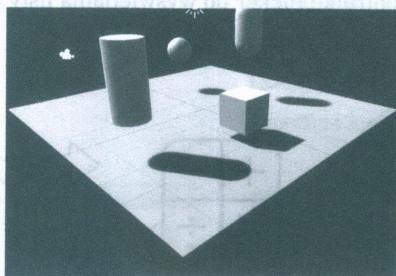


Ćwiczenia z Unity nr 1

Poznawaj C# z Unity

Witaj w pierwszych ćwiczeniach z Unity w C#. *Rusz głową.* Pisanie kodu jest umiejętnością i, podobnie jak inne umiejętności, wymaga **praktyki i eksperymentowania**, jeśli chcesz robić postępy. Unity jest wartościowym narzędziem, które Ci w tym pomoże. W tych ćwiczeniach zaczniesz stosować w praktyce wiedzę, jaką zdobyłeś w rozdziałach 1. i 2.

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



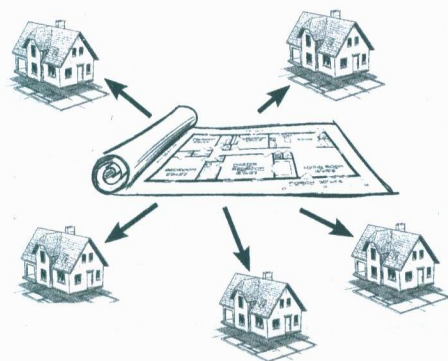
Przestrzenie nazw i klasy

Organizowanie kodu

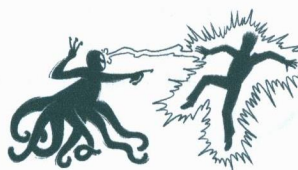
3

Świetni programiści dbają o organizację kodu i danych.

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



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



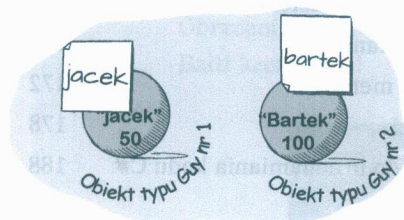
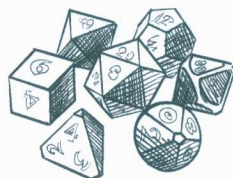
4

Dane, typy, obiekty i referencje

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!).



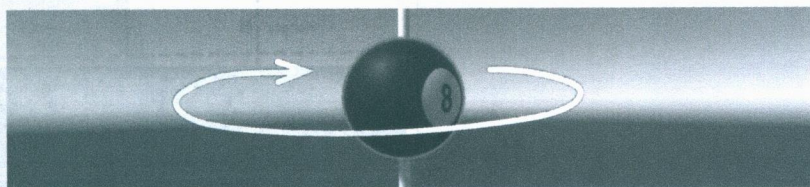
Typ zmiennej określa, jakiego rodzaju dane może ona przechowywać	192
C# udostępnia kilka typów do przechowywania liczb całkowitych	193
Porozmawiajmy o łańcuchach znaków	195
Litera! to wartość bezpośrednio zapisana w kodzie	196
Rzutowanie umożliwia kopiowanie wartości, których C# nie potrafi automatycznie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

Ćwiczenia z Unity nr 2

Pisanie kodu C# w Unity

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

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



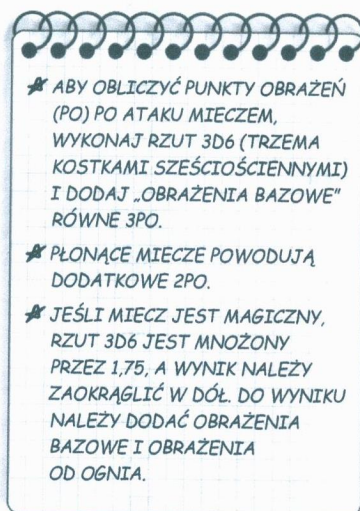
Hermetyzacja

5

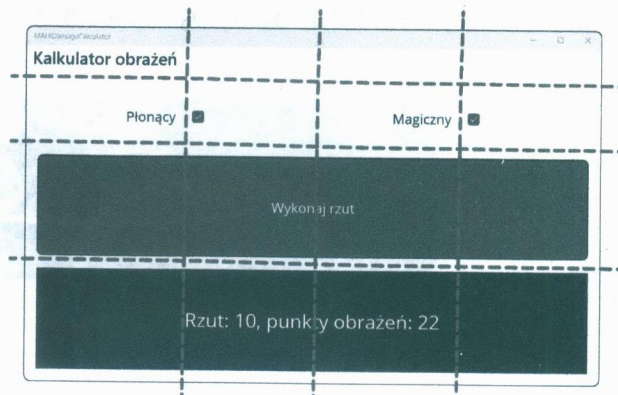
O tym, jak obiekty strzegą swoich tajemnic

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.



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



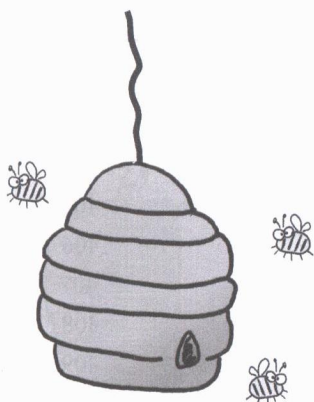
Dziedziczenie

6

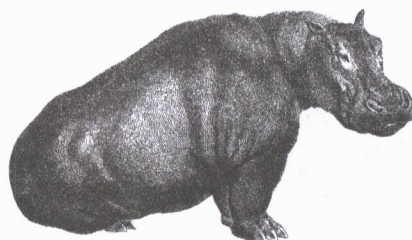
Drzewo genealogiczne obiektów

Czasem CHCESZ być taki jak Twoi rodzice.

Czy natrafiłeś kiedyś na klasę, która robi **prawie** dokładnie to, co ma robić **Tvoja** klasa? Czy naszała 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**.



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

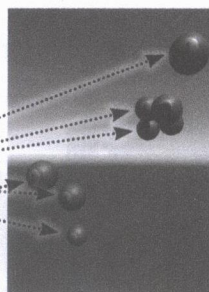
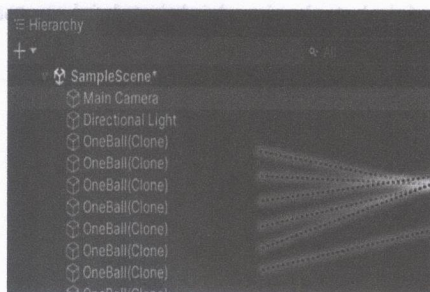


Ćwiczenia z Unity nr 3

Instancje obiektów gry

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

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



7

Interfejsy, rzutowanie i instrukcja „is”

Spełnianie obietnic przez klasy

Chcesz, aby obiekt wykonywał określone zadanie? Zastosuj interfejs.

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



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

Chrońcie
rój za wszelką cenę.



Wyliczenia i kolekcje

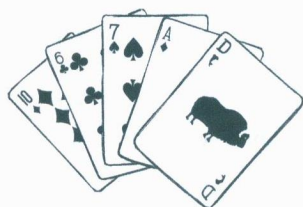
Porządkowanie danych

8

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”.



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

Sortowanie według gatunków kaczek.



Ćwiczenia z Unity nr 4

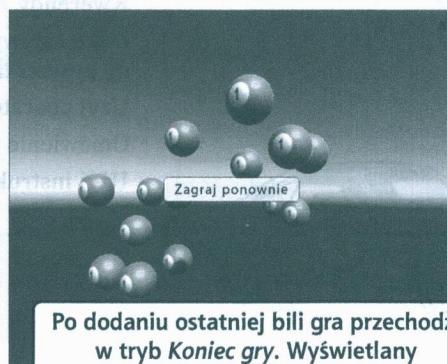
Interfejsy użytkownika

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

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



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



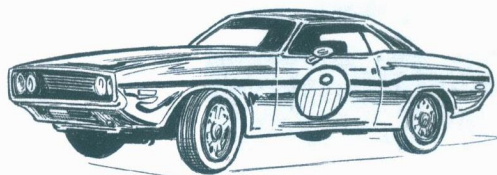
Po dodaniu ostatniej bili gra przechodzi w tryb *Koniec gry*. Wyświetlany jest przycisk *Zagraj ponownie*, a gra nie dodaje nowych bil.

Kontroluj swoje dane

9

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

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



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

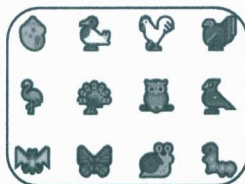
10

Odczyt i zapis plików

Zachowaj dla mnie ostatni bajt

Czasem trwałość się przydaje.

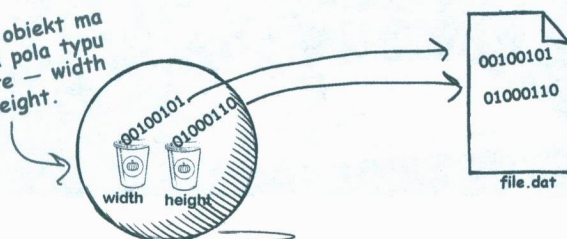
Do tej pory wszystkie omawiane programy miały krótki czas życia. Były uruchamiane, działały przez pewien czas, po czym kończyły pracę. Jednak ten model nie zawsze wystarcza — zwłaszcza gdy przetwarzasz ważne informacje. Potrzebna jest możliwość **zapisywania pracy**. W tym rozdziale zobaczysz, jak **zapisywać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

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

Ten obiekt ma
dwa pola typu
byte — width
i height.

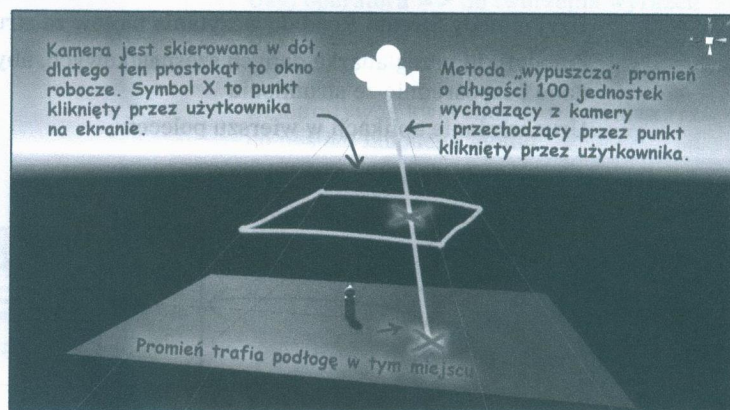


Ćwiczenia z Unity nr 5

Ray casting

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

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

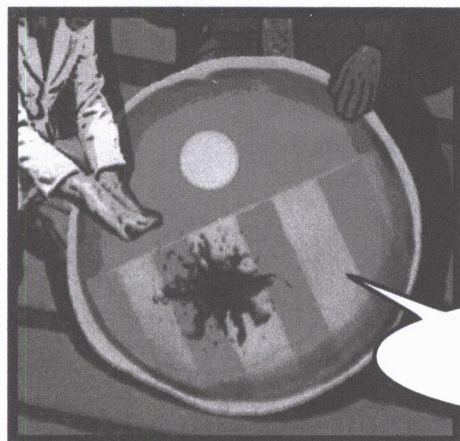
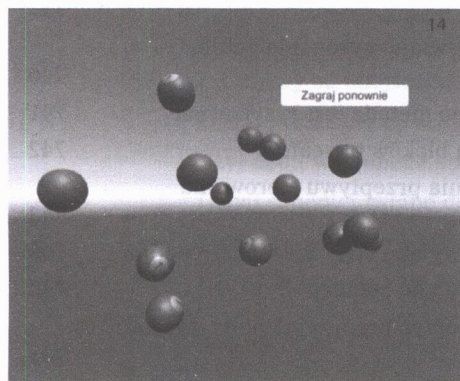


KAPITAN WSPANIAŁY

ŚMIERĆ OBIEKTU

C#. Rusz głową

4 zł

Rozdział
11.

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

Muszę... zrobić...
— Wzdycha. — *jedną...*
ostatnią... rzecz...

12

Obsługa wyjątków

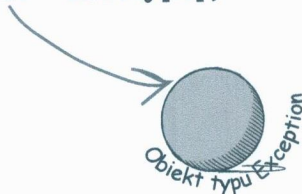
Gaszenie pożarów robi się nudne

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];
```



Przeglądarka danych szesnastkowych wczytuje nazwę pliku z wiersza poleceń	732
Gdy program zgłasza wyjątek, CLR generuje obiekt wyjątku	736
Wszystkie obiekty wyjątków dziedziczą po klasie System.Exception	737
Istnieją pliki, które nie umożliwiają wykonania rzutu szesnastkowego	740
Co się dzieje, gdy wywoływana metoda jest ryzykowna?	741
Obsługa wyjątków za pomocą bloków try-catch	742
Użyj debugera do prześledzenia przepływu sterowania w bloku try-catch	743
Ogólny blok catch obsługuje wyjątki typu System.Exception	745
Używaj wyjątku odpowiedniego do sytuacji	750
Filtry wyjątków pomagają tworzyć precyzyjne bloki do ich obsługi	754
Najgorszy blok catch W HISTORII — ogólny blok catch z komentarzami	756
Tymczasowe rozwiązania są akceptowalne (tymczasowo)	757
Użyj systemu NuGet, aby dodać do aplikacji bibliotekę do rejestrowania dzienników	759
Dodaj rejestrowanie dzienników do aplikacji ExceptionExperiment	760



Zastanawiam się, co się stanie, gdy kliknę tutaj...

Metoda Process przestanie działać, jeśli otrzyma błędne dane wejściowe!



Ćwiczenia z Unity nr 6

Nawigowanie po scenie

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

Zacznijmy od miejsca, w którym zakończyliśmy poprzednie ćwiczenia z Unity	764
Dodaj platformę do sceny	765
Użyj opcji wstępnego obliczania, aby umożliwić chodzenie po platformie	766
Dodaj schody i rampę do siatki nawigacyjnej	767
Spraw, by gracz omijał przeszkody	769
Bądź kreatywny!	770
Skorowidz	773



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.