

Stwórz grę w Unity, a nauczysz się programowania w C#!

Wydanie V

Pisanie kodu, które sprawia radość

Harrison Ferrone

Helion 



Tytuł oryginału: Learning C# by Developing Games with Unity 2020:
An enjoyable and intuitive approach to getting started with C# programming and Unity, 5th Edition

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-8144-5

Copyright © Packt Publishing 2020. First published in the English language under the title 'Learning C# by Developing Games with Unity 2020 - 5th Edition - (9781800207806)'

Polish edition copyright © 2021 by Helion S.A.
All rights reserved.

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

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

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

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/stwgr5>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	13
O recenzentach	15
Przedmowa	17
Rozdział 1. Poznaj środowisko	21
Wymagania techniczne	22
Pierwsze kroki z Unity 2020	22
Korzystanie z macOS	27
Tworzenie nowego projektu	28
Poruszanie się w edytorze	29
Korzystanie z C# w Unity	31
Korzystanie ze skryptów C#	31
Wprowadzenie do edytora Visual Studio	33
Czas na działanie — otwieranie skryptu C#	33
Uważaj na niedopasowanie nazw	34
Synchronizacja plików C#	35
Poznanie dokumentacji	35
Dostęp do dokumentacji Unity	35
Wyszukiwanie zasobów dotyczących C#	38
Podsumowanie	39
Pytania — obsługa skryptów	39
Rozdział 2. Bloki budulcowe programowania	41
Definiowanie zmiennych	42
Nazwy są ważne	42
Zmienne pełnią rolę symboli zastępczych	43
Czas na działanie — tworzenie zmiennej	43
Czas na działanie — modyfikacja wartości zmiennej	45

Metody	46
Metody definiują działania	46
Metody to także symbole zastępcze	47
Czas na działanie — tworzenie prostej metody	47
Klasy — wprowadzenie	48
Popularne klasy w Unity	49
Klasa jest planem obiektów	49
Korzystanie z komentarzy	50
Praktyczne lewe ukośniki	50
Komentarze wielowierszowe	51
Czas na działanie — wprowadzanie komentarzy	51
Wykorzystanie bloków budulcowych programowania w praktyce	52
Skrypty stają się komponentami	52
Pomocna dłoń od klasy MonoBehaviour	54
Próba gry Narodziny bohatera — klasa MonoBehaviour w Scripting API	54
Komunikowanie się pomiędzy klasami	54
Podsumowanie	55
Quiz — bloki budulcowe języka C#	55
Rozdział 3. Zmienne, typy i metody	57
Pisanie poprawnego kodu w C#	58
Debugowanie kodu	59
Deklarowanie zmiennych	60
Deklaracje typów i wartości	60
Deklarowanie samego typu	61
Korzystanie z modyfikatorów dostępu	61
Wybór poziomu bezpieczeństwa	62
Czas na działanie — tworzenie zmiennej prywatnej	62
Typy danych	63
Popularne wbudowane typy danych	63
Czas na działanie — korzystanie z różnych typów danych	64
Czas na działanie — tworzenie interpolowanych ciągów znaków	66
Konwersja typów	66
Wynioskowane deklaracje	67
Typy niestandardowe	68
Typy — ogólna charakterystyka	68
Nazwy zmiennych	68
Najlepsze praktyki	68
Zasięg zmiennej	69
Operatory	71
Operatory arytmetyczne i operatory przypisania	71
Czas na działanie — wykonywanie nieprawidłowych działań na typach	72
Definiowanie metod	73
Podstawowa składnia	73
Modyfikatory i parametry	74
Konwencje nazewnictwa	76
Metody są „objazdem” w przepływie sterowania programem	76
Metody z parametrami	77
Czas na działanie — dodawanie parametrów do metody	78

Określanie zwracanych wartości	79
Czas na działanie — dodawanie zwracanej wartości	79
Wykorzystywanie zwracanych wartości	80
Czas na działanie — przechwytywanie zwracanych wartości	80
Korzystanie z popularnych metod w Unity	81
Metoda Start	82
Metoda Update	82
Podsumowanie	83
Quiz — zmienne i metody	84
Rozdział 4. Przepływ sterowania i kolekcje	85
<hr/>	
Instrukcje wyboru	86
Instrukcja if-else	86
Korzystanie z operatora NOT	89
Zagnieżdżanie instrukcji	91
Ocena wielu warunków	92
Instrukcja switch	94
Podstawowa składnia	94
Dopasowywanie wzorców	95
Przypadki fall-through	96
Quiz nr 1 — instrukcje warunkowe	98
Kolekcje — wprowadzenie	98
Tablice	99
Indeksy tablic	100
Wyjątki przekroczenia zakresu	101
Listy	101
Słowniki	104
Podstawowa składnia	104
Korzystanie z par zapisanych w słowniku	105
Quiz nr 2 — wszystko o kolekcjach	107
Instrukcje iteracyjne	107
Pętle for	107
Czas na działanie — wyszukiwanie elementu	109
Pętle foreach	110
Pętle while	112
Czas na działanie — śledzenie życia gracza	112
Do nieskończoności i dalej	113
Podsumowanie	114
Rozdział 5. Klasy, struktury i programowanie obiektowe	115
<hr/>	
Definiowanie klas	116
Podstawowa składnia	116
Tworzenie egzemplarzy klasy	117
Dodawanie pól klasy	118
Korzystanie z konstruktorów	119
Deklarowanie metod klasy	121
Deklarowanie struktur	123
Podstawowa składnia	123

Typy referencyjne i typy wartości	125
Typy referencyjne	125
Typy wartości	127
Myślenie w kategoriach obiektów	128
Hermetyzacja	128
Dziedziczenie	130
Kompozycja	132
Polimorfizm	132
Przegląd informacji o paradygmacie OOP	133
Zastosowanie OOP w Unity	134
Obiekty są realizacjami klas	134
Dostęp do komponentów	135
Podstawowa składnia	136
Przeciągnij i upuść	139
Podsumowanie	140
Quiz — wszystko o OOP	140
Rozdział 6. Ubrudź sobie ręce silnikiem Unity	141
<hr/>	
Elementarz projektu gry	142
Dokumentacja projektowa gry	142
Dokumentacja jednostronicowa gry Narodziny bohatera	143
Budowanie poziomu	144
Tworzenie prymitywów	144
Myślenie w 3D	146
Materiały	148
White-boxing	150
Podstawy oświetlenia	158
Tworzenie oświetlenia	159
Właściwości komponentów oświetlenia	160
Animacje w Unity	161
Tworzenie klipów	161
Nagrywanie klatek kluczowych	163
Krzywe i styczne	165
System cząstek	168
Czas na działanie — dodanie efektów blasku	168
Podsumowanie	169
Quiz — podstawowe własności silnika Unity	170
Rozdział 7. Ruch, sterowanie kamerą i kolizje	171
<hr/>	
Poruszanie postacią	172
Konfigurowanie gracza	173
Wektory	174
Pobieranie danych wejściowych od gracza	175
Dodawanie kamery śledzącej	179
Czas na działanie — skrypt z implementacją zachowania kamery	179
System fizyki w Unity	181
Komponenty Rigidbody w ruchu	183
Komponenty Collider i kolizje	187

Zastosowanie wyzwalaczy komponentów Collider	190
System fizyki — przegląd	193
Podsumowanie	194
Quiz — sterowanie graczem i system fizyki	194
Rozdział 8. Skrypty do obsługi mechaniki gry	195
Obsługa skoków	196
Typy wyliczeniowe	196
Typy wartości w enumeracjach	197
Czas na działanie — naciskanie spacji, aby skakać!	198
Zastosowanie masek warstw	199
Czas na działanie — ustawienie warstw obiektów	199
Czas na działanie — jeden skok naraz	201
Mechanizm strzelania	203
Tworzenie egzemplarzy obiektów	204
Czas na działanie — tworzenie prefabrykatu pocisku	204
Czas na działanie — dodanie mechaniki strzelania	205
Zarządzanie obiektami GameObject	207
Czas na działanie — niszczenie pocisków	208
Tworzenie menedżera gry	208
Śledzenie właściwości gracza	209
Czas na działanie — tworzenie menedżera gry	209
Pobieranie i ustawianie właściwości	210
Czas na działanie — dodawanie zmiennych wspierających	212
Czas na działanie — aktualizacja kolekcji przedmiotów	213
„Polerowanie” gry	215
Graficzny interfejs użytkownika	215
Czas na działanie — dodawanie elementów interfejsu użytkownika	216
Warunki wygrywania i przegrywania	218
Czas na działanie — wygrana	219
Dyrektwy using i przestrzenie nazw	220
Czas na działanie — zatrzymywanie i ponowne uruchamianie	221
Podsumowanie	222
Quiz — mechanika gry	223
Rozdział 9. Podstawy sztucznej inteligencji. Zachowania nieprzyjaciół	225
Nawigacja w Unity	226
Komponenty nawigacyjne	226
Ruchome agenty nieprzyjaciół	229
Programowanie proceduralne	229
Mechanika gry nieprzyjaciela	236
Znajdź i zniszcz	236
Refaktoryzacja i zastosowanie zasady DRY	241
Czas na działanie — refaktoryzacja metody restart	241
Podsumowanie	242
Quiz — mechanizmy AI i nawigacja	243

Rozdział 10. Więcej o typach, metodach i klasach	245
Więcej informacji o modyfikatorach dostępu	246
Właściwości stałe i tylko do odczytu	246
Wykorzystanie słowa kluczowego static	247
Czas na działanie — tworzenie klasy statycznej	247
Więcej informacji o metodach	249
Przeciążanie metod	249
Czas na działanie — przeciążanie metody restartowania poziomu	249
Parametry ref	251
Czas na działanie — śledzenie restartów	251
Parametry out	252
Więcej informacji o OOP	253
Interfejsy	253
Czas na działanie — tworzenie interfejsu menedżera	254
Czas na działanie — implementacja interfejsu	255
Klasy abstrakcyjne	256
Rozszerzanie klas	258
Czas na działanie — rozszerzanie klasy String	258
Czas na działanie — korzystanie z metod rozszerzeń	259
Więcej o przestrzeniach nazw	261
Alias typów	261
Podsumowanie	261
Quiz — wchodzimy o poziom wyżej	262
Rozdział 11. Stosy, kolejki i kolekcje HashSet	263
Stosy — wprowadzenie	264
Podstawowa składnia	264
Zdejmowanie elementów ze stosu i podglądanie ich	266
Popularne metody	267
Kolejki	268
Podstawowa składnia	269
Dodawanie, usuwanie i podglądanie elementów kolejek	269
Najważniejsze metody	270
Korzystanie z kolekcji HashSet	270
Podstawowa składnia	271
Wykonywanie działań na zbiorach	271
Podsumowanie	273
Quiz — zaawansowane kolekcje	273
Rozdział 12. Typy generyczne, delegaty i nie tylko	275
Wprowadzenie do typów generycznych	275
Obiekty generyczne	276
Metody generyczne	278
Ograniczenia parametrów typu	280
Delegowanie działań	281
Podstawowa składnia	281
Delegaty jako typy parametrów	283

Zdarzenia	284
Podstawowa składnia	285
Obsługa subskrypcji zdarzeń	286
Obsługa wyjątków	288
Zgłaszanie wyjątków	288
Korzystanie z bloków try-catch	291
Elementarz wzorców projektowych	293
Popularne wzorce stosowane w grach	294
Podsumowanie	295
Quiz — zaawansowane możliwości języka C#	295
Rozdział 13. Dalsza podróż	297
<hr/>	
To tylko namiastka	297
Przypomnienie zasad programowania obiektowego	298
Podejście do projektów Unity	299
Własności silnika Unity, których nie opisałem	299
Następne kroki	300
Zasoby związane z językiem C#	300
Zasoby związane z Unity	300
Certyfikaty z Unity	301
Narodziny bohatera — pokaż grę światu	302
Podsumowanie	302
Odpowiedzi do quizów	303
<hr/>	

Bloki budulcowe programowania

Każdy język programowania, gdy zetkniesz się z nim po raz pierwszy, wygląda jak starożytna greka. Język C# nie jest wyjątkiem. Dobra wiadomość jest jednak taka, że poza tą początkową tajemniczością wszystkie języki programowania składają się z tych samych podstawowych bloków budulcowych. Zmienne, metody i klasy (lub obiekty) to DNA konwencjonalnego programowania. Zrozumienie tych prostych pojęć otwiera cały świat zróżnicowanych i złożonych zastosowań. Tak samo jest z ludźmi — chociaż we wszystkich mieszkańcach Ziemi są tylko cztery różne nukleobazy DNA, to każdy z nas jest unikatowym organizmem.

Jeśli jesteś nowicjuszem w programowaniu, w tym rozdziale znajdziesz wiele przydatnych informacji, których poznanie może wywrzeć wpływ na pierwsze wiersze kodu, jakie kiedykolwiek napiszesz. Nie chodzi mi jednak o to, aby przeciążać Twój mózg faktami i liczbami. Ma on dać Ci holistyczny wgląd w bloki budulcowe programowania z wykorzystaniem przykładów z codziennego życia.

Ten rozdział prezentuje ogólny widok elementów, z których składa się program. Zapoznanie się ze sposobem współdziałania poszczególnych elementów przed bezpośrednim przystąpieniem do kodowania nie tylko pomoże początkującym programistom postawić pierwsze kroki, ale również, dzięki łatwym do zapamiętania porównaniom, pozwoli utrwalić zdobyte wiadomości. Dość marudzenia. W tym rozdziale skoncentruję się na następujących tematach:

- Czym są zmienne i jak ich używać?
- Przeznaczenie metod.
- Klasy i ich relacja względem obiektów.
- Przekształcanie skryptów C# na komponenty Unity.
- Komunikacja pomiędzy komponentami i notacja z kropką.

Definiowanie zmiennych

Zacznijmy od prostego pytania: Czym jest zmienna? W zależności od punktu widzenia istnieje kilka różnych sposobów odpowiedzi:

- **Pojęciowo** zmienna jest najbardziej podstawową jednostką programowania — tym samym, czym atom jest dla świata fizycznego. Wszystko zaczyna się od zmiennych, a programy nie mogą bez nich istnieć.
- **Technicznie** zmienna jest niewielkim fragmentem pamięci komputera, w której jest przechowywana przypisana wartość. Każda zmienna definiuje miejsce, gdzie jest zapisana związana z nią informacja (ta lokalizacja nazywa się adresem pamięci), określa, jaka jest jej wartość i typ danych (na przykład liczby, słowa lub listy).
- **Praktycznie** zmienna jest kontenerem. Możesz stworzyć nową zmienną, kiedy chcesz, wypełnić ją informacjami, przenieść w inne miejsce, zastąpić to, co zawiera, i odwoływać się do niej w razie potrzeby. Zmienne mogą być przydatne nawet wtedy, gdy są puste.

Analogią do zmiennej w życiu jest skrzynka pocztowa. Pamiętasz ją?



Mogą znaleźć się w niej listy, rachunki, zdjęcie ciotki Marysi — cokolwiek. Chodzi o to, że w skrzynce pocztowej mogą być przechowywane różne rzeczy — mogą mieć nazwy, zawierać informacje (poczta fizyczna), a ich zawartość może się zmieniać (jeśli masz odpowiednie uprawnienia).

Nazwy są ważne

Odnosząc się do poprzedniego zdjęcia, gdybym poprosił Cię, abys podszedł do skrzynki pocztowej i ją otworzył, pierwszą rzeczą, o którą prawdopodobnie byś zapytał, byłoby: Którą? Gdybym odpowiedział, że rodzinną skrzynkę Kowalskich albo brązową skrzynkę pocztową,

albo okrągłą skrzynkę pocztową, to zyskałbyś niezbędny kontekst do tego, aby otworzyć tę skrzynkę pocztową, o którą mi chodzi. Na podobnej zasadzie, gdy tworzysz zmienne, powinieneś nadać im unikatowe nazwy, do których można się później odwołać. Specyfiką odpowiedniego formatowania i nadawania opisowych nazw zajmiemy się w rozdziale 3.

Zmienne pełnią rolę symboli zastępczych

Gdy tworzysz zmienną i nadajesz jej nazwę, w gruncie rzeczy tworzysz symbol zastępczy (*placeholder*) wartości, którą chcesz przechowywać. Dla przykładu rozważmy następujące proste równanie arytmetyczne:

$$2 + 9 = 11$$

Nie ma tu niczego tajemniczego. Co jednak zrobić, aby liczba 9 była zmienną? Rozważmy następujący blok kodu:

```
mojaZmienna = 9
```

Teraz możemy użyć nazwy zmiennej `mojaZmienna` jako substytutu wartości 9 wszędzie, gdzie tego potrzebujemy:

$$2 + \text{mojaZmienna} = 11$$

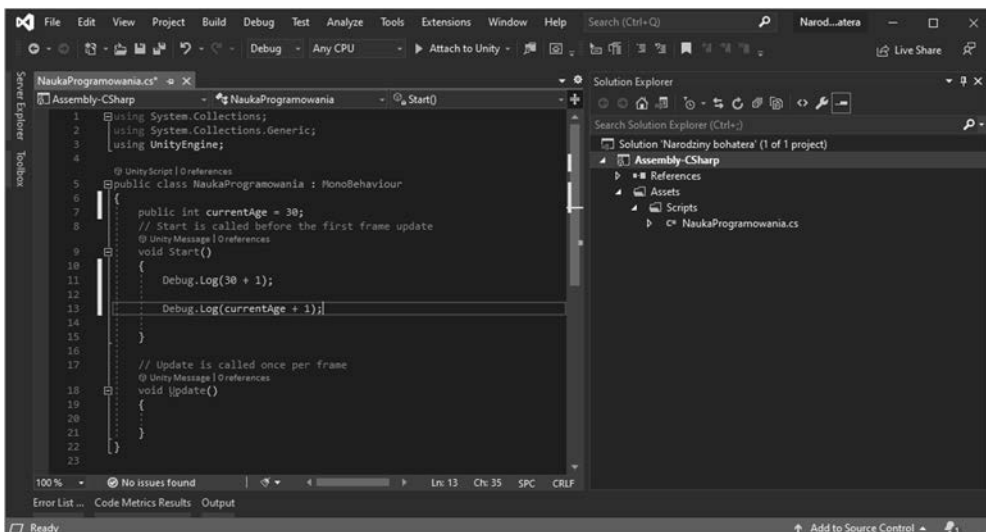
Jeśli zastanawiasz się, czy wykorzystaniem zmiennych rządzą specjalne zasady, odpowiedź brzmi: „Tak”. Opowiemy o nich w następnym rozdziale, więc na razie się nimi nie martw.

Chociaż ten przykład nie jest realnym kodem w C#, ilustruje moc zmiennych i ich zastosowanie w roli symboli zastępczych. W następnym punkcie zaczniesz tworzyć własne zmienne, więc kontynuuj lekturę!

Czas na działanie — tworzenie zmiennej

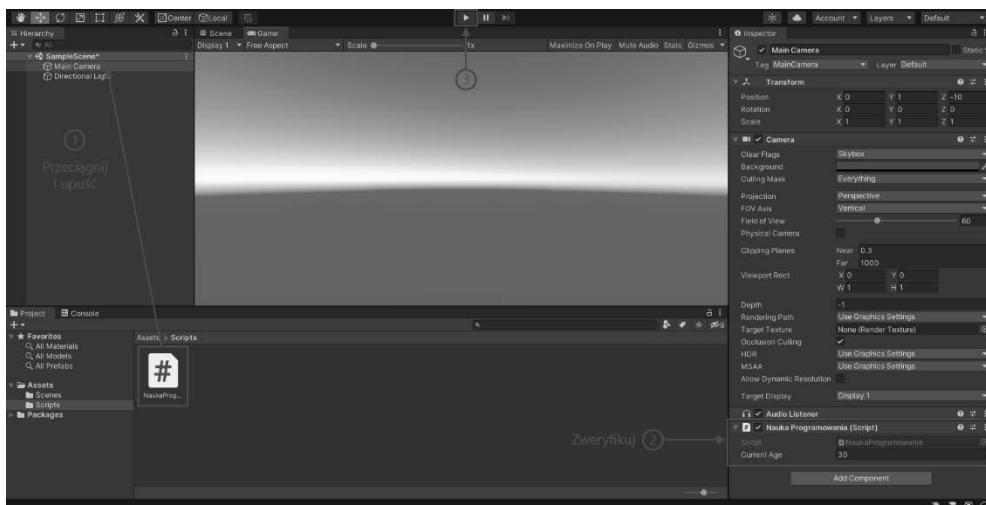
Dość teorii. Stwórzmy prawdziwą zmienną w naszym skrypcie *NaukaProgramowania*:

1. Kliknij dwukrotnie skrypt *NaukaProgramowania*, aby otworzyć go w Visual Studio, i wprowadź wiersze 7., 11. i 13. (na razie nie martw się o składnię — po prostu zadбай o to, aby Twój skrypt wyglądał tak samo, jak skrypt pokazany na zrzucie ekranu na następnej stronie).
2. Zapisz plik za pomocą kombinacji klawiszy *Command* + *S* na klawiaturze Mac lub *Ctrl* + *S* na klawiaturze Windows.

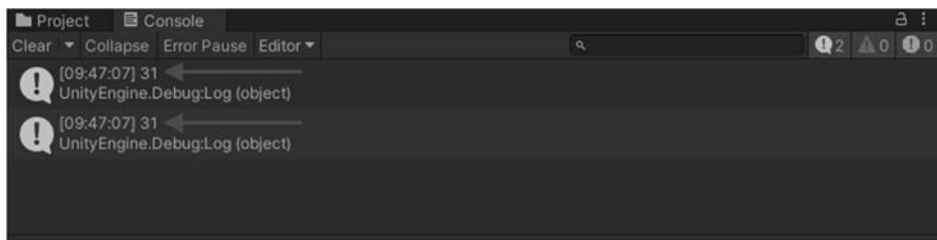


Aby skrypty mogły działać w Unity, muszą być dodane do kolekcji GameObjects na scenie. Gra *Narodziny bohatera* domyślnie zawiera kamerę i kierunkowe światło, które zapewnia oświetlenie sceny. Dla uproszczenia dodajmy więc skrypt *NaukaProgramowania* do kamery:

1. Przeciągnij i upuść skrypt *NaukaProgramowania.cs* na obiekt *Main Camera*.
2. Wybierz pozycję *Main Camera* tak, by wyświetliła się w panelu *Inspector*, i zadбай o to, by komponent *NaukaProgramowania.cs* (Script) został prawidłowo podłączony.
3. Kliknij *Play* i obserwuj wynik w panelu *Console*:



Instrukcje `Debug.Log()` spowodowały wyświetlenie wyniku prostych równań matematycznych umieszczonych pomiędzy nawiasami. Jak widać na poniższym zrzucie ekranu panelu *Console*, równanie, które wykorzystywało zmienną, działało tak samo, jakby wykorzystano w nim liczbę:

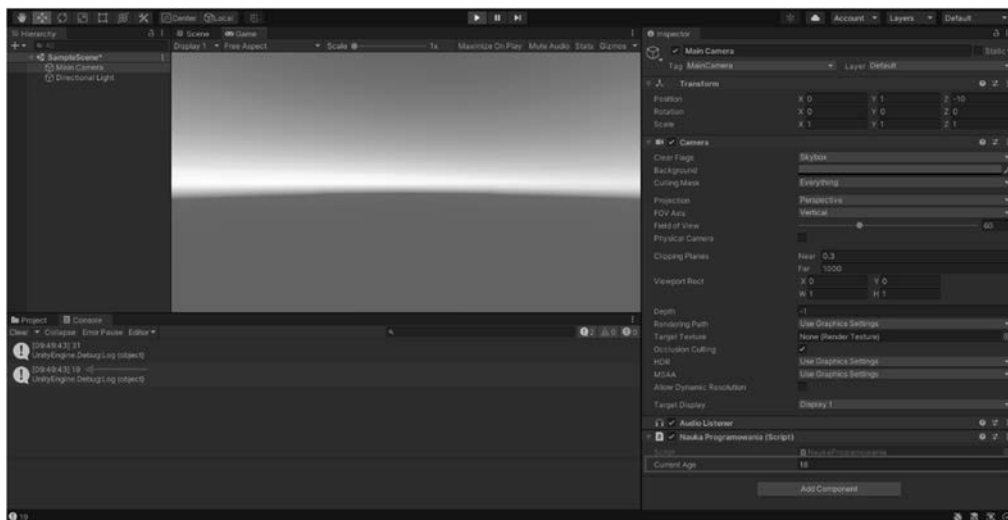


Opis sposobu, w jaki Unity przekształca skrypty C# na komponenty, zamieszczę na końcu tego rozdziału. Najpierw jednak pokażę, jak można zmienić wartość jednej ze zdefiniowanych zmiennych.

Czas na działanie — modyfikacja wartości zmiennej

Ponieważ w wierszu 7. zadeklarowałeś `currentAge` jako zmienną, zapisana w niej wartość może się zmienić. Następnie, we wszystkich miejscach w kodzie, gdzie została użyta zmienna, będzie wykorzystana jej zaktualizowana wartość. Zobaczmy, jak to działa w praktyce:

1. Jeśli scena nadal działa, zatrzymaj grę, klikając przycisk *Play*.
2. W panelu *Inspector* zmień wartość *Current Age* na 18 i ponownie odtwórz scenę. Jednocześnie obserwuj wynik w panelu *Console*:



Pierwszy wynik nadal będzie wynosić 31, ale drugi, ponieważ zmieniliśmy wartość zmiennej, będzie wynosił 19.

Celem pokazanego przykładu nie było szczegółowe omówienie składni zmiennych, ale zaprezentowanie, że zmienne działają jak kontenery, które można utworzyć raz, a następnie odwoływać się do nich w innym miejscu. Więcej informacji o zmiennych znajdziesz w rozdziale 3.

Teraz, gdy wiesz, jak tworzyć zmienne w języku C# i przypisywać do nich wartości, nadszedł czas, by przejść do kolejnego ważnego bloku budulcowego programowania: metod!

Metody

Zmienna sama w sobie służy do niczego więcej niż śledzenie przypisanej do niej wartości. Chociaż ma to kluczowe znaczenie, same zmienne nie wystarczą do stworzenia złożonych aplikacji. Co zatem zrobić, aby tworzyć operacje i implementować zachowania w kodzie? Krótka odpowiedź brzmi: należy skorzystać z metod.

Zanim dotrzemy do tego, czym są metody i jak z nich korzystać, warto zapoznać się z niezbędną terminologią. W świecie programowania powszechnie występują pojęcia *metoda* i *funkcja*, które są używane zamiennie, zwłaszcza w kontekście Unity. Ponieważ C# jest językiem obiektowym (to zagadnienie opiszę w rozdziale 5., „Klasy, struktury i programowanie obiektowe”), w pozostałej części tej książki będę posługiwać się terminem *metoda*, co jest zgodne ze standardowymi wytycznymi obowiązującymi dla języka C#.

Gdy w dokumentacji *Scripting Reference* lub dowolnej innej dokumentacji napotkasz termin *funkcja*, możesz przyjąć, że chodzi o *metodę*.

Metody definiują działania

Podobnie jak w przypadku zmiennych, definicja metod w programowaniu może być bardzo rozwlekła lub niebezpiecznie krótka. Oto trzy podejścia do rozważenia:

- **Pojęciowo** metody opisują sposoby wykonywania działań w aplikacjach.
- **Technicznie** metoda jest blokiem kodu zawierającym wykonywalne instrukcje, które są uruchamiane, gdy metoda zostanie wywołana za pomocą jej nazwy. Metody mogą przyjmować argumenty (zwane również parametrami), które mogą być wykorzystane wewnątrz zasięgu metody.
- **Praktycznie** metoda jest kontenerem dla zestawu instrukcji, które są uruchamiane za każdym razem, gdy metoda zostanie wywołana. Te kontenery także mogą przyjmować zmienne jako dane wejściowe. Do zmiennych przekazanych jako parametry można się odwoływać tylko wewnątrz metody.

Ogólnie rzecz biorąc, metody stanowią szkielet każdego programu — łączą kod ze sobą. Prawie wszystko w programach jest zbudowane z metod.

Metody to także symbole zastępcze

W celu wyjaśnienia koncepcji metod rozważmy uproszczony przykład dodawania dwóch liczb. Gdy piszesz skrypt, to w gruncie rzeczy układasz wiersze kodu, które komputer powinien uruchomić po kolei. Gdy po raz pierwszy musisz dodać do siebie dwie liczby, możesz po prostu zrobić to tak jak w poniższym bloku kodu:

```
pierwszaLiczba + innaLiczba
```

Później jednak zauważysz, że w innym miejscu również musisz dodać do siebie dwie liczby. Zamiast kopiowania i wklejenia tego samego wiersza kodu, co skutkuje powstaniem kodu „spaghetti” i czego należy za wszelką cenę unikać, możesz utworzyć metodę, która zajmie się tym działaniem:

```
DodajLiczby
{
    pierwszaLiczba + innaLiczba
}
```

Teraz metoda `DodajLiczby`, podobnie jak zmienna, jest symbolem zastępczym w pamięci. Jednak zamiast wartości, zawiera blok instrukcji. Gdy posłużysz się nazwą metody (czyli ją wywołasz) w dowolnym miejscu skryptu, spowoduje to natychmiastowe wstawienie zapisanych w niej instrukcji. Nie musisz w tym celu powtarzać żadnego kodu.

Jeśli odkryjesz, że w kółko wpisujesz te same wiersze kodu, to prawdopodobnie nie zauważyłeś, iż można uprościć powtarzane działania i umieścić je w metodach.

Ciągle wpisywanie tego samego kodu jest przez programistów żartobliwie nazywane tworzeniem *kodu spaghetti*. Być może słyszałeś również o stosowanej przez programistów zasadzie DRY (*Don't Repeat Yourself* — dosłownie: nie powtarzaj się), którą powinieneś powtarzać jak mantrę.

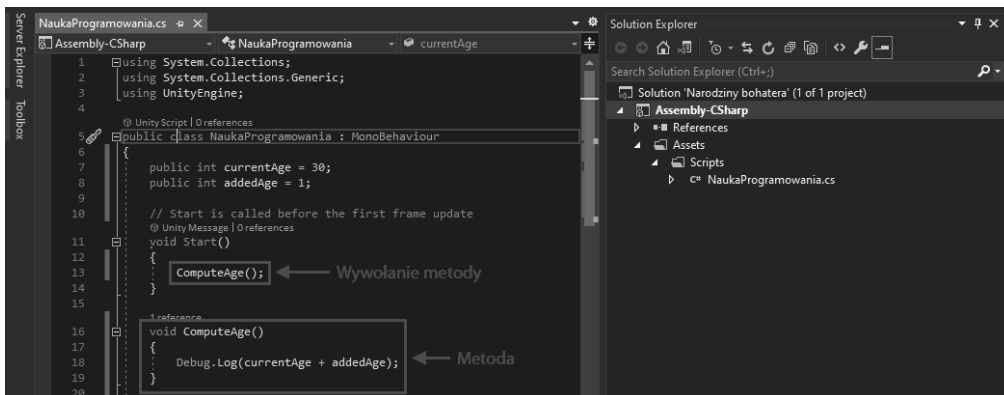
Po zaprezentowaniu nowego pojęcia w pseudokodzie najlepiej zaimplementować je samodzielnie. Aby ośwoić pojęcie metod, zrobimy to w następnym punkcie.

Czas na działanie — tworzenie prostej metody

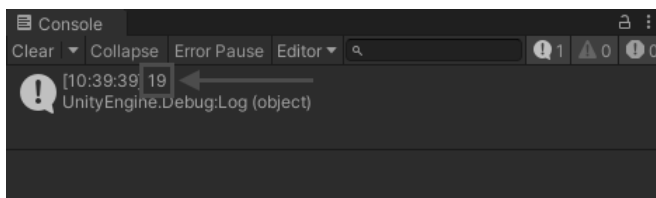
Otwórzmy ponownie skrypt *NaukaProgramowania* i zobaczymy, jak działają metody w języku C#. Podobnie jak w przypadku przykładu użycia zmiennych, skopiuj kod do skryptu, aby wyglądał dokładnie tak, jak na poniższym zrzucie ekranu. Dla zapewnienia zwiezłości usunąłem poprzedni przykład kodu ze skryptu. Oczywiście możesz go pozostawić:

1. Otwórz skrypt *NaukaProgramowania* w Visual Studio i dodaj wiersze 8., 13. oraz od 16. do 19.

2. Zapisz plik, a następnie naciśnij *Play* w Unity, aby zobaczyć nowe wyjście w panelu *Console*:



Zdefiniowałeś swoją pierwszą metodę w wierszach od 16. do 19. i wywołałeś ją w wierszu 13. Teraz wszędzie, gdziekolwiek zostanie wywołana metoda `ComputeAge()`, zostaną do siebie dodane dwie zmienne, a wynik zostanie wyświetlony na konsoli:



Za pomocą panelu *Inspector* wypróbuj różne wartości zmiennych, aby zobaczyć działanie metody w praktyce! Więcej informacji na temat składni kodu, który napisałeś przed chwilą, znajdziesz w następnym rozdziale.

Po ogólnym przedstawieniu pojęcia metod jesteśmy gotowi do opisania najbardziej obszerneho tematu w dziedzinie programowania — klas!

Klasy — wprowadzenie

Powiedziałem wcześniej, że zmienne przechowują informacje, a metody wykonują działania. Jednak same zmienne i metody to za mało. Potrzebny jest sposób stworzenia czegoś w rodzaju superkontenera. Taki superkontener miałby swoje zmienne i metody, do których można odwoływać się z wewnątrz. Potrzebujemy klas:

- **Pojęciowo** klasa przechowuje w pojedynczym kontenerze powiązane ze sobą informacje, działania i zachowania. Klasy mogą nawet komunikować się ze sobą.

- **Technicznie** klasy są strukturami danych. Mogą zawierać zmienne, metody i inne informacje programowe. Do nich wszystkich można się odwoływać po stworzeniu obiektu klasy.
- **Praktycznie** klasa jest planem obiektu. Określa reguły i sposób działania dowolnego obiektu (zwanego egzemplarzem) utworzonego za pomocą tego planu.

Prawdopodobnie uświadomiłeś sobie, że klasy otaczają nas nie tylko w Unity, ale także w świecie rzeczywistym. W dalszej części tego rozdziału zajmę się najczęściej używanymi klasami w Unity oraz opisem sposobu ich działania.

Popularne klasy w Unity

Zanim zaczniesz się zastanawiać, jak wygląda klasa w C#, powinieneś zdać sobie sprawę, że korzystasz z klasy od początku tego rozdziału. Domyślnie każdy skrypt utworzony w Unity jest klasą. Można to rozpoznać po słowie kluczowym `class` w wierszu 5.:

```
public class NaukaProgramowania: MonoBehaviour
```

`MonoBehaviour` oznacza, że tę klasę można dołączyć do obiektu `GameObject` na scenie Unity. W C# klasy mogą istnieć samodzielnie. Przekonasz się o tym, gdy zaczniesz samodzielnie tworzyć klasy w rozdziale 5.

Pojęcia *skrypt* i *klasa* w kontekście zasobów Unity są czasami używane zamiennie. W celu zachowania spójności będę nazywał pliki C# *skryptami*, jeśli będą dołączane do kolekcji obiektów `GameObjects`, oraz *klasami*, jeśli będą samodzielne.

Klasa jest planem obiektów

W ramach ostatniego przykładu pomyślmy o lokalnym urzędzie pocztowym. Jest to odrębne, samodzielne środowisko, które ma właściwości takie jak fizyczny adres (zmienna) oraz zdolność do wykonywania działań, takich jak wysyłanie kuponu Twojego tajnego dekodera (metoda).

To sprawia, że urząd pocztowy jest doskonałym przykładem potencjalnej klasy, którą możemy naszkicować za pomocą następującego bloku pseudokodu:

```
PostOffice
{
    // Zmienna
    Address = "Otwieracz listów 1234"
    // Metody
    DeliverMail()
    SendMail()
}
```

Najważniejsze do zapamiętania w przypadku klas jest to, że jeśli informacje i zachowania są zgodne z predefiniowanym projektem, to możliwe jest wykonywanie złożonych działań oraz realizacja komunikacji pomiędzy klasami.

Na przykład, gdybyśmy mieli inną klasę, która chciałaby wysłać list za pośrednictwem klasy `PostOffice`, nie musielibyśmy się zastanawiać, co zrobić, by wykonać tę operację. Moglibyśmy po prostu wywołać funkcję `SendMail` klasy `PostOffice` w następujący sposób:

```
PostOffice.SendMail()
```

Moglibyśmy także użyć tej klasy do ustalenia adresu urzędu pocztowego. Dzięki temu moglibyśmy ustalić, gdzie należy kierować nasze listy:

```
PostOffice.Address
```

Jeśli zastanawiasz się nad wykorzystaniem kropek (tzw. notacji z kropką) pomiędzy słowami, proszę o trochę cierpliwości — zajmę się tym pod koniec rozdziału.

Twój podstawowy zestaw narzędzi programistycznych jest już kompletny (cóż, przynajmniej w teorii). Pozostałą część tego rozdziału poświęcę na dokładniejsze zapoznanie Cię ze składnią i praktycznymi zastosowaniami zmiennych, metod i klas.

Korzystanie z komentarzy

Być może zauważyłeś, że w skrypcie *NaukaProgramowania* są dwa dziwne wiersze szarego tekstu zaczynające się od dwóch lewych ukośników (`//`). Wiersze te zostały utworzone domyślnie podczas tworzenia skryptu. To są komentarze do kodu — proste, ale potężne narzędzie dla programistów.

W języku `C#` jest kilka sposobów tworzenia komentarzy. Środowisko `Visual Studio` (oraz inne aplikacje do edycji kodu), dzięki wbudowanym skrótom często jeszcze bardziej ułatwia ich tworzenie.

Niektórzy profesjonaliści nie nazwaliby komentarzy niezbędnym blokiem budulcowym programowania, ale choć szanuję tę opinię, to jednak się z nią nie zgadzam. Prawidłowe komentowanie kodu za pomocą opisowych informacji jest jednym z najbardziej podstawowych nawyków, jakie powinien mieć nowoczesny programista.

Praktyczne lewe ukośniki

W skrypcie *NaukaProgramowania* znajdują się komentarze jednowierszowe:

```
//To jest komentarz jednowierszowy
```

`Visual Studio` nie interpretuje wierszy zaczynających się od dwóch lewych ukośników (bez spacji pomiędzy nimi) jako kodu, dzięki czemu można je dowolnie formułować.

Komentarze wielowierszowe

Zgodnie z nazwą można śmiało założyć, że jednowierszowe komentarze dotyczą tylko jednego wiersza kodu. Jeśli interesują Cię komentarze wielowierszowe, musisz użyć lewego ukośnika i gwiazdki jako symbolu otwarcia komentarza oraz gwiazdki i lewego ukośnika jako znacznika zamknięcia komentarza:


```
/* to jest
   komentarz wielowierszowy */
```

Możesz także komentować i odkomentowywać bloki kodu, zaznaczając je i korzystając z polecenia `+?` — skrót na komputerach MacOS oraz `Ctrl + K + C` w systemie Windows.

Można oglądać przykładowe komentarze w kodzie, ale żeby zapoznać się z tym mechanizmem, lepiej samodzielnie spróbować je stworzyć. Na rozpoczęcie wprowadzania komentarzy w kodzie nigdy nie jest za wcześnie!

Czas na działanie — wprowadzanie komentarzy

Visual Studio zapewnia również przydatną własność automatycznego generowania komentarzy. Wpisz trzy lewe ukośniki w wierszu poprzedzającym dowolny wiersz kodu (z deklaracją zmiennej, metody, klasy lub innych konstrukcji). Spowoduje to wstawienie bloku komentarza z podsumowaniem. Otwórz skrypt *NaukaProgramowania* i wprowadź trzy lewe ukośniki nad deklaracją metody `ComputeAge()`:



```
16  /// <summary>
17  /// Wyświetla zmodyfikowany wiek gracza
18  /// </summary>
19  1 reference
20  void ComputeAge()
21  {
22  Debug.Log(currentAge + addedAge);
}
```

Powinieneś zobaczyć wygenerowany przez Visual Studio trzywierszowy komentarz z opisem metody, umieszczony pomiędzy dwoma znacznikami `<summary>`. Możesz oczywiście zmodyfikować ten tekst lub wprowadzić nowe wiersze. W tym celu wystarczy wcisnąć `Enter`, tak jak w zwykłym dokumencie tekstowym. Pamiętaj, aby nie usunąć znaczników.

Przeznaczenie tych szczegółowych komentarzy staje się jasne, gdy chcesz się czegoś dowiedzieć o metodzie, którą napisałeś. Jeśli umieścisz komentarz `<summary>`, zaczynający się od potrójnego lewego ukośnika, i naprowadzisz wskaźnik myszki na nazwę metody w Visual Studio, na ekranie wyświetli się podsumowanie zawierające opis metody:

```

10 // Start is called before the first frame update
11 @ Unity Message | 0 references
12 void Start()
13 {
14     ComputeAge();
15 }
16 void NaukaProgramowania.ComputeAge()
17 // Wyświetla zmodyfikowany wiek gracza
18 // <summary>
19 // Wyświetla zmodyfikowany wiek gracza
20 // </summary>
21 1 reference
22 void ComputeAge()
23 {

```

Powinniśmy teraz zastanowić się, jak połączyć wszystkie bloki budulcowe, które poznaliśmy w tym rozdziale, w środowisku do tworzenia gier Unity. Tym właśnie zajmę się w następnym punkcie!

Wykorzystanie bloków budulcowych programowania w praktyce

Po omówieniu bloków budulcowych programowania nadszedł czas, aby przed zakończeniem tego rozdziału wykonać kilka działań w środowisku Unity. W szczególności musimy się dowiedzieć więcej na temat sposobu, w jaki Unity obsługuje skrypty C# dołączone do obiektów `GameObject`. W tym przykładzie będziemy nadal korzystać ze skryptu `NaukaProgramowania` oraz obiektu `GameObject` `Main Camera`.

Skrypty stają się komponentami

Wszystkie elementy `GameObject` to skrypty. Niektóre z nich napisali dobrzy ludzie z zespołu Unity, inne musimy napisać sami. Komponenty specyficzne dla Unity, takie jak `Transform`, oraz odpowiadające im skrypty, po prostu nie powinny być przez nas edytowane.

W chwili, gdy utworzony skrypt zostanie upuszczony w kontenerze `GameObject`, staje się kolejnym komponentem tego obiektu, dlatego pojawia się w panelu *Inspector*. Dla Unity ten skrypt „chodzi, mówi i działa” jak każdy inny komponent. „Pod spodem” zawiera zmienne publiczne, które można w dowolnym momencie zmienić. Pomimo, że nie powinniśmy edytować komponentów dostarczanych przez Unity, możemy uzyskać dostęp do ich właściwości i metod, dzięki czemu są to potężne narzędzia do programowania.

Gdy skrypt staje się komponentem, Unity wprowadza pewne automatyczne korekty czytelności. Być może zauważyłeś, że kiedy dodaliśmy skrypt *NaukaProgramowania* do obiektu *Main Camera*, Unity wyświetlił ten skrypt jako *Nauka Programowania*, a zmienną *currentAge* zastąpił nazwą *Current Age*.

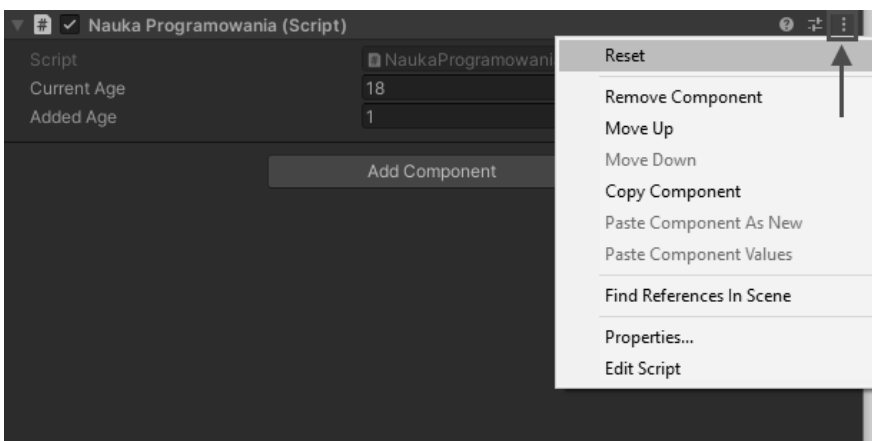
Co prawda w poprzednim punkcie, „Czas na działanie”, już aktualizowałeś zmienną w panelu *Inspector*, ale to ważna czynność, dlatego warto ją opisać bardziej szczegółowo. Istnieją dwie sytuacje, w których można modyfikować wartość właściwości:

- W trybie odtwarzania.
- W trybie programowania.

Zmiany wprowadzone w trybie odtwarzania odnoszą skutek natychmiast, w czasie rzeczywistym, co jest świetne z punktu widzenia testowania i dostrajania gry. Trzeba jednak zapamiętać, że wszelkie zmiany wprowadzone w trybie odtwarzania zostaną utracone, gdy zatrzymamy grę i powrócimy do trybu programowania. W trybie programowania Unity zapisze wszelkie modyfikacje wprowadzone do zmiennych. Oznacza to, że gdy wyjdiesz z Unity, a następnie uruchomisz program ponownie, zmiany będą zachowane.

Zmiany wartości wprowadzone w panelu *Inspector* nie modyfikują skryptu, ale zastępują wszelkie wartości przypisane do skryptu w czasie, gdy Unity znajdował się w trybie odtwarzania.

Jeśli chcesz cofnąć zmiany wprowadzone w panelu *Inspector*, możesz zresetować skrypt do jego wartości domyślnych (czasami nazywanych wstępnymi). Kliknij ikonę z trzema kropkami po prawej stronie dowolnego komponentu, a następnie wybierz *Reset*, jak pokazałem na poniższym zrzucie ekranu:



Powinno to dać Ci trochę spokoju — jeśli Twoje zmienne wymkną Ci się spod kontroli, zawsze masz do dyspozycji „twardy reset”.

Pomocna dłoń od klasy MonoBehaviour

Wiemy, że skrypty C# to klasy, ale skąd Unity wie, aby niektóre ze skryptów przekształcić w komponenty, a inne nie? Krótka odpowiedź jest taka, że *NaukaProgramowania* (a także dowolny inny skrypt stworzony w Unity) dziedziczy po klasie MonoBehaviour. Stąd Unity wie, że tę klasę C# można przekształcić na komponent.

Zagadnienie dziedziczenia klas jest nieco zaawansowane na tym etapie nauki programowania. Pomyśl jednak o dziedziczeniu tak, jakby klasa MonoBehaviour pożyczyla kilka swoich zmiennych i metod skryptowi *NaukaProgramowania*. Szczegółowe informacje o dziedziczeniu klas znajdziesz w rozdziale 5.

Metody Start() i Update(), z których skorzystaliśmy w pierwszych przykładach kodu, należą do klasy MonoBehaviour. Unity uruchamia te metody automatycznie dla każdego skryptu dodanego do obiektu GameObject. Metoda Start() uruchamia się raz w momencie rozpoczynania odtwarzania sceny, natomiast metoda Update() uruchamia się raz dla każdej ramki (w zależności od szybkości klatek na określonym komputerze).

Teraz, gdy masz podstawową wiedzę o dokumentacji Unity, mam do wykonania krótkie, opcjonalne zadanie!

Próba gry Narodziny bohatera — klasa MonoBehaviour w Scripting API

Teraz nadszedł czas, abyś nauczył się samodzielnego wykorzystywania dokumentacji Unity. Nie ma na to lepszego sposobu od wypróbowania kilku popularnych metod klasy MonoBehaviour:

- Aby lepiej zrozumieć rolę metod Start() i Update() w Unity oraz cele, do jakich są wykorzystywane, spróbuj wyszukać je w dokumentacji *Scripting API*.
- Jeśli chcesz, wykonaj dodatkowy krok i poszukaj w dokumentacji opisu klasy MonoBehaviour. W ten sposób poznasz więcej szczegółowych informacji na jej temat.

Zanim zagłębię się w tajniki programowania w C#, muszę opisać ostatni, kluczowy blok budulcowy programowania — komunikację pomiędzy klasami.

Komunikowanie się pomiędzy klasami

Do tej pory zajmowaliśmy się klasami lub szerzej komponentami Unity jako oddzielnymi, niezależnymi podmiotami. W rzeczywistości jednak są one ze sobą mocno „powiązane”. Bez skorzystania z pewnego rodzaju interakcji lub komunikacji pomiędzy klasami trudno byłoby stworzyć jakąkolwiek sensowną aplikację.

Jeśli przypominasz sobie zaprezentowany wcześniej przykładowy kod obsługi urzędu pocztowego, to pamiętasz, że skorzystałem tam z kropek do oddzielania od siebie klas, zmiennych

i metod. Gdybyśmy pomyśleli o klasach jak o katalogach informacji, to notacja z kropką byłaby narzędziem indeksowania:

```
PostOffice.Address
```

Notacja z kropką pozwala na odwoływanie się do zmiennych, metod lub innych typów danych wewnątrz klasy. Dotyczy to również danych zagnieżdżonych lub podklas. Zagadnienie to opiszemy dokładniej w rozdziale 5.

Notacja z kropką opisuje również komunikację między klasami. Jest ona używana za każdym razem, gdy klasa potrzebuje informacji o innej klasie lub chce wykonać jedną z jej metod:

```
PostOffice.DeliverMail()
```

Notacja z kropką czasami jest określana jako korzystanie z operatora w postaci kropki (.). Nie zdziw się, jeśli spotkasz się z takim określeniem w dokumentacji.

Jeśli notacja z kropką jeszcze nie jest dla Ciebie jasna, nie martw się, z czasem się do niej przyzwyczaisz. Jest krwioobiegiem całego programowania, pozwala przekazywać informacje i kontekst, gdziekolwiek jest to potrzebne.

Podsumowanie

Na kilku krótkich stronach przebyliśmy długą drogę. Ta lektura pozwoliła jednak zrozumieć teorię podstawowych pojęć, takich jak zmienne, metody i klasy. Powinno to dać Ci solidne podstawy. Warto pamiętać, że opisane bloki budulcowe mają swoje realistyczne odpowiedniki w prawdziwym świecie. Zmienne służą do przechowywania wartości, podobnie jak skrzynki pocztowe służą do przechowywania listów. Metody przechowują instrukcje, które przypominają receptury, jakich należy przestrzegać, aby uzyskać przewidywany produkt, a klasy są projektami przypominającymi realne projekty budynków. Jeśli chcesz, aby budynek przetrwał dłuży czas, nie da się go zbudować bez dobrze przemyślanego projektu.

W pozostałej części tej książki zagłębimy się w tajniki składni języka C#. Zaczniemy od podstaw. W kolejnym rozdziale pokażę tylko nieco więcej szczegółów: jak tworzyć zmienne, zarządzać typami wartości oraz korzystać z prostych i bardziej złożonych metod.

Quiz — bloki budulcowe języka C#

1. Jakie jest główne przeznaczenie zmiennej?
2. Jaką rolę w skryptach odgrywają metody?
3. W jaki sposób skrypt staje się komponentem?
4. Jakie jest przeznaczenie notacji z kropką?

PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Ciekawość to pierwszy krok do programowania!

Unity jest zaawansowanym środowiskiem przeznaczonym do tworzenia gier 3D. Równocześnie może stanowić początek wspaniałej przygody z C#, który jest nowoczesnym i wszechstronnym językiem programowania. Taka ścieżka nauki pozwala na uniknięcie niezrozumiałego dla początkujących żargonu czy niejasnej logiki programowania. Poszczególne konstrukcje języka i ich zastosowanie nagle stają się proste — dzięki niestandardowym skryptom pisanim w C# można rozszerzyć możliwości Unity i tworzyć wciągające, świetnie wyglądające gry. Oto sposób na przyjemną i angażującą, a przy tym skuteczną naukę programowania!

To piąte, uzupełnione i zaktualizowane wydanie cenionego podręcznika, dzięki któremu poznasz od podstaw koncepcje programowania w języku C#, dowiesz się, czym są zmienne i klasy, a także jakie są zasady programowania zorientowanego obiektowo. Po zapoznaniu się z elementarzem C# przejdziesz do tworzenia gier w Unity. Nauczysz się tworzyć skrypty obsługi prostej mechaniki gry i zdobędziesz praktyczne doświadczenie w programowaniu zgodnym z najlepszymi praktykami kodowania. A tym samym wejdziesz na wyższy poziom w zakresie posługiwania się Unity i językiem C#. Po uważnej lekturze zdobędziesz umiejętności potrzebne do realizowania projektów własnych gier w Unity i C#.

W książce między innymi:

- precyzyjne przykłady przydatne w nauce podstaw programowania w języku C#
- skrypty i ich implementacja w Unity
- tworzenie podstawowych mechanizmów gier
- interfejsy, klasy abstrakcyjne, stosy, kolejki, obsługa wyjątków i błędów w C#
- podstawy sztucznej inteligencji do zastosowania w grach

Harrison Ferrone — inżynier. Pisze dokumentacje techniczne dla firmy Microsoft, a także tworzy treści instruktażowe do serwisów LinkedIn i Pluralsight. Publikuje artykuły na raywenderlich.com, prowadzi również działalność dydaktyczną. Kiedy nie pracuje, zajmuje się kotami, czyta książki i z nostalgią wspomina lektury szkolne.

	<i>Sprawdź nasze szkolenia!</i>	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	 AKADEMIA IT & BUSINESS	ISBN 978-83-283-8144-5	
 0 801 339900			9 788328 381445
 0 601 339900	WWW.SZKOLENIA.HELION.PL	Cena: 69,00 zł	
INFORMATYKA W NAJLEPSZYM WYDANIU			

Packt